

R / T E

REDUCING INTERNET TRANSPORT LATENCY

Project acronym: **RITE**

Project number: 317700

Work package: Network and interaction

Deliverable number and name:

Deliverable 2.3: Report on Prototype Development and Evaluation of Network and Interaction Techniques

**Title:** Report on Prototype Development and Evaluation of Network and Interaction Techniques

**Work Package:** WP2

**Version:** 01

**Date:** September 9, 2015

**Pages:** 368

**Author:**

Bob Briscoe

**Co-Author(s):**

Nicolas Kuhn, David Ros, Ing-Jyh Tsang, Siyu Tang, Gorrry Fairhurst, Naeem Khademi, Koen de Schepper, Per Hurtig, Anna Brunstrom, Andreas Petlund, Mohammad Rajiullah, Iffat Ahmed, Minoo Kargar. Olga Bondarenko, Chamil Kulatunga

**To:**

Jorge Carvalho

Project Officer

**Status:**

- ☐ Draft
- ☐ To be reviewed
- ☐ Proposal
- ☒ Final / Released to CEC

**Confidentiality:**

- ☒ PU – Public
- ☐ PP – Restricted to other programme participants
- ☐ RE – Restricted to a group
- ☐ CO – Confidential

**Revision:**

(Dates, Reviewers, Comments)

**Contents:**

This report describes the design and evaluation of latency reducing techniques involving interaction between end-system transport mechanisms and novel network mechanisms. It is the final report from work package 2 of the RITE project. It describes mechanisms ready for deployment in real-life scenarios as outcomes of Tasks 2.2 and 2.3. It identifies where mechanisms are ready for evaluation on the project's industrial testbeds. Nonetheless, mechanisms developed later in the project that will not be evaluated due to lack of time are also recorded.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Key Contributions of Evaluated Prototype Mechanisms	4
<b>2</b>	<b>Dropping packets with Active Queue Management</b>	<b>6</b>
2.1	AQM recommendations	6
2.2	Characterization guidelines	7
2.3	Comparison of self-tuning AQM algorithms	8
2.3.1	Notation	9
2.3.2	Sensitivity of CoDel and PIE to network conditions	9
2.3.3	Operating ranges of CoDel and PIE	9
2.3.4	Tunability of CoDel and PIE	9
2.3.5	Discussion	10
2.4	Constraints on AQMs from TCP: Insights from Curvy RED	10
2.4.1	AQM: Fixed Delay Target Considered Harmful	10
2.4.2	Scaling AQM configuration	11
2.4.3	Insights from Curvy RED: Status and Next Steps	11
2.5	Review of PIE for the IETF	11
2.5.1	PIE Review: Main Concerns	12
2.6	Impact of scheduling on AQM performance	13
2.6.1	Flow isolation of FQ-CoDel and thin streams	13
2.6.2	Flow prioritization of FQ-CoDel and thin streams	13
2.6.3	Flow starvation prevention of FQ-CoDel	13
2.6.4	Discussion	14
2.7	Rural AQMs	14
2.7.1	Proposing a <i>Rural</i> parameterization in ns-2	14
2.7.2	RTT sensitivity of the <i>Rural</i> parameterization using NETEM	15
2.7.3	Assessing the benefits of <i>Rural</i> parameterization using a real satellite link	16
2.7.4	Discussion	16
2.8	Interaction between Lower-than Best Effort & AQMs	17
2.8.1	Congestion window evolutions	17
2.8.2	Discussion	18
2.9	AQM detection active measurement tool	18
2.9.1	Queue Detection Approach: AQM or Tail-drop	19
2.9.2	Experiments	19
2.9.3	AQM Detection Tool: Status and Plans	22
2.10	MADPIE	22

2.10.1	Adding deterministic drops in PIE . . . . .	22
2.10.2	Proof of concept . . . . .	22
2.10.3	Traffic mix and RTT mix . . . . .	23
2.10.4	Discussion . . . . .	25
2.10.5	MADPIE Status . . . . .	25
2.11	Discussion on squared AQMs . . . . .	26
2.11.1	A simpler RED: Curvy RED . . . . .	27
2.11.2	A simpler PIE: $PI^2$ . . . . .	27
<b>3</b>	<b>Network signals: ECN markings and interactions with TCP</b>	<b>29</b>
3.1	ECN benefits . . . . .	29
3.1.1	ECN's lack of deployment . . . . .	29
3.1.2	Potential ECN benefits in the future . . . . .	30
3.1.3	Today's ECN benefits . . . . .	30
3.2	ECN path support evaluation . . . . .	30
3.3	Guidelines for Adding ECN to Protocols that Encapsulate IP . . . . .	33
3.3.1	ECN Encapsulation: Motivation . . . . .	33
3.3.2	ECN Encapsulation: Standards Co-ordination, Status & Plans . . . . .	35
3.4	ECN Roadmap . . . . .	36
3.5	Alternative Backoff with ECN . . . . .	38
3.5.1	Problem with current AQM mechanisms . . . . .	38
3.5.2	Motivation behind ABE . . . . .	38
3.5.3	Solution: Alternative Backoff with ECN (ABE) . . . . .	38
3.5.4	Evaluation of ABE . . . . .	39
3.6	Coupled AQMs for mixed Congestion Controls . . . . .	39
3.6.1	The Low Loss, Low Latency and throughput Scalable (L4S) service . . . . .	39
3.6.2	Dual Queue for Low Latency . . . . .	39
3.6.3	Evaluation . . . . .	41
3.7	TCP Prague: Low Loss, Low Latency, Scalable <i>and</i> Safe . . . . .	43
3.7.1	Improvements to prevent harm to other traffic . . . . .	43
3.7.2	Improvements optimising latency . . . . .	44
<b>4</b>	<b>Getting up to Speed (GUTS) Fast</b>	<b>46</b>
4.1	What Use is Top Speed Without Acceleration? . . . . .	46
4.2	Existing GUTS solutions . . . . .	46
4.3	Up to Speed with Queue View (QV) . . . . .	47
4.3.1	QV: Status . . . . .	47
4.4	Generic Accurate ECN Feedback in TCP . . . . .	48

4.4.1	Accurate TCP-ECN Feedback: Status . . . . .	48
4.4.2	Accurate TCP-ECN Feedback: High Level Overview . . . . .	49
<b>5</b>	<b>Conclusions</b>	<b>51</b>
	<b>References</b>	<b>52</b>
	<b>Appendices</b>	<b>57</b>
<b>A</b>	<b>Active Queue Management Publications</b>	<b>58</b>
A.1	IETF Recommendations Regarding Active Queue Management . . . . .	60
A.2	AQM Evaluation Guidelines . . . . .	92
A.3	Operating ranges and tunability of CoDel and PIE . . . . .	127
A.4	Insights from Curvy RED (Random Early Detection) . . . . .	133
A.5	Review: Proportional Integral controller Enhanced (PIE) AQM) . . . . .	140
A.6	Evaluation of Priority Scheduling and Flow Starvation for Thin Streams with FQ-CoDel . . . . .	155
A.7	Tackling Bufferbloat in Capacity-limited Networks . . . . .	160
A.8	On the coexistence of AQM and LBE . . . . .	165
A.9	Improving PIE's performance over high-delay paths . . . . .	169
A.10	Identifying Bottleneck Drop Strategies: an Active Measurement Approach . . . . .	173
<b>B</b>	<b>Network Signals Publications</b>	<b>177</b>
B.1	IETF Liaison on ECN and IEEE Protocols . . . . .	179
B.2	IETF Liaison on ECN and 3GPP Protocols . . . . .	182
B.3	Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP . . . . .	184
B.4	TCP Alternative Backoff with ECN (ABE) . . . . .	214
B.5	Alternative Backoff: Achieving Low Latency and High Throughput with ECN and AQM . . . . .	220
B.6	The Benefits of using Explicit Congestion Notification (ECN) . . . . .	235
B.7	'Data Centre to the Home': Ultra-Low Latency for All . . . . .	254
B.8	DualQ Coupled AQM for Low Latency, Low Loss and Scalable Throughput . . . . .	266
<b>C</b>	<b>Getting up to Speed Fast Publications</b>	<b>288</b>
C.1	What Use is Top Speed without Acceleration? . . . . .	288
C.2	Up to Speed with Queue View (QV) . . . . .	304
C.3	Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback . . . . .	315
C.4	More Accurate ECN Feedback in TCP . . . . .	333



## Abbreviations

This section provides definitions of key terms and defines the abbreviations used in the remainder of the report.

<b>3G</b> 3rd Generation	<b>DRR</b> Deficit Round Robin
<b>3GPP</b> 3rd Generation Partnership Project	<b>DSCP</b> Differentiated Services Code Point
<b>AccECN</b> Accurate ECN	<b>DSL</b> Digital Subscriber Line
<b>ACK</b> Acknowledgement	<b>E-UTRAN</b> Evolved Universal Terrestrial Radio Access Network
<b>ADU</b> Application Data Unit	<b>E2E</b> End-to-End
<b>API</b> Application Programming Interface	<b>ECE</b> Echo Congestion Experienced
<b>AQM</b> Active Queue Management	<b>ECN</b> Explicit Congestion Notification
<b>ARED</b> Adaptive Random Early Drop	<b>ECT</b> ECN-Capable Transport
<b>BE</b> Best Effort	<b>eNB</b> eNodeB, Evolved Node B (LTE Base-Station)
<b>BCP</b> Best Current Practice	<b>FIFO</b> First-In First-Out
<b>BDP</b> Bandwidth Delay Product	<b>FQ</b> Fair queuing
<b>BoF</b> Birds-of-a-Feather	<b>FQ-CoDel</b> Flow queuing CoDel
<b>CAIDA</b> Cooperative Association for Internet Data Analysis	<b>FSE</b> Flow State Exchange
<b>CC</b> Congestion Control	<b>FTP</b> File Transfer Protocol
<b>CDF</b> Cumulative Density Function	<b>GB</b> Gigabyte
<b>CDG</b> CAIA Delay Gradient	<b>GPRS)</b> General Packet Radio Service
<b>CDN</b> Content Delivery Network	<b>GRE)</b> Generic Routing Encapsulation
<b>CE</b> Congestion Experienced	<b>GTP)</b> GPRS Tunnelling Protocol
<b>CM</b> Congestion Manager	<b>GUTS</b> Getting up to Speed
<b>CMT-SCTP</b> Concurrent Multipath Transfer for SCTP	<b>HAS</b> HTTP Adaptive Streaming
<b>CoDel</b> Controlled Delay	<b>HSS</b> Hybrid Slow-Start
<b>CUBIC</b> Cubic function congestion control	<b>HTTP</b> HyperText Transfer Protocol
<b>cwnd</b> Congestion WiNdoW	<b>HULL</b> High-bandwidth Ultra Low Latency
<b>CWR</b> Congestion Window Reduced	<b>ICMP</b> Internet Control Message Protocol
<b>CWV</b> Congestion Window Validation	<b>IETF</b> Internet Engineering Task Force
<b>DASH</b> Dynamic Adaptive Streaming over HTTP	<b>IP</b> Internet Protocol
<b>DC</b> Data Center	<b>IRTF</b> Internet Research Task Force
<b>DCLC</b> Data Center Latency Control (IRTF research group)	<b>ISP</b> Internet Service Provider
<b>DCTCP</b> Data Center TCP	<b>IW</b> Initial Window
<b>DiffServ</b> Differentiated Services	<b>LAN</b> Local Area Network
<b>DNS</b> Domain Name System	<b>LEDBAT</b> Low Extra Delay Background Transport
<b>DOCSIS</b> Data Over Cable Service Interface Specification	<b>LKM</b> Loadable Kernel Module
<b>DoS</b> Denial of Service	<b>LTE</b> Long Term Evolution
	<b>MAC</b> Media Access Control

---

<b>MB</b> Megabyte	<b>RTP</b> Real-Time Protocol
<b>MPLS</b> Multi Protocol Label Switching	<b>RTT</b> Round Trip Time
<b>MPTCP</b> Multipath TCP	<b>RW</b> Restart Window
<b>MSS</b> Maximum Segment Size	<b>SACK</b> Selective Acknowledgement
<b>NAT</b> Network Address Translation	<b>SAE</b> System Architecture Evolution
<b>NS-2</b> Network Simulator 2	<b>SBD</b> Shared Bottleneck Detection
<b>NV-GRE</b> Network Virtualization using Generic Routing Encapsulation	<b>SCTP</b> Stream Control Transmission Protocol
<b>NVO</b> Network Virtualization Overlay	<b>SDO</b> standards Development Organisation
<b>NVP</b> Non Validated Period	<b>SFQ</b> Stochastic Fair Queuing
<b>OWD</b> One Way Delay	<b>SGW</b> Serving Gateway
<b>PCI</b> Packet Congestion Indication	<b>SKB</b> Socket Buffer
<b>PCN</b> Pre-Congestion Notification	<b>SQ</b> Source Quench
<b>PDI</b> Packet Drop Indication	<b>SVD</b> Singular Value Decomposition
<b>PDPC</b> Packet Discard Prevention Counter	<b>SYN</b> Synchronize packet
<b>PDV</b> Packet Delay Variation	<b>TFRC</b> TCP Friendly Rate Control
<b>PGW</b> Packet Gateway	<b>TCP</b> Transmission Control Protocol
<b>PIE</b> Proportional Integral controller Enhanced	<b>TCP'</b> Paced and RTT Independent TCP (TCP-PRIme)
<b>PLT</b> Page Load Time	<b>TCPM</b> TCP Maintenance (IETF Working Group)
<b>PSP</b> PipeACK Sampling Period	<b>TOS</b> Type Of Service
<b>QV</b> Queue View	<b>TRILL</b> TRansparent Interconnection of Lots of Links
<b>RCP</b> Rate Control Protocol	<b>TSQ</b> TCP Small Queue
<b>RED</b> Random Early Drop	<b>UDP</b> User Datagram Protocol
<b>RFC</b> Request For Comments	<b>URG</b> TCP Urgent flag
<b>RITE</b> Reducing Internet Transport Latency End-to-End	<b>VCP</b> Variable-structure congestion Control Protocol
<b>RMCA</b> T RTP Media Congestion Avoidance Techniques (IETF Working Group)	<b>VXLAN</b> Virtual Extensible LAN
<b>RT</b> Real-Time	<b>WiFi</b> Wireless Fidelity
<b>RTCWEB</b> Real Time Collaboration on world wide WEB (IETF Working Group)	<b>WG</b> Working Group
<b>RTO</b> Retransmission Time Out	<b>WP</b> Work Package
	<b>WRED</b> Weighted Random Early Drop

<b>Participant organisation name</b>	<b>Participant Short Name</b>	<b>Country</b>
Simula Research Laboratory	SRL	Norway
BT	BT	UK
Alcatel-Lucent	ALU	Belgium
University of Oslo	UiO	Norway
Karlstad University	KaU	Sweden
Institut Mines-Télécom	IMT	France
University of Aberdeen	UoA	UK

# 1 Introduction

The RITE project chose to focus on sources of latency due to interactions between systems. This was largely because sources of latency within a single system tend to have already been addressed by the relevant developer. The RITE project was scoped to focus on transport layer matters, and it was structured to deal separately with the two main types of system interaction: interactions between end-system in WP1 and interactions between end-systems and network elements in WP2.

The present deliverable is the final report on WP2 work, which addresses the latter category—interaction between end-systems and the network. D2.1 and D2.2 reported on the progress of this work in the previous two years, but D2.2 was not public. Therefore, for the public record, work is briefly repeated here in D2.3 even if has changed little since D2.2. Such repetition is identified in the relevant sections, for the benefit of those who have already read D2.2.

This deliverable mainly focuses on Prototype Development and Evaluation of designs developed in the earlier years of the project. However, as the work has progressed, new insights have led to a continuous stream of innovations, all of which could not be evaluated within the 3-year duration of the project. Therefore, the newly introduced designs are reported for the record, and the lack of evaluation is highlighted. WP3 is devoted to more detailed testbed evaluation of a few mechanisms selected for further study, and deliverable 3.3 reports on that work. Therefore the evaluations reported in the present deliverable are generally confined to initial feasibility testing, typically using analytical or simulation techniques. Testbed evaluations are also sometimes outlined at high level.

In the RITE project's extensive survey of the techniques for reducing latency, three transport-related mechanisms were identified that could significantly reduce latency by improving the interaction between end-systems and the network: i) active queue management (AQM) to prevent capacity-seeking behaviour by end-systems (TCP) from filling buffers; ii) using explicit congestion notification (ECN) for much faster and more frequent signalling of the state of the network to end-systems than can be achieved with loss alone; and iii) techniques for seeking out capacity more rapidly, either when the flow in question starts or when other flows end.

The project structured the work of WP2 around these three activities, which naturally provide the structure for this final deliverable, as they did for D2.2 one year earlier.

## 1.1 Key Contributions of Evaluated Prototype Mechanisms

Below, the main contribution of each activity to the project goal of reducing Internet transport latency are highlighted. Work that was introduced late in the project and therefore has not been fully evaluated is omitted, no matter how important we expect it might be:

### (i) Active Queue Management

**Recommendations Regarding AQM** We have contributed to the publication of these recommendations as RFC 7567 and Best Current Practice (BCP) 197. This is now the IETF's primary publication aimed at reducing queuing latency. It sets the ground rules for using AQM, stating the key requirements of a good AQM algorithm and the research issues. The document does not itself standardise specific algorithms; specifications for these are following in later IETF documents.

**AQM Evaluation Guidelines** We have led the creation of this Internet Draft defining, which has been adopted by the IETF AQM WG and is progressing towards publication as an RFC. The document is useful for the analysis of both novel and existing AQM mechanisms, not only for standardisation purposes but also for the research community. These guidelines distil our own experience, adapted following feedback from the community to ensure applicability in a wide range of scenarios.

**Operating Range and Tunability of CoDel and PIE** We have evaluated and compared the performance of CoDel, PIE and the older ARED self-tuning AQM algorithms, using both experimental evaluations and simulations in *ns-2*. The more recent focus has been on their ability to control queuing delay when operating conditions are on the edge of the typical range seen on the Internet.

**Impact of Scheduling on AQM Performance** An analysis of the impact of including scheduling with an AQM algorithm has been undertaken, by separating the FQ-Codel algorithm into its constituent parts, and applying it to thin streams, not just continuous TCP flows as in other studies.

**A Rural parameterization for AQMs** A separate ‘rural’ parameterization has been proven to be necessary for modern so-called ‘auto-tuning’ AQMs. Auto-tuning only applies to link-rate not to base round trip time (RTT), so performance of extended rural access networks, especially over satellite links, suffers without such a special parameterization.

**Coexistence of Lower-than Best Effort with AQMs** A study of whether Lower-than Best Effort transport protocols can scavenge spare capacity in the presence of modern AQMs. It was found that LBE protocols such as Low Extra Delay Background Transport (LEDBAT) or CAIA Delay Gradient (CDG) have great difficulty using spare capacity when the delay target of the AQM is very low, such as the 5 ms target of CoDel. However, they can work with other AQMs, although LEDBAT has to be configured with a much lower target delay than the standard.

**MADPIE** A proposed extension to PIE to add deterministic as well as random drops to prevent oscillations in queuing delay on high RTT paths. No additional effect is intended or measured on low RTT paths.

## (ii) Network Signals

**Benefits of ECN** To promote the future use of ECN we led the creation of an Internet draft documenting the different (potential) benefits of ECN. The objective is to make sure that ECN signals are not perverted by current and future network infrastructures. The draft has been adopted by the IETF and is progressing towards publication as an RFC.

**ECN Path Support** We contributed to an evaluation of the current state for ECN support on the paths in the Internet and in end-systems.

**Guidelines for Adding ECN to Lower Layers** Not having full support for ECN is partly due to the non-existence of clear guidelines on how ECN should travel between encapsulated layers. We led the creation of an IETF draft that defines the interface between IP and numerous lower layer protocols, many under the authority of other standards bodies. This draft has been adopted by the IETF and is progressing towards publication as an RFC.

**Alternative Backoff with ECN** We have proposed and evaluated a technique called Alternative Backoff with ECN (ABE), that requires sender-only modification. It is designed to give advantageous capacity utilization when the standard ECN capability is enabled at the client and in the network, consequently incentivizing both to be turned on.

**ECN for Low Latency, Low Loss, Scalable Service (L4S)** We have demonstrated that redefining the meaning of an ECN mark and how to respond to it can provide a universal low latency service for the Internet. We have explored and investigated different strategies that will reduce response times to congestion, hence reducing latency.

**Coexistence of L4S and Classic Traffic: Coupled DualQ AQM** To support this redefined low latency ECN response, the network can help to provide migration and backwards compatibility strategies. We have proposed and evaluated coupled AQMs for throughput fairness between classes of flows that have different congestion controllers.

## (iii) Getting up to Speed Fast

**What Use is Top Speed without Acceleration?** We have shown that the current design of the transport protocols is not able to take advantage of the full capacity of the network any more, as link capacities have increased but typical flows have remained so short that they complete before they have accelerated to full-speed. We have shown that all approaches proposed to address this problem will not be able to continue to scale with increasing link capacities.

**Generic Accurate ECN Feedback in TCP** Currently TCP echoes an ECN congestion signal only once per RTT, revealing only the existence not the extent of congestion. We have persuaded the IETF that a solution to this problem needs to be standardised, and contributed as co-author to the statement of requirements, which the IETF has published as RFC 7560. A solution has also been proposed.

A preferred solution will not only serve current needs (e.g. DCTCP), but also a future TCP receiver should generically reflect rich ECN information and its timing. Then sender-only solutions to the getting up to speed problem can be deployed to exploit this generic feedback without having to get new receiver capabilities deployed as well.

## 2 Dropping packets with Active Queue Management

Large packet buffers in network devices along the end-to-end path can be the cause of excessive latency. The phenomenon has been called “Bufferbloat” [1], and results from excessive queues that are seldom emptied, so that the almost constant large queuing delay impacts on the end-to-end latency and especially the performance of delay sensitive applications.

Active Queue Management (AQM) mechanisms can be introduced in routers and switches to control the number of packets in a buffer. These schemes drop (or mark, if ECN is supported) packets before the buffer overflows, depending on the standing queue size and/or the queuing delay. AQM mechanisms aim to manage bufferbloat by reducing end-to-end latency, reducing packet drops, and avoiding lock-out phenomena<sup>1</sup>.

Deliverable D2.2 evaluated recent AQM proposals including Controlled Delay Active Queue Management (CoDel) and Proportional Integral controller Enhance (PIE) algorithms. Based on the conclusions in deliverable D2.2, during the third year of the RITE project, the AQM activity has focused on (i) further assessing the performance of CoDel and PIE, (ii) evaluating how these schemes perform in specific scenarios, (iii) proposing parameterizations for specific scenarios and (iv) proposing new dropping/marketing schemes.

The rest of this section is organized as follows:

- Contribute to the evaluation of existing schemes:
  - § 2.1 - IETF recommendations guidelines for AQM schemes;
  - § 2.2 - AQM characterization guidelines;
  - § 2.3 - Definition of CoDel and PIE operating range and tunability;
  - § 2.4 - Constraints on AQMs from TCP: Insights from Curvy RED;
  - § 2.5 - PIE I-D review for IETF.
- Prototype parameterizations for existing AQM schemes in specific scenarios:
  - § 2.6 - AQM schemes and scheduling;
  - § 2.7 - AQM schemes in capacity-limited networks;
  - § 2.8 - AQM and Lower-Than Best Effort protocols;
  - § 2.9 - AQM detection tool
- AQM prototypes:
  - § 2.10 - Maximum and Average queuing Delay with PIE (MADPIE);
  - § 2.11 - Squared AQM.

### 2.1 AQM recommendations

RITE researchers have been contributing to the IETF initiative to create new AQM recommendation [2] to replace the IRTF-developed informational ‘RED Manifesto’, RFC2309, with a new focus on reducing network latency using AQM technology.

Section 2 of the document summarises the benefits that active queue management can bring. Section 3 identifies issues and classes of mechanisms, and Section 4 provides standards requirements for using and developing AQM methods. Although Sections 2 & 3 have been significantly reworked since this activity was last reported in internal deliverable D2.2, the key recommendations (repeated below) have remained relatively stable:

<sup>1</sup>The lock-out phenomena is the result of single connection or a few flows to monopolize other connections from getting room in the queue. See the IETF AQM recommendations document in [2] or in the Appendix A.1

1. Network devices SHOULD implement some AQM mechanism to manage queue lengths, reduce end-to-end latency, and avoid lock-out phenomena within the Internet. The choice of mechanism is not specified in [2], but the constraints are, and the document specifically updates the RFC2309 requirement to use the Random Early Detection (RED) algorithm.
2. Deployed AQM algorithms SHOULD support Explicit Congestion Notification (ECN) as well as loss to signal congestion to endpoints. This paves the way for other documents to define how ECN should be used (see related RITE work, § 3 and §4).
3. The algorithms that the IETF recommends SHOULD NOT require operational (especially manual) configuration or tuning. This is a key feature of new algorithms, but one that needs caution, and RITE research is exploring the parameter sensitivity of these methods (§ 2.3).
4. AQM algorithms SHOULD respond to measured congestion, not application profiles. This building on previous IETF work, such as RFC7141 [3] (developed in part by the EC Trilogy 2 project).
5. AQM algorithms SHOULD NOT interpret specific transport protocol behaviours, and need to be robust to a wide range of applications and network traffic.
6. Transport protocol congestion control algorithms SHOULD maximize their use of available capacity (when there is data to send) without incurring undue loss or undue round trip delay. The interaction between transport and AQM is the topic of other current IETF developments within the TCPM and AQM working groups.
7. Research, engineering, and measurement efforts are needed regarding the design of mechanisms to deal with flows that are unresponsive to congestion notification or are responsive, but are more aggressive than present TCP. Although the document will define the trajectory for deployment and standards there is still much research needed to ensure evolution of methods.

The IETF recommendations regarding AQM [2] has been published as RFC7567 and Internet Best Current Practice (BCP) 197. The document does not itself standardise specific algorithms, specifications for these will follow in later IETF documents.

This document is available in Appendix A.1.

## 2.2 Characterization guidelines

The merits of any new proposal of an AQM algorithm have to be thoroughly assessed before deployment and/or standardisation. One of the objectives of the AQM Working Group at the IETF is thus to produce an Informational RFC containing a set of *characterization guidelines*. Several RITE partners have been leading the writing of an Internet Draft that should be adopted soon by the Working Group to fulfill this milestone [4]. Such guidelines will be useful for the analysis of both novel and existing AQM mechanisms, not only for standardisation purposes but also for the research community at large. These guidelines have been distilled from our own experiments (§ 2.3), and have been adapted to ensure both their applicability in a wide range of settings and their acceptance by the IETF community. One important aspect of the guidelines is that they are not bound to a particular evaluation toolset, and they should be useful both for simulation studies as well as for tests using real implementations.

AQM characterization guidelines have to regard not only whether an AQM mechanism offers an improvement over a simple drop-tail buffer or over another reference AQM, but also considerations such as:

- What is the minimum set of standard experiments, metrics and performance indicators that should be used to assess an AQM's performance. The idea is that any new AQM proposal—and, especially, any proposal submitted to the IETF for standardisation—should be evaluated *at a minimum* by means of the proposed tests and indicators. Such a minimum set offers a common basis of comparison and focuses evaluation on key aspects of the algorithms. For this reason, considering e.g. context-dependent scenarios (like, say, very specific application types or link technologies) has been deemed out of scope for this draft by the WG; a more exhaustive “characterization suite” would be tackled later by the Working Group in another document.

- Deployment safety: How does an AQM behave in the face of e.g. changing traffic patterns or different endpoint congestion-control algorithms? How sensitive is the performance of the algorithm to different parameter settings? If the algorithm is adaptive / self-tuning, is it robust under different operating conditions?
- Interactions with Explicit Congestion Notification (ECN) and flow scheduling: recent proposals such as FQ\_CoDel [5] work by coupling a scheduling mechanism with a packet dropping mechanism; hence, it is important for a set of recommendations on how to evaluate AQMs to take into account such couplings, due to potential interactions arising from e.g. applying one mechanism when packets are enqueued and the other when they are dequeued. The current version of the characterization guidelines do not propose guidelines to assess the performance of scheduling algorithms: it is out of the scope of this document that focuses on dropping and/ or marking AQM schemes. The characterization guidelines may be complemented with another one on guidelines for assessing combination of packet scheduling and AQM. We note that such a document will inherit all the guidelines from this document plus any additional scenarios relevant for packet scheduling such as flow starvation evaluation or impact of the number of hash buckets.
- Non-performance related issues such as implementation complexity, cost and self-tuning ability. These parameters, and any tradeoff between them and pure performance aspects, have to be carefully considered before adopting a new AQM in real-world deployment scenarios.

The draft, in its current version, discusses the considerations above and provides general guidance on:

- End-to-end metrics of interest for AQM performance evaluation. The focus on end-to-end measurements stems from the fact that in real-world experiments it may well be not practical, or even not possible, to obtain fine-grained measurements from network equipment buffers (or at least, not as detailed as what is feasible with simulations).
- Experiments that focus on evaluating some key aspects of an AQM's operation, including: (a) any tradeoffs between latency and throughput, (b) RTT fairness, (c) an AQM's ability to absorb bursts while keeping latency in check, (d) stability in the face of different operating conditions. This includes outlining a common set of topologies, types of traffic, types of endpoint congestion-control methods, etc.
- How to “fairly” compare different AQM methods, based on the recommended experiments and metrics.

The first version of the draft was presented at the 89th IETF meeting in London (March 2014), and updated versions were presented in the 90th, the 91th, the 92th AQM WG meetings at IETF.

See the current version of this document in Appendix A.2 for more details.

### 2.3 Comparison of self-tuning AQM algorithms

In deliverable D2.1, we presented a preliminary experimental comparison of ARED [6], CoDel [7] and PIE [8] where we evaluated the impact of their respective *target delay* parameter on their capability to control the queuing delay without harming link utilisation. We also identified a spectrum of evaluation scenarios which had to be investigated to better assess the performance of these AQM schemes.

In deliverable D2.2, we presented a detailed study of self-tuning AQMs. We have run simulations and real-life experiments that cover the spectrum of scenarios proposed in D2.1.

In this deliverable D2.3, based on our work presented in D2.2 that have been slightly extended and based on recently published articles, we further assess the range of conditions in which the default parameters of PIE and CoDel fail to perform well. We also evaluate how tunable CoDel and PIE are, i.e., whether they can easily be adjusted to (1) achieve any desired trade-off between queuing delay and bottleneck utilization and (2) let CoDel and PIE improve their performance outside their operating range.

For more details on the summary that is proposed in the rest of this section, see the current version of this document in Appendix A.3.



### 2.3.1 Notation

CoDel and PIE use similarly-named *target delay* (denoted by  $\tau$ ) and *update interval* (denoted by  $\lambda$ ) parameters in a different manner. When needed for clarity, we use the notation  $\tau_x, \lambda_x$  to denote the parameters of AQM algorithm  $x$ .

### 2.3.2 Sensitivity of CoDel and PIE to network conditions

In D2.2 we have shown that we saw that CoDel is sensitive to the traffic load; that CoDel and PIE are sensitive to the base RTT; that CoDel and PIE are sensitive to link speeds and “break” when link capacities are very low.

In light of the results presented in D2.2, we can conclude that:

- CoDel may not work well with RTTs in the interval [10 ms, 1 s] and is sensitive to the traffic load;
- the self-tuning of PIE, supposed to make it robust and optimized for various network conditions, shows some limits.

The default parameters of PIE and CoDel may not suit every situation, making these AQM schemes to drastically reduce the bottleneck utilization or to allow a queuing delay that is far from their target.

### 2.3.3 Operating ranges of CoDel and PIE

Based on our results presented in Appendix A.3 and those of [9, 10, 11, 12], we outline below the operating ranges of PIE and CoDel. Considering only the RTT and the capacity of the bottleneck, denoted  $base_{RTT}$  and  $C_{bot}$ , PIE and CoDel, with their default parameters, have trouble with controlling the queuing delay and maintaining a high bottleneck utilization if:

- $base_{RTT} \geq 200$  ms (Appendix A.3 and [10, 9]);
- $C_{bot} < 2$  Mbps (Appendix A.3 and [11]);
- $C_{bot} > 100$  Mbps ([9]).

We have observed that these AQMs do not fulfill their design goal outside these bounds, but we cannot guarantee that they work as expected within them. If we consider other parameters than  $base_{RTT}$  and  $C_{bot}$ , our results and those presented in [12] illustrate that there is a high correlation between the traffic load and the performance of CoDel.

### 2.3.4 Tunability of CoDel and PIE

With the specific characteristics of e.g. rural broadband ( $base_{RTT} \approx 300$  ms and  $C_{bot} \approx 1$  Mbps) and data-center networks ( $base_{RTT} \approx 10^{-5}$  ms and  $C_{bot} \approx 10^4$  Mbps), the limited operating range of CoDel and PIE prevents them from working as expected in these contexts. As one example, we have proposed specific parameterizations for the rural broadband networks use case.

We have further evaluated the “tunability” of PIE and CoDel, that is, assessed to what extent we can obtain different trade-offs than the default ones by changing only  $\tau$  and  $\lambda$ ; this would be a straightforward way to adapt their operating ranges to specific network conditions. PIE has already been updated for data-centers [13] by changing the  $\alpha$  and  $\beta$  parameters. However, the IETF recommendations on AQM [2] advocate the selection of default parameters appropriate to the general Internet with auto-tuning to avoid RED’s deployment issues.

We saw that the median queuing delay can easily be modified with PIE and CoDel by changing the target delay only; however, this works for CoDel only when the congestion level is low. Setting the update interval to the RTT reduces the RTT sensitivity of CoDel, but in many deployment cases, the person configuring the router will not be aware of the path RTT. Increasing the update interval tends to result in smoother evolution of the drop probability for PIE.

### 2.3.5 Discussion

Deploying AQM is essential step in reducing end-to-end latency [2]. Widespread deployment of early AQM proposals has not happened yet, mainly due to their sensitivity to the network characteristics. Two recently proposed AQM schemes attempt to tackle bufferbloat while addressing these issues. Before considering actual deployment of CoDel and PIE, it is essential to outline their operating ranges (in terms of RTT, bottleneck capacity and congestion levels).

We evaluate the limits of PIE's self-tuning ability and the sensitivity of CoDel to network conditions. Our conclusions, based on ns-2 simulations, are consistent with recently published studies that emulates networks using Linux implementations of the algorithms. We found that both schemes show performance issues when the RTT is higher than 200 ms, or the bottleneck capacity is lower than 2 Mbps. We also measure how the default parameterization of PIE and CoDel lets PIE allow more queuing delay than CoDel and achieve higher bottleneck utilization.

PIE and CoDel have had to be updated for cable-modems networks [14]: in the context of networks with characteristics that make them fit into the operating ranges, the adequate performance of CoDel and PIE can not be guaranteed as other parameters may have an impact, such as lower layers characteristics. In light of our results and the discussions in [15, 16], we claim that there is no overall winner for the best AQM: each algorithm proposes a specific trade-off, and works within limited network or traffic characteristics.

## 2.4 Constraints on AQMs from TCP: Insights from Curvy RED

### 2.4.1 AQM: Fixed Delay Target Considered Harmful

This activity allowed us to explain why attempting to keep delay constant is a bad idea. Unfortunately, this is the goal of both PIE and CoDel. At high load, keeping delay constant requires excessively high drop. This high drop itself takes over from queuing delay as the dominant cause of delay, particularly for short flows. It might be argued that this is not critical, because it only occurs at high load. However, the point of recent AQM redesigns like PIE and CoDel was meant to be to auto-tune to adapt to the environment, which should include high load.

We came across this insight when we were comparing PIE and CoDel against our Dual Queue Coupled AQM. BT was interested in replacing the differential scheduler in its Broadband Network Gateways (BNGs aka BRASs) with AQM, so that all traffic could have low delay service, rather than having to manage a bandwidth limit on a subset of traffic. However, BT was concerned that an AQM might introduce excessive drop for the VoIP traffic that had previously been given higher priority service. We were increasing the number of flows, and we found, for instance, that the FQ\_CoDel drop level rose to 4.8% when load was increased to 20 TCP Reno flows in a 40 Mb/s link (that is, 2 Mb/s per flow, which is not a particularly high load). FQ\_CoDel was configured with its default target delay of 5 ms.

This activity used a new AQM we developed for the DualQ Coupled AQM, called Curvy RED. However, it is included at this point in the deliverable, because it is really more an evaluation of PIE and CoDel, than an evaluation of Curvy RED. Curvy RED gave us insight because it includes a curviness parameter ( $u$ ), which can be thought of as a generalisation of a range of different AQMs: when  $u = 1$  it models RED; when  $u \rightarrow \infty$  it models PIE or CoDel (at least in the steady state), and we used  $u = 2$  in our DualQ AQM experiments. The analysis simply modelled the AQM and the TCP equation as a pair of simultaneous equations, to understand the interaction between the two.

The analysis has been recorded in a brief technical report, included in Appendix A.4. The main insight is summarised well by Figure 2.1, which is taken from the report. The horizontal axis is normalised load, which is essentially the number of TCP flows relative to the capacity, which is inversely proportional to the rate of one TCP flow (see report for the precise definition). An AQM of a particular curviness will cause queuing delay and loss to rise as shown by the pairs of curves for each value of  $u$ . It can be seen that the AQM has to make drop rise harder to clamp queuing delay to a constant value (like PIE and CoDel).

One conclusion is that, at high load, a link is better able to preserve reasonable performance if the delay target is softened into a curve rather than a hard cap.

Another conclusion is that ECN escapes the dilemma: for ECN-capable traffic, a fixed delay target only increases marking, not loss. And marking is solely a signal, not an impairment.

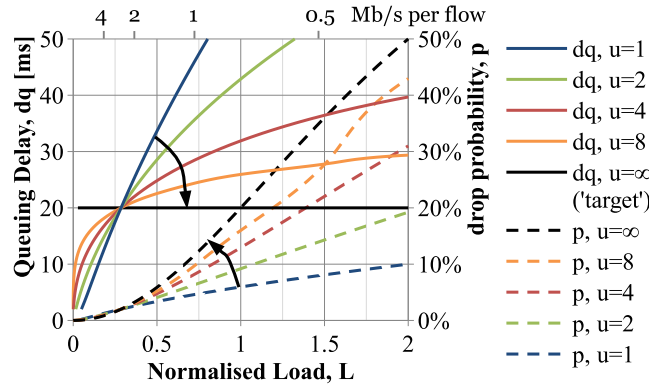


Figure 2.1: Dilemma between Two Impairments: Delay and Loss against Normalised Load; for TCP Reno and the Curvy RED Algorithm with Increasing Curviness,  $u$ ; Base RTT = 20 ms

#### 2.4.2 Scaling AQM configuration

The second part of the same report (in Appendix A.4) builds on the analysis in the first part to investigate whether AQM configuration should depend on link rate and on flow aggregation. The analysis proves that, once an optimal configuration has been found for one capacity, the same AQM with the same configuration can be deployed in different parts of a network whatever the capacity. But the configuration has to be changed if maximum base RTT is different (e.g. in a data centre vs. the public Internet).

A surprising corollary of this analysis concerns cases with a highly aggregated number of flows through a bottleneck. Although aggregation reduces queue variation, if the target queuing delay of the AQM at that bottleneck is reduced to take advantage of this aggregation, TCP will still increase the loss level because of the reduction in round trip time. The way to resolve this dilemma is to overprovision (a formula is provided).

#### 2.4.3 Insights from Curvy RED: Status and Next Steps

The analysis deliberately assumes that TCP is not limited by a minimum window of 2 segments. Of course it is in practice, and this will alter the results. A modified analysis has been done with this constraint, but not yet written up. However, we want to ensure that this does not confuse the main message of the work: that a constant delay target is not a useful goal. And it also must not confuse the related message that TCP should be modified to remove this constraint, which makes it force queuing delay up on low RTT paths, despite the efforts of AQMs to clamp delay.

The many thousands of experiments comparing Curvy RED in our DualQ Coupled AQM with PIE, FQ\_CoDel and RED are reported under the network signalling activity in §3.6, and the associated papers in the appendices. Nonetheless, all experiments used a curviness value of 2. We plan to validate the analytical predictions of this work experimentally, both in terms of the relationship between drop and queuing delay and in terms of scaling with link rate and flow aggregation.

We plan to add this enhanced analysis and full evaluation to the existing report to make a decent paper.

At the IETF meeting in July 2015, this work was presented in the AQM working group, and was greatly appreciated. We also received useful and extensive written feedback, which should improve the final paper. The developers and proponents of PIE and CoDel have not indicated how they intend to respond, e.g. whether they need to change their approach, or at least relax their recommended target delay.

### 2.5 Review of PIE for the IETF

The IETF AQM working group chairs asked for an expert review of the standards track specification of PIE [17] before it is allowed to proceed to completion of the AQM working group stage, which precedes Steering Group approval to

become an RFC. The review is available online and an edited down version is included in Appendix A.5.

The review has received only superficial discussion from PIE authors, developers and proponents in the 4 months since it was submitted. The primary author did apologise for the limited response, saying “Some of Bob’s comments are tough to address :-). It could be a paper on its own. I will update as much as I can in the next week or so.”

### 2.5.1 PIE Review: Main Concerns

The review is a desk analysis of the specification, although it draws on known experimental evaluations of PIE, both by the RITE project and others. It is primarily a code review (of the pseudocode in the specification, but also sometimes of the Linux source available online). It is also a review of the scholarly paper on which PIE is based (the review is actually longer than the paper!). As well as questioning the adequacy of the document as a standards specification, it also includes (sometimes informal) proofs that a number of the techniques, enhancements and algorithms in PIE (e.g. auto-tuning, averaging, derandomisation) are either misguided, redundant or sometimes even counterproductive.

In summary, the review questions the following two lines in the ‘Discussion’ section of the specification:

- “PIE is simple to implement”
- “PIE does not require any user configuration”

It says: “I am afraid I have to say that I do not believe either statement is warranted any more. PIE has lost its way a bit. The implementation has not retained the elegance of the theory. The performance benefit from so-called ‘enhancements’ is questionable or non-existent, whereas the added complexity is very apparent. Also PIE now contains a large number of hard-coded constants (I counted 20) that ought to be scenario-dependent configuration variables.”

The following list is a summary of the concerns raised, or the full review can be found in the above Appendix:

1. Proposed Standard, but no normative language
  - (a) work needed to distinguish between design intent and specific implementation
  - (b) unclear how strongly the enhancements are recommended
2. Has PIE been separately tested with and without each enhancement, to justify each?
3. Needs to enumerate whether it satisfies each AQM Design Guideline
  - (a) If not, say why or fix.
  - (b) Particular concerns:
    - i. No spec of ECN behaviour
    - ii. No autotuning of the two main parameters
    - iii. Transport specific (Reno-based?) autotuning of  $\alpha$  &  $\beta$
4. Rationale for a PI controller not properly articulated
5. Technical flaws/concerns
  - (a) Turning PIE off
  - (b) ‘Autotuning’  $\alpha$  &  $\beta$  parameters
  - (c) Averaging problems
  - (d) Burst allowance unnecessary?
  - (e) Needs a Large Delay to Make the Delay Small
  - (f) Derandomization: a waste of cycles
  - (g) Bound drop probability at 100%  $\rightarrow$  DoS vulnerability?
  - (h) Avoiding Large Packet Lock-Out under Extreme Load.

## 6. Numerous magic numbers

- (a)  $\sim 20$  constants, 13 of which are not in the header block.
- (b) About half ought to be made to depend on other constants
- (c) Need to state how to set the remaining constants for different environments

## 7. Implementation suggestion for Autotuning $\alpha$ & $\beta$

## 2.6 Impact of scheduling on AQM performance

FlowQueue-CoDel (FQ-CoDel) [5] is a scheduling scheme that features prioritization and flow isolation. FQ-CoDel creates one sub-queue per flow and applies CoDel on each of them. The awareness of the latency resulting from over-provisioned buffers has been accompanied by an increase in real-time applications such as Voice over Internet Protocol, video conferencing, interactive Web applications, gaming or financial trading applications.

Since FQ-CoDel features a mechanism that prioritizes low-rate traffic, the benefits for the increasing number of thin streams that carry latency sensitive applications needs to be assessed. Moreover, the potential interest for its deployment might be all the more interesting in the context of bandwidth-limited networks. This paper fills the gap in research evaluating FQ-CoDel when the traffic is a mix of thin streams and bulk flows and evaluates which part of FQ-CoDel (CoDel, prioritization, flow isolation) provides improvement. Because FQ-CoDel features flow prioritization, we also evaluate to what extent its flow starvation prevention mechanism works.

An extended version of what is proposed below can be found in Appendix A.6.

### 2.6.1 Flow isolation of FQ-CoDel and thin streams

We have proposed a custom non-prioritization version of FQ-CoDel, FQ-CoDel Without Prioritization (FQ-CoDel WP). FQ-CoDel WP does not make the distinction between the “new” and “old” list and subqueues are created one after the other, while the scheduler still visits subqueues similar to SFQ. This is used to assess the benefits of the flow isolation for the thin streams applications.

It appears from the results presented in Appendix A.6 that (1) dropping packets with CoDel enables a latency reduction; (2) flow isolation alone cannot reduce the queuing delay experience; (3) flow isolation techniques are sensitive to the traffic load. FQ-CoDel features flow isolation and this results in lower queuing delay than with CoDel alone, showing that the flow isolation technique, along with CoDel drops, can offer the best of the two schemes.

### 2.6.2 Flow prioritization of FQ-CoDel and thin streams

We compare the performance of the default FQ-CoDel with “FQ-CoDel WP 100ms”, which is a modified version of FQ-CoDel in which the target of CoDel is increased to 100 ms (instead of the default 5 ms) and prioritization is disabled. Therefore, the differences between FQ-CoDel WP 100ms and FQ-CoDel are (1) CoDel will be more aggressive and (2) the absence of prioritization in FQ-CoDel.

Based on the results presented in Appendix A.6, we can conclude that the CoDel part of FQ-CoDel is essential to provide low latency in the context of capacity-limited networks. Also, the flow prioritization of FQ-CoDel provides a useful latency reduction and makes the queuing delay of the thin streams less sensitive to the traffic load.

### 2.6.3 Flow starvation prevention of FQ-CoDel

The interarrival time of thin stream packets can impact the prioritization algorithm of FQ-CoDel. Flows with a different pattern of packet-interarrival times but similar packet sizes can be treated differently.

We implemented FQ-CoDel Without its Starvation Prevention Mechanism (FQ-CoDel WSPM) i.e., a version of FQ-CoDel where starvation prevention is disabled and does not move empty subqueues to the “old” list.

The FQ-CoDel thin-stream prioritization mechanisms could therefore be vulnerable to a Denial of Service (DoS) attack where large number of thin streams would be intentionally injected into the network bottleneck to attempt to lock the DRR scheduler in the “new” list, thus starving the “old” list flows. Additional mechanisms may therefore be needed to protect the network from unresponsive flows.

#### 2.6.4 Discussion

We measured that the flow isolation of FQ-CoDel is the main factor resulting in improved latency performance for thin flows over a congested bottlenecks with limited bandwidth. However, such isolation would be impossible when the classifier is presented with a traffic aggregate that it cannot dissect (e.g., when encrypted Virtual Private network, VPN, tunnels are used). We have also identified that the CoDel mechanism in FQ-CoDel provides non-negligible improvements for this specific traffic, which means that when the classifier cannot operate, latency might still be reduced.

Flow prioritization has been introduced within FQ-CoDel to favor low-rate traffic, or latency sensitive applications such as web traffic. We have measured that this scheme does not provide noticeable performance improvements for thin stream traffic, but does makes FQ-CoDel vulnerable to DoS attack.

## 2.7 Rural AQMs

There is no single “typical” bottleneck speed and there are a wide variety of access technologies; fibre deployments in urban areas may offer  $\approx 100$  Mbps, but long-haul DSL used in many developed countries for rural access networks may offer only  $\approx 1$  Mbps and some links may have still lower capacities. In such rural scenarios, bottleneck utilisation is expected to be very high. Higher delays are also sometimes inherent for rural networks since backhaul links might be longer or satellite systems may form a part of the end-to-end path. In addition, bloated buffers could easily add further delay to these networks.

As discussed in § 2.3.3, in the context of capacity-limited networks, the default parameters of CoDel and PIE are not suitable. In deliverable D2.2, we have proposed to change CoDel’s target delay  $\tau$  and interval  $\lambda$  so that they achieve a more desirable trade-off between queuing delay and bottleneck utilization.

To further analyze the benefits of providing a *Rural* parameterization (as opposed to the *Urban* parameterization) that is specific to rural broadband networks, we have decided to focus on CoDel and FQ-CoDel. PIE has already been shown to need an updated parameterization for specific scenarios (such as data-centres [13]). CoDel has been shown to not always control the queuing delay with a limited impact on the bottleneck utilization (Appendix A.3 and [18, 12, 9]), and it was uncertain whether CoDel could be tuned for objectives and network conditions other than the ones for which it has been designed: this led to our interest in examining CoDel and not PIE.

In this deliverable, we have extended the work presented in deliverable D2.2, by providing a parameterization that suits both CoDel and FQ-CoDel and evaluating the RTT sensitivity of our *Rural* parameterization with simulations. We have also used a testbed and emulated specific networks to validate the interest of our parameterization.

In the UK, 21% of rural areas are currently unable to achieve a DSL downlink speed of 2 Mbps [19]. Despite introducing a larger RTT (e.g., 500 ms or more) satellite technology may be deployed in such cases to realise a commercially viable broadband service [20]. We have therefore evaluated our *Rural* parameterization using a real satellite link.

The summarized version of this work that is presented in this section can be found in a more detailed version in Appendix A.7. This work is based on earlier preliminary work [11].

### 2.7.1 Proposing a *Rural* parameterization in ns-2

We have run simulations in ns-2 that let us propose a *Rural* parameterization that for both CoDel and FQ-CoDel that, for capacity-limited networks, both enables a high bottleneck utilisation (around 90% or higher) and limits the queuing delay.

In Appendix A.7, we provide more details on the characteristics of the topology and the traffic that we have generated



to determine the new parameterization of CoDel and FQ-CoDel. We have considered (1) repeated downloads of a file of size 1.7 MB; (2) TCP bulk transfer; and (3) “thin” TCP flows.

We have found out that the choice of an update interval of 300 ms and a target delay of 65 ms yields a reasonable compromise between queuing delay ( $< 50$  ms) and utilisation (slightly above 90%).

### 2.7.2 RTT sensitivity of the *Rural* parameterization using NETEM

The *Rural* parameterization ( $\tau = 65$  ms;  $\lambda = 300$  ms) was derived with an RTT of 300 ms. As deployments will experience various RTTs, we compare *Rural* and *Default* parameterizations with a 1 Mbps link across a range of end-to-end RTTs from 50 ms to 550 ms. The improvement  $imp_M$  for a given metric  $M$  is computed as  $imp_M = (M_{Default} - M_{Rural}) / M_{Default}$  (for FCT in Fig. 2.3a) and the increase  $inc_N$  for a metric  $N$  is computed as  $inc_N = N_{Rural} - N_{Default}$  (for queuing delay in Fig. 2.3b and the bulk throughput in Fig. 2.3c).

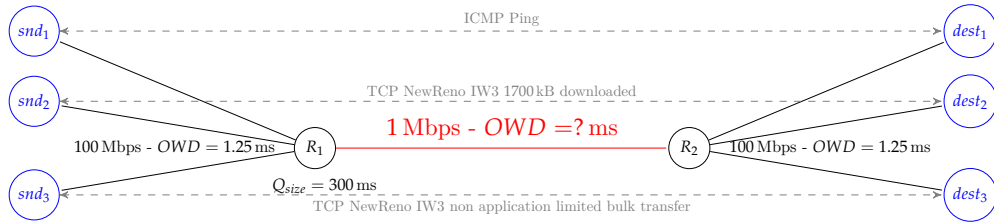


Figure 2.2: Topology and traffic used to assess the RTT sensitivity of the *Rural* parameterization

Figure 2.2 presents the topology and traffic used in this section. The following traffic was considered: (1) repeated downloads of a file of size 1.7 MB, with an inter-file interval exponentially distributed with mean 9.5 s; (2) TCP bulk transfer, lasting for the entire length of the simulation; (3) ICMP pings that are used to measure the queuing delay. interval of 300 ms and a packet size of 150 B; these may represent gaming traffic. Both (1) and (2) use a packet size of 1500 B. The traffic mixed simulated was: one flow for each of flow types (1), (2) and (3)). The flows start randomly within the first second of the experiment. The experiments last 300 s and were run 100 times, each with a different seed. We measure the bottleneck utilisation, the 1.7 MB File Completion Time (FCT) and the queuing delay experienced by the ping flow.

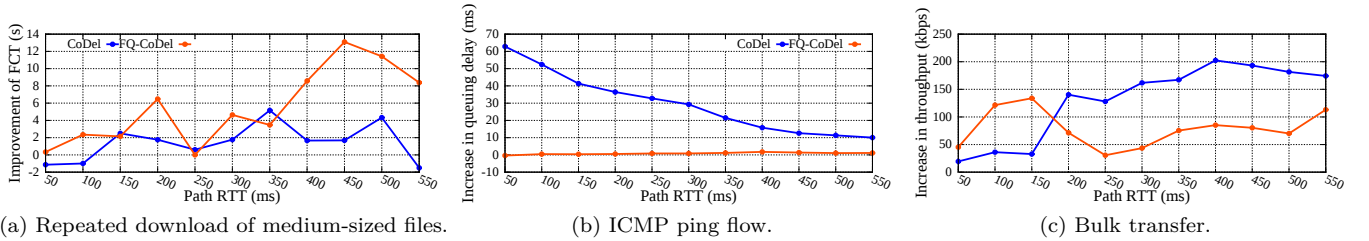


Figure 2.3: Base RTT and *Rural* and *Default* parameterizations

Figure 2.3a shows that *Rural* parameterization improves the FCT for every RTT. In Fig. 2.3b, we can see that the queuing delay for thin streams is increased with CoDel with *Rural* parameterization, because  $\tau_{Rural} > \tau_{Default}$ , but this is not the case with FQ-CoDel, due to its prioritisation scheme. The *Rural* parameterization increases the bulk throughput.

Our *Rural* parameterization proposes an interesting trade-off by increasing the utilisation of the limited capacity and limiting the queuing delay for various RTT values.

### 2.7.3 Assessing the benefits of *Rural* parameterization using a real satellite link

As explained in the introduction of this section, despite introducing a larger RTT (e.g., 500 ms or more) satellite technology may be deployed in such cases to realise a commercially viable broadband service [20]. This section assesses the difference between our parameterization and the default parameterizations, along with providing a comparison between DropTail, SFQ, FQ-CoDel and CoDel.

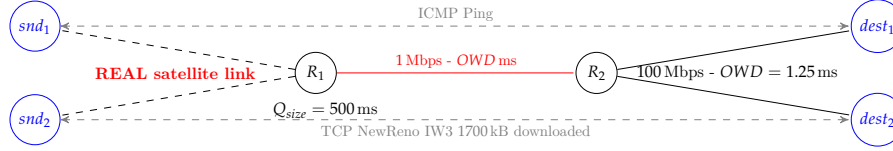


Figure 2.4: Topology used to assess the benefits of using the *Rural* parameterization on a real satellite network

Figure 2.4 shows the topology used in this section. The forward satellite link is between *snd*<sub>1</sub> and *R*<sub>1</sub>, the bottleneck is restricted to 1 Mbps and AQM is used at *R*<sub>1</sub>; in addition to the queue management schemes tested so far, this link allowed also the use of Stochastic FQ (SFQ). There is background traffic that we can not control on the satellite link. Therefore, we only consider small file downloads and measure the FCT and the experienced RTT.

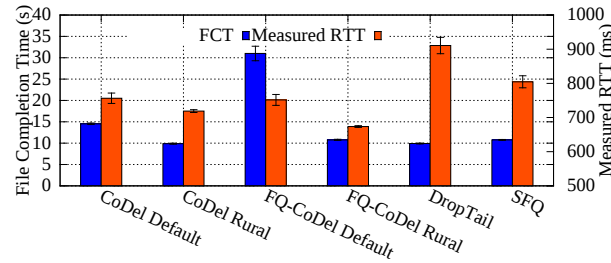


Figure 2.5: FCT and measured RTT over a satellite network

Figure 2.5 shows that introducing CoDel or FQ-CoDel with *Default* parameterization results in (1) a higher FCT than DropTail or SFQ and (2) a lower RTT. With CoDel or FQ-CoDel with *Rural* parameterization, the RTT is lower than with the other evaluated schemes, without negative impact on the FCT. With adequately parameterized AQM, as opposed to with DropTail, the latency is reduced by  $\approx 60\%$  (i.e., by 300 ms), which would seriously improve rural broadband users' experience.

### 2.7.4 Discussion

Consistent with other analysis of PIE, we have proven CoDel can also be tuned to better match the needs of a “non-default” network scenario. We propose a configuration with a larger target delay and update interval for CoDel: this resulted in reduced download time of medium-sized files, higher bottleneck utilisation and limited queuing delay to an acceptable level, for different traffic profiles.

With our parameterization, CoDel and FQ-CoDel would allow a better quality of service for users browsing the web from a rural location since the latency is reduced and capacity utilization is close to that without any AQM scheme.

This work has been presented to EUCNC 2015 and provoked some discussions during the presentation. This work contributes in warning the community that the default parameterizations of CoDel and FQ-CoDel may not result in good performance which would prevent the deployment of AQM in specific scenarios. We intend to put these issues to the authors of the draft that is standardising the CoDel algorithm, as we show in a specific context that updating the interval variable to reflect the RTT is necessary when the RTT is higher than 300 ms.



## 2.8 Interaction between Lower-than Best Effort & AQMs

To ensure the safe deployment of AQM schemes and their ability to tackle the bufferbloat, the IETF works on characterization guidelines [4] which advise to consider a scenario where the impact of introducing AQM on the fairness of Lower-than-Best Effort (LBE) protocols is assessed. Indeed, as shown in [21], when there are AQMs deployed in one router, LBE may fail in their design goal, that is to exploit the remaining and unused capacity of a link. To further assess this problem, this section presents simulation results where we propose different parameterizations of the AQM and the LBE protocols used in [21]. We expect to verify the validity of their conclusions when the targets of the protocols are different.

The LBE protocols that are considered in our work are LEDBAT [22] and CDG [23]. We focus on LEDBAT because it is deployed in BitTorrent and on CDG because it has been shown to be a good LBE candidate [24, 25]. The difference with the work in [21] is that we consider a better parameterization for LEDBAT (following [26, 27]) and that they do not consider CDG which estimates the state of the router's queue and might therefore be able to cope with AQM. Since LBE schemes use estimations of the state of the queue or the queuing delay, an AQM that allows higher amount of buffering than the ones considered in [21] should be considered: we complement their conclusions by adding simulation results with PIE that allows on average 20 ms queuing delay, whereas CoDel allows only 5 ms queuing delay.

The summarized version of this work that is presented in this section can be found in a more detailed version in Appendix A.8.

### 2.8.1 Congestion window evolutions

Figure 2.6 describes the topology used in this section. The traffic generated is the following: (1) one non application-limited LBE bulk flow using LEDBAT or CDG as congestion control policy between  $snd_1$  and  $dest_1$ ; (2) one non application-limited TCP bulk flow using NewReno as congestion control policy between  $snd_2$  and  $dest_2$ . The Initial congestion Window (IW) is set to 10 packets for (2); the SACK option is enabled for (1) and (2). To avoid any synchronization effect, (1) randomly starts in  $[0; 1]$  s and (2) in  $[1; 2]$  s.

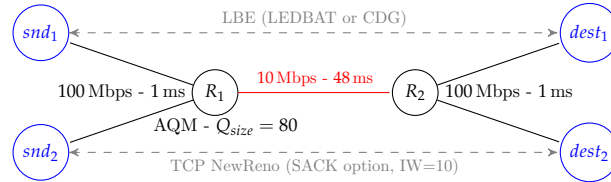


Figure 2.6: Topology and traffic

We denote by  $\tau_{LB}$  the target delay of LEDBAT,  $\tau_{PIE}$  the target delay of PIE and  $\tau_{CoDel}$  the target delay of CoDel. Each scenario is entitled “LBE-AQM”. As one example, “LB25-ARED” is the scenario where the LBE protocol is LEDBAT with  $\tau_{LB} = 25$  ms and the AQM is ARED. Each simulation run lasts 50 seconds. Our work is based on NS-2 simulations; LEDBAT, CDG and any TCP variants used in this article use the TCP-Linux updated to linux kernel version 3.17.4 in NS-2.<sup>2</sup>

Figure 2.7 shows the evolution of the congestion window of the TCP NewReno and the LBE flow, for various AQM schemes and different values of  $\tau_{AQM}$ .

The queuing delay is maintained around 20 ms with PIE ( $\tau_{PIE} = 20$  ms). LEDBAT will let itself introduce  $\tau_{LB}$  queuing delay; this results in LEDBAT not being LBE when  $\tau_{LB} \geq 25$  ms. With  $\tau_{LB} = 5$  ms (which is lower than  $\tau_{PIE}$ ), LBE is possible. With CoDel as an AQM, no values of  $\tau_{LB}$  let LEDBAT able to provide an LBE service, which is in accordance with the results published in [21]. This is due to the low value of  $\tau_{CoDel}$  that makes CoDel try to maintain a 5 packets queue. In these conditions, there is not much room for an LBE service.

The congestion window of CDG is lower than the one of TCP New Reno, with PIE. CDG seems able to carry out LBE services in these conditions. However, with CoDel, the targeted queuing delay is so small that, even with CDG,

<sup>2</sup>More details at: <http://heim.ifi.uio.no/michawe/research/tools/ns/index.html>

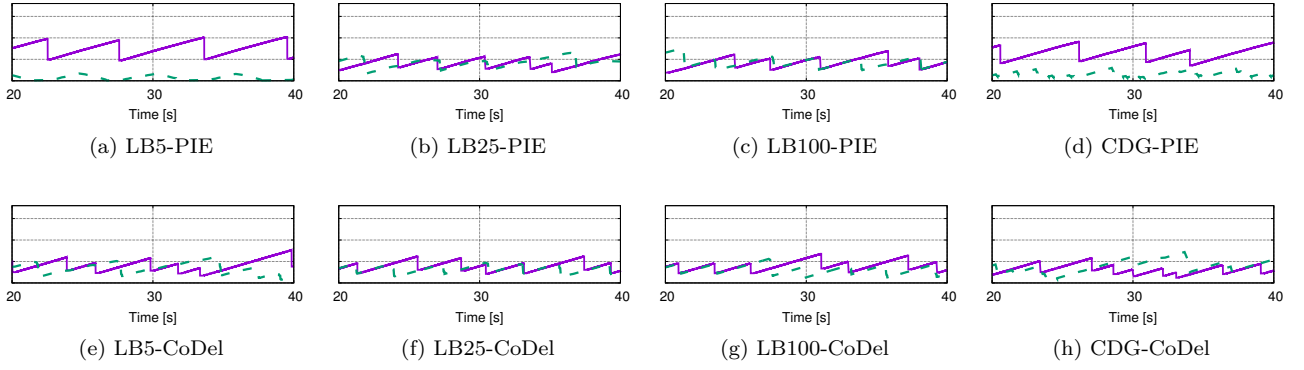


Figure 2.7: Congestion window evolution

no LBE service seems possible.

### 2.8.2 Discussion

This work assesses the capacity of CDG and LEDBAT to carry out LBE services when there is an AQM on the path. We summarize the results presented in this paper in Table 2.1. This table features cases with DropTail and ARED that are not presented in this summary: more details can be found in Appendix A.8

Table 2.1: Possible coexistence of LBE and AQM

	DropTail	ARED $\tau = 5$ ms	PIE $\tau = 20$ ms	CoDel $\tau = 5$ ms
LEDBAT $\tau_{LB} = 5$ ms	✓	✓	✓	✗
LEDBAT $\tau_{LB} = 25$ ms	✓	✗	✗	✗
LEDBAT $\tau_{LB} = 100$ ms	✗	✗	✗	✗
CDG	✓	✓	✓	✗

We believe that the deployment of CDG or LEDBAT with a target delay of 5 ms as LBE is possible. Also, because the target delay of CoDel is lower than the one of PIE, no LBE service is possible if CoDel is deployed, whereas it may be possible with PIE. We expect, as a future work, to verify this conclusions with experiments in a real testbed, using the recently posted implementation of CDG for Linux kernels [28].

## 2.9 AQM detection active measurement tool

Pointer to appendix A.10

The type and details of queue management mechanisms that are deployed at the congestion points of networks are not always publicly known. Knowing the kinds of a bottleneck drop scheme is crucial in understanding the performance of the network and choosing how to develop the right end-host mechanisms. The best-known queue scheme is tail-drop FIFO that only drops packets when its buffer is full. Active queue management schemes (AQM) start throwing packets out of their buffer as soon as they realize a long queue has become persistent. We present an end-to-end measurement method to detect the kind of queue scheme at a bottleneck router<sup>3</sup>. We have developed an active measurement tool

<sup>3</sup>We use the term router throughout, although more generally other devices might act as bottlenecks

and evaluated our measurement methodology on an experimental test-bed. The results of our experiments show that the proposed approach provides the basis for building a tool to identify some AQMs deployed on the bottleneck automatically.

### 2.9.1 Queue Detection Approach: AQM or Tail-drop

The path between a *Sender* and a *Receiver* consists of a sequence of *network routers* that forward packets to the next hop. When sending a flow of data, one such router along the path has the least available bandwidth for the flow; it is considered the *bottleneck* of this path. If the *Sender* sends packets quicker than the bottleneck router of the path can handle, it begins to fill up its queue, and the router must eventually drop some of the packets.

Consider the FIFO case; The tail-drop FIFO establishes a single queue for each outgoing link, and forward packets on that link in the order in which they arrive. Packets are dropped only when the queue is full. On the other hand, AQM policies proactively drop packets when the queue starts to fill up before reaching the state when the queue is completely full [29, 30, 31]. As a result, a simple way to distinguish them is to look for packet loss occurring before the queue is full.

Suppose the *Sender* transmits a *periodic packet stream* to the *Receiver*. The stream consists of  $n$  packets, *Maximum sized* (MSS), with  $t$  seconds inter-transmission time (ITT). The *Sender* adds a timestamp  $t_k$  and sequence number to the payload of each packet  $k$  ( $1 \leq k \leq n$ ) prior to its transmission. Having the transmission time  $t_k$  and the arrival time  $a_k$ , the *Receiver* computes the *queueing delay* of all received packets. To do so, first the *Receiver* calculates the one-way delay (OWD) for all delivered packets as  $D_k = a_k - t_k$ ; the *queueing delay* of packet  $k$  would be  $Q_k = D_k - D_m$ , where  $D_m$  is the minimum OWD. Looking at the packets' *queueing delay* in combination with loss occurrences, the *Receiver* can infer at which point the queue saturated, and whether any loss happened before that point or not. We here show how the measured queueing delay and loss patterns can provide information about the bottleneck router's queue management type.

If the stream rate is larger than the available bandwidth of the bottleneck router, the queue of the router gradually builds up. Therefore, packet  $k$  will wait in the queue for a longer time interval than packet  $k - 1$ . Consequently, the packets' *queueing delay*  $\{Q_1, \dots, Q_{k-1}, Q_k, \dots, Q_n\}$  has an increasing trend while the queue is being filled up. If  $n$  is large enough, at some point, the queue becomes full and some of the packets get dropped.

Normally, the rate at which packets enter the queue becomes equal to the rate at which packets leave the queue, when the queue is saturated. Therefore, the packets that the bottleneck router receives after its queue is filled up experiences approximately the same *queueing delay*. If there is any packet loss while the *queue delay* is growing, before the *queueing delays* get approximately constant, the *Receiver* can infer that the bottleneck router is employing AQM as its queue management. This approach is designed to work in the presence of variations in other traffic, but the additional noise will have to be filtered out.

### 2.9.2 Experiments

To test the proposed method, we developed an *active measurement tool*. The tool is composed of a *Client* process running at the client and a *Server* process running at the server. The tool uses UDP for the periodic packet stream. There is also a TCP connection between the two end-points which is used to transfer control messages such as the stream specifications, the abortion or the end of measurement process, etc.

Figure 2.8 shows the network topology employed to test the proposed approach. The *Client* and *Server* machines are connected through two Linux machines: one *bottleneck router* and one emulating network delay using *Netem*[32]. The links between the machines are all 1Gbps, except for the link between the *Router* and the *Netem* machine, where we have used *ethtool* to reduce the capacity to 10Mbps. The Linux *router* uses *tc* to shape different bottleneck AQMs and *tail-drop* queue schemes into the *router*. We tested all the queue schemes with their default parameters which are listed in Table 2.2.

We first tested the proposed approach where there is no background traffic. The results are shown in Figures 2.9a and 2.10a. Figure 2.9a illustrates the *queue delay* variations of four periodic streams from the client to the server. All four streams have  $n = 3000$  packets and  $t = 500\mu s$  ITT (*MSS* size). *CoDel* [31], *PIE* [29], and *ARED* [30] will, as expected, drop some packets before their buffer is fully saturated. Tail-drop queues, (*packet-based* and *byte-based*), do

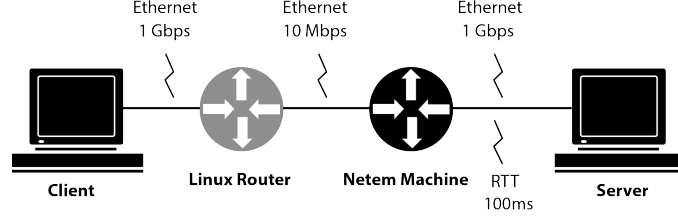


Figure 2.8: The Network Topology

Table 2.2: Different queue parameters

Queue Type	Parameters
Packet-based	limit 1000p
Byte-based	limit 1514000b
ARED	limit 1514000b $th_{min}125Kb$ $th_{max}375Kb$
CoDel	limit 1000p target 5.0ms interval 100.0ms
PIE	limit 1000p target 5.0ms interval 30.0ms

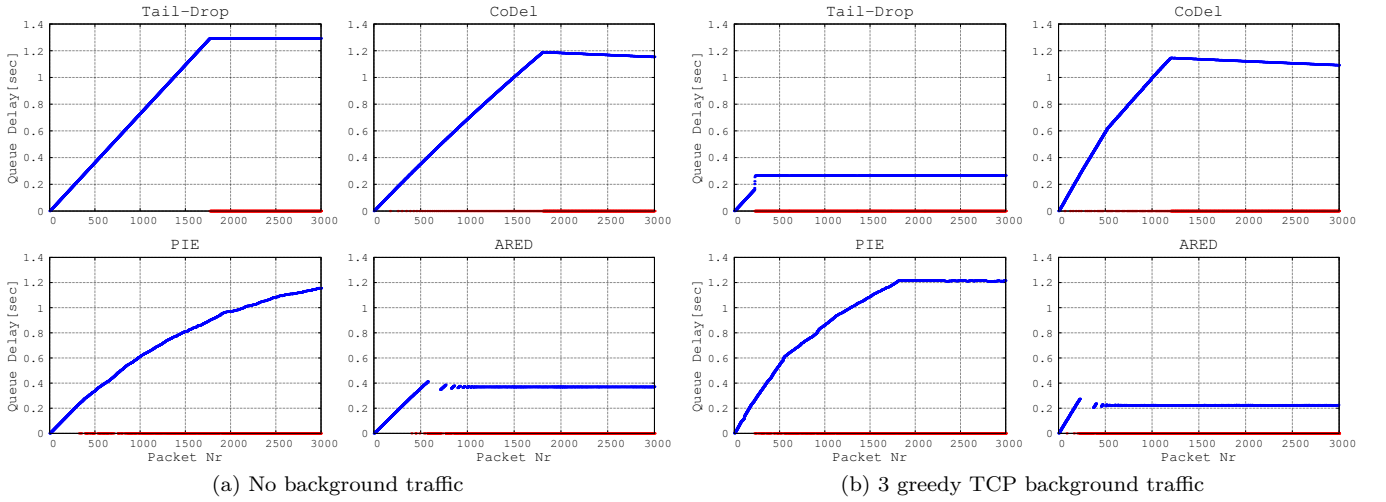


Figure 2.9: Queue delay for different queue schemes - blue dots show the received packets' queue delay in seconds, red dots are the packets which got dropped

not show losses until after the queue is full.

Looking at the queue delay curves along with the packet losses (Figure 2.9a), the *Server* can differentiate between AQM and tail-drop. However, to figure out which kind of AQM is being employed in the bottleneck, we need to look closer at the drop policies' *loss patterns*. We used a Gaussian kernel density estimator [33] to show different *loss distributions* produced by the five drop policies (Figure 2.10a). To do that, we consider each sequence number  $k$  as a data point ( $1 \leq k \leq n$ ). We then compute the *loss density* for each data point  $L_k$  using (1):

$$L_k = \sum_{\forall j} \frac{1}{\sqrt{2\pi}} e^{-\frac{|k-j|}{2}} \quad (1)$$

Where  $|k - j|$  is the distance from data point  $k$  to the packet loss with sequence number  $j$ . Figure 2.10a illustrates *loss density* curves for different drop policies. All AQMs gradually drop packets, while tail-drops show zero loss rate at first and a significant increase in loss occurrences when there is no more space in their buffer. Once *CoDel* starts dropping packets, it constantly reduces the time between two packet drops because the stream does not slow down

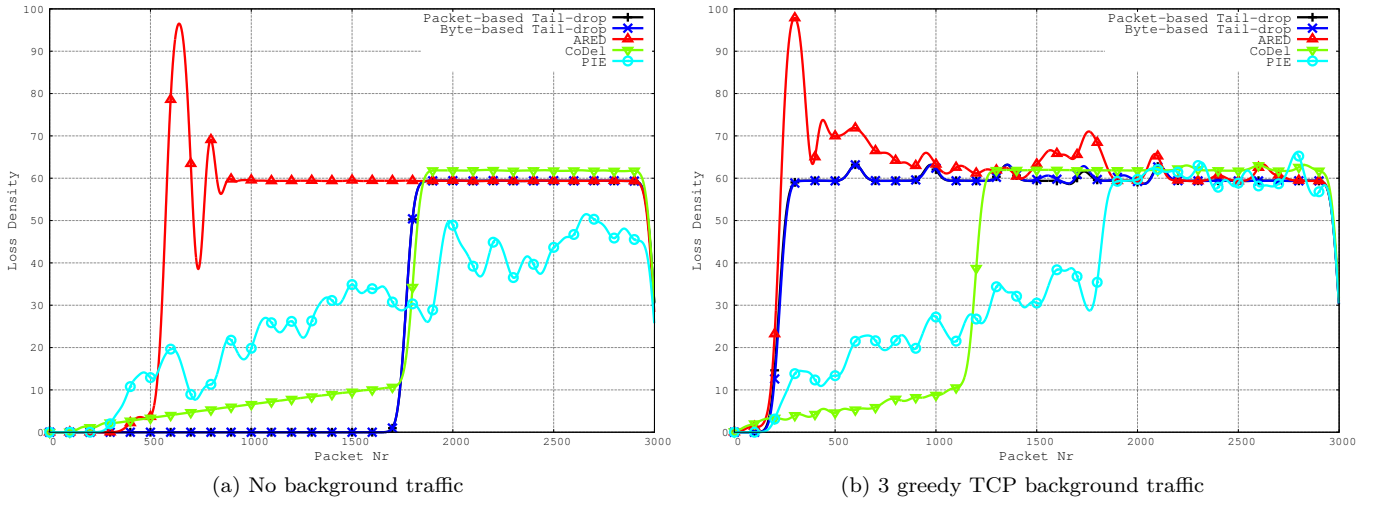


Figure 2.10: Loss Density for different queue schemes

and the queue delay stays above the threshold. This effect is visible in the *CoDel* loss density curve. The *ARED* loss density grows notably after receiving 500 packets. Then the loss rate decreases and stays almost constant for the rest of the test period. Compared to *CoDel* and *ARED*, *PIE* shows more variable loss density. This could be explained by the fact that *PIE* drops packets at enqueue using a probability, which is calculated periodically based on queue delay.

Ideally we would prefer to apply the *queue detection tool* when there is no other traffic; without any competing traffic, detecting varying tail-drops and AQMs would be easier. However, situations may occur where a probe may be needed even in the presence of other traffic. Figures 2.9b and 2.10b show that the results are distinguishable also in the presence of three greedy TCP flows. Differentiating between different types of AQMs might however become difficult because of the noise introduced by the background traffic. What happens here is that the measurement traffic forces competing TCPs to withdraw by creating a huge number of losses. A question arising is why *queue delays* still start from *zero* while there are other streams in the queue. The reason is that the *Server* uses minimum *OWD* ( $D_m$ ) to calculate each packet's *queue delay*. Consequently, the fastest packet (the packet with the minimum *OWD*) is defined to have *zero* queue delay in both scenarios.

Historically, router performance has been measured in terms of either packet throughput or byte throughput. As a result, routers have been built with either a flat buffer that implements byte-based queueing, or buffers that are structured as maximum-sized cells that can hold one packet each and thus, implement packet-based queueing. Both of these architectures (and probably more) are deployed in the Internet, and we know from other work [34] that they affect different kinds of traffic differently.

The idea behind separating packet-based from byte-based queues is simple. Byte-based queues<sup>4</sup> will allow small packets to fit in the end of a nearly-full queue, giving small packets an advantage. Packet-based queues, however, will treat small and large packets in the same way. Therefore, we suggest that by observing the loss rate behaviour of a queue exposed to different packet sizes, we can deduce whether the queue is byte-based or packet-based.

We further performed experiments about this phenomenon and the results are discussed as follows. Figure 2.11 depicts the loss rate behaviour of a *packet-based* versus *byte-based* tail-drop queue when the packet size changes from very small packets (96bytes) to MSS sized packets. For each packet size, the *Client* sends 3000 *back-to-back* packets to the *Server*. The loss rate of a *packet-based* tail-drop queue does not depend on the packet-size, while a *byte-based* queue will drop more packets as the packet-size increases. Even when there are three greedy TCP background flows, the difference between *packet-based* and *byte-based* tail-drop is still clearly observable.

<sup>4</sup>At least if byte-based tail-drop is implemented naively—to avoid packet-size bias some byte-based schemes drop an arriving packet if the buffer is already one maximum-sized segment from being full [34].

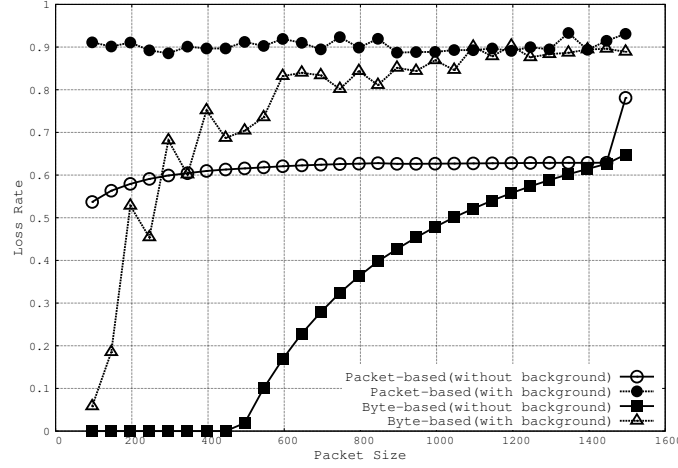


Figure 2.11: Packet-based VS. Byte-based

### 2.9.3 AQM Detection Tool: Status and Plans

Currently, we are working on a paper to submit to “ValueTools” focusing on the tool itself and another one to “Passive and Active Measurement” (PAM) with the full evaluation results and analysis, both from “in-the-wild” measurements and from the testbeds. We are also planning on making the source of the tool available by the end of the RITE project, once the results are stable enough.

## 2.10 MADPIE

In § 2.3, we have illustrated that PIE is RTT sensitive. We have measured wide oscillations in queuing delay when the RTT increases. This results in temporarily high dropping probability, low bottleneck utilization and high maximum queuing delay. To reduce the RTT sensitivity of PIE and improve the performance of latency sensitive applications over large RTT paths, we propose to mix the ideas behind CoDel and PIE to (1) control the maximum queuing delay with deterministic drops, following CoDel’s principle and (2) control the average queuing delay with random drops, following PIE’s principle. Our proposal, Maximum and Average queuing Delay with PIE (MADPIE) extends PIE and adds deterministic drops to prevent the queuing delay from growing beyond critical points.

For more details on the summary that is proposed in the rest of this section, see the current version of a paper included in Appendix A.9.

### 2.10.1 Adding deterministic drops in PIE

MADPIE uses the same random drops as PIE, the main difference between the two algorithms being that we add a deterministic drop in MADPIE. On top of the parameters of PIE (such as  $\tau_{RD}$  (PIE’s target delay) and  $\lambda_{RD}$  PIE’s interval), MADPIE only requires two internal parameters that are: (1)  $\tau_{DD}$ , the queuing delay above which the deterministic drops occur and (2)  $\lambda_{DD}$ , the frequency at which the  $p_{max}$  is updated. Every  $\lambda_{DD}$ , if the estimated queuing delay is higher than  $\tau_{DD}$ ,  $p_{max}$  is set to 1;  $p_{max}$  is set to 0 by default and otherwise. In MADPIE, if a packet is not dropped nor marked by the random drop algorithm, and if  $p_{max} == 1$  then the packet is dropped or marked and  $p_{max}$  is set to 0. There can be a maximum of one deterministic drop every  $\lambda_{DD}$ .

### 2.10.2 Proof of concept

The aim of this section is to illustrate how MADPIE differs from PIE when the RTT increases.

$n_{DD}$ ,  $n_{RD}$ ,  $n_{BO}$  and  $n_{tot}$  respectively represent the number of drop events induced by a deterministic drop (DD) (MADPIE only), a random drop (RD) (PIE and MADPIE), a buffer overflow (BO) (PIE and MADPIE) and the total



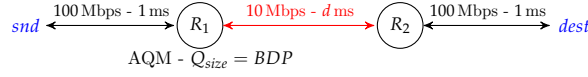


Figure 2.12: Topology used to prove the concept of MADPIE

number of drops.  $r_x = n_x/n_{tot}$  is the ratio of the drop events induced by the event  $x$ . Figure 2.13 shows  $r_{RD}$  and  $r_{DD}$  for PIE and MADPIE as a function of the queuing delay when the drop occurred.

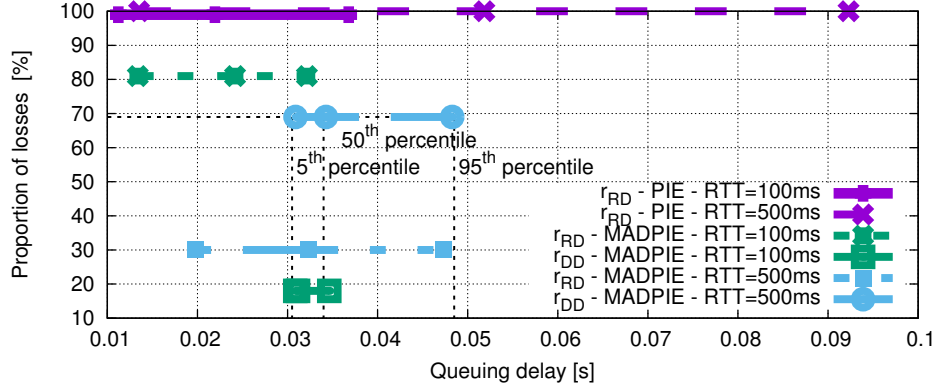


Figure 2.13: Queuing delay and proportion of losses

As one example (dashed lines in Figure 2.13), when the RTT is 500 ms and the AQM is MADPIE,  $r_{DD} \approx 70\%$  and when the deterministic drops occurred, the 5 % percentile of the queuing delay was  $\approx 30$  ms, the 50 % percentile  $\approx 34$  ms and the 95 % percentile  $\approx 48$  ms. With PIE, most of the drops are induced by the PIE algorithm and not buffer overflow and when the RTT is 500 ms, the queuing delay raises up to more than 90 ms. With MADPIE, when the RTT is 100 ms, the random drop part of MADPIE is responsible for more than 80 % of the drops, whereas when the RTT is 500 ms, the deterministic part of MADPIE is responsible for around 70 % of the drops with a consequent queuing delay reduction. Therefore, MADPIE drops more packets when the RTT increases and this prevents high queuing delays.

### 2.10.3 Traffic mix and RTT mix

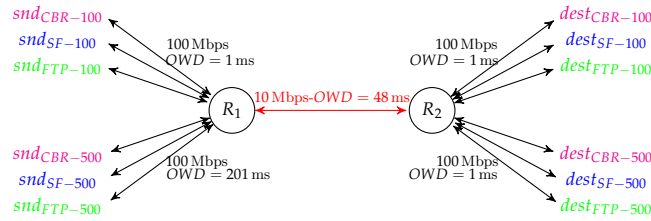


Figure 2.14: Topology and traffic mix used to evaluate the RTT sensitivity

Figure 2.14 presents the topology used in this section.  $N_{X-Y}$  represents the number of flows for the application  $X$  on the  $Y$  ms RTT path. Between  $snd_{CBR-100}$  and  $dest_{CBR-100}$  (resp.  $snd_{CBR-500}$  and  $dest_{CBR-500}$ ),  $N_{CBR-100}$  (resp.  $N_{CBR-500}$ ) Constant Bit-Rate (CBR) flows use User Datagram Protocol (UDP) with a sending rate of 87 kbps and a packet size of 218 B. Between  $snd_{SF-100}$  and  $dest_{SF-100}$  (resp.  $snd_{SF-500}$  and  $dest_{SF-500}$ ),  $N_{SF-100}$  (resp.  $N_{SF-500}$ ) flows transfer files of  $S$  kB ( $S \in 15; 44; 73; 102$ ). Between  $snd_{FTP-100}$  and  $dest_{FTP-100}$  (resp.  $snd_{FTP-500}$  and  $dest_{FTP-500}$ ),  $N_{FTP-100}$  (resp.  $N_{FTP-500}$ ) TCP bulk flows are generated. TCP flows use CUBIC as congestion control policy. All the flows randomly start between 0 s and 1 s. Each run lasts 100 s and is repeated 20 times. The metrics are sampled ever second (except for the queuing delay and the one way delay that sampled per-packet). We choose to present the

results with  $N_{CBR-100} = N_{CBR-500} = 4$ ,  $N_{SF-100} = N_{SF-500} = 20$  and  $N_{FTP-100} = N_{FTP-500} = 2$ . As the RTT of the paths are not the same, the queue size at  $R_1$  is set to the BDP of the higher RTT.

More information on the traffic generation can be found in Appendix A.9.

**2.10.3.1 CBR traffic - between  $snd_{SF-X}$  and  $dest_{SF-X}$**  The results for the CBR traffic are shown in Figure 2.15. We provide in Figure 2.15a an example to explain how to read Figure 2.15b.

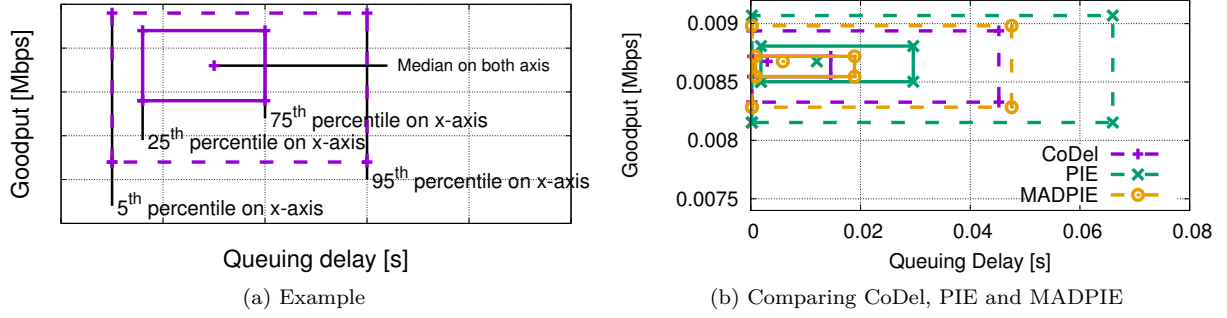


Figure 2.15: Goodput and queuing delay - CBR traffic for which  $RTT = 100$  ms

Due to the presence of the flows that experience an RTT of 500 ms, the flows that experience an RTT of 100 ms face a queuing that momentarily raise above 60 ms with PIE (*i.e.* 55 % of the one way delay). Since the default *target delay* of CoDel is 5 ms, the allowed queuing delay is lower than with PIE and MADPIE, for which the *target delay* is set to 20 ms. By default, CoDel would maintain a lower queuing delay than PIE, which leverages the impact of the flows that experience an RTT of 500 ms. Also, because CoDel uses deterministic drops, the queuing delay can not raise much higher before the first drops are induced. These results show that MADPIE takes the best of the two schemes: with MADPIE, the median queuing delay is lower than with PIE and the queuing delay is kept under control, and the higher values of the queuing delay are close to those with CoDel. While with PIE, the queuing delay momentarily increase above 60 ms (*i.e.* 55 % of the RTT), with MADPIE, the introduction of the flows that experience an RTT of 500 ms have less impact: the queuing delay momentarily raises above 40 ms, that is 45 % of the RTT. MADPIE provides a latency reduction of  $\approx 13\%$  for the 75<sup>th</sup> percentiles and of  $\approx 21\%$  for the 95<sup>th</sup> percentiles and the performance are close to those of CoDel.

**2.10.3.2 Small files download - between  $snd_{SF-X}$  and  $dest_{SF-X}$**  We show in Figure 2.16 the downloading time of small files for the flows that experience an RTT of 100 ms (Figure 2.16a) and an RTT of 500 ms (Figure 2.16b).

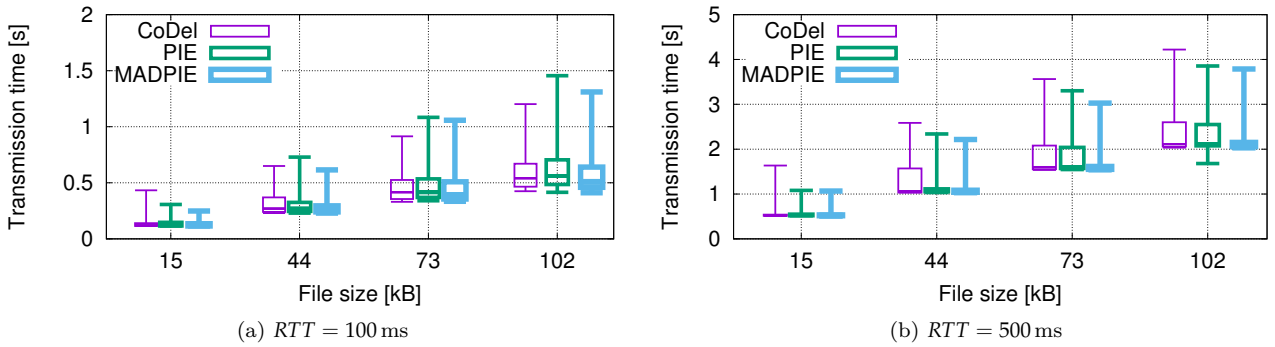


Figure 2.16: Small file downloading time



The 5<sup>th</sup>, the 25<sup>th</sup>, the 50<sup>th</sup> and the 75<sup>th</sup> are quite the same whether the AQM is PIE, MADPIE or CoDel; the 75<sup>th</sup> percentile is slightly lower with MADPIE. The 95<sup>th</sup> percentile is always lower with MADPIE. When the file size is higher than 73 kB, the 95<sup>th</sup> percentile is slightly lower with CoDel or MADPIE than with PIE, probably thanks to the lower queuing delay that is experienced by packets that have been dropped and retransmitted.

For the small files downloads that experience a base RTT of 500 ms, the performance are quite the same for CoDel and PIE: with PIE, the bottleneck utilisation is higher (see § 2.10.3.3 for more details) and with CoDel the queuing delay is lower (see § 2.10.3.1 for more details). MADPIE somehow takes the best of the good schemes and thus, provides lower downloading times for small files.

**2.10.3.3 Bulk flows - between  $snd_{FTP-X}$  and  $dest_{FTP-X}$**  We show in Figure 2.17 the goodput of the bulk transfers for the flows that experience an RTT of 100 ms (Figure 2.17a) and an RTT of 500 ms (Figure 2.17b). CoDel shows a lower goodput than PIE, which is due to its lower *target delay*: PIE allows more buffering. MADPIE slightly reduces the bottleneck utilization for the bulk flows that experience an RTT of 100 ms. The same happens for the flows that experience an RTT of 500 ms. With MADPIE, the resulting goodput is a trade-off between CoDel and PIE.

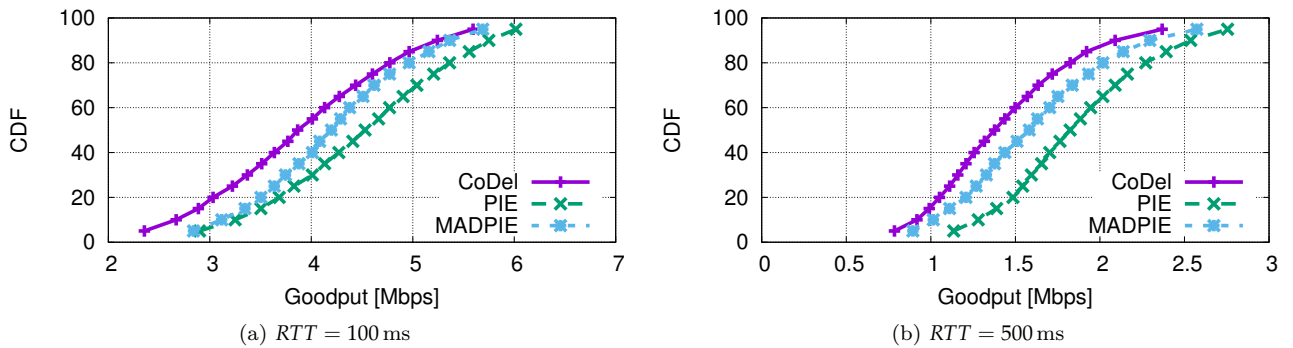


Figure 2.17: Bulk flow goodput

#### 2.10.4 Discussion

The results presented in this section confirm that AQM must be deployed for the sake of latency sensitive applications. Indeed, even with a buffer that is not over-provisioned, we measure that the queuing delay without AQM can drastically increase, resulting in (1) high end-to-end delay for VoIP applications, (2) high transmission time for small files. The benefits of not using AQM is only seen for bulk transfers, where goodput is slightly reduced when AQM is introduced.

We modified the PIE algorithm to make it less dependent on the RTT of the path. MADPIE extends PIE by adding, on top of the random drops, deterministic drops similar to those CoDel's algorithm introduces. Through simulation we found that our modification does not affect PIE when the RTT is lower than 100 ms. The deterministic drops are more dominant when the RTT increases, which results in lower maximum queuing delays and better performance for VoIP traffic and small files downloads at the expense of bulk transfers losing a little capacity.

#### 2.10.5 MADPIE Status

The paper in Appendix A.9 was submitted to a journal by the end of August 2015 <sup>5</sup>.

<sup>5</sup>Paper co-authored by N. Kuhn and D. Ros; because of double-blind submission, authors are not listed and the project not acknowledged yet

## 2.11 Discussion on squared AQMs

One of the difficulties of defining AQM algorithms is caused by the non-linearity of the control signal (drop or mark) to the rate that classic TCP endpoints are sending. To be "TCP friendly" classic TCP implementations need to produce a sending rate that is inverse proportional to the square root of the signal probability. As a result the load or number of flows is proportional to the square root of the signal probability.

This non-linearity has a few consequences for the design of AQMs:

- Non-linear algorithms are usually too complex to handle at packet rate. As a result complex algorithms are performed at a lower rate, or heuristic/empirical algorithms are defined that have no theoretical basis but experimentally behave "good enough".
- The operation range, with respect to the number of flows, of an AQM is bounded, depending to the range where the configuration parameters are effective (producing results that are within the objectives of the AQM).
- Adaptation algorithms are defined to extend the operation range. Typically they are applied step-wise, potentially adding oscillation-like effects to the traffic.
- To handle square roots of numbers smaller than 1 (typically smaller than 1%), twice the precision is required for the internal variables, compared to the non-square-rooted values.
- The AQM cannot be used to control scalable congestion controllers.

From our work in the network signals activity on AQMs for scalable congestion controllers (meaning the number of flows or load is proportional to the signal probability) we came to the conclusion that classic AQMs can be linearised in a very simple way. Figure 2.18 shows the concept of "Thinking Twice at the Output". By applying a squaring function just before applying the signal probability, the AQM can work on the linear scalable probability, and the feedback to the congestion controller is adapted to the square root that is part of the response function for classic congestion controllers. A squared probability is very simple to implement. If  $p$  is compared with 2 random values instead of one, the probability is effectively squared. Generating (or lookup of pre-generated) random values is done at line speed today. Applying this simple extension removes all above mentioned disadvantages.

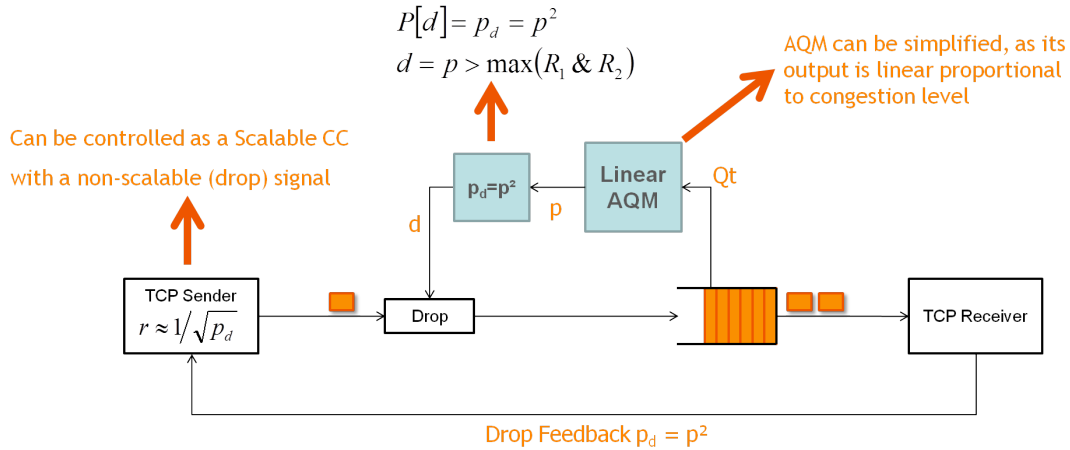


Figure 2.18: Linearity of Thinking Twice at the Output of an AQM

The following two subsections briefly describe two examples where a squared AQM design can benefit. These two have been considered in the RITE project, but the application of this concept to other (novel) AQMs might become useful in the future.

### 2.11.1 A simpler RED: Curvy RED

Figure 2.19 shows a RED AQM on the left that controls an average queuing delay proportional to the number of flows for a scalable congestion controller (such as DCTCP and Relentless TCP). It consists of a single linear mapping starting at the origin ( $p=0$  if  $q=0$ ), and has to be squared before being applied as a drop probability for Classic TCP controllers. The resemblance between Curvy RED and RED (here supporting multiple piece-wise linear segments) can easily be seen in Figure 2.19. The SW/HW code to implement the Curvy RED and the precision of the variables is much smaller than that for standard RED, and more complex piece-wise linear RED variants (such as Gentle-RED). For more info on Curvy RED, see § 2.4. The idea is also evaluated as part of the DualQ Coupled AQM (see § 3.6), where it is used as the Classic Q AQM for the drop based flows. The DualQ Coupled AQM is implemented and evaluated in the Video and Web testbed as part of WP3 work.

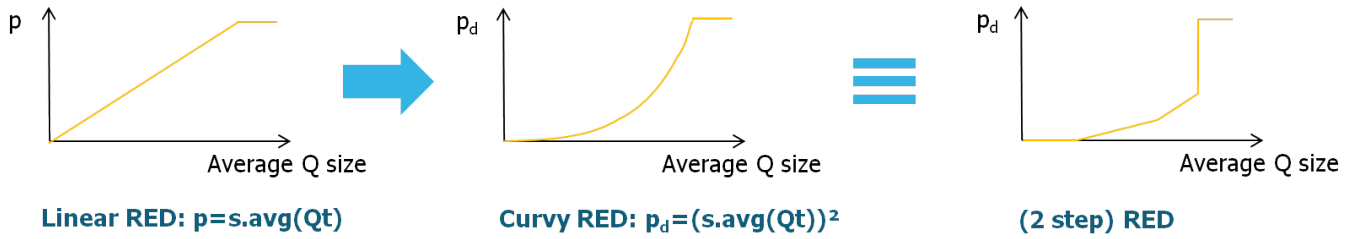


Figure 2.19: Curvy RED: a linear AQM which can be approximated by a more complex (2-step piece-wise linear) RED AQM

### 2.11.2 A simpler PIE: $PI^2$

PIE [17] has been proposed as an improvement to the original PI controller based AQM. Several improvements on PIE have been proposed, but it can be both simplified and optimized by the "Thinking Twice at the Output" concept.

As the PI controller needs to control a non-linear system (Classic TCP flows), it needs to adapt its proportional ( $\beta$ ) and integral ( $\alpha$ ) gain factors to the level of congestion. The PIE implementation proposes to "correct"  $\alpha$  and  $\beta$  based on the current  $p$  value in a stepwise manner, to keep the control process in a stable regime.

Figure 2.20 shows our  $PI^2$  proposed implementation that avoids the need for the stepwise gain factor adaptations, which results in full load-spectrum support, while still using a normal PI controller. The PIE paper [8] proposed initially to divide the gain factors by 2 for  $p < 10\%$  and to divide by 8 for  $p < 1\%$ . Table 2.3 shows the changes in the output drop probability of  $d$  on different input deviations for both PIE and  $PI^2$ . As can be seen, the output values of PIE and  $PI^2$  are very close together, suggesting that the  $PI^2$  controller will work at least as efficiently as the PIE controller, and over a wider operating range.

In our review of PIE for the IETF we hinted towards the benefits of squaring PIE's drop probability (§5.2 of Appendix A.5). We showed that PIE's step-wise auto-tuning could be achieved more simply and robustly by multiplying the auto-tuning parameters by the drop probability.

Given that these ideas arrived relatively late in the project,  $PI^2$  has been implemented in ns2, but not fully evaluated yet. Further ns2 evaluation, implementation and testbed evaluation will be needed, but results might not be available within the scope of the RITE project due to the limited time left.

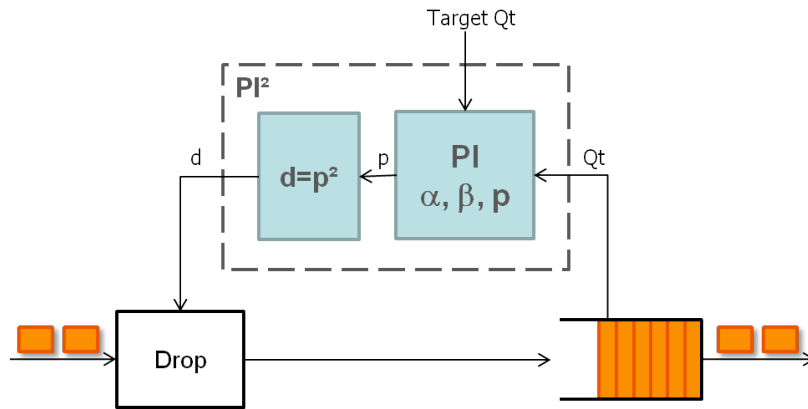


Figure 2.20:  $PI^2$ : no need to autotune  $\alpha$  and  $\beta$  to level of drop probability

parameter	PIE			$PI^2$		
Internal p	30,00%	3,00%	0,30%	54,77%	17,32%	5,48%
I-gain $\alpha$	0,125	0,0625	0,015625	0,125	0,125	0,125
P-gain $\beta$	1,25	0,625	0,15625	1,25	1,25	1,25
$\Delta$ Integral	$\Delta d$ by PIE			$\Delta d$ by $PI^2$		
100ms	1,2500%	0,6250%	0,1562%	1,3849%	0,4486%	0,1525%
10ms	0,1250%	0,0625%	0,0156%	0,1370%	0,0434%	0,0138%
1ms	0,0125%	0,0062%	0,0015%	0,0136%	0,0043%	0,0013%
-1ms	-0,0125%	-0,0062%	-0,0015%	-0,0136%	-0,0043%	-0,0013%
-10ms	-0,1250%	-0,0625%	-0,0156%	-0,1367%	-0,0431%	-0,0135%
$\Delta$ Proportional	$\Delta d$ by PIE			$\Delta d$ by $PI^2$		
100ms	12,5000%	6,2500%	1,5625%	15,2555%	5,8926%	2,9318%
10ms	1,2500%	0,6250%	0,1562%	1,3849%	0,4486%	0,1525%
1ms	0,1250%	0,0625%	0,0156%	0,1370%	0,0434%	0,0138%
-1ms	-0,1250%	-0,0625%	-0,0156%	-0,1367%	-0,0431%	-0,0135%
-10ms	-1,2500%	-0,6250%	-0,1562%	-1,3536%	-0,4173%	-0,1213%

Table 2.3: Output comparison between the stepwise corrected PIE at its targeted midpoints and the full load-spectrum corrected linear  $PI^2$  AQMs.

### 3 Network signals: ECN markings and interactions with TCP

The integration and evaluation of the different mechanisms relying on network signals that have been proposed in the context of WP2 and presented in D2.1 and D2.2, has helped converging these ideas into a selection of mechanisms that offered promise towards our objective to reduce latency. The overall results of this activity can be structured in four phases: (1) defining the benefits of using ECN; (2) proposing a roadmap laying out paths to reach those benefits; (3) proposing prototypes that realise crucial steps of the roadmap; and (4) identifying future potential for further improvements.

The work items are mapped to the relevant sub-sections as follows:

- 1 Our work to define the full range of benefits of ECN is discussed in § 3.1;
- 2 With these amazing benefits in mind, we have developed an ECN Roadmap to reach them. Pre-requisites were to measure what the current deployment of ECN is (§ 3.2) and to identify that the challenges of extending ECN deployment are, not only at layer 3, but also below (§ 3.3). Then in § 3.4 we present the roadmap towards full exploitation of the low latency potential of ECN;
- 3.1 Prototype 1: one first step of the roadmap consists of easy-to-deploy changes that incentivize wider deployment of ECN in routers, clients and servers, presented in § 3.5;
- 3.2 Prototype 2: the second step of the roadmap is smarter interactions between end nodes and routers, presented in § 3.6;
- 4 Further improvements at the end-node level to more fully achieve the amazing benefits of ECN are presented in § 3.7.

#### 3.1 ECN benefits

Explicit Congestion Notification (ECN) is a TCP/IP extension to signal network congestion without packet loss, which has barely seen deployment though it was standardised and implemented more than a decade ago.

In a congested network that has not deployed ECN in the network devices, packets may be lost where the transport congestion control algorithms have not responded to the congestion quickly enough for reductions in the congestion window to mitigate such losses. When a loss occurs this loss needs to be detected by the sender and the packet retransmitted. This incurs path latency. For TCP without ECN, the congestion window is only reduced in response to packet loss, which makes loss a necessary occurrence for the operation of TCP. A bulk sender will cause the congestion window to grow until a loss is experienced and then reduce and retransmit the lost packet.

For applications such as VoIP, gaming, or streaming video this loss, and the delay until retransmission, can be very disruptive, which is one of the reasons many VoIP and streaming video implementations have used UDP based protocols rather than TCP defining their own congestion control algorithms.

Other causes of packet loss, such as radio interference, might still be present. However, through the use of ECN, packet loss can be avoided in all but the most extreme congestion, and congestion control algorithms can become more responsive by providing explicit signalling of congestion on the path. In order for ECN to be effective however, it requires end-to-end signalling and the co-operation of network devices (routers, middle boxes, etc) on the path.

##### 3.1.1 ECN's lack of deployment

Explicit Congestion Notification (ECN) is an important indicator that next generation end-system transport protocols can use to significantly reduce path latencies [35]. Despite existing for more than 15 years, ECN is still to see significant deployment and use in the Internet. However, this could be set to change for two key reasons: (1) Increasing awareness of the benefits of reducing latency and (2) Growing momentum to deploy Active Queue Management (AQM) methods in network devices, an essential pre-requisite to enabling ECN.

### 3.1.2 Potential ECN benefits in the future

Our work in RITE to encourage the use of ECN by end systems is documented in detail in D1.3. This identifies two initiatives:

- Section 6.1 of D1.3 proposes a simple adjustment to TCP’s response when receiving ECN-marked packets (this alters the multiplicative back-off factor of TCP). This can bring significant improvements to latency on paths where AQM methods are deployed. The mechanism is called “Alternative Backoff with ECN” (ABE). It simply consists of a parameter adjustment to  $\beta_{ecn}$  to the recommended values in [36]. ABE’s performance has been evaluated in a RITE work published as [36] (see Appendix B.5) in collaboration with an external partner and has been taken up by RITE for standardization in the IETF as [37] (see Appendix B.4).
- Section 6.2 of D1.3 looks at how a new class of transport protocol designed for use within data centres (e.g. DCTCP), and how the protocol design can evolve to work across an ECN-enabled Internet path. Evaluation of DCTCP has shown that marking immediately in the network, with accurate feedback to the receiver, and a sender response proportional to the level of congestion can together result in really low queuing latency. This work also enabled us to understand that, without ECN, AQMs cannot clamp down too hard on queuing delay, because TCP pushes up loss too high as a consequence (see § 2.4 about Constraints on AQMs from TCP). Section 3.6 will further show that the sender response of DCTCP makes the congestion signal scale with the throughput. This allows TCP to keep its sawteeth very small, which allows the AQM to use a very low queuing threshold without compromising utilisation. In turn, this leads to not only low delay, but low variation in delay. The explanation of all these points has been brought together in the rationale section of a paper under submission, and included in Appendix B.7 While it requires a change in the semantics of ECN and a need for new transport mechanisms, this approach can pave the way for a Low-Latency, Low-Loss and Scalable (L4S) transport service.

### 3.1.3 Today’s ECN benefits

In addition to the above two initiatives that incentivize the use of ECN either in its current standard form or by changing its semantics, Section 2 of the RITE IETF draft on ECN Benefits [35] identifies benefits to Internet applications that use an ECN-enabled transport. ECN avoids the inefficiency of dropping data that has already made it across at least part of the network path. This immediately translates into a (slight) gain in throughput because packets that would have otherwise been dropped are marked (but still delivered). While this is a minor benefit [35], it can yield larger gains as a consequence – e.g. reduced Head-Of-Line Blocking (HOL-Blocking) delay for transports such as TCP that provide in-order delivery of received data segments (see the results of testbed experiments using ABE in WP3).<sup>6</sup>

Other benefits of using ECN include: a reduced probability of RTO expiry, making incipient congestion visible, and benefits to applications that do not retransmit lost packets. We encourage the reader of this document to refer to D1.3 as well as [35] (see Appendix B.6) for detailed information.

## 3.2 ECN path support evaluation

The University of Aberdeen has collaborated with ETH Zurich<sup>7</sup> to perform a number of measurements to determine:

1. the proportion of webserver that would provide ECN when requested
2. the proportion of network paths that would fail if ECN negotiation was attempted
3. the proportion of network paths that would successfully pass ECN signalling information

<sup>6</sup>§§ 5.2 & 5.3 of D1.3, give an evaluation and discussion of different ways to remove HOL blocking at different layers, in HTTP/2, in QUIC, and in RITE’s own Inner Space proposal). If ECN were available, all these techniques would be less necessary, although HOL blocking is only one form of latency that they target.

<sup>7</sup>ETH Zurich’s contributions were supported by the European Commission through the Seventh Framework Grant Agreement mPlane (FP7-318627)

Table 3.1: ECN negotiation statistics, of 581,711 IPv4 hosts and 17,028 IPv6 hosts, all vantage points, 27 Aug - 9 Sep 2014, compared to previous measurements.

IPv4		IPv6		2011	2012	Description
hosts	pct	hosts	pct	pct	pct	
<b>326743</b>	<b>56.17%</b>	<b>11138</b>	<b>65.41%</b>	11.2%	29.48%	<b>Capable of negotiating ECN</b>
324607	55.80%	11121	65.31%	–	–	...and always negotiate
2136	0.37%	17	0.11%	–	–	...sometimes negotiate, of which...
107	0.02%	1	0.01%	–	–	negotiation depends on path
27	0.02%	0	0.00%	–	–	sometimes reflect SYN ACKflags
248791	43.23%	3961	26.23%	82.8%	70.52%	Not capable of negotiating ECN
2013	0.35%	83	0.48%	–	–	...and reflect SYN ACKflags
6177	1.06%	1929	11.33%	–	–	Never connect with ECN

The ecnspider tool was developed to perform these measurements. This tool was a highly parallellised HTTP client that enabled and disabled the Linux kernel's ECN negotiation option and opened connections in batches to connect to a pre-resolved list of webserver. A collection of scripts were developed and used to mangle packets for the third test, measuring path transparency for ECN signalling information.

Three vantage points were used to conduct the experiments. These vantage points were connected to the Internet (they were not intended to test edge network equipment, such as home routers). The servers at these vantage points were running Ubuntu 14.04 (kernel 3.13.0 without syn retry fallback as in RFC3168), run by commercial hosting provider Digital Ocean, in London, New York, and Singapore. Initial investigation showed that all ECN signalling works properly on this provider's networks, and all sites have native dual-stack connectivity.

The sample of webserver used for these measurements were, as with similar studies in the past, chosen from Alexa's publicly available top million websites list. The website names were then resolved to at most one IPv4 and one IPv6 address per site, eliminating duplicate IP addresses. Name resolution was performed on the 27th of August 2014 from the London vantage point using Google's public DNS server, resulting in 581,737 unique IPv4 addresses and 17,029 unique IPv6 addresses.

Packet captures were performed during the measurements for later analysis using QoF, an IPFIX Metering and Exporting process, derived from the YAF flowmeter, designed for passive measurement of per-flow performance characteristics. This allowed us to analyse the IP and TCP flags on received packets in flows generated by the ecnspider tool.

The first measurements were performed from the three vantage points from the 27th of August to the 9th September 2014. These measurements aimed to determine the proportion of webserver with ECN support and the proportion of paths that would be broken should an attempt be made to negotiate ECN in order to detect ECN dependent path failure.

The first interesting result from these tests is that over half of the website hosts tested were capable of negotiating ECN. This is compared to two previous studies in 2012 where less than 30% of hosts were capable of negotiating ECN. This increase has been attributed to the enabling by default of ECN "server mode" in the Linux kernel, which allows servers to provide ECN support when it is requested by clients but will not request ECN itself.

For IPv4, 1.06% of hosts showed a failure to connect when ECN negotiation was attempted, however for IPv6 this was over 11%.

Analysis of the flows was performed to determine which ECN-related IP and TCP flags were seen. The results in table 3.2 are grouped by IP version and also by whether or not the flow had successfully negotiated ECN, or reflected the TCP ECN flags on the SYN/ACK.

It was observed that there were 6939 IPv4 and 2013 IPv6 hosts that always negotiated ECN but never send an ECT codepoint as seen from any of the three vantage points. While the specification for ECN does not require the marking of every packet sent with an ECT codepoint, this may be an indication of middleboxes (e.g., firewalls, traffic normalisers, front-end load balancers, service level agreement enforcement, and other boxes that perform deep packet inspection and subsequent modification of the network packets). on the path that could be bleaching these bits in the IP header.

Table 3.2: Signaling anomalies - relationship between ECN IP and TCP flags, of 581,711 IPv4 hosts and 17,028 IPv6 hosts, all vantage points, 27 Aug - 9 Sep 2014.

Marking	IPv4 (N=581711)			IPv6 (N=17028)		
	ECN	Reflect	No ECN	ECN	Reflect	No ECN
only ECT(0)	315605	693	1995	8998	1	46
ECT(0) + ECT(1)	0	0	0	4	1	7
ECT(0) on SYN ACK	7780	0	46	89	0	82
only ECT(1)	3	1	17	0	10	12
ECT(1) on SYN ACK	4	0	16	7	0	31
only CE	11	1	7	0	0	48
CE + ECT	5	2	0	23	66	39
CE on SYN ACK	11	0	5	22	0	87
none	6939	1343	243150	2013	5	3694

The anomaly with the potential to create the biggest problem here was ECN being negotiated but all packets being received being marked with CE. There were only 11 cases of this occurring with IPv4 and no cases of this occurring with IPv6. Further, there were 5 cases with IPv4 and 23 cases with IPv6 where an ECT codepoint and CE marking was seen, potentially indicating that a CE marking router was present on the path.

There were 2688 hosts for IPv4 and 47 hosts for IPv6 where ECN had not been negotiated however ECT(0) markings were seen on the packets. Further, CE markings were seen from 8 hosts with IPv4 and 48 with IPv6 where ECN had not been negotiated. Because it was only possible to perform captures at one end of the connection, this may be for a number of reasons. ECT(0) flags may be being reflected, TCP flags may be being bleached where successful negotiations have occurred but we were unable to verify this, or a middlebox on the path may be incorrectly setting these bits.

To further verify that ECN signaling is correctly working end-to-end, we ran CE and ECT blackhole experiments on the 24th of September 2014, and ECE and CWR response tests on 23rd September 2014 both from the London vantage point.

While the ECE response test succeeded for IPv4 in 94.8% (309,842 hosts) of all ECN-enabled cases, the CWR response test succeeded only in 44.3% (144,290 hosts) of the cases. Further, we found 690 IPv4 hosts responding with ECE and 351 hosts responding with CWR even though ECN had not been successfully negotiated. There also appears to be significant impairment or implementation error in ECN signalling for IPv6, with only 7 hosts responding ECE and 9 CWR.

We have shown that while webserver support for ECN continues to increase, there does not appear to have been any appreciable reduction in the proportion of potential connectivity failure linked to ECN since 2011. The vast majority of connectivity problems we found with ECN negotiation were close to the server, i.e., cases in which routing changes during a connection would not lead to connection failure in the middle of an ECN-enabled flow. The fallback behaviour defined in RFC 3168 was shown to eliminate connectivity risk for these cases, such that enabling ECN by default would lead only to increased connection latency when attempting to connect to about five of every thousand websites.

The paper describing these measurements, B. Trammell, M. Kühlewind, D. Boppart, I. Learmonth, G. Fairhurst, and R. Scheffenegger, Enabling Internet-Wide Deployment of Explicit Congestion Notification appears in Proceedings of the Passive and Active Measurement Conference 2015, pp 193-205. This work was also presented to the Internet Congestion Control Research Group at the 2015 Dallas IETF meeting.

A smaller scale of the experiment was repeated by the University of Aberdeen in collaboration with Simula Research Laboratory to test from additional vantage points using a multi-homed server connected to NorNet. A new DNS resolution was performed for the top 100,000 websites from the Alexa Top Sites list on the 2nd February of the list also retrieved on that date. This resolution was performed from AS224 (UNINETT) again using Google's public DNS server. Attempts to contact the resolved webserver using IPv4 and IPv6 were made from AS224 and AS5381 (POWERTECH). While the original study had used three vantage points that were geographically distributed, they were all from the same provider. These extra vantage points were on an academic and a commercial ISPs networks and so could be used to verify that our original results are representative of behaviours in other networks.



A total of 97743 IPv4 and 6892 IPv6 unique addresses were resolved from the list of 100,000 websites. For IPv4, there was no ECN-dependent connectivity failures in 97.7% of cases. In 1.4% of cases the hosts appeared down and only in 0.08% of cases were there apparent ECN-dependent failures in connectivity. In 57.5% of cases, ECN could be successfully negotiated with the end host.

For IPv6, there was no ECN-dependent connectivity failures in 95.51% of cases. In 4.4% of cases the hosts appeared down and only in 0.2% of cases were there apparent ECN-dependent failures in connectivity. In 57.5% of cases, ECN could be successfully negotiated with the end host.

These numbers show a small increase in the proportion of sites where ECN could be successfully negotiated and a small decrease in the number of ECN-related failures, although it should be noted that the first study looked at the top 1,000,000 sites where this smaller scale study only looked at the top 100,000 websites.

While our study shows that it is not particularly unsafe for operating system vendors to activate ECN on the client-side by default, we cannot yet unreservedly recommend doing so by default. For a tiny minority of sites (about one in 60,000), we cannot rule out path-dependent connectivity issues not solved by fallback as in RFC 3168. A similar proportion of sites exhibit indiscriminate CE marking, which would cause severe throughput degradation with use of ECN. These numbers are small enough that targeted collaboration with the operations community based on additional measurement is a viable way forward, coupled with development of ECN probing techniques that are able to detect such anomalies and respond by the endpoints disabling use of ECN. Such techniques have been proposed within the TCPM working group of the IETF but have not yet been developed.

In summary data provided by RITE measurements is encouraging to the deployment of ECN as a latency-reducing mechanism. Server-side support has been observed to be more widely deployed, and paths generally allow ECN marks to propagate. This shows substantial improvement with respect to previous studies. The data also suggests that the mechanism to negotiate use of ECN would likely behave correctly if enabled by clients. Related work within RITE has led to a change in the IETF's recommendations, encouraging vendors and operators: RFC 7567 (IETF Recommendations Regarding Active Queue Management) now recommends that routers should support ECN and provide means to configure this, and a further RITE-authored document (ECN Benefits, to be published 2015) will motivate ECN deployment within the Internet, and explain the benefits in terms of reduced latency.

There is however a need for further action to complete deployment of ECN and enable this to be enabled by default in clients:

- There is a need to benchmark home network equipment to ensure the edge ISP networks can support ECN.
- There is a need to continue to address implementation errors in stacks and middle boxes.
- There is a need to complete work on new mechanisms to verify that paths correctly mark ECN-capable packets.
- Encouragement to develop and deploy ECN marking techniques and transport protocols that can utilise these signals.

### 3.3 Guidelines for Adding ECN to Protocols that Encapsulate IP

This section is an edited down version of a similar summary given in the previous (internal) deliverable D2.2 released in Aug'14. Progress since then is reported in §3.3.2 on standards co-ordination, status and plans.

#### 3.3.1 ECN Encapsulation: Motivation

The benefits of ECN outlined in §3.1 can only be fully realised if support for ECN is added to the relevant subnetwork technology, as well as to IP. When a lower layer marks a packet with ECN, the marking needs to be explicitly propagated up the layers. The same is true if a buffer marks the outer header of a packet that encapsulates inner tunnelled headers.

Because IP is the interoperability protocol across all subnet technologies, the addition of ECN to IP effectively changed the interface to every lower layer protocol. When ECN was first standardised in 2001, the specification [38] recognised

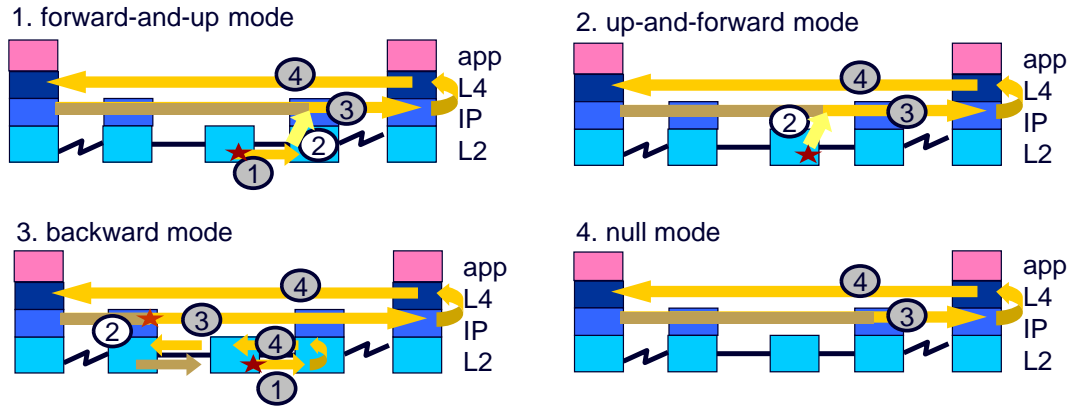


Figure 3.1: The Four Arrangements for Propagating Congestion Notification up to the IP Layer

that a programme of work would be necessary to define ECN propagation by amending lower layer protocols and tunnelling protocols. That programme has already produced standards for propagation of ECN up from MPLS to IP (as well as from MPLS to MPLS) [39] and for ECN propagation from any IP-in-IP tunnel [40].

However, rather than a few individuals with ECN expertise working through an endless list of layer 2 protocols and tunneling protocols, it was decided to spread the load of this task, and produce guidelines so that others without detailed knowledge of ECN could co-operate in this task. The draft guidelines for propagating ECN between lower layers and IP are reproduced in Appendix B.3.

The main contribution is to classify any lower layer protocol into one of the four arrangements shown in Figure 3.1. It is then possible to give generic guidelines applicable to each arrangement, and therefore cover all known lower layer protocols.

Figure 3.2 shows two motivating examples for ensuring that ECN markings propagate correctly from lower layers.

Figure 3.2a shows the prevalence of tunnels in the 3GPP LTE System Architecture Evolution (SAE). Given that ECN is now mandated in the MAC spec for the Evolved Universal Terrestrial Radio Access Network (E-UTRAN) [41], ECN propagation will need to be correctly defined for the tunnelling protocols used, e.g. the General Packet Radio Service (GPRS) Tunnelling Protocol (GTP—see Appendix B.3 for the full list of references).

Figure 3.2b shows schematically a small selection of traffic flows through a data centre. ECN is more widely used within private data centres than in the public Internet. In a multi-tenant data centre, as traffic passes into the network, it will be encapsulated (usually in the hypervisor) in order to ensure forwarding isolation between tenants unless explicitly routed through a higher layer gateway. There is yet to be convergence on one tunnelling standard, with contenders being NV-GRE, VXLAN and MPLS, therefore ECN propagation has to be defined for all these (again, see Appendix B.3 for the full list of references). Another layer of encapsulation might also be used when traffic passes from one data centre to another over wide area networks.

Whether or not there is any tunnelling between hypervisors, an outer layer of Ethernet encapsulation will always be added to IP packets in order to cross a data centre. It has proved difficult to find space to add ECN to the IEEE 802 family of protocols. But fortunately, layer 3 switches are widely used, which use the outer layer 2 header for forwarding control, but they can bury into an inner IP header for transport control functions such as Diffserv and ECN.

This is an example of an important role of the guidelines; to explain the conditions under which a lower layer protocol will *not* need to be modified to support ECN. They describe two modes of operation that allow ECN to be used across a large number of existing lower layer technologies without modifying any protocol standards, as long as the caveats in the guidelines are satisfied:

- the ‘L3 switch’ case already described for Ethernet above;
- and the ‘null’ mode, where the lower layer subnet is non-blocking by design.

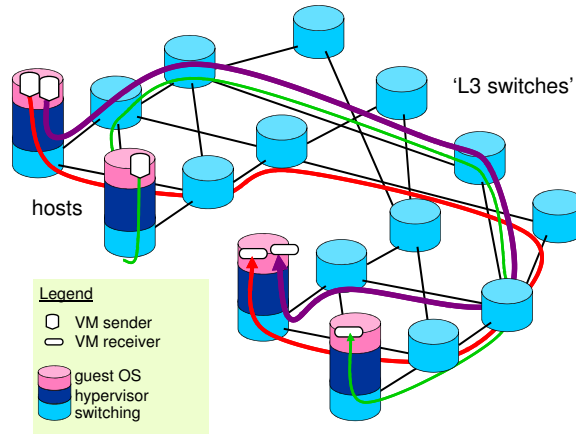
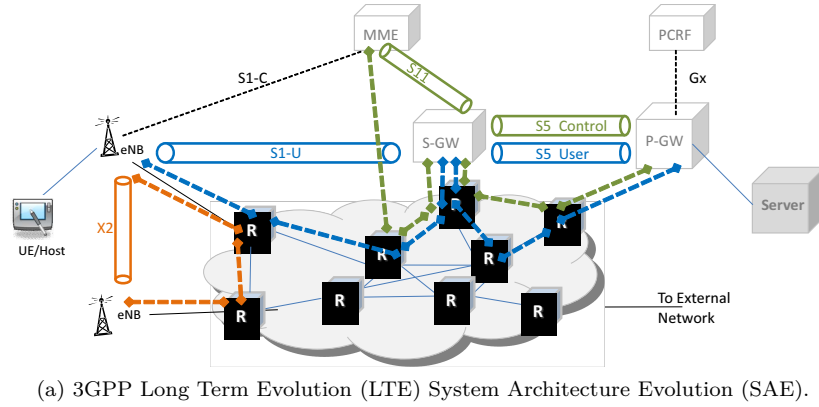


Figure 3.2: Motivating Examples of ECN Encapsulation.

Also, there is a large class of tunnelling protocols that encapsulate IP-in-IP but with another shim protocol between the two. The guidelines explain how these can all be updated in a single standards action.

### 3.3.2 ECN Encapsulation: Standards Co-ordination, Status & Plans

It was decided to write the guidelines as an IETF document (Appendix B.3), given that the IETF is the standards authority for IP. However, the guidelines define the interface between IP and numerous lower layer protocols, many of which (but not all) are under the authority of other standards bodies.

The draft guidelines have already been used by the authors of extensions to the TRansparent Interconnection of Lots of Links (TRILL) protocol, which is owned by the IETF TRILL WG. Indeed, they have now used the guidelines twice, because TRILL extensibility was redesigned.

The draft guidelines were adopted onto the agenda of the IETF's transport services working group (tsvwg) in March 2014. Since then, we have prepared liaison requests for the IETF leadership to approve and to send to the IEEE and the 3GPP. These liaisons are reproduced respectively in Appendices B.1 & B.2.<sup>8</sup>

The IEEE liaison period passed without comment, which the liaison officer expected, because the IEEE is not currently working on congestion notification technology.

<sup>8</sup>We originally recruited two co-authors for the draft who already work in these two standards bodies (one holding the position of liaison officer between the IETF and IEEE802.1, and also being an expert in IEEE congestion notification technology).

The 3GPP liaison period is still open for comments<sup>9</sup> Liaison with the 3GPP is expected to be more controversial, because the 3GPP has incorporated ECN into a number of its systems for real-time multimedia, and there is concern that some of the approaches could be incompatible with the IETF's end-to-end definition of ECN.

Also, the IETF tsvwg will be liaising internally with other IETF working groups, e.g. TRILL (which is already engaged), the Network Virtualization Overlays (NVO) Working Group and the Internet Area WG (int), which is responsible for the Generic Routing Encapsulation (GRE) and many other tunnelling protocols.

### 3.4 ECN Roadmap

Early experience of using ECN across the general Internet encountered a number of operational difficulties when the network path either failed to transfer ECN-capable packets or inappropriately changed the ECN code points. Our recent survey (in collaboration with external partners, see § 3.2) reported a growing number of network paths that at least allow ECN code points to pass unhindered. Nonetheless, TCP with ECN enabled has not been used widely in the current general Internet.

More recently ECN has been deployed in controlled environments - for example, within a data centre or within a well-managed private network where routers can be configured to support ECN. In this context, Data Centre TCP (DCTCP) has been proposed and is available within a range of operating systems. DCTCP is described in an Internet Draft published by the authors. DCTCP was not intended for use in the general Internet, not because it would not work at scale, but because DCTCP traffic would not co-exist with existing Internet traffic, so it would require all end-systems and network nodes to be upgraded at once.

In the public internet, ECN is widely supported on servers, however turned off by default on clients, largely because, during Microsoft's early attempt to add ECN to Windows Vista, a number of home gateways black-holed ECN connections or even crashed. A recent announcement from Apple has proposed turning on ECN on clients. On routers there is currently hardly any deployment of ECN for marking ECN capable packets. Many Cisco router models implement ECN, but no operators are known to have turned it on in production networks. ECN is now on by default in (FQ-)CoDel in mainline Linux which can be expected to be deployed in home routers.

Section 3.1 highlights the benefits that ECN deployment could provide. ECN being deployed now does not exploit its full low latency potential. Classic ECN just removes the latency induced by losses, but not queuing. However smart ECN techniques, such as the dual queue (§ 3.6), would involve interactions between network routers, servers and clients. Therefore, RITE proposes the following roadmap for the large scale deployment of ECN to happen.

#### Step 1a: Socialize the Benefits

As well as creating an IETF RFC on the benefits of ECN (§ 3.1), various inherent benefits of ECN have been a repeated theme in many of the dissemination activities of the RITE project.

#### Step 1b: Establish the Deployment Baseline

Establish the existing extent of ECN deployment and the various path behaviours that could challenge ECN deployment. The RITE project has contributed to the most recent evaluation of ECN path support (§ 3.2), building on earlier investigations.

Part of the reason for liaising between the IETF, IEEE and 3GPP (§ 3.3) has been to establish readiness for ECN and the presence of any blockages, such as alternative uses of the ECN field in data centres or mobile networks.

#### Step 2: Sender-only changes

RITE has developed a simple, easy-to-deploy update to TCP that may accelerate ECN deployment. This new mechanism is called the TCP Alternative Backoff for ECN (ABE—see § 3.5). This has been implemented in Linux and has been specified in an IETF Internet-Draft.

<sup>9</sup>Due to an oversight, the IETF had approved the statement, but omitted to send it until July 2015.

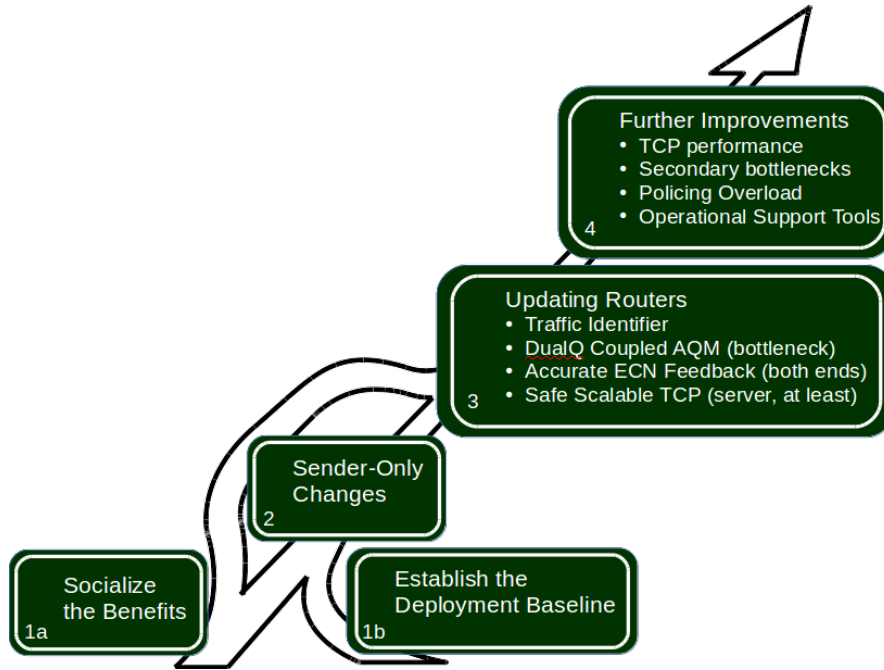


Figure 3.3: ECN Roadmap

Activating ABE on the sender-side should incentivize clients and routers to enable ECN, because it improves throughput by solving the low utilization problem that PIE and CoDel suffer from in longer RTT scenarios.

### Step 3: Updating routers

It is believed that network operators will need a highly compelling reason to deploy ECN in their networks, which in many cases will also involve requiring their equipment vendors to implement ECN support.

It is possible that Apple’s initiative to turn on classic ECN in their devices could persuade operators to move, although the mere absence of loss might not be a sufficient motivator. It is possible that ABE might give just enough extra motivation to make operators move.

Nonetheless, we believe that operators need to be motivated by the end-game, in order to take the first step. The RITE project has expressed the full power of ECN in its ‘Data Centre to the Home’ demonstrator and testbed (§3.6). We have demonstrated one of the new products it enables (Panoramic Interactive Video), which has already captured the imagination of the marketing divisions of a number of companies, not least industrial partners in RITE. There is even a possibility that operators might jump straight to the end-game (termed L4S for low latency, low loss, scalable service) without passing through the first step of classic ECN deployment.

The L4S service requires the following deployment sub-steps:

1. Accurate ECN feedback (§4.4) on both client and server, which is in the process of standardization in the IETF’s TCPCM WG;
2. An agreed Identifier for L4S traffic (informal discussions have started in the IETF, but no formal proposal has been made—see §3.7.1);
3. Scalable TCP at least on the server designed to be safe alongside existing traffic (TCP Prague: §3.7);
4. The DualQ Coupled AQM in at least the downstream bottleneck (§3.6).

One possible deployment scenario is that a network operator initially arranges all these steps, solely to provide L4S service for its own value-added services. This would apply equally in DSL, cable or mobile scenarios, although we have focused on DSL in RITE, where the Broadband Network Gateway (BNG) is downstream bottleneck.

Another possible scenario is that L4S sees initial deployment in data centres, in order to deploy low latency DCTCP, where legacy TCP traffic cannot be immediately removed. Such data centre deployment could enable more inter-data centre use of DCTCP, and gradually migrate out to end-customers, perhaps starting with corporates.

#### Step 4: Further improvements

Once the latency benefits of ECN have been realised through initial deployments, there will be plenty of follow-up work to improve performance further and to build infrastructure to support production operations.

- § 3.7.2 lists some additional performance improvements that would become possible in TCP beyond the the initial ‘safety-first’ modifications (‘TCP Prague’).
- Beyond support for L4S in the main bottlenecks, support would also need to be added in secondary bottlenecks (e.g. the buffer into the upstream link from residential gateways and/or from mobile devices). Also, in some scenarios (e.g. data centres or layer 2 access networks such as PONs) the DualQ Coupled AQM might need to be added in lower layers (§ 3.3 outlines RITE’s preparatory work for this eventuality).
- Some form of policing of the long term envelope of L4S is likely to be needed in the future, as pointed out in the Recommendations on AQM that RITE co-authored (see § 2.1 and RFC 7567 in the associated appendix). This includes methods for mitigating the impact of non-conformant and malicious flows.
- There will also be a need for tools to enabling AQM (and ECN) deployment and to measure the performance (a need also identified in RFC 7567).

### 3.5 Alternative Backoff with ECN

#### 3.5.1 Problem with current AQM mechanisms

The problem we face today is that modern AQM schemes can interact badly with the traditional TCP response to congestion notification. A common rule-of-thumb is to allocate buffering at least equivalent to a path’s intrinsic “bandwidth delay product” (BDP) enabling TCP to achieve close to 100% path utilization. Yet the design goal of AQM schemes, such as Controlled Delay (CoDel) and Proportional-Integral controller Enhanced (PIE), is to effectively instantiate a shallow bottleneck buffer with burst tolerance. So TCP performance suffers once a path’s BDP exceeds the bottleneck AQM scheme’s effective buffer size, whether congestion is signalled by packet loss or ECN.

#### 3.5.2 Motivation behind ABE

While packet loss can be due to full buffers or the channel conditions, an ECN mark is an explicit indication of the existence of an AQM on the path most likely to be the state-of-the-art AQM mechanisms such a CoDel or PIE instantiating 5 ms to 20 ms shallow buffers by default. Therefore an ECN mark could induce a less conservative response (in terms of multiplicative decrease factor) compared to packet loss, in order to achieve 100% path utilization.

#### 3.5.3 Solution: Alternative Backoff with ECN (ABE)

Step 2 of the ECN deployment roadmap outlined in § 3.4 introduces the Alternative Backoff with ECN (ABE) mechanism as a simple and easy-to-deploy update to TCP that may accelerate ECN deployment before more sophisticated and complex mechanisms can be deployed. ABE consists of a parameter change to TCP’s response to ECN Congestion Experienced (CE) marks ( $\beta_{ecn}$ , which is the multiplicative decrease factor used in response to an ECN mark) while keeping the response to packet loss intact. ABE sets  $\beta_{ecn}$  to derived recommended values from the evaluations of [36] (see Appendix B.5) which are normally in the range of 0.7 to 0.85.



### 3.5.4 Evaluation of ABE

ABE has been evaluated under a variety of scenarios using real-life tests and simulations by RITE in collaboration with an external partner. The results of our evaluations are recorded in [36] (see Appendix B.5). Using a mix of experiments, theory and simulations with standard NewReno and CUBIC flows, our evaluations show significant performance gains in lightly-multiplexed scenarios (common on the access links), without losing the delay-reduction benefits of deploying CoDel or PIE. ABE is a sender-side-only modification that can be deployed incrementally (requiring no flag-day) and offers a compelling reason to deploy and enable ECN across the Internet. ABE has also been offered for standardization in the IETF as [37] (see Appendix B.4) and been presented to the IETF TCPM working group. Based on the recommendation from TCPM WG chairs, RITE partners plan to pursue ABE's standardization activities through the ICCRG.

The detailed description and evaluation of ABE mechanism concerns only end-system changes, so it is presented in Section 6.1 and Appendix E of D1.3, although it is also included in Appendix B.5 for convenience. It would not be appropriate to discuss the ABE evaluation results in this deliverable, since ABE mechanism is a sender-side only change and does not constitute any change to the network.

## 3.6 Coupled AQMs for mixed Congestion Controls

This section presents a Dual Queue Coupled AQM that both exploits the results of the discussions on Immediate ECN presented in previous deliverables d2.1 and D2.2 and the initial ideas on Coupling AQMs as described in D2.2. We selected the most promising Coupled AQM scheme supporting coexistence between Classic and Low Latency congestion controllers as it was also the most simple to implement. We also took advantage of the fact that Data Centre TCP was available, implemented on different platforms and had already quite some properties as we had foreseen for a low latency congestion controller. As a result, by only providing network support for it, we could arrange a deployable scenario for a low latency service in a controlled environment such as for instance from the Data Centre to the Home. This work is described in a paper under submission, titled "Data Centre to the Home: Ultra-Low Latency for All" [42]. Also an IETF draft (draft-briscoe-aqm-dualq-coupled-00 [43]) was submitted to the AQM working group to define the DualQ Coupled AQM as a standardized AQM. For full detail of this work we refer to the paper and IETF draft in appendix (see Appendix B.7 and Appendix B.8). This section will give a top-level overview of its content.

### 3.6.1 The Low Loss, Low Latency and throughput Scalable (L4S) service

The DualQ Coupled AQM uses two queues for two services: "Classic" and "Low-Latency, Low-Loss and Scalable" (L4S), denoted resp. by subscripts 'C' and 'L'. The 'Classic' service is intended for all the behaviours that currently co-exist with TCP Reno (TCP Cubic, Compound, SCTP, etc). The 'L4S' service is intended for DCTCP traffic but it is also more general, it will allow a set of congestion controls similar to DCTCP (e.g. Relentless) to evolve.

In our tests, we use DCTCP unmodified, despite its known deficiencies (listed as further work in § 3.7). Our focus is on getting the network service in place. Then, without any management intervention, applications can exploit it by migrating to scalable controls like DCTCP, which can then evolve while their benefits are being enjoyed by everyone on the Internet (See Step 2 of the ECN roadmap presented in § 3.4).

### 3.6.2 Dual Queue for Low Latency

To achieve low latency for L4S traffic in the presence of Classic traffic, we defined a Coupled AQM across two queues as shown in Figure 3.4.

When multiple queues are used, a mechanism for classification and scheduling must be defined. A very simple classification is based on ECN capability. If the ECN field in the IP header is not cleared, we classify the packet into the L4S (L) queue, otherwise into Classic (C). More sophisticated classification might be necessary if also classic traffic is using ECN (see § 3.4). To minimise latency for L4S traffic, we schedule the L4S queue with strict priority, and the Classic queue only when the L4S queue is empty.

To control the prioritized L4S traffic, the ECN signal from the L4S queue is coupled to the queuing time of the Classic

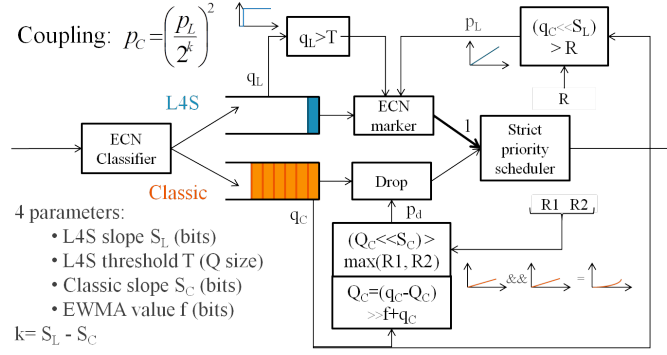


Figure 3.4: Dual Queue Coupled AQM

queue. The coupled feedback ensures that flows share capacity correctly across the two queues and that the L4S queue size remains enough empty to provide scheduling opportunities for keep the Classic flows. Whenever there are packets in the Classic queue, the coupled ECN feedback that the L4S queue emits already depends on its own utilisation (via the Classic queue). However, the L4S queue needs to be able to emit ECN signals if L4S load causes the L4S queue itself to grow, particularly if there is no Classic traffic to generate any coupled feedback. For now, in addition to any coupled feedback, the L4S queue applies a shallow step function without any smoothing delay, as used for DCTCP in data centres. Further work is needed to explore potentially more optimal L4S-only AQM strategies.

As explained in D2.2 the Classic traffic needs a squared drop probability to be throughput aligned with the L4S traffic. When there is traffic in both queues, (2) gives the applied coupling between the drop and marking probabilities. For implementation efficiency, we approximate the denominator by an integer power of 2.

$$p_C = \left(\frac{p_L}{2^k}\right)^2 \quad (2)$$

The resulting coupled AQM needs just 2 parameters for each queue. For the Classic queue:

- $S_C$ : To convert the current Classic queue sojourn time into a dropping probability in the range  $[0,1)$  requires a scaling parameter. To make multiplication efficient, we use an integer power of two, so we define the slope of the AQM's square curve, relating Classic queuing time to drop probability from the Classic queue as  $2^{S_C}$ ;
- $f$ : To support smoothing the sojourn time of the Classic queue and make multiplication efficient, we will use an integer power of two for the EWMA constant, which we define as  $2^{-f}$ .

For the L4S queue:

- $S_L$ : As for the Classic queue, we define the slope of the AQM's marking function as  $2^{S_L}$ ;
- $T$ : The queue size at which step threshold marking starts in the L4S queue.

For the classic drop based queue the sojourn time is averaged (denoted by the capital  $Q_C$ ), the L4S traffic gets immediate feedback without averaging or delays.

The queuing time of the Classic queue drives the marking probability of L4S packets and dropping probability of Classic packets with a linear intensity based on the scaling parameters  $S_L$  and  $S_C$ . For either case,  $2^{-S_L}$  or  $2^{-S_C}$  represent the queuing time in the Classic queue at which marking or dropping probability reaches 100%. The difference in marking intensity  $k = S_L - S_C$  represents a coupling rate weight factor between the two queues, but it is not an additional parameter. Some value has to be chosen for  $k$  in (2) to relate the rates of L4S and Classic flows, given the possible discrepancy between their RTTs. This is a policy decision for the network operator. In our experiments, we adopted the throughput equality policy to allow an easy comparison of the results. However, there is no implication that this policy is recommended. Taking all factors into account,  $k = 0$  might be a good recommendation for the general Internet. It will promote the migration to L4S for all traffic.



The pseudocode in Algorithm 1 summarises the implementation for scheduling and dequeuing a packet. The Linux implementation of our AQM, L4S and Classic together consume fewer instructions per packet than even the simplest form of RED.

---

**Algorithm 1** Dequeue for Dual Queue Coupled AQM

---

```

1: if LQ.DEQUEUE(pkt) then
2:   if (LQ.LEN() > T)  $\vee$  ((CQ.TIME() << SL) > RND()) then
3:     MARK(pkt)
4:   end if
5:   RETURN(pkt) ▷ return the packet and stop here
6: end if
7: while CQ.DEQUEUE(pkt) do
8:   QC += (PKT.TIME() - QC) >> fC ▷ C EWMA
9:   if (QC << SC) > MAX(RND(), RND()) then
10:    DROP(pkt) ▷ Squared drop, redo loop
11:  else
12:    RETURN(pkt) ▷ return the packet and stop here
13:  end if
14: end while

```

---

### 3.6.3 Evaluation

We have evaluated the proposed DualQ AQM mechanism in ALU video testbed that provides a realistic setting, and that allows to run repeatable experiments (controlled environment). Figure 3.5 depicts the testbed, which consists of a classical residential service delivery network. The client computers in the home network and the application servers at the global ISP are Linux machines, which can be configured to use any TCP variant and start applications and test traffic. The two client-server pairs (A and B) are respectively configured with the same TCP variants and applications. Instead of in the BNG internally, a Linux server (AQM server) is used to create the per customer bottleneck and to implement the DualQ Coupled AQM before that bottleneck.

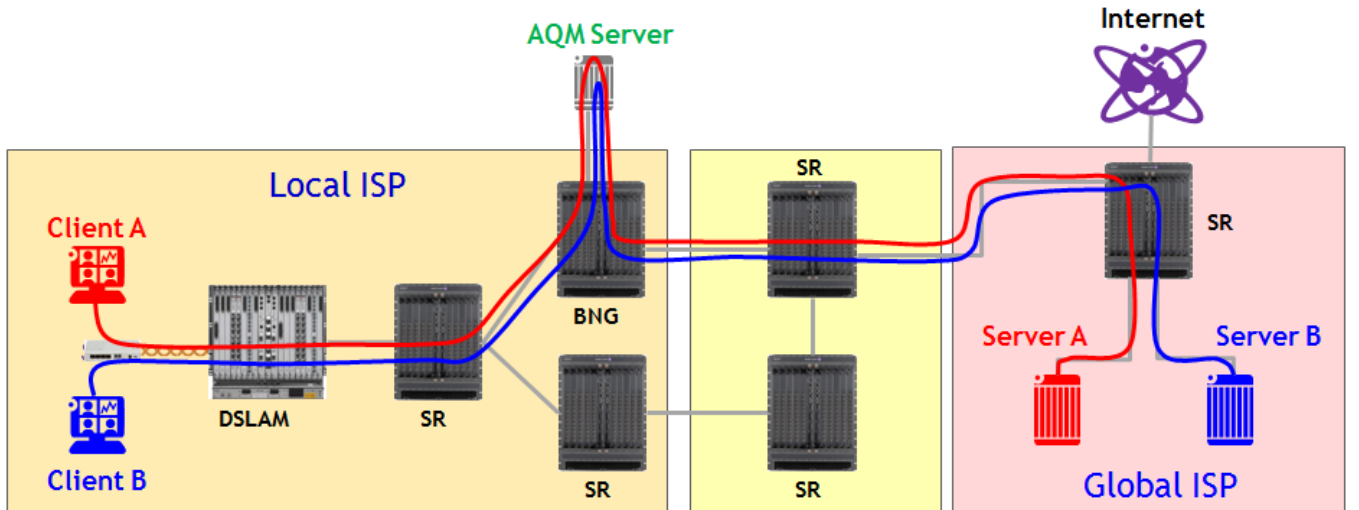


Figure 3.5: Testbed configuration

To assess our primary objective (long term throughput equivalence), we performed experiments on different AQMs with long-running flows. We used steady flows not as an example of a realistic Internet traffic mix, rather as a situation where starvation can typically occur. We also evaluated the AQMs under dynamic load to verify the impact of low latency and loss on completion time of short flows. The results are plotted in Figures 3.6 for TCP Cubic and DCTCP

flows. On each client 0 to 10 file downloads are started from its matching server, resulting in 120 combinations. Each combination ran for 250s and the throughput for every flow (average per second) is plotted in the first matrix, and the queuing delay CDF (based on sojourn time for every packet) in the second matrix. The row and column labels indicate the TCP variant (C:Cubic, D:DCTCP) followed by the number of active flows on the respective client-server pairs.

Our DualQ Coupled AQM is able to guarantee equal throughput between DCTCP and Cubic flows. It deviates slightly due to the Classic Q size, which grows on higher load, resulting in less throughput for the Classic flows, and is smaller at lower load, resulting in a higher throughput for the Classic flows. DCTCP clearly has a smaller throughput variance than Cubic flows. Also the queue variation is very high for Cubic, compared to the hardly noticeable step function for DCTCP.

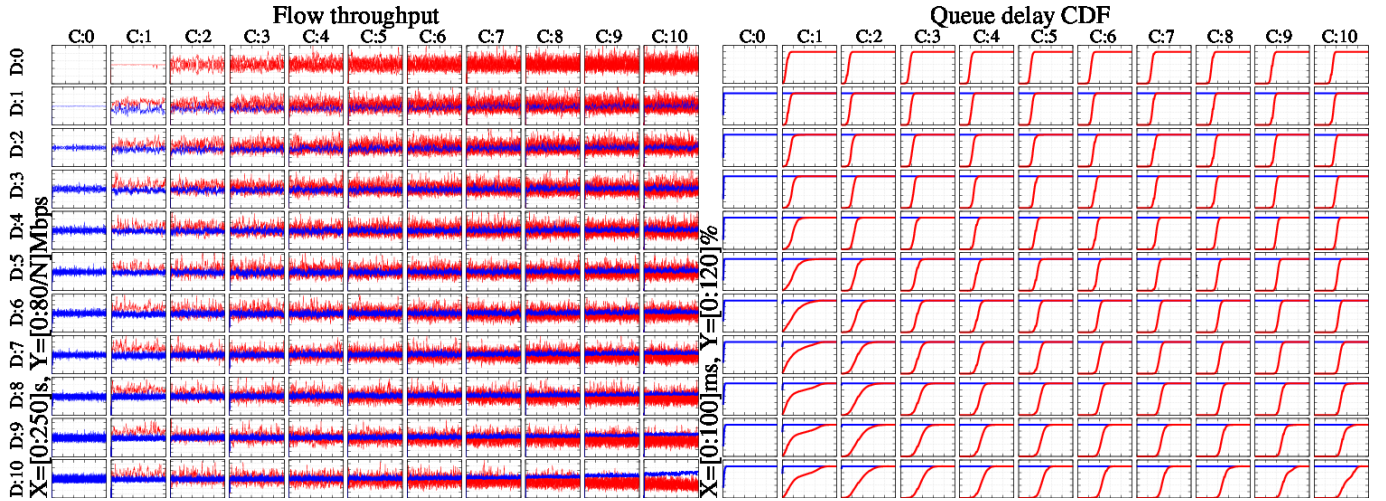


Figure 3.6: Per flow throughput and Queue Delay CDF on the DualQ Coupled AQM for long flows  
D: Number of DCTCP flows (blue), C: Number of Cubic flows (red), N: the number of flows.

To assess the dynamic behaviour, 25 load combinations were tested on the client-server pairs. Figures 3.7 show the results for these dynamic experiment. The row and column labels (X0+L; X0+H, X1; X1+L; X1+H) indicate TCP variant (X=C,D), the number of long-running flows (0 to 1), and the dynamic load level which is an exponential arrival process with 10 (L=low load) or 100 (H=high load) average arrivals per second (or none), requesting a Pareto distributed download size  $\alpha = 0,9$  with a minimum size of 1KB.

In the completion time results for Cubic, we see 2 levels of timeouts: at 1s (due to lost SYN) and 300ms (due to lost FIN). The current DCTCP implementation uses ECN for all packets, avoiding these timeouts.

Comparing the Cubic completion times with the DCTCP completion times, we can again conclude that our DualQ AQM can maintain the stunningly low queuing delays for L4S traffic, without sacrificing the delay for the Classic flows. Parallel long running DCTCP flows can achieve a reasonable but not optimal link utilisation because it reduces the throughput more appropriately on congestion signals, but can only regain the available capacity incrementally when a short flow ends. One issue with DCTCP also becomes apparent for flows bigger than the initial windows size of 10. As marking probability is much higher, slow start will get consistently prematurely interrupted. A good result is that no slow start overshoot is detected (zero Q delay), but it leads to unnecessarily longer completion times for the medium sized flows. The outcome suggests that a gradual slow start exit scheme is possible. These observations were taken into account for defining an optimal L4S congestion controller, listed in § 3.7.

We also compared the results of our AQM with other reference scenarios, including other state of the art AQMs available in Linux. See the full paper attached in the appendix. No other mechanism could achieve comparable completion times with such low self-inflicted delay, even not with per flow queuing schedulers (FQ-X) that identify flows in the network.

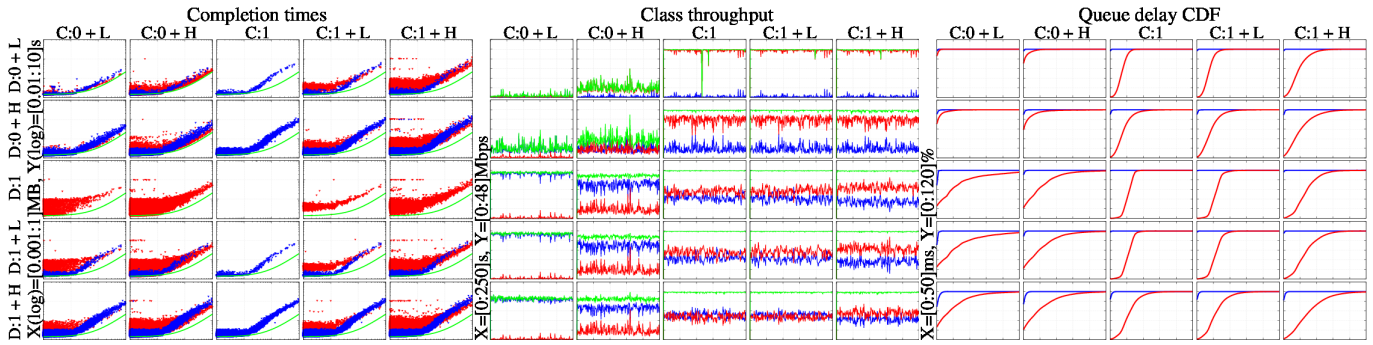


Figure 3.7: Completion times, Class throughput and Queue delay CDF on the DualQ Coupled AQM for dynamic load D: Number of DCTCP flows (blue), C: Number of Cubic flows (red), L: 10 exponential arrivals per second, H: 100 exponential arrivals per second.

Green = theoretical minimal completion time or total throughput, showing utilisation.

### 3.7 TCP Prague: Low Loss, Low Latency, Scalable *and* Safe

The TCP-PRIME work in deliverable 2.2 described mechanisms to improve congestion control optimized for low latency. During the validation of our DualQ Coupled AQM, we used DCTCP for the L4S traffic, as it was available and had the right properties based on the analysis that we did also in the context of the Coupled AQM work. Especially its throughput scalable marking property (under non-on/off marking) and its support for immediate network marking inspired us to use DCTCP. We knew that DCTCP was not designed for coexistence with Internet traffic, and there were significant deficiencies or at least non-optimality with respect to our low latency objectives, but using a TCP implementation with mainstream availability, supported on several platforms (including a commercial one—Microsoft Windows Server) would help with the acceptance of our work. Testing and evaluation with DCTCP worked surprisingly well, but it confirmed our concerns, and we were able to identify additional issues that would need improvement in our target L4S congestion controller.

This section describes the list of possible improvements to DCTCP. A first set is required for compatibility and safe Internet deployment as it may harm other traffic. A second set of improvements can be considered for enhancing the performance of DCTCP when deployed in the bigger Internet context. These improvements can be done in parallel and become active when the network starts to support the Coupled AQM concept.

We proposed a draft of this list at an ad hoc meeting of about 30 people interested in developing DCTCP, including developers of all the major operating systems. The meeting was held in July 2015 at the Prague IETF. The participants developed the list into the form below and decided to call the new initiative ‘TCP Prague’ [44].

#### 3.7.1 Improvements to prevent harm to other traffic

**3.7.1.1 Fall back to Reno or Cubic behaviour on loss** Currently due to incorrect code in the Linux implementation of DCTCP, no distinction is made between a loss and an ECN signal. On detecting a loss, DCTCP reduces its window by the same small amount as it would with a mark. This makes the congestion signal more frequent than with Classic TCP, so the retransmission load is much higher, resulting in significantly more aggressive behaviour. Without the bug, Linux DCTCP was intended to respond to loss as Classic TCP. Such a less aggressive and loss compatible response allows a gradual upgrade of congested bottlenecks. Whether the other platforms (Windows, FreeBSD) fall back correctly will need verification in our testbed.

**3.7.1.2 Negotiate Accurate ECN feedback semantics** For an L4S sender, a pre-requisite is accurate ECN feedback, which feeds back the extent of marking not just the existence of at least one mark per RTT, which is all classic ECN uses. Given these altered feedback semantics are a change to the TCP wire protocol, they need to be negotiated. Such an effort is well-advanced in the IETF in collaboration between RITE partners and others (see § 4.4

and the IETF documents in its associated appendices [45, 46]).

**3.7.1.3 Standardised packet identifier** In a situation where Classic ECN and L4S ECN capable flows are passing a network bottleneck, the AQM needs to be able to identify whether Classic or L4S classification, marking and scheduling is required. For this a packet identifier will need to be standardised.

Currently two proposals are under discussion in IETF. The first proposal would let L4S senders use the ECT(1) codepoint as an initial mark. The disadvantage of this is that the CE codepoint would be shared, and once a packet is marked it cannot be classified anymore. Classifying all marked packets as L4S would be a possible compromise. Classic Traffic has very little marked packets, and would benefit from the faster prioritised congestion notification, but potential out of order delivery of the marked packets might cause unwanted side effects.

A second proposal is to use a newly defined Diffserv codepoint for L4S. L4S could start off in controlled environments only, where the Diffserv codepoint and appropriate L4S handling could be supported wherever required. A network operator could use L4S for its premium services, or support L4S traffic to selected partners. A disadvantage of this approach is that the default behaviour for best effort networks is to bleach DSCPs. Therefore, the receiver can identify if L4S service is supported end-to-end, but in inter-domain scenarios, the service would only work in the first network on the path. Later, if classic ECN all migrates to L4S, the ECN capability alone could become a sufficient identifier. Unlike Diffserv, ECN is defined to propagate unchanged between networks.

**3.7.1.4 Handle a window of less than two segments** One of the observations when evaluating the DualQ Coupled AQM was that the minimum window of 2 packets causes a standing queue when the RTT is low. If every flow has at least 2 packets in flight, and the bandwidth delay product (in units of packets) is smaller than twice the number of flows, the excess packets will end up in the bottleneck queue. This is a problem for the L4S queue in the DualQ Coupled AQM, as it will never get empty, even when marking 100%, the Classic traffic will starve. A solution for the starvation of Classic traffic is possible in the AQM itself (for instance scheduling based on a biased maximum sojourn time instead of with strict priority—see Appendix B.8), but this will not prevent pollution of the ultra low latency queue. Therefore DCTCP and TCP Prague senders (at least) should support pacing for sub minimum segment size windows, to prevent excessive standing queues in the network.

**3.7.1.5 Heuristic testing for classic ECN bottlenecks** If, despite every measure taken to prevent incorrect marking, L4S traffic is handled by a Classic AQM, senders could fall back to Classic congestion control, if this situation can be detected. Alternatively an optimistic deployment strategy can be taken assuming that the end-points need to detect if the current bottleneck is using Classic ECN. The idea would be to detect a Classic bottleneck by whether appreciable delay growth accompanies the marking.

## 3.7.2 Improvements optimising latency

**3.7.2.1 Average ECN feedback over its own RTT** Current DCTCP implementations apply exponential moving window averaging with a fixed value. The goal should be that ECN feedback is averaged over its own RTT, not the hard-coded RTT suitable only for data-centres. Another possible solution is to reduce the window by half a segment size per ECN Echo, similar like Relentless TCP [47]. This might be needed to support larger RTTs which are common on the Internet.

**3.7.2.2 Proportional slow start** Currently DCTCP exits slow start as Classic TCP would, when receiving a single congestion signal. Another observations when evaluating the DualQ Coupled AQM was that DCTCP flows due to more frequent congestion feedback (in the order of several per RTT) have a higher probability to exit slow start too soon, unnecessarily increasing completion time. As a result DCTCP flows underperform for medium sized data transfers. Also during slow start a response proportional to congestion should be applied, resulting in a gradual exit from slow start.

**3.7.2.3 Faster than additive increase** DCTCP is reacting proportionally to multiple signals per RTT to congestion. This is important to avoid buffer build-up in the network. If for multiple RTTs no congestion signal is received, it might suggest that a faster increase is appropriate. This is important to also preserve a good utilisation if link capacity becomes available again and shallow queues are used.

**3.7.2.4 RTT independent throughput** From our TCP-PRIme work we coined the idea to have a TCP congestion controller which send with a rate only dependent on the marking probability and not dependent on the RTT. Large queues in the network have also advantages. Next to the obvious link utilisation advantage, its RTT fairness equalisation role is less appreciated. Large queues help all flows to experience a similar RTT. If the queue is dimensioned according to the largest delay to support (RM), the maximum RTT unfairness is only 2. A flow with a theoretical base RTT of 0 will experience a RTT of RM, and a flow with a base RTT being RM will experience an RTT of 2\*RM. When queues in the network are very small, they lose their equalisation role, and an alternative solution is needed to take over this role, preferably in the endpoints. In our TCP-PRIme work we proposed measuring marks per fixed time interval to determine the sending rate or window, or to measure the RTT to compensate the rate or window, aligning it with a reference RTT.

**3.7.2.5 Pacing** As defined above, pacing small windows is already essential to preserve a low queuing delay. Also application limited flows can cause large bursts if a large window is available after some idle time. Pacing will be required if we want to avoid such bursts for our L4S traffic, especially for flows with larger base RTTs, requiring larger windows.



## 4 Getting up to Speed (GUTS) Fast

Internet transport protocols have to be designed on the basis that each time a flow starts the available capacity is unknown. Even a re-start after idling cannot assume the capacity or other traffic has remained unchanged. So senders universally use some variant of the TCP slow-start algorithm, which sends an initial handful of packets, waits for feedback, then doubles how much it sends in each subsequent round as long as it has sensed no losses in the feedback. Over the years, every doubling of bottleneck capacity has added another round trip of delay before slow-start can exploit it.

Slow-start, with various optimizations, has been considered as an acceptable compromise between acceleration and overshoot. However, slow-start does not scale. Over the years, every doubling of bottleneck capacity has added another round trip of delay before slow-start can exploit it. This would not be a problem if flow sizes in the Internet would double as fast as link capacity doubles. With larger link capacity, larger transfers become feasible, but it does not follow that all the demand for the smaller flows disappears. Therefore an ever-growing proportion of a typical user's sessions become limited by slow-start, not capacity. The longer we fail to solve this flow-start problem, investing in capacity will make less and less difference to more and more people.

### 4.1 What Use is Top Speed Without Acceleration?

The aim of this activity to show that getting up to the speed is becoming a major component of latency as the network links get faster. This activity started with quantifying the latency penalty from the start-up phase of typical Internet transfer through evaluating a model of the start-of-the-art slow-start dynamics and characterising the typical flow lengths. Both Deliverable 1.1 and 2.2 discuss the slow-start model where we modeled the average rate of a single TCP flow for different bottleneck capacities in order to show how the slow-start problem influences the performance. In our analysis, for example, we show that unless a flow becomes larger than 1 MB, it hardly even starts to exploit a dedicated 80Mb/s link. The problem becomes much worse if the round trip time is longer. For example, if the round trip time increases ten times, only flows 10 times larger than the 1 MB one can start to make use of 80Mb/s capacity.

In order to substantiate the slow-start model we also carried out a longitudinal study of the typical flow lengths in the Internet using a decade of traces collected by Cooperative Association for Internet Data Analysis (CAIDA) [48]. Our study suggests that the distribution of flow sizes is not changing much as capacity grows. The majority of flows is short in size and has hardly changed over time.

The work since deliverable 2.2 has further analysed the CAIDA traces to identify parallel flows in access links. The major outcome of the analysis is that most of the time the access link is empty when a new flow arrives which suggest the potentiality of a better flow start-up scheme to utilise the idle capacity. One of the limitation of CAIDA traces is that these measurements are taken at the peering points, and therefore do not see CDN traffic within ISPs. We also have supplemented the trace analysis with newer traces collected at the Norwegian University of Adger. The result based on analysing the university traffic is consistent with CAIDA's.

Furthermore, in this activity, a scaling model has been produced, supported by the analysis of CAIDA traces to show in which dimensions traffic has been scaling over the last decade. According to the model, the capacity increase of network links can be used to absorb larger flows or more simultaneous flows with an expectation that increases in capacity should make individual flows faster. Based on our longitudinal analysis of CAIDA traces, as there is no clear change of either the flow size distribution or number of parallel flows distribution, an additional speed-up of Internet flows can only be achieved using better flow start-up mechanism to utilise the increasing capacity investment. The whole analyses has been drafted in a paper titled "What Use is Top Speed without Acceleration?" (see Appendix C.1 ). In this paper we have also shown that the impact of the flow start-up limitation largely depends on the value-judgements for different flow sizes as well as quality requirements to users.

### 4.2 Existing GUTS solutions

Very briefly in Deliverable 1.1 and in detail in Deliverable 2.2, we surveyed existing attempts to solve the flow start-up limitation. Deliverable 2.2, using scaling analysis, further showed why existing unilateral proposals of tuning the start-up mechanisms (e.g. by opening multiple flows or using larger initial window) would not scale indefinitely. For

example, to keep up with the capacity increase, the number of parallel flows would have to grow more than exponentially over time—the number of flows has to scale faster than capacity growth, which itself increases exponentially over the years. Besides, using a large number of flows introduces other problems like flow-state memory exhaustion in servers and NATs, as well as the multiple initial windows overflowing buffers on slower links. The deliverable also explained that those solutions (e.g. RCP [49], XCP [50]) that could scale are inherently not incrementally deployable and therefore not feasible for the Internet. For example, RCP and XCP require all traffic to use the new scheme and require a new packet header, which are not feasible to deploy in the general Internet. Please see Appendix C.1 for the whole analysis of existing attempts to solve the getting up to the speed problem. According to our analysis, there appears to be no feasible solution. This implies that if we are at the limit of host-only solutions, changing the network interface is an industry co-ordination problem of epic proportions.

### 4.3 Up to Speed with Queue View (QV)

The central idea of Queue View is for buffers on the path to continuously encode the length of their queue by setting markers in the explicit congestion notification (ECN) field in packet headers. Figure 4.1 shows packets arriving at a queue with ECT(0) set in the 2-bit ECN field of the IP header, represented by the tag ‘0’. ECT(0) is the name of the ‘10’ codepoint that a sender with an ECN-capable transport (ECT) must set by default according to RFC3168 [38]. In the remainder of this work, we will use ‘0’ to represent ECT(0) and ‘1’ to represent ECT(1), which is the term for the ‘01’ codepoint, currently set aside for experimental use. The ‘0’ & ‘1’ terminology also generalises the discussion to include any similar pair of codes in protocols other than IP.

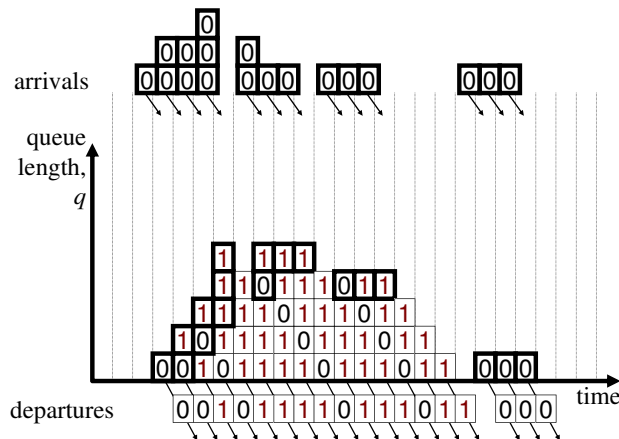


Figure 4.1: QV: the idea. Only one zero at a time in the queue makes the spacing between zeroes into a measure of queue length

Solely for illustration purposes, the figure divides time into slots along the horizontal axis and one equal sized packet leaves the buffer in each slot. Then, whenever more than one packet joins the queue within a timeslot (shown with thicker borders), the queue grows. Actually the approach does not use time-slots and works just as well with variable sized packets.

#### 4.3.1 QV: Status

QV was presented in the internal deliverable D2.2. Due to pressure of other RITE activities this work was reduced in priority and no further progress is planned before the end of the project. The work on QV so far is included in Appendix C.1 as a public record. This appendix ends with a list of further work needed, and the status of each item.

## 4.4 Generic Accurate ECN Feedback in TCP

The introduction of the GUTS section (Section 4) showed how scaling the performance of the Internet will become increasingly limited by the ability to get up to speed fast. And it was further shown that all attempts so far to work round the problem will not be able to scale (Section 4.2). The problem is the lack of visibility of available capacity when a data flow first starts, because available capacity can change within milliseconds as other flows come and go. This is essentially a problem of lack of information at the interface between the host and the network—that is the interface between the network layer and the end-to-end transport layer.

In Section 4.3, Queue View (QV) was introduced as a possible solution to this problem. By using one of the last available codepoints in the IP header, QV reveals more accurate and instantaneous information to the data receiver about available bottleneck capacity. In order for the sender to be able to use this additional information, the end-to-end protocol for feeding back such information from the receiver to the sender has to be upgraded. Many of the more modern transport protocols already provide accurate ECN feedback information (e.g. DCCP, SCTP, RTP-UDP). But the most widely used transport, TCP, does not, and the alternative transport protocols have proved nearly impossible to deploy through middleboxes.

Because of the way ECN was originally specified for TCP, only one feedback signal can be transmitted per Round-Trip Time (RTT). This is sufficient for pre-existing TCP congestion control mechanisms that perform only one reduction in sending rate per RTT, independent of the number of ECN congestion marks. However, QV is one of a number of recently proposed or deployed mechanisms that need more accurate ECN feedback than ‘classic’ ECN [38] to work correctly. The other mechanisms include two initiatives on the IETF’s standardisation agenda: Data Centre TCP (DCTCP [51]) and Congestion Exposure (ConEx [52]). This is fortunate, because there is a very high bar to changing TCP, given it is critical to the functioning of the Internet. So having multiple players all wanting change, increases the pressure for the IETF to work on it.

### 4.4.1 Accurate TCP-ECN Feedback: Status

At the very start of the RITE project (October 2012), we started the process of persuading the IETF to modify TCP to include more accurate ECN feedback. This involved persuading the IETF’s TCP maintenance and Minor Modifications (tcpm) WG to charter a work-item to specify the requirements for such a change. Ten draft revisions later this has been approved and published as RFC 7560 [45], which is also included in Appendix C.3. In parallel, we have been developing technical proposals on how to meet the requirements. The IETF would not charter standardisation of such work until the requirements were approved. However, now the requirements RFC is approved, we expect the current negotiations with the working group chairs to succeed in chartering this work item.

In D2.2 we included and described version 03 of our technical proposal to add more accurate ECN feedback to TCP. It used a number of ingenious coding techniques to squeeze 3 counters into a 3-bit field and it overloaded the rarely used 16-bit Urgent Pointer with optional supplementary information adding more significant bits to the 3-bit field plus information about the order of arriving codepoints. It satisfied all the measurable requirements (as they stood at that time), which was an impressive feat, because it was generally believed that tradeoffs would be needed because all the requirements taken together were very challenging. However, we received reactions like “Too complex”, “If I can’t get the main idea in one slide, it’s not going to fly,” or “I have tried, but I cannot understand it”.

Having ‘gone back to the drawing board’ we have made a different set of tradeoffs for version 04 [46], which has been submitted to the IETF (it is included in Appendix C.4. Table 4.1 summarises how it scores against each requirement (the column headed AccECN-04). As with version 03, the scoring is given with and without the supplementary AccECN TCP Option, which be stripped by an incorrectly designed middlebox as it traverses some Internet paths. The table also shows how alternative approaches score for comparison, including our previous attempt (headed AccECN-03). It will be seen that we believe that the new version (04) is much simpler, but it consumes more option space (‘overhead’), while on all other dimensions it scores equally to version 03.

The new version (04) of the Accurate ECN protocol is now in the process of being implemented in the Linux kernel (by Mirja K’uhlewind) outside the RITE project). The long process that was needed to sense the IETF’s preferred tradeoff between requirements has meant that evaluation of the final design might not be possible before the end of the RITE project. Nonetheless, the wait has been worthwhile, because we now believe we have found the right set of tradeoffs to satisfy the IETF and Linux communities—a thoroughly evaluated but unwanted solution costs more and



Requirement	Classic ECN	ECN Nonce	DCTCP	AccECN-03			AccECN-04	
				Urg-Ptr	TCP opt	essential	TCP opt	essential
Resilience	+	+	-	+	+	o	+	o
Timeliness	o	o	-	+	+	+	+	+
Integrity	-	o	+*	+*	+*	+*	+*	+*
Accuracy	-	-	-	+	+	+	+	+
Ordering	-	-	+	+	+	-	+	-
Simplicity	++	+	o	-	-	o	+	+
Overhead	++	o	o	+	o	++	-	++
Compatibility	o	o	-	o	-	o	-	o

Table 4.1: AccECN Protocol Features and Comparison with Alternatives

Scored from very positive ++ through neutral o to very negative - -

\* Compatible with an independent zero-overhead integrity solution [53]

is worth less than a solution that has the potential for considerable impact.

At the IETF in Prague in July 2015, we convened an ad hoc ‘Birds of a Feather’ meeting of those interested in implementing the necessary modifications to DCTCP to make it safe to use on the public Internet. It exploited the excitement that the RITE demonstration of ultra-low latency queuing using DCTCP over the public Internet (‘Data Centre to the Home’, see subsection 3.6). The meeting had two goals: i) to agree a list of necessary but sufficient of modifications; and ii) to identify those interested and create the necessary community coordination mechanisms (mailing list, wiki etc). Although the meeting was convened in less than 24 hours, it attracted over 30 people, including the TCP and/or DCTCP implementers from all the major operating system developers (Windows, Linux/Google Android, FreeBSD/Apple iOS). One delegate (from Google) remarked that all the designers and implementers who control the behaviour of nearly 100% of the Internet’s traffic were in the room. The minutes of the meeting are available online [44].

A robust replacement for the Accurate ECN Feedback in DCTCP was high on the agreed list. It was agreed to use the IETF processes to coordinate the design and implementation of this new evolutionary branch of TCP, to be called ‘TCP Prague’. Therefore, once the AccECN design has stabilised, we can expect multiple independent implementations, and the meeting discussed how they will be evaluated — primarily by organising interop sessions.

#### 4.4.2 Accurate TCP-ECN Feedback: High Level Overview

Figure 4.2 gives a high level overview of the new AccECN protocol design. As with the previous version, it is divided into an essential part (the 3-bits coloured green) and a dispensible supplementary part (the AccECN TOP Option coloured orange). The 3 bits are used during TCP’s initial handshake to negotiate use of the AccECN capability or otherwise to fall-back to the most advanced form of ECN that both ends support.

Once negotiation has been successful, these three bits are overloaded as a 3-bit field (‘ACE’ in the inset of the figure) that repeatedly communicates the 3 least significant bits of the Data Receiver’s count of how many ECN marks it has seen. Obviously, a 3 bit counter will wrap frequently, and loss of ACKs might mean the Data Sender misses a whole cycle of the field (see the draft in Appendix C.4 for how cycling of the field is sensed and other details of the necessary safety measures).

The supplementary AccECN Option provides sufficient space for up to three 24-bit counters (one for each ECN codepoint). This is the cause of the extra overhead, but it is sufficient for a more dependable feedback channel — it would be highly unlikely that a whole cycle of a counter could go unnoticed due to a long string of losses. The extra space is also sufficient for feedback in bytes, rather than packets. Again, full details are in the draft (Appendix C.4).



## 5 Conclusions

This deliverable has summarized the reduction in Internet latency from modifications to the interface (implicit or explicit) between networks elements and end-systems. The status of each prototype development, evaluation or production of a standard or a guideline has been outlined. Table 5.1 summarises all the activities, showing whether they involved development of a prototype, a re-parameterization of existing prototype code, or development of new prototype tools. It shows which activities involved evaluation (whether evaluation of our own prototype or of the code of others), whether the outcome is being pursued for standardisation and/or whether it meets our intention to issue guidance to the industry on how to reduce latency.

	§	Prototype Development	Prototype parameterization	Prototype Tool	Evaluation	Standardisation	Guidelines
AQM Recommendations	§ 2.1					✓	✓
AQM Characterisation Guidelines	§ 2.2					✓	✓
Comparison of self-tuning AQM algorithms	§ 2.3		✓		✓		
Constraints on AQMs from TCP: Insights from Curvy RED	§ 2.4				✓		✓
Review of PIE for the IETF	§ 2.5				✓	✓	
Impact of scheduling on AQM performance	§ 2.6		✓		✓		
Rural AQMs	§ 2.7		✓		✓		
Interaction between Lower-than Best Effort & AQMs	§ 2.8		✓		✓		
AQM detection active measurement tool	§ 2.9			✓	✓		
MADPIE: Maximum and Average queuing Delay with PIE	§ 2.10	✓			✓		
Discussion on squared AQMs	§ 2.11	✓					
ECN Benefits	§ 3.1						✓
ECN path support evaluation	§ 3.2			✓	✓		
Guidelines for Adding ECN to Protocols that Encapsulate IP	§ 3.3					✓	✓
ECN Roadmap	§ 3.4					✓	
ABE: Alternative Backoff with ECN	§ 3.5	✓			✓	✓	
Coupled AQMs for mixed Congestion Controls	§ 3.6	✓			✓	✓	
TCP Prague: Low Loss, Low Latency, Scalable <i>and</i> Safe	§ 3.7						✓
What Use is Top Speed Without Acceleration?	§ 4.1				✓		
Existing GUTS solutions	§ 4.2				✓		
QV: Up to Speed with Queue View	§ 4.3	✓			✓		
Generic Accurate ECN Feedback in TCP	§ 4.4	✓					✓

Table 5.1: Categorization of the Types of Activity in this Report

Collectively, the work summarized above has significantly contributed to reducing Internet latency and we believe it thoroughly satisfies all the objectives identified for WP2 in the Description of Work for the RITE project. As promised, we have delivered both immediate incremental steps to reduce latency in specific circumstances, and a full system to nearly completely remove queuing latency without losing utilization, with zero loss, zero configuration and management and incremental deployability over the current Internet.

## References

- [1] CACM staff, “BufferBloat: What’s wrong with the Internet?” *Communications of the ACM*, vol. 55, no. 2, Feb. 2012.
- [2] F. Baker and G. Fairhurst, “IETF recommendations regarding active queue management,” Internet Draft, RFC 7567, 2015. [Online]. Available: <http://tools.ietf.org/html/draft-ietf-aqm-recommendation>
- [3] B. Briscoe and J. Manner, “Byte and packet congestion notification,” RFC 7141 (Best Current Practice), Internet Engineering Task Force, Feb. 2014. [Online]. Available: <http://www.ietf.org/rfc/rfc7141.txt>
- [4] N. Kuhn (Ed.), N. Khademi (Ed.), P. Natarajan (Ed.), and D. Ros, “AQM Characterization Guidelines,” Internet Draft draft-kuhn-aqm-eval-guidelines, work in progress, May 2015. [Online]. Available: <http://tools.ietf.org/html/draft-ietf-aqm-eval-guidelines>
- [5] T. Hoeiland-Joergensen, P. McKenney, D. Täht, J. Gettys, and E. Dumazet, “Flowqueue-codel,” Internet Draft draft-hoeiland-joergensen-aqm-fq-codel, work in progress, Jun. 2014. [Online]. Available: <http://tools.ietf.org/html/draft-hoeiland-joergensen-aqm-fq-codel>
- [6] S. Floyd, R. Gummadi, and S. Shenker, “Adaptive RED: An Algorithm for Increasing the Robustness of RED’s Active Queue Management,” ICIR, Technical report, Aug. 2001.
- [7] K. Nichols and V. Jacobson, “Controlling queue delay,” *ACM Queue*, vol. 10, no. 5, May 2012.
- [8] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, “PIE: A lightweight control scheme to address the bufferbloat problem,” in *Proceedings of the IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, 2013, pp. 148–155.
- [9] J. Schwardmann, D. Wagner, and M. Kühlewind, “Evaluation of ARED, CoDel and PIE,” *20th Eunice Open European Summer School and Conference*, 2014.
- [10] N. Khademi, D. Ros, and M. Welzl, “The new AQM kids on the block: An experimental evaluation of CoDel and PIE,” in *17th IEEE Global Internet Symposium (IEEE INFOCOM 2014 Workshop)*, Toronto, April 2014, pp. 85–90.
- [11] E. Grigorescu, C. Kulatunga, and G. Fairhurst, “Evaluation of the impact of packet drops due to aqm over capacity limited paths,” *CSWS*, 2013.
- [12] I. Järvinen and M. Kojo, “Evaluating CoDel, PIE, and HRED AQM Techniques with Load Transients,” *LCN*, 2014.
- [13] R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, “PIE: A lightweight control scheme to address the bufferbloat problem — further studies,” Presentation at the ICCRG meeting, 86th IETF, Orlando, Mar. 2013. [Online]. Available: <http://www.ietf.org/proceedings/86/slides/slides-86-iccr-5.pdf>
- [14] C. Greg White, “Active queue management algorithms for DOCSIS 3.0: A simulation study of CoDel, SFQ-CoDel and PIE in DOCSIS 3.0 networks,” *Cable Television Laboratories*, 2013.
- [15] A. Sivaraman, K. Winstein, S. Subramanian, and H. Balakrishnan, “No silver bullet: Extending SDN to the data plane,” in *HotNets-XII*, 2013.
- [16] N. Kuhn, E. Lochin, and O. Mehani, “Revisiting Old Friends: Is CoDel Really Achieving What RED Cannot? rural,” in *ACM SIGCOMM Workshop CSWS*, 2014.
- [17] R. Pan, P. Natarajan, F. Baker, B. Ver Steeg, M. Prabhu, C. Piglione, V. Subramanian, and G. White, “PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem,” Internet Engineering Task Force, Internet Draft draft-ietf-aqm-pie-01, Mar. 2015, (Work in progress). [Online]. Available: <http://tools.ietf.org/html/draft-ietf-aqm-pie>
- [18] A. B. Bagayoko, G. Fairhurst, N. Khademi, N. Kuhn, C. Kulatunga, and D. Ros, “Operating ranges and tunability of CoDel and PIE,” *Under submission*, 2015.

- [19] S. C. for Rural Communities, “Mind the Gap: Digital England — A Rural Perspective,” June 2014.
- [20] L. Townsend, A. Sathiaselam, G. Fairhurst, and C. Wallace, “Enhanced broadband access as a solution to the social and economic problems of the rural digital divide,” *Local Economy*, 2013.
- [21] Y. Gong, D. Rossi, C. Testa, S. Valenti, and D. Taht, “Fighting the bufferbloat: on the coexistence of aqm and low priority congestion control (extended version),” *Computer Networks*, 2014.
- [22] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, “Low Extra Delay Background Transport (LEDBAT),” RFC Editor, RFC 6817, Dec. 2012.
- [23] D. A. Hayes and G. Armitage, “Revisiting TCP congestion control using delay gradients,” in *Networking 2011*. Springer, 2011.
- [24] G. Armitage and N. Khademi, “Using delay-gradient TCP for multimedia-friendly transport in home networks,” in *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, Oct 2013, pp. 509–515.
- [25] M. Welzl, S. Gjessing, and N. Khademi, “Less-than-best-effort service for community wireless networks: Challenges at three layers,” in *Wireless On-demand Network Systems and Services (WONS)*, 2014.
- [26] N. Kuhn, O. Mehani, A. Sathiaselam, and E. Lochin, “Less-than-best-effort capacity sharing over high BDP networks with ledbat,” in *VTC 2013-Fall, 78th Vehicular Technology Conference*, 2013.
- [27] S. Q. V. Trang, N. Kuhn, E. Lochin, C. Baudoin, E. Dubois, and P. Gelard, “On the existence of optimal LEDBAT parameters,” in *IEEE International Conference on Communications (ICC)*, 2014.
- [28] K. K. Jonassen, “Implementing CAIA Delay-Gradient in Linux,” Master’s thesis, University of Oslo, 2015. [Online]. Available: [http://folk.uio.no/kennetkl/jonassen\\_thesis.pdf](http://folk.uio.no/kennetkl/jonassen_thesis.pdf)
- [29] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, “PIE: A lightweight control scheme to address the bufferbloat problem.” in *HPSR*. IEEE, 2013.
- [30] S. Floyd, R. Gummadi, and S. Shenker, “Adaptive red: An algorithm for increasing the robustness of red’s active queue management,” ACIRI, Tech. Rep., 2001.
- [31] K. Nichols and V. Jacobson, “Controlling Queue Delay,” *Queue*, vol. 10, no. 5, May 2012.
- [32] “Network Emulation (netem),” Linux Foundation, Nov 2009. [Online]. Available: <http://www.linuxfoundation.org/collaborate/workgroups/networking/netem>
- [33] “Kernel Density Estimators.” [Online]. Available: [http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL\\_COPIES/AV0405/MISHRA/kde.html](http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0405/MISHRA/kde.html)
- [34] L. Stewart, D. Hayes, G. Armitage, M. Welzl, and A. Petlund, “Multimedia-unfriendly TCP Congestion Control and Home Gateway Queue Management,” in *Proc. of the ACM Multimedia Systems Conference (MMSys)*, San Jose, California, Feb. 2011, pp. 35–44. [Online]. Available: <http://dx.doi.org/10.1145/1943552.1943558>
- [35] G. Fairhurst and M. Welzl, “The benefits of using Explicit Congestion Notification (ECN),” Internet draft, draft-ietf-aqm-ecn-benefits-06 (work in progress), Jul. 2015. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-aqm-ecn-benefits>
- [36] N. Khademi, M. Welzl, G. Armitage, C. Kulatunga, D. Ros, G. Fairhurst, S. Gjessing, and S. Zander, “Alternative Backoff: Achieving Low Latency and High Throughput with ECN and AQM,” Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150710A, 10 July 2015. [Online]. Available: <http://caia.swin.edu.au/reports/150710A/CAIA-TR-150710A.pdf>
- [37] N. Khademi, M. Welzl, G. Armitage, and G. Fairhurst, “TCP Alternative Backoff with ECN (ABE),” Internet Engineering Task Force, Internet Draft draft-khademi-alternativebackoff-ecn-00, Jun. 2015, (Work in Progress). [Online]. Available: <https://tools.ietf.org/html/draft-khademi-alternativebackoff-ecn-00>

- [38] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ECN) to IP," RFC 3168 (Proposed Standard), Internet Engineering Task Force, Sep. 2001. [Online]. Available: <http://www.ietf.org/rfc/rfc3168.txt>
- [39] B. Davie, B. Briscoe, and J. Tay, "Explicit congestion marking in MPLS," RFC 5129 (Proposed Standard), Internet Engineering Task Force, Jan. 2008, updated by RFC 5462. [Online]. Available: <http://www.ietf.org/rfc/rfc5129.txt>
- [40] B. Briscoe, "Tunnelling of explicit congestion notification," RFC 6040 (Proposed Standard), Internet Engineering Task Force, Nov. 2010. [Online]. Available: <http://www.ietf.org/rfc/rfc6040.txt>
- [41] G. R. W. 2, "Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2," 3GPP, Technical Specification TS 36.300, 2014.
- [42] K. De Schepper, O. Bondarenko, I.-J. Tsang, and B. Briscoe, "'Data Centre to the Home': Ultra-Low Latency for All," in *Under submission*, Jul. 2015.
- [43] K. De Schepper, B. Briscoe, O. Bondarenko, and I.-J. Tsang, "DualQ Coupled AQM for Low Latency, Low Loss and Scalable Throughput," Internet Engineering Task Force, Internet Draft draft-briscoe-aqm-dualq-coupled-00, Aug. 2015, (Work in Progress). [Online]. Available: <http://datatracker.ietf.org/doc/draft-briscoe-aqm-dualq-coupled>
- [44] B. Briscoe, "[tcpPrague] Notes: DCTCP evolution 'bar BoF': Tue 21 Jul 2015, 17:40, Prague," Archived mailing list posting URL: <https://mailarchive.ietf.org/arch/msg/tcpprague/mwWncQg3egPd15FI-tYWiEvRDrvA>, Jul. 2015. [Online]. Available: <https://mailarchive.ietf.org/arch/msg/tcpprague/mwWncQg3egPd15FI-tYWiEvRDrvA>
- [45] M. Kühlewind, R. Scheffenegger, and B. Briscoe, "Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback," Internet Engineering Task Force, Request for Comments rfc7560, Aug. 2015. [Online]. Available: <https://tools.ietf.org/html/rfc7560>
- [46] B. Briscoe, R. Scheffenegger, and M. Kühlewind, "More Accurate ECN Feedback in TCP," Internet Engineering Task Force, Internet Draft draft-kuehlewind-tcpm-accurate-ecn-04, Sep. 2015, (Work in Progress). [Online]. Available: <http://tools.ietf.org/html/draft-kuehlewind-tcpm-accurate-ecn>
- [47] M. Mathis, "Relentless Congestion Control," in *Proc. Int'l Wkshp on Protocols for Future, Large-scale & Diverse Network Transports (PFLDNeT'09)*, May 2009. [Online]. Available: [http://www.hpcc.jp/pfldnet2009/Program\\_files/1569198525.pdf](http://www.hpcc.jp/pfldnet2009/Program_files/1569198525.pdf)
- [48] CAIDA, "University of California, San Diego Supercomputer." [Online]. Available: <http://www.caida.org>
- [49] N. Dukkupati, "Rate control protocol (RCP): Congestion control to make flows complete quickly," Ph.D. dissertation, Department of Electrical Engineering, Stanford University, 2007. [Online]. Available: <http://yuba.stanford.edu/~nanditad/thesis-NanditaD.pdf>
- [50] D. Katabi, "Decoupling Congestion Control from the Bandwidth Allocation Policy and its Application to High Bandwidth-Delay Product Networks," Ph.D. dissertation, MIT, Mar. 2003.
- [51] S. Bensley, L. Eggert, D. Thaler, P. Balasubramanian, and G. Judd, "Microsoft's Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters," Internet Engineering Task Force, Internet Draft draft-bensley-tcpm-dctcp-05, Jul. 2015. [Online]. Available: <https://tools.ietf.org/html/draft-bensley-tcpm-dctcp-05>
- [52] M. Mathis and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism and Requirements," Internet Engineering Task Force, Internet Draft draft-ietf-conex-abstract-mech-12, Jul. 2014, (Work in progress). [Online]. Available: <http://tools.ietf.org/html/draft-ietf-conex-abstract-mech>
- [53] T. Moncaster, B. Briscoe, and A. Jacquet, "A TCP Test to Allow Senders to Identify Receiver Non-Compliance," Internet Engineering Task Force, Internet Draft draft-moncaster-tsvwg-rcv-cheat, Jul. 2014, (Work in progress). [Online]. Available: <http://tools.ietf.org/html/draft-moncaster-tsvwg-rcv-cheat>



- [54] B. Briscoe, “Insights from Curvy RED (Random Early Detection),” Available online, URL: <http://riteproject.eu/publications/>, BT, Technical report TR-TUB8-2015-003, May 2015. [Online]. Available: [http://www.bobbriscoe.net/projects/latency/credi\\_tr.pdf](http://www.bobbriscoe.net/projects/latency/credi_tr.pdf)
- [55] B. Briscoe, “Review: Proportional Integral controller Enhanced (PIE) Active Queue Management (AQM),” Available on-line, URL: <http://riteproject.eu/publications/>, BT, Technical Report TR-TUB8-2015-001, May 2015. [Online]. Available: [http://www.bobbriscoe.net/projects/latency/piervw\\_tr.pdf](http://www.bobbriscoe.net/projects/latency/piervw_tr.pdf)
- [56] E. Grigorescu, C. Kulatunga, G. Fairhurst, and N. Kuhn, “Evaluation of Priority Scheduling and Flow Starvation for Thin Streams with FQ-CoDelkojo,” *European Conference on Networks and Communications (EUCNC) - RITE special session*, 2015.
- [57] C. Kulatunga, N. Kuhn, G. Fairhurst, and D. Ros, “Tackling Bufferbloat in Capacity-limited Networks,” *European Conference on Networks and Communications (EUCNC)*, 2015.
- [58] N. Kuhn, X. Corbillon, and E. Lochin, “On the coexistence of AQM and LBE,” *Under submission*, 2015.
- [59] N. Kuhn and D. Ros, “MADPIE: Maximum and Average queuing Delay with PIE,” *Under submission*, 2015.
- [60] D. Black (IETF tsvwg co chair), “Explicit Congestion Notification (ECN) and IEEE Protocols,” Available on-line, URL: <https://datatracker.ietf.org/liaison/1364/>, Nov. 2014. [Online]. Available: <https://datatracker.ietf.org/liaison/1364/>
- [61] D. Black (IETF tsvwg co chair), “Explicit Congestion Notification for Lower Layer Protocols,” Available on-line, URL: <https://datatracker.ietf.org/liaison/1424/>, Jul. 2015. [Online]. Available: <https://datatracker.ietf.org/liaison/1424/>
- [62] B. Briscoe, J. Kaippallimalil, and P. Thaler, “Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP,” Internet Engineering Task Force, Internet Draft draft-ietf-tsvwg-ecn-encap-guidelines-02, Mar. 2015, (Work in Progress). [Online]. Available: <http://tools.ietf.org/html/draft-ietf-tsvwg-ecn-encap-guidelines>
- [63] B. Briscoe, M. Rajiullah, A. Brunstrom, and A. Petlund, “What Use is Top Speed without Acceleration?,” Reducing Internet Transport Latency (RITE) project, Tech. Rep., Jul. 2015.
- [64] B. Briscoe and P. Hurtig, “Up to Speed with Queue View (QV),” RITE EU FP7 Project 317700, Technical Report, Aug. 2014, (Unpublished).
- [65] M. Sångfors, R. Ludwig, M. Meyer, and J. Peisa, “Buffer Management for Rate-Varying 3G Wireless Links Supporting TCP Traffic,” in *Proc Vehicular Technology Conference*, Apr. 2003.
- [66] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, “Less Is More: Trading a Little Bandwidth for Ultra-Low Latency in the Data Center,” in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI’12)*, Apr. 2012.
- [67] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, “pathChirp: Efficient Available Bandwidth Estimation for Network Paths,” in *Passive and Active Measurement Workshop (PAM’03)*, 2003. [Online]. Available: <http://www.spin.rice.edu/PDF/PAM2003.pdf>
- [68] M. Kühlewind and B. Briscoe, “Chirping for Congestion Control — Implementation Feasibility,” in *Proc. Int’l Wkshp on Protocols for Future, Large-scale & Diverse Network Transports (PFLDNeT’10)*, Nov. 2010. [Online]. Available: <http://pfld.net/2010/paper/session2-3.pdf>
- [69] N. Spring, D. Wetherall, and D. Ely, “Robust Explicit Congestion Notification (ECN) signaling with nonces,” RFC 3540 (Experimental), Internet Engineering Task Force, Jun. 2003. [Online]. Available: <http://www.ietf.org/rfc/rfc3540.txt>
- [70] M. Kühlewind, R. Scheffenegger, and B. Briscoe, “Problem Statement and Requirements for a More Accurate ECN Feedback,” Internet Engineering Task Force, Internet Draft draft-ietf-tcpm-accecn-reqs-07, Jul. 2014, (Work in Progress). [Online]. Available: <http://tools.ietf.org/internet-drafts/draft-ietf-tcpm-accecn-reqs>

- 
- [71] M. Westerlund, I. Johansson, C. Perkins, P. O'Hanlon, and K. Carlberg, "Explicit congestion notification (ECN) for RTP over UDP," RFC 6679 (Proposed Standard), Internet Engineering Task Force, Aug. 2012. [Online]. Available: <http://www.ietf.org/rfc/rfc6679.txt>
  - [72] S. Floyd, M. Allman, A. Jain, and P. Sarolahti, "Quick-Start for TCP and IP," RFC 4782 (Experimental), Internet Engineering Task Force, Jan. 2007. [Online]. Available: <http://www.ietf.org/rfc/rfc4782.txt>
  - [73] D. D. Clark, "The design philosophy of the DARPA internet protocols," *Proc. ACM SIGCOMM'88, Computer Communication Review*, vol. 18, no. 4, pp. 106–114, Aug. 1988. [Online]. Available: <http://www.acm.org/sigcomm/ccr/archive/1995/jan95/ccr-9501-clark.pdf>
  - [74] F. Gont, "Deprecation of ICMP Source Quench messages," RFC 6633 (Proposed Standard), Internet Engineering Task Force, May 2012. [Online]. Available: <http://www.ietf.org/rfc/rfc6633.txt>
  - [75] B. Briscoe, R. Woundy, and A. Cooper, "Congestion Exposure (ConEx) Concepts and Use Cases," RFC 6789 (Informational), Internet Engineering Task Force, Dec. 2012. [Online]. Available: <http://www.ietf.org/rfc/rfc6789.txt>



# Appendices

## A Active Queue Management Publications

In this appendix the publications of the Active Queue Management activity are included:

- F. Baker and G. Fairhurst, “IETF recommendations regarding active queue management,” Internet Draft, RFC 7567, 2015. [Online]. Available: <http://tools.ietf.org/html/draft-ietf-aqm-recommendation>

**Abstract:** This memo presents recommendations to the Internet community concerning measures to improve and preserve Internet performance. It presents a strong recommendation for testing, standardization, and widespread deployment of active queue management (AQM) in network devices, to improve the performance of today’s Internet. It also urges a concerted effort of research, measurement, and ultimate deployment of AQM mechanisms to protect the Internet from flows that are not sufficiently responsive to congestion notification.

The note replaces the recommendations of RFC 2309 based on fifteen years of experience and new research.

- N. Kuhn (Ed.), N. Khademi (Ed.), P. Natarajan (Ed.), and D. Ros, “AQM Characterization Guidelines,” Internet Draft draft-kuhn-aqm-eval-guidelines, work in progress, May 2015. [Online]. Available: <http://tools.ietf.org/html/draft-ietf-aqm-eval-guidelines>

**Abstract:** Unmanaged large buffers in today’s networks have given rise to a slew of performance issues. These performance issues can be addressed by some form of Active Queue Management (AQM) mechanism, optionally in combination with a packet scheduling scheme such as fair queuing. The IETF Active Queue Management and Packet Scheduling working group was formed to standardize AQM schemes that are robust, easily implementable, and successfully deployable in today’s networks. This document describes various criteria for performing precautionary characterizations of AQM proposals. This document also helps in ascertaining whether any given AQM proposal should be taken up for standardization by the AQM WG.

- A. B. Bagayoko, G. Fairhurst, N. Khademi, N. Kuhn, C. Kulatunga, and D. Ros, “Operating ranges and tunability of CoDel and PIE,” *Under submission*, 2015

**Abstract:** Bufferbloat is excessive delay due to the accumulation of packets in a router’s oversized queues. CoDel and PIE are two recent Active Queue Management (AQM) algorithms that have been proposed to address bufferbloat by reducing the queuing delay while trying to maintain a high bottleneck utilization. In this paper, we outline what are the operating ranges, that is the network characteristics (in terms of round-trip times and bottleneck capacity), for which these algorithms achieve their design goals. We highlight deployment scenarios where both AQM schemes result in poor performance when used with default parameters, and we evaluate to what extent we can tune these to achieve various trade-offs. We find that, by appropriate tuning (1) the amount of buffering can easily be controlled with PIE, (2) the RTT sensitivity of CoDel can be reduced. Also, we observe there is more correlation between the congestion level, the achieved queuing delay and the targeted delay with CoDel than with PIE. An overall winner for the best AQM scheme does not exist, as each scheme proposes a specific trade-off, and each works within a limited range of network characteristics or traffic profiles.

- B. Briscoe, “Insights from Curvy RED (Random Early Detection),” Available online, URL: <http://riteproject.eu/publications/>, BT, Technical report TR-TUB8-2015-003, May 2015. [Online]. Available: [http://www.bobbriscoe.net/projects/latency/credi\\_tr.pdf](http://www.bobbriscoe.net/projects/latency/credi_tr.pdf)

**Abstract:** Active queue management (AQM) drops packets early in the growth of a queue, to prevent a capacity-seeking sender (e.g. TCP) from keeping the buffer full. An AQM can mark instead of dropping packets if they indicate support for explicit congestion notification (ECN). Two modern AQMs (PIE and CoDel) are designed to keep queuing delay to a target by dropping packets as load varies.

This memo uses Curvy RED and an idealised but sufficient model of TCP traffic to explain why attempting to keep delay constant is a bad idea, because it requires excessively high drop at high loads. This high drop itself takes over from queuing delay as the dominant cause of delay, particularly for short flows. At high load, a link is better able to preserve reasonable performance if the delay target is softened into a curve rather than a hard cap.

The analysis proves that the same AQM can be deployed in different parts of a network whatever the capacity with the same optimal configuration.

A surprising corollary of this analysis concerns cases with a highly aggregated number of flows through a bottleneck. Although aggregation reduces queue variation, if the target queuing delay of the AQM at that bottleneck is reduced to take advantage of this aggregation, TCP will still increase the loss level because of the reduction in round trip time. The way to resolve this dilemma is to overprovision (a formula is provided).

Nonetheless, for traffic with ECN enabled, there is no harm in an AQM holding queuing delay constant or configuring an AQM to take advantage of any reduced delay due to aggregation without over-provisioning. Recently, the requirement of the ECN standard that ECN must be treated the same as drop has been questioned. The insight that the goals of an AQM for drop and for ECN should be different proves that this doubt is justified.

- B. Briscoe, “Review: Proportional Integral controller Enhanced (PIE) Active Queue Management (AQM),” Available on-line, URL: <http://riteproject.eu/publications/>, BT, Technical Report TR-TUB8-2015-001, May 2015. [Online]. Available: [http://www.bobbriscoe.net/projects/latency/piervw\\_tr.pdf](http://www.bobbriscoe.net/projects/latency/piervw_tr.pdf)

**Abstract:** The PIE draft ends with two assertions in the Discussion section:

“PIE is simple to implement”

“PIE does not require any user configuration”

I am afraid I have to say that I do not believe either statement is warranted any more. PIE has lost its way a bit. The implementation has not retained the elegance of the theory. The performance benefit from so-called ‘enhancements’ is questionable or non-existent, whereas the added complexity is very apparent. Also PIE now contains a large number of hard-coded constants (I counted 20) that ought to be scenario-dependent configuration variables.

- E. Grigorescu, C. Kulatunga, G. Fairhurst, and N. Kuhn, “Evaluation of Priority Scheduling and Flow Starvation for Thin Streams with FQ-CoDelkojo,” *European Conference on Networks and Communications (EUCNC) - RITE special session*, 2015

**Abstract:** Bufferbloat is the result of oversized buffers and induced high end-to-end latency experienced by applications across the Internet. This additional delay can adversely impact thin streams that frequently exchange small amounts of data, but have stringent latency requirements. Active Queue Management (AQM) techniques, such as Controlled Delay (CoDel), can control the queuing delay in a network device to ensure low latency by dropping packets to indicate incipient congestion. FlowQueue- CoDel (FQ-CoDel) is a scheduling scheme that creates one sub-queue per flow and applies CoDel on each of them. FQ-CoDel features: (1) priority scheduling for low-rate traffic; (2) flow isolation; (3) queue management with CoDel. First, this paper fills a gap in the understanding of FQ-CoDel by analyzing what features are of interests for providing low latency for thin streams applications. Second, this paper provides the first analysis of the limits of the flow starvation mechanisms and show that FQ-CoDel is vulnerable to Denial of Service (DoS) attacks.

- C. Kulatunga, N. Kuhn, G. Fairhurst, and D. Ros, “Tackling Bufferbloat in Capacity-limited Networks,” *European Conference on Networks and Communications (EUCNC)*, 2015

**Abstract:** Over-provisioned network buffers, often at the Internet edge, induce large queuing delay and high latency; this issue is known as Bufferbloat. In response to this, a set of recently proposed Active Queue Management (AQM) algorithms attempt to reduce standing queues, while maintaining the bottleneck utilisation at an acceptable level. This paper assesses the performance of two AQM schemes (CoDel and FQ-CoDel) over capacity-limited networks with large Round-Trip Time (RTT). In such settings, these AQM schemes have difficulty controlling the buffering level, resulting in both momentarily high queuing delay and low bottleneck utilisation, even if the methods are claimed to be insensitive to link rates and round-trip delays. We explore this issue and show that it is possible to adapt the parameterisation of CoDel and FQ-CoDel to offer a higher bottleneck utilisation while maintaining a low queuing delay. We present experiments over an emulated test bed and a satellite network to confirm that our new parameterisation improves the download time of moderate-size files and reduces the latency for capacity-limited and large-RTT networks.

- N. Kuhn, X. Corbillon, and E. Lochin, “On the coexistence of AQM and LBE,” *Under submission*, 2015

**Abstract:** Active Queue Management (AQM) schemes are essential to guarantee low delay for latency sensitive applications. Their deployment is a required step to tackle bufferbloat. It has been shown that AQM and Lower-than Best Effort (LBE), that carry non-priority traffic, can hardly coexist. In this paper, we show that with adequate parameterizations, it is possible to provide both low end-to-end delay for latency sensitive applications with AQM and low priority traffic with LBE protocols.

- N. Kuhn and D. Ros, “MADPIE: Maximum and Average queuing Delay with PIE,” *Under submission*, 2015

**Abstract:** Bufferbloat is excessive latency due to over-provisioned network buffers. PIE and CoDel are two recently proposed Active Queue Management (AQM) algorithms, designed to tackle bufferbloat by lowering the queuing delay without degrading the bottleneck utilization. PIE uses a proportional integral controller to maintain the average queuing delay at a desired level; however, large Round Trip Times (RTT) result in large spikes in queuing delays, which induce high dropping probability and low utilization. To deal with this problem, we propose Maximum and Average queuing Delay with PIE (MADPIE). Loosely based on the drop policy used by CoDel to keep queuing delay bounded, MADPIE is a simple extension to PIE that adds deterministic packet drops at controlled intervals. By means of simulations, we observe that our proposed change does not affect PIE’s performance when RTT 100 ms. The deterministic drops are more dominant when the RTT increases, which results in lower maximum queuing delays and better performance for VoIP traffic and small file downloads, with no major impact on bulk transfers.

Internet Engineering Task Force (IETF)  
Request for Comments: 7567  
BCP: 197  
Obsoletes: 2309  
Category: Best Current Practice  
ISSN: 2070-1721

F. Baker, Ed.  
Cisco Systems  
G. Fairhurst, Ed.  
University of Aberdeen  
July 2015

## IETF Recommendations Regarding Active Queue Management

### Abstract

This memo presents recommendations to the Internet community concerning measures to improve and preserve Internet performance. It presents a strong recommendation for testing, standardization, and widespread deployment of active queue management (AQM) in network devices to improve the performance of today's Internet. It also urges a concerted effort of research, measurement, and ultimate deployment of AQM mechanisms to protect the Internet from flows that are not sufficiently responsive to congestion notification.

Based on 15 years of experience and new research, this document replaces the recommendations of RFC 2309.

### Status of This Memo

This memo documents an Internet Best Current Practice.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on BCPS is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <http://www.rfc-editor.org/info/rfc7567>.

RFC 7567

Active Queue Management Recommendations

July 2015

## Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

RFC 7567

Active Queue Management Recommendations

July 2015

## Table of Contents

1. Introduction . . . . .	4
1.1. Congestion Collapse . . . . .	4
1.2. Active Queue Management to Manage Latency . . . . .	5
1.3. Document Overview . . . . .	6
1.4. Changes to the Recommendations of RFC 2309 . . . . .	7
1.5. Requirements Language . . . . .	7
2. The Need for Active Queue Management . . . . .	7
2.1. AQM and Multiple Queues . . . . .	11
2.2. AQM and Explicit Congestion Marking (ECN) . . . . .	12
2.3. AQM and Buffer Size . . . . .	12
3. Managing Aggressive Flows . . . . .	13
4. Conclusions and Recommendations . . . . .	16
4.1. Operational Deployments SHOULD Use AQM Procedures . . . . .	17
4.2. Signaling to the Transport Endpoints . . . . .	17
4.2.1. AQM and ECN . . . . .	18
4.3. AQM Algorithm Deployment SHOULD NOT Require Operational Tuning . . . . .	20
4.4. AQM Algorithms SHOULD Respond to Measured Congestion, Not Application Profiles . . . . .	21
4.5. AQM Algorithms SHOULD NOT Be Dependent on Specific Transport Protocol Behaviors . . . . .	22
4.6. Interactions with Congestion Control Algorithms . . . . .	22
4.7. The Need for Further Research . . . . .	23
5. Security Considerations . . . . .	25
6. Privacy Considerations . . . . .	25
7. References . . . . .	25
7.1. Normative References . . . . .	25
7.2. Informative References . . . . .	26
Acknowledgements . . . . .	31
Authors' Addresses . . . . .	31

## 1. Introduction

The Internet protocol architecture is based on a connectionless end-to-end packet service using the Internet Protocol, whether IPv4 [RFC791] or IPv6 [RFC2460]. The advantages of its connectionless design -- flexibility and robustness -- have been amply demonstrated. However, these advantages are not without cost: careful design is required to provide good service under heavy load. In fact, lack of attention to the dynamics of packet forwarding can result in severe service degradation or "Internet meltdown". This phenomenon was first observed during the early growth phase of the Internet in the mid 1980s [RFC896] [RFC970]; it is technically called "congestion collapse" and was a key focus of RFC 2309.

Although wide-scale congestion collapse is not common in the Internet, the presence of localized congestion collapse is by no means rare. It is therefore important to continue to avoid congestion collapse.

Since 1998, when RFC 2309 was written, the Internet has become used for a variety of traffic. In the current Internet, low latency is extremely important for many interactive and transaction-based applications. The same type of technology that RFC 2309 advocated for combating congestion collapse is also effective at limiting delays to reduce the interaction delay (latency) experienced by applications [Bril5]. High or unpredictable latency can impact the performance of the control loops used by end-to-end protocols (including congestion control algorithms using TCP). There is now also a focus on reducing network latency using the same technology.

The mechanisms described in this document may be implemented in network devices on the path between endpoints that include routers, switches, and other network middleboxes. The methods may also be implemented in the networking stacks within endpoint devices that connect to the network.

### 1.1. Congestion Collapse

The original fix for Internet meltdown was provided by Van Jacobsen. Beginning in 1986, Jacobsen developed the congestion avoidance mechanisms [Jacobson88] that are now required for implementations of the Transport Control Protocol (TCP) [RFC793] [RFC1122]. ([RFC7414] provides a roadmap to help identify TCP-related documents.) These mechanisms operate in Internet hosts to cause TCP connections to "back off" during congestion. We say that TCP flows are "responsive" to congestion signals (i.e., packets that are dropped or marked with explicit congestion notification [RFC3168]). It is primarily these



TCP congestion avoidance algorithms that prevent the congestion collapse of today's Internet. Similar algorithms are specified for other non-TCP transports.

However, that is not the end of the story. Considerable research has been done on Internet dynamics since 1988, and the Internet has grown. It has become clear that the congestion avoidance mechanisms [RFC5681], while necessary and powerful, are not sufficient to provide good service in all circumstances. Basically, there is a limit to how much control can be accomplished from the edges of the network. Some mechanisms are needed in network devices to complement the endpoint congestion avoidance mechanisms. These mechanisms may be implemented in network devices.

### 1.2. Active Queue Management to Manage Latency

Internet latency has become a focus of attention to increase the responsiveness of Internet applications and protocols. One major source of delay is the buildup of queues in network devices. Queueing occurs whenever the arrival rate of data at the ingress to a device exceeds the current egress rate. Such queueing is normal in a packet-switched network and is often necessary to absorb bursts in transmission and perform statistical multiplexing of traffic, but excessive queueing can lead to unwanted delay, reducing the performance of some Internet applications.

RFC 2309 introduced the concept of "Active Queue Management" (AQM), a class of technologies that, by signaling to common congestion-controlled transports such as TCP, manages the size of queues that build in network buffers. RFC 2309 also describes a specific AQM algorithm, Random Early Detection (RED), and recommends that this be widely implemented and used by default in routers.

With an appropriate set of parameters, RED is an effective algorithm. However, dynamically predicting this set of parameters was found to be difficult. As a result, RED has not been enabled by default, and its present use in the Internet is limited. Other AQM algorithms have been developed since RFC 2309 was published, some of which are self-tuning within a range of applicability. Hence, while this memo continues to recommend the deployment of AQM, it no longer recommends that RED or any other specific algorithm is used by default. It instead provides recommendations on IETF processes for the selection of appropriate algorithms, and especially that a recommended algorithm is able to automate any required tuning for common deployment scenarios.

Deploying AQM in the network can significantly reduce the latency across an Internet path, and, since the writing of RFC 2309, this has become a key motivation for using AQM in the Internet. In the context of AQM, it is useful to distinguish between two related classes of algorithms: "queue management" versus "scheduling" algorithms. To a rough approximation, queue management algorithms manage the length of packet queues by marking or dropping packets when necessary or appropriate, while scheduling algorithms determine which packet to send next and are used primarily to manage the allocation of bandwidth among flows. While these two mechanisms are closely related, they address different performance issues and operate on different timescales. Both may be used in combination.

### 1.3. Document Overview

The discussion in this memo applies to "best-effort" traffic, which is to say, traffic generated by applications that accept the occasional loss, duplication, or reordering of traffic in flight. It also applies to other traffic, such as real-time traffic that can adapt its sending rate to reduce loss and/or delay. It is most effective when the adaption occurs on timescales of a single Round-Trip Time (RTT) or a small number of RTTs, for elastic traffic [RFC1633].

Two performance issues are highlighted:

The first issue is the need for an advanced form of queue management that we call "Active Queue Management", AQM. Section 2 summarizes the benefits that active queue management can bring. A number of AQM procedures are described in the literature, with different characteristics. This document does not recommend any of them in particular, but it does make recommendations that ideally would affect the choice of procedure used in a given implementation.

The second issue, discussed in Section 4 of this memo, is the potential for future congestion collapse of the Internet due to flows that are unresponsive, or not sufficiently responsive, to congestion indications. Unfortunately, while scheduling can mitigate some of the side effects of sharing a network queue with an unresponsive flow, there is currently no consensus solution to controlling the congestion caused by such aggressive flows. Methods such as congestion exposure (ConEx) [RFC6789] offer a framework [CONEX] that can update network devices to alleviate these effects. Significant research and engineering will be required before any solution will be available. It is imperative that work to mitigate the impact of unresponsive flows is energetically pursued to ensure acceptable performance and the future stability of the Internet.

RFC 7567

Active Queue Management Recommendations

July 2015

Section 4 concludes the memo with a set of recommendations to the Internet community on the use of AQM and recommendations for defining AQM algorithms.

#### 1.4. Changes to the Recommendations of RFC 2309

This memo replaces the recommendations in [RFC2309], which resulted from past discussions of end-to-end performance, Internet congestion, and RED in the End-to-End Research Group of the Internet Research Task Force (IRTF). It results from experience with RED and other algorithms, and the AQM discussion within the IETF [AQM-WG].

Whereas RFC 2309 described AQM in terms of the length of a queue, this memo uses AQM to refer to any method that allows network devices to control the queue length and/or the mean time that a packet spends in a queue.

This memo also explicitly obsoletes the recommendation that Random Early Detection (RED) be used as the default AQM mechanism for the Internet. This is replaced by a detailed set of recommendations for selecting an appropriate AQM algorithm. As in RFC 2309, this memo illustrates the need for continued research. It also clarifies the research needed with examples appropriate at the time that this memo is published.

#### 1.5. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

### 2. The Need for Active Queue Management

Active Queue Management (AQM) is a method that allows network devices to control the queue length or the mean time that a packet spends in a queue. Although AQM can be applied across a range of deployment environments, the recommendations in this document are for use in the general Internet. It is expected that the principles and guidance are also applicable to a wide range of environments, but they may require tuning for specific types of links or networks (e.g., to accommodate the traffic patterns found in data centers, the challenges of wireless infrastructure, or the higher delay encountered on satellite Internet links). The remainder of this section identifies the need for AQM and the advantages of deploying AQM methods.

The traditional technique for managing the queue length in a network device is to set a maximum length (in terms of packets) for each queue, accept packets for the queue until the maximum length is reached, then reject (drop) subsequent incoming packets until the queue decreases because a packet from the queue has been transmitted. This technique is known as "tail drop", since the packet that arrived most recently (i.e., the one on the tail of the queue) is dropped when the queue is full. This method has served the Internet well for years, but it has four important drawbacks:

1. Full Queues

The "tail drop" discipline allows queues to maintain a full (or, almost full) status for long periods of time, since tail drop signals congestion (via a packet drop) only when the queue has become full. It is important to reduce the steady-state queue size, and this is perhaps the most important goal for queue management.

The naive assumption might be that there is a simple trade-off between delay and throughput, and that the recommendation that queues be maintained in a "non-full" state essentially translates to a recommendation that low end-to-end delay is more important than high throughput. However, this does not take into account the critical role that packet bursts play in Internet performance. For example, even though TCP constrains the congestion window of a flow, packets often arrive at network devices in bursts [Leland94]. If the queue is full or almost full, an arriving burst will cause multiple packets to be dropped from the same flow. Bursts of loss can result in a global synchronization of flows throttling back, followed by a sustained period of lowered link utilization, reducing overall throughput [Flo94] [Zha90].

The goal of buffering in the network is to absorb data bursts and to transmit them during the (hopefully) ensuing bursts of silence. This is essential to permit transmission of bursts of data. Queues that are normally small are preferred in network devices, with sufficient queue capacity to absorb the bursts. The counterintuitive result is that maintaining queues that are normally small can result in higher throughput as well as lower end-to-end delay. In summary, queue limits should not reflect the steady-state queues we want to be maintained in the network; instead, they should reflect the size of bursts that a network device needs to absorb.

RFC 7567

Active Queue Management Recommendations

July 2015

## 2. Lock-Out

In some situations tail drop allows a single connection or a few flows to monopolize the queue space, thereby starving other connections, preventing them from getting room in the queue [Flo92].

## 3. Mitigating the Impact of Packet Bursts

A large burst of packets can delay other packets, disrupting the control loop (e.g., the pacing of flows by the TCP ACK clock), and reducing the performance of flows that share a common bottleneck.

## 4. Control Loop Synchronization

Congestion control, like other end-to-end mechanisms, introduces a control loop between hosts. Sessions that share a common network bottleneck can therefore become synchronized, introducing periodic disruption (e.g., jitter/loss). "Lock-out" is often also the result of synchronization or other timing effects

Besides tail drop, two alternative queue management disciplines that can be applied when a queue becomes full are "random drop on full" or "head drop on full". When a new packet arrives at a full queue using the "random drop on full" discipline, the network device drops a randomly selected packet from the queue (this can be an expensive operation, since it naively requires an  $O(N)$  walk through the packet queue). When a new packet arrives at a full queue using the "head drop on full" discipline, the network device drops the packet at the front of the queue [Lakshman96]. Both of these solve the lock-out problem, but neither solves the full-queues problem described above.

In general, we know how to solve the full-queues problem for "responsive" flows, i.e., those flows that throttle back in response to congestion notification. In the current Internet, dropped packets provide a critical mechanism indicating congestion notification to hosts. The solution to the full-queues problem is for network devices to drop or ECN-mark packets before a queue becomes full, so that hosts can respond to congestion before buffers overflow. We call such a proactive approach AQM. By dropping or ECN-marking packets before buffers overflow, AQM allows network devices to control when and how many packets to drop.

In summary, an active queue management mechanism can provide the following advantages for responsive flows.

1. Reduce number of packets dropped in network devices

Packet bursts are an unavoidable aspect of packet networks [Willinger95]. If all the queue space in a network device is already committed to "steady-state" traffic or if the buffer space is inadequate, then the network device will have no ability to buffer bursts. By keeping the average queue size small, AQM will provide greater capacity to absorb naturally occurring bursts without dropping packets.

Furthermore, without AQM, more packets will be dropped when a queue does overflow. This is undesirable for several reasons. First, with a shared queue and the "tail drop" discipline, this can result in unnecessary global synchronization of flows, resulting in lowered average link utilization and, hence, lowered network throughput. Second, unnecessary packet drops represent a waste of network capacity on the path before the drop point.

While AQM can manage queue lengths and reduce end-to-end latency even in the absence of end-to-end congestion control, it will be able to reduce packet drops only in an environment that continues to be dominated by end-to-end congestion control.

2. Provide a lower-delay interactive service

By keeping a small average queue size, AQM will reduce the delays experienced by flows. This is particularly important for interactive applications such as short web transfers, POP/IMAP, DNS, terminal traffic (Telnet, SSH, Mosh, RDP, etc.), gaming or interactive audio-video sessions, whose subjective (and objective) performance is better when the end-to-end delay is low.

3. Avoid lock-out behavior

AQM can prevent lock-out behavior by ensuring that there will almost always be a buffer available for an incoming packet. For the same reason, AQM can prevent a bias against low-capacity, but highly bursty, flows.

Lock-out is undesirable because it constitutes a gross unfairness among groups of flows. However, we stop short of calling this benefit "increased fairness", because general fairness among flows requires per-flow state, which is not provided by queue management. For example, in a network device using AQM with only

FIFO scheduling, two TCP flows may receive very different shares of the network capacity simply because they have different RTTs [Floyd91], and a flow that does not use congestion control may receive more capacity than a flow that does. AQM can therefore be combined with a scheduling mechanism that divides network traffic between multiple queues (Section 2.1).

#### 4. Reduce the probability of control loop synchronization

The probability of network control loop synchronization can be reduced if network devices introduce randomness in the AQM functions that trigger congestion avoidance at the sending host.

#### 2.1. AQM and Multiple Queues

A network device may use per-flow or per-class queueing with a scheduling algorithm to either prioritize certain applications or classes of traffic, limit the rate of transmission, or provide isolation between different traffic flows within a common class. For example, a router may maintain per-flow state to achieve general fairness by a per-flow scheduling algorithm such as various forms of Fair Queueing (FQ) [Dem90] [Sut99], including Weighted Fair Queueing (WFQ), Stochastic Fairness Queueing (SFQ) [McK90], Deficit Round Robin (DRR) [Shr96] [Nic12], and/or a Class-Based Queue scheduling algorithm such as CBQ [Floyd95]. Hierarchical queues may also be used, e.g., as a part of a Hierarchical Token Bucket (HTB) or Hierarchical Fair Service Curve (HFSC) [Sto97]. These methods are also used to realize a range of Quality of Service (QoS) behaviors designed to meet the need of traffic classes (e.g., using the integrated or differentiated service models).

AQM is needed even for network devices that use per-flow or per-class queueing, because scheduling algorithms by themselves do not control the overall queue size or the sizes of individual queues. AQM mechanisms might need to control the overall queue sizes to ensure that arriving bursts can be accommodated without dropping packets. AQM should also be used to control the queue size for each individual flow or class, so that they do not experience unnecessarily high delay. Using a combination of AQM and scheduling between multiple queues has been shown to offer good results in experimental use and some types of operational use.

In short, scheduling algorithms and queue management should be seen as complementary, not as replacements for each other.

RFC 7567

Active Queue Management Recommendations

July 2015

## 2.2. AQM and Explicit Congestion Marking (ECN)

An AQM method may use Explicit Congestion Notification (ECN) [RFC3168] instead of dropping to mark packets under mild or moderate congestion. ECN-marking can allow a network device to signal congestion at a point before a transport experiences congestion loss or additional queueing delay [ECN-Benefit]. Section 4.2.1 describes some of the benefits of using ECN with AQM.

## 2.3. AQM and Buffer Size

It is important to differentiate the choice of buffer size for a queue in a switch/router or other network device, and the threshold(s) and other parameters that determine how and when an AQM algorithm operates. The optimum buffer size is a function of operational requirements and should generally be sized to be sufficient to buffer the largest normal traffic burst that is expected. This size depends on the amount and burstiness of traffic arriving at the queue and the rate at which traffic leaves the queue.

One objective of AQM is to minimize the effect of lock-out, where one flow prevents other flows from effectively gaining capacity. This need can be illustrated by a simple example of drop-tail queueing when a new TCP flow injects packets into a queue that happens to be almost full. A TCP flow's congestion control algorithm [RFC5681] increases the flow rate to maximize its effective window. This builds a queue in the network, inducing latency in the flow and other flows that share this queue. Once a drop-tail queue fills, there will also be loss. A new flow, sending its initial burst, has an enhanced probability of filling the remaining queue and dropping packets. As a result, the new flow can be prevented from effectively sharing the queue for a period of many RTTs. In contrast, AQM can minimize the mean queue depth and therefore reduce the probability that competing sessions can materially prevent each other from performing well.

AQM frees a designer from having to limit the buffer space assigned to a queue to achieve acceptable performance, allowing allocation of sufficient buffering to satisfy the needs of the particular traffic pattern. Different types of traffic and deployment scenarios will lead to different requirements. The choice of AQM algorithm and associated parameters is therefore a function of the way in which congestion is experienced and the required reaction to achieve acceptable performance. The latter is the primary topic of the following sections.



### 3. Managing Aggressive Flows

One of the keys to the success of the Internet has been the congestion avoidance mechanisms of TCP. Because TCP "backs off" during congestion, a large number of TCP connections can share a single, congested link in such a way that link bandwidth is shared reasonably equitably among similarly situated flows. The equitable sharing of bandwidth among flows depends on all flows running compatible congestion avoidance algorithms, i.e., methods conformant with the current TCP specification [RFC5681].

In this document, a flow is known as "TCP-friendly" when it has a congestion response that approximates the average response expected of a TCP flow. One example method of a TCP-friendly scheme is the TCP-Friendly Rate Control algorithm [RFC5348]. In this document, the term is used more generally to describe this and other algorithms that meet these goals.

There are a variety of types of network flow. Some convenient classes that describe flows are: (1) TCP-friendly flows, (2) unresponsive flows, i.e., flows that do not slow down when congestion occurs, and (3) flows that are responsive but are less responsive to congestion than TCP. The last two classes contain more aggressive flows that can pose significant threats to Internet performance.

#### 1. TCP-friendly flows

A TCP-friendly flow responds to congestion notification within a small number of path RTTs, and in steady-state it uses no more capacity than a conformant TCP running under comparable conditions (drop rate, RTT, packet size, etc.). This is described in the remainder of the document.

#### 2. Non-responsive flows

A non-responsive flow does not adjust its rate in response to congestion notification within a small number of path RTTs; it can also use more capacity than a conformant TCP running under comparable conditions. There is a growing set of applications whose congestion avoidance algorithms are inadequate or nonexistent (i.e., a flow that does not throttle its sending rate when it experiences congestion).

The User Datagram Protocol (UDP) [RFC768] provides a minimal, best-effort transport to applications and upper-layer protocols (both simply called "applications" in the remainder of this document) and does not itself provide mechanisms to prevent congestion collapse or establish a degree of fairness [RFC5405].

Examples that use UDP include some streaming applications for packet voice and video, and some multicast bulk data transport. Other traffic, when aggregated, may also become unresponsive to congestion notification. If no action is taken, such unresponsive flows could lead to a new congestion collapse [RFC2914]. Some applications can even increase their traffic volume in response to congestion (e.g., by adding Forward Error Correction when loss is experienced), with the possibility that they contribute to congestion collapse.

In general, applications need to incorporate effective congestion avoidance mechanisms [RFC5405]. Research continues to be needed to identify and develop ways to accomplish congestion avoidance for presently unresponsive applications. Network devices need to be able to protect themselves against unresponsive flows, and mechanisms to accomplish this must be developed and deployed. Deployment of such mechanisms would provide an incentive for all applications to become responsive by either using a congestion-controlled transport (e.g., TCP, SCTP [RFC4960], and DCCP [RFC4340]) or incorporating their own congestion control in the application [RFC5405] [RFC6679].

### 3. Transport flows that are less responsive than TCP

A second threat is posed by transport protocol implementations that are responsive to congestion, but, either deliberately or through faulty implementation, reduce the effective window less than a TCP flow would have done in response to congestion. This covers a spectrum of behaviors between (1) and (2). If applications are not sufficiently responsive to congestion signals, they may gain an unfair share of the available network capacity.

For example, the popularity of the Internet has caused a proliferation in the number of TCP implementations. Some of these may fail to implement the TCP congestion avoidance mechanisms correctly because of poor implementation. Others may deliberately be implemented with congestion avoidance algorithms that are more aggressive in their use of capacity than other TCP implementations; this would allow a vendor to claim to have a "faster TCP". The logical consequence of such implementations would be a spiral of increasingly aggressive TCP implementations, leading back to the point where there is effectively no congestion avoidance and the Internet is chronically congested.

Another example could be an RTP/UDP video flow that uses an adaptive codec, but responds incompletely to indications of congestion or responds over an excessively long time period.

Such flows are unlikely to be responsive to congestion signals in a time frame comparable to a small number of end-to-end transmission delays. However, over a longer timescale, perhaps seconds in duration, they could moderate their speed, or increase their speed if they determine capacity to be available.

Tunneled traffic aggregates carrying multiple (short) TCP flows can be more aggressive than standard bulk TCP. Applications (e.g., web browsers primarily supporting HTTP 1.1 and peer-to-peer file-sharing) have exploited this by opening multiple connections to the same endpoint.

Lastly, some applications (e.g., web browsers primarily supporting HTTP 1.1) open a large numbers of successive short TCP flows for a single session. This can lead to each individual flow spending the majority of time in the exponential TCP slow start phase, rather than in TCP congestion avoidance. The resulting traffic aggregate can therefore be much less responsive than a single standard TCP flow.

The projected increase in the fraction of total Internet traffic for more aggressive flows in classes 2 and 3 could pose a threat to the performance of the future Internet. There is therefore an urgent need for measurements of current conditions and for further research into the ways of managing such flows. This raises many difficult issues in finding methods with an acceptable overhead cost that can identify and isolate unresponsive flows or flows that are less responsive than TCP. Finally, there is as yet little measurement or simulation evidence available about the rate at which these threats are likely to be realized or about the expected benefit of algorithms for managing such flows.

Another topic requiring consideration is the appropriate granularity of a "flow" when considering a queue management method. There are a few "natural" answers: 1) a transport (e.g., TCP or UDP) flow (source address/port, destination address/port, protocol); 2) Differentiated Services Code Point, DSCP; 3) a source/destination host pair (IP address); 4) a given source host or a given destination host, or various combinations of the above; 5) a subscriber or site receiving the Internet service (enterprise or residential).

The source/destination host pair gives an appropriate granularity in many circumstances. However, different vendors/providers use different granularities for defining a flow (as a way of "distinguishing" themselves from one another), and different granularities may be chosen for different places in the network. It may be the case that the granularity is less important than the fact that a network device needs to be able to deal with more unresponsive

flows at *\*some\** granularity. The granularity of flows for congestion management is, at least in part, a question of policy that needs to be addressed in the wider IETF community.

#### 4. Conclusions and Recommendations

The IRTF, in producing [RFC2309], and the IETF in subsequent discussion, have developed a set of specific recommendations regarding the implementation and operational use of AQM procedures. The recommendations provided by this document are summarized as:

1. Network devices SHOULD implement some AQM mechanism to manage queue lengths, reduce end-to-end latency, and avoid lock-out phenomena within the Internet.
2. Deployed AQM algorithms SHOULD support Explicit Congestion Notification (ECN) as well as loss to signal congestion to endpoints.
3. AQM algorithms SHOULD NOT require tuning of initial or configuration parameters in common use cases.
4. AQM algorithms SHOULD respond to measured congestion, not application profiles.
5. AQM algorithms SHOULD NOT interpret specific transport protocol behaviors.
6. Congestion control algorithms for transport protocols SHOULD maximize their use of available capacity (when there is data to send) without incurring undue loss or undue round-trip delay.
7. Research, engineering, and measurement efforts are needed regarding the design of mechanisms to deal with flows that are unresponsive to congestion notification or are responsive, but are more aggressive than present TCP.

These recommendations are expressed using the word "SHOULD". This is in recognition that there may be use cases that have not been envisaged in this document in which the recommendation does not apply. Therefore, care should be taken in concluding that one's use case falls in that category; during the life of the Internet, such use cases have been rarely, if ever, observed and reported. To the contrary, available research [Choi04] says that even high-speed links in network cores that are normally very stable in depth and behavior experience occasional issues that need moderation. The recommendations are detailed in the following sections.

#### 4.1. Operational Deployments SHOULD Use AQM Procedures

AQM procedures are designed to minimize the delay and buffer exhaustion induced in the network by queues that have filled as a result of host behavior. Marking and loss behaviors provide a signal that buffers within network devices are becoming unnecessarily full and that the sender would do well to moderate its behavior.

The use of scheduling mechanisms, such as priority queueing, classful queueing, and fair queueing, is often effective in networks to help a network serve the needs of a range of applications. Network operators can use these methods to manage traffic passing a choke point. This is discussed in [RFC2474] and [RFC2475]. When scheduling is used, AQM should be applied across the classes or flows as well as within each class or flow:

- o AQM mechanisms need to control the overall queue sizes to ensure that arriving bursts can be accommodated without dropping packets.
- o AQM mechanisms need to allow combination with other mechanisms, such as scheduling, to allow implementation of policies for providing fairness between different flows.
- o AQM should be used to control the queue size for each individual flow or class, so that they do not experience unnecessarily high delay.

#### 4.2. Signaling to the Transport Endpoints

There are a number of ways a network device may signal to the endpoint that the network is becoming congested and trigger a reduction in rate. The signaling methods include:

- o Delaying transport segments (packets) in flight, such as in a queue.
- o Dropping transport segments (packets) in transit.
- o Marking transport segments (packets), such as using Explicit Congestion Control [RFC3168] [RFC4301] [RFC4774] [RFC6040] [RFC6679].

Increased network latency is used as an implicit signal of congestion. For example, in TCP, additional delay can affect ACK clocking and has the result of reducing the rate of transmission of new data. In the Real-time Transport Protocol (RTP), network latency impacts the RTCP-reported RTT, and increased latency can trigger a sender to adjust its rate. Methods such as Low Extra Delay

RFC 7567

Active Queue Management Recommendations

July 2015

Background Transport (LEDBAT) [RFC6817] assume increased latency as a primary signal of congestion. Appropriate use of delay-based methods and the implications of AQM presently remain an area for further research.

It is essential that all Internet hosts respond to loss [RFC5681] [RFC5405] [RFC4960] [RFC4340]. Packet dropping by network devices that are under load has two effects: It protects the network, which is the primary reason that network devices drop packets. The detection of loss also provides a signal to a reliable transport (e.g., TCP, SCTP) that there is incipient congestion, using a pragmatic but ambiguous heuristic. Whereas, when the network discards a message in flight, the loss may imply the presence of faulty equipment or media in a path, or it may imply the presence of congestion. To be conservative, a transport must assume it may be the latter. Applications using unreliable transports (e.g., using UDP) need to similarly react to loss [RFC5405].

Network devices SHOULD use an AQM algorithm to measure local congestion and to determine the packets to mark or drop so that the congestion is managed.

In general, dropping multiple packets from the same sessions in the same RTT is ineffective and can reduce throughput. Also, dropping or marking packets from multiple sessions simultaneously can have the effect of synchronizing them, resulting in increasing peaks and troughs in the subsequent traffic load. Hence, AQM algorithms SHOULD randomize dropping in time, to reduce the probability that congestion indications are only experienced by a small proportion of the active flows.

Loss due to dropping also has an effect on the efficiency of a flow and can significantly impact some classes of application. In reliable transports, the dropped data must be subsequently retransmitted. While other applications/transports may adapt to the absence of lost data, this still implies inefficient use of available capacity, and the dropped traffic can affect other flows. Hence, congestion signaling by loss is not entirely positive; it is a necessary evil.

#### 4.2.1. AQM and ECN

Explicit Congestion Notification (ECN) [RFC4301] [RFC4774] [RFC6040] [RFC6679] is a network-layer function that allows a transport to receive network congestion information from a network device without incurring the unintended consequences of loss. ECN includes both

transport mechanisms and functions implemented in network devices; the latter rely upon using AQM to decide when and whether to ECN-mark.

Congestion for ECN-capable transports is signaled by a network device setting the "Congestion Experienced (CE)" codepoint in the IP header. This codepoint is noted by the remote receiving endpoint and signaled back to the sender using a transport protocol mechanism, allowing the sender to trigger timely congestion control. The decision to set the CE codepoint requires an AQM algorithm configured with a threshold. Non-ECN capable flows (the default) are dropped under congestion.

Network devices SHOULD use an AQM algorithm that marks ECN-capable traffic when making decisions about the response to congestion. Network devices need to implement this method by marking ECN-capable traffic or by dropping non-ECN-capable traffic.

Safe deployment of ECN requires that network devices drop excessive traffic, even when marked as originating from an ECN-capable transport. This is a necessary safety precaution because:

1. A non-conformant, broken, or malicious receiver could conceal an ECN mark and not report this to the sender;
2. A non-conformant, broken, or malicious sender could ignore a reported ECN mark, as it could ignore a loss without using ECN;
3. A malfunctioning or non-conforming network device may "hide" an ECN mark (or fail to correctly set the ECN codepoint at an egress of a network tunnel).

In normal operation, such cases should be very uncommon; however, overload protection is desirable to protect traffic from misconfigured or malicious use of ECN (e.g., a denial-of-service attack that generates ECN-capable traffic that is unresponsive to CE-marking).

When ECN is added to a scheme, the ECN support MAY define a separate set of parameters from those used for controlling packet drop. The AQM algorithm SHOULD still auto-tune these ECN-specific parameters. These parameters SHOULD also be manually configurable.

Network devices SHOULD use an algorithm to drop excessive traffic (e.g., at some level above the threshold for CE-marking), even when the packets are marked as originating from an ECN-capable transport.

#### 4.3. AQM Algorithm Deployment SHOULD NOT Require Operational Tuning

A number of AQM algorithms have been proposed. Many require some form of tuning or setting of parameters for initial network conditions. This can make these algorithms difficult to use in operational networks.

AQM algorithms need to consider both "initial conditions" and "operational conditions". The former includes values that exist before any experience is gathered about the use of the algorithm, such as the configured speed of interface, support for full-duplex communication, interface MTU, and other properties of the link. Other properties include information observed from monitoring the size of the queue, the queueing delay experienced, rate of packet discard, etc.

This document therefore specifies that AQM algorithms that are proposed for deployment in the Internet have the following properties:

- o AQM algorithm deployment SHOULD NOT require tuning. An algorithm MUST provide a default behavior that auto-tunes to a reasonable performance for typical network operational conditions. This is expected to ease deployment and operation. Initial conditions, such as the interface rate and MTU size or other values derived from these, MAY be required by an AQM algorithm.
- o AQM algorithm deployment MAY support further manual tuning that could improve performance in a specific deployed network. Algorithms that lack such variables are acceptable, but, if such variables exist, they SHOULD be externalized (made visible to the operator). The specification should identify any cases in which auto-tuning is unlikely to achieve acceptable performance and give guidance on the parametric adjustments necessary. For example, the expected response of an algorithm may need to be configured to accommodate the largest expected Path RTT, since this value cannot be known at initialization. This guidance is expected to enable the algorithm to be deployed in networks that have specific characteristics (paths with variable or larger delay, networks where capacity is impacted by interactions with lower-layer mechanisms, etc).



- o AQM algorithm deployment MAY provide logging and alarm signals to assist in identifying if an algorithm using manual or auto-tuning is functioning as expected. (For example, this could be based on an internal consistency check between input, output, and mark/drop rates over time.) This is expected to encourage deployment by default and allow operators to identify potential interactions with other network functions.

Hence, self-tuning algorithms are to be preferred. Algorithms recommended for general Internet deployment by the IETF need to be designed so that they do not require operational (especially manual) configuration or tuning.

#### 4.4. AQM Algorithms SHOULD Respond to Measured Congestion, Not Application Profiles

Not all applications transmit packets of the same size. Although applications may be characterized by particular profiles of packet size, this should not be used as the basis for AQM (see Section 4.5). Other methods exist, e.g., Differentiated Services queueing, Pre-Congestion Notification (PCN) [RFC5559], that can be used to differentiate and police classes of application. Network devices may combine AQM with these traffic classification mechanisms and perform AQM only on specific queues within a network device.

An AQM algorithm should not deliberately try to prejudice the size of packet that performs best (i.e., preferentially drop/mark based only on packet size). Procedures for selecting packets to drop/mark SHOULD observe the actual or projected time that a packet is in a queue (bytes at a rate being an analog to time). When an AQM algorithm decides whether to drop (or mark) a packet, it is RECOMMENDED that the size of the particular packet not be taken into account [RFC7141].

Applications (or transports) generally know the packet size that they are using and can hence make their judgments about whether to use small or large packets based on the data they wish to send and the expected impact on the delay, throughput, or other performance parameter. When a transport or application responds to a dropped or marked packet, the size of the rate reduction should be proportionate to the size of the packet that was sent [RFC7141].

An AQM-enabled system MAY instantiate different instances of an AQM algorithm to be applied within the same traffic class. Traffic classes may be differentiated based on an Access Control List (ACL), the packet DSCP [RFC5559], enabling use of the ECN field (i.e., any of ECT(0), ECT(1) or CE) [RFC3168] [RFC4774], a multi-field (MF) classifier that combines the values of a set of protocol fields

(e.g., IP address, transport, ports), or an equivalent codepoint at a lower layer. This recommendation goes beyond what is defined in RFC 3168 by allowing that an implementation MAY use more than one instance of an AQM algorithm to handle both ECN-capable and non-ECN-capable packets.

#### 4.5. AQM Algorithms SHOULD NOT Be Dependent on Specific Transport Protocol Behaviors

In deploying AQM, network devices need to support a range of Internet traffic and SHOULD NOT make implicit assumptions about the characteristics desired by the set of transports/applications the network supports. That is, AQM methods should be opaque to the choice of transport and application.

AQM algorithms are often evaluated by considering TCP [RFC793] with a limited number of applications. Although TCP is the predominant transport in the Internet today, this no longer represents a sufficient selection of traffic for verification. There is significant use of UDP [RFC768] in voice and video services, and some applications find utility in SCTP [RFC4960] and DCCP [RFC4340]. Hence, AQM algorithms should demonstrate operation with transports other than TCP and need to consider a variety of applications. When selecting AQM algorithms, the use of tunnel encapsulations that may carry traffic aggregates needs to be considered.

AQM algorithms SHOULD NOT target or derive implicit assumptions about the characteristics desired by specific transports/applications. Transports and applications need to respond to the congestion signals provided by AQM (i.e., dropping or ECN-marking) in a timely manner (within a few RTTs at the latest).

#### 4.6. Interactions with Congestion Control Algorithms

Applications and transports need to react to received implicit or explicit signals that indicate the presence of congestion. This section identifies issues that can impact the design of transport protocols when using paths that use AQM.

Transport protocols and applications need timely signals of congestion. The time taken to detect and respond to congestion is increased when network devices queue packets in buffers. It can be difficult to detect tail losses at a higher layer, and this may sometimes require transport timers or probe packets to detect and respond to such loss. Loss patterns may also impact timely detection, e.g., the time may be reduced when network devices do not drop long runs of packets from the same flow.

A common objective of an elastic transport congestion control protocol is to allow an application to deliver the maximum rate of data without inducing excessive delays when packets are queued in buffers within the network. To achieve this, a transport should try to operate at rate below the inflection point of the load/delay curve (the bend of what is sometimes called a "hockey stick" curve) [Jain94]. When the congestion window allows the load to approach this bend, the end-to-end delay starts to rise -- a result of congestion, as packets probabilistically arrive at non-overlapping times. On the one hand, a transport that operates above this point can experience congestion loss and could also trigger operator activities, such as those discussed in [RFC6057]. On the other hand, a flow may achieve both near-maximum throughput and low latency when it operates close to this knee point, with minimal contribution to router congestion. Choice of an appropriate rate/congestion window can therefore significantly impact the loss and delay experienced by a flow and will impact other flows that share a common network queue.

Some applications may send data at a lower rate or keep less segments outstanding at any given time. Examples include multimedia codecs that stream at some natural rate (or set of rates) or an application that is naturally interactive (e.g., some web applications, interactive server-based gaming, transaction-based protocols). Such applications may have different objectives. They may not wish to maximize throughput, but may desire a lower loss rate or bounded delay.

The correct operation of an AQM-enabled network device **MUST NOT** rely upon specific transport responses to congestion signals.

#### 4.7. The Need for Further Research

The second recommendation of [RFC2309] called for further research into the interaction between network queues and host applications, and the means of signaling between them. This research has occurred, and we as a community have learned a lot. However, we are not done.

We have learned that the problems of congestion, latency, and buffer-sizing have not gone away and are becoming more important to many users. A number of self-tuning AQM algorithms have been found that offer significant advantages for deployed networks. There is also renewed interest in deploying AQM and the potential of ECN.

Traffic patterns can depend on the network deployment scenario, and Internet research therefore needs to consider the implications of a diverse range of application interactions. This includes ensuring

RFC 7567

Active Queue Management Recommendations

July 2015

that combinations of mechanisms, as well as combinations of traffic patterns, do not interact and result in either significantly reduced flow throughput or significantly increased latency.

At the time of writing (in 2015), an obvious example of further research is the need to consider the many-to-one communication patterns found in data centers, known as incast [Ren12], (e.g., produced by Map/Reduce applications). Such analysis needs to study not only each application traffic type but also combinations of types of traffic.

Research also needs to consider the need to extend our taxonomy of transport sessions to include not only "mice" and "elephants", but "lemmings". Here, "lemmings" are flash crowds of "mice" that the network inadvertently tries to signal to as if they were "elephant" flows, resulting in head-of-line blocking in a data center deployment scenario.

Examples of other required research include:

- o new AQM and scheduling algorithms
- o appropriate use of delay-based methods and the implications of AQM
- o suitable algorithms for marking ECN-capable packets that do not require operational configuration or tuning for common use
- o experience in the deployment of ECN alongside AQM
- o tools for enabling AQM (and ECN) deployment and measuring the performance
- o methods for mitigating the impact of non-conformant and malicious flows
- o implications on applications of using new network and transport methods

Hence, this document reiterates the call of RFC 2309: we need continuing research as applications develop.

## 5. Security Considerations

While security is a very important issue, it is largely orthogonal to the performance issues discussed in this memo.

This recommendation requires algorithms to be independent of specific transport or application behaviors. Therefore, a network device does not require visibility or access to upper-layer protocol information to implement an AQM algorithm. This ability to operate in an application-agnostic fashion is an example of a privacy-enhancing feature.

Many deployed network devices use queueing methods that allow unresponsive traffic to capture network capacity, denying access to other traffic flows. This could potentially be used as a denial-of-service attack. This threat could be reduced in network devices that deploy AQM or some form of scheduling. We note, however, that a denial-of-service attack that results in unresponsive traffic flows may be indistinguishable from other traffic flows (e.g., tunnels carrying aggregates of short flows, high-rate isochronous applications). New methods therefore may remain vulnerable, and this document recommends that ongoing research consider ways to mitigate such attacks.

## 6. Privacy Considerations

This document, by itself, presents no new privacy issues.

## 7. References

### 7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<http://www.rfc-editor.org/info/rfc3168>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<http://www.rfc-editor.org/info/rfc4301>>.

RFC 7567                      Active Queue Management Recommendations                      July 2015

- [RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", BCP 124, RFC 4774, DOI 10.17487/RFC4774, November 2006, <<http://www.rfc-editor.org/info/rfc4774>>.
- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, DOI 10.17487/RFC5405, November 2008, <<http://www.rfc-editor.org/info/rfc5405>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<http://www.rfc-editor.org/info/rfc5681>>.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<http://www.rfc-editor.org/info/rfc6040>>.
- [RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August 2012, <<http://www.rfc-editor.org/info/rfc6679>>.
- [RFC7141] Briscoe, B. and J. Manner, "Byte and Packet Congestion Notification", BCP 41, RFC 7141, DOI 10.17487/RFC7141, February 2014, <<http://www.rfc-editor.org/info/rfc7141>>.

## 7.2. Informative References

- [AQM-WG] IETF, "Active Queue Management and Packet Scheduling (aqm) WG", <<http://datatracker.ietf.org/wg/aqm/charter/>>.
- [Bri15] Briscoe, B., Brunstrom, A., Petlund, A., Hayes, D., Ros, D., Tsang, I., Gjessing, S., Fairhurst, G., Griwodz, C., and M. Welzl, "Reducing Internet Latency: A Survey of Techniques and their Merit", IEEE Communications Surveys & Tutorials, 2015.
- [Choi04] Choi, B., Moon, S., Zhang, Z., Papagiannaki, K., and C. Diot, "Analysis of Point-To-Point Packet Delay In an Operational Network", March 2004.
- [CONEX] Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism and Requirements", Work in Progress, draft-ietf-conex-abstract-mech-13, October 2014.

RFC 7567

Active Queue Management Recommendations

July 2015

- [Dem90] Demers, A., Keshav, S., and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm, Internetworking: Research and Experience", SIGCOMM Symposium proceedings on Communications architectures and protocols, 1990.
- [ECN-Benefit] Fairhurst, G. and M. Welzl, "The Benefits of using Explicit Congestion Notification (ECN)", Work in Progress, draft-ietf-aqm-ecn-benefits-05, June 2015.
- [Flo92] Floyd, S. and V. Jacobsen, "On Traffic Phase Effects in Packet-Switched Gateways", 1992, <<http://www.icir.org/floyd/papers/phase.pdf>>.
- [Flo94] Floyd, S. and V. Jacobsen, "The Synchronization of Periodic Routing Messages", 1994, <[http://ee.lbl.gov/papers/sync\\_94.pdf](http://ee.lbl.gov/papers/sync_94.pdf)>.
- [Floyd91] Floyd, S., "Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic.", Computer Communications Review , October 1991.
- [Floyd95] Floyd, S. and V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks", IEEE/ACM Transactions on Networking, August 1995.
- [Jacobson88] Jacobson, V., "Congestion Avoidance and Control", SIGCOMM Symposium proceedings on Communications architectures and protocols, August 1988.
- [Jain94] Jain, R., Ramakrishnan, KK., and C. Dah-Ming, "Congestion avoidance scheme for computer networks", US Patent Office 5377327, December 1994.
- [Lakshman96] Lakshman, TV., Neidhardt, A., and T. Ott, "The Drop From Front Strategy in TCP Over ATM and Its Interworking with Other Control Features", IEEE Infocomm, 1996.
- [Leland94] Leland, W., Taqqu, M., Willinger, W., and D. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)", IEEE/ACM Transactions on Networking, February 1994.

RFC 7567                      Active Queue Management Recommendations                      July 2015

- [McK90]      McKenney, PE. and G. Varghese, "Stochastic Fairness Queuing", 1990,  
<<http://www2.rdrop.com/~paulmck/scalability/paper/sfq.2002.06.04.pdf>>.
- [Nic12]      Nichols, K. and V. Jacobson, "Controlling Queue Delay", Communications of the ACM, Vol. 55, Issue 7, pp. 42-50, July 2012.
- [Ren12]      Ren, Y., Zhao, Y., and P. Liu, "A survey on TCP Incast in data center networks", International Journal of Communication Systems, Volumes 27, Issue 8, pages 116-117, 1990.
- [RFC768]      Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980,  
<<http://www.rfc-editor.org/info/rfc768>>.
- [RFC791]      Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981,  
<<http://www.rfc-editor.org/info/rfc791>>.
- [RFC793]      Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981,  
<<http://www.rfc-editor.org/info/rfc793>>.
- [RFC896]      Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984,  
<<http://www.rfc-editor.org/info/rfc896>>.
- [RFC970]      Nagle, J., "On Packet Switches With Infinite Storage", RFC 970, DOI 10.17487/RFC0970, December 1985,  
<<http://www.rfc-editor.org/info/rfc970>>.
- [RFC1122]      Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989,  
<<http://www.rfc-editor.org/info/rfc1122>>.
- [RFC1633]      Braden, R., Clark, D., and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC 1633, DOI 10.17487/RFC1633, June 1994,  
<<http://www.rfc-editor.org/info/rfc1633>>.



RFC 7567                      Active Queue Management Recommendations                      July 2015

- [RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, DOI 10.17487/RFC2309, April 1998, <<http://www.rfc-editor.org/info/rfc2309>>.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <<http://www.rfc-editor.org/info/rfc2460>>.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <<http://www.rfc-editor.org/info/rfc2474>>.
- [RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <<http://www.rfc-editor.org/info/rfc2475>>.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <<http://www.rfc-editor.org/info/rfc2914>>.
- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <<http://www.rfc-editor.org/info/rfc4340>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<http://www.rfc-editor.org/info/rfc4960>>.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<http://www.rfc-editor.org/info/rfc5348>>.
- [RFC5559] Eardley, P., Ed., "Pre-Congestion Notification (PCN) Architecture", RFC 5559, DOI 10.17487/RFC5559, June 2009, <<http://www.rfc-editor.org/info/rfc5559>>.

RFC 7567                      Active Queue Management Recommendations                      July 2015

- [RFC6057] Bastian, C., Klieber, T., Livingood, J., Mills, J., and R. Woundy, "Comcast's Protocol-Agnostic Congestion Management System", RFC 6057, DOI 10.17487/RFC6057, December 2010, <<http://www.rfc-editor.org/info/rfc6057>>.
- [RFC6789] Briscoe, B., Ed., Woundy, R., Ed., and A. Cooper, Ed., "Congestion Exposure (ConEx) Concepts and Use Cases", RFC 6789, DOI 10.17487/RFC6789, December 2012, <<http://www.rfc-editor.org/info/rfc6789>>.
- [RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LED BAT)", RFC 6817, DOI 10.17487/RFC6817, December 2012, <<http://www.rfc-editor.org/info/rfc6817>>.
- [RFC7414] Duke, M., Braden, R., Eddy, W., Blanton, E., and A. Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents", RFC 7414, DOI 10.17487/RFC7414, February 2015, <<http://www.rfc-editor.org/info/rfc7414>>.
- [Shr96] Shreedhar, M. and G. Varghese, "Efficient Fair Queueing Using Deficit Round Robin", IEEE/ACM Transactions on Networking, Vol. 4, No. 3, July 1996.
- [Sto97] Stoica, I. and H. Zhang, "A Hierarchical Fair Service Curve algorithm for Link sharing, real-time and priority services", ACM SIGCOMM, 1997.
- [Sut99] Suter, B., "Buffer Management Schemes for Supporting TCP in Gigabit Routers with Per-flow Queueing", IEEE Journal on Selected Areas in Communications, Vol. 17, Issue 6, pp. 1159-1169, June 1999.
- [Willinger95] Willinger, W., Taqqu, M., Sherman, R., Wilson, D., and V. Jacobson, "Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level", SIGCOMM Symposium proceedings on Communications architectures and protocols, August 1995.
- [Zha90] Zhang, L. and D. Clark, "Oscillating Behavior of Network Traffic: A Case Study Simulation", 1990, <<http://groups.csail.mit.edu/ana/Publications/Zhang-DDC-Oscillating-Behavior-of-Network-Traffic-1990.pdf>>.

RFC 7567

Active Queue Management Recommendations

July 2015

#### Acknowledgements

The original draft of this document describing best current practice was based on [RFC2309], an Informational RFC. It was written by the End-to-End Research Group, which is to say Bob Braden, Dave Clark, Jon Crowcroft, Bruce Davie, Steve Deering, Deborah Estrin, Sally Floyd, Van Jacobson, Greg Minshall, Craig Partridge, Larry Peterson, KK Ramakrishnan, Scott Shenker, John Wroclawski, and Lixia Zhang. Although there are important differences, many of the key arguments in the present document remain unchanged from those in RFC 2309.

The need for an updated document was agreed to in the TSV area meeting at IETF 86. This document was reviewed on the [aqm@ietf.org](mailto:aqm@ietf.org) list. Comments were received from Colin Perkins, Richard Scheffenegger, Dave Taht, John Leslie, David Collier-Brown, and many others.

Gorry Fairhurst was in part supported by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

#### Authors' Addresses

Fred Baker (editor)  
Cisco Systems  
Santa Barbara, California 93117  
United States

Email: [fred@cisco.com](mailto:fred@cisco.com)

Godred Fairhurst (editor)  
University of Aberdeen  
School of Engineering  
Fraser Noble Building  
Aberdeen, Scotland AB24 3UE  
United Kingdom

Email: [gorry@erg.abdn.ac.uk](mailto:gorry@erg.abdn.ac.uk)  
URI: <http://www.erg.abdn.ac.uk>

Internet Engineering Task Force  
Internet-Draft  
Intended status: Informational  
Expires: December 21, 2015

N. Kuhn, Ed.  
Telecom Bretagne  
N. Khademi, Ed.  
University of Oslo  
P. Natarajan, Ed.  
Cisco Systems  
D. Ros  
Simula Research Laboratory AS  
June 19, 2015

AQM Characterization Guidelines  
draft-ietf-aqm-eval-guidelines-04

Abstract

Unmanaged large buffers in today's networks have given rise to a slew of performance issues. These performance issues can be addressed by some form of Active Queue Management (AQM) mechanism, optionally in combination with a packet scheduling scheme such as fair queuing. The IETF Active Queue Management and Packet Scheduling working group was formed to standardize AQM schemes that are robust, easily implementable, and successfully deployable in today's networks. This document describes various criteria for performing precautionary characterizations of AQM proposals. This document also helps in ascertaining whether any given AQM proposal should be taken up for standardization by the AQM WG.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 21, 2015.

Internet-Draft

AQM Characterization Guidelines

June 2015

## Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4.e](#) of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1.	Introduction . . . . .	4
1.1.	Reducing the latency and maximizing the goodput . . . . .	5
1.2.	Guidelines for AQM evaluation . . . . .	5
1.3.	Requirements Language . . . . .	6
1.4.	Glossary . . . . .	6
2.	End-to-end metrics . . . . .	6
2.1.	Flow completion time . . . . .	7
2.2.	Flow start up time . . . . .	7
2.3.	Packet loss . . . . .	7
2.4.	Packet loss synchronization . . . . .	8
2.5.	Goodput . . . . .	8
2.6.	Latency and jitter . . . . .	9
2.7.	Discussion on the trade-off between latency and goodput . . . . .	9
3.	Generic set up for evaluations . . . . .	10
3.1.	Topology and notations . . . . .	10
3.2.	Buffer size . . . . .	12
3.3.	Congestion controls . . . . .	12
4.	Transport Protocols . . . . .	13
4.1.	TCP-friendly sender . . . . .	13
4.1.1.	TCP-friendly sender with the same initial congestion window . . . . .	14
4.1.2.	TCP-friendly sender with different initial congestion windows . . . . .	14
4.2.	Aggressive transport sender . . . . .	14
4.3.	Unresponsive transport sender . . . . .	15
4.4.	Less-than Best Effort transport sender . . . . .	15
5.	Round Trip Time Fairness . . . . .	16
5.1.	Motivation . . . . .	16
5.2.	Recommended tests . . . . .	16
5.3.	Metrics to evaluate the RTT fairness . . . . .	17
6.	Burst Absorption . . . . .	17

Internet-Draft

AQM Characterization Guidelines

June 2015

6.1.	Motivation . . . . .	17
6.2.	Recommended tests . . . . .	18
7.	Stability . . . . .	19
7.1.	Motivation . . . . .	19
7.2.	Recommended tests . . . . .	19
7.2.1.	Definition of the congestion Level . . . . .	19
7.2.2.	Mild congestion . . . . .	20
7.2.3.	Medium congestion . . . . .	20
7.2.4.	Heavy congestion . . . . .	20
7.2.5.	Varying congestion levels . . . . .	20
7.2.6.	Varying available capacity . . . . .	21
7.3.	Parameter sensitivity and stability analysis . . . . .	22
8.	Various Traffic Profiles . . . . .	22
8.1.	Traffic mix . . . . .	22
8.2.	Bi-directional traffic . . . . .	23
9.	Multi-AQM Scenario . . . . .	23
9.1.	Motivation . . . . .	23
9.2.	Details on the evaluation scenario . . . . .	24
10.	Implementation cost . . . . .	24
10.1.	Motivation . . . . .	24
10.2.	Recommended discussion . . . . .	25
11.	Operator Control and Auto-tuning . . . . .	25
11.1.	Motivation . . . . .	25
11.2.	Required discussion . . . . .	26
12.	Interaction with ECN . . . . .	26
12.1.	Motivation . . . . .	26
12.2.	Recommended discussion . . . . .	26
13.	Interaction with Scheduling . . . . .	26
13.1.	Motivation . . . . .	27
13.2.	Recommended discussion . . . . .	27
13.3.	Assessing the interaction between AQM and scheduling . . . . .	27
14.	Discussion on Methodology, Metrics, AQM Comparisons and Packet Sizes . . . . .	27
14.1.	Methodology . . . . .	27
14.2.	Comments on metrics measurement . . . . .	28
14.3.	Comparing AQM schemes . . . . .	28
14.3.1.	Performance comparison . . . . .	28
14.3.2.	Deployment comparison . . . . .	29
14.4.	Packet sizes and congestion notification . . . . .	29
15.	Conclusion . . . . .	30
16.	Acknowledgements . . . . .	31
17.	Contributors . . . . .	31
18.	IANA Considerations . . . . .	32
19.	Security Considerations . . . . .	32
20.	References . . . . .	32
20.1.	Normative References . . . . .	32
20.2.	Informative References . . . . .	32
	Authors' Addresses . . . . .	34

## 1. Introduction

Active Queue Management (AQM) [[I-D.ietf-aqm-recommendation](#)] addresses the concerns arising from using unnecessarily large and unmanaged buffers to improve network and application performance. Several AQM algorithms have been proposed in the past years, most notably Random Early Detection (RED), BLUE, and Proportional Integral controller (PI), and more recently CoDel [[CODEL](#)] and PIE [[PIE](#)]. In general, these algorithms actively interact with the Transmission Control Protocol (TCP) and any other transport protocol that deploys a congestion control scheme to manage the amount of data they keep in the network. The available buffer space in the routers and switches should be large enough to accommodate the short-term buffering requirements. AQM schemes aim at reducing mean buffer occupancy, and therefore both end-to-end delay and jitter. Some of these algorithms, notably RED, have also been widely implemented in some network devices. However, the potential benefits of the RED scheme have not been realized since RED is reported to be usually turned off. The main reason of this reluctance to use RED in today's deployments comes from its sensitivity to the operating conditions in the network and the difficulty of tuning its parameters.

A buffer is a physical volume of memory in which a queue or set of queues are stored. In real implementations of switches, a global memory is shared between the available devices: the size of the buffer for a given communication does not make sense, as its dedicated memory may vary over the time and real-world buffering architectures are complex. For the sake of simplicity, when speaking of a specific queue in this document, "buffer size" refers to the maximum amount of data the buffer may store, which can be measured in bytes or packets. The rest of this memo therefore refers to the maximum queue depth as the size of the buffer for a given communication.

Bufferbloat [[BB2011](#)] is the consequence of deploying large unmanaged buffers on the Internet, which has lead to an increase in end-to-end delay: the buffering has often been measured to be ten times or hundred times larger than needed. This results in poor performance for latency-sensitive applications such as real-time multimedia (e.g., voice, video, gaming, etc). The degree to which this affects modern networking equipment, especially consumer-grade equipment's, produces problems even with commonly used web services. Active queue management is thus essential to control queuing delay and decrease network latency.

The Active Queue Management and Packet Scheduling Working Group (AQM WG) was recently formed within the TSV area to address the problems with large unmanaged buffers in the Internet. Specifically, the AQM

Internet-Draft

AQM Characterization Guidelines

June 2015

WG is tasked with standardizing AQM schemes that not only address concerns with such buffers, but also are robust under a wide variety of operating conditions.

In order to ascertain whether the WG should undertake standardizing an AQM proposal, the WG requires guidelines for assessing AQM proposals. This document provides the necessary characterization guidelines. There may be a debate on whether a scheduling scheme is additional to an AQM algorithm or is a part of an AQM algorithm. The rest of this memo refers to AQM as a dropping/marking policy that does not feature a scheduling scheme. This document may be complemented with another one on guidelines for assessing combination of packet scheduling and AQM. We note that such a document will inherit all the guidelines from this document plus any additional scenarios relevant for packet scheduling such as flow starvation evaluation or impact of the number of hash buckets.

#### 1.1. Reducing the latency and maximizing the goodput

The trade-off between reducing the latency and maximizing the goodput is intrinsically linked to each AQM scheme and is key to evaluating its performance. This trade-off MUST be considered in various scenarios to ensure the safety of an AQM deployment. Whenever possible, solutions ought to aim at both maximizing goodput and minimizing latency. This document provides guidelines that enable the reader to quantify (1) reduction of latency, (2) maximization of goodput and (3) the trade-off between the two.

These guidelines provide the tools to understand the deployment costs versus the potential gain in performance from the introduction of the proposed scheme.

#### 1.2. Guidelines for AQM evaluation

The guidelines help to quantify performance of AQM schemes in terms of latency reduction, goodput maximization and the trade-off between these two. The guidelines also help to discuss safe deployment of AQM, including self-adaptation, stability analysis, fairness, design and implementation complexity and robustness to different operating conditions.

This memo details generic characterization scenarios against which any AQM proposal must be evaluated, irrespective of whether or not an AQM is standardized by the IETF. This documents recommends the relevant scenarios and metrics to be considered.

The document presents central aspects of an AQM algorithm that must be considered whatever the context, such as burst absorption



capacity, RTT fairness or resilience to fluctuating network conditions. These guidelines do not cover every possible aspect of a particular algorithm. In addition, it is worth noting that the proposed criteria are not bound to a particular evaluation toolset.

This document details how an AQM designer can rate the feasibility of their proposal in different types of network devices (switches, routers, firewalls, hosts, drivers, etc) where an AQM may be implemented. However, these guidelines do not present context-dependent scenarios (such as 802.11 WLANs, data-centers or rural broadband networks).

### 1.3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [RFC2119].

### 1.4. Glossary

- o AQM: there may be a debate on whether a scheduling scheme is additional to an AQM algorithm or is a part of an AQM algorithm. The rest of this memo refers to AQM as a dropping/marketing policy that does not feature a scheduling scheme.
- o buffer: a physical volume of memory in which a queue or set of queues are stored.
- o buffer size: the maximum amount of data that may be stored in a buffer, measured in bytes or packets.

## 2. End-to-end metrics

End-to-end delay is the result of propagation delay, serialization delay, service delay in a switch, medium-access delay and queuing delay, summed over the network elements along the path. AQM schemes may reduce the queuing delay by providing signals to the sender on the emergence of congestion, but any impact on the goodput must be carefully considered. This section presents the metrics that could be used to better quantify (1) the reduction of latency, (2) maximization of goodput and (3) the trade-off between these two. This section provides normative requirements for metrics that can be used to assess the performance of an AQM scheme.

Some metrics listed in this section are not suited to every type of traffic detailed in the rest of this document. It is therefore not necessary to measure all of the following metrics: the chosen metric may not be relevant to the context of the evaluation scenario (e.g.

latency vs. goodput trade-off in application-limited traffic scenarios). Guidance is provided for each metric.

### 2.1. Flow completion time

The flow completion time is an important performance metric for the end-user when the flow size is finite. Considering the fact that an AQM scheme may drop/mark packets, the flow completion time is directly linked to the dropping/marketing policy of the AQM scheme. This metric helps to better assess the performance of an AQM depending on the flow size. The Flow Completion Time (FCT) is related to the flow size (Fs) and the goodput for the flow (G) as follows:

$$\text{FCT [s]} = \text{Fs [B]} / ( \text{G [Mbps]} / 8 )$$

If this metric is used to evaluate the performance of web transfers, we propose to rather consider the time needed to download all the objects that compose the web page, as this makes more sense in terms of user experience than assessing the time needed to download each object.

### 2.2. Flow start up time

The flow start up time is the time between the request has been sent from the client and the server starts to transmit data. The amount of packets dropped by an AQM may seriously affect the waiting period during which the data transfer has not started. This metric would specifically focus on the operations such as DNS lookups, TCP opens of SSL handshakes.

### 2.3. Packet loss

Packet loss can occur within a network device, this can impact the end-to-end performance measured at receiver.

The tester SHOULD evaluate loss experienced at the receiver using one of the two metrics:

- o the packet loss probability: this metric is to be frequently measured during the experiment. The long-term loss probability is of interest for steady-state scenarios only;
- o the interval between consecutive losses: the time between two losses is to be measured.

The packet loss probability can be assessed by simply evaluating the loss ratio as a function of the number of lost packets and the total

number of packets sent. This might not be easily done in laboratory testing, for which these guidelines advice the tester:

- o to check that for every packet, a corresponding packet was received within a reasonable time, as explained in [\[RFC2680\]](#).
- o to keep a count of all packets sent, and a count of the non-duplicate packets received, as explained in the [section 10 of \[RFC2544\]](#).

The interval between consecutive losses, which is also called a gap, is a metric of interest for VoIP traffic and, as a result, has been further specified in [\[RFC3611\]](#).

#### 2.4. Packet loss synchronization

One goal of an AQM algorithm ought be to help to avoid global synchronization of flows sharing a bottleneck buffer on which the AQM operates ([\[RFC2309\]](#),[\[I-D.ietf-aqm-recommendation\]](#)). The "degree" of packet-loss synchronization between flows SHOULD be assessed, with and without the AQM under consideration.

As discussed e.g. in [\[LOSS-SYNCH-MET-08\]](#), loss synchronization among flows may be quantified by several slightly different metrics that capture different aspects of the same issue. However, in real-world measurements the choice of metric could be imposed by practical considerations -- e.g. whether fine-grained information on packet losses in the bottleneck available or not. For the purpose of AQM characterization, a good candidate metric is the global synchronization ratio, measuring the proportion of flows losing packets during a loss event. [\[YU06\]](#) used this metric in real-world experiments to characterize synchronization along arbitrary Internet paths; the full methodology is described in [\[YU06\]](#).

If an AQM scheme is evaluated using real-life network environments, it is worth pointing out that some network events, such as failed link restoration may cause synchronized losses between active flows and thus confuse the meaning of this metric.

#### 2.5. Goodput

The goodput has been defined in the [section 3.17 of \[RFC2647\]](#) as the number of bits per unit of time forwarded to the correct destination interface of the Device Under Test (DUT) or the System Under Test (SUT), minus any bits lost or retransmitted. This definition induces that the test setup needs to be qualified to assure that it is not generating losses on its own.

Measuring the end-to-end goodput provides an appreciation of how well an AQM scheme improves transport and application performance. The measured end-to-end goodput is linked to the dropping/marking policy of the AQM scheme -- e.g. the fewer the number of packet drops, the fewer packets need retransmission, minimizing the impact of AQM on transport and application performance. Additionally, an AQM scheme may resort to Explicit Congestion Notification (ECN) marking as an initial means to control delay. Again, marking packets instead of dropping them reduces the number of packet retransmissions and increases goodput. End-to-end goodput values help to evaluate the AQM scheme's effectiveness of an AQM scheme in minimizing packet drops that impact application performance and to estimate how well the AQM scheme works with ECN.

The measurement of the goodput allows the tester evaluate to which extent an AQM is able to maintain a high bottleneck utilization. This metric should be also obtained frequently during an experiment as the long-term goodput is relevant for steady-state scenarios only and may not necessarily reflect how the introduction of an AQM actually impacts the link utilization during at a certain period of time. Fluctuations in the values obtained from these measurements may depend on other factors than the introduction of an AQM, such as link layer losses due to external noise or corruption, fluctuating bandwidths (802.11 WLANs), heavy congestion levels or transport layer's rate reduction by congestion control mechanism.

#### 2.6. Latency and jitter

The latency, or the one-way delay metric, is discussed in [\[RFC2679\]](#). There is a consensus on a adequate metric for the jitter, that represents the one-way delay variations for packets from the same flow: the Packet Delay Variation (PDV), detailed in [\[RFC5481\]](#), serves well all use cases.

The end-to-end latency differs from the queuing delay: it is linked to the network topology and the path characteristics. Moreover, the jitter also strongly depends on the traffic pattern and the topology. The introduction of an AQM scheme would impact these metrics and therefore they should be considered in the end-to-end evaluation of performance.

#### 2.7. Discussion on the trade-off between latency and goodput

The metrics presented in this section may be considered as explained in the rest of this document, in order to discuss and quantify the trade-off between latency and goodput.

Internet-Draft

AQM Characterization Guidelines

June 2015

This trade-off can also be illustrated with figures following the recommendations of section 5 of [TCPEVAL2013]. Each of the end-to-end delay and the goodput SHOULD be measured frequently for every fixed time interval.

With regards to the goodput, and in addition to the long-term stationary goodput value, it is RECOMMENDED to take measurements every multiple of RTTs. We suggest a minimum value of 10 x RTT (to smooth out the fluctuations) but higher values are encouraged whenever appropriate for the presentation depending on the network's path characteristics. The measurement period MUST be disclosed for each experiment and when results/values are compared across different AQM schemes, the comparisons SHOULD use exactly the same measurement periods.

With regards to latency, it is highly RECOMMENDED to take the samples on per-packet basis whenever possible depending on the features provided by hardware/software and the impact of sampling itself on the hardware performance. It is generally RECOMMENDED to provide at least 10 samples per RTT.

From each of these sets of measurements, the 10th and 90th percentiles and the median value SHOULD be computed. For each scenario, a graph can be generated, with the x-axis showing the end-to-end delay and the y-axis the goodput. This graph provides part of a better understanding of (1) the delay/goodput trade-off for a given congestion control mechanism, and (2) how the goodput and average queue size vary as a function of the traffic load.

### 3. Generic set up for evaluations

This section presents the topology that can be used for each of the following scenarios, the corresponding notations and discusses various assumptions that have been made in the document.

#### 3.1. Topology and notations

Internet-Draft

AQM Characterization Guidelines

June 2015

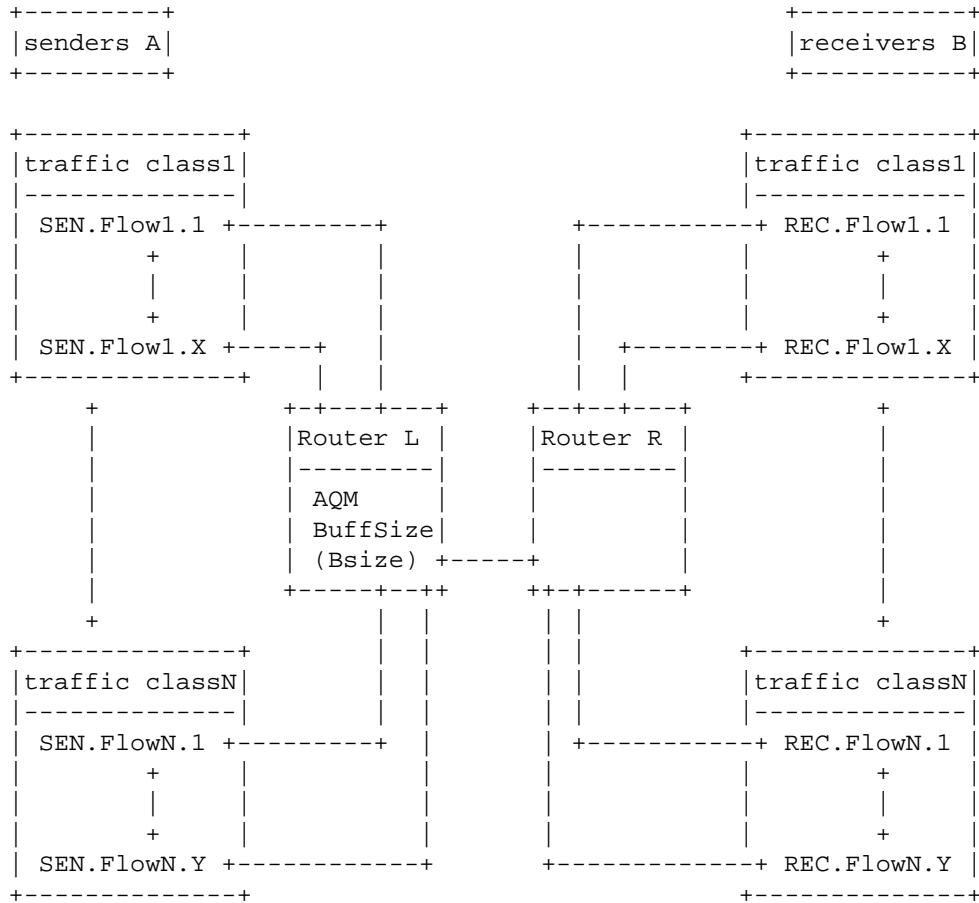


Figure 1: Topology and notations

Figure 1 is a generic topology where:

- o various classes of traffic can be introduced;
- o the timing of each flow could be different (i.e., when does each flow start and stop);
- o each class of traffic can comprise various number of flows;
- o each link is characterized by a couple (RTT, capacity);
- o flows are generated between A and B, sharing a bottleneck (Routers L and R);

- o the tester SHOULD consider both scenarios of asymmetric and symmetric bottleneck links in terms of bandwidth. In case of asymmetric link, the capacity from senders to receivers is higher than the one from receivers to senders; the symmetric link scenario provides a basic understanding of the operation of the AQM mechanism whereas the asymmetric link scenario evaluates an AQM mechanism in a more realistic setup;
- o in asymmetric link scenarios, the tester SHOULD study the bi-directional traffic between A and B (downlink and uplink) with the AQM mechanism deployed on one direction only. The tester MAY additionally consider a scenario with AQM mechanism being deployed on both directions. In each scenario, the tester SHOULD investigate the impact of drop policy of the AQM on TCP ACK packets and its impact on the performance.

Although this topology may not perfectly reflect actual topologies, the simple topology is commonly used in the world of simulations and small testbeds. It can be considered as adequate to evaluate AQM proposals, similarly to the topology proposed in [TCPEVAL2013]. Testers ought to pay attention to the topology that has been used to evaluate an AQM scheme when comparing this scheme with a new proposed AQM scheme.

### 3.2. Buffer size

The size of the buffers should be carefully chosen, and is to be set to the bandwidth-delay product; the bandwidth being the bottleneck capacity and the delay the larger RTT in the considered network. The size of the buffer can impact on the AQM performance and is a dimensioning parameter that will be considered when comparing AQM proposals.

If the context or the application requires a specific buffer size, the tester MUST justify and detail the way the maximum queue size is set. Indeed, the maximum size of the buffer may affect the AQM's performance and its choice SHOULD be elaborated for a fair comparison between AQM proposals. While comparing AQM schemes the buffer size SHOULD remain the same across the tests.

### 3.3. Congestion controls

This memo features three kind of congestion controls:

- o Standard TCP congestion control: the base-line congestion control is TCP NewReno with SACK, as explained in [RFC5681].

- o Aggressive congestion controls: a base-line congestion control for this category is TCP Cubic.
- o Less-than Best Effort (LBE) congestion controls: an LBE congestion control 'results in smaller bandwidth and/or delay impact on standard TCP than standard TCP itself, when sharing a bottleneck with it.' [[RFC6297](#)]

Other transport congestion controls can OPTIONALLY be evaluated in addition. Recent transport layer protocols are not mentioned in the following sections, for the sake of simplicity.

#### 4. Transport Protocols

Network and end-devices need to be configured with a reasonable amount of buffer space to absorb transient bursts. In some situations, network providers tend to configure devices with large buffers to avoid packet drops triggered by a full buffer and to maximize the link utilization for standard loss-based TCP traffic.

AQM algorithms are often evaluated by considering Transmission Control Protocol (TCP) [[RFC0793](#)] with a limited number of applications. TCP is a widely deployed transport. It fills up unmanaged buffers until the TCP sender receives a signal (packet drop) that reduces the sending rate. The larger the buffer, the higher the buffer occupancy, and therefore the queuing delay. An efficient AQM scheme sends out early congestion signals to TCP to bring the queuing delay under control.

Not all applications using TCP use the same flavor of TCP. Variety of senders generate different classes of traffic which may not react to congestion signals (aka non-responsive flows [[I-D.ietf-aqm-recommendation](#)]) or may not reduce their sending rate as expected (aka Transport Flows that are less responsive than TCP[[I-D.ietf-aqm-recommendation](#)], also called "aggressive flows"). In these cases, AQM schemes seek to control the queuing delay.

This section provides guidelines to assess the performance of an AQM proposal for various traffic profiles -- different types of senders (with different TCP congestion control variants, unresponsive, aggressive).

##### 4.1. TCP-friendly sender



#### 4.1.1. TCP-friendly sender with the same initial congestion window

This scenario helps to evaluate how an AQM scheme reacts to a TCP-friendly transport sender. A single long-lived, non application-limited, TCP NewReno flow, with an Initial congestion Window (IW) set to 3 packets, transfers data between sender A and receiver B. Other TCP friendly congestion control schemes such as TCP-friendly rate control [RFC5348] etc MAY also be considered.

For each TCP-friendly transport considered, the graph described in [Section 2.7](#) could be generated.

#### 4.1.2. TCP-friendly sender with different initial congestion windows

This scenario can be used to evaluate how an AQM scheme adapts to a traffic mix consisting of TCP flows with different values of the IW.

For this scenario, two types of flows MUST be generated between sender A and receiver B:

- o A single long-lived non application-limited TCP NewReno flow;
- o A single long-lived application-limited TCP NewReno flow, with an IW set to 3 or 10 packets. The size of the data transferred must be strictly higher than 10 packets and should be lower than 100 packets.

The transmission of the non application-limited flow must start before the transmission of the application-limited flow and only after the steady state has been reached by non application-limited flow.

For each of these scenarios, the graph described in [Section 2.7](#) could be generated for each class of traffic (application-limited and non application-limited). The completion time of the application-limited TCP flow could be measured.

#### 4.2. Aggressive transport sender

This scenario helps testers to evaluate how an AQM scheme reacts to a transport sender that is more aggressive than a single TCP-friendly sender. We define 'aggressiveness' as a higher increase factor than standard upon a successful transmission and/or a lower than standard decrease factor upon a unsuccessful transmission (e.g. in case of congestion controls with Additive-Increase Multiplicative-Decrease (AIMD) principle, a larger AI and/or MD factors). A single long-lived, non application-limited, TCP Cubic flow transfers data between

sender A and receiver B. Other aggressive congestion control schemes MAY also be considered.

For each flavor of aggressive transports, the graph described in [Section 2.7](#) could be generated.

#### 4.3. Unresponsive transport sender

This scenario helps testers to evaluate how an AQM scheme reacts to a transport sender that is less responsive than TCP. Note that faulty transport implementations on an end host and/or faulty network elements en-route that "hide" congestion signals in packet headers [[I-D.ietf-aqm-recommendation](#)] may also lead to a similar situation, such that the AQM scheme needs to adapt to unresponsive traffic. To this end, these guidelines propose the two following scenarios.

The first scenario can be used to evaluate queue build up. It considers unresponsive flow(s) whose sending rate is greater than the bottleneck link capacity between routers L and R. This scenario consists of a long-lived non application limited UDP flow transmits data between sender A and receiver B. Graphs described in [Section 2.7](#) could be generated.

The second scenario can be used to evaluate if the AQM scheme is able to keep responsive fraction under control. This scenario considers a mixture of TCP-friendly and unresponsive traffics. It consists of a long-lived non application-limited UDP flow and a single long-lived, non-application-limited, TCP New Reno flow that transmit data between sender A and receiver B. As opposed to the first scenario, the rate of the UDP traffic should not be greater than the bottleneck capacity, and should not be higher than half of the bottleneck capacity. For each type of traffic, the graph described in [Section 2.7](#) could be generated.

#### 4.4. Less-than Best Effort transport sender

This scenario helps to evaluate how an AQM scheme reacts to LBE congestion controls that 'results in smaller bandwidth and/or delay impact on standard TCP than standard TCP itself, when sharing a bottleneck with it.' [[RFC6297](#)]. The potential fateful interaction when AQM and LBE techniques are combined has been shown in [[LBE-AQM](#)]; this scenario helps to evaluate whether the coexistence of the proposed AQM and LBE techniques may be possible.

Single long-lived non application-limited TCP NewReno flows transfer data between sender A and receiver B. Other TCP-friendly congestion control schemes MAY also be considered. Single long-lived non application-limited LEDBAT [[RFC6817](#)] flows transfer data between

sender A and receiver B. We recommend to set the target delay and gain values of LEDBAT respectively to 5 ms and 10 [[LEDBAT-PARAM](#)]. Other LBE congestion control schemes, any of those listed in [[RFC6297](#)], MAY also be considered.

For each of the TCP-friendly and LBE transports, the graph described in [Section 2.7](#) could be generated.

## 5. Round Trip Time Fairness

### 5.1. Motivation

The ability of AQM schemes to control the queuing delay highly depends on the way end-to-end protocols react to congestion signals. When the RTT varies, the behaviour of a congestion control is impacted and this impacts the ability of an AQM scheme to control the queue. It is therefore important to assess the AQM schemes for a set of RTTs (e.g., from 5 ms to 200 ms).

The asymmetry in terms of difference in intrinsic RTT between various paths sharing the same bottleneck SHOULD be considered so that the fairness between the flows can be discussed since in this scenario, a flow traversing on shorter RTT path may react faster to congestion and recover faster from it compared to another flow on a longer RTT path. The introduction of AQM schemes may potentially improve this type of fairness.

Introducing an AQM scheme may cause the unfairness between the flows, even if the RTTs are identical. This potential unfairness SHOULD be investigated as well.

### 5.2. Recommended tests

The RECOMMENDED topology is detailed in Figure 1:

- o To evaluate the inter-RTT fairness, for each run, two flows divided into two categories. Category I which RTT between sender A and Router L SHOULD be 100ms. Category II which RTT between sender A and Router L should be in [5ms;560ms]. The maximum value for the RTT represents the RTT of a satellite link that, according to the [section 2 of \[RFC2488\]](#) should be at least 558ms.
- o To evaluate the impact of the RTT value on the AQM performance and the intra-protocol fairness (the fairness for the flows using the same paths/congestion control), for each run, two flows (Flow1 and Flow2) should be introduced. For each experiment, the set of RTT SHOULD be the same for the two flows and in [5ms;560ms].

A set of evaluated flows MUST use the same congestion control algorithm.

### 5.3. Metrics to evaluate the RTT fairness

The outputs that MUST be measured are:

- o for the inter-RTT fairness: (1) the cumulative average goodput of the flow from Category I, `goodput_Cat_I` ([Section 2.5](#)); (2) the cumulative average goodput of the flow from Category II, `goodput_Cat_II` ([Section 2.5](#)); (3) the ratio `goodput_Cat_II/goodput_Cat_I`; (4) the average packet drop rate for each category ([Section 2.3](#)).
- o for the intra-protocol RTT fairness: (1) the cumulative average goodput of the two flows ([Section 2.5](#)); (2) the average packet drop rate for the two flows ([Section 2.3](#)).

## 6. Burst Absorption

"AQM mechanisms need to control the overall queue sizes, to ensure that arriving bursts can be accommodated without dropping packets" [[I-D.ietf-aqm-recommendation](#)]

### 6.1. Motivation

An AQM scheme can result in bursts of packet arrivals due to various reasons. Dropping one or more packets from a burst can result in performance penalties for the corresponding flows, since dropped packets have to be retransmitted. Performance penalties can result in failing to meet SLAs and be a disincentive to AQM adoption.

The ability to accommodate bursts translates to larger queue length and hence more queuing delay. On the one hand, it is important that an AQM scheme quickly brings bursty traffic under control. On the other hand, a peak in the packet drop rates to bring a packet burst quickly under control could result in multiple drops per flow and severely impact transport and application performance. Therefore, an AQM scheme ought to bring bursts under control by balancing both aspects -- (1) queuing delay spikes are minimized and (2) performance penalties for ongoing flows in terms of packet drops are minimized.

An AQM scheme that maintains short queues allows some remaining space in the queue for bursts of arriving packets. The tolerance to bursts of packets depends upon the number of packets in the queue, which is directly linked to the AQM algorithm. Moreover, one AQM scheme may implement a feature controlling the maximum size of accepted bursts, that can depend on the buffer occupancy or the currently estimated

queuing delay. The impact of the buffer size on the burst allowance may be evaluated.

## 6.2. Recommended tests

For this scenario, tester MUST evaluate how the AQM performs with the following traffic generated from sender A to receiver B:

- o Web traffic with IW10;
- o Bursty video frames;
- o Constant bit rate UDP traffic.
- o A single bulk TCP flow as background traffic.

Figure 2 presents the various cases for the traffic that MUST be generated between sender A and receiver B.

+-----+-----+-----+-----+-----+				
Case  Traffic Type				
+-----+-----+-----+-----+-----+				
	Video	Webs (IW 10)	CBR	Bulk TCP Traffic
+-----+-----+-----+-----+-----+				
I	0	1	1	0
+-----+-----+-----+-----+-----+				
II	0	1	1	1
+-----+-----+-----+-----+-----+				
III	1	1	1	0
+-----+-----+-----+-----+-----+				
IV	1	1	1	1
+-----+-----+-----+-----+-----+				

Figure 2: Bursty traffic scenarios

A new web page download could start after the previous web page download is finished. Each web page could be composed by at least 50 objects and the size of each object should be at least 1kB. 6 TCP parallel connections SHOULD be generated to download the objects, each parallel connections having an initial congestion window set to 10 packets.

For each of these scenarios, the graph described in [Section 2.7](#) could be generated. Metrics such as end-to-end latency, jitter, flow completion time MAY be generated. For the cases of frame generation of bursty video traffic as well as the choice of web traffic pattern, we leave these details and their presentation to the testers.

## 7. Stability

### 7.1. Motivation

Network devices can experience varying operating conditions depending on factors such as time of the day, deployment scenario, etc. For example:

- o Traffic and congestion levels are higher during peak hours than off-peak hours.
- o In the presence of a scheduler, the draining rate of a queue can vary depending on the occupancy of other queues: a low load on a high priority queue implies a higher draining rate for the lower priority queues.
- o The available capacity at the physical layer can vary over time (e.g., a lossy channel, a link supporting traffic in a higher diffserv class).

Whether the target context is a not stable environment, the ability of an AQM scheme to maintain its control over the queuing delay and buffer occupancy can be challenged. This document proposes guidelines to assess the behavior of AQM schemes under varying congestion levels and varying draining rates.

### 7.2. Recommended tests

Note that the traffic profiles explained below comprises non application-limited TCP flows. For each of the below scenarios, the results described in [Section 2.7](#) SHOULD be generated. For [Section 7.2.5](#) and [Section 7.2.6](#) they SHOULD incorporate the results in per-phase basis as well.

Wherever the notion of time has explicitly mentioned in this subsection, time 0 starts from the moment all TCP flows have already reached their congestion avoidance phase.

#### 7.2.1. Definition of the congestion Level

In these guidelines, the congestion levels are represented by the projected packet drop rate, had a drop-tail queue was chosen instead of an AQM scheme. When the bottleneck is shared among non-application-limited TCP flows.  $l_r$ , the loss rate projection can be expressed as a function of  $N$ , the number of bulk TCP flows, and  $S$ , the sum of the bandwidth-delay product and the maximum buffer size, both expressed in packets, based on Eq. 3 of [[SCL-TCP](#)]:

Internet-Draft

AQM Characterization Guidelines

June 2015

$$l_r = 0.76 * N^2 / S^2$$

$$N = S * \sqrt{1/0.76} * \sqrt{l_r}$$

These guidelines use the loss rate to define the different congestion levels, but they do not stipulate that in other circumstances, measuring the congestion level gives you an accurate estimation of the loss rate or vice-versa.

#### 7.2.2. Mild congestion

This scenario can be used to evaluate how an AQM scheme reacts to a light load of incoming traffic resulting in mild congestion -- packet drop rates around 0.1%. The number of bulk flows required to achieve this congestion level,  $N_{mild}$ , is then:

$$N_{mild} = \text{round}(0.036*S)$$

#### 7.2.3. Medium congestion

This scenario can be used to evaluate how an AQM scheme reacts to incoming traffic resulting in medium congestion -- packet drop rates around 0.5%. The number of bulk flows required to achieve this congestion level,  $N_{med}$ , is then:

$$N_{med} = \text{round} (0.081*S)$$

#### 7.2.4. Heavy congestion

This scenario can be used to evaluate how an AQM scheme reacts to incoming traffic resulting in heavy congestion -- packet drop rates around 1%. The number of bulk flows required to achieve this congestion level,  $N_{heavy}$ , is then:

$$N_{heavy} = \text{round} (0.114*S)$$

#### 7.2.5. Varying congestion levels

This scenario can be used to evaluate how an AQM scheme reacts to incoming traffic resulting in various level of congestions during the experiment. In this scenario, the congestion level varies within a large time-scale. The following phases may be considered: phase I - mild congestion during 0-20s; phase II - medium congestion during 20-40s; phase III - heavy congestion during 40-60s; phase I again, and so on.

#### 7.2.6. Varying available capacity

This scenario can be used to help characterize how the AQM behaves and adapts to bandwidth changes. The experiments are not meant to reflect the exact conditions of Wi-Fi environments since its hard to design repetitive experiments or accurate simulations for such scenarios.

To emulate varying draining rates, the bottleneck capacity between nodes 'Router L' and 'Router R' varies over the course of the experiment as follows:

- o Experiment 1: the capacity varies between two values within a large time-scale. As an example, the following phases may be considered: phase I - 100Mbps during 0-20s; phase II - 10Mbps during 20-40s; phase I again, and so on.
- o Experiment 2: the capacity varies between two values within a short time-scale. As an example, the following phases may be considered: phase I - 100Mbps during 0-100ms; phase II - 10Mbps during 100-200ms; phase I again, and so on.

The tester MAY choose a phase time-interval value different than what is stated above, if the network's path conditions (such as bandwidth-delay product) necessitate. In this case the choice of such time-interval value SHOULD be stated and elaborated.

The tester MAY additionally evaluate the two mentioned scenarios (short-term and long-term capacity variations), during and/or including TCP slow-start phase.

More realistic fluctuating capacity patterns MAY be considered. The tester MAY choose to incorporate realistic scenarios with regards to common fluctuation of bandwidth in state-of-the-art technologies.

The scenario MAY consist of TCP NewReno flows between sender A and receiver B. To better assess the impact of draining rates on the AQM behavior, the tester MUST compare its performance with those of drop-tail and SHOULD provide a reference document for their proposal discussing performance and deployment compared to those of drop-tail. Burst traffic, such as presented in [Section 6.2](#), could also be considered to assess the impact of varying available capacity on the burst absorption of the AQM.



### 7.3. Parameter sensitivity and stability analysis

The control law used by an AQM is the primary means by which the queuing delay is controlled. Hence understanding the control law is critical to understanding the behavior of the AQM scheme. The control law could include several input parameters whose values affect the AQM scheme's output behavior and its stability. Additionally, AQM schemes may auto-tune parameter values in order to maintain stability under different network conditions (such as different congestion levels, draining rates or network environments). The stability of these auto-tuning techniques is also important to understand.

Transports operating under the control of AQM experience the effect of multiple control loops that react over different timescales. It is therefore important that proposed AQM schemes are seen to be stable when they are deployed at multiple points of potential congestion along an Internet path. The pattern of congestion signals (loss or ECN-marking) arising from AQM methods also need to not adversely interact with the dynamics of the transport protocols that they control.

AQM proposals SHOULD provide background material showing control theoretic analysis of the AQM control law and the input parameter space within which the control law operates as expected; or could use another way to discuss the stability of the control law. For parameters that are auto-tuned, the material SHOULD include stability analysis of the auto-tuning mechanism(s) as well. Such analysis helps to understand an AQM control law better and the network conditions/deployments under which the AQM is stable.

## 8. Various Traffic Profiles

This section provides guidelines to assess the performance of an AQM proposal for various traffic profiles such as traffic with different applications or bi-directional traffic.

### 8.1. Traffic mix

This scenario can be used to evaluate how an AQM scheme reacts to a traffic mix consisting of different applications such as:

- o Bulk TCP transfer
- o Web traffic
- o VoIP

- o Constant Bit Rate (CBR) UDP traffic
- o Adaptive video streaming

Various traffic mixes can be considered. These guidelines RECOMMEND to examine at least the following example: 1 bi-directional VoIP; 6 Webs pages download (such as detailed in [Section 6.2](#)); 1 CBR; 1 Adaptive Video; 5 bulk TCP. Any other combinations could be considered and should be carefully documented.

For each scenario, the graph described in [Section 2.7](#) could be generated for each class of traffic. Metrics such as end-to-end latency, jitter and flow completion time MAY be reported.

#### 8.2. Bi-directional traffic

Control packets such as DNS requests/responses, TCP SYN/ACKs are small, but their loss can severely impact the application performance. The scenario proposed in this section will help in assessing whether the introduction of an AQM scheme increases the loss probability of these important packets.

For this scenario, traffic MUST be generated in both downlink and uplink, such as defined in [Section 3.1](#). These guidelines RECOMMEND to consider a mild congestion level and the traffic presented in [Section 7.2.2](#) in both directions. In this case, the metrics reported MUST be the same as in [Section 7.2](#) for each direction.

The traffic mix presented in [Section 8.1](#) MAY also be generated in both directions.

### 9. Multi-AQM Scenario

#### 9.1. Motivation

Transports operating under the control of AQM experience the effect of multiple control loops that react over different timescales. It is therefore important that proposed AQM schemes are seen to be stable when they are deployed at multiple points of potential congestion along an Internet path. The pattern of congestion signals (loss or ECN-marking) arising from AQM methods also need to not adversely interact with the dynamics of the transport protocols that they control.

### 9.2. Details on the evaluation scenario

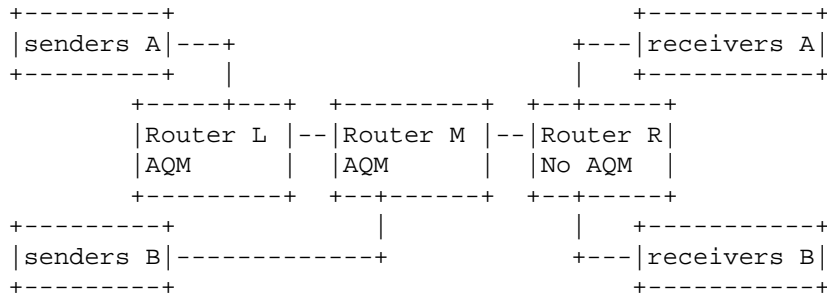


Figure 3: Topology for the Multi-AQM scenario

This scenario can be used to evaluate how having AQM schemes in sequence impact the induced latency reduction, the induced goodput maximization and the trade-off between these two. The topology presented in Figure 3 could be used. We recommend that the AQM schemes introduced in Router L and Router M should be the same; any other configurations could be considered. For this scenario, we recommend to consider a mild congestion level, the number of flows specified in [Section 7.2.2](#) being equally shared among senders A and B. Any other relevant combination of congestion levels could be considered. We recommend to measure the metrics presented in [Section 7.2](#).

## 10. Implementation cost

### 10.1. Motivation

Successful deployment of AQM is directly related to its cost of implementation. Network devices can need hardware or software implementations of the AQM mechanism. Depending on a device's capabilities and limitations, the device may or may not be able to implement some or all parts of the AQM logic.

AQM proposals SHOULD provide pseudo-code for the complete AQM scheme, highlighting generic implementation-specific aspects of the scheme such as "drop-tail" vs. "drop-head", inputs (e.g. current queuing delay, queue length), computations involved, need for timers, etc. This helps to identify costs associated with implementing the AQM scheme on a particular hardware or software device. This also helps the WG understand which kind of devices can easily support the AQM and which cannot.

## 10.2. Recommended discussion

AQM proposals SHOULD highlight parts of AQM logic that are device dependent and discuss if and how AQM behavior could be impacted by the device. For example, a queueing-delay based AQM scheme requires current queueing delay as input from the device. If the device already maintains this value, then it can be trivial to implement the AQM logic on the device. If the device provides indirect means to estimate the queueing delay (for example: timestamps, dequeuing rate), then the AQM behavior is sensitive to the precision of the queueing delay estimations are for that device. Highlighting the sensitivity of an AQM scheme to queueing delay estimations helps implementers to identify appropriate means of implementing the mechanism on a device.

## 11. Operator Control and Auto-tuning

### 11.1. Motivation

One of the biggest hurdles of RED deployment was/is its parameter sensitivity to operating conditions -- how difficult it is to tune RED parameters for a deployment to achieve acceptable benefit from using RED. Fluctuating congestion levels and network conditions add to the complexity. Incorrect parameter values lead to poor performance.

Any AQM scheme is likely to have parameters whose values affect the control law and behaviour of an AQM. Exposing all these parameters as control parameters to a network operator (or user) can easily result in a unsafe AQM deployment. Unexpected AQM behavior ensues when parameter values are set improperly. A minimal number of control parameters minimizes the number of ways a possibly naive user can break a system where an AQM scheme is deployed at. Fewer control parameters make the AQM scheme more user-friendly and easier to deploy and debug.

[I-D.ietf-aqm-recommendation] states "AQM algorithms SHOULD NOT require tuning of initial or configuration parameters in common use cases." A scheme ought to expose only those parameters that control the macroscopic AQM behavior such as queue delay threshold, queue length threshold, etc.

Additionally, the safety of an AQM scheme is directly related to its stability under varying operating conditions such as varying traffic profiles and fluctuating network conditions, as described in [Section 7](#). Operating conditions vary often and hence the AQM needs to remain stable under these conditions without the need for additional external tuning. If AQM parameters require tuning under

these conditions, then the AQM must self-adapt necessary parameter values by employing auto-tuning techniques.

#### 11.2. Required discussion

AQM proposals SHOULD describe the parameters that control the macroscopic AQM behavior, and identify any parameters that require tuning to operational conditions. It could be interesting to also discuss that even if an AQM scheme may not adequately auto-tune its parameters, the resulting performance may not be optimal, but close to something reasonable.

If there are any fixed parameters within the AQM, their setting SHOULD be discussed and justified.

If an AQM scheme is evaluated with parameter(s) that were externally tuned for optimization or other purposes, these values MUST be disclosed.

#### 12. Interaction with ECN

Deployed AQM algorithms SHOULD support Explicit Congestion Notification (ECN) as well as loss to signal congestion to endpoints "[I-D.ietf-aqm-recommendation]". The benefits of providing ECN support for an AQM scheme are described in [ECN-Benefit].

##### 12.1. Motivation

(ECN) [RFC3168] is an alternative that allows AQM schemes to signal receivers about network congestion that does not use packet drop.

##### 12.2. Recommended discussion

An AQM scheme can support ECN [I-D.ietf-aqm-recommendation], in which case testers MUST discuss and describe the support of ECN.

#### 13. Interaction with Scheduling

A network device may use per-flow or per-class queuing with a scheduling algorithm to either prioritize certain applications or classes of traffic, limit the rate of transmission, or to provide isolation between different traffic flows within a common class [I-D.ietf-aqm-recommendation].

### 13.1. Motivation

Coupled with an AQM scheme, a router may schedule the transmission of packets in a specific manner by introducing a scheduling scheme. This algorithm may create sub-queues and integrate a dropping policy on each of these sub-queues. Another scheduling policy may modify the way packets are sequenced, modifying the timestamp of each packet.

### 13.2. Recommended discussion

The scheduling and the AQM conjointly impact on the end-to-end performance. During the characterization process of a dropping policy, the tester **MUST** discuss the feasibility to add scheduling combined with the AQM algorithm. This discussion as an instance, **MAY** explain whether the dropping policy is applied when packets are being enqueued or dequeued.

### 13.3. Assessing the interaction between AQM and scheduling

These guidelines do not propose guidelines to assess the performance of scheduling algorithms. Indeed, as opposed to characterizing AQM schemes that is related to their capacity to control the queuing delay in a queue, characterizing scheduling schemes is related to the scheduling itself and its interaction with the AQM scheme. As one example, the scheduler may create sub-queues and the AQM scheme may be applied on each of the sub-queues, and/or the AQM could be applied on the whole queue. Also, schedulers might, such as FQ-CoDel [FQ-CoDel] or FavorQueue [FAVOUR], introduce flow prioritization. In these cases, specific scenarios should be proposed to ascertain that these scheduler schemes not only helps in tackling the bufferbloat, but also are robust under a wide variety of operating conditions. This is out of the scope of this document that focus on dropping and/or marking AQM schemes.

## 14. Discussion on Methodology, Metrics, AQM Comparisons and Packet Sizes

### 14.1. Methodology

One key objective behind formulating the guidelines is to help ascertain whether a specific AQM is not only better than drop-tail but also safe to deploy. Testers therefore need to provide a reference document for their proposal discussing performance and deployment compared to those of drop-tail.

A description of each test setup **SHOULD** be detailed to allow this test to be compared with other tests. This also allows others to

Internet-Draft

AQM Characterization Guidelines

June 2015

replicate the tests if needed. This test setup SHOULD detail software and hardware versions. The tester could make its data available.

The proposals SHOULD be evaluated on real-life systems, or they MAY be evaluated with event-driven simulations (such as ns-2, ns-3, OMNET, etc). The proposed scenarios are not bound to a particular evaluation toolset.

The tester is encouraged to make the detailed test setup and the results publicly available.

#### 14.2. Comments on metrics measurement

The document presents the end-to-end metrics that ought to be used to evaluate the trade-off between latency and goodput in [Section 2](#). In addition to the end-to-end metrics, the queue-level metrics (normally collected at the device operating the AQM) provide a better understanding of the AQM behavior under study and the impact of its internal parameters. Whenever it is possible (e.g. depending on the features provided by the hardware/software), these guidelines advice to consider queue-level metrics, such as link utilization, queuing delay, queue size or packet drop/mark statistics in addition to the AQM-specific parameters. However, the evaluation MUST be primarily based on externally observed end-to-end metrics.

These guidelines do not aim to detail on the way these metrics can be measured, since the way these metrics are measured is expected to depend on the evaluation toolset.

#### 14.3. Comparing AQM schemes

This document recognizes that the guidelines mentioned above may be used for comparing AQM schemes.

AQM schemes need to be compared against both performance and deployment categories. In addition, this section details how best to achieve a fair comparison of AQM schemes by avoiding certain pitfalls.

##### 14.3.1. Performance comparison

AQM schemes MUST be compared against all the generic scenarios presented in this memo. AQM schemes MAY be compared for specific network environments such as data centers, home networks, etc. If an AQM scheme has parameter(s) that were externally tuned for optimization or other purposes, these values MUST be disclosed.

AQM schemes belong to different varieties such as queue-length based schemes (ex. RED) or queueing-delay based scheme (ex. CoDel, PIE). AQM schemes expose different control knobs associated with different semantics. For example, while both PIE and CoDel are queueing-delay based schemes and each expose a knob to control the queueing delay -- PIE's "queueing delay reference" vs. CoDel's "queueing delay target", the two tuning parameters of the two schemes have different semantics, resulting in different control points. Such differences in AQM schemes can be easily overlooked while making comparisons.

This document RECOMMENDS the following procedures for a fair performance comparison between the AQM schemes:

1. comparable control parameters and comparable input values: carefully identify the set of parameters that control similar behavior between the two AQM schemes and ensure these parameters have comparable input values. For example, to compare how well a queue-length based AQM scheme controls queueing delay vs. a queueing-delay based AQM scheme, a tester can identify the parameters of the schemes that control queue delay and ensure that their input values are comparable. Similarly, to compare how well two AQM schemes accommodate packet bursts, the tester can identify burst-related control parameters and ensure they are configured with similar values.
2. compare over a range of input configurations: there could be situations when the set of control parameters that affect a specific behavior have different semantics between the two AQM schemes. As mentioned above, PIE has tuning parameters to control queue delay that has a different semantics from those used in CoDel. In such situations, these schemes need to be compared over a range of input configurations. For example, compare PIE vs. CoDel over the range of target delay input configurations.

#### 14.3.2. Deployment comparison

AQM schemes MUST be compared against deployment criteria such as the parameter sensitivity ([Section 7.3](#)), auto-tuning ([Section 11](#)) or implementation cost ([Section 10](#)).

#### 14.4. Packet sizes and congestion notification

An AQM scheme may be considering packet sizes while generating congestion signals. [\[RFC7141\]](#) discusses the motivations behind this. For example, control packets such as DNS requests/responses, TCP SYNs/ACKs are small, but their loss can severely impact the application performance. An AQM scheme may therefore be biased



Internet-Draft

AQM Characterization Guidelines

June 2015

towards small packets by dropping them with smaller probability compared to larger packets. However, such an AQM scheme is unfair to data senders generating larger packets. Data senders, malicious or otherwise, are motivated to take advantage of such AQM scheme by transmitting smaller packets, and could result in unsafe deployments and unhealthy transport and/or application designs.

An AQM scheme SHOULD adhere to the recommendations outlined in [RFC7141], and SHOULD NOT provide undue advantage to flows with smaller packets [I-D.ietf-aqm-recommendation].

## 15. Conclusion

Figure 4 lists the scenarios and their requirements.

Scenario	Sec.	Requirement
Transport Protocols	4.	
TCP-friendly sender	4.1	Scenario MUST be considered
Aggressive sender	4.2	Scenario MUST be considered
Unresponsive sender	4.3	Scenario MUST be considered
LBE sender	4.4	Scenario MAY be considered
Round Trip Time Fairness	5.2	Scenario MUST be considered
Burst Absorption	6.2	Scenario MUST be considered
Stability	7.	
Varying congestion levels	7.2.5	Scenario MUST be considered
Varying available capacity	7.2.6	Scenario MUST be considered
Parameters and stability	7.3	This SHOULD be discussed
Various Traffic Profiles	8.	
Traffic mix	8.1	Scenario is RECOMMENDED
Bi-directional traffic	8.2	Scenario MAY be considered
Multi-AQM	9.2	Scenario MAY be considered
Implementation Cost	10.2	Pseudo-code SHOULD be provided
Operator Control	11.2	Tuning SHOULD NOT be required
Interaction with ECN	12.2	MUST be discussed if supported
Interaction with Scheduling	13.2	Feasibility MUST be discussed

Figure 4: Summary of the scenarios and their requirements

## 16. Acknowledgements

This work has been partially supported by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

## 17. Contributors

Many thanks to S. Akhtar, A.B. Bagayoko, F. Baker, D. Collier-Brown, G. Fairhurst, J. Gettys, T. Hoiland-Jorgensen, C.

Internet-Draft

AQM Characterization Guidelines

June 2015

Kulatunga, W. Lautenschlager, A.C. Morton, R. Pan, D. Taht and M. Welzl for detailed and wise feedback on this document.

## 18. IANA Considerations

This memo includes no request to IANA.

## 19. Security Considerations

Some security considerations for AQM are identified in [I-D.ietf-aqm-recommendation]. This document, by itself, presents no new privacy nor security issues.

## 20. References

### 20.1. Normative References

- [I-D.ietf-aqm-recommendation]  
Baker, F. and G. Fairhurst, "IETF Recommendations Regarding Active Queue Management", [draft-ietf-aqm-recommendation-11](#) (work in progress), February 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [RFC 2119](#), 1997.
- [RFC7141] Briscoe, B. and J. Manner, "Byte and Packet Congestion Notification", [RFC 7141](#), 2014.

### 20.2. Informative References

- [BB2011] "BufferBloat: what's wrong with the internet?", ACM Queue vol. 9, 2011.
- [CODEL] Nichols, K. and V. Jacobson, "Controlling Queue Delay", ACM Queue , 2012.
- [ECN-Benefit]  
Welzl, M. and G. Fairhurst, "The Benefits to Applications of using Explicit Congestion Notification (ECN)", IETF (Work-in-Progress) , February 2014.
- [FAVOUR] Anelli, P., Diana, R., and E. Lochin, "FavorQueue: a Parameterless Active Queue Management to Improve TCP Traffic Performance", Computer Networks vol. 60, 2014.

Internet-Draft

AQM Characterization Guidelines

June 2015

- [FQ-CoDel] Hoeiland-Joergensen, T., McKeeney, P., Taht, D., Gettys, J., and E. Dumazet, "FlowQueue-Codel", IETF (Work-in-Progress) , January 2015.
- [LBE-AQM] Gong, Y., Rossi, D., Testa, C., Valenti, S., and D. Taht, "Fighting the bufferbloat: on the coexistence of AQM and low priority congestion control", Computer Networks, Elsevier, 2014, 60, pp.115 - 128 , 2014.
- [LEDBAT-PARAM] Trang, S., Kuhn, N., Lochin, E., Baudoin, C., Dubois, E., and P. Gelard, "On The Existence Of Optimal LEDBAT Parameters", IEEE ICC 2014 - Communication QoS, Reliability and Modeling Symposium , 2014.
- [LOSS-SYNCH-MET-08] Hassayoun, S. and D. Ros, "Loss Synchronization and Router Buffer Sizing with High-Speed Versions of TCP", IEEE INFOCOM Workshops , 2008.
- [PIE] Pan, R., Natarajan, P., Piglione, C., Prabhu, MS., Subramanian, V., Baker, F., and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem", IEEE HPSR , 2013.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, [RFC 793](#), September 1981.
- [RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", [RFC 2309](#), April 1998.
- [RFC2488] Allman, M., Glover, D., and L. Sanchez, "Enhancing TCP Over Satellite Channels using Standard Mechanisms", [BCP 28](#), [RFC 2488](#), January 1999.
- [RFC2544] Bradner, S. and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices", [RFC 2544](#), March 1999.
- [RFC2647] Newman, D., "Benchmarking Terminology for Firewall Performance", [RFC 2647](#), August 1999.
- [RFC2679] Almes, G., Kalidindi, S., and M. Zekauskas, "A One-way Delay Metric for IPPM", [RFC 2679](#), September 1999.

Internet-Draft

AQM Characterization Guidelines

June 2015

- [RFC2680] Almes, G., Kalidindi, S., and M. Zekauskas, "A One-way Packet Loss Metric for IPPM", [RFC 2680](#), September 1999.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", [RFC 3168](#), September 2001.
- [RFC3611] Friedman, T., Caceres, R., and A. Clark, "RTP Control Protocol Extended Reports (RTCP XR)", [RFC 3611](#), November 2003.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", [RFC 5348](#), September 2008.
- [RFC5481] Morton, A. and B. Claise, "Packet Delay Variation Applicability Statement", [RFC 5481](#), March 2009.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", [RFC 5681](#), September 2009.
- [RFC6297] Welzl, M. and D. Ros, "A Survey of Lower-than-Best-Effort Transport Protocols", [RFC 6297](#), June 2011.
- [RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", [RFC 6817](#), December 2012.
- [SCL-TCP] Morris, R., "Scalable TCP congestion control", IEEE INFOCOM , 2000.
- [TCPEVAL2013] Hayes, D., Ros, D., Andrew, L., and S. Floyd, "Common TCP Evaluation Suite", IRTF ICCRG , 2013.
- [YU06] Jay, P., Fu, Q., and G. Armitage, "A preliminary analysis of loss synchronisation between concurrent TCP flows", Australian Telecommunication Networks and Application Conference (ATNAC) , 2006.

Authors' Addresses

Internet-Draft

AQM Characterization Guidelines

June 2015

Nicolas Kuhn (editor)  
Telecom Bretagne  
2 rue de la Chataigneraie  
Cesson-Sevigne 35510  
France

Phone: +33 2 99 12 70 46  
Email: nicolas.kuhn@telecom-bretagne.eu

Naeem Khademi (editor)  
University of Oslo  
Department of Informatics, PO Box 1080 Blindern  
N-0316 Oslo  
Norway

Phone: +47 2285 24 93  
Email: naeemk@ifi.uio.no

Preethi Natarajan (editor)  
Cisco Systems  
510 McCarthy Blvd  
Milpitas, California  
United States

Email: prenatar@cisco.com

David Ros  
Simula Research Laboratory AS  
P.O. Box 134  
Lysaker, 1325  
Norway

Phone: +33 299 25 21 21  
Email: dros@simula.no

# Operating ranges and tunability of CoDel and PIE

Amadou Baba Bagayoko<sup>1</sup>, Gorrry Fairhurst<sup>2</sup>, Naeem Khademi<sup>3</sup>, Nicolas Kuhn<sup>1</sup>,  
Chamil Kulatunga<sup>2</sup>, David Ros<sup>4</sup>

<sup>1</sup> IMT Télécom Bretagne / IRISA, France

<sup>2</sup> School of Engineering, University of Aberdeen, United Kingdom

<sup>3</sup> Department of Informatics, University of Oslo, Norway

<sup>4</sup> Simula Research Laboratory, Fornebu, Norway

## ABSTRACT

Bufferbloat is excessive delay due to the accumulation of packets in a router's oversized queues. CoDel and PIE are two recent Active Queue Management (AQM) algorithms that have been proposed to address bufferbloat by reducing the queuing delay while trying to maintain a high bottleneck utilization. In this paper, we outline what are the operating ranges, that is the network characteristics (in terms of round-trip times and bottleneck capacity), for which these algorithms achieve their design goals. We highlight deployment scenarios where both AQM schemes result in poor performance when used with default parameters, and we evaluate to what extent we can tune these to achieve various trade-offs. We find that, by appropriate tuning (1) the amount of buffering can easily be controlled with PIE, (2) the RTT sensitivity of CoDel can be reduced. Also, we observe there is more correlation between the congestion level, the achieved queuing delay and the targeted delay with CoDel than with PIE. An overall winner for the best AQM scheme does not exist, as each scheme proposes a specific trade-off, and each works within a limited range of network characteristics or traffic profiles.

## Keywords

AQM; Bufferbloat; CoDel; PIE; congestion control.

## 1. INTRODUCTION

The combination of large buffers and loss-based congestion control mechanisms result in persistently full buffers and increased end-to-end delay; this issue, known as bufferbloat [4], may be a serious hindrance to the increasing number of latency sensitive applications. Even though there is some discussion on whether bufferbloat is a widespread problem [2, 10], its presence has been actually observed in mobile networks [12, 1].

Active Queue Management (AQM) has been proposed as a way to alleviate bufferbloat, i.e., it should be able to reduce end-to-end latency and avoid lock-out phenomena within the Internet [3]. Widespread deployment of AQM has not happened since the first proposals dating back more than a decade, mainly due to their sensitivity to the operating conditions in the network and to the characteristics of the low layers, and especially the difficulty of tuning their parameters. Indeed, even though the hard-to-tune RED was quickly followed up by the Adaptive Random Early Detection (ARED) [6, 5] algorithm, and even though many other

mechanisms have since been proposed to ease setting parameters, AQM schemes have been reported to be usually turned off. Controlled Delay (CoDel) [18] and Proportional Integral controller Enhanced (PIE) [19] are two recently proposed AQM schemes that aim at tackling bufferbloat, by controlling the queuing delay while attempting to address the problems that (A)RED encountered in terms of deployment and stability. The authors of PIE mention in [19] that PIE “self-tunes its parameters to optimize system performance” depending on network conditions, and the authors of CoDel say in [18] that CoDel “controls delay, while being insensitive to round-trip delays, link rates, and traffic loads.”

In this paper, based on ns-2 simulation results, we try to evaluate the limits of PIE's self-tuning ability and CoDel's sensitivity to network conditions by evaluating their operating ranges (in terms of congestion levels, round-trip times or link rates). In fact, CoDel has been found to be sensitive to the congestion level [11, 13], and PIE and CoDel have difficulties controlling the queuing delay both in high RTT and high capacity paths [21] and in low capacity paths [8]. Our work aims not only at verifying these conclusions, but also at assessing the range of conditions in which the default parameters of PIE and CoDel fail to perform well. We also evaluate how tunable CoDel and PIE are, i.e., whether they can easily be adjusted to (1) achieve any desired trade-off between queuing delay and bottleneck utilization and (2) let CoDel and PIE improve their performance outside their operating range.

The remainder of this paper is structured as follows. In Section 2, we briefly describe the CoDel and PIE algorithms. We present the simulation setup in Section 3. Section 4 looks into the operating ranges of the AQM schemes. In Section 5, we study the tunability of CoDel and PIE, that is, how their parameters can be set to achieve different trade-offs. Section 6 concludes the paper.

## 2. CODEL AND PIE ALGORITHMS

In this section, we sketch how CoDel and PIE work, focusing on how these algorithms use similarly-named *target delay* (denoted by  $\tau$ ) and *update interval* (denoted by  $\lambda$ ) parameters in a different manner. When needed for clarity, we use the notation  $\tau_x, \lambda_x$  to denote the parameters of AQM algorithm  $x$ .

### 2.1 CoDel

CoDel keeps track of  $enq_i$ , the *enqueue* time of each packet

$p_i$ . At the *dequeue* time  $deq_i$ , CoDel measures the queuing delay of each dequeued packet,  $\delta_i = deq_i - enq_i$ . The algorithm has two states, a dropping state and a non-dropping state, and the initial state is non-dropping. CoDel keeps track of the number of consecutive drops in a variable  $n_{drop}$ , that is initialized as:  $n_{drop} = 1$ . We denote by  $\mu_{n_{drop}}$  the interval after which CoDel may enter (or stay in) the dropping state and drop one packet;  $\mu_{n_{drop}}$  depends on the number of consecutive drops, as follows:

$$\mu_{n_{drop}} = \lambda_{CoDel} / \sqrt{n_{drop}}. \quad (1)$$

Let  $t_1$  denote the time of the last drop. When a packet  $p_i$  is about to be dequeued at time  $t$ :

- if  $\delta_i > \tau_{CoDel}$  and  $t > t_1 + \mu_{n_{drop}}$ , then: (a) CoDel enters (or stays in) the dropping state, (b)  $p_i$  is dropped, (c)  $n_{drop}$  is increased by 1;
- else: (a) CoDel enters the non dropping state, (b)  $p_i$  is not dropped, (c)  $n_{drop}$  is set to 1.

Then, at  $t_2$ ,  $\mu_{n_{drop}}$  is updated as per (1).

## 2.2 PIE

PIE estimates the current queuing delay,  $E[T]$ , by considering the current queue length,  $qlen$ , and the draining rate of the queue,  $depart\_rate$  as follows:  $E[T] = qlen / depart\_rate$ .  $E[T]_{old}$  represents the previous estimation of the queuing delay. PIE updates its drop probability,  $p_{drop}$ , every  $\lambda_{PIE}$  according to Equation (2). When a packet is enqueued, it is dropped with probability  $p_{drop}$ .

$$p_{drop} = p_{drop} + \alpha \times (E[T] - \tau_{PIE}) + \beta \times (E[T] - E[T]_{old}) \quad (2)$$

The  $\alpha$  parameter determines how the deviation of current latency from the target value affects the drop probability. The  $\beta$  term exerts additional adjustments depending on whether the latency is trending up or down. PIE scales up these parameters to make  $p_{drop}$  adapt more rapidly when  $p_{drop} \notin [0.01, 0.1]$ .

## 2.3 Default parameterizations

We choose to evaluate the operating ranges of CoDel and PIE using what is considered to be their default parameterizations. The default values of  $\tau$  and  $\lambda$  are:  $\tau_{CoDel} = 5$  ms,  $\lambda_{CoDel} = 100$  ms,  $\tau_{PIE} = 20$  ms and  $\lambda_{PIE} = 30$  ms.

The default settings of CoDel and PIE, together with their different ways of deciding whether to drop a packet, achieve different trade-offs between the bottleneck utilization and the queuing delay. Intuitively, this can be explained as follows:

- PIE adapts its drop probability every  $\lambda_{PIE}$ , whereas CoDel may drop a packet only every  $\mu_{n_{drop}}$  in order to give end-points time to react to a drop.
- PIE tries to maintain the average queuing delay around  $\tau_{PIE}$ , whereas CoDel would tend to act as a delay limiter, since drops are only applied if delays grow above  $\tau_{CoDel}$ .
- PIE allows more buffering than CoDel which would result in higher bottleneck utilization.

## 2.4 Scheduling and AQM schemes

A recent proposal, Flow-Queueing CoDel (FQ-CoDel) [9] combines Stochastic Fair Queuing (SFQ), priority queuing and CoDel. The IETF recommendations on AQM [3] note that “queue management” and “scheduling schemes” are closely related, but address different performance issues. They may be used in combination, but they should not be confused [3]. This article focuses on queue management alone and the operating ranges of CoDel and PIE, and does not consider FQ-CoDel or any other combination of AQM and scheduling.

## 3. NETWORK CHARACTERISTICS

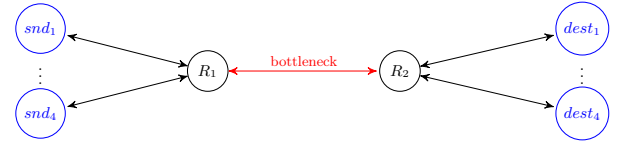


Figure 1: Dumbbell simulation topology.

Figure 1 presents the topology for the ns-2 simulations. We define three congestion levels, *light*, *moderate* and *heavy*, where respectively 4, 16 and 64 long-lived TCP flows are associated with 4 sender/receiver pairs sharing the bottleneck. For the following simulation evaluations, the congestion control is TCP NewReno with SACK option enabled, and a packet size of 1500 B is used throughout.

The capacity of the bottleneck is 10 Mbps, and the capacities of the links  $snd_{1...n} \leftrightarrow R_1$  and  $dest_{1...n} \leftrightarrow R_2$  are 100 Mbps, unless otherwise noted. The size of buffers for the simulations are set according to the bandwidth-delay product, where the bandwidth is the capacity of the bottleneck and the delay is the base RTT (i.e., in the absence of queuing).

Figure 2 serves as example of the figures used in this paper: the boxes encompass the first to third quartile (25 % to 75 % of the samples) and the median value is shown as a point. On the  $x$  axis, the distance  $\Delta$  to the target delay is the difference between the queuing delay  $\delta$  and the target delay  $\tau$ :

$$\Delta = \delta - \tau \quad (3)$$

We represent the bottleneck utilization as a function of  $\Delta$ . Figure 2 highlights that both PIE and CoDel try to maximise the bottleneck utilisation, while keeping the queuing delay under control with respect to their design goals.

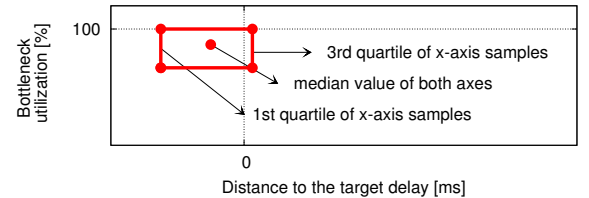


Figure 2: Plot style used in Figures 3 to 8.

The bottleneck utilization is averaged every 5 s and we measure the per-packet queuing delay. Each simulation lasts



300 s and has been repeated with 10 different random seeds, so each figure summarizes 600 link utilization samples and several thousands of per packet delay samples.

## 4. SENSITIVITY OF CODEL AND PIE TO NETWORK CONDITIONS

### 4.1 Various congestion levels

Traffic and congestion levels are higher during peak hours than off-peak hours. When an AQM scheme is deployed in a non-stationary traffic environment, its ability to actually maintain its control on the buffer occupancy is challenged. Thus, this section assesses the sensitivity of CoDel and PIE to different congestion levels.

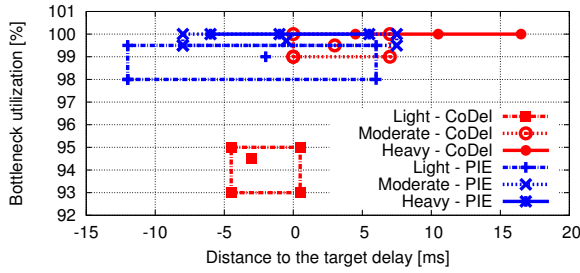


Figure 3: Impact of the congestion level.

For the tests discussed in this section, the base RTT was set to 100 ms, and the one-way delay (OWD) of each link is  $RTT/6$ .

Figure 3 shows that PIE's ability to control the median queuing delay does not depend strongly on the level of congestion. Under the three levels of congestion considered, the median queuing delay is kept close to  $\tau_{PIE}$ . We notice that the spread decreases as the congestion level increases. PIE's algorithm enables it to have an median queuing delay very close to  $\tau_{PIE}$  with the traffic considered. However, probably because PIE considers previous queue states, adaptation of drop probability to changes of the queue occupancy can be slow, resulting in non-negligible variations in queuing delay.

The figure also shows that the median queuing delay with CoDel is lower than, but close to,  $\tau_{CoDel}$ , which is consistent with claims in [17]. However, we can see that higher loads can result in delays higher than 20 ms; under heavy load, median queuing delay is  $\approx 15$  ms (i.e.,  $\Delta \approx 10$  ms).

As discussed in § 2.3, by default PIE should allow more buffering than CoDel. This is verified by the results presented in this section (remember that  $\Delta = 0$  means  $\delta = 20$  ms for PIE but  $\delta = 5$  ms for CoDel).

To summarize, with PIE, the achieved median queuing delay is close to  $\tau_{PIE}$  for the tested congestion levels. However the variation of the queuing delay may be high. With CoDel, we noted a stronger correlation between the congestion level, the queuing delay and  $\tau$ .

### 4.2 RTT sensitivity

A different base RTT—that is, the minimum RTT, without any queuing delays—can impact the behaviour of congestion controllers and, as a result, the ability of an AQM to control the queue. Thus, this section examines the sensitivity of CoDel and PIE to different base RTTs.

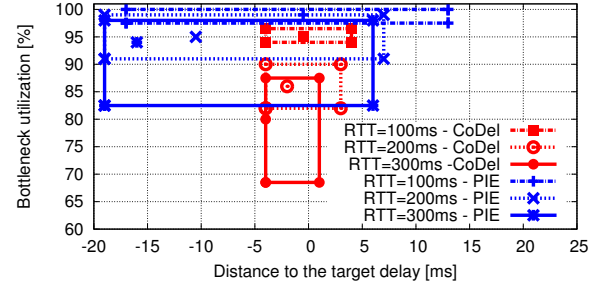


Figure 4: Impact of the base RTT.

The analysis below considers the following edge link characteristics:  $RTT_{(src1 \dots N \leftrightarrow R1)} = RTT_{(dest1 \dots N \leftrightarrow R2)} = 1.25$  ms. The RTT of the bottleneck varies so that the base RTT is one of: 100 ms, 200 ms and 300 ms. We do not consider a lower RTT: since CoDel has shown poorer performance with larger RTTs [18, Figure 8], we prefer focusing on an RTT higher than 100 ms.

Delays of 200 and 300 ms can be considered as common for intercontinental paths. We consider a scenario where the likely congestion level in such a network is light.

The results presented in Figure 4 show that both CoDel and PIE are sensitive to the base RTT value. For both algorithms, when the RTT increases:

- the median bottleneck utilization is lower—down to 80 % with CoDel and down to 94 % with PIE when the RTT is 300 ms;
- the median queuing delay decreases;
- there is more variation in bottleneck utilization and less variation in queuing delay.

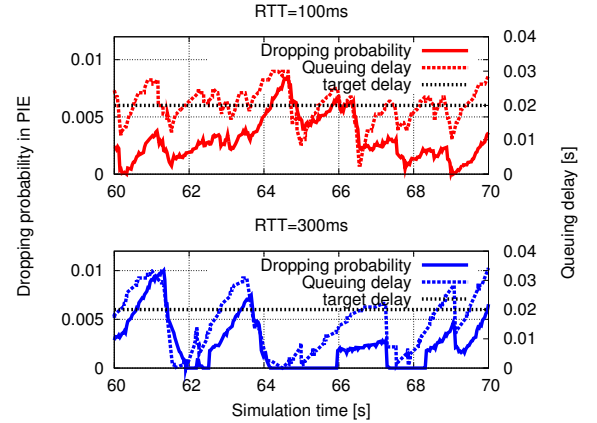


Figure 5: Behavior of PIE with RTTs of 100 and 300 ms.

To better understand the impact of higher RTTs on the behavior of AQM schemes, Figure 5 presents the queuing delay (averaged every second) and the dropping probability of PIE for two RTT values during a representative time interval. When the RTT is 100 ms, the queuing delay tends to oscillate around  $\tau_{PIE}$ . When the RTT is 300 ms, the maximum queuing delay and the maximum dropping probability

are not higher than when the RTT is 100 ms. The main difference observed when the RTT increases, is wider oscillations in buffer occupancy and queuing delay: when the queuing delay is higher than  $\tau_{PIE}$ , the dropping probability increases in order to maintain a queuing delay around  $\tau_{PIE}$ . However, when the RTT increases, the increase of the dropping probability does not result in a queuing delay around  $\tau_{PIE}$  but in a momentarily empty buffer.

### 4.3 Capacity limited networks

There is no single “typical” bottleneck speed and there is a wide variety of access technologies; fiber deployments in urban areas may offer around 100 Mbps, but long-haul DSL used in many developed countries for rural access networks may offer only 1 Mbps and some links may still have lower capacities. Thus, this section looks at the sensitivity of CoDel and PIE to low link capacities.

The following analysis considers the following edge link characteristics:  $RTT_{(src_1 \dots N \leftrightarrow R_1)} = RTT_{(dest_1 \dots N \leftrightarrow R_2)} = 1.25$  ms. The base RTT is 100 ms. The capacity of the bottleneck is set to either 500 kbps, 1 Mbps or 2 Mbps. We consider a scenario where the congestion level is light, as defined in § 3.

Table 1 shows the time needed to transmit 1500 B (the packet size used in this paper) and the resulting size of the average or maximum queue size that PIE or CoDel, respectively, tries to maintain. In these scenarios  $\tau_{CoDel}$  and  $\tau_{PIE}$  are so low that the targeted queuing delays are by default not achievable as they mean extremely short queues: this can be seen in Figure 6 with queuing delays way above the targets when the bottleneck is set to 500 kbps.

Table 1: Bottleneck capacity, BDP and  $\tau$ .

	bottleneck capacity		
	500 kbps	1 Mbps	2 Mbps
transmission time of 1500 B (ms)	24	12	6
BDP (in 1500 B packets)	4.2	8.3	16.6
PIE: avg. queue size aimed (in pkts)	$\approx 1$	$\approx 2$	$\approx 3$
CoDel: max. queue size aimed (in pkts)	$< 1$	$< 1$	$\approx 1$

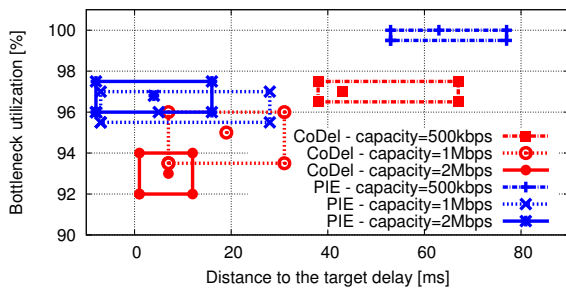


Figure 6: Impact of low bottleneck capacities.

### 4.4 Discussion

In § 4.1, we saw that CoDel is sensitive to the traffic load; in § 4.2, that CoDel and PIE are sensitive to the base RTT; and in § 4.3, that CoDel and PIE are sensitive to link speeds and “break” when link capacities are very low: the default parameters of PIE and CoDel may not suit every situation, making these AQM schemes to drastically reduce the bottle-

neck utilization or to allow a queuing delay that is far from  $\tau$ .

Moreover, in light of the results presented so far, we can conclude that:

- CoDel may not work well with RTTs in the interval [10 ms, 1 s] and is sensitive to the traffic load;
- the self-tuning of PIE, supposed to make it robust and optimized for various network conditions, shows some limits.

Therefore, suggesting that no knobs need to be turned may be misleading.

The AQM characterization guidelines detail that the assessments “are not bound to a particular evaluation tool-set” [15]. Our high-level conclusions, based on ns-2 simulations, are consistent not only with other works that used ns-2 as well [11], but also with studies based on IKR SimLib and its Linux Virtual Machine (VM) integration [21] or emulated networks [14] using the Linux implementations of the algorithms.

## 5. TUNABILITY OF CODEL AND PIE

The analysis below is based not only on the results presented in § 4, but also on previously published work by other authors. We focus on cases where the default parameters are not suitable, and  $\tau$  and  $\lambda$  were changed to assess whether CoDel and PIE can be easily tuned to (1) achieve any desired trade-off between queuing delay and bottleneck utilization and (2) let CoDel and PIE improve their performance outside their “default” operating ranges.

### 5.1 Operating ranges of CoDel and PIE

We define the operating range as the network characteristics (bottleneck capacity, RTT) for which the AQM schemes work as desired, i.e., maintaining a high bottleneck utilization and a low queuing delay.

Based on our results and those of [21, 14, 8, 11], we outline below the operating ranges of PIE and CoDel. Considering only the RTT and the capacity of the bottleneck, denoted  $base_{RTT}$  and  $C_{bot}$ , PIE and CoDel, with their default parameters, have trouble with controlling the queuing delay and maintaining a high bottleneck utilization if:

- $base_{RTT} \geq 200$  ms (§ 4.2 and [14, 21]);
- $C_{bot} < 2$  Mbps (§ 4.3 and [8]);
- $C_{bot} > 100$  Mbps ([21]).

We have observed that these AQMs do not fulfill their design goal outside these bounds, but we cannot guarantee that they work as expected within them. If we consider other parameters than  $base_{RTT}$  and  $C_{bot}$ , our results and those presented in [11] illustrate that there is a high correlation between the traffic load and the performance of CoDel.

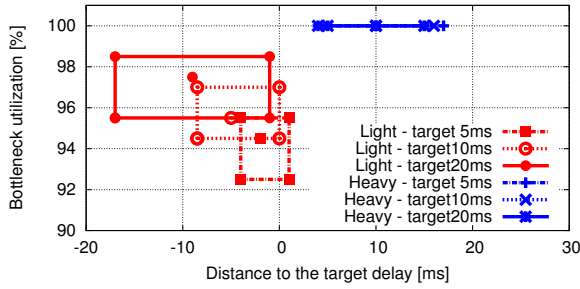
With the specific characteristics of e.g. rural broadband ( $base_{RTT} \approx 300$  ms and  $C_{bot} \approx 1$  Mbps) and data-center networks ( $base_{RTT} \approx 10^{-5}$  ms and  $C_{bot} \approx 10^4$  Mbps), the limited operating range of CoDel and PIE prevents them from working as expected in these contexts. We therefore turn to evaluating the “tunability” of PIE and CoDel, that is, to assess to what extent we can obtain different trade-offs than the default ones by changing only  $\tau$  and  $\lambda$ ; this would be a

straightforward way to adapt their operating ranges to specific network conditions. PIE has already been updated for data-centers [20] by changing the  $\alpha$  and  $\beta$  parameters (see Eq. (2)). However, the IETF recommendations on AQM [3] advocate the selection of default parameters appropriate to the general Internet with auto-tuning to avoid RED's deployment issues. In [17], the authors say that  $\lambda_{\text{CoDel}}$  should be set to the RTT and  $\tau_{\text{CoDel}}$  to 5% of  $\lambda_{\text{CoDel}}$ . The objective is to make CoDel less sensitive to different base RTTs, but the viability of these settings was not shown by the authors of [17]. We should also verify whether  $\tau_{\text{CoDel}}$  is actually close to the queuing delay when it is set to another value than 5% of  $\lambda_{\text{CoDel}}$ .

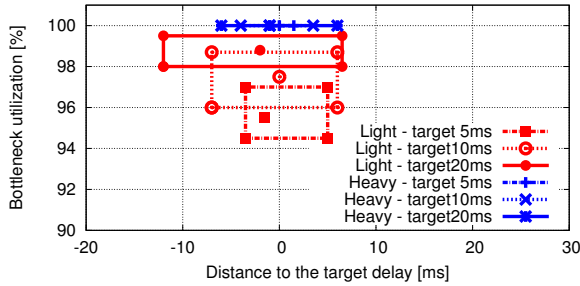
## 5.2 Impact of the target delay

We first evaluate the sensitivity of CoDel and PIE to changes of  $\tau$  for various congestion levels. The network characteristics (RTT, link capacity, traffic generation) are the same as in § 4.1.

In our simulations, we also tested with other values of  $\tau$  but results were qualitatively similar, so for the sake of clarity we only present those corresponding to  $\tau = \{5, 10, 20\}$  ms.



(a) CoDel.



(b) PIE.

Figure 7: Sensitivity to  $\tau$ ;  $\lambda$  is set to its default value for each AQM.

Figure 7 shows that, as  $\tau$  increases, both CoDel and PIE allow more buffering, resulting in higher bottleneck utilization. Also, when  $\tau$  increases, the queuing delay  $\delta$ , that is  $\Delta + \tau$ , increases. For both schemes, when the congestion level is heavy, an increase in  $\tau$  does not significantly change the bottleneck utilization or the queuing delay.

With CoDel (Figure 7a), when the congestion level is light and for all the target delays considered, 75 % of the queuing delay samples are lower than  $\tau_{\text{CoDel}}$ , which is in line with

what CoDel has been designed for. When the congestion level is heavy, the median and maximum queuing delays are higher than  $\tau_{\text{CoDel}}$ , no matter the value of  $\tau_{\text{CoDel}}$ . Thus, CoDel does act as a delay limiter as long as the traffic load is not high.

With PIE (Figure 7b), under the two levels of congestion considered, and for the various target delays considered, the median queuing delay is kept close to  $\tau_{\text{PIE}}$ . Increasing  $\tau_{\text{PIE}}$  results in higher bottleneck utilization when the congestion level is light. Therefore, different trade-offs can be achieved with PIE: by changing  $\tau_{\text{PIE}}$ , one can accurately tune the median queuing delay to any desired value, bearing in mind that increasing  $\tau_{\text{PIE}}$  results in a higher bottleneck utilization and more variability in queuing delay.

## 5.3 Impact of the update interval

In this section, we evaluate the sensitivity of CoDel and PIE to changes of  $\lambda$  when the congestion level is light. The network characteristics (RTT, link capacity, traffic generation) are the same as in § 4.1 with a base RTT of 300 ms.

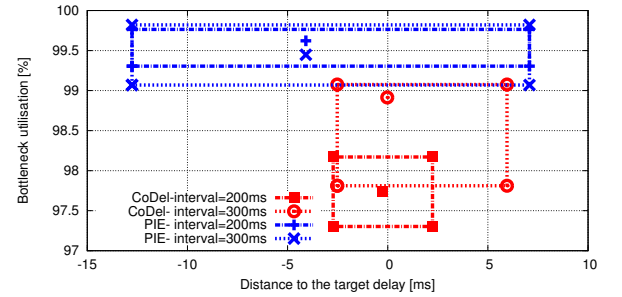


Figure 8: Sensitivity to  $\lambda$ .  $RTT = 300$  ms, default  $\tau$ .

The performance issues that had been observed in § 4.2 are alleviated when  $\lambda_{\text{CoDel}}$  is higher than 100 ms and set closer to the RTT, as illustrated in Figure 8; contrary to what is seen Figure 4, utilization is very close to 100%. However, in most deployment cases, the path RTT is not known at the router deploying the AQM.

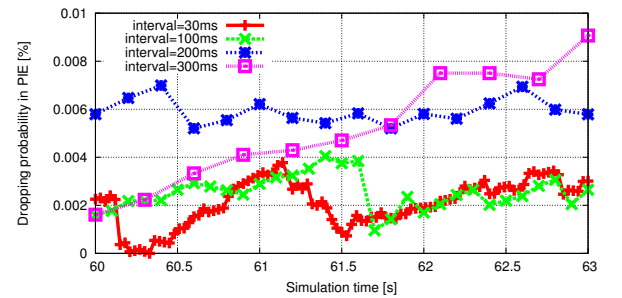


Figure 9: PIE's dropping probability for various values of  $\lambda_{\text{PIE}}$ .

By increasing  $\lambda_{\text{PIE}}$  when the RTT increases, the median bottleneck utilization can be improved (compare with the results in Figure 4), but the high variability of the queuing delay cannot be reduced. To visualize the impact of  $\lambda_{\text{PIE}}$  on PIE's behavior, we represent in Figure 9 the evolution of the drop probability, during a representative time interval, for

various values of  $\lambda$ . This figure shows that increasing  $\lambda_{PIE}$  tends to result in a smoother evolution of the drop probability, which does not actually reduce the RTT sensitivity of PIE in terms of stabilizing queuing delay.

## 6. CONCLUSION

Deploying AQM is essential step in reducing end-to-end latency [3]. Widespread deployment of early AQM proposals has not happened yet, mainly due to their sensitivity to the network characteristics. Two recently proposed AQM schemes attempt to tackle bufferbloat while addressing these issues. Before considering actual deployment of CoDel and PIE, it is essential to outline their operating ranges (in terms of RTT, bottleneck capacity and congestion levels). In this paper, we evaluate the limits of PIE's self-tuning ability and the sensitivity of CoDel to network conditions. Our conclusions, based on ns-2 simulations, are consistent with recently published studies that emulates networks using Linux implementations of the algorithms. We found that both schemes show performance issues when the RTT is higher than 200 ms, or the bottleneck capacity is lower than 2 Mbps. We also measure how the default parameterization of PIE and CoDel lets PIE allow more queuing delay than CoDel and achieve higher bottleneck utilization.

Because the use of CoDel and PIE with their default parameterizations results in poor performance for some deployment use-cases, such as data-centers or rural broadband networks, manual tuning can hardly be avoided. Also, within the operating ranges, the desirable performance may differ from the trade-off that can be obtained with the default parameters of CoDel and PIE. As a result, we have also studied to what extent we can manually tune them. We saw that the median queuing delay can easily be modified with PIE and CoDel by changing the target delay only; however, this works for CoDel only when the congestion level is low. Setting the update interval to the RTT reduces the RTT sensitivity of CoDel, but in many deployment cases, the person configuring the router will not be aware of the path RTT. Increasing the update interval tends to result in smoother evolution of the drop probability for PIE.

PIE and CoDel have had to be updated for cable-modems networks [7]: in the context of networks with characteristics that make them fit into the operating ranges, the adequate performance of CoDel and PIE can not be guaranteed as other parameters may have an impact, such as lower layers characteristics. In light of our results and the discussions in [22, 16], we claim that there is no overall winner for the best AQM: each algorithm proposes a specific trade-off, and works within limited network or traffic characteristics.

## Acknowledgment

This work was part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the authors.

## References

- [1] S. Alfredsson, G. Del Giudice, J. Garcia, A. Brunstrom, L. De Cicco, and S. Mascolo. "Observations of Bufferbloat in Swedish Cellular Networks". In: *Proceedings of the 9th Swedish National Computer Networking Workshop (SNCNW 2013)*: 2013.
- [2] M. Allman. "Comments on Bufferbloat". In: *SIGCOMM Computer Communication Review* (2013).
- [3] F. Baker and G. Fairhurst. *IETF Recommendations Regarding Active Queue Management*. IETF. 2014.
- [4] "BufferBloat: What's Wrong with the Internet?" In: *Communications of the ACM* (2012).
- [5] S. Floyd, R. Gummadi, and S. Shenker. *Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management*. Tech. rep. 2001.
- [6] S. Floyd and V. Jacobson. "Random Early Detection Gateways for Congestion Avoidance". In: *Networking, IEEE/ACM Transactions on* (1993).
- [7] Greg White, CableLabs. "Active Queue Management Algorithms for DOCSIS 3.0: A Simulation Study of CoDel, SFQ-CoDel and PIE in DOCSIS 3.0 Networks". In: *Cable Television Laboratories* (2013).
- [8] E. Grigorescu, C. Kulatunga, and G. Fairhurst. "Evaluation of the Impact of Packet Drops due to AQM over Capacity Limited Paths". In: *CSWS* (2013).
- [9] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet. *FlowQueue-Codel*. IETF. 2014.
- [10] O. Hohlfeld, E. Pujol, F. Ciucu, A. Feldmann, and P. Barford. "A QoE Perspective on Sizing Network Buffers". In: *ACM IMC* (2014).
- [11] I. Järvinen and M. Kojo. "Evaluating CoDel, PIE, and HRED AQM Techniques with Load Transients". In: *LCN* (2014).
- [12] H. Jiang, Z. Liu, Y. Wang, K. Lee, and I. Rhee. "Understanding Bufferbloat in Cellular Networks". In: *CellNet*. ACM, 2012.
- [13] N. Khademi, D. Ros, and M. Welzl. "The New AQM Kids on the Block: An Experimental Evaluation of CoDel and PIE". In: *Global Internet Symposium Workshop* (2014).
- [14] N. Khademi, D. Ros, and M. Welzl. "The New AQM Kids on the Block: Much Ado About Nothing?" In: *Tech. Rep.* 434 (2014).
- [15] N. Kuhn, P. Natarajan, D. Ros, and N. Khademi. *AQM Characterization Guidelines*. IETF. 2014.
- [16] N. Kuhn, E. Lochin, and O. Mehani. "Revisiting Old Friends: Is CoDel Really Achieving What RED Cannot?" In: *ACM SIGCOMM Workshop CSWS*. 2014.
- [17] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar. *Controlled Delay Active Queue Management*. IETF. 2014.
- [18] K. Nichols and V. Jacobson. "Controlling Queue Delay". In: *ACM Queue* (2012).
- [19] R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. "PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem". In: *IEEE HPSR*. 2013.
- [20] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. "PIE: A lightweight control scheme to address the bufferbloat problem. Further Studies- PIE for Data Centers". In: *IETF 86*.
- [21] J. Schwardmann, D. Wagner, and M. Kühlewind. "Evaluation of ARED, CoDel and PIE". In: *20th Eunice Open European Summer School and Conference* (2014).
- [22] A. Sivaraman, K. Winstein, S. Subramanian, and H. Balakrishnan. "No Silver Bullet: Extending SDN to the Data Plane". In: *HotNets-XII*. 2013.

# Insights from Curvy RED (Random Early Detection)

Bob Briscoe\*

14 Aug 2015

## Abstract

Active queue management (AQM) drops packets early in the growth of a queue, to prevent a capacity-seeking sender (e.g. TCP) from keeping the buffer full. An AQM can mark instead of dropping packets if they indicate support for explicit congestion notification (ECN [RFB01]). Two modern AQMs (PIE [PNB<sup>+</sup>15] and CoDel [NJ12]) are designed to keep queuing delay to a target by dropping packets as load varies.

This memo uses Curvy RED and an idealised but sufficient model of TCP traffic to explain why attempting to keep delay constant is a bad idea, because it requires excessively high drop at high loads. This high drop itself takes over from queuing delay as the dominant cause of delay, particularly for short flows. At high load, a link is better able to preserve reasonable performance if the delay target is softened into a curve rather than a hard cap.

The analysis proves that the same AQM can be deployed in different parts of a network whatever the capacity with the same optimal configuration.

A surprising corollary of this analysis concerns cases with a highly aggregated number of flows through a bottleneck. Although aggregation reduces queue variation, if the target queuing delay of the AQM at that bottleneck is reduced to take advantage of this aggregation, TCP will still increase the loss level because of the reduction in round trip time. The way to resolve this dilemma is to overprovision (a formula is provided).

Nonetheless, for traffic with ECN enabled, there is no harm in an AQM holding queuing delay constant or configuring an AQM to take advantage of any reduced delay due to aggregation without over-provisioning. Recently, the requirement of the ECN standard [RFB01] that ECN must be treated the same as drop has been questioned. The insight that the goals of an AQM for drop and for ECN should be different proves that this doubt is justified.

\*ietf@bobbriscoe.net, Formerly of BT Research & Technology, UK, during the drafting of this report. Now independent.

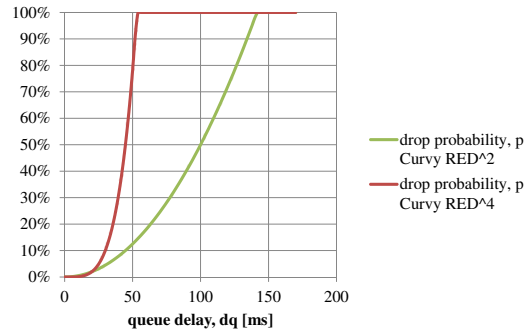


Figure 1: Two Example Curvy RED algorithms

## 1 Curvy RED

Curvy RED [DSBTB15] is an active queue management (AQM) algorithm that is both a simplification and a generalisation of Random Early Detection (RED [FJ93]). Two examples are shown in Figure 1 and Figure 2 shows a close-up of their normal operating regions.

The drop probability,  $p$ , of a Curvy RED AQM is:

$$p = \left( \frac{d_q}{D_q} \right)^u, \quad (1)$$

where  $d_q$  is averaged queue delay. The two parameters are:

$u$ : the exponent (cUrviness) of the AQM;

$D_q$ : the scaling parameter, i.e. the value of  $d_q$  when  $p$  hits 100%.  $1/D_q$  is the slope of the curve.

An implementation would not be expected to vary cUrviness,  $u$ . Rather an optimal value of  $u$  would be determined in advance.

The queuing delay used by Curvy RED is averaged, but averaging is outside the scope of his memo, which is only concerned with insights from steady-state conditions.

The scaling parameters  $D_q$  of the curves in Figure 1 are arranged so that they both pass through



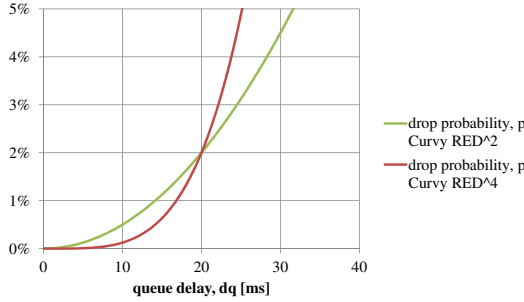


Figure 2: Usual Operating Region of The Same Two Example Curvy RED AQM algorithms

(20 ms, 2%), which we call the design point.<sup>1</sup> All the curves in Figure 2 and Figure 3 are arranged to pass through this same design point<sup>2</sup>, which makes them comparable over the operating region of about 0–40 ms either side of this point, shown in Figure 2.

In the following, the dependence of both drop and queuing delay on load will be derived. The approach is relatively simple. The rate of each flow has to reduce to fit more flows into the same capacity, the number of flows being a measure of increasing load. The formula for the rate of a TCP flow is well known; flow rate reduces if either drop or round trip time (RTT) increases, and RTT increases if queuing delay increases. The Curvy RED formula constrains the relationship between drop and queuing delay, so one can be substituted for the other in the TCP formula to cast it solely in terms of either drop or queuing delay, rather than both. Therefore either drop or queuing delay can be stated in terms of the number of flows sharing out the capacity.

Figure 3 shows the resulting dependence of both queuing delay (left axis) and drop probability (right axis) on load. There is a pair of curves for each of a selection of different values of curviness,  $u$ , in the Curvy RED algorithm. The rest of this section gives the derivation of these curves, and defines the normalised load metric used for the horizontal axis.

In practice, traffic is not all TCP and TCP is not all Reno. Nonetheless, to gain insight it will be sufficient to assume all flows are TCP Reno. Focusing on Reno is justified for bandwidth delay products typical in today's public Internet, where TCP Cubic typically operates in a TCP Reno emulation mode (e.g. with 20ms round trip time, Cubic re-

mains in Reno mode up to 500Mb/s). The following Reno-based analysis would apply for Cubic in Reno mode, except with a slightly different constant of proportionality. Even if the dominant congestion control were Cubic in its pure Cubic mode, the exponents and constants would be different in form, but the structure of the analysis would be the same.

In typical traffic a large proportion of TCP flows are short and complete while still in slow start. However, again for intuitive simplicity, only flows in congestion avoidance will be considered.<sup>3</sup>

The load on a link is proportional to the number of simultaneous flows being transmitted,  $n$ . If the capacity of the link (which may vary) is  $X$  and the mean bit-rate of the flows is  $x$ , then:

$$n = \frac{X}{x}. \quad (2)$$

The rate of a TCP flow depends on round trip delay,  $d_R$  and drop probability,  $p$ . A precise but complex formula for this dependence has been derived, but the simplest model [MSMO97] will suffice for insight purposes:

$$x = \frac{Ks}{d_R\sqrt{p}}, \quad (3)$$

where  $s$  is the maximum segment size (MSS) of TCP and  $K$  is  $\sqrt{3/2}$  for TCP Reno (or 1.68 for Cubic in Reno mode).

The round trip time,  $d_R$  consists of the base RTT  $D_R$  plus queuing delay  $d_q$ , such that:

$$d_R = D_R + d_q \quad (4)$$

For a set of flows with different base round trip times, it is sufficient to define  $D_R$  as the harmonic mean of the set of base RTTs.  $d_q$  is common to all the flows, because they all pass through the same queue<sup>4</sup>, so then  $d_R$  is the average RTT including this queuing delay  $d_q$ .

Substituting Equation 3 Equation 4 in Equation 2:

$$n = \frac{X(D_R + d_q)}{Ks} \sqrt{p}. \quad (5)$$

By substituting from Equation 1 into Equation 5, the number of flows can be given as a function of either queuing delay  $d_q$  or loss probability,  $p$ :

$$n = \frac{X(D_R + d_q)}{Ks} \left( \frac{d_q}{D_q} \right)^{u/2} \quad (6)$$

<sup>3</sup>If necessary, the traffic can be characterised as an *effective* number,  $n$ , of Reno flows in congestion avoidance, for instance by reverse engineering the output of an algorithm designed to measure queue variation  $\nu$  such as ADT [SSK06], i.e.  $n = (\text{BDP}/\nu)^2$ .

<sup>4</sup>Assuming the common case of a single bottleneck.

<sup>1</sup>The values are an arbitrary example, not a recommendation.

<sup>2</sup>By re-arranging Equation 1 as  $D_q = d_q^*/(p^*)^{1/u}$ , where the design point is  $(d_q^*, p^*)$ .

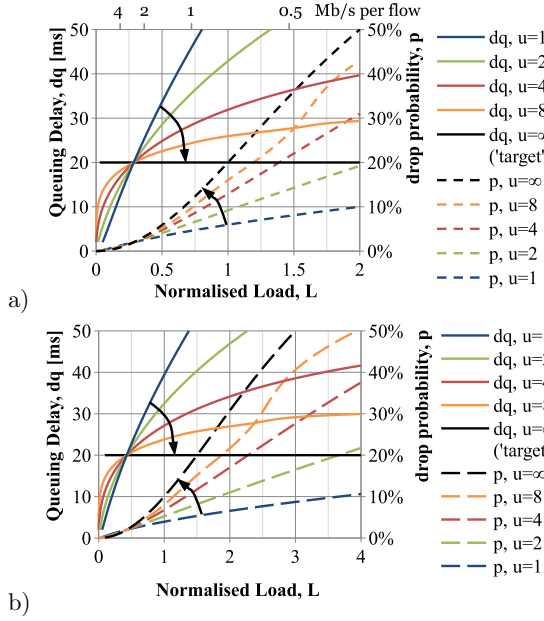


Figure 3: Dilemma between Two Impairments: Delay and Loss against Normalised Load; for TCP Reno and the Curvy RED Algorithm with Increasing Curviness,  $u$ ; a)  $D_R = 20$  ms; b)  $D_R = 10$  ms

$$n = \frac{X(D_R + D_q p^{1/u}) p^{1/2}}{Ks} \quad (7)$$

Equation 6 & Equation 7 are plotted against normalised load in Figure 3 for an example set of Curvy RED algorithms with curviness parameters  $u = 1, 2, 4, 8, \infty$ , with base RTT  $D_R = 20$  ms.  $u = \infty$  represents an algorithm like PIE or CoDel that attempts to clamp queuing delay to a target value whatever the load.

How to read Figure 3: For any one value of  $u$ , only two plots apply. Each pair shares a common colour. The solid line in each pair shows growth in queuing delay (left axis) with load and the dashed line shows growth in drop (right axis). For instance, for  $u = 1$ , the two blue plots apply, which are uppermost and lowermost. As the value used for  $u$  increases, the relevant pair of plots to use progress inwards; down from the top and up from the bottom, like a closing pair of pincers.

Normalised load  $L$  is used as the horizontal axis, because it is proportional to the number of flows  $n$  but scales with link capacity  $X$ , by the following relationship:

$$L = \frac{Ks}{D_R} \frac{n}{X}, \quad (8)$$

where  $Ks/D_R$  can be considered constant, at least as long as MSS and the typical topology of the Internet are constant. The plot uses  $s = 1500$  B.

It can be seen that, as the curviness parameter is increased, the AQM pushes harder against growth of queuing delay as load increases. However, given TCP is utilising the same capacity, it has to cause more loss (right axis) if it cannot cause more queuing delay (left axis).

In the extreme, infinite curviness represents the intent of AQMs such as CoDel and PIE that aim to clamp queuing delay to a target value (the black horizontal straight line). As a consequence, the TCP flows force loss to rise more quickly with load (the black dashed line). The experiments comparing Curvy RED with RED, PIE and fq-CoDel in [DSBTB15] in a realistic broadband testbed verify this analysis, at least for  $u = 2$ . Indeed the high drop levels found with PIE and fq-CoDel in these experiments motivated the analysis in this report.

To reduce losses at high load, the default target delay of fq-CoDel probably needs to be increased to a value closer to PIE's target default of 20 ms. However then, when load is below the design point, both PIE and CoDel will tend to allow a small standing queue to build so that the queue is sufficient to reach the target delay. Whereas Curvy RED should give lower queuing delay when load is light.

Normalised load is plotted on the horizontal axis to make the plots independent of capacity as long as individual flows have the same rate. As an example, we will set the constants to average base RTT,  $D_R = 20$  ms and MSS,  $s = 1500$  B. Then, if the average rate of individual flows,  $x = 1$  Mb/s normalised load will be  $L = 1500 * 8 * \sqrt{3/2} / (0.02 * 1E6) = 0.73$ , whether there are  $n = 4$  flows in  $X = 4$  Mb/s of capacity, or  $n = 400$  flows in  $X = 400$  Mb/s.

From Equation 8 it can be seen that normalised load depends on the choice of average base delay used to characterise all the paths,  $D_R$ . The horizontal scale of Figures 3a) and 3b) has been contrived so that a vertical line through them both represents the same mean rate per flow,  $x$ .<sup>5</sup> A few selected values of rate per flow are shown along the top of Figure 3a).

For instance, if  $D_R = 20$  ms then  $x$  is 1 Mb/s when  $L = 0.73$ , but if  $D_R = 10$  ms,  $x$  is 1 Mb/s when  $L = 1.46$ . At twice the normalised load in both plots,  $x$  will halved. As well as this scaling, there is still residual dependence on  $D_R$  in the shape of the plots as well, even though normalised load is defined to include average RTT  $D_R$ .

<sup>5</sup>Interestingly, normalised load can be expressed as proportionate to the product of number of flows and the ratio of two delays:  $L = nKD_s/D_R$ , where  $D_s$  is the serialisation delay of a maximum sized segment, because  $D_s = s/X$ .

## 2 Invariance with Capacity

### 2.1 Does Flow Aggregation Increase or Decrease the Queue?

There used to be a rule of thumb that a router buffer should be sized (in bytes) for the bandwidth delay product. Then, in 2004 Appenzeller et al. [AKM04, GM06] pointed out that for large aggregates  $\text{BDP}/\sqrt{n}$  would be sufficient. This applies only if TCP's sawteeth are desynchronized so that the variance of all the sawteeth shrinks with the square root of the number of flows.<sup>6</sup>

A link for  $100\times$  more flows will typically be sized with  $100\times$  more capacity, so the queue will drain  $100\times$  faster. Therefore, if the buffer is sized for  $\text{BDP}/\sqrt{n}$ , queuing delay will be  $\sqrt{n}\times$  ( $=10\times$ ) lower.<sup>7</sup>

In the previous section we showed that queuing delay *grows* with number of flows. How does this reconcile with the idea that buffers can *shrink* as more flows are aggregated?

It is certainly true that, with more flows, variability of the queue will shrink. So, if curvy RED allows the extra flows to increase the queue, most of the variation will be at the tail of the queue in front of a small standing queue at the head. However, without more capacity, the alternative would be for the AQM to introduce more drop to prevent the additional delay—the dilemma in Figure 3 cannot be escaped.<sup>8</sup>

Nonetheless, if the greater number of flows is expected to persist, capacity can be increased. This creates enough space again for each flow, so less queuing delay and less loss is needed. For example, if there are permanently ten times more flows, then provisioning ten times more capacity will bring *normalised* load back to its original value, because normalised load,  $L \propto n/X$ .

Note that our definition of normalised load also factors out a change in MSS. For a given capacity and number of flows, increasing the MSS increases the normalised load. It may be counter-intuitive that the same bit-rate in larger packets will change normalised load. This is an artefact of the design of the TCP Reno algorithm (and most other TCP variants), which congests the link more (more delay and/or drop) for a larger MSS, because it adds one MSS to the load per RTT (the additive increase).

<sup>6</sup>This square root comes from the Central Limit Theorem and has nothing to do with the TCP Reno rate equation.

<sup>7</sup>Note that this square-root law does not apply indefinitely; only for medium levels of aggregation. Once the queue size has reduced to about half a dozen packets, it does not get any smaller with increased aggregation [GM06].

<sup>8</sup>except using ECN —see § 3

### 2.2 AQM Configuration and Scale

AQM configuration should be invariant with link capacity, but it will need to change in environments where the expected range of RTTs is significantly different. Therefore, AQM configurations in a data centre will be different, not because of high link capacities, but because of shorter RTTs.

The parameter  $D_q$  represents a delay well outside the normal operating region, so it has no intuitive meaning. Therefore it is better to configure the AQM against a design point, such as  $d_q = 20\text{ ms}$ ,  $p = 2\%$ . The operator will need to set this design point where delay and loss start to become troublesome for the most sensitive applications (e.g. interactive voice). By setting the AQM to pass through this point, it will ensure a good compromise between delay and loss. Then the most sensitive application will survive at the highest possible load, rather than holding one impairment unnecessarily low so that the other is pushed so high that it breaks the sensitive app.

The ideal curviness of the AQM depends on the dominant congestion control regime (this memo is written assuming TCP Reno is dominant, which includes Cubic in Reno mode). For a particular dominant congestion control, curviness depends only on the best compromise to resolve the dilemma in Figure 3, and can otherwise be assumed constant.

Having determined an AQM configuration as above, it will be applicable for all link rates as long as expected round trip times are unchanged. It would seem that higher flow aggregation would allow queuing delay to be reduced proportionate to  $1/\sqrt{n}$  without losing utilisation. However, *the dilemma in Figure 3 still applies, so loss probability would increase*. Therefore, once a good balance between queuing delay and loss has been determined it will be best to stick with that for any level of aggregation. Then, on a highly aggregated link, in comparison to an equivalent level of normalised load on a smaller link with fewer flows, queue variation will be smaller but it will have to sit behind a larger standing queue. Nonetheless, this is preferable to changing the AQM scaling parameter to trade more loss for less queuing.

This should help to explain why Vu-Brugier *et al* [VBSLS07] found that they could not reduce the size of a buffer to  $\text{BDP}/\sqrt{n}$  without triggering high volumes of user complaints due to excessive loss levels.

If there are too many flows for the capacity, no amount of AQM configuration will help. Capacity needs to be appropriately upgraded. The following formula can be used to determine capacity  $X$  given



the expected number of flows  $n$  and expected average base RTT  $D_R$ , where  $d_q^*$  and  $p^*$  are the values of queuing delay and drop at the design point, as determined above.

$$X = \frac{Kns}{(D_R + d_q^*)\sqrt{p^*}}.$$

Alternatively, the formula can be used to determine the number of flows  $n$  that a particular capacity  $X$  can reasonably be expected to support.

For a bottleneck link, the only way to take advantage of the  $1/\sqrt{n}$  reduction in queuing delay due to aggregation without increasing loss is to over-provide capacity, so that  $n/X$  reduces as much as  $(D_R + d_q^*)$  reduces, thus holding  $p^*$  unchanged.

The necessary degree of overprovisioning can be calculated for the worst case where the overprovisioned link remains as the bottleneck. In this case, the design point for queuing delay at low aggregation,  $d_q^*$  is reduced to  $d_q^*/\sqrt{n}$  at high aggregation, then the over-provisioning factor should be:

$$\frac{X'}{X} = \frac{(D_R + d_q^*)}{(D_R + d_q^*/\sqrt{n})}.$$

For example, if average base RTT,  $D_R = 20ms$  a link for  $100\times$  more flows could have the design point for queuing delay reduced from 20 ms to 2 ms while keeping the loss design point constant at 2%. But only if aggregated capacity is over-provided by  $(20 + 20)/(20 + 2) = 1.8$ , i.e.  $180\times$  more capacity for  $100\times$  more flows.

If a core link is over-provisioned such that it is rarely the bottleneck, none of this analysis applies. Its buffer can be sized at  $BDP/\sqrt{n}$  without sacrificing utilisation but accommodating queue variation due to TCP flows bottlenecked elsewhere, and no regard needs to be taken of the risk of loss, which will be vanishingly small.

### 3 Escaping the TCP Dilemma with ECN

Another insight that can be drawn from this analysis is that the dilemma in Figure 3 disappears with ECN. For ECN, the AQM can mark packets without introducing any impairment. There is therefore no downside to clamping down queuing delay for ECN-capable packets. This proves that it is wrong to treat ECN the same as drop.

When ECN was first standardised [RFB01], it was defined as equivalent to drop. Even though earlier proposals had considered treating ECN differently, it had to be standardised as equivalent otherwise

no consensus could be reached on how to define its meaning.

It is now known that drop probability should be proportional to the square of ECN marking, and the constant of proportionality need not be standardised [DSBTB15]. Therefore it might now be feasible to reach consensus on a standard meaning for ECN different from but related to drop.

## 4 Conclusions & Further Work

Although Curvy RED seems to be a useful AQM, we are not necessarily recommending it here. We are merely using the concept of curviness to draw insights.

The curviness parameter of Curvy RED can be considered to represent the operator's policy for the tradeoff between delay and loss whenever load exceeds the intended design point. In contrast, PIE and RED embody a hard-coded policy, which dictates that holding down delay is paramount, at the expense of more loss.

Given losses from short interactive flows (e.g. Web) cause considerable delay to session completion, trading less queuing delay for more loss is unlikely to be the optimal policy to reduce delay. Also, as load increases, it may lead real-time applications such as conversational video and VoIP to degrade or fail sooner. Allowing some additional flex in queueing delay with consequently less increase in loss is likely to give more favourable performance for a mix of Internet applications.

This dilemma can be escaped by using ECN instead of loss to signal congestion. Then queuing delay can be capped and the only consequence is higher marking, not higher drop.

The analysis also shows that, once a design point defining an acceptable queuing delay and loss has been defined, the same configuration can be used for the AQM at any link rate, but only for a similar RTT environment. A shallower queuing delay configuration can be used at high aggregation, but only if the higher loss is acceptable, or if the capacity is suitably over-provisioned. Formulae for all these configuration trade-offs have been provided.

We intend to conduct experiments to give advice on a compromise level of curviness that best protects a range of delay-sensitive and loss-sensitive applications during high load. It will also be necessary to verify the theory for all values of curviness, and for other AQMs without their default parameter settings.

Further research is needed to understand how best to average the queue length, and how best to configure the averaging parameter.

## 5 Acknowledgements

Thanks to Koen De Schepper of Bell Labs for suggesting to use the harmonic mean of the base RTTs, rather than assuming equal RTT flows. Thank you to the following for their review comments: Wolfram Lautenschläger and the participants in the RITE project, especially, Gorry Fairhurst, Michael Welzl, Anna Brunström, David Hayes, Nicolas Kuhn and Naeem Khademi.

The author's contribution is part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed here are solely those of the authors.

## References

- [AKM04] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing Router Buffers. *Proc. ACM SIGCOMM'04, Computer Communication Review*, 34(4), September 2004.
- [DSBTB15] Koen De Schepper, Olga Bondarenko, Ing-Jyh Tsang, and Bob Briscoe. 'Data Centre to the Home': Ultra-Low Latency for All. June 2015. (Under Submission).
- [FJ93] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.
- [GM06] Yashar Ganjali and Nick McKeown. Update on Buffer Sizing in Internet Routers. *ACM SIGCOMM Computer Communication Review*, 36, October 2006.
- [MSMO97] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithms. *SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, 1997.
- [NJ12] Kathleen Nichols and Van Jacobson. Controlling queue delay. *ACM Queue*, 10(5), May 2012.
- [PNB<sup>+</sup>15] Rong Pan, Preethi Natarajan, Fred Baker, Bill Ver Steeg, Mythili Prabhu, Chiara Piglion, Vijay Subramanian, and Greg White. PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem. Internet Draft draft-ietf-aqm-pie-01, Internet Engineering Task Force, March 2015. (Work in progress).
- [RFB01] K. K. Ramakrishnan, Sally Floyd, and David Black. The Addition of Explicit Congestion Notification (ECN) to IP. Request for Comments 3168, Internet Engineering Task Force, September 2001.
- [SSK06] R. Stanojevic, R. Shorten, and C. Kellett. Adaptive tuning of Drop-Tail buffers for reducing queueing delays. *IEEE Comm. Letters*, 10(7), July 2006.
- [VBSLS07] G. Vu-Brugier, R. S. Stanojevic, D. J. Leith, and R. N. Shorten. A Critique of Recently Proposed Buffer-sizing Strategies. *Computer Communication Review*, 37(1):43–48, January 2007.

## Document history

Version	Date	Author	Details of change
Draft 00A	19 May 2015	Bob Briscoe	First Draft
Draft 00B	23 May 2015	Bob Briscoe	Explained normalised load
Draft 00C	23 May 2015	Bob Briscoe	Changed notation & added Aggregation section
Draft 00D	08 Jun 2015	Bob Briscoe	Corrected explanation about packet size, plus minor corrections
Draft 00E	10 Jun 2015	Bob Briscoe	Cited DCttH.
Issue 01	25 Jul 2015	Bob Briscoe	Generalised from equal flows; Included Base RTT in Normalised Load; Changed $s$ from packet size to MSS. Justified standing queue due to flow aggregation.
Issue 01A	26 Jul 2015	Bob Briscoe	Editorial Corrections.
Issue 01B	29 Jul 2015	Bob Briscoe	Clarified applicability and more comprehensive advice on scaling AQM config.
Issue 01C	14 Aug 2015	Bob Briscoe	Corrections.

# Review: Proportional Integral controller Enhanced (PIE) Active Queue Management (AQM) [PNB<sup>+</sup>15]

Bob Briscoe\*

09 May 2015

## Main Concerns

The following summary of my concerns, can be used as a table of contents for the rest of the review:

1. Proposed Standard, but no normative language
  1. work needed to distinguish between design intent and specific implementation
  2. unclear how strongly the enhancements are recommended
2. Has PIE been separately tested with and without each enhancement, to justify each?
3. Needs to enumerate whether it satisfies each AQM Design Guideline
  1. If not, say why or fix.
  2. Particular concerns:
    - i. No spec of ECN behaviour
    - ii. No autotuning of the two main parameters
    - iii. Transport specific (Reno-based?) autotuning of  $\alpha$  &  $\beta$
4. Rationale for a PI controller not properly articulated
5. Technical flaws/concerns
  1. Turning PIE off
  2. 'Autotuning'  $\alpha$  &  $\beta$  parameters
  3. Averaging problems
  4. Burst allowance unnecessary?
  5. Needs a Large Delay to Make the Delay Small
  6. Derandomization: a waste of cycles
  7. Bound drop probability at 100%  $\rightarrow$  DoS vulnerability?
  8. Avoiding Large Packet Lock-Out under Extreme Load.
6. Numerous magic numbers
  1.  $\sim 20$  constants, 13 of which are not in the header block.
  2. About half ought to be made to depend on other constants
  3. Need to state how to set the remaining constants for different environments
7. Implementation suggestion for Autotuning  $\alpha$  &  $\beta$

---

\*[ietf@bobbriscoe.net](mailto:ietf@bobbriscoe.net), BT Research & Technology, B54/77, Adastral Park, Martlesham Heath, Ipswich, IP5 3RE, UK

My technical points run to 15 pages. The review ends with a 7 page tailpiece of editorial nits, references, etc.

The PIE draft ends with two assertions in the Discussion section:

“PIE is simple to implement”

“PIE does not require any user configuration”

I am afraid I have to say that I do not believe either statement is warranted any more. PIE has lost its way a bit. The implementation has not retained the elegance of the theory. The performance benefit from so-called ‘enhancements’ is questionable or non-existent, whereas the added complexity is very apparent. Also PIE now contains a large number of hard-coded constants (I counted 20) that ought to be scenario-dependent configuration variables.

## 1 Proposed Standard, but No Normative Language

See email thread “PIE (and CoDel) drafts: proposed standard vs informational?” (It seems like it would not be a big issue to make the PIE draft informational)

If normative language were included, we’d need to consider, “What is the essence of PIE?” My suggestions:

- MUST calculate drop probability as  $p = p + \alpha(\dots) + \beta(\dots)$
- MAY `est_del = qlen/depart_rate`  
Alternative calculations are discussed:
  - solely at enqueue
  - using timestamps at dequeue
- MUST turn on & off
- MUST autotune  $\alpha$  &  $\beta$
- MUST handle bursts for drop, by default SHOULD NOT for ECN
- MAY derandomize, but only for per-flow queuing (see later)

And, in each case, what is the outcome that MUST be achieved, not just the RECOMMENDED formula for the the reference implementation?

## 2 Justify Each Enhancement

Each enhancement is (mostly) justified theoretically, which is good. But has PIE been separately tested with and without each enhancement to validate the theory? For instance:

- with and without derandomization?
- with different burst sizes?
- with and without autotuning  $\alpha$  &  $\beta$ ?

## 3 AQM Recommendations

Check whether PIE satisfies all the requirements in the Recommendations [BF15], and if it doesn’t comply with the SHOULDs, say why. I noticed a few specific “non-compliances”:

- Recommendation 4.2.1. “AQM and ECN”
- Recommendation 4.3 Autotuning
- Recommendation 4.5 Transport-Independent?

### 3.1 Recommendation 4.2.1. “AQM and ECN”

“Network devices SHOULD use an AQM algorithm that marks ECN-capable traffic”

In the PIE draft, there are three mentions of marking or dropping in general sections, but only dropping is described in the specific sections on the PIE algorithm.

The AQM Recommendations draft [BF15] also says that ECN parameters SHOULD be separately configurable, so the PIE draft ought to say which specific parameters would be separately configurable for ECN. IMO, at least burst size should be separately configurable for ECN (irrespective of whether ECN and non-ECN flows are queued separately or together).

### 3.2 Recommendation 4.3 Autotuning

“AQM algorithm deployment SHOULD NOT require operational tuning”

However, the PIE draft has not attempted this. Indeed, when the Introduction discusses PIE parameters (`target_del` and `max_burst`?), it only says could, not should:

```
"While these parameters can be fixed to work
  in various traffic conditions, they could be made self-tuning to
  optimize system performance.
"
```

### 3.3 Recommendation 4.5 Transport-Independent?

“AQM algorithms SHOULD NOT be dependent on specific transport protocol behaviours”

I believe the autotuning factors in the lookup table in the appendix of the IETF draft (`status_update()` block) are specific to TCP Reno. Whether they are or not, the draft needs to say how they were derived. And how they would be derived if conditions change, e.g. Cubic (or even rmcat!) becomes dominant instead of Reno.

Most importantly, we need to know whether these values can be hard-coded for ever, or whether they must be configurable.

These factors were originally derived using the theory in the HPSR’13 paper on PIE [PPP+13]. However, as shown in the table below, the factors in the pseudocode are different from the paper (which used Misra *et al*’s model of TCP Reno to derive them). The draft still says it is based on the same model of TCP Reno, so why are the factors different? If the analysis has been generalised since the HPSR’13 paper, we need to know what logic these new choices are based on.

$p$	factor in I-D [PNB+15]	factor in original HPSR paper [PPP+13]
< 0.1%	1/128	
< 1%	1/16	1/8
< 10%	1/2	1/2
else	1	1

BTW, there’s a mistake in the theoretical analysis section of the HPSR’13 paper on PIE [PPP+13], so the effect on the tuning factors in the draft will need to be checked:

CURRENT:

$$\kappa = \alpha R_o / (p_o T)$$

SHOULD BE:

$$\kappa = 2\alpha R_o / (p_o T)$$

BTW2, the control law in CoDel is also specific to TCP Reno.

## 4 Articulate the Rationale for a PI Controller

The draft doesn't actually say that a PI controller is chosen to keep the average latency constant even under high load. The 3rd para of Intro says that, unlike RED's use of queue length, PIE uses latency as the metric. However, that alone could just mean that the latency still increases with load.

All the rationale in section 4.2 doesn't state this fundamental reasoning either, except by reference, citing the original paper on PI [HMTG01].

## 5 Technical Flaws/Concerns

### 5.1 Turning PIE Off

My concern is about buffers that are more often completely idle than active (e.g. in most access networks). PIE sets itself to `inactive` when `qdelay` has been less than `QDELAY_REF` for long enough that `drop_prob` has reduced to zero.

But it seems that PIE will not reduce `drop_prob` unless there is at least a low rate of packet arrivals. I think PIE is blind to any period when there are no packets at all, and just picks up where it left off the next time a packet does arrive.

I am unsure, because it is not clear from the pseudocode where the `status_update()` routine is called from. Is it a parallel background process, or triggered by packet arrivals? At the start of `status_update()`, it tests whether `T_UPDATE` has been reached, which implies that it is called on each packet arrival (because this test would not be necessary if it were triggered by a timer interrupt).

If the timer check in `status_update` is meant to be triggered by packet arrivals, when traffic just stops at the end of user activity, PIE will stop updating `drop_prob`. Then when some time later the user becomes active again, PIE tests whether it should be in the active state by whether `drop_prob` is zero. But nothing has triggered updates to `drop_prob` in the meantime. So PIE will fool itself into thinking it is in the active state and pick up all the variables where it left off, perhaps hours ago.

If this is corrected by making `status_update()` into a timer-triggered process, then care will be needed to ensure it doesn't keep a machine awake that would otherwise go into an energy-saving sleep mode when it has been inactive for a while.

### 5.2 'Autotuning' $\alpha$ & $\beta$ Parameters

The following lookup table is abbreviated from that in the `status_update()` code block:

```
if (p < 0.1%) {
    p += [alpha*(qdelay - QDELAY_REF) + beta*(qdelay-PIE->qdelay_old_)]/128;
} else if (p < 1%) {
    p += [alpha*(qdelay - QDELAY_REF) + beta*(qdelay-PIE->qdelay_old_)]/16;
} else if (p < 10%) {
    p += [alpha*(qdelay - QDELAY_REF) + beta*(qdelay-PIE->qdelay_old_)]/2;
} else {
    p += alpha*(qdelay - QDELAY_REF) + beta*(qdelay-PIE->qdelay_old_);
}
```

I am concerned that using a look-up table is not 'autotuning'. It does not tune beyond the bounds of the table. I.e. everything below 0.1% (the lowest value) will have to make do with division by 128 (see Fig 1). To properly cover the operational range of values of  $p$  would require far too many "else ifs" in sequence.

For instance a single Cubic flow with RTT of 100 ms bottlenecked in a link at the rates below causes the `drop_prob` shown.

6 Mb/s	0.1%	
12 Mb/s	0.04%	←the table in the draft stops about here!
24 Mb/s	0.016%	
48 Mb/s	0.0065%	
96 Mb/s	0.0025%	
192 Mb/s	0.0010%	
384 Mb/s	0.00040%	
768 Mb/s	0.00016%	

So the table in the PIE code only covers up to about 12 Mb/s, which might represent some upstream access links today, but it is missing nearly two orders of magnitude for even the normal residential downstream rates sold today (e.g. 80 Mb/s).

1 Gb/s access links are fairly normal today for private networks, and PIE should be applicable on aggregated links too, as well as in data centres where 10 Gb/s is becoming the norm.

Further, equipment implemented today will probably sit in the network for 10–20 years. Access link rates have typically doubled every 1.6 years. So, in 10–20 years, link rates might be  $75\times$ – $6000\times$  their rates today.

Anyway, that look-up table is surely unnecessary. Below I’ve repeated the look-up table of  $p$  against ‘factor’ from the draft and added more columns to show that the factors in the look-up table approximate to a linear relationship with  $p$  (compare  $p_{\text{mid}}$  and  $\text{factor}/16$ ). This is not surprising because the draft says that the factors were chosen so that the changes in  $p$  were proportionate to the size of  $p$  before the change.

$p$	$p_{\text{mid}}$	$p_{\text{mid}}$	<b>factor</b>	<b>factor/16</b>
<0.1%	0.03%	1/3333	1/128	1/2048
<1%	0.3%	1/333	1/16	1/256
<10%	3%	1/33	1/2	1/32
<b>else</b>	30%	1/3	1	1/16

where:

$p$  : PIE→drop\_prob

$p_{\text{mid}}$  : the (geometric) mid-points of each range, as a percentage  
and as a fraction

**factor** : The factor in the “if else” look-up table in the pseudocode, repeated at the start of this section.

So, why not just set

$$\text{factor} = p * \text{ALPHA\_DEFAULT}$$

with  $\text{ALPHA\_DEFAULT} = 16$ ? It’s a decent approximation of the look-up table of factors in the draft, particularly given the draft says that the exact values of the factors are not critically sensitive. Importantly, it continues indefinitely as  $p$  gets smaller, whereas the look-up table approach always stops somewhere (see Fig 1).

Previously, each factor was chosen as a power of two, so multiplying it with the “ $\alpha(\dots) + \beta(\dots)$ ” expression could be done with just adds and bit-shifts. Even though ‘factor’ now seems to have to be a double, this multiplication can still be done with just adds and bit-shifts (see “Implementation Suggestions” later).

Note: My colleague Koen de Schepper reckons the factors in the pseudocode reduce too fast (see my earlier question about why the factors in the draft reduce faster than those in the HPSR’13 paper). If he’s right, whether we use the factors in the draft or my approximation to them, PIE will be too sluggish at high bit-rates. If the draft authors agree that the factors should reduce in smaller steps, then the approach I propose above will not be applicable. However, there would be a different solution.



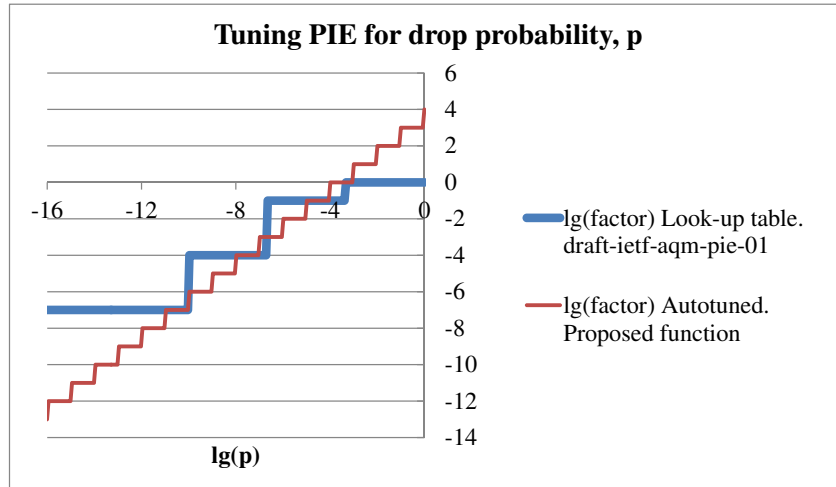


Figure 1: Comparison of Proposed Autotuning Function with Look-Up Table Approach

### 5.3 Averaging

#### 5.3.1 Queue Sampling Rate

The queue is only sampled and  $p$  is only updated once every  $T\_UPDATE$  ( $= 16\text{ms}$  by default). This might be appropriate for properly ACK-clocked or paced long-running flows. However, more bursty traffic is often the norm, with lots of short flows arriving at irregular intervals. In such bursty scenarios, the queue will tend to vary rapidly, so numerous samples will be needed to characterise the average queue. By spacing samples so widely apart, it seems likely to take a very long time for  $p$  to reach an appropriate value, by which time it will probably no longer be appropriate.

PIE's sampling rate might be appropriate to minimise processing on cost-reduced equipment like a residential gateway for a link rate of say  $8\text{Mb/s}$  (it then equates to about one in ten 1500 B packets). But, even then, it seems unnecessarily infrequent. The draft should explain the tradeoffs that were made before settling on this sampling rate, and whether evaluations have shown that a shorter sampling time (if feasible) would be beneficial?

The draft says “Given modern high speed links, this period translates into once every tens, hundreds or even thousands of packets.” However, it seems that the benefit of reducing background processing beyond tens of packets becomes insignificant compared to the sloppiness of such infrequent updates to  $p$ .

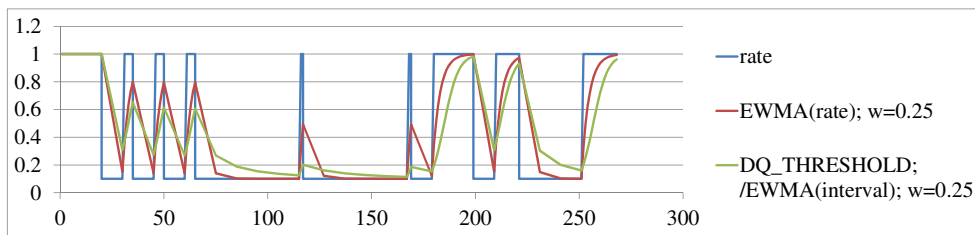
#### 5.3.2 Averaging is Not Exponentially Weighted

In the `set_status()` block of the pseudocode (or Section 4.2 of the body of the draft), the queuing delay is determined as follows:

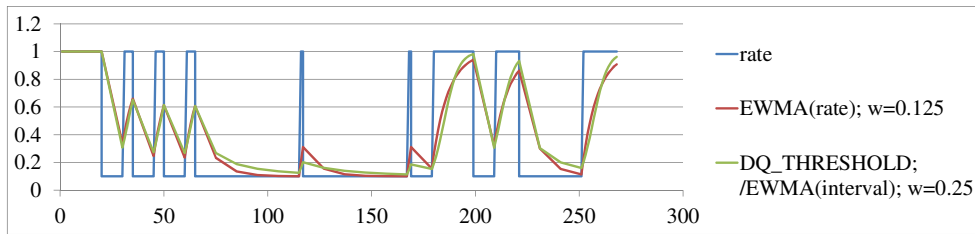
```
qdelay = queue_.byte_length() * avg_dq_time_/DQ_THRESHOLD;
```

which is explained (in Section 4.2) as  $qlen / \text{depart\_rate}$ , where

```
depart_rate = DQ_THRESHOLD/avg_dq_time_
```



(a) Equal weights



(b) EWMA weight adjusted to attempt to match the non-EWMA

Figure 2: Exponentially Weighted Moving Average vs. PIE's Non-EWMA Approach for an Example Sequence of Rate Variations

However, taking an exponentially weighted moving average (EWMA) of the dequeue times in the denominator is not the same as taking the EWMA of the rates. The pseudocode continually measures the sequence of dequeue times,  $t_1, t_2, t_3, \dots$  that it takes to transmit constant amounts of bytes (DQ\_THRESHOLD). Then the exponentially weighted moving average (EWMA) of the rate should be:

$$\begin{aligned} \text{ewma}(\text{depart\_rate}) &= \text{DQ\_THRESHOLD} * \text{ewma}(1/t_1, 1/t_2, 1/t_3, \dots) \\ &\neq \text{DQ\_THRESHOLD} / \text{ewma}(t_1, t_2, t_3, \dots) \end{aligned}$$

Fig 2 compares a correctly calculated EWMA with the non-EWMA in the PIE draft. They are both shown averaging the departure rate of a link that varies between 1 unit and 0.1 units for the same sequence of example durations. For the record, the formula for the correct EWMA is:

$$\text{avg\_depart\_rate} \leftarrow [1 - (1 - a)^{1/\text{dq\_time}}] * \text{dq\_time} + (1 - a)^{1/\text{dq\_time}} * \text{avg\_depart\_rate},$$

where  $a$  is the weighting constant of the EWMA. This correct EWMA formula has solely been used to compare it with the PIE approach; it is not intended to be used in the PIE code, because it is too complex—more complex than a typical EWMA, because the times between each measurement event depend on the measurement.

The top chart (2a) shows that the PIE approach is considerably slower to converge than the correct EWMA with the same weight (1/4 in both cases). In the bottom chart (2b), the weight of the correct EWMA has been scaled down to 1/8 in an attempt to match the outcome of the PIE code. Looking closely it can be seen that the PIE approach is faster than exponential at first, then slower than exponential later. This explains why, whatever constant scaling is chosen, even if one part of the trace can be made to match, other parts will fail to match.

It would be hard to implement a correct EWMA unless constant time periods were used instead of constant amounts of bytes. It might still be acceptable to use the PIE formula as an approximation. However, it should be considered whether there might be pathological scenarios where the error would be unacceptable.

(See also the comment at the end of §6 about how the EWMA constant should be set relative to DQ\_THRESHOLD.)

## 5.4 Burst allowance unnecessary?

The recommended BURST\_ALLOWANCE of 150 ms seems both very large and unnecessary. For typical interactions with CDNs placed about 10 ms away, this will suppress any response from PIE for about 15 round trips. PIE’s ‘silence period’ occurs every time PIE starts after an application-limited or idle period, which is a very common occurrence in access links dedicated to single customers.

A burst allowance is intended to ensure that bursts that appear and disappear of their own accord do not get hit unnecessarily by a loss. Bursts that “disappear of their own accord” means bursts due to coincidences in arrivals from multiple flows. It does not mean bursts that TCP removes after it receives round trip feedback. The AQM is meant to respond faster than TCP, because TCP is meant to respond to the signals from the AQM. If the AQM responds more slowly than TCP’s round trip time response, TCP will have to either drive up queuing delay or drive the queue to drop-tail in order to get a signal to respond to.

As the draft says, the PIE algorithm without a burst allowance already naturally filters its response to bursts, for two reasons: i) it misses bursts within the queue sampling period T\_UPDATE; ii) its drop probability only evolves in small steps in response to changing queue length.

This seems the best way to filter bursts, because the larger and longer the burst, the less the response to it will be filtered out. Introducing a fixed period of 150 ms during which PIE is completely unable to send any signals, no matter how large the burst, seems very wrong, especially given PIE already has a good filtering mechanism that self-tunes to the traffic.

Given the draft recommends setting BURST\_ALLOWANCE to 150 ms, it should at least point to experiments that explain why this choice gives good performance compared to: i) a smaller burst allowance; ii) no burst allowance; or iii) no burst allowance and a less aggressive (lower) value for **beta** (and perhaps also for **alpha**). This will at least allow operators to judge whether the experiments are appropriate to the deployment in question (e.g. appropriate round trip times and appropriate idle periods in the traffic).

## 5.5 Needs a Large Delay to Make the Delay Small

The departure rate measurement only works if the queue grows larger than  $2^{14}$  B. For even a medium rate broadband uplink (8 Mb/s), this represents 16 ms delay. So, the queue has to reach 16 ms of delay for the algorithm to start working. And unless the target delay is set as high as 16 ms, PIE cannot maintain a queue large enough to update its own rate estimation. Therefore, it seems that the constraints of rate estimation set a lower bound on the target delay, which is not a healthy reason for choosing a target delay value.

For a 2 Mb/s link (the lower bound of the definition of broadband), PIE has to build a 66 ms queue in order to estimate its departure rate.

## 5.6 Derandomization

I've said this before on this list about the derandomization in RED, but perhaps it was not understood...

Unless there is only one flow in the queue (e.g. per-flow queuing), derandomizing the spacing between drops just wastes cycles. So, if PIE is in a shared FIFO queue, the derandomization routine might as well be called `heat_the_room()`.

There's a formal proof of this in my PhD dissertation [Bri09, §7.7.1], but here's the intuition. Imagine 2 equal but desynchronized flows A & B are feeding the queue. Even if PIE makes the spacing between drops regular, some drops will hit flow A and some will hit flow B. As long as the flows are not synchronized, PIE will not drop alternately from A,B,A,B,A,B, etc. The flow that is hit by each drop will be randomized. So the spacing between drops within each flow will no longer be regular, even though the spacing between drops in the whole aggregate is regular.

If this is still not obvious, here's a numerical example, with 4 desynchronized but equal flows A,B,C&D. For illustration, let's make the derandomization perfect and I will show that it becomes nearly random again when one considers the spacing in each flow rather than in the aggregate.

Let's say drop probability = 1% and derandomization is so perfect that it drops packet number 50, 150, 250, etc. Over such a distance between drops, it is reasonable to assume that there will be an equal chance of hitting any of the flows.

The drop at packet#50 hits flow A with chance 1 in 4, and misses it with chance 3 in 4.  
 The drop at packet#150 hits flow A with chance 1 in 4, and misses it with chance 3 in 4.  
 The drop at packet#250 hits flow A with chance 1 in 4, and misses it with chance 3 in 4.  
 etc.

Let's assume one of these drops has hit flow A at packet#X.

Then the chance of a drop from flow A:

at #X	AND #X+100 =	1/4 = 25.00%
at #X AND NOT #X+100	AND #X+200 = $(3/4)^1 * 1/4$	= 18.75%
at #X AND NOT #X+100 AND NOT #X+200	AND #X+300 = $(3/4)^2 * 1/4$	= 14.06%
at #X AND NOT #X+100 AND NOT #X+200 AND NOT #X+300	AND #X+400 = $(3/4)^3 * 1/4$	= 10.55%

But, for flow A, there are only *about*  $100/4 = 25$  packets between #50 and #150, because the other  $\sim 75$  packets are in the other flows.

So, in general, the chance of about  $25 * n$  packets between drops in one of the flows is

$$1/4 * (3/4)^{(n-1)}$$

So, this so-called 'perfect' derandomization gives the following chance for each of the following spacings between drops in flows A (and equivalently in the other flows):

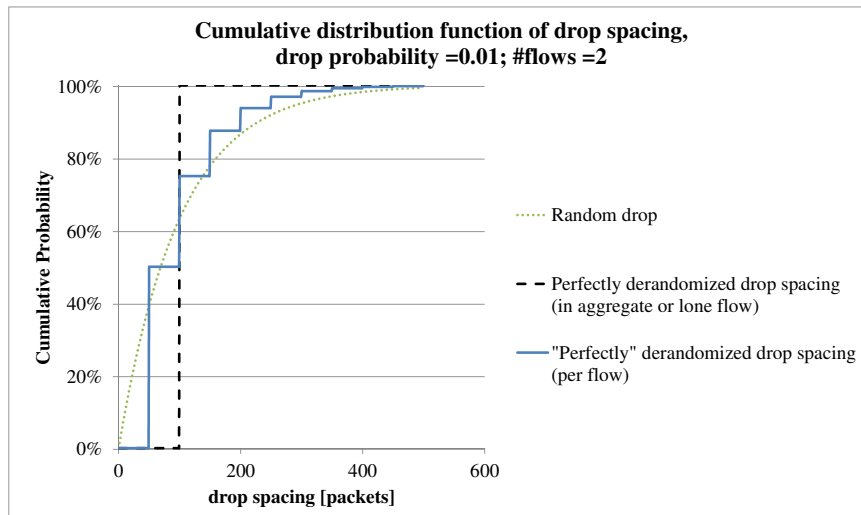
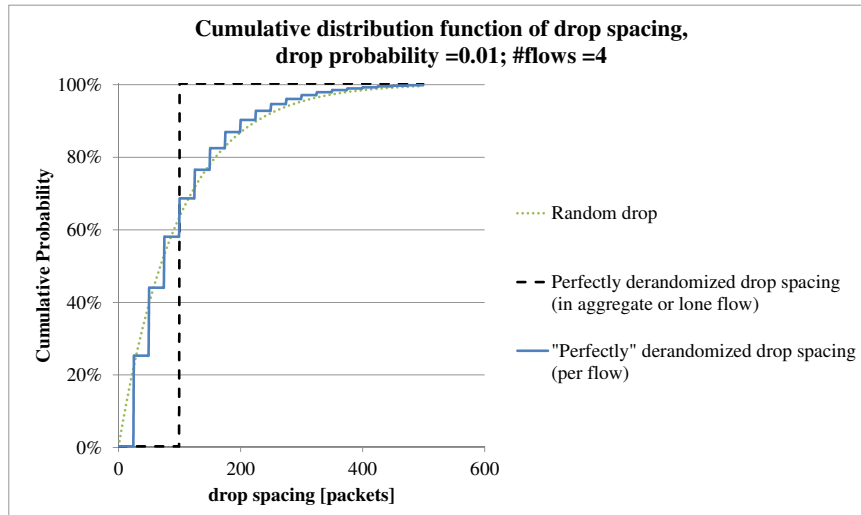


Figure 3: Rerandomization of Derandomized Drop Spacing

Drop Spacing	Probability	
~ 25	25.00%	
~ 50	18.75%	
~ 75	14.06%	
~ 100	10.55%	←intended spacing
~ 125	7.91%	
~ 150	5.93%	
~ 175	4.45%	
~ 200	3.34%	
etc.		

In fact, for any number of flows greater than one, if the cumulative distribution function of spacings with derandomization is laid on top of the CDF without, they follow nearly the same curve, except the derandomized one is stepped (see Fig 3). Derandomization does avoid bunching of drops, but it does not prevent some drops arriving very far apart, as the figure shows. This is true no matter how few flows there are, unless there is just one (see the lower figure for 2 flows).

In practice, the outcome of derandomization will be even more random than shown in the plots, for two reasons:

- These plots are for a fluid flow. In practice TCP's rate varies considerably (the classic saw-tooth), so the steps would be less-pronounced. IOW, the futile attempt to derandomize drop spacing would be 'rerandomized' even more than shown.
- These plots assume perfect derandomization. The derandomisation code in PIE deliberately leaves some randomness in the spacings. Once these more random spacing are spread randomly over different flows, there would hardly be any derandomization left.

Admittedly, if per-flow queuing is used (with one PIE per flow), PIE's derandomization will make the drop spacing more regular. But then, what benefit does derandomizing spacing give anyway?

With random spacing, sometimes one flow gets hit more than once per RTT, and sometimes another flow gets lucky and doesn't get hit at all. But surely this small benefit from more regular drops would only be noticeable if all flows were long-running, smooth and stable. In normal traffic with lots of short flows as well as large flows coming and going, I doubt the difference of regular drop spacings would be noticeable amongst all the other noise.

Also, if  $p$  has been high, so that `accu_prob` has approached 8.5, then some flows depart or the link goes idle,  $p$  will reduce, but the artificial limit at 8.5 could well cause a drop when actually  $p$  is now very low and a drop is not warranted. IOW, derandomization makes drop depend on the history of  $p$ , not just the current state of PIE (which already depends on history in a deliberate way, but should not depend on history in this accidental way as well).

In summary:

- Is it worth including the derandomization code if multiple flows share the queue? No.
- Is it worth including the derandomization code with per-flow queuing? I'm not convinced.

That's why I say derandomization merely contributes to the heat death of the Universe. Ironic.

## 5.7 Bound drop probability below 100%?

This code fragment has been added to `status_update()`:

```
//bound drop probability
if (PIE->drop_prob_ < 0)
    PIE->drop_prob_ = 0
if (PIE->drop_prob_ > 1)
    PIE->drop_prob_ = 1
```

The lower bound of zero is fine. I believe the upper bound is broken. I think that allowing the possibility of 100% drop in the code creates a potential DoS vulnerability. The early RED implementations suffered from this problem, but they were fixed later.

Explanation: If the arriving load is persistently say 200% of link capacity, without an AQM the queue would grow to infinity, but tail-drop would drop 50% and utilization of the output link would be 100%. With an AQM it still only needs to drop 50%. If an AQM is programmed to drop 100%, it will give absolutely zero throughput — the output will completely dry up — 0% utilization.

I know PIE has been tested with unresponsive traffic and it was good (while CoDel was not). I assume that's because the PIE controller stabilized on a `drop_prob` that was just right to still achieve 100% utilization at the output, and keep average delay at the target. But I think that would only happen if the unresponsive traffic was stable.

If a botnet used a flooding attack that continually increased, I think it could push `drop_prob` up towards 100% by exploiting the `beta` term in the PIE controller. I guess it could pulse this attack every time the botnet hit its sending limit (or the limit of an upstream link). Under such an attack, I think PIE would keep latency low, but utilization would drop to near-zero.

My hypothesis would need to be tested. But if this attack were possible, it would be best to switch to tail drop above a certain `drop_prob`.

This was the way various RED implementations were fixed some years ago. For instance, search for “forced” in the Apple derivative of the OpenBSD RED code here: <[http://www.opensource.apple.com/source/xnu/xnu-2050.48.11/bsd/net/classq/classq\\_red.c](http://www.opensource.apple.com/source/xnu/xnu-2050.48.11/bsd/net/classq/classq_red.c)> I recall that Fred Baker said he did something similar in the Cisco RED code.

## 5.8 Lock-out Bias against Large Packets

If PIE has to fall back to tail drop, at least it should always stop enqueueing packets when there is < 1MTU (max transmission unit) free at the tail. This prevents the bias against large packets that tail drop otherwise suffers from, which is a benefit of AQM that does not need to be abandoned even when the system is driven to tail drop.

In other words, in the `enqueue()` block in the pseudocode:

```
CURRENT
    if (queue_.is_full()) {
        drop(packet);
    }
SUGGESTED:
    if (queue_.byte_length() > (buffer_size - MTU) ) {
        drop(packet);
    }
```

With suitable definitions of the two new variables, `buffer_size` and `MTU`.

## 6 Numerous Magic Numbers

I counted 20 magic numbers in the pseudocode. Below, I have tried to reduce this by showing how some of them should depend on others.

However, my first gripe is that they should all be called out in the header of the pseudocode. I found 13 of them hard-coded within the pseudocode.

Configurable Parameters:

- QDELAY\_REF. AQM Latency Target (default: 16 [ms])
- BURST\_ALLOWANCE. AQM Latency Target (default: 150 [ms])

Internal Parameters:

- Weights in the drop probability calculation:
  - **alpha** (default: 1/8 [Hz])
  - **beta** (default: 1 + 1/4 [Hz])
  - the look-up table of factors:
 

drop_prob	factor
< 0.1%	1/128
< 1%	1/16
< 10%	1/2
else	1
- DQ\_THRESHOLD (default:  $2^{14}$  [B])
- T\_UPDATE: a period to calculate drop probability (default: 16 [ms])
- QUEUE\_SMALL = (1/3) \* Buffer limit in bytes
- **accu\_prob**
  - min (default 0.85)
  - max (default 8.5)
- uncongested queue test#1 (default: QDELAY\_REF/2)
- uncongested queue test#2 (default: **drop\_prob** < 20%)
- uncongested queue test#2 (default: **queue\_.byte\_length()** <= 2 \* MEAN\_PKTSIZE))
- EWMA constant for **avg\_dq\_time\_** (default: 1/4)

Not all these parameters are scenario-independent. So there should be a category between ‘configurable’ and ‘internal’ called perhaps ‘scenario-dependent’.

The spec needs to state how to set these scenario-dependent parameters for different environments. And if one internal parameter depends on another, it should be set relative to the other, not as a separate constant. Specifically:

- The look-up table of factors to scale alpha and beta might need to be changed if the dominant transport changes (see earlier).
- DQ\_THRESHOLD =  $2^{14}$ B is equivalent to about 10–11 Ethernet frames. In buffers carrying large numbers of flows, the average queue can be kept about this large, or smaller, without losing utilisation. So, if  $2^{14}$  were not configurable, PIE’s rate measurement might only be triggered rarely in these highly aggregated buffers.
- T\_UPDATE = 16ms would not be appropriate in short RTT scenarios (e.g. data centres or private networks), where it could lead to the same drop probability being used for many RTTs.



- It would be better to initialise `T_UPDATE` as a number of bytes divided by the nominal link rate, rather than a fixed amount of time. Then, without having to change the code, it will tend to scale with the available processing capacity, which tends to be sized per packet. E.g.  
`B_UPDATE = 96kB`  
`T_UPDATE = B_UPDATE / NOMINAL_LINK_RATE` // = 16ms for a 48Mb/s link
- `alpha` and `beta` should be specified in terms of `T_UPDATE` in the code. E.g.  
`ALPHA_DEFAULT = T_UPDATE/1ms`  
`BETA_DEFAULT = ALPHA_DEFAULT * 10{Note1}`  
 Then if `T_UPDATE = 16 ms`, `ALPHA_DEFAULT = 16`, `BETA_DEFAULT = 10*16=160`  
 And if `T_UPDATE` is ever changed to say 4 ms, `ALPHA_DEFAULT` and `BETA_DEFAULT` will correctly scale to 4 and 40 respectively.
- `QUEUE_SMALL`. Why 1/3 of the buffer? Why depend on the buffer size at all? This makes PIE depend on a sensible buffer size setting, while one of the reasons for introducing AQM is to decouple queue size from buffer size. Whatever, 1/3 seems like it was chosen assuming a buffer for a small number of flows. `QUEUE_SMALL` should surely be different for a highly aggregated link?
- `accu_prob`. Why limits of 0.85 and 8.5? Have it been tested whether these values are sufficient to prevent periods of synchronisation with TCP? Might these values need to be reconfigured if we find synchronisation occurs in certain scenarios?
- The three uncongested queue tests seem specific to a buffer intended for a small number of flows. Might they need to be reconfigured for a highly aggregated link?  
`drop_prob_ < 20%` seems a very high threshold for ‘uncongested’.  
 Might this have to be reconfigured after operational experience?
- I think the EWMA-constant for `avg_dq_time_` (default: 1/4) should depend on `DQ_THRESHOLD`. Specifically, `EWMA-constant = DQ_THRESHOLD/216`.  
 Because the lower that `DQ_THRESHOLD` is set, the more often there is a reading for `dq_time`, so the contribution of each reading to the average needs to be reduced. And this still gives `EWMA-constant = 1/4` with the recommended value of `DQ_THRESHOLD = 214`.

## References

- [BF15] Fred Baker and Gorry Fairhurst. IETF Recommendations Regarding Active Queue Management. Internet Draft draft-ietf-aqm-recommendation-11, Internet Engineering Task Force, February 2015. (Work in Progress).
- [Bri09] Robert (Bob) Briscoe. *Re-feedback: Freedom with Accountability for Causing Congestion in a Connectionless Internetwork*. PhD thesis, UC London, 2009.
- [HMTG01] C. V. Hollot, Vishal Misra, Donald F. Towsley, and Weibo Gong. On Designing Improved Controllers for AQM Routers Supporting TCP Flows. In *INFOCOM*, pages 1726–1734, 2001.
- [PNB<sup>+</sup>15] Rong Pan, Preethi Natarajan, Fred Baker, Bill Ver Steeg, Mythili Prabhu, Chiara Piglion, Vijay Subramanian, and Greg White. PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem. Internet Draft draft-ietf-aqm-pie-01, Internet Engineering Task Force, March 2015. (Work in progress).
- [PPP<sup>+</sup>13] Rong Pan, Preethi Natarajan Chiara Piglion, Mythili Prabhu, Vijay Subramanian, Fred Baker, and Bill Ver Steeg. PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem. In *High Performance Switching and Routing (HPSR'13)*. IEEE, 2013.

## Document history

Version	Date	Author	Details of change
00A	01-May-15	Bob Briscoe	First Draft;
00B	01-May-15	Bob Briscoe	Converted from txt to Tech Report, added figures.
00C	06-May-15	Bob Briscoe	Added critiques of averaging, burst allowance, rate estimation, large-packet lock-out.
01	08-May-15	Bob Briscoe	Issued without further change.
01A	09 May 2015	Bob Briscoe	Clarified large packet lock-out (§5.8) and integer log (§??)

# Evaluation of Priority Scheduling and Flow Starvation for Thin Streams with FQ-CoDel

Eduard Grigorescu, Chamil Kulatunga, Gorrry Fairhurst

School of Engineering, University of Aberdeen, UK  
{eduard, chamil, gorrry}@erg.abdn.ac.uk

Nicolas Kuhn

Télécom Bretagne, IRISA  
nicolas.kuhn@telecom-bretagne.eu

**Abstract**— Bufferbloat is the result of oversized buffers and induced high end-to-end latency experienced by applications across the Internet. This additional delay can adversely impact thin streams that frequently exchange small amounts of data, but have stringent latency requirements. Active Queue Management (AQM) techniques, such as Controlled Delay (CoDel), can control the queuing delay in a network device to ensure low latency by dropping packets to indicate incipient congestion. FlowQueue-CoDel (FQ-CoDel) is a scheduling scheme that creates one sub-queue per flow and applies CoDel on each of them. FQ-CoDel features: (1) priority scheduling for low-rate traffic; (2) flow isolation; (3) queue management with CoDel. First, this paper fills a gap in the understanding of FQ-CoDel by analyzing what features are of interests for providing low latency for thin streams applications. Second, this paper provides the first analysis of the limits of the flow starvation mechanisms and show that FQ-CoDel is vulnerable to Denial of Service (DoS) attacks.

**Keywords**— Bufferbloat, AQM, Scheduling, CoDel, FQ-CoDel, Thin-streams, Flow Starvation

## I. INTRODUCTION

Network devices require buffers to store bursts of incoming packets prior to forwarding. These buffers have traditionally implemented a FIFO/DropTail buffer policy, with passive queue management that drops packets only when a queue is full. While buffers are needed to achieve statistical multiplexing or to guarantee high bottleneck utilization when the available capacity fluctuates, excessive buffering can lead to large queues resulting in high network latency. This issue is known as Bufferbloat [1]. Although capacity between Internet core routers is often over-provisioned, this is rarely the case for access networks, using technologies such as ADSL, satellite and mobile broadband [8]. Latency has become a major issue in such access networks in the past decade where excessive queuing, affects application performance [9].

Active Queue Management (AQM) techniques, such as Random Early Detection (RED) [2], have been proposed for over a decade to appropriately manage the buffer by indicating impending congestion to responsive transport protocols, to avoid building a standing queue. However, AQM schemes have been reported to be usually turned off, as they were hard to tune. More recent protocols, such as CoDel [3] or PIE [4], have been designed to especially tackle the Bufferbloat, but with RED deployment issues in mind. Recent IETF work recommends the deployment of AQM as one solution to reduce latency [6]. On top of AQM schemes, scheduling

algorithms, managing packet scheduling and isolation/capacity allocation among flows, can be introduced. As one example of a scheme that mixes both classes, FlowQueue-CoDel (FQ-CoDel) [7] is a scheduling scheme that features prioritization and flow isolation. FQ-CoDel creates one sub-queue per flow and applies CoDel on each of them. The awareness of the latency resulting from over-provisioned buffers has been accompanied by an increase in real-time applications such as Voice over Internet Protocol, gaming or financial trading applications. As one example, the latency experienced by gamers can directly impact the perceived value of the network service [10]. These thin streams applications generally send sparse streams of small time-critical packets [14].

Since FQ-CoDel features a mechanism that prioritizes low-rate traffic, the benefits for the increasing number of thin streams that carry latency sensitive applications needs to be assessed. This paper fills the gap in research evaluating FQ-CoDel when the traffic is a mix of thin streams and bulk flows and evaluates which part of FQ-CoDel (CoDel, prioritization, flow isolation) provides improvement. Because FQ-CoDel features flow prioritization, we also evaluate to what extent its flow starvation prevention mechanism works.

The remainder of this paper is organized as follows. Section II describes FQ-CoDel, by clearly identifying when each internal mechanism adds value. Section III presents the simulation setup used to assess the suitability of FQ-CoDel for carrying latency sensitive thin streams over a capacity limited path. Section IV discusses which part of FQ-CoDel might provide improvements in the queuing delay experienced by thin stream applications. Section V assesses the limits of the flow starvation prevention mechanism. Section VI concludes this work.

## II. FLOWQUEUE-CODEL ALGORITHM

AQM algorithms seek to control network buffering level by sending messages to trigger transport congestion control, *i.e.*, by early dropping/marketing packets.

### A. Over the Need for Flow Scheduling for Thin-Streams

AQM dropping techniques on their own may not be sufficient to satisfy the strict latency requirements for thin stream applications [5],[6]. Indeed, traffic such as file transfers or unresponsive constant rate streaming flows, with different time constraints, may share the bottleneck with thin streams.

This would result in a non-negligible queuing delay experienced by the thin streams.

Some form of flow isolation might be required to separate and protect the time sensitive small flows from larger and more aggressive flows. Scheduling algorithms can provide per-flow or per-class queuing to isolate traffic classes to guarantee the specific constraints of latency sensitive applications. As one example, if thin flows are assigned different subqueues to other flows, a scheduling scheme may protect the thin flows from the background traffic. This would avoid an increasing queuing delay for a latency sensitive application when the background flows build in a subqueue.

One simple isolation method is per-flow queuing [11] in which each flow is deterministically assigned its own virtual sub-queue. However, this requires network devices to classify each flow, which can be difficult when dealing with traffic aggregates or when encryption is used. Each of the subqueues is served in a round-robin manner, improving fairness between flows [5]. A more complex example is Stochastic Fair Queuing (SFQ) [5] that provides a statistical alternative in the way the subqueues are served.

### B. FlowQueue scheduling in FQ-CoDel

FQ-CoDel uses a modified Deficit Round Robin (DRR) scheduler. The default flow classifier of FQ-CoDel hashes incoming packets stochastically to a subqueue based on a 5-tuple classifier – IP source and destination address, protocol, and port numbers. The scheduler of FQ-CoDel can be applied on any flow isolation technique and is not limited to a 5-tuple classifier. A byte-based scheduler, rather than a packet-based scheduler, is used to select the next packet for transmission.

The scheduling in FQ-CoDel is based on three lists of subqueues that are represented in Figure 1, the “new” list, the “old” list and the “empty state” list. In the rest of this subsection, we will detail how these lists are managed. We focus on how they can prevent flow starvation and how they prioritize some classes of traffic.

At initialization, FQ-CoDel creates a set of subqueues (by default 1024 subqueues), all are placed in the “empty state” list. When there is at least one byte of data that enters a subqueue, the subqueue is defined as *active*. This subqueue is initially placed in the “new” list, but may later be moved to the “old” list.

When a packet is enqueued, it is added at the end of the subqueue corresponding to the 5-tuple classifier: if the 5-tuple of this packet does not correspond to any list, it is added to an existing “empty list”, which becomes a “new” list.

We describe here how the *deficit* is decreased and how the decision to dequeue data is taken:

- The scheduler first cycles through the “new” list of subqueues, allowing each subqueue to dequeue up to one “quantum” of packets and updating the deficit value.
- For each subqueue, the scheduler checks the deficit. If the deficit becomes negative, the subqueue is moved to the end of the “old” list and its deficit is updated to the sum of the size of the quantum and the previous deficit.

- When a subqueue in the “new” list is found to be inactive (no packets in the buffer), it is placed at the end of the “old” list, and its deficit is re-initialized to the “quantum” size.

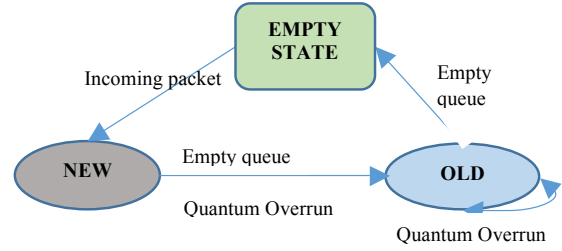


Figure 1. FQ-CoDel State Machine

- When the “new” list becomes empty, the scheduler examines the “old” list by repeating the algorithm used on the “new” list. When a subqueue from the “old” list becomes empty, it is removed.

### C. Flow Starvation Prevention Mechanism in FQ-CoDel

When the deficit of a given queue,  $Q$ , becomes negative, it is moved to the end of the “old” list. This ensures that when FQ-CoDel loops over the “old” list, a subqueue that had been previously pushed to the end of the “old” list would be given the opportunity to transmit a quantum of data before  $Q$ . When a queue is empty, FQ-CoDel pushes the queue at the end of the “old” list. This feature, referred to as the Starvation Prevention Mechanism, is supposed to prevent starvation, providing some transmission opportunities for the flows already placed in the “old” list.

### D. AQM in FQ-CoDel

Each subqueue is individually managed using the CoDel algorithm: FQ-CoDel classifies each packet, it timestamps the packet and appends it to the tail of the selected subqueue. CoDel controls the maximum size of each subqueue, using its default parameters: target delay of 5 ms and interval 100 ms. CoDel applies its control law and may discard at least one packet from the head of a scheduled subqueue if needed, before returning a packet for dequeuing (or no packet if the subqueue becomes empty). This should avoid buffer overflow and guarantee low queuing delay, if there are many flows and the scheduling introduces a non-negligible queuing delay.

## III. EVALUATION TOOL SET FOR THE CAPACITY LIMITED NETWORK USE CASE

This section justifies our focus on the capacity-limited network use case. We also present the simulation tool set and the network topology used in our simulations.

A 10 Mbps (or higher speed) bottleneck that experiences congestion, has a transmission speed that is sufficiently low to result in negligible transmission delay, even for large packets, compared to the latency required by typical thin stream applications (approx. 1.2 ms for a packet of 1500 B sent at 10 Mbps). When the bottleneck has a smaller rate (e.g., the downlink of a rural access link operating at 1-2 Mbps or an

uplink, operating at 1/10 of this speed), the packet transmission delay increases further (e.g., 12-6 ms for 1500 B packets). The cumulative effect of scheduling many competing flows can result in some flows becoming “choked”. Because (1) the impact of the thin streams on the flow starvation of FQ-CoDel is exacerbating and (2) the latency sensitive applications would be more affected when there is no priority scheduling, we therefore focus on this use case.

Figure 2 presents the dumb-bell topology for our simulations in *ns-2*. The capacity of the bottleneck was 2 Mbps and the one-way-delay was 47.5 ms. The non-bottleneck links were configured with 1.25 ms one-way-delay and 100 Mbps capacity. The RTT of the network path was 100 ms.

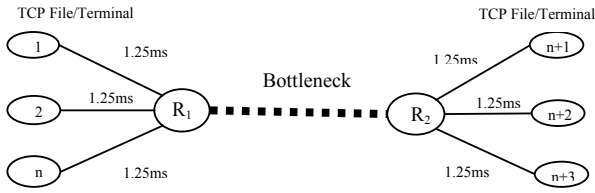


Figure 2. Dumb-bell simulation topology

At node R1, the buffer size was set to twice the size of the Bandwidth-Delay Product (BDP) and either FQ-CoDel or CoDel applied to the node. For other nodes, the queuing discipline was DropTail, with a buffer size of 300 packets.

#### IV. ANALYZING THE PERFORMANCE OF THIN STREAMS WITH FQ-CODEL

FQ-CoDel may be divided into different constituent mechanisms. There is one AQM scheme per subqueue, one priority scheme and a set of flow isolation techniques. If FQ-CoDel is to be used to support thin streams, it is important to assess which mechanism within FQ-CoDel is actually responsible for realizing any benefits observed.

##### A. The Benefits of Flow Isolation for Thin Streams

To support our evaluation, this subsection introduces a custom non-prioritization version of FQ-CoDel, FQ-CoDel Without Prioritization (FQ-CoDel WP). FQ-CoDel WP does not make the distinction between the “new” and “old” list and subqueues are created one after the other, while the scheduler still visits subqueues similar to SFQ. This is used to assess the benefits of the flow isolation for the thin streams applications.

This subsection compares the suitability of using FQ-CoDel WP, CoDel or SFQ to carry thin streams applications. The following traffic was considered: (1) 1 to 5 TCP bulk flows, using File Transfer Protocol (FTP) for the entire period of the simulation with a packet size of 1500 B. (2) 1 thin TCP with an inter-packet interval of 638 ms and a packet size of 100 B, which is representative of the traffic generated by the game Anarchy Online [12]. (1) and (2) used TCP New Reno with the SACK option and an initial window of 3 packets. Figure 3 shows the Cumulative Density Function (CDF) of the queuing delay experienced by the thin-stream flows, when the AQM is CoDel, SFQ or FQ-CoDel WP.

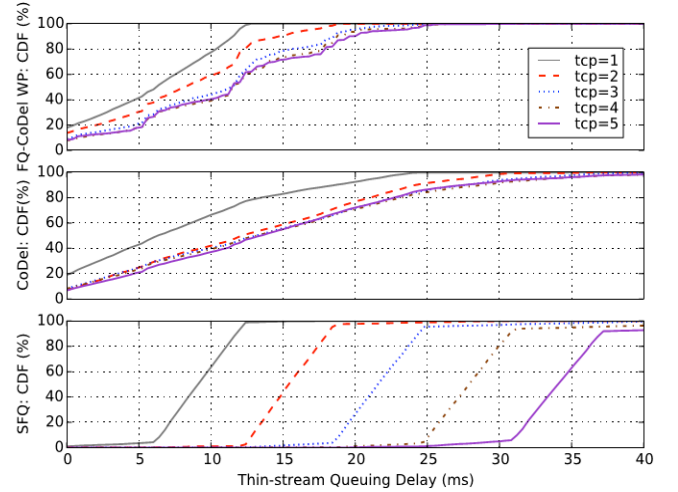


Figure 3. CDF vs. Gaming Flow Queue Latency

With SFQ, the queuing delay increases as the level of network congestion increases (increasing number of TCP flows). For each TCP packet, the Round Robin scheduler of SFQ results in a 6 ms transmission time per packet (1550 B for a 2 Mbps bottleneck). As expected, a Round Robin scheduler that services a higher number of queues leads to increased queuing delay: we observe a linear increase of the queuing delay when the number of TCP flows increases. As an example, the thin stream subqueue will experience  $5 \times 6\text{ms} \geq 30\text{ms}$  queuing delay with 5 bulk TCP flows before it has the opportunity to again be serviced.

With CoDel, the observed queuing delay is lower than with SFQ, whatever the number of TCP bulk flows. The early drops in CoDel tend to maintain a small queue that reduces the delay experienced by the thin stream flows. This queuing delay does significantly increase with the number of flows competing for queue space, as compared to SFQ.

With FQ-CoDel WP, the queuing delay experienced by the thin streams is lower than when using CoDel and lower than with SFQ. These results show that (1) dropping packets with CoDel enables a latency reduction; (2) flow isolation alone cannot reduce the queuing delay experience; (3) the performance of flow isolation techniques are sensitive to the traffic load. We can conclude that when FQ-CoDel features flow isolation, it results in lower queuing delay than with CoDel alone, showing that the flow isolation technique, along with CoDel drops, can offer the best of the two schemes.

##### B. The Importance of Thin Stream Prioritization and CoDel

In this subsection, we will explore the benefits of introducing prioritization in FQ-CoDel to reduce the queuing delay experienced by thin streams. We compare the performance of the default FQ-CoDel with “FQ-CoDel WP 100ms”, which is a modified version of FQ-CoDel in which the target of CoDel is increased to 100 ms (instead of the default 5 ms) and prioritization is disabled. Therefore, the differences between FQ-CoDel WP 100ms and FQ-CoDel are

- (1) CoDel will allow more queuing in FQ-CoDel WP 100 ms
- and (2) the absence of prioritization in FQ-CoDel WP 100 ms.

The traffic considered in this section is the same as in IV-A. Figure 4 presents the CDF of the queuing delay experienced by the thin streams for various numbers of TCP flows with FQ-CoDel or FQ-CoDel WP 100ms as an AQM.

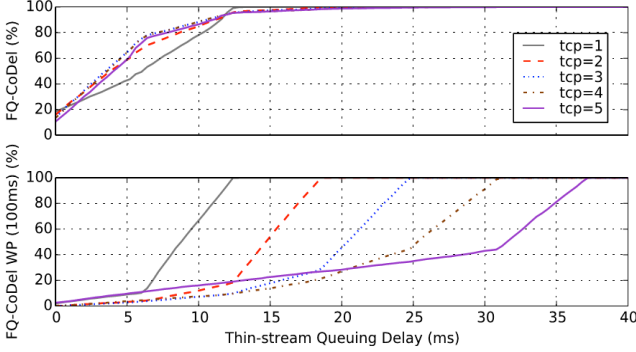


Figure 4. CDF vs. Game Flow Latency for FQ-CoDel (5 ms target delay) and non-prioritized FQ-CoDel WP (100ms target delay)

The performance with FQ-CoDel WP 100ms shows that without prioritization and with CoDel less aggressive, the performance of this scheme is close to those of SFQ. The flow isolation of FQ-CoDel may reduce the latency (as shown in the previous section), but the CoDel part of the algorithm has a non-negligible benefit in reducing the queuing delay.

Also, if we compare the performance of FQ-CoDel and those of FQ-CoDel WP, shown respectively in Figure 4 and Figure 3, we see that the prioritization contributes in reducing the queuing delay experienced by the thin streams. FQ-CoDel provides lower queuing delay, for the traffic loads considered, whereas it is sensible to the traffic load with FQ-CoDel WP.

Therefore, based on the results presented in this subsection, we can conclude that the flow prioritization of FQ-CoDel provides a useful latency reduction and makes the queuing delay of the thin streams less sensible to the traffic load. We also confirm the conclusions of section IV-A, which are that the CoDel part of FQ-CoDel is essential to provide low latency in the context of capacity-limited networks.

### C. Thin Stream with various inter-packet arrival times

Depending on the burst size and the frequency between bursts, when a second burst of packets reach the queue, the packets of the previous burst might still be in the “old” list, or they might have left the queue and the second burst would be prioritized. Flows with a different pattern of packet inter-arrival times but similar packet sizes can be treated differently.

We consider three cases of traffic generation: (1) 1 gaming flow, 1 TCP bulk flow and 1 VoIP flow; (2) 1 gaming flow, 1 TCP bulk flow and 5 VoIP flows; (3) 1 gaming flow, 5 TCP bulk flows and 1 VoIP flow. Both the VoIP and the gaming flows use TCP. The inter-packet arrival time for the VoIP flows is in 20-30 ms and in 600-1000 ms for the gaming flows. The packet size for both applications is on average 100 B. Results are shown in Figure 5.

With FQ-CoDel WP 100 ms and FQ-CoDel WP, the queuing delay for both voice and gaming flows increases in case (3), because of the high number of TCP flows, showing again the importance of CoDel coupled with isolation and prioritization. However, in cases (1) and (2), the load level is lower than in case (3), and we see a small gain of using the priority scheme of FQ-CoDel. With FQ-CoDel WP, we notice a small difference between gaming and voice flows this may be related to their different inter-packet arrival times, showing that the priority scheme of FQ-CoDel actually lets the AQM scheme reduce the queuing delay for applications with different inter-packet arrival times.

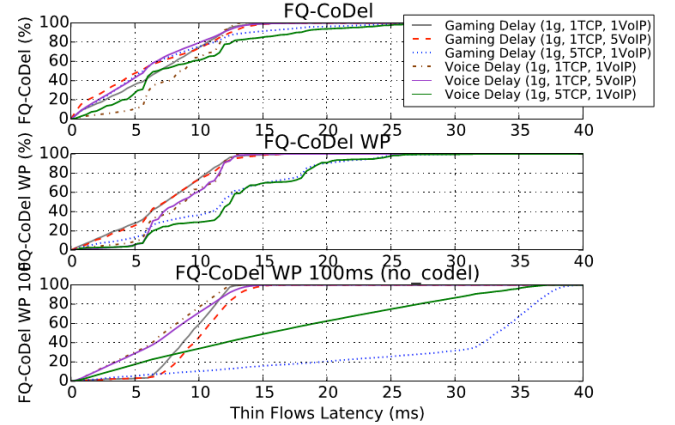


Figure 5. Thin Flows Latency for prioritized and non-prioritized FQ-CoDel

The results presented in this subsection let us conclude that FQ-CoDel can deal with thin streams flows that have different inter-packet arrival times. We also highlighted that the prioritization mechanism seems to provide benefits. Another possible source of improvement might be the starvation prevention mechanism; we expect future work to evaluate this.

## V. FLOW STARVATION AND FQ-CODEL

We discuss here the limits of the flow starvation prevention mechanism. The starvation prevention mechanism of FQ-CoDel is the following: FQ-CoDel would place an empty queue from the “new” list to the end of the “old” list. Also, if a queue is empty in the “old” list, it would be removed and considered as “new” when its packets reach the queue.

We assess the limits of the starvation prevention mechanism of FQ-CoDel. Under Denial of Service (DoS) attacks, FQ-CoDel scheduler may loop only over the “new” list, preventing the flow starvation mechanism to work. It is crucial therefore to assess the performance of the flow starvation mechanism when thin streams contribute a large proportion of the traffic compared to bottleneck capacity. A starvation prevention method that may prevent this issue from occurring. To verify the benefits of using this mechanism, we implemented FQ-CoDel Without its Starvation Prevention Mechanism (FQ-CoDel WSPM). FQ-CODEL WSPM is a version of FQ-CoDel where starvation prevention is disabled, that is while looping over the “new” list, the algorithm would not move empty lists from the “new” list to the “old” list, but rather wait that the deficit for the list is negative.



The following traffic were considered: (1) 1 TCP bulk flows, using FTP for the entire period of the simulation with a packet size of 1500 B; (2) 5 to 45 thin unresponsive UDP flows with an inter-packet interval of 30 ms and a packet size of 100 B, which is representative of the traffic generated by a Skype session [12]. (1) used TCP New Reno with the SACK option enabled and an initial window of 3 packets.

Figure 6 presents the throughput achieved by the TCP flow as a function of the number of thin flows, with FQ-CoDel and with FQ-CoDel WSPM. When the number of thin streams is greater than 30, the TCP bulk flow becomes starved, both when using FQ-CoDel and with FQ-CoDel WSPM. This shows that unresponsive traffic can impact the performance, since the flow starvation prevention mechanism of FQ-CoDel is not sufficient to prevent the resulting congestion. The lack of a difference between FQ-CoDel with and without this mechanism highlights that it does not provide significant benefit when there is a high level of congestion.

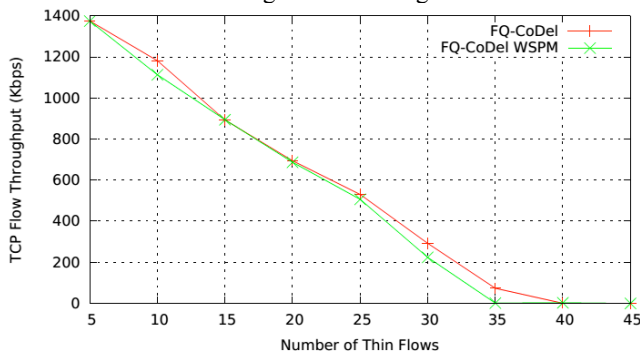


Figure 6. TCP Throughput with FQ-CoDel and with FQ-CoDel WSPM

We finally consider the response to overload. An FQ-CoDel scheduler could be vulnerable to a Denial of Service (DoS) attack where traffic intentionally tries to disrupt normal operation, e.g., a large number of thin streams would be intentionally injected into the network bottleneck. This may cause the scheduler to become locked serving only flows in “new” list subqueues, leading to unwanted interactions between TCP flows and bursts of unresponsive flows that result in delay. For such a low capacity link, flows in the “old” list would barely receive an opportunity for transmission, and would become starved. CoDel would also drop queued packets from the old list. This suggests the need to design more sophisticated overload protection [6].

We note that other modern AQM algorithms, such as PIE, can also be combined with isolation methods [6] by introducing mechanisms similar to those described in this paper. The combined methods have been reported to support latency-sensitive thin-stream applications [8]. We leave exploration of the reasons behind the performance limits of the flow starvation prevention mechanism in capacity-limited networks as a part of our future work on this topic.

## VI. CONCLUSION

This paper describes the operation of the various components mechanisms in FQ-CoDel and explores how thin-stream applications can benefit from these mechanisms to

reduce their experienced latency and mitigate the impact of sharing capacity with other types of network traffic. We measured that the flow isolation of FQ-CoDel is the main factor resulting in improved latency performance for thin flows over bottlenecks with limited bandwidth. Such isolation would be impossible with a traffic aggregate that it cannot dissect (e.g., when encrypted Virtual Private network tunnels are used). We identified that CoDel in FQ-CoDel can provide improvements for this specific traffic, this means that when a classifier cannot be used, low latency may still be guaranteed.

Flow prioritization has been introduced within FQ-CoDel to that favors low-rate traffic, or latency sensitive applications such as web traffic. Simulations have shown that this scheme provides a fair improvement of performance for thin stream traffic. We believe in general that deployed AQM algorithms should be made robust against overload and especially denial of service attacks, otherwise all the efforts spent in making the deployment of AQM a reality will be eroded. In the light of the results presented in this document, we encourage further research prior to deployment of the current version of FQ-CoDel. Other modern AQM algorithms, such as PIE, can also work in conjunction with isolation methods to better support latency-sensitive application. Therefore, we believe that more efforts should be spent on evaluating the performance of such hybrid mechanisms to support their large-scale deployment.

## ACKNOWLEDGEMENTS

This research was supported by the RCUK DE programme to the dot.rural Digital Economy Hub; EP/G066051/1 and part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

## REFERENCES

- [1] Gettys, Jim. *Bufferbloat: Dark Buffers in the Internet*. IEEE Internet Computing, IEEE, 1990.
- [2] Jacobson, Van & Floyd, Sally. *Random Early Detection Gateways for Congestion Avoidance*. IEEE/ACM Transactions on Networking, 1993.
- [3] Nichols, Kathleen & Jacobson Van. *Controlling Queue Delay*. IETF work-in-progress, 2015.
- [4] Pan, Rong, et al.. *PIE: A lightweight control scheme to address the bufferbloat problem*. IETF work-in-progress, 2015.
- [5] McKeeney, Paul E. *Stochastic Fairness Queuing*. Beaverton : s.n., June 1990. In proceedings of INFCOM. Vol. 2, pg. 733-740.
- [6] Baker, F & Fairhurst, G. *AQM Recommendations Regarding Active Queue Management*. IETF work-in-Progress
- [7] Hoeiland-Joergensen, T., et al. *FlowQueue-CoDel*. IETF work-in-progress, 2015
- [8] White, Greg. *Active Queue Management in DOCSIS 3.X Cable Modems..* Cable Television Laboratories. 2013.
- [9] N. Leavitt, "Network-Usage Changes Push Internet Traffic to the Edge", IEEE Computer, Vol 43, Issue 10, October 2010.
- [10] Claypool, M. and Claypool, K. Latency and player actions in online games. *Communications of the ACM* 49, 11 (Nov. 2005), 40-45.
- [11] Nabeshima, M.; Yata, K. An Effective Queue Management Scheme for Data Communication. *IEEE Proceedings-Communications*. 2005
- [12] Petlund, Andreas et al.. *TCP mechanisms for improving the user experience for time-dependent thin-stream applications*. 33rd IEEE Conference on Local Computer Networks, 2008. LCN 2008
- [13] L.D Cicco, S. Mascolo, and V. Palmisano. *Skype Video Congestion Control: An experimental investigation*. Computer Networks, 2011
- [14] Petlund, Andreas. *Improving latency for interactive, thin-stream applications over reliable transport*. PhD Thesis. 2010

# Tackling Bufferbloat in Capacity-limited Networks

Chamil Kulatunga<sup>†</sup>, Nicolas Kuhn<sup>‡</sup>, Gorrry Fairhurst<sup>†</sup>, David Ros<sup>\*</sup>

<sup>†</sup>School of Engineering, University of Aberdeen, Aberdeen, UK. {chamil,gorrry}@erg.abdn.ac.uk

<sup>‡</sup>IMT Télécom Bretagne, IRISA, France. nicolas.kuhn@telecom-bretagne.eu

<sup>\*</sup>Simula Research Laboratory, Fornebu, Norway. dros@simula.no

**Abstract**—Over-provisioned network buffers, often at the Internet edge, induce large queuing delay and high latency; this issue is known as Bufferbloat. In response to this, a set of recently proposed Active Queue Management (AQM) algorithms attempt to reduce standing queues, while maintaining the bottleneck utilisation at an acceptable level. This paper assesses the performance of two AQM schemes (CoDel and FQ-CoDel) over capacity-limited networks with large Round-Trip Time (RTT). In such settings, these AQM schemes have difficulty controlling the buffering level, resulting in both momentarily high queuing delay and low bottleneck utilisation, even if the methods are claimed to be insensitive to link rates and round-trip delays. We explore this issue and show that it is possible to adapt the parameterisation of CoDel and FQ-CoDel to offer a higher bottleneck utilisation while maintaining a low queuing delay. We present experiments over an emulated test bed and a satellite network to confirm that our new parameterisation improves the download time of moderate-size files and reduces the latency for capacity-limited and large-RTT networks.

**Keywords**—Bufferbloat, Rural Broadband, AQM, CoDel.

## I. INTRODUCTION

Oversized buffers along Internet paths may lead to large standing queues and high queuing delay, this issue is known as *Bufferbloat* [1]. Active Queue Management (AQM) proactively drop packets before router buffer space is exhausted. This signals incipient congestion to endpoints and avoids persistently large queues. AQM aims at reducing queuing latency and is one piece of the solution to Bufferbloat.

Random Early Detection (RED) [2] is an AQM proposed two decades ago, but has not been widely enabled in Internet routers due to the need to tune its parameters depending on the actual traffic and network conditions. Newly proposed AQM algorithms, such as Controlled Delay (CoDel) [3], Proportional Integral controller Enhanced (PIE) [4] and Flow-Queueing CoDel (FQ-CoDel) [5]), claim to address this difficulty of configuring AQM parameters.

In a rural broadband network, a service is often characterised by much less capacity than in other contexts. The greater cable distance to many rural locations is one key factor that can limit offered capacity, but the type of installed access technology and the lower population density can also impact available capacity. In the UK, 21% of rural areas are currently unable to access to a DSL downlink speed of 2Mbps [6]. Despite introducing a larger RTT (e.g., 500 ms or more) satellite technology may be deployed in such cases to realise a commercially viable broadband service [7].

In [3], the authors say that CoDel “controls delay, while insensitive to round-trip delays, link rates, and traffic loads”.

The claim that CoDel can self-tune its dropping policy suggests that no knobs needs to be tuned. However its behaviour has been shown to depend on the congestion level [8] and the RTT of the path [9]. The benefits of introducing AQM mechanisms in capacity-limited and large RTT networks are not yet well understood.

In this paper, we evaluate the potential of deploying AQM in such networks to tackle Bufferbloat. We propose to tune the parameterisation of CoDel and FQ-CoDel to both limit the queuing delay and optimise use of resources for a large-RTT and capacity-limited network. Evaluation in an emulated test bed shows that our parameterisation reduces the transfer time for medium-sized files. This analysis is supported by tests using a broadband satellite access network.

The remainder of this paper is structured as follows. Section II details the CoDel and FQ-CoDel algorithms. Section III illustrates how low-capacity, high-RTT paths are challenging for these self-tuning AQM schemes. Based on ns-2 simulations, Section IV develops our updated parameterisation of CoDel. Section IV also evaluates the impact of various base RTTs on the performance of CoDel and FQ-CoDel. Experimental results, including from an actual satellite broadband link, are shown in Section V. Section VI concludes the paper.

## II. DELAY-BASED AQM SCHEMES

Delay-based AQM algorithms typically use a target delay,  $\tau$ , determining the allowable standing queuing delay, and an update interval,  $\lambda$ , determining the frequency at which the dropping policy is updated. PIE has already been shown to need an updated parameterisation for specific scenarios (such as data-centers [10] or cable modems [11]). CoDel has been shown to not always control the queuing delay with a limited impact on the bottleneck utilization [8], [9], and it was uncertain whether CoDel could be tuned for objectives and network conditions other than the ones for which it has been designed: this led to our interest in examining CoDel and not PIE. We also wanted to evaluate the performance of a hybrid scheduling/AQM scheme, which is possible with FQ-CoDel, whereas there is no reference algorithm for FQ-PIE. Therefore, we will focus on CoDel and FQ-CoDel, for which  $\tau = 5$  ms and  $\lambda = 100$  ms have been suggested as default values [3], [5].

### A. CoDel

CoDel [3] tracks  $enq_i$ , the *enqueue* time of each packet  $p_i$ . At the *dequeue* time,  $deq_i$ , CoDel computes the queuing delay of



each dequeued packet:  $\delta_i = \text{deq}_i - \text{enq}_i$ . CoDel has two states, *dropping* and *non-dropping*, and starts on the latter state. We denote by  $\mu$  the interval after which CoDel may change its state and drop one packet. The initial value of  $\mu$  is  $\lambda$ . We denote by  $n_{\text{drop}}$  the number of consecutive drops, initialized as  $n_{\text{drop}} = 1$ . Between  $t_1 = t$  and  $t_2 = t + \mu$ , if there is a packet  $i$  such that  $\text{deq}_i \in [t_1, t_2]$  and  $\delta_i > \tau$ , then: (1) CoDel enters the dropping state, (2) the next packet to be dequeued is dropped, (3)  $n_{\text{drop}} += 1$  and (4)  $\mu$  is set to  $\mu / \sqrt{n_{\text{drop}}}$ ; else: (1) CoDel enters the non-dropping state, (2) there is no packet drop, (3)  $n_{\text{drop}}$  is reset to 1 and (4)  $\mu$  is reset to  $\lambda$ .

### B. FQ-CoDel

FQ-CoDel [5] maintains one sub-queue per flow. There is one instance of CoDel per sub-queue. FQ-CoDel features priority-queuing by giving priority to new incoming flows and fair-queuing. When a packet arrives, if there is already another packet belonging to this flow in an existing sub-queue, the incoming packet is added to the latter, else a new sub-queue is instantiated and the incoming packet added to it. FQ-CoDel maintains two lists of sub-queues, *new* and *old*. When looking for a packet to dequeue, FQ-CoDel starts by looping over the *new* list. If the selected sub-queue has already dequeued a quantum of bytes (default 1514 B), this sub-queue is put at the end of the *old* list. If the selected sub-queue in the *new* list has dequeued less than a quantum of bytes, the sub-queue is selected. If there are no sub-queues in the *new* list where less than a quantum of bytes has been dequeued, FQ-CoDel loops over the *old* list in the same manner. Once a sub-queue has been selected, the dropping policy of CoDel is applied to the sub-queue: a packet is dequeued from it if the algorithm of CoDel does not drop this packet.

## III. AQM SCHEMES IN CAPACITY-LIMITED NETWORKS

### A. Topology and traffic

The topology presented in Fig. 1 was used with experiments run with Linux kernel 3.14.4 and a network emulated by *netem*. Throughout the paper, we consider that there is a sender/receiver pair for each flow (2 flows in this section), all of them sharing the same bottleneck. All flows use TCP NewReno with SACK and an initial window of 3 segments.

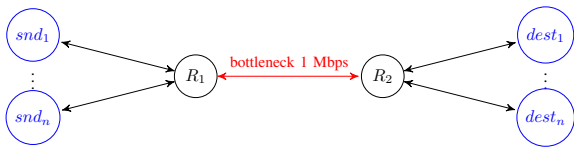


Figure 1: Dumb-bell simulation topology.

The symmetrical bottleneck between  $R_1$  and  $R_2$  has 1 Mbps capacity and a range of RTT values were tested. The bottleneck buffer size is 25 packets of size 1500 B (i.e., a 300 ms worst-case queuing delay). Other links have a 1.25 ms one-way delay and 100 Mbps capacity with a DropTail buffer of 300 packets. AQM schemes are applied at node  $R_1$  but not at  $R_2$ . In this section, the default parameterisation of CoDel and FQ-CoDel

is used ( $\tau = 5$  ms and  $\lambda = 100$  ms). 300 s, was replicated ten times and the following plots represent average values and 95% confidence intervals.

### B. Issues with low-capacity links

We consider one bulk transfer in parallel with repeated downloads of a 1.7 MB file (inter-file intervals are exponentially distributed with mean 9.5 s). In Fig. 2, we plot the throughput of the bulk transfer (averaged every second) and the average File Completion Time (FCT) for the repeated downloads. Queuing delays (not shown) were as high as 300 ms with DropTail, whereas with CoDel and FQ-CoDel they tended to stay below  $\approx 30$  ms (i.e., at most a couple of packets in the queue).

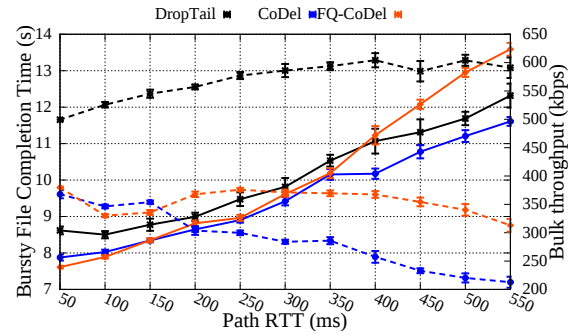


Figure 2: Performance with different queue management methods, for various path RTTs (solid lines: FCT for a repeated download; dashed lines: bulk transfer throughput).

These results illustrate the main problem with the considered AQMs when bottleneck capacity is low. AQM does maintain a much lower queuing delay, which should help with achieving a lower FCT. However, the net effect of AQM is a large drop in bulk throughput, for both short and long RTTs, for a relatively modest gain in FCT ( $\approx 10\%$  at best, for medium-sized files).

Such degradation of throughput can be intuitively explained as follows. With a 1 Mbps link, it takes 12 ms to transmit one 1500-B packet. An AQM algorithm with a 5 ms target delay over a 1 Mbps link is thus attempting to maintain a queue of less than one packet, and an often empty queue translates into lower throughput and low utilisation.

## IV. TUNING CoDEL TO CAPACITY-LIMITED NETWORKS

Our goal can be stated as: *we seek to find a combination of  $\tau$  and  $\lambda$  for both CoDel and FQ-CoDel that, for capacity-limited networks, both enables a high bottleneck utilisation (around 90% or higher) and limits the queuing delay*. We consider both  $\tau$  and  $\lambda$  because the resulting queuing delay and bottleneck utilisation depend on their interaction: drops are only applied if the queuing delay grow above  $\tau$  and only every  $\mu_{n_{\text{drop}}}$  in order to give end-points time to react to a drop.

Figure 2 shows that throughput degradation with FQ-CoDel is less sensitive to the RTT, compared with CoDel; more

generally, the flow-queuing mechanism may result in different dynamics for the individual flows. However, if multiple flows coexist in a single queue (e.g., when flows are aggregated in a VPN), FQ-CoDel would behave similarly as CoDel. Therefore we will consider a parameterisation for CoDel, and verify whether it suits FQ-CoDel too. In this section, we use the ns-2 simulator to easily cover a wide range of values of  $\tau$  and  $\lambda$ . The validity of our parameterisation is verified with emulations in Section V.

#### A. Topology and traffic

The topology shown in Fig. 1 was simulated with ns-2. The symmetrical bottleneck has 1 Mbps capacity and a 300 ms RTT; the bottleneck buffer size is set to 25 packets. Other links have a 1.25 ms one-way delay and 100 Mbps capacity with a DropTail queue of 300 packets. On the bottleneck link, CoDel and FQ-CoDel are used in node  $R_1$  but not in node  $R_2$ .

The following traffic was considered: (1) repeated downloads of a file of size 1.7 MB, with an inter-file interval exponentially distributed with mean 9.5 s; (2) TCP bulk transfer, lasting for the entire length of the simulation; (3) “thin” TCP flows with a constant inter-packet interval of 300 ms and a packet size of 150 B; these may represent gaming traffic. Both (1) and (2) use a packet size of 1500 B. TCP characteristics are the same as in Section III-A. Two traffic mixes were simulated: *low load* (one flow for each of flow types (1), (2) and (3)) and *high load* (two flows for each of (1), (2) and (3)). The flows start randomly within the first second of the simulation. The simulations last 300 s and were run 100 times, each with a different seed.

#### B. Tuning $\lambda$ to maximise the goodput

We first want to find a  $\lambda$  so that  $\approx 90\%$  of the link capacity is exploited while limiting the impact on the queuing delay.

The authors of [12] explain that  $\lambda$  should be set to the largest RTT of the flows sharing the bottleneck. Such setting is said to allow CoDel to keep control of the queuing delay. However, in many deployment cases, the person configuring the router will not be aware of the path RTT. Moreover, in a low-capacity scenario, changing  $\lambda$  alone may not be enough to obtain both a good bottleneck utilisation and control of the delay. When  $\lambda$  increases, CoDel reduces the frequency at which it updates the dropping policy. A larger  $\lambda$  induces less packet drops, resulting in a higher bottleneck utilisation at the expense of higher queuing delay. This can be seen in Fig. 3. With CoDel, increasing  $\lambda$  improves utilisation, with  $\lambda = \text{RTT} = 300$  ms bringing utilisation reasonably close to 90% for both traffic profiles. But, for all the values of  $\lambda$  tested (which include the default one of 100 ms), delays are always  $> \tau$ .

#### C. Tuning $\tau$ to maximise the goodput

We next turn to finding a value of  $\tau$  so that at least 90 % of the available capacity is exploited while limiting the impact on the queuing delay. Based on the results in §IV-B,  $\lambda$  is set to 300 ms for both AQMs. Figure 4 presents the same performance metrics of Fig. 3 for various  $\tau$ . The choice  $\tau =$

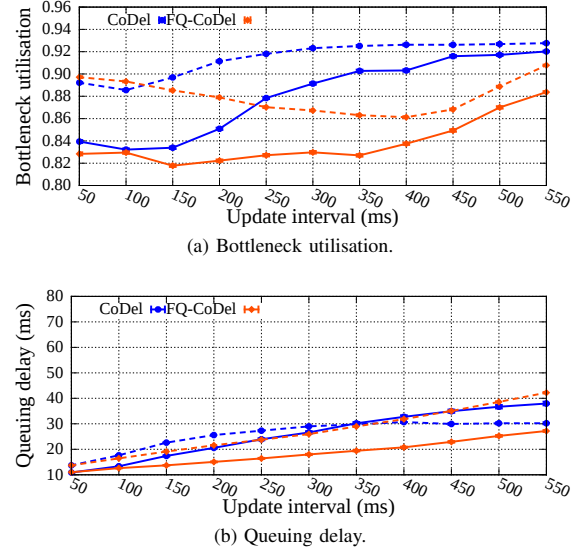


Figure 3: Utilisation and delay for various  $\lambda$ , with  $\tau = 5$  ms (solid lines: low load; dashed lines: high load).

65 ms yields a reasonable compromise between queuing delay ( $< 50$  ms) and utilisation (slightly above 90%).

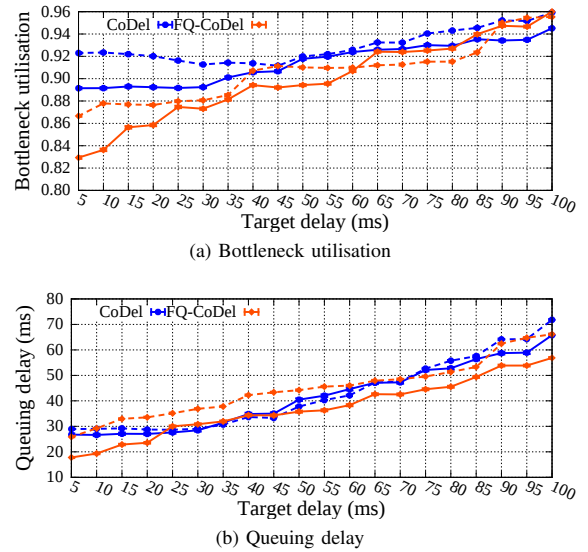


Figure 4: Utilisation and delay for various  $\tau$ , with  $\lambda = 300$  ms (solid lines: low load; dashed lines: high load).

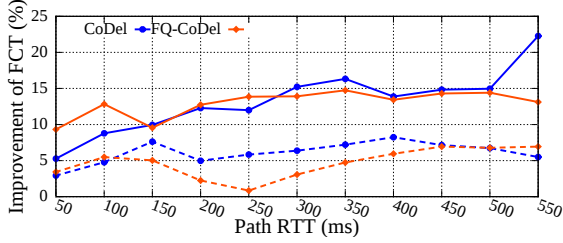
#### D. Rural and Default parameterisations

We propose to set  $\tau$  to 65 ms and  $\lambda$  to 300 ms for both CoDel and FQ-CoDel, so that at least 90 % the bottleneck capacity of the constrained network is utilised and the queuing delay limited to a reasonable value, much smaller than the base RTT. While these parameters may be appropriate for other capacity-limited cases, we do not claim they suit every

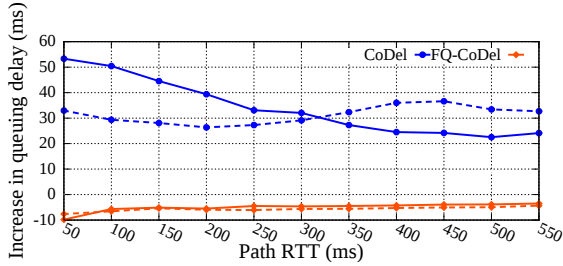
capacity-limited network. We therefore evaluate if this parameterisation is robust to a range of RTT values. We refer to the AQM parameter settings optimised to suit the low-bandwidth, high-RTT scenario studied above as *Rural* parameterisation, in contrast with the *Default* parameterisation of an AQM.

#### E. RTT sensitivity

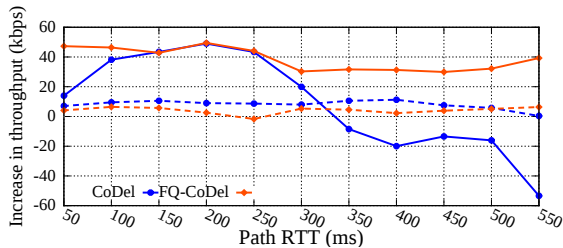
As deployments will experience various RTTs, we compare *Rural* and *Default* parameterisations with a 1 Mbps link across a range of end-to-end RTTs from 50 ms to 550 ms. The improvement  $imp_M$  for a given metric  $M$  is computed as  $imp_M = (M_{Default} - M_{Rural}) / M_{Default}$  (for FCT in Fig. 5a) and the increase  $inc_N$  for a metric  $N$  is computed as  $inc_N = N_{Rural} - N_{Default}$  (for queuing delay in Fig. 5b and the bulk throughput in Fig. 5c).



(a) Repeated download of medium-sized files.



(b) Thin-stream flow.



(c) Bulk transfer flow.

Figure 5: Impact of *Rural* parameterisation for various base RTTs (solid lines: low load; dash lines: high load).

Figure 5a shows that *Rural* parameterisation improves the FCT for every RTT. In Fig. 5b, we can see that the queuing delay for thin streams is increased with CoDel with *Rural* parameterisation, because  $\tau_{Rural} > \tau_{Default}$ , but this is not the case with FQ-CoDel, due to its prioritisation scheme. Except when the RTT is higher than 350 ms and CoDel is used, *Rural* parameterisation increases the bulk throughput.

#### F. Flows with different RTTs

We next compare the performance of *Rural* and *Default* parameterisations by monitoring a repeated-download flow competing with another similar, “background” flow but having a different base RTT. The RTT ratios of the two flows were varied between 0.09 and 11, i.e.,  $RTT \in [50, 550]$  ms.

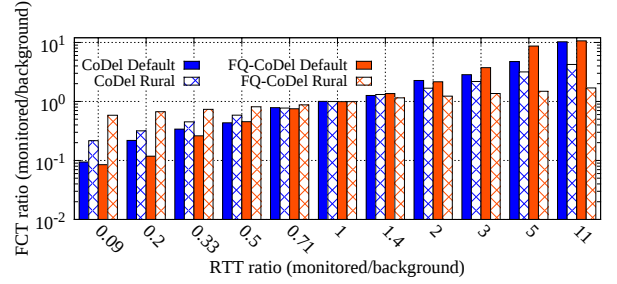


Figure 6: FCT with two competing RTTs.

The results presented in Fig. 6 show that both AQMs with *Rural* parameterisation reduce the FCT for the large RTT flow and reduces the difference of download time for flows that experience different base RTT: the FCT ratio ( $\frac{FCT_{Monitored}}{FCT_{Background}}$ ) moves closer to one with *Rural* parameterisation, for both small and large RTT ratios.

#### G. Discussion

We do not claim that AQM would provide the best quality of service for various applications when the link capacity or traffic patterns are different; however, we show that (1) adequately tuned AQM schemes can improve the FCTs and the link utilisation, and that (2) CoDel can be tuned to achieve various trade-offs.

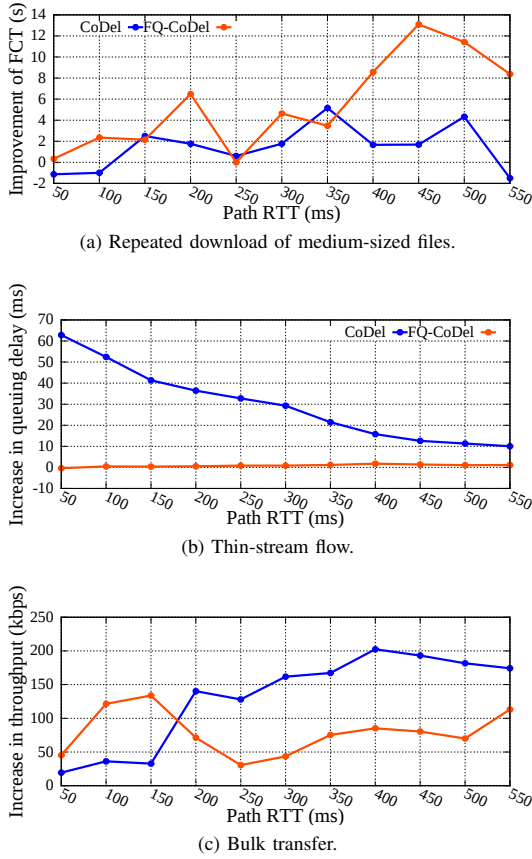
### V. EXPERIMENTAL EVALUATION

#### A. Performance over a delay-emulated test bed

The experiments described in Section III-B were repeated with *Rural* parameterisation and the results compared with those presented in that section. Figure 7 presents the improvement in terms of FCT and the increase in queuing delay and throughput over *Default* parameterisation as explained in § IV-E. ICMP ping flows were generated to measure the queuing delay, shown in Fig. 7b. Figure 7a shows that *Rural* parameterisation improves the FCT for most RTT values. As expected, the higher throughput for the bulk flow, presented in Fig. 7c, results in a higher queuing delay. We notice a difference between the results presented in Figures 5 and 7, which is due to the different traffic characteristics and different source codes of FQ-CoDel in ns-2 and Linux.

#### B. Performance over a satellite network

Finally, we present results obtained with a operational rural satellite broadband service: the forward satellite link is between  $snd_1$  and  $R_1$  in Fig. 1, the bottleneck is restricted to 1 Mbps and AQM is used at  $R_1$ ; in addition to the queue management schemes tested so far, this link allowed also the

Figure 7: Base RTT and *Rural* and *Default* parameterisations

use of Stochastic FQ (SFQ). Figure 8 shows that introducing CoDel or FQ-CoDel with *Default* parameterisation results in (1) a higher FCT than DropTail or SFQ and (2) a lower RTT. With CoDel or FQ-CoDel with *Rural* parameterisation, the RTT is lower than with the other evaluated schemes, without negative impact on the FCT. With adequately parameterised AQM, as opposed to with DropTail, the latency is reduced by  $\approx 60\%$  (i.e., by 300 ms), which would seriously increase rural broadband users' experience.

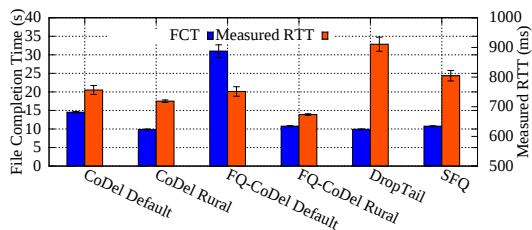


Figure 8: FCT and measured RTT over a satellite network

## VI. CONCLUSION

This paper characterised the current default parameterisation of CoDel and FQ-CoDel within a limited-capacity and large-

RTT scenario, which is representative of a rural broadband access network. We show that the default parameters do not offer the desired control of queuing for a single-queue AQM, as claimed in [3]. Consistent with other analysis of PIE, we have proven CoDel can also be tuned to better match the needs of a “non-default” network scenario.

We propose a configuration with a larger target delay and update interval for CoDel: this resulted in reduced download time of medium-sized files, higher bottleneck utilisation and limited queuing delay to an acceptable level, for different traffic profiles. However, this also resulted in queuing delay larger than the *target delay*. Under a less-controlled queuing delay, the tested flow isolation technique, FQ-CoDel, with our parameterisation, provided low latency and was seen to significantly improve performance for delay-sensitive traffic. FQ-CoDel also features priority queuing which improves the performance of applications transmitting a low amount of data. However, we note that in some cases, flow separation may be difficult (e.g., when encryption is used) and CoDel can then control delay to a reasonable level. With our parameterisation, CoDel and FQ-CoDel would allow a better quality of service for users browsing the web from a rural location since the latency is reduced and the capacity exploited close to the one without AQM schemes. As a future work, we will evaluate the benefits of our parameterisation for real-time services.

## ACKNOWLEDGMENT

This research was supported by the RCUK DE award to the dot.rural Digital Economy Hub; EP/G066051/1 and by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

## REFERENCES

- [1] J. Gettys, “Bufferbloat: Dark buffers in the Internet,” *IEEE Internet Computing*, vol. 15, no. 3, pp. 96–96, 2011.
- [2] S. Floyd and V. Jacobson, “Random early detection gateways for congestion avoidance,” *Networking, IEEE/ACM Transactions on*, 1993.
- [3] K. Nichols and V. Jacobson, “Controlling queue delay,” *ACM Queue*, 2012.
- [4] R. Pan, P. Natarajan, C. Piglion, M. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, “PIE: A lightweight control scheme to address the Bufferbloat problem,” in *IEEE HPSR 2013*, 2013, pp. 148–155.
- [5] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, “Flowqueue-codel (work in progress),” 2013, IETF.
- [6] SamKnows - Commission for Rural Communities, “Mind the Gap: Digital England – A Rural Perspective,” June 2014.
- [7] L. Townsend, A. Sathiseelan, G. Fairhurst, and C. Wallace, “Enhanced broadband access as a solution to the social and economic problems of the rural digital divide,” *Local Economy*, 2013.
- [8] I. Järvinen and M. Kojo, “Evaluating CoDel, PIE, and HRED AQM Techniques with Load Transients,” *LCN*, 2014.
- [9] J. Schwarzmann, D. Wagner, and M. Kühlewind, “Evaluation of ARED, CoDel and PIE,” *20th EUNICE*, 2014.
- [10] R. Pan, P. Natarajan, C. Piglion, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, “PIE: A lightweight control scheme to address the bufferbloat problem. Further Studies– PIE for Data Centers,” in *IETF 86 - ICCRG presentation*, 2013.
- [11] G. White and R. Pan, “A PIE-Based AQM for DOCSIS Cable Modems (work in progress),” 2014, IETF.
- [12] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, “Controlled delay active queue management (work in progress),” 2014, IETF.

# On the coexistence of AQM and LBE

Nicolas Kuhn<sup>†</sup>, Xavier Corbillon<sup>†</sup>, Emmanuel Lochin<sup>‡</sup>

<sup>†</sup>IMT Télécom Bretagne, IRISA, France

<sup>‡</sup>Université de Toulouse, ISAE, France

**Abstract**—Active Queue Management (AQM) schemes are essential to guarantee low delay for latency sensitive applications. Their deployment is a required step to tackle bufferbloat. It has been shown that AQM and Lower-than Best Effort (LBE), that carry non-priority traffic, can hardly coexist. In this paper, we show that with adequate parameterizations, it is possible to provide both low end-to-end delay for latency sensitive applications with AQM and low priority traffic with LBE protocols.

**Keywords**—AQM, LBE, LEDBAT, CDG

## I. INTRODUCTION

Active queue management (AQM) schemes can be introduced in network routers with the rationale of controlling the amount of buffering and reducing the loss synchronizations. Large buffers and the absence of AQM deployment resulted in huge latencies; this problem is known as bufferbloat [1]. Even if there is some debate on whether bufferbloat is a real issue [2], it has been observed in cellular networks [3], [4]. Since AQM can control the queuing delay, their deployment is “one piece to the solution of bufferbloat” [5]. The first AQM proposals, such as Random Early Detection (RED) [6], dating back more than a decade, have been reported to be usually turned off, mainly because of the difficulty to tune their parameters. Even if Adaptive RED (ARED) [7] has been proposed to ease the parameterization of RED, it has been proposed to control the buffering mainly when TCP flows are in the network. Controlled Delay (CoDel) [5] and Proportional Integral controller Enhanced (PIE) [8] are two recently proposed AQM schemes that have been designed to tackle bufferbloat by lowering the queuing delay while addressing RED’s stability issues and acknowledge the presence of flows that do not react to congestion signals, such as UDP.

To ensure the safe deployment of AQM schemes and their ability to tackle the bufferbloat, the IETF works on characterization guidelines [9] which advise to consider a scenario where the impact of introducing AQM on the fairness of Lower-than-Best Effort (LBE) protocols is assessed. Indeed, as shown in [10], when there are AQMs deployed in one router, LBE may fail in their design goal, that is to exploit the remaining and unused capacity of a link. To further assess this problem, this paper presents simulation results where we propose different parameterizations of the AQM and the LBE protocols used in [10]. We expect to verify the validity of their conclusions when the targets of the protocols are different.

The LBE protocols that are considered in our work are LEDBAT [11] and CDG [12]. We focus on LEDBAT because it is deployed in BitTorrent and on CDG because it has been shown to be a good LBE candidate [13], [14]. The

difference with the work in [10] is that we consider a better parameterization for LEDBAT (following [15], [16]) and that they do not consider CDG which estimates the state of the router’s queue and might therefore be able to cope with AQM. Since LBE schemes use estimations of the state of the queue or the queuing delay, an AQM that allows higher amount of buffering than the ones considered in [10] should be considered: we complement their conclusions by adding simulation results with PIE that allows on average 20 ms queuing delay, whereas CoDel allows only 5 ms queuing delay.

The remainder of this paper is organized as follows. Section II presents the LBE protocols and the AQM schemes exploited in this article. In Section III, we propose to evaluate the interaction between LBE protocols and AQM schemes by considering two flows sharing a bottleneck and by observing the evolution of their congestion window. Section IV considers the same traffic, but focus on how the capacity is shared. This work is concluded in Section V.

## II. DESCRIPTION OF THE LBE AND AQM SCHEMES

### A. LBE candidates

1) *Low Extra Delay Background Transport (LEDBAT)*: LEDBAT is characterized by the following parameters: target queuing delay ( $\tau_{LB}$ ), impact of the delay variation ( $\gamma = 1/\tau_{LB}$ ), minimum One-Way Delay ( $D_{min}$ ) and current One-Way Delay ( $D_{ack}$ ). For each ACK received, the new congestion window ( $cwnd$ ) value is updated according to:

$$cwnd = cwnd + \frac{\gamma(\tau_{LB} - (D_{ack} - D_{min}))}{cwnd} \quad (1)$$

LEDBAT congestion control is based on queuing delay variations (*i.e.*, the queuing delay is used as a primary congestion notification), estimated by  $(D_{ack} - D_{min})$ . The target queuing delay  $\tau_{LB}$  embodies the maximum queuing time that LEDBAT is allowed to introduce.

2) *CAIA-Delay Gradient (CDG)*: is a congestion control that reacts to packet loss and delay variations. CDG uses delay-gradient measurements to estimate  $Q_{state}$ , the state of a network queue and detect non congestion related losses. We analyzed the original FreeBSD implementation of CDG [12] to implement a version of CDG for NS-2. More information on the validation of our implementation can be found in [17].

### B. AQM schemes

This section provides details on ARED, PIE and CoDel; these AQM schemes feature a similarly-named parameter,



$\tau_{AQM}$ , that embodies the acceptable standing queuing delay above which they are increasing the rate at which they drop packets.  $\tau_{AQM}$  is used in a different manner in each of these schemes.

1) *Adaptive RED* [6], [7]: RED is an AQM that randomly drops incoming packets with a probability  $p$  that is related to the size of the queue,  $qlen$ , as explained in (2): when the queue size is lower than  $min_{th}$ , no packets are dropped; when the queue size is  $max_{th}$ , the dropping probability is  $max_p$ .

$$p = \begin{cases} 0 & \text{if } qlen < min_{th} \\ \frac{max_p \times (qlen - min_{th})}{max_{th} - min_{th}} & \text{if } min_{th} < qlen < max_{th} \\ 1 & \text{if } qlen > max_{th} \end{cases} \quad (2)$$

However, RED encountered deployment issues, as its performance could hardly be estimated in advance. The optimal parameterization of  $min_{th}$ ,  $max_{th}$  and  $max_p$  behaviour depends on the congestion level and lower layers characteristics.

To overcome from these deployment issues, ARED has been proposed. ARED sets  $min_{th}$  and  $max_{th}$  according to (3) [7].  $\tau_{ARED}$  is expressed in seconds and  $C_{pkts}$  is the link capacity expressed in packets per seconds.

$$\begin{aligned} min_{th} &= \max \left[ 5, \frac{\tau_{ARED} \times C_{pkts}}{2} \right] \\ max_{th} &= 3 \times min_{th} \end{aligned} \quad (3)$$

Every  $\lambda_{ARED} = 500$  ms, ARED adapts the values of  $max_p$ , the dropping probability when the queue size is  $max_{th}$ , so that the average queue size oscillates between  $min_{th}$  and  $max_{th}$ . More details on the update of  $max_p$  can be found in [7].

2) *Controlling Queue Delay* [5]: CoDel maintains two states: dropping and non-dropping modes. At initialization, CoDel is in non-dropping mode and  $\mu_{CoDel}$ , the interval, is set to  $\lambda_{CoDel} = 100$  ms. Then every  $\mu_{CoDel}$ , CoDel measures the maximum queuing delay experienced by the dequeued packets. If the measured queuing delay goes above  $\tau_{CoDel}$ , the next packet to be dequeued is dropped and  $\mu_{CoDel}$  is updated following (4) where  $n_{drop}$  is the number of consecutive drops.  $\tau_{CoDel}$  represents the maximum acceptable standing queue delay above which it is dropping or preparing to drop.

$$\mu_{CoDel} \leftarrow \lambda_{CoDel} / \sqrt{n_{drop}}. \quad (4)$$

3) *Proportional Integral controller Enhanced (PIE)* [8]: PIE randomly drops an incoming packet with a probability  $p$  which is updated every  $\lambda_{PIE} = 30$  ms according to (5).  $E[T]$  and  $E[T]_{old}$  represent the current and previous estimation of the queuing delay.  $\tau_{PIE}$  represents the queuing delay that PIE will try to maintain.  $\alpha$  determines how the deviation of current latency from  $\tau_{PIE}$  affects the drop probability, whereas  $\beta$  exerts additional adjustments depending on whether the latency is trending up or down.  $\alpha$  and  $\beta$  will determine how rapidly drop probabilities will be increased (or decreased) from 0 to 1. PIE scales up these parameters to make  $p$  adapt more rapidly when  $p \notin [0.01, 0.1]$ .

$$p_{drop} = p_{drop} + \alpha \times (E[T] - \tau_{PIE}) + \beta \times (E[T] - E[T]_{old}) \quad (5)$$

### C. Parameterization of the AQM and LBE

We will use the default parameterizations of the AQM schemes:  $\tau_{ARED} = 5$  ms,  $\tau_{CoDel} = 5$  ms and  $\tau_{PIE} = 20$  ms. It is worth mentioning that even if  $\tau_{ARED}$  is 5 ms, it may target higher queuing delays. As illustrated in (3),  $min_{th}$  can not be below 5 packets. The *real* target delay,  $\tau'_{ARED}$ , may be much higher than 5 ms: if the capacity of the bottleneck is 10 Mbps and the packet size is 1500 B,  $\tau_{ARED} \times C_{pkts} / 2 = (0.005 \times 10^7 / (1500 \times 8)) / 2 \approx 2$ . Then, the used parameters are  $min_{th} = 5$ ,  $max_{th} = k \times min_{th} = 15$  and  $\tau'_{ARED} = (k + 1) / 2 \times min_{th} = 10$  ms (more details can be found in the 4<sup>th</sup> footnote of [7, p.7]). The default value for  $\tau_{LB}$  is 100 ms. It has however been shown not to be optimal in [15], [16]. Therefore, we will consider a range of  $\tau_{LB}$  in  $\{5; 25; 100\}$  ms.

Because of (1) AQM schemes would allow only  $\approx \tau_{AQM}$  of queuing delay and of (2) the maximum queuing time that LEDBAT allows itself to introduce is  $\tau_{LB}$ , we suspect that by construction, these mechanisms can hardly coexist.

## III. CONGESTION WINDOW EVOLUTION

### A. Topology and traffic

Figure 1 describes the topology used in this section. The traffic generated is the following: (1) one non application-limited LBE bulk flow using LEDBAT or CDG as congestion control policy between  $snd_1$  and  $dest_1$ ; (2) one non application-limited TCP bulk flow using NewReno as congestion control policy between  $snd_2$  and  $dest_2$ . The Initial congestion Window (IW) is set to 10 packets for (2); the SACK option is enabled for (1) and (2). To avoid any synchronization effect, (1) randomly starts in  $[0; 1]$  s and (2) in  $[1; 2]$  s.

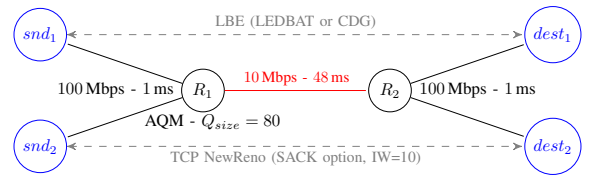


Figure 1: Topology and traffic

Each scenario is entitled “LBE-AQM”. As one example, “LB25-ARED” is the scenario where the LBE protocol is LEDBAT with  $\tau_{LB} = 25$  ms (§ II-A1) and the AQM is ARED (§ II-B1). Each simulation run lasts 50 seconds. Our work is based on NS-2 simulations; LEDBAT, CDG and any TCP variants used in this article use the TCP-Linux updated to linux kernel version 3.17.4 in NS-2.<sup>1</sup>

<sup>1</sup>More details at: <http://heim.ifi.uio.no/michawe/research/tools/ns/index.html>

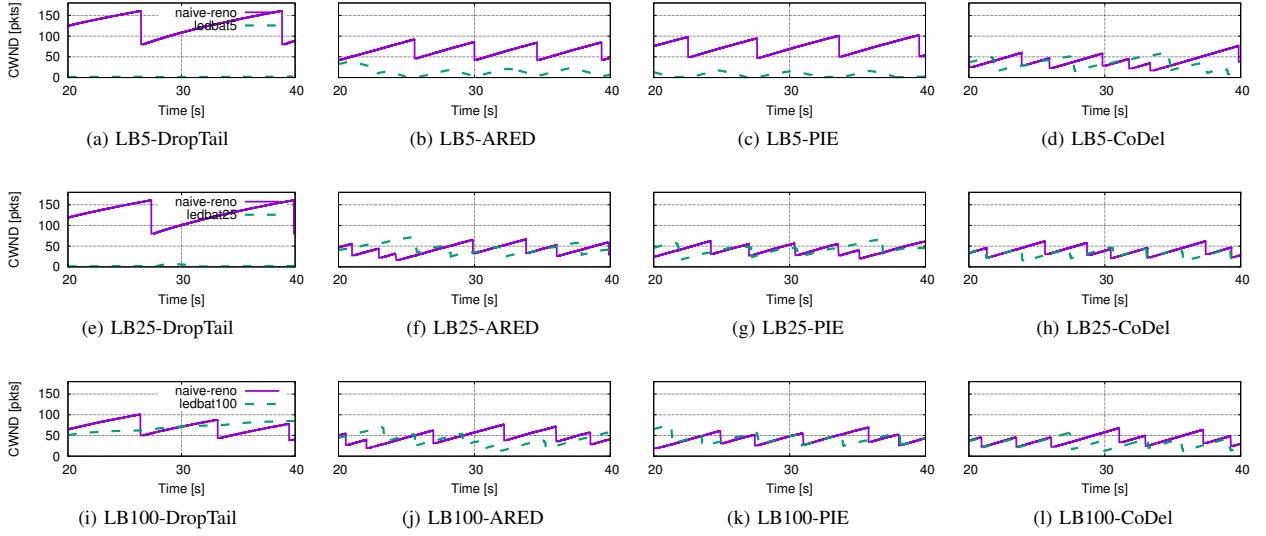


Figure 2: Congestion window evolution: depending on combination of  $\tau_{LB}$  and  $\tau_{AQM}$ , LEDBAT may provide LBE service

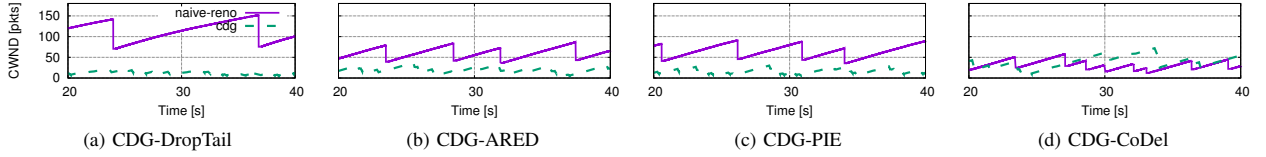


Figure 3: Congestion window evolution: CDG may provide LBE service

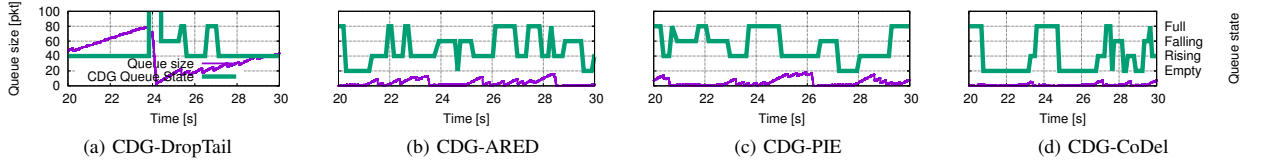


Figure 4: Queue size and queue state estimations in CDG

### B. $\tau_{LB}$ and $\tau_{AQM}$

Figure 2 shows the evolution of the congestion window of the TCP NewReno and the LEDBAT flow, for various AQM schemes and different values of  $\tau_{LB}$ .

With DropTail, we see the same problem as the one illustrated in [16, Fig. 1] and in [18], that is LEDBAT not providing an LBE service because of how  $D_{min}$  is updated and on the high target value. Indeed, when  $\tau_{LB} = 5$  ms or  $\tau_{LB} = 25$  ms, LEDBAT can provide an LBE service.

The queuing delay is maintained around 10 ms with ARED (see § II-C for more explanations) and around 20 ms with

PIE ( $\tau_{PIE} = 20$  ms). LEDBAT will let itself introduce  $\tau_{LB}$  queuing delay; this results in LEDBAT not being LBE when  $\tau_{LB} \geq 25$  ms. With  $\tau_{LB} = 5$  ms (which is lower than  $\tau_{PIE}$  and  $\tau'_{ARED}$ ), LBE is possible, which is consistent with our previous work in [15], [16]. This results however is poor performance with DropTail, as seen in Figure 2a.

With CoDel as an AQM, no values of  $\tau_{LB}$  let LEDBAT able to provide an LBE service, which is in accordance with the results published in [10]. This is due to the low value of  $\tau_{CoDel}$  that makes CoDel try to maintain a 5 packets queue. In these conditions, there is not much room for an LBE service.

### C. AQM makes CDG estimate small buffers

Figure 3 shows the evolution of the congestion window of the TCP NewReno and the CDG flows, for various AQM schemes and Figure 4 shows the evolution of the queue size and the queue state estimations in CDG. The congestion window of CDG is lower than the one of TCP New Reno, without AQM or with PIE or ARED. CDG seems able to carry out LBE services in these conditions. On the congestion control point of view, because AQM schemes have fixed targeted queuing delay, they tend to be seen as a small buffer. Therefore, with PIE and ARED, CDG estimates that the queue is full when the queuing delay reaches the targeted queuing delay, as illustrated in Figure 4. Again, with CoDel, the targeted queuing delay is so small that, even with CDG, no LBE service seems possible.

### IV. CAPACITY SHARING

To verify the assertions of § III, we propose to evaluate the capacity sharing between LBE and Best-Effort with and without AQM and with different LBE variants. The topology and traffic generated is consistent with what is presented in § III-A. Every run lasts 50 s and is repeated 5 times. We gather the goodput, sampled every second, at  $dest_1$  and  $dest_2$ . We keep only 70 % of the latest samples to remove the impact of the slow start. We compute  $C_{BE}$  (resp.  $C_{LBE}$ ) that is the median capacity used by Best-Effort flows (resp. LBE) and  $C_{bot}$  is the capacity of the bottleneck. We represent  $\rho = C_{LBE}/(C_{LBE} + C_{BE})$  and  $\eta = (C_{LBE} + C_{BE})/C_{bot}$  in Table I.

Table I: Possible coexistence of LBE and AQM

	DropTail	ARED $\tau = 5$ ms	PIE $\tau = 20$ ms	CoDel $\tau = 5$ ms
LEDBAT $\tau_{LB} = 5$ ms	$\rho = 0.004$ $\eta = 1$	$\rho = 0.24$ $\eta = 0.94$	$\rho = 0.10$ $\eta = 0.94$	$\rho = 0.52$ $\eta = 0.94$
LEDBAT $\tau_{LB} = 25$ ms	$\rho = 0.004$ $\eta = 1$	$\rho = 0.45$ $\eta = 1$	$\rho = 0.43$ $\eta = 1$	$\rho = 0.45$ $\eta = 0.95$
LEDBAT $\tau_{LB} = 100$ ms	$\rho = 0.009$ $\eta = 1$	$\rho = 0.47$ $\eta = 1$	$\rho = 0.46$ $\eta = 1$	$\rho = 0.45$ $\eta = 0.92$
CDG	$\rho = 0.070$ $\eta = 1$	$\rho = 0.33$ $\eta = 1$	$\rho = 0.31$ $\eta = 1$	$\rho = 0.53$ $\eta = 0.98$

When there is an AQM, both CDG and LEDBAT increase their channel occupancy. When the AQM is CoDel, no parameterization of LEDBAT nor CDG can carry out LBE services, due to the small buffer that is maintained by CoDel.  $\rho$  is indeed around 50 % in all cases. With PIE and ARED, as more buffering is allowed, LEDBAT with a target delay of 5 ms and CDG can provide LBE services. When the target delay is higher than 25 ms,  $\rho$  gets close to 50 %.

### V. CONCLUSION

This paper assesses the capacity of CDG and LEDBAT to carry out LBE services when there is an AQM on the path. We summarize the results presented in this paper in Table II. We believe that the deployment of CDG or LEDBAT with a target delay of 5 ms as LBE. Also, because the target delay of CoDel is lower than the one of PIE and ARED, no LBE service is possible if CoDel is deployed, whereas it may be possible with ARED or PIE. We expect, as a future work, to verify this conclusions with experiments in a real testbed, using the recently posted implementation of CDG for Linux kernels [19].

Table II: Possible coexistence of LBE and AQM

	DropTail	ARED $\tau = 5$ ms	PIE $\tau = 20$ ms	CoDel $\tau = 5$ ms
LEDBAT $\tau_{LB} = 5$ ms	✓	✓	✓	✗
LEDBAT $\tau_{LB} = 25$ ms	✓	✗	✗	✗
LEDBAT $\tau_{LB} = 100$ ms	✗	✗	✗	✗
CDG	✓	✓	✓	✗

### ACKNOWLEDGMENT

This research was supported by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

### REFERENCES

- [1] “BufferBloat: What’s wrong with the Internet?” *Queue ACM*, 2012.
- [2] M. Allman, “Comments on Bufferbloat,” *SIGCOMM Computer Communication Review*, Jan. 2013.
- [3] H. Jiang, Z. Liu, Y. Wang, K. Lee, and I. Rhee, “Understanding Bufferbloat in cellular networks,” in *CellNet*. ACM, 2012.
- [4] S. Alfredsson, G. Del Giudice, J. Garcia, A. Brunstrom, L. De Cicco, and S. Mascolo, “Observations of bufferbloat in swedish cellular networks,” in *SNCNW*, 2013.
- [5] K. Nichols and V. Jacobson, “Controlling queue delay,” *ACM Queue*, 2012.
- [6] S. Floyd and V. Jacobson, “Random early detection gateways for congestion avoidance,” *Networking, IEEE/ACM Transactions on*, 1993.
- [7] S. Floyd, R. Gummadi, and S. Shenker, “Adaptive red: An algorithm for increasing the robustness of red’s active queue management,” 2001.
- [8] R. Pan, P. Natarajan, C. Piglion, M. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, “PIE: A lightweight control scheme to address the Bufferbloat problem,” in *IEEE HPSR 2013*, 2013, pp. 148–155.
- [9] N. Kuhn, P. Natarajan, D. Ros, and N. Khademi, “AQM Characterization Guidelines,” 2015, IETF.
- [10] Y. Gong, D. Rossi, C. Testa, S. Valenti, and D. Taht, “Fighting the bufferbloat: on the coexistence of AQM and low priority congestion control (extended version),” *Computer Networks*, 2014.
- [11] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, “Low extra delay background transport (LEDBAT),” RFC Editor, RFC 6817, Dec. 2012.
- [12] D. A. Hayes and G. Armitage, “Revisiting TCP congestion control using delay gradients,” in *Networking 2011*. Springer, 2011.
- [13] G. Armitage and N. Khademi, “Using delay-gradient tcp for multimedia-friendly transport in home networks,” in *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, Oct 2013, pp. 509–515.
- [14] M. Welzl, S. Gjessing, and N. Khademi, “Less-than-best-effort service for community wireless networks: Challenges at three layers,” in *Wireless On-demand Network Systems and Services (WONS)*, 2014.
- [15] N. Kuhn, O. Mehani, A. Sathiseelan, and E. Lochin, “Less-than-Best-Effort Capacity Sharing over High BDP Networks with LEDBAT,” in *VTC 2013-Fall, 78th Vehicular Technology Conference*, 2013.
- [16] S. Q. V. Trang, N. Kuhn, E. Lochin, C. Baudoin, E. Dubois, and P. Gelard, “On the existence of optimal ledbat parameters,” in *IEEE International Conference on Communications (ICC)*, 2014.
- [17] N. Iya, N. Kuhn, F. Verdichio, and G. Fairhurst, “Analyzing the impact of bufferbloat on latency-sensitive applications,” in *IEEE International Conference on Communications (ICC)*, 2015.
- [18] D. Ros and M. Welzl, “Assessing ledbat’s delay impact,” *Communications Letters, IEEE*, vol. 17, no. 5, pp. 1044–1047, May 2013.
- [19] K. K. Jonassen, “Implementing CAIA Delay-Gradient in Linux,” *Master Thesis*, 2015. [Online]. Available: [http://folk.uio.no/kennetkl/jonassen\\_thesis.pdf](http://folk.uio.no/kennetkl/jonassen_thesis.pdf)



# Improving PIE's performance over high-delay paths

**Abstract**—Bufferbloat is excessive latency due to over-provisioned network buffers. PIE and CoDel are two recently proposed Active Queue Management (AQM) algorithms, designed to tackle bufferbloat by lowering the queuing delay without degrading the bottleneck utilization. PIE uses a proportional integral controller to maintain the average queuing delay at a desired level; however, large Round Trip Times (RTT) result in large spikes in queuing delays, which induce high dropping probability and low utilization. To deal with this problem, we propose Maximum and Average queuing Delay with PIE (MADPIE). Loosely based on the drop policy used by CoDel to keep queuing delay bounded, MADPIE is a simple extension to PIE that adds deterministic packet drops at controlled intervals. By means of simulations, we observe that our proposed change does not affect PIE's performance when  $RTT < 100$  ms. The deterministic drops are more dominant when the RTT increases, which results in lower maximum queuing delays and better performance for VoIP traffic and small file downloads, with no major impact on bulk transfers.

**Keywords**—Active queue management; Internet latency; PIE.

## I. INTRODUCTION

ACTIVE queue management (AQM) schemes can be introduced in network routers with the goal of controlling the amount of buffering and reducing the loss synchronization. Large buffers and the absence of AQM deployment have resulted in huge latencies; this problem is known as bufferbloat [1]. Since AQM can control the queuing delay, its deployment “can significantly reduce the latency across an Internet path” [2]. The first AQM proposals, such as Random Early Detection (RED) [3], dating back more than a decade, have been reported to be usually turned off, mainly because of the difficulty to tune their parameters. Even if Adaptive RED (ARED) [4] was proposed to ease the parameterization of RED, it was designed to control the buffering when traffic is composed mainly of TCP flows. Proportional Integral controller Enhanced (PIE) [5] and Controlled Delay (CoDel) [6] are two recent AQM schemes that have been designed to tackle bufferbloat by lowering the queuing delay while addressing RED's stability issues and considering the presence of transports that do not react to congestion signals, such as UDP.

PIE and CoDel share two main concepts that can be mapped into two algorithm parameters: (1) a *target delay* ( $\tau$ ) represents the acceptable standing queuing delay above which an AQM drops packets more aggressively; (2) an update *interval* ( $\lambda$ ) represents the reactivity of an AQM. These two parameters have different usages in the two algorithms. In CoDel,  $\tau$  embodies an upper bound on the allowed queuing delay; if the minimum queuing delay over an interval of duration  $\lambda$  is higher than  $\tau$ , then a packet is dropped with probability 1, else no packet is dropped. PIE uses  $\tau$  to increase or decrease a dropping *probability*, based on the deviations of estimated

queuing delay from such target delay:  $\tau$  is therefore the desired average queuing delay.

CoDel has been shown to have auto-tuning issues and its performance is sensitive to the traffic load [7]. Also, its default 5 ms of maximum allowed queuing delay can be damaging for low-speed bottlenecks [8] and its interval value is based on the assumption that the Round Trip Time (RTT) is 100 ms [9]. On the other hand, PIE has been shown to be less sensitive to traffic loads [7], its default target delay of 20 ms should be less problematic with low capacity bottlenecks, and it does not make assumptions on the RTT. However, in this paper, we show that PIE is sensitive to the RTT, as we observe wide oscillations in queuing delay when the RTT increases. This results in temporarily high maximum queuing delay, high dropping probability, and low bottleneck utilization.

To reduce the RTT sensitivity of PIE and improve the performance of latency sensitive applications over large RTT paths (e.g., rural broadband or satellite access), our proposal, Maximum and Average queuing Delay with PIE (MADPIE) extends PIE by adding deterministic drops to prevent the queuing delay from growing beyond a critical value, loosely mimicking CoDel's drop policy.

The rest of this article is organized as follows. Section II details the MADPIE algorithm. In Section III, by means of simulations we illustrate the issues that PIE faces when the RTT increases, and how the deterministic drops in MADPIE help to correct those issues. Section IV provides an evaluation of the trade-off between allowing more bandwidth for bulk transfers and improving the performance of latency sensitive applications with MADPIE and PIE, as opposed to DropTail. Finally, Section V concludes this work.

## II. ADDING DETERMINISTIC DROPS TO PIE

PIE drops an incoming packet when  $p \leq p_{drop}$ , where  $p$  is drawn at random from a uniform distribution in  $[0, 1]$ , and  $p_{drop}$  is an internal variable updated every  $\lambda = 30$  ms according to:

$$p_{drop} \leftarrow p_{drop} + \alpha \times (E[T] - \tau) + \beta \times (E[T] - E[T]_{old}). \quad (1)$$

$E[T]$  and  $E[T]_{old}$  represent the current and previous estimation of the queuing delay.  $\tau$  is PIE's target delay.  $\alpha$  determines how the deviation of current queuing delay from  $\tau$  affects the drop probability, whereas  $\beta$  exerts additional adjustments depending on whether the queuing delay is trending up or down.

MADPIE uses the same random drop policy as PIE, the only difference between the two algorithms being that we add a deterministic drop policy. MADPIE requires only one additional parameter: the queuing delay  $\tau_{DD}$  above which deterministic drops occur. An indicator variable  $p_{max}$ , initialized to 0, tells whether a packet must be dropped ( $p_{max} = 1$ ) or not ( $p_{max} = 0$ ) by the deterministic policy. Every  $\lambda$ , if the estimated queuing delay is  $> \tau_{DD}$ ,  $p_{max}$  is set to 1. Then,

if a packet is *not* dropped nor marked by the random drop algorithm and  $p_{max} = 1$ , then a packet is dropped or marked and  $p_{max}$  is reset to 0. Thus, there can be a maximum of one deterministic drop every  $\lambda$ .

### III. PROOF OF CONCEPT

The aim of this section is to illustrate how MADPIE's behaviour differs from that of PIE when the RTT increases.

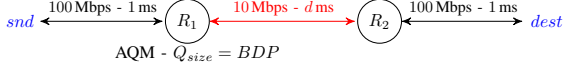


Fig. 1. Topology used to prove the MADPIE concept.

Fig. 1 presents the topology used in this section. The one-way delay of the bottleneck link is set to  $d = 48$  ms or  $d = 248$  ms (which corresponds to a base RTT<sup>1</sup> of 100 ms and 500 ms, respectively). The queue size at  $R_1$  is set to the Bandwidth-Delay Product (BDP). The AQM introduced at  $R_1$  is either PIE ( $\tau = 20$  ms and  $\lambda = 30$  ms) or MADPIE ( $\tau = 20$  ms,  $\lambda = 30$  ms,  $\tau_{DD} = 30$  ms). We simulate 10 TCP bulk flows from *snd* to *dest*, using CUBIC as congestion control policy, for 300 s. The Initial congestion Window (IW) is set to 10 packets and the SACK option is enabled. The flows randomly start in  $[0; 1]$  s. All TCP variants used in this article were provided by the NS-2 TCP Linux module updated to linux kernel version 3.17.4.<sup>2</sup>

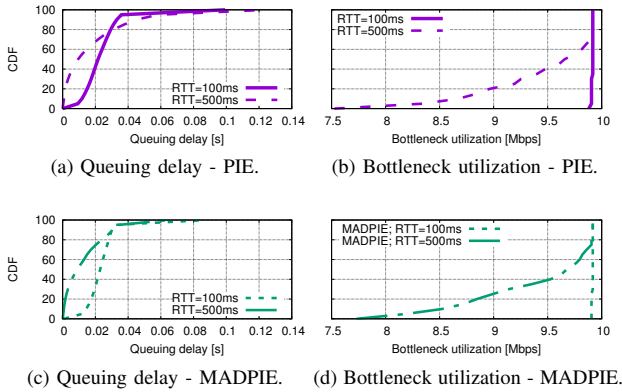


Fig. 2. Bottleneck utilization and queuing delay.

In Fig. 2, we present the Cumulative Distribution Function (CDF) of the queuing delay (measured per packet) and the bottleneck utilization (sampled every second). When the RTT is 100 ms, apart from the maximum queuing delay that is slightly lower with MADPIE than with PIE ( $\approx 100$  ms with PIE,  $\approx 80$  ms with MADPIE), there is no noticeable performance difference between MADPIE and PIE. When the RTT is 500 ms, Fig. 2a and 2c show that for 20 % of the samples, the

queuing delay is higher than 20 ms with MADPIE whereas it is higher than 30 ms with PIE. Also, with MADPIE as opposed to with PIE, the maximum queuing delay is reduced by  $\approx 60$  ms. It is worth pointing out that 90 % of the samples show a queuing delay lower than 30 ms (that is  $\tau_{DD}$ ) with MADPIE as opposed to 50 ms with PIE. Fig. 2b and 2d show that this latency reduction does not induce a lower bottleneck utilization.

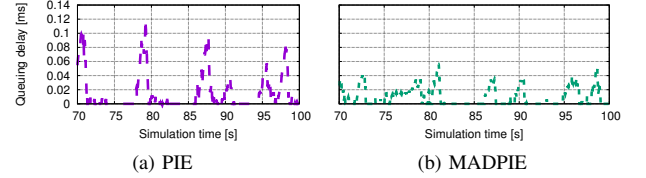


Fig. 3. Evolution of queuing delay over time with  $RTT = 500$  ms.

This latency reduction provided by MADPIE can be further explained by looking at the queuing delay evolution in Fig. 3. With PIE, a higher RTT results in wider oscillations in queuing delay: as the queuing delay gets much higher than  $\tau$ , the dropping probability increases in order to maintain a lower queuing delay. This however results in a momentarily empty buffer. PIE's burst allowance of 100 ms lets the queuing delay to frequently grow above 100 ms, as the buffer was previously empty. With MADPIE, it is possible to initially allow the same bursts, but the deterministic drops would then prevent an excessive growth of both the queuing delay and the drop probability if the buffer is frequently empty then full, which is what happens when the RTT is 500 ms.

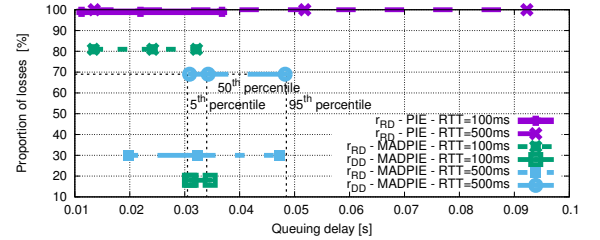


Fig. 4. Queuing delay and proportion of losses.

To better understand how MADPIE's behaviour differs from that of PIE when the RTT increases, we look at the contribution of random and deterministic drops to the overall drop rate. Let us denote by  $n_{DD}$ ,  $n_{RD}$ ,  $n_{BO}$  and  $n_{tot}$  the number of drop events induced by a deterministic drop (only MADPIE), a random drop (PIE and MADPIE), a buffer overflow (PIE and MADPIE) and the total number of drops, respectively. Let  $r_x = n_x / n_{tot}$  be the proportion of drop events of type  $x$ . Fig. 4 shows  $r_{RD}$  and  $r_{DD}$  as a function of the queuing delay when the drop occurred. As one example (dashed lines in Fig. 4), when the RTT is 500 ms and the AQM is MADPIE,  $r_{DD} \approx 70$  % and when the deterministic drops occurred, the 5 % percentile of the queuing delay was  $\approx 30$  ms, the 50 %

<sup>1</sup>That is, the minimum RTT, without any queuing delays.

<sup>2</sup>More details at: <http://heim.ifi.uio.no/michawe/research/tools/ns/index.html>

percentile  $\approx 34$  ms and the 95% percentile  $\approx 48$  ms. With PIE, most of the drops are induced by the AQM algorithm and not by buffer overflow and, when the RTT is 500 ms, the queuing delay raises up to more than 90 ms. With MADPIE, when the RTT is 100 ms, the random drop part of MADPIE is responsible for more than 80% of the drops, whereas when the RTT is 500 ms, the deterministic part of MADPIE is responsible for around 70% of the drops with a consequent queuing delay reduction.

#### IV. PERFORMANCE OF MADPIE WITH A TRAFFIC MIX

We compare now the performance of DropTail (DT), PIE and MADPIE when the traffic comes from a mix of various applications.

##### A. Traffic and topology

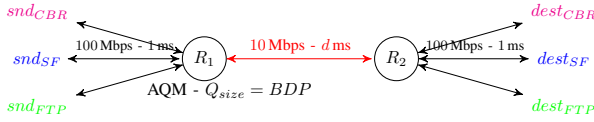


Fig. 5. Topology and traffic mix used to evaluate MADPIE.

Fig. 5 presents the topology used in this section. The one-way delay  $d$  of the bottleneck link is set to 48 ms, 148 ms or 248 ms (*i.e.* a base RTT of 100 ms, 300 ms or 500 ms). The queue size at  $R_1$  is set to the BDP. The AQM introduced at  $R_1$  is either PIE ( $\tau = 20$  ms and  $\lambda = 30$  ms) or MADPIE ( $\tau = 20$  ms,  $\lambda = 30$  ms,  $\tau_{DD} = 25$  ms).

Between  $snd_{CBR}$  and  $dest_{CBR}$ , there are  $N_{CBR}$  Constant Bit-Rate (CBR) UDP flows with a sending rate of 87 kbps and a packet size of 218 B. The intent is to model Voice-over-IP (VoIP) or gaming traffic, such as in [10, p. 17]. Between  $snd_{SF}$  and  $dest_{SF}$ ,  $N_{SF}$  flows transfer files of  $S$  kB ( $S \in \{15; 44; 73; 102\}$ ). When a download is finished, a new random value is taken for  $S$  and another download starts after  $\tau$  seconds, with  $\tau$  randomly generated according to an exponential law of mean 9.5 s. This traffic lets us assess the benefits of using MADPIE for short flows. Between  $snd_{FTP}$  and  $dest_{FTP}$ ,  $N_{FTP}$  TCP bulk flows are generated. TCP flows use CUBIC congestion control, and TCP options are the same as those specified in § III. All the flows randomly start between 0 and 1 s. Each run lasts 100 s and is repeated 20 times with independent seeds. The metrics are sampled every second (except for the queuing delay and the one way delay that are sampled per-packet). We choose to present the results with  $N_{CBR} = 4$ ,  $N_{SF} = 20$  and  $N_{FTP} = 10$ , as this traffic mix stresses both PIE and MADPIE.

##### B. CBR traffic

The performance for CBR traffic is shown in Fig. 6. Fig. 6a explains how to interpret Fig. 6b, 6c and 6d. We present the average cumulative goodput as a function of the queuing delay, as advised in [11].

The results with DT, shown in Fig. 6b, illustrate that queuing delay can be very high, impacting latency-sensitive applications (the higher percentiles for the queuing delay when the RTT is 500 ms are not shown as they do not fit in the current scale). The goodput may sometimes be over 87 kbps as delayed packets at the bottleneck queue may arrive in bursts at the receiver. The comparison of the results of PIE (in Fig. 6c) and MADPIE (in Fig. 6d) confirms that when the RTT is 100 ms, MADPIE does not differ much from PIE. When the RTT increases, the deterministic drops induced by MADPIE allow a reduction in the experienced queuing delay of 5 ms for the 75<sup>th</sup> percentile and of 30 ms for the 95<sup>th</sup> percentile, without noticeable impact on the goodput.

##### C. Small-file downloads

We represent in Fig. 7 the download time of files of various sizes, with and without AQM schemes; the boxplots show the 5<sup>th</sup>, 25<sup>th</sup>, 75<sup>th</sup> and 95<sup>th</sup> percentiles; the line in the middle of the box is the median. With DT, the download time is higher than with any of the two AQMs for every file size and RTT considered. Comparison of the results with MADPIE and PIE shows that MADPIE reduces worst-case transmission times. For example, with MADPIE as compared with PIE, (1) the 95<sup>th</sup> percentile of the download time for 73 Kb is reduced by  $\approx 700$  ms when  $RTT = 300$  ms; (2) the 75<sup>th</sup> percentile of the download time for 102 Kb is reduced by  $\approx 500$  ms when  $RTT = 500$  ms. This can be explained by the fact that, with MADPIE, the few packets that compose a short file transfer have a lower probability of experiencing high queuing delays, and of arriving at the queue when the random-drop probability is high (hence suffering losses in a burst).

##### D. Bulk flows

Fig. 8 shows the CDF of the goodput for the bulk flows. With DT, the impact of the RTT can hardly be noticed. Without AQM at  $R_1$ , the bottleneck utilization is higher than with any of PIE or MADPIE. With the latter, when the RTT is large the gain for latency sensitive applications comes at the expense of a small degradation in goodput for bulk flows.

#### V. CONCLUSION

In this paper, we have proposed MADPIE, a simple change to the PIE algorithm that makes it less dependent on path RTTs in lightly-multiplexed scenarios. MADPIE extends PIE by adding, on top of the random drops, a deterministic drop policy loosely based on CoDel's. The proportion of deterministic drops increases when the RTT increases. MADPIE can both keep the same target delay as PIE and reduce the maximum queuing delay, making the goodput of bulk flows close to the one achieved with PIE, guaranteeing lower queuing delay for VoIP-like traffic and reducing the download time of small files.

We do not claim that our proposal is the only, or best, way of tuning or adapting PIE. However, it is a very simple addition to PIE's code (a handful of lines, in our ns-2 implementation) that can complement specific parameter tunings, and its impact on the performance of PIE seems negligible when RTTs are not large (*i.e.* outside the operating conditions for which it has been conceived).

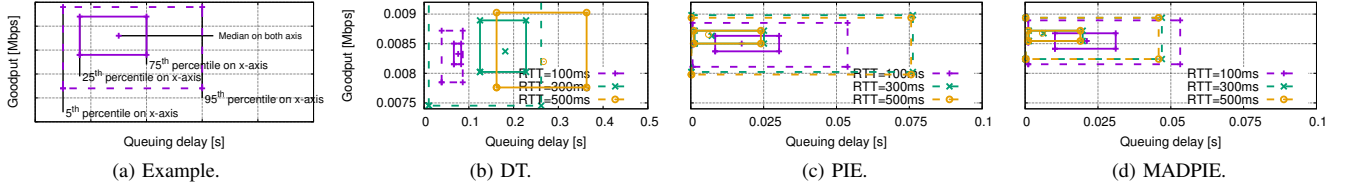


Fig. 6. Average goodput and queuing delay for the CBR traffic.

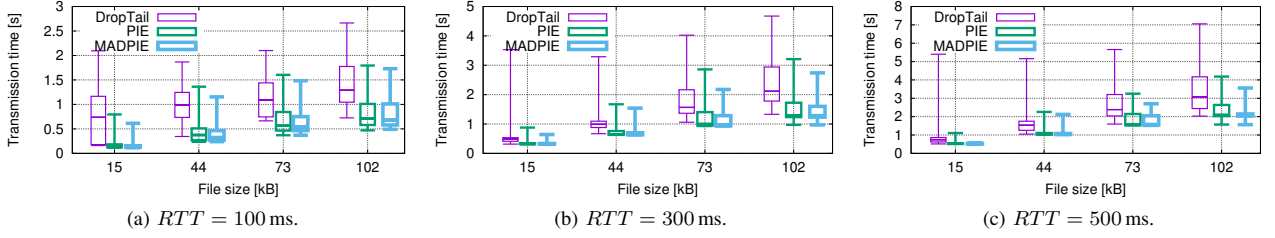


Fig. 7. Small file download time.

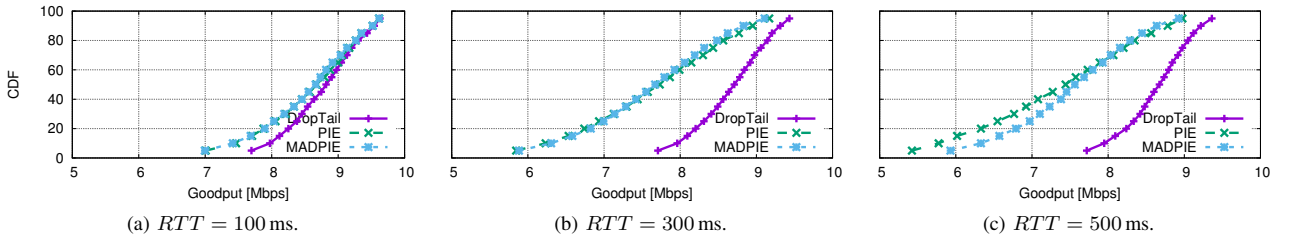


Fig. 8. Goodput of the bulk flows.

## REFERENCES

- [1] J. Gettys, "Bufferbloat: Dark Buffers in the Internet," *IEEE Internet Computing*, vol. 15, no. 3, pp. 96–96, May 2011.
- [2] F. Baker and G. Fairhurst, "IETF Recommendations Regarding Active Queue Management," Best Current Practice RFC 7567, IETF, Jul. 2015. [Online]. Available: <http://tools.ietf.org/html/rfc7567>
- [3] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.
- [4] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An algorithm for increasing the robustness of RED's active queue management," ICIR, Technical report, Aug. 2001. [Online]. Available: <http://www.icir.org/floyd/red.html>
- [5] R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, and B. Versteeg, "PIE: A lightweight control scheme to address the Bufferbloat problem," in *Proceedings of IEEE HPSR*, Taipei, Jul. 2013.
- [6] K. Nichols and V. Jacobson, "Controlling Queue Delay," *ACM Queue*, vol. 10, no. 5, pp. 20:20–20:34, May 2012.
- [7] I. Järvinen and M. Kojo, "Evaluating CoDel, PIE, and HRED AQM techniques with load transients," in *Proceedings of IEEE LCN*, Edmonton, Sep. 2014, pp. 159–167.
- [8] C. Kulatunga, N. Kuhn, G. Fairhurst, and D. Ros, "Tackling Bufferbloat in Capacity-limited Networks," in *24th European Conference on Networks and Communications (EuCNC)*, Paris, Jun. 2015.
- [9] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled delay active queue management," Internet Draft draft-ietf-aqm-code1, work in progress, Apr. 2015. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-aqm-code1>
- [10] G. White and D. Rice, "Active Queue Management Algorithms for DOCSIS 3.0," Cable Television Laboratories, Tech. Rep., Apr. 2013. [Online]. Available: [http://www.cablelabs.com/wp-content/uploads/2013/11/Active\\_Queue\\_Management\\_Algorithms\\_DOCSIS\\_3\\_0.pdf](http://www.cablelabs.com/wp-content/uploads/2013/11/Active_Queue_Management_Algorithms_DOCSIS_3_0.pdf)
- [11] N. Kuhn, P. Natarajan, N. Khademi, and D. Ros, "AQM characterization guidelines," Internet Draft draft-ietf-aqm-eval-guidelines, work in progress, Jul. 2015. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-aqm-eval-guidelines>

# Identifying Bottleneck Drop Strategies: an Active Measurement Approach

**Abstract**—The type and details of queue management mechanisms that are deployed at the congestion points of networks are not always publicly known. Knowing the kinds of a bottleneck drop scheme is crucial to understanding the performance of the network and choosing how to develop the right end-host mechanisms. The best-known queue scheme is tail-drop FIFO that only drop packets when its buffer is full. Active queue management schemes (AQM) start throwing packets out of their buffer as soon as they realize their buffer is growing large quickly. In this paper, we present an end-to-end measurement method to detect the kind of queue scheme on a bottleneck router. We have developed an active measurement tool and evaluated our measurement methodology on an experimental test-bed. The results of our experiments show that the proposed approach provides the basis for building a tool to identify some AQMs deployed on the bottleneck automatically.

**Index Terms**—Queue scheme, active queue management, tail-drop FIFO

## I. INTRODUCTION

The number of residential broadband subscribers are constantly increasing. In the first half of the year 2013, fixed broadband take-up (lines as a percentage of population) increased by 0.7% in EU countries, which corresponds to a household penetration of around 76% [1]. According to [2], worldwide fixed broadband subscribers are expected to grow from 683 million at the end of 2013 to 837 million by the end of 2019.

Significant increase in broadband penetration has caught researchers' interest in evaluating the performance of such networks. There are a lot of performance measurement tools available on the Internet [3], [4]. They most commonly report upload and download speeds, but also other performance and security issues to users. In addition, there have been numerous efforts to measure the performance of different Internet service providers (ISPs) around the world [5]–[9]. The majority of previous network performance measurement studies have performed active or passive measurements to evaluate common Quality of Service (QoS) metrics such as average download and upload rates, queue delay, buffer length, and so on.

Although these previous studies were successful in shedding some light on different aspects of access networks performance, they mostly ignored to consider the type of queueing scheme being employed in the ISP's routers. A router queue management scheme has a direct impact on the service provided to the user during congestion. As a result, to understand the performance of the network we

need to figure out what kinds of drop scheme are employed in the congested routers. Knowing the bottleneck queue scheme can help us to choose the right end-host mechanisms, whether the goal is low latency, high throughput or something else.

Tail-drop FIFO is the classic queue drop policy in routers, which only packets arriving after the queue is full are discarded. Active queue management policies (AQM), however, start dropping packets before their buffer gets full [10]–[12] based on various strategies. Although AQMs have been extensively studied [13]–[16], relatively little is known about the extent to which they are deployed in practice.

In this paper, we propose an approach which enables us to detect the type of queue used in a bottleneck router. The basic idea of our approach is to figure out whether the bottleneck drops packets before its buffer is full or if all the packet losses happen after the buffer becomes full. We developed a preliminary active measurement tool and tested it on an experimental test-bed. The results of our experiments show that the proposed approach provides the basis for identifying some AQMs deployed on the bottleneck automatically.

The remainder of this paper is organized as follows. First we briefly describe the measurement methodology in section II. Then, we present our experimental results in section III. Finally, section IV concludes the paper and discusses potential future works.

## II. QUEUE DETECTION APPROACH: AQM OR TAIL-DROP

The path between a *Sender* and a *Receiver* consists of a sequence of *network routers* that forward packets to the next hop. When sending a flow of data, one such router along the path has the least available bandwidth for the flow; it is considered the *bottleneck* of this path. If the *Sender* sends packets quicker than the bottleneck router of the path can handle, the packets begin to fill up the bottleneck buffer, and the bottleneck router must eventually drop some of the packets.

Consider the tail-drop FIFO case; Tail-drop FIFO establishes a single queue for each outgoing link, and forward packets on that link in the order in which they arrive. Packets are dropped only when the queue is full. On the other hand, AQM policies proactively drop packets when the queue starts to fill up before reaching the state when the queue is completely full [10]–[12]. As a result, a simple way to distinguish them is to look for packet loss occurring before the queue is full.



Suppose the *Sender* transmits a *periodic packet stream* to the *Receiver*. The stream consists of  $n$  packets, *Maximum* sized (MSS), with  $t$  seconds inter-transmission time (ITT). The *Sender* adds a timestamp  $t_k$  and sequence number to the payload of each packet  $k$  ( $1 \leq k \leq n$ ) prior to its transmission. Having the transmission time  $t_k$  and the arrival time  $a_k$ , the *Receiver* computes the *queueing delay* of all received packets. To do so, first the *Receiver* calculates the one-way delay (OWD) for all delivered packets as  $D_k = a_k - t_k$ ; the *queueing delay* of packet  $k$  would be  $Q_k = D_k - D_m$ , where  $D_m$  is the minimum OWD. Looking at the packets' *queueing delay* in combination with loss occurrences, the *Receiver* can infer at which point the queue saturated, and whether any loss happened before that point or not. We here show how the measured queueing delay and loss patterns can provide information about the bottleneck router's queue management type.

If the stream rate is larger than the available bandwidth of the bottleneck router, the queue of the router gradually builds up. Therefore, packet  $k$  will wait in the queue for a longer time interval than packet  $k-1$ . Consequently, the packets' *queueing delay*  $\{Q_1, \dots, Q_{k-1}, Q_k, \dots, Q_n\}$  has an increasing trend while the queue is being filled up. If  $n$  is large enough, at some point, the queue becomes full and some of the packets get dropped.

Normally, the rate at which packets enter the queue becomes equal to the rate at which packets leave the queue, when the queue is saturated. Therefore, the packets that the bottleneck router receives after its queue is filled up experiences approximately the same *queueing delay*. If there is any packet loss while the *queue delay* is growing, before the *queueing delays* get approximately constant, the *Receiver* can infer that the bottleneck router is employing AQM as its queue management.

### III. EXPERIMENTS

To test the proposed method, we developed an *active measurement tool*. The tool is composed of a *Sender* process running at the client and a *Receiver* process running at the server. The tool uses UDP for the periodic packet stream. There is also a TCP connection between the two end-points which is used to transfer control messages such as the stream specifications, the abortion or the end of measurement process, etc.

Figure 1 shows the network topology employed to test the proposed approach. The *Sender* and *Receiver* machines are connected through two Linux machines: one *bottleneck router* and one emulating network delay using *Netem* [17]. The links between the machines are all 1Gbps, except for the link between the *Router* and the *Netem* machine, where we have used *ethtool* [18] to reduce the capacity to 10Mbps. The Linux *router* uses *tc* to shape different bottleneck AQMs and *tail-drop* queue schemes into the *router*. We tested all the queue schemes with their default parameters which are listed in Table I.

We first tested the proposed approach where there is no background traffic. The results are shown in Figures 2a

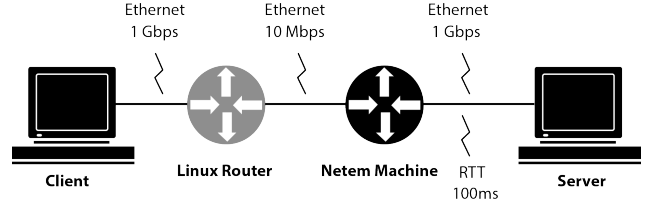


Fig. 1: The Network Topology

TABLE I: Different queue parameters

Queue Type	Parameters
Packet-based	limit 1000p
Byte-based	limit 1514000b
ARED	limit 1514000b $th_{min}125Kb$ $th_{max}375Kb$
CoDel	limit 1000p target 5.0ms interval 100.0ms
PIE	limit 1000p target 5.0ms interval 30.0ms

and 3a. Figure 2a illustrates the *queue delay* variations of four periodic streams from the client to the server. All four streams have  $n = 3000$  packets and  $t = 500\mu s$  ITT (MSS size). *CoDel* [11], *PIE* [12], and *ARED* [10] will, as expected, drop some packets before their buffer is fully saturated. Tail-drop queues, (*packet-based* and *byte-based*), do not show losses until after the queue is full.

Looking at the queue delay curves along with the packet losses (Figure 2a), the *Receiver* can differentiate between AQM and tail-drop. However, to figure out which kind of AQM is being employed in the bottleneck, we need to look closer at the drop policies' *loss patterns*. We used a Gaussian kernel density estimator [19] to show different *loss distributions* produced by the five drop policies (Figure 3a). To do that, we consider each sequence number  $k$  as a data point ( $1 \leq k \leq n$ ). We then compute the *loss density* for each data point  $L_k$  using (1):

$$L_k = \sum_{\text{Over } j} \frac{1}{\sqrt{2\pi}} e^{-\frac{|k-j|}{2}} \quad (1)$$

Where  $|k-j|$  is the distance from data point  $k$  to the packet loss with sequence number  $j$ . Figure 3a illustrates *loss density* curves for different drop policies. All AQMs gradually drop packets, while tail-drops show zero loss rate at first and a significant increase in loss occurrences when there is no more space in their buffer. Once *CoDel* starts dropping packets, it constantly reduces the time between two packet drops because the stream does not slow down and the queue delay stays above the threshold. This effect is visible in the *CoDel* loss density curve. The *ARED* loss density grows notably after receiving 500 packets. Then the loss rate decreases and stays almost constant for the rest of the test period. Compared to *CoDel* and *ARED*, *PIE* shows more variable loss density. This could be explained by the fact that *PIE* drops packets at enqueue using a probability, which is calculated periodically based on queue delay.

Ideally we would prefer to apply the *queue detection tool* when there is no other traffic; without any competing

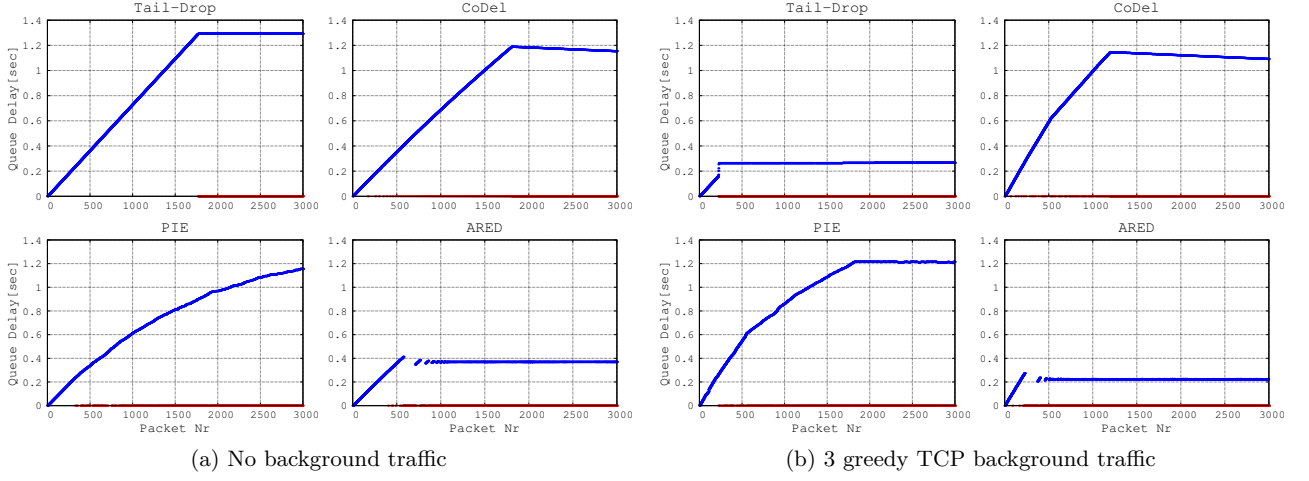


Fig. 2: Queue delay for different queue schemes - the red dots are the packets that got lost

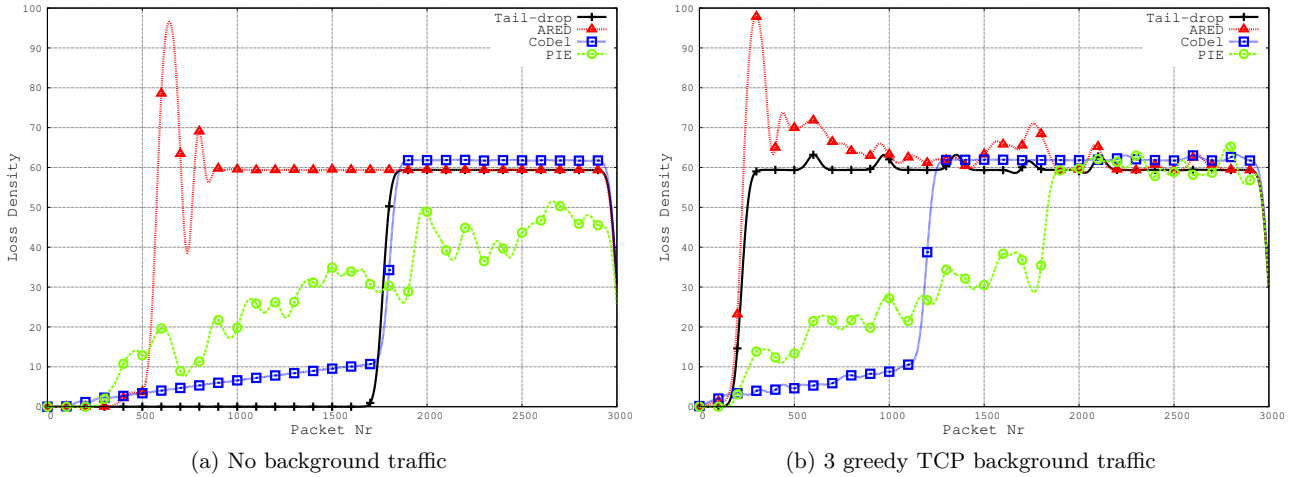


Fig. 3: Loss Density for different queue schemes

traffic, detecting varying tail-drops and AQMs would be easier. However, situations may occur where a probe may be needed even in the presence of other traffic. Figures 2b and 3b show that the results are distinguishable also in the presence of three greedy TCP flows. Differentiating between different types of AQMs might however become difficult because of the noise introduced by the background traffic. What happens here is that the measurement traffic forces competing TCPs to withdraw by creating a huge number of losses. A question arising is why *queue delays* still start from *zero* while there are other streams in the queue. The reason is that the *Receiver* uses minimum *OWD* ( $D_m$ ) to calculate each packet's *queue delay*. Consequently, the fastest packet (the packet with the minimum *OWD*) will have *zero* queue delay in both scenarios.

Historically, router performance has been measured in terms of either packet throughput or byte throughput. As a result, routers have been built with either a flat buffer that implements byte-based queueing, or buffers

that are structured as maximum-sized cells that can hold one packet each and thus, implement packet-based queueing. Both of these architectures (and probably more) are deployed in the Internet, and we know from other work [20] that they affect different kinds of traffic differently.

The idea behind separating packet-based from byte-based queues is simple. Byte-based queues will allow small packets to fit in the end of a nearly-full queue, giving small packets an advantage. Packet-based queues, however, will treat small and large packets in the same way. Therefore, we suggest that by observing the loss rate behaviour of a queue exposed to different packet sizes, we can deduce whether the queue is byte-based or packet-based. We further performed experiments about this phenomenon and the results are discussed as follows. Figure 4 depicts the loss rate behaviour of a *packet-based* versus *byte-based* tail-drop queue when the packet size changes from very small packets (96bytes) to MSS sized packets. For each packet size, the *Sender* sends 3000 *back-to-back* packets to the

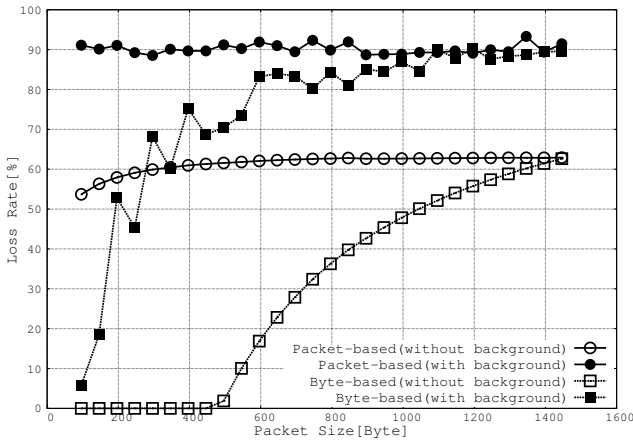


Fig. 4: Packet-based VS. Byte-based

*Receiver.* The loss rate of a *packet-based* tail-drop queue does not depend on the packet-size, while a *byte-based* queue will drop more packets as the packet-size increases. Even when there are three greedy TCP background flows, the difference between *packet-based* and *byte-based* tail-drop is still clearly observable.

#### IV. CONCLUSION AND FUTURE WORK

In this paper, we presented an approach that can detect the type of queue management scheme used in a bottleneck router. We evaluated our proposed approach using an active measurement tool in a controlled test-bed. The initial results show that using matching of the loss-density and queue delay curves against expected patterns, we might be able to build a tool for automatically identifying the kind of queue scheme deployed on the bottleneck.

In our experiments, we assumed that the edge network is the bottleneck. However, in reality, the network might be congested somewhere else. Currently, the tool is not able to detect whether the congested bottleneck is the home gateway or if it is caused by another router in the path. In future work, we might try to combine something similar to **traceroute** with the tool to improve on this situation.

Additionally, we intend to improve the tool to enable the *Sender* to automatically find a suitable sending rate so that we don't get limited by local queues. Furthermore, we have plans to investigate the possible effects of having *serial bottlenecks*. Sending a very high rate periodic packet stream (too low ITT) makes the closest drop source to overshadow the actual bottleneck's results.

#### ACKNOWLEDGEMENTS

This work is funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700) and the TimeIn project (213265-O70) by Research Council of Norway.

#### REFERENCES

- [1] D. A. F. EUROPE. Broadband access in the eu. [Online]. Available: [http://ec.europa.eu/information\\_society/newsroom/cf/dae/document.cfm?doc\\_id=4590](http://ec.europa.eu/information_society/newsroom/cf/dae/document.cfm?doc_id=4590)
- [2] L. Windsor Oaks Group. Market outlook report. [Online]. Available: [http://broadbandtrends.com/yahoo\\_site\\_admin/assets/docs/BBT\\_GlobalBBOutlook2014\\_141080\\_TOC.89125746.pdf](http://broadbandtrends.com/yahoo_site_admin/assets/docs/BBT_GlobalBBOutlook2014_141080_TOC.89125746.pdf)
- [3] M-lab's Internet Measurement Tools. [Online]. Available: <http://www.measurementlab.net/tests>
- [4] Netalyzr. [Online]. Available: <http://netalyzr.icsi.berkeley.edu/>
- [5] W. de Donato, A. Botta, and A. Pescapé, "Hobbit: A platform for monitoring broadband performance from the user network," in *Traffic Monitoring and Analysis*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, vol. 8406, pp. 65–77.
- [6] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu, "Characterizing Residential Broadband Networks," in *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, 2007.
- [7] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: Illuminating the edge network," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10, 2010.
- [8] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapé, "Broadband internet performance: A view from the gateway," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011.
- [9] S. B. Srikanth Sundaresan and N. Feamster. BISmark: A Testbed for Deploying Measurements and Applications in Broadband Access Networks. [Online]. Available: <https://www.usenix.org/system/files/conference/atc14/atc14-paper-sundaresan.pdf>
- [10] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive red: An algorithm for increasing the robustness of red's active queue management," Test, Tech. Rep., 2001.
- [11] K. Nichols and V. Jacobson, "Controlling queue delay," *Queue*, vol. 10, no. 5, May 2012.
- [12] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "Pie: A lightweight control scheme to address the bufferbloat problem," in *HPSR*. IEEE, 2013.
- [13] E. Grigorescu, C. Kulatunga, and G. Fairhurst, "Evaluation of the impact of packet drops due to AQM over capacity limited paths," in *21st IEEE International Conference on Network Protocols, ICNP*, 2013.
- [14] N. Khademi, D. Ros, and M. Welzl, "The new aqm kids on the block: An experimental evaluation of codel and pie," in *Computer Communications Workshops (INFOCOM WKSHPS)*, 2014 IEEE Conference on, April 2014.
- [15] N. Kuhn, E. Lochin, and O. Mehani, "Revisiting old friends: Is codel really achieving what red cannot?" in *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, ser. CSWS '14, 2014.
- [16] G. White and D. Rice, "Active Queue Management Algorithms for DOCSIS 3.0," in *Cable Television Laboratories, Inc.*, 2013.
- [17] "Netem - Linux network emulator," <http://www.linuxfoundation.org/collaborate/workgroups/networking/netem>, accessed: 2015-03-25.
- [18] "ethtool - utility for controlling network drivers and hardware," <https://www.kernel.org/pub/software/network/ethtool/>, accessed: 2015-03-25.
- [19] Kernel Density Estimators. [Online]. Available: [http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL\\_COPIES/AV0405/MISHRA/kde.html](http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0405/MISHRA/kde.html)
- [20] L. Stewart, D. Hayes, G. Armitage, M. Welzl, and A. Petlund, "Multimedia-unfriendly TCP congestion control and home gateway queue management," in *Proc. of the ACM Multimedia Systems Conference (MMSys)*, San Jose, California, Feb. 2011, pp. 35–44. [Online]. Available: <http://dx.doi.org/10.1145/1943552.1943558>



## B Network Signals Publications

In this appendix the publications of the Network Signals activity are included:

- G. Fairhurst and M. Welzl, “The benefits of using Explicit Congestion Notification (ECN),” Internet draft, draft-ietf-aqm-ecn-benefits-06 (work in progress), Jul. 2015. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-aqm-ecn-benefits>

**Abstract:** The goal of this document is to describe the potential benefits when applications use a transport that enables Explicit Congestion Notification (ECN). The document outlines the principal gains in terms of increased throughput, reduced delay and other benefits when ECN is used over a network path that includes equipment that supports ECN-marking. It also discusses challenges for successful deployment of ECN. It does not propose new algorithms to use ECN, nor does it describe the details of implementation of ECN in endpoint devices (Internet hosts), routers or other network devices.

- D. Black (IETF tsvwg co chair), “Explicit Congestion Notification (ECN) and IEEE Protocols,” Available online, URL: <https://datatracker.ietf.org/liaison/1364/>, Nov. 2014. [Online]. Available: <https://datatracker.ietf.org/liaison/1364/>

**Abstract:** This liaison statement is addressed at IEEE 802, and particularly though not exclusively the 802.1 WG for congestion marking in bridges and other WGs for other forms of forwarding. The IETF wishes to notify relevant WGs that it has started work on guidelines for adding ECN to protocols that may encapsulate IP and interfacing these protocols with ECN in IP. Then IP may act in its role as an interoperability protocol over multiple forwarding protocols. This activity is led by the IETF’s transport services working group (tsvwg).

==Requests==

The IETF tsvwg kindly asks IEEE 802:

1. To inform the IETF tsvwg of which IEEE work-groups could be affected by this work.
2. To inform the IETF tsvwg of any specific IEEE specifications affected by this work.
3. to forward this liaison statement to these affected work-groups, and to invite them to review the latest draft of the guidelines, available here: - <http://tools.ietf.org/html/draft-ietf-tsvwg-ecn-encap-guidelines>

- D. Black (IETF tsvwg co chair), “Explicit Congestion Notification for Lower Layer Protocols,” Available online, URL: <https://datatracker.ietf.org/liaison/1424/>, Jul. 2015. [Online]. Available: <https://datatracker.ietf.org/liaison/1424/>

**Abstract:** This liaison statement is to inform 3GPP, in particular those groups including those involved in 3GPP Release-10 work on the work item ECSRA\_LA (TR23.860) - SA4, CT4, SA2 and RAN2. Please distribute to all groups that have used or plan to use IETF ECN / AQM RFCs in 3GPP specifications.

The IETF has started work on guidelines for adding ECN to protocols that may encapsulate IP and interfacing these protocols with ECN in IP. Then IP may act in its role as an interoperability protocol over multiple forwarding protocols. This activity is led by the IETF’s transport services working group (tsvwg).

Actions:

The IETF tsvwg kindly asks 3GPP:

1. to tell the IETF tsvwg which 3GPP working groups could be affected by this work.
2. To inform the IETF tsvwg of any specific 3GPP specifications affected by this work.
3. to forward this liaison statement to these affected working groups, and to invite them to review the latest draft of the guidelines, available here: <<http://tools.ietf.org/html/draft-ietf-tsvwg-ecn-encap-guidelines>>

- B. Briscoe, J. Kaippallimalil, and P. Thaler, “Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP,” Internet Engineering Task Force, Internet Draft draft-ietf-tsvwg-ecn-encap-guidelines-02, Mar. 2015, (Work in Progress). [Online]. Available: <http://tools.ietf.org/html/draft-ietf-tsvwg-ecn-encap-guidelines>

**Abstract:** The purpose of this document is to guide the design of congestion notification in any lower layer or tunnelling protocol that encapsulates IP. The aim is for explicit congestion signals to propagate consistently from lower layer protocols into IP. Then the IP internetwork layer can act as a portability layer to carry congestion

notification from non-IP-aware congested nodes up to the transport layer (L4). Following these guidelines should assure interworking between new lower layer congestion notification mechanisms, whether specified by the IETF or other standards bodies.

- N. Khademi, M. Welzl, G. Armitage, and G. Fairhurst, “TCP Alternative Backoff with ECN (ABE),” Internet Engineering Task Force, Internet Draft draft-khademi-alternativebackoff-ecn-00, Jun. 2015, (Work in Progress). [Online]. Available: <https://tools.ietf.org/html/draft-khademi-alternativebackoff-ecn-00>

**Abstract:** This memo updates the TCP sender-side reaction to a congestion notification received via Explicit Congestion Notification (ECN). The updated method is less conservative than the TCP reaction in response to loss. The intention is to achieve good throughput when the queue at the bottleneck is smaller than the bandwidth-delay product of the connection. This is more likely when an Active Queue Management (AQM) mechanism has ECN-marked a packet than when a packet was lost. Future versions of this document will discuss SCTP as well as other transports using ECN.

- N. Khademi, M. Welzl, G. Armitage, C. Kulatunga, D. Ros, G. Fairhurst, S. Gjessing, and S. Zander, “Alternative Backoff: Achieving Low Latency and High Throughput with ECN and AQM,” Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150710A, 10 July 2015. [Online]. Available: <http://caia.swin.edu.au/reports/150710A/CAIA-TR-150710A.pdf>

**Abstract:** CoDel and PIE are recently proposed Active Queue Management (AQM) mechanisms that minimize the time packets spend enqueued at a bottleneck, instantiating shallow, 5 ms to 20 ms buffers with short-term packet burst tolerance. However, shallow buffering causes noticeable TCP performance degradation when a path’s underlying round trip time (RTT) heads above 60 ms to 80 ms (not uncommon with cross-continental and inter-continental traffic). Using less-aggressive multiplicative backoffs is known to compensate for shallow bottleneck buffering. We propose ABE: “Alternative Backoff with ECN”, which consists of enabling Explicit Congestion Notification (ECN) and letting individual TCP senders use a larger multiplicative decrease factor in reaction to ECN-marks from AQM-enabled bottlenecks. Using a mix of experiments, theory and simulations with standard NewReno and CUBIC flows, we show significant performance gains in lightly-multiplexed scenarios, without losing the delay-reduction benefits of deploying CoDel or PIE. ABE is a sender-side-only modification that can be deployed incrementally (requiring no flag-day) and offers a compelling reason to deploy and enable ECN across the Internet.

- K. De Schepper, O. Bondarenko, I.-J. Tsang, and B. Briscoe, “‘Data Centre to the Home’: Ultra-Low Latency for All,” in *Under submission*, Jul. 2015

**Abstract:** Data Centre TCP (DCTCP) was designed to provide predictably low queuing latency, near-zero loss, and throughput scalability using explicit congestion notification (ECN) and an extremely simple marking behaviour on switches. However, DCTCP does not co-exist with existing TCP traffic—throughput starves. So, until now, DCTCP could only be deployed where a clean-slate environment could be arranged, such as in private data centres. This paper proposes ‘Coupled Active Queue Management (AQM)’ to allow scalable congestion controls like DCTCP to safely co-exist with classic Internet traffic. In extensive tests within the edge gateway of a realistic broadband access testbed, the Coupled AQM ensures that a flow runs at about the same rate whether it uses DCTCP or TCP Reno/Cubic, but without inspecting transport layer flow identifiers. DCTCP achieves sub-millisecond average queuing delay and zero congestion loss under a wide range of mixes of DCTCP and ‘classic’ broadband Internet traffic, without compromising the performance of the classic traffic. The solution also reduces network complexity and eliminates network configuration.

- K. De Schepper, B. Briscoe, O. Bondarenko, and I.-J. Tsang, “DualQ Coupled AQM for Low Latency, Low Loss and Scalable Throughput,” Internet Engineering Task Force, Internet Draft draft-briscoe-aqm-dualq-coupled-00, Aug. 2015, (Work in Progress). [Online]. Available: <http://datatracker.ietf.org/doc/draft-briscoe-aqm-dualq-coupled-00>

**Abstract:** Data Centre TCP (DCTCP) was designed to provide predictably low queuing latency, near-zero loss, and throughput scalability using explicit congestion notification (ECN) and an extremely simple marking behaviour on switches. However, DCTCP does not co-exist with existing TCP traffic—throughput starves. So, until now, DCTCP could only be deployed where a clean-slate environment could be arranged, such as in private data centres. This specification defines ‘DualQ Coupled Active Queue Management (AQM)’ to allow scalable congestion controls like DCTCP to safely co-exist with classic Internet traffic. The Coupled AQM ensures that a flow runs at about the same rate whether it uses DCTCP or TCP Reno/Cubic, but without

inspecting transport layer flow identifiers. When tested in a residential broadband setting, DCTCP achieved sub-millisecond average queuing delay and zero congestion loss under a wide range of mixes of DCTCP and 'Classic' broadband Internet traffic, without compromising the performance of the Classic traffic. The solution also reduces network complexity and eliminates network configuration.

## Liaison statement

### Explicit Congestion Notification (ECN) and IEEE Protocols

<b>Submission date</b>	2014-11-24
<b>From</b>	Transport Area Working Group (David Black (mailto:david.black@emc.com))
<b>To</b>	IEEE 802 (Paul Nikolich - IEEE 802 Chair (mailto:p.nikolich@ieee.org))
<b>Cc</b>	Gorry Fairhurst (mailto:gorry@erg.abdn.ac.uk), James Polk (mailto:jmpolk@cisco.com), Martin Stiernerling (mailto:mls.ietf@gmail.com), Spencer Dawkins (mailto:spencerdawkins.ietf@gmail.com), Pat Thaler (mailto:pthaler@broadcom.com), Bob Briscoe (mailto:bob.briscoe@bt.com), Glenn Parsons - IEEE 802.1 WG chair (mailto:gparsons@ieee.org), Eric Gray (mailto:eric.gray@ericsson.com), Dan Romascanu (mailto:dromasca@avaya.com), Russ Housley (mailto:housley@vigilsec.com)
<b>Response contact</b>	David Black (mailto:david.black@emc.com)
<b>Technical contact</b>	Bob Briscoe (mailto:bob.briscoe@bt.com)
<b>Purpose</b>	For comment
<b>Deadline</b>	2015-02-20 Action Needed
<b>Attachments</b>	(None)

**Body**

```

To: "IEEE 802 - attn. Paul Nikolich, Chair" <p.nikolich@ieee.org>
From: "IETF TSVWG" <tsvwg-chairs@tools.ietf.org>
CC: "Martin Stiemerling" <mls.ietf@gmail.com>, "Spencer Dawkins"
    <spencerdawkins.ietf@gmail.com>,
    "Pat Thaler" <pthaler@broadcom.com>, "Bob Briscoe" <bob.briscoe@bt.com>,
    "Glenn Parsons - IEEE 802.1 WG chair" <gparsons@ieee.org>,
    "Eric Gray" <eric.gray@ericsson.com>, "Dan Romascanu" <dromasca@avaya.com>,
    "Russ Housley" <housley@vigilsec.com>

==Background==
In 2001, the IETF introduced explicit congestion notification (ECN) to the
Internet Protocol
as a proposed standard, see RFC 3168. The purpose of ECN was to notify
congestion without
having to drop packets. The IETF originally specified ECN for cases where
buffers were IP-aware.

However, ECN is now being used in a number of environments where some of the
forwarding devices
are solely layer-2 devices, where IEEE protocols encapsulate IP. Originally
there was little
if any guidance on how to add explicit congestion notification to lower layer
protocols, nor
on the interface between lower layers and ECN in IP.

==Notification==
This liaison statement is addressed at IEEE 802, and particularly though not
exclusively the
802.1 WG for congestion marking in bridges and other WGs for other forms of
forwarding. The
IETF wishes to notify relevant WGs that it has started work on guidelines for
adding ECN to
protocols that may encapsulate IP and interfacing these protocols with ECN in
IP. Then IP may
act in its role as an interoperability protocol over multiple forwarding
protocols. This
activity is led by the IETF's transport services working group (tsvwg).

==Requests==
The IETF tsvwg kindly asks IEEE 802:
1) To inform the IETF tsvwg of which IEEE work-groups could be affected by
this work.
2) To inform the IETF tsvwg of any specific IEEE specifications affected by
this work.
3) to forward this liaison statement to these affected work-groups, and to
invite them to
    review the latest draft of the guidelines, available here:
    - http://tools.ietf.org/html/draft-ietf-tsvwg-ecn-encap-guidelines
Review comments are particularly welcome on:
    - comprehensibility for the IEEE community
    - usefulness and applicability
    - technical feasibility

Review comments may be posted directly to the IETF tsvwg mailing list
<mailto:tsvwg@ietf.org>. Postings from non-subscribers may be delayed by
moderation.
Alternatively, subscription is open to all at: <
https://www.ietf.org/mailman/listinfo/tsvwg>.

==Relevant IETF Specifications==
The following IETF specifications or drafts are particularly relevant to this
activity
(the relevance of each of them is explained in the first item below):
* draft-ietf-tsvwg-ecn-encap-guidelines
    - http://tools.ietf.org/html/draft-ietf-tsvwg-ecn-encap-guidelines
* RFC3168 updated by RFC4301, RFC6040 (ECN in resp. IP/TCP, IPsec & IP-in-IP
tunnels)
    - https://tools.ietf.org/html/rfc3168
    - https://tools.ietf.org/html/rfc4301
    - https://tools.ietf.org/html/rfc6040
* RFC6679 (ECN in RTP)
    - https://tools.ietf.org/html/rfc6679
* RFC5129 updated by RFC5462 (ECN in MPLS)
    - https://tools.ietf.org/html/rfc5129
    - https://tools.ietf.org/html/rfc5462
* RFC4774 (Specifying alternative semantics for the ECN field)
    - https://tools.ietf.org/html/rfc4774
* draft-ietf-aqm-recommendation
    - http://tools.ietf.org/html/draft-ietf-aqm-recommendation

Sincerely yours,
-- David Black (IETF tsvwg co-chair) <david.black@emc.com>

```

## Liaison statement

### Explicit Congestion Notification for Lower Layer Protocols

<b>Submission date</b>	2015-07-20
<b>From</b>	Transport Area Working Group (David Black (mailto:david.black@emc.com))
<b>To</b>	3GPP (Susanna.Kooistra@etsi.org (mailto:Susanna.Kooistra@etsi.org))
<b>Cc</b>	Gonzalo Camarillo (mailto:gonzalo.camarillo@ericsson.com), Gorrry Fairhurst (mailto:gorrry@erg.abdn.ac.uk), Martin Stiernerling (mailto:mls.ietf@gmail.com), Spencer Dawkins (mailto:spencerdawkins.ietf@gmail.com), John Kaippallimalil (mailto:John.Kaippallimalil@huawei.com), Bob Briscoe (mailto:ietf@bobbriscoe.net), Transport Area Working Group Discussion List (mailto:tsvwg@ietf.org)
<b>Response contact</b>	David Black (mailto:david.black@emc.com)
<b>Technical contact</b>	Bob Briscoe (mailto:ietf@bobbriscoe.net)
<b>Purpose</b>	For comment
<b>Deadline</b>	2015-10-30 Action Needed
<b>Attachments</b>	(None)

Liaison statement: Explicit Congestion Notification for...

<https://datatracker.ietf.org/liaison/1424/>**Body**

To: 3GPP SA, 3GPP CT, 3GPP RAN, 3GPP SA4, 3GPP SA2, 3GPP RAN2  
 From: IETF TSVWG

In 2001, the IETF introduced explicit congestion notification (ECN) to the Internet Protocol as a proposed standard [RFC3168]. The purpose of ECN was to notify congestion without having to drop packets. The IETF originally specified ECN for cases where buffers were IP-aware. However, ECN is now being used in a number of environments including codec selection and rate adaptation, where 3GPP protocols such as PDCP encapsulate IP. As active queue management (AQM) and ECN become widely deployed in 3GPP networks and interconnected IP networks, it could be incompatible with the standardized use of ECN across the end-to-end IP transport [RFC7567].

The IETF is now considering new uses of ECN for low latency [draft-welzl-ecn-benefits] that would be applicable to 5G mobile flows. However, the IETF has realized that it has given little if any guidance on how to add explicit congestion notification to lower layer protocols or interfaces between lower layers and ECN in IP.

This liaison statement is to inform 3GPP, in particular those groups including those involved in 3GPP Release-10 work on the work item ECSRA\_LA (TR23.860) - SA4, CT4, SA2 and RAN2. Please distribute to all groups that have used or plan to use IETF ECN /AQM RFCs in 3GPP specifications.

The IETF has started work on guidelines for adding ECN to protocols that may encapsulate IP and interfacing these protocols with ECN in IP. Then IP may act in its role as an interoperability protocol over multiple forwarding protocols. This activity is led by the IETF's transport services working group (tsvwg).

**Actions:**

The IETF tsvwg kindly asks 3GPP:

- 1) to tell the IETF tsvwg which 3GPP working groups could be affected by this work.
- 2) To inform the IETF tsvwg of any specific 3GPP specifications affected by this work.
- 3) to forward this liaison statement to these affected working groups, and to invite them to review the latest draft of the guidelines, available here:  
 < <http://tools.ietf.org/html/draft-ietf-tsvwg-ecn-encap-guidelines> >

Review comments are particularly welcome on:

- comprehensibility for the 3GPP community
- usefulness and applicability
- technical feasibility

Review comments may be posted directly to the IETF tsvwg mailing list <<mailto:tsvwg@ietf.org>>. Postings from non-subscribers may be delayed by moderation. Alternatively, subscription is open to all at: <  
<https://www.ietf.org/mailman/listinfo/tsvwg>>.

The following IETF specifications or drafts are particularly relevant to this activity (the relevance of each of them is explained in the first item below):

- \* draft-ietf-tsvwg-ecn-encap-guidelines
- \* RFC3168 updated by RFC4301, RFC6040 (ECN in respectively: IP/TCP, IPsec & IP-in-IP tunnels)
- \* RFC6679 (ECN in RTP)
- \* RFC5129 updated by RFC5462 (ECN in MPLS)
- \* RFC4774 (Specifying alternative semantics for the ECN field)
- \* RFC7567 (Recommendations Regarding Active Queue Management)
- \* draft-welzl-ecn-benefits (Benefits to Applications of Using ECN)

Yours,

--David L. Black (TSVWG co-chair)

ISOC (<https://www.internetsociety.org/>)

IETF Trust (<https://trustee.ietf.org/>)

RFC Editor (<https://www.rfc-editor.org/>)

IRTF

(<https://www.irtf.org/>)

IESG (<https://www.ietf.org/iesg/>)

IETF (<https://www.ietf.org/>)

IAB (<https://www.iab.org/>)

IANA & IAOC (<https://iaoc.ietf.org/>)

IETF Tools (<https://tools.ietf.org/>)

IANA (<https://www.iana.org/>)

About | [/release/about](#) | IETF Datatracker | Version 6.4.1.pl | [/release/6.4.1.pl/](#) | 2015-08-30 | Report a bug | <https://tools.ietf.org/tools/ietfdb/newticket>

Transport Area Working Group  
Internet-Draft  
Updates: [3819](#) (if approved)  
Intended status: Best Current Practice  
Expires: September 27, 2015

B. Briscoe  
BT  
J. Kaippallimalil  
Huawei  
P. Thaler  
Broadcom Corporation  
March 26, 2015

Guidelines for Adding Congestion Notification to Protocols that  
Encapsulate IP  
draft-ietf-tsvwg-ecn-encap-guidelines-02

Abstract

The purpose of this document is to guide the design of congestion notification in any lower layer or tunnelling protocol that encapsulates IP. The aim is for explicit congestion signals to propagate consistently from lower layer protocols into IP. Then the IP internetwork layer can act as a portability layer to carry congestion notification from non-IP-aware congested nodes up to the transport layer (L4). Following these guidelines should assure interworking between new lower layer congestion notification mechanisms, whether specified by the IETF or other standards bodies.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 27, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.



Internet-Draft

ECN Encapsulation Guidelines

March 2015

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4.e](#) of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Scope . . . . .	5
2. Terminology . . . . .	6
3. Guidelines in All Cases . . . . .	7
4. Modes of Operation . . . . .	7
4.1. Feed-Forward-and-Up Mode . . . . .	8
4.2. Feed-Up-and-Forward Mode . . . . .	10
4.3. Feed-Backward Mode . . . . .	10
4.4. Null Mode . . . . .	12
5. Feed-Forward-and-Up Mode: Guidelines for Adding Congestion Notification . . . . .	12
5.1. IP-in-IP Tunnels with Tightly Coupled Shim Headers . . . . .	13
5.2. Wire Protocol Design: Indication of ECN Support . . . . .	13
5.3. Encapsulation Guidelines . . . . .	15
5.4. Decapsulation Guidelines . . . . .	17
5.5. Sequences of Similar Tunnels or Subnets . . . . .	18
5.6. Reframing and Congestion Markings . . . . .	19
6. Feed-Up-and-Forward Mode: Guidelines for Adding Congestion Notification . . . . .	19
7. Feed-Backward Mode: Guidelines for Adding Congestion Notification . . . . .	21
8. IANA Considerations (to be removed by RFC Editor) . . . . .	22
9. Security Considerations . . . . .	22
10. Conclusions . . . . .	22
11. Acknowledgements . . . . .	23
12. Comments Solicited . . . . .	23
13. References . . . . .	23
13.1. Normative References . . . . .	23
13.2. Informative References . . . . .	24
<a href="#">Appendix A</a> . Outstanding Document Issues . . . . .	27
<a href="#">Appendix B</a> . Changes in This Version (to be removed by RFC Editor) . . . . .	27
Authors' Addresses . . . . .	29

## 1. Introduction

The benefits of Explicit Congestion Notification (ECN) described below can only be fully realised if support for ECN is added to the relevant subnetwork technology, as well as to IP. When a lower layer buffer drops a packet obviously it does not just drop at that layer; the packet disappears from all layers. In contrast, when a lower layer marks a packet with ECN, the marking needs to be explicitly propagated up the layers. The same is true if a buffer marks the outer header of a packet that encapsulates inner tunnelled headers. Forwarding ECN is not as straightforward as other headers because it has to be assumed ECN may be only partially deployed. If an egress at any layer is not ECN-aware, or if the ultimate receiver or sender is not ECN-aware, congestion needs to be indicated by dropping a packet, not marking it.

The purpose of this document is to guide the addition of congestion notification to any subnet technology or tunnelling protocol, so that lower layer equipment can signal congestion explicitly and it will propagate consistently into encapsulated (higher layer) headers, otherwise the signals will not reach their ultimate destination.

ECN is defined in the IP header (v4 & v6) [[RFC3168](#)] to allow a resource to notify the onset of queue build-up without having to drop packets, by explicitly marking a proportion of packets with the congestion experienced (CE) codepoint.

Given a suitable marking scheme, ECN removes nearly all congestion loss and it cuts delays for two main reasons:

- o It avoids the delay when recovering from congestion losses, which particularly benefits small flows or real-time flows, making their delivery time predictably short [[RFC2884](#)];
- o As ECN is used more widely by end-systems, it will gradually remove the need to configure a degree of delay into buffers before they start to notify congestion (the cause of bufferbloat). This is because drop involves a trade-off between sending a timely signal and trying to avoid impairment, whereas ECN is solely a signal not an impairment, so there is no harm triggering it earlier.

Some lower layer technologies (e.g. MPLS, Ethernet) are used to form subnetworks with IP-aware nodes only at the edges. These networks are often sized so that it is rare for interior queues to overflow. However, this has often be more due to the inability of the original TCP protocol to saturate the links. For many years, fixes such as window scaling [[RFC1323](#)] proved hard to deploy. But now that modern

operating systems are finally capable of saturating interior links, even the buffers of well-provisioned interior switches will need to signal episodes of queuing.

Propagation of ECN is defined for MPLS [RFC5129], and is being defined for TRILL [I-D.ietf-trill-rfc7180bis], but it remains to be defined for a number of other subnetwork technologies.

Similarly, ECN propagation is yet to be defined for many tunnelling protocols. [RFC6040] defines how ECN should be propagated for IP-in-IP [RFC2003] and IPsec [RFC4301] tunnels. However, as Section 9.3 of RFC3168 pointed out, ECN support will need to be defined for other tunnelling protocols, e.g. L2TP [RFC2661], GRE [RFC1701], [RFC2784], PPTP [RFC2637] and GTP [GTPv1], [GTPv1-U], [GTPv2-C].

Incremental deployment is the most tricky aspect when adding support for ECN. The original ECN protocol in IP [RFC3168] was carefully designed so that a congested buffer would not mark a packet (rather than drop it) unless both source and destination hosts were ECN-capable. Otherwise its congestion markings would never be detected and congestion would just deteriorate further. However, to support congestion marking below the IP layer, it is not sufficient to only check that the two end-points support ECN; correct operation also depends on the decapsulator at each subnet egress faithfully propagating congestion notifications to the higher layer. Otherwise, a legacy decapsulator might silently fail to propagate any ECN signals from the outer to the forwarded header. Then the lost signals would never be detected and again congestion would deteriorate further. The guidelines given later require protocol designers to carefully consider incremental deployment, and suggest various safe approaches for different circumstances.

Of course, the IETF does not have standards authority over every link layer protocol. So this document gives guidelines for designing propagation of congestion notification across the interface between IP and protocols that may encapsulate IP (i.e. that can be layered beneath IP). Each lower layer technology will exhibit different issues and compromises, so the IETF or the relevant standards body must be free to define the specifics of each lower layer congestion notification scheme. Nonetheless, if the guidelines are followed, congestion notification should interwork between different technologies, using IP in its role as a 'portability layer'.

Therefore, the capitalised term 'SHOULD' or 'SHOULD NOT' are often used in preference to 'MUST' or 'MUST NOT', because it is difficult to know the compromises that will be necessary in each protocol design. If a particular protocol design chooses to contradict a

Internet-Draft

ECN Encapsulation Guidelines

March 2015

'SHOULD (NOT)' given in the advice below, it MUST include a sound justification.

It has not been possible to give common guidelines for all lower layer technologies, because they do not all fit a common pattern. Instead they have been divided into a few distinct modes of operation: feed-forward-and-upward; feed-upward-and-forward; feed-backward; and null mode. These modes are described in [Section 4](#), then in the following sections separate guidelines are given for each mode.

This document updates the advice to subnetwork designers about ECN in [Section 13 of \[RFC3819\]](#).

### 1.1. Scope

This document only concerns wire protocol processing of explicit notification of congestion and makes no changes or recommendations concerning algorithms for congestion marking or for congestion response (algorithm issues should be independent of the layer the algorithm operates in).

The question of congestion notification signals with different semantics to those of ECN in IP is touched on in a couple of specific cases (e.g. QCN [[IEEE802.1Qau](#)]) and with schemes with multiple severity levels such as PCN [[RFC6660](#)]). However, no attempt is made to give guidelines about schemes with different semantics that are yet to be invented.

The semantics of congestion signals can be relative to the traffic class. Therefore correct propagation of congestion signals could depend on correct propagation of any traffic class field between the layers. In this document, correct propagation of traffic class information is assumed, while what 'correct' means and how it is achieved is covered elsewhere (e.g. [[RFC2983](#)]) and is outside the scope of the present document.

Note that these guidelines do not require the subnet wire protocol to be changed to accommodate congestion notification. Another way to add congestion notification without consuming header space in the subnet protocol might be to use a parallel control plane protocol.

This document focuses on the congestion notification interface between IP and lower layer protocols that can encapsulate IP, where the term 'IP' includes v4 or v6, unicast, multicast or anycast. However, it is likely that the guidelines will also be useful when a lower layer protocol or tunnel encapsulates itself (e.g. Ethernet MAC in MAC [[IEEE802.1Qah](#)]) or when it encapsulates other protocols.

Internet-Draft

ECN Encapsulation Guidelines

March 2015

In the feed-backward mode, propagation of congestion signals for multicast and anycast packets is out-of-scope (because it would be so complicated that it is hoped no-one would attempt such an abomination).

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [[RFC2119](#)].

Further terminology used within this document:

Protocol data unit (PDU): Information that is delivered as a unit among peer entities of a layered network consisting of protocol control information (typically a header) and possibly user data (payload) of that layer. The scope of this document includes layer 2 and layer 3 networks, where the PDU is respectively termed a frame or a packet (or a cell in ATM). PDU is a general term for any of these. This definition also includes a payload with a shim header lying somewhere between layer 2 & 3.

Transport: The end-to-end transmission control function, conventionally considered at layer-4 in the OSI reference model. Given the audience for this document will often use the word transport to mean low level bit carriage, whenever the term is used it will be qualified, e.g. 'L4 transport'.

Encapsulator: The link or tunnel endpoint function that adds an outer header to a PDU (also termed the 'link ingress', the 'subnet ingress', the 'ingress tunnel endpoint' or just the 'ingress' where the context is clear).

Decapsulator: The link or tunnel endpoint function that removes an outer header from a PDU (also termed the 'link egress', the 'subnet egress', the 'egress tunnel endpoint' or just the 'egress' where the context is clear).

Incoming header: The header of an arriving PDU before encapsulation.

Outer header: The header added to encapsulate a PDU.

Inner header: The header encapsulated by the outer header.

Outgoing header: The header forwarded by the decapsulator.

CE: Congestion Experienced [[RFC3168](#)]

Internet-Draft

ECN Encapsulation Guidelines

March 2015

ECT: ECN-Capable Transport [[RFC3168](#)]

Not-ECT: Not ECN-Capable Transport [[RFC3168](#)]

ECN-PDU: A PDU that is part of a feedback loop within which all the nodes that need to propagate explicit congestion notifications back to the Load Regulator are ECN-capable. An IP packet with a non-zero ECN field implies that the endpoints are ECN-capable, so this would be an ECN-PDU. However, ECN-PDU is intended to be a general term for a PDU at any layer, not just IP.

Not-ECN-PDU: A PDU that is part of a feedback-loop within which some nodes necessary to propagate explicit congestion notifications back to the load regulator are not ECN-capable.

Load Regulator: For each flow of PDUs, the transport function that is capable of controlling the data rate. Typically located at the data source, but in-path nodes can regulate load in some congestion control arrangements (e.g. admission control or policing nodes). Note the term "a function capable of controlling the load" deliberately includes a transport that doesn't actually control the load but ideally it ought to (e.g. a sending application without congestion control that uses UDP).

Congestion Baseline: The location of the function on the path that initialised the values of all congestion notification fields in a sequence of packets, before any are set to the congestion experienced (CE) codepoint if they experience congestion further downstream. Typically the original data source at layer-4.

### 3. Guidelines in All Cases

[RFC 3168](#) specifies that the ECN field in the IP header is intended to be marked by active queue management algorithms. Any congestion notification from an algorithm that does not conform to the recommendations in [[I-D.ietf-aqm-recommendation](#)] MUST NOT be propagated from a lower layer into the ECN field in IP.

### 4. Modes of Operation

This section sets down the different modes by which congestion information is passed between the lower layer and the higher one. It acts as a reference framework for the following sections, which give normative guidelines for designers of explicit congestion notification protocols, taking each mode in turn:

Feed-Forward-and-Up: Nodes feed forward congestion notification towards the egress within the lower layer then up and along the

layers towards the end-to-end destination at the transport layer. The following local optimisation is possible:

**Feed-Up-and-Forward:** A lower layer switch feeds-up congestion notification directly into the ECN field in the higher layer (e.g. IP) header, irrespective of whether the node is at the egress of a subnet.

**Feed-Backward:** Nodes feed back congestion signals towards the ingress of the lower layer and (optionally) attempt to control congestion within their own layer.

**Null:** Nodes cannot experience congestion at the lower layer except at ingress nodes (which are IP-aware or equivalently higher-layer-aware).

#### 4.1. Feed-Forward-and-Up Mode

Like IP and MPLS, many subnet technologies are based on self-contained protocol data units (PDUs) or frames sent unreliably. They provide no feedback channel at the subnetwork layer, instead relying on higher layers (e.g. TCP) to feed back loss signals.

In these cases, ECN may best be supported by standardising explicit notification of congestion into the lower layer protocol that carries the data forwards. It will then also be necessary to define how the egress of the lower layer subnet propagates this explicit signal into the forwarded upper layer (IP) header. It can then continue forwards until it finally reaches the destination transport (at L4). Then typically the destination will feed this congestion notification back to the source transport using an end-to-end protocol (e.g. TCP). This is the arrangement that has already been used to add ECN to IP-in-IP tunnels [RFC6040], IP-in-MPLS and MPLS-in-MPLS [RFC5129].

This mode is illustrated in Figure 1. Along the middle of the figure, layers 2, 3 & 4 of the protocol stack are shown, and one packet is shown along the bottom as it progresses across the network from source to destination, crossing two subnets connected by a router, and crossing two switches on the path across each subnet. Congestion at the output of the first switch (shown as \*) leads to a congestion marking in the L2 header (shown as C in the illustration of the packet). The chevrons show the progress of the resulting congestion indication. It is propagated from link to link across the subnet in the L2 header, then when the router removes the marked L2 header, it propagates the marking up into the L3 (IP) header. The router forwards the marked L3 header into subnet 2, and when it adds a new L2 header it copies the L3 marking into the L2 header as well,

as shown by the 'C's in both layers (assuming the technology of subnet 2 also supports explicit congestion marking).

Note that there is no implication that each 'C' marking is encoded the same; a different encoding might be used for the 'C' marking in each protocol.

Finally, for completeness, we show the L3 marking arriving at the destination, where the host transport protocol (e.g. TCP) feeds it back to the source in the L4 acknowledgement (the 'C' at L4 in the packet at the top of the diagram).

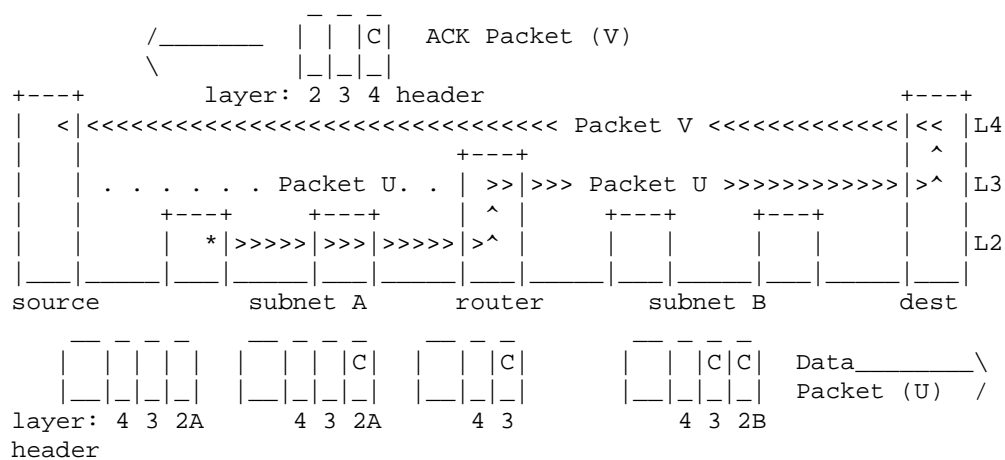


Figure 1: Feed-Forward-and-Up Mode

Of course, modern networks are rarely as simple as this text-book example, often involving multiple nested layers. For example, a 3GPP mobile network may have two IP-in-IP (GTP) tunnels in series and an MPLS backhaul between the base station and the first router. Nonetheless, the example illustrates the general idea of feeding congestion notification forward then upward whenever a header is removed at the egress of a subnet.

Note that the FECN (forward ECN) bit in Frame Relay and the explicit forward congestion indication (EFCI [ITU-T.I.371]) bit in ATM user data cells follow a feed-forward pattern. However, in ATM, this is only as part of a feed-forward-and-backward pattern, at ATM, this is never to interface to IP ECN at the subnet egress. To our knowledge, Frame Relay FECN is solely used to detect where more capacity should be provisioned [Buck00].



#### 4.2. Feed-Up-and-Forward Mode

Ethernet is particularly difficult to extend incrementally to support explicit congestion notification. One way to support ECN in such cases has been to use so called 'layer-3 switches'. These are Ethernet switches that bury into the Ethernet payload to find an IP header and manipulate or act on certain IP fields (specifically Diffserv & ECN). For instance, in Data Center TCP [DCTCP], layer-3 switches are configured to mark the ECN field of the IP header within the Ethernet payload when their output buffer becomes congested. With respect to switching, a layer-3 switch acts solely on the addresses in the Ethernet header; it doesn't use IP addresses, and it doesn't decrement the TTL field in the IP header.

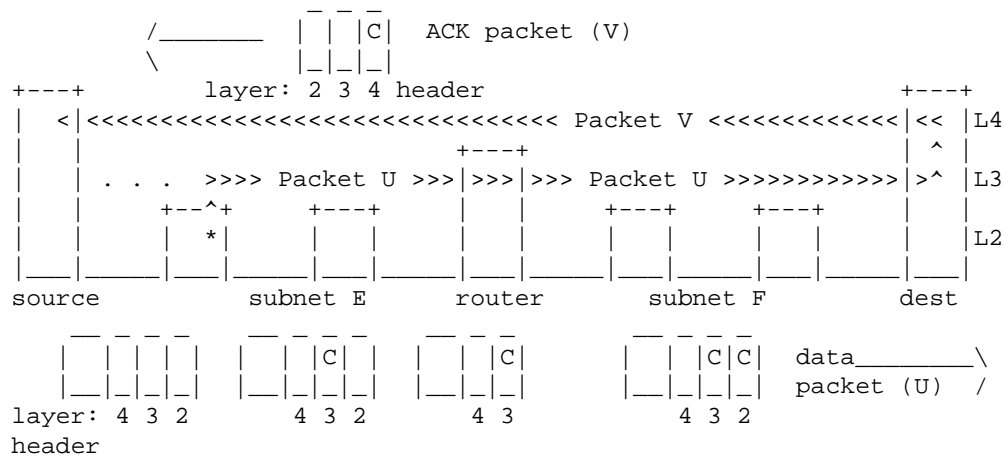


Figure 2: Feed-Up-and-Forward Mode

By comparing Figure 2 with Figure 1, it can be seen that subnet E (perhaps a subnet of layer-3 Ethernet switches) works in feed-up-and-forward mode by notifying congestion directly into L3 at the point of congestion, even though the congested switch does not otherwise act at L3. In this example, the technology in subnet F (e.g. MPLS) does support ECN natively, so when the router adds the layer-2 header it copies the ECN marking from L3 to L2 as well.

#### 4.3. Feed-Backward Mode

In some layer 2 technologies, explicit congestion notification has been defined for use internally within the subnet with its own feedback and load regulation, but typically the interface with IP for ECN has not been defined.

The diagram illustrates the flow of packets and feedback control in a network. It shows a sequence of packets (X, W, U) and their corresponding feedback control signals (V) across different layers (2, 3, 4) and network components (source, subnet G, router, subnet H, dest).

**Packet X (ACK packet):** Shown at the top, it is a feedback control signal (V) that travels from the destination back to the source. It is labeled "ACK packet (X)".

**Packet W:** Shown in the middle, it is a data packet that travels from the source through subnet G, the router, and subnet H to the destination. It is labeled "Packet W".

**Packet U:** Shown at the bottom, it is a data packet that travels from the source through subnet G, the router, and subnet H to the destination. It is labeled "Packet U".

**Feedback Control:** The diagram shows the flow of feedback control signals (V) from the destination back to the source. These signals are labeled "Feedback control cell/frame (V)".

**Layers:** The diagram is organized into layers (2, 3, 4) for each component. Layer 2 is the data layer, Layer 3 is the network layer, and Layer 4 is the transport layer. The header is shown in Layer 4.

**Network Components:** The components shown are source, subnet G, router, subnet H, and dest. Each component has a corresponding layer structure (2, 3, 4) and a header.

**Flow:** The flow of packets and feedback control signals is indicated by arrows. Packets X, W, and U travel from left to right. Feedback control signals (V) travel from right to left.

Figure 3: Feed-Backward Mode

ATM's feed-backward approach doesn't fit well when layered beneath IP's feed-forward approach--unless the initial data source is the same node as the ATM ingress. Figure 3 shows the feed-backward approach being used in subnet H. If the final switch on the path is congested (\*), it doesn't feed-forward any congestion indications on packet (U). Instead it sends a control cell (V) back to the router at the ATM ingress.

However, the backward feedback doesn't reach the original data source directly because IP doesn't support backward feedback (and subnet G is independent of subnet H). Instead, the router in the middle throttles down its sending rate but the original data sources don't reduce their rates. The resulting rate mismatch causes the middle router's buffer at layer 3 to back up until it becomes congested, which it signals forwards on later data packets at layer 3 (e.g. packet W). Note that the forward signal from the middle router is not triggered directly by the backward signal. Rather, it is triggered by congestion resulting from the middle router's mismatched rate response to the backward signal.

In response to this later forward signalling, end-to-end feedback at layer-4 finally completes the tortuous path of congestion indications back to the origin data source, as before.

#### 4.4. Null Mode

Often link and physical layer resources are 'non-blocking' by design. In these cases congestion notification may be implemented but it does not need to be deployed at the lower layer; ECN in IP would be sufficient.

A degenerate example is a point-to-point Ethernet link. Excess loading of the link merely causes the queue from the higher layer to back up, while the lower layer remains immune to congestion. Even a whole meshed subnetwork can be made immune to interior congestion by limiting ingress capacity and careful sizing of links, particularly if multi-path routing is used to ensure even worst-case patterns of load cannot congest any link.

#### 5. Feed-Forward-and-Up Mode: Guidelines for Adding Congestion Notification

Feed-forward-and-up is the mode already used for signalling ECN up the layers through MPLS into IP [[RFC5129](#)] and through IP-in-IP tunnels [[RFC6040](#)]. These RFCs take a consistent approach and the following guidelines are designed to ensure this consistency continues as ECN support is added to other protocols that encapsulate IP. The guidelines are also designed to ensure compliance with the more general best current practice for the design of alternate ECN schemes given in [[RFC4774](#)].

The rest of this section is structured as follows:

- o [Section 5.1](#) addresses the most straightforward cases, where [[RFC6040](#)] can be applied directly to add ECN to tunnels that are effectively the same as IP-in-IP tunnels.

Internet-Draft

ECN Encapsulation Guidelines

March 2015

- o The subsequent sections give guidelines for adding ECN to a subnet technology that uses feed-forward-and-up mode like IP, but it is not so similar to IP that [RFC6040] rules can be applied directly. Specifically:
  - \* Sections 5.2, 5.3 and 5.4 respectively address how to add ECN support to the wire protocol and to the encapsulators and decapsulators at the ingress and egress of the subnet.
  - \* Section 5.5 deals with the special, but common, case of sequences of tunnels or subnets that all use the same technology
  - \* Section 5.6 deals with the question of reframing when IP packets do not map 1:1 into lower layer frames.

#### 5.1. IP-in-IP Tunnels with Tightly Coupled Shim Headers

A common pattern for many tunnelling protocols is to encapsulate an inner IP header with shim header(s) then an outer IP header. In many cases the shim header(s) always have to be tightly coupled to the outer IP header because they are not sufficient as outer headers in their own right. In such cases the shim header(s) and the outer IP header are always added (or removed) in the same operation. Therefore, in all such tightly coupled IP-in-IP tunnelling protocols, the rules in [RFC6040] for propagating the ECN field between the two IP headers SHOULD be applied directly.

Examples of tightly coupled IP-in-IP tunnelling protocols where [RFC6040] can be applied directly are:

- o L2TP [RFC2661]
- o GRE [RFC1701], [RFC2784]
- o PPTP [RFC2637]
- o GTP [GTPv1], [GTPv1-U], [GTPv2-C]
- o VXLAN [RFC7348].

#### 5.2. Wire Protocol Design: Indication of ECN Support

This section is intended to guide the redesign of any lower layer protocol that encapsulate IP to add native ECN support at the lower layer. It reflects the approaches used in [RFC6040] and in [RFC5129]. Therefore IP-in-IP tunnels or IP-in-MPLS or MPLS-in-MPLS

encapsulations that already comply with [RFC6040] or [RFC5129] will already satisfy this guidance.

A lower layer (or subnet) congestion notification system:

1. SHOULD NOT apply explicit congestion notifications to PDUs that are destined for legacy layer-4 transport implementations that will not understand ECN, and
2. SHOULD NOT apply explicit congestion notifications to PDUs if the egress of the subnet might not propagate congestion notifications onward into the higher layer.

We use the term ECN-PDUs for a PDU on a feedback loop that will propagate congestion notification properly because it meets both the above criteria. And a Not-ECN-PDU is a PDU on a feedback loop that does not meet both criteria, and will therefore not propagate congestion notification properly. A corollary of the above is that a lower layer congestion notification protocol:

3. SHOULD be able to distinguish ECN-PDUs from Not-ECN-PDUs.

Note that there is no need for all interior nodes within a subnet to be able to mark congestion explicitly. A mix of ECN and drop signals from different nodes is fine. However, if any interior nodes might generate ECN markings, guideline 2 above says that all relevant egress node(s) SHOULD be able to propagate those markings up to the higher layer.

In IP, if the ECN field in each PDU is cleared to the Not-ECT (not ECN-capable transport) codepoint, it indicates that the L4 transport will not understand congestion markings. A congested buffer must not mark these Not-ECT PDUs, and therefore drops them instead.

The mechanism a lower layer uses to distinguish the ECN-capability of PDUs need not mimic that of IP. All the above guidelines say is that the lower layer system, as a whole, should achieve the same outcome. For instance, ECN-capable feedback loops might use PDUs that are identified by a particular set of labels or tags. Alternatively, logical link protocols that use flow state might determine whether a PDU can be congestion marked by checking for ECN-support in the flow state. Other protocols might depend on out-of-band control signals.

The per-domain checking of ECN support in MPLS [RFC5129] is a good example of a way to avoid sending congestion markings to transports that will not understand them, without using any header space in the subnet protocol.

In MPLS, header space is extremely limited, therefore [RFC5129](#) does not provide a field in the MPLS header to indicate whether the PDU is an ECN-PDU or a Not-ECN-PDU. Instead, interior nodes in a domain are allowed to set explicit congestion indications without checking whether the PDU is destined for a transport that will understand them. Nonetheless, this is made safe by requiring that the network operator upgrades all decapsulating edges of a whole domain at once, as soon as even one switch within the domain is configured to mark rather than drop during congestion. Therefore, any edge node that might decapsulate a packet will be capable of checking whether the higher layer transport is ECN-capable. When decapsulating a CE-marked packet, if the decapsulator discovers that the higher layer (inner header) indicates the transport is not ECN-capable, it drops the packet--effectively on behalf of the earlier congested node (see Decapsulation Guideline 1 in [Section 5.4](#)).

It was only appropriate to define such an incremental deployment strategy because MPLS is targeted solely at professional operators, who can be expected to ensure that a whole subnetwork is consistently configured. This strategy might not be appropriate for other link technologies targeted at zero-configuration deployment or deployment by the general public (e.g. Ethernet). For such 'plug-and-play' environments it will be necessary to invent a failsafe approach that ensures congestion markings will never fall into black holes, no matter how inconsistently a system is put together. Alternatively, congestion notification relying on correct system configuration could be confined to flavours of Ethernet intended only for professional network operators, such as IEEE 802.1ah Provider Backbone Bridges (PBB).

QCN [[IEEE802.1Qau](#)] provides another example of how to indicate to lower layer devices that the end-points will not understand ECN. An operator can define certain 802.1p classes of service to indicate non-QCN frames and an ingress bridge is required to map arriving not-QCN-capable IP packets to one of these non-QCN 802.1p classes.

### 5.3. Encapsulation Guidelines

This section is intended to guide the redesign of any node that encapsulates IP with a lower layer header when adding native ECN support to the lower layer protocol. It reflects the approaches used in [[RFC6040](#)] and in [[RFC5129](#)]. Therefore IP-in-IP tunnels or IP-in-MPLS or MPLS-in-MPLS encapsulations that already comply with [[RFC6040](#)] or [[RFC5129](#)] will already satisfy this guidance.

1. Egress Capability Check: A subnet ingress needs to be sure that the corresponding egress of a subnet will propagate any congestion notification added to the outer header across the

subnet. This is necessary in addition to checking that an incoming PDU indicates an ECN-capable (L4) transport. Examples of how this guarantee might be provided include:

- \* by configuration (e.g. if any label switches in a domain support ECN marking, [RFC5129] requires all egress nodes to have been configured to propagate ECN)
  - \* by the ingress explicitly checking that the egress propagates ECN (e.g. TRILL uses IS-IS to check path capabilities before using critical options [I-D.ietf-trill-rfc7180bis])
  - \* by inherent design of the protocol (e.g. by encoding ECN marking on the outer header in such a way that a legacy egress that does not understand ECN will consider the PDU corrupt and discard it, thus at least propagating a form of congestion signal).
2. Egress Fails Capability Check: If the ingress cannot guarantee that the egress will propagate congestion notification, the ingress SHOULD disable ECN when it forwards the PDU at the lower layer. An example of how the ingress might disable ECN at the lower layer would be by setting the outer header of the PDU to identify it as a Not-ECN-PDU, assuming the subnet technology supports such a concept.
  3. Standard Congestion Monitoring Baseline: Once the ingress to a subnet has established that the egress will correctly propagate ECN, on encapsulation it SHOULD encode the same level of congestion in outer headers as is arriving in incoming headers. For example it might copy any incoming congestion notification into the outer header of the lower layer protocol.

This ensures that all outer headers reflect congestion accumulated along the whole upstream path since the Load Regulator, not just since the ingress of the subnet. A node that is not the Load Regulator SHOULD NOT re-initialise the level of CE markings in the outer to zero.

This guideline is intended to ensure that any bulk congestion monitoring of outer headers (e.g. by a network management node monitoring ECN in passing frames) is most meaningful. For instance, if an operator measures CE in 0.4% of passing outer headers, this information is only useful if the operator knows where the proportion of CE markings was last initialised to 0% (the Congestion Baseline). Such monitoring information will not be useful if some subnet ingress nodes reset all outer CE markings while others copy incoming CE markings into the outer.

Most information can be extracted if the Congestion Baseline is standardised at the node that is regulating the load (the Load Regulator--typically the data source). Then the operator can measure both congestion since the Load Regulator, and congestion since the subnet ingress. The latter might be measurable by subtracting the level of CE markings on inner headers from that on outer headers (see [Appendix C of \[RFC6040\]](#)).

#### 5.4. Decapsulation Guidelines

This section is intended to guide the redesign of any node that decapsulates IP from within a lower layer header when adding native ECN support to the lower layer protocol. It reflects the approaches used in [\[RFC6040\]](#) and in [\[RFC5129\]](#). Therefore IP-in-IP tunnels or IP-in-MPLS or MPLS-in-MPLS encapsulations that already comply with [\[RFC6040\]](#) or [\[RFC5129\]](#) will already satisfy this guidance.

A subnet egress SHOULD NOT simply copy congestion notification from outer headers to the forwarded header. It SHOULD calculate the outgoing congestion notification field from the inner and outer headers using the following guidelines. If there is any conflict, rules earlier in the list take precedence over rules later in the list:

1. If the arriving inner header is a Not-ECN-PDU it implies the L4 transport will not understand explicit congestion markings.  
Then:
  - \* If the outer header carries an explicit congestion marking, the packet SHOULD be dropped--the only indication of congestion that the L4 transport will understand.
  - \* If the outer is an ECN-PDU that carries no indication of congestion or a Not-ECN-PDU the PDU SHOULD be forwarded, but still as a Not-ECN-PDU.
2. If the outer header does not support explicit congestion notification (a Not-ECN-PDU), but the inner header does (an ECN-PDU), the inner header SHOULD be forwarded unchanged.
3. In some lower layer protocols congestion may be signalled as a numerical level, such as in the control frames of quantised congestion notification [\[IEEE802.1Qau\]](#). If such a multi-bit encoding encapsulates an ECN-capable IP data packet, a function will be needed to convert the quantised congestion level into the frequency of congestion markings in outgoing IP packets.



4. Congestion indications may be encoded by a severity level. For instance increasing levels of congestion might be encoded by numerically increasing indications, e.g. pre-congestion notification (PCN) can be encoded in each PDU at three severity levels in IP or MPLS [[RFC6660](#)].

If the arriving inner header is an ECN-PDU, where the inner and outer headers carry indications of congestion of different severity, the more severe indication SHOULD be forwarded in preference to the less severe.

5. The inner and outer headers might carry a combination of congestion notification fields that should not be possible given any currently used protocol transitions. For instance, if Encapsulation Guideline 3 in [Section 5.3](#) had been followed, it should not be possible to have a less severe indication of congestion in the outer than in the inner. It MAY be appropriate to log unexpected combinations of headers and possibly raise an alarm.

If a safe outgoing codepoint can be defined for such a PDU, the PDU SHOULD be forwarded rather than dropped. Some implementers discard PDUs with currently unused combinations of headers just in case they represent an attack. However, an approach using alarms and policy-mediated drop is preferable to hard-coded drop, so that operators can keep track of possible attacks but currently unused combinations are not precluded from future use through new standards actions.

#### 5.5. Sequences of Similar Tunnels or Subnets

In some deployments, particularly in 3GPP networks, an IP packet may traverse two or more IP-in-IP tunnels in sequence that all use identical technology (e.g. GTP).

In such cases, it would be sufficient for every encapsulation and decapsulation in the chain to comply with [RFC 6040](#). Alternatively, as an optimisation, a node that decapsulates a packet and immediately re-encapsulates it for the next tunnel MAY copy the incoming outer ECN field directly to the outgoing outer and the incoming inner ECN field directly to the outgoing inner. Then the overall behavior across the sequence of tunnel segments would still be consistent with [RFC 6040](#).

[Appendix C of RFC6040](#) describes how a tunnel egress can monitor how much congestion has been introduced within a tunnel. A network operator might want to monitor how much congestion had been introduced within a whole sequence of tunnels. Using the technique

Internet-Draft

ECN Encapsulation Guidelines

March 2015

in [Appendix C of RFC6040](#) at the final egress, the operator could monitor the whole sequence of tunnels, but only if the above optimisation were used consistently along the sequence of tunnels, in order to make it appear as a single tunnel. Therefore, tunnel endpoint implementations SHOULD allow the operator to configure whether this optimisation is enabled.

When ECN support is added to a subnet technology, consideration SHOULD be given to a similar optimisation between subnets in sequence if they all use the same technology.

#### 5.6. Reframing and Congestion Markings

The guidance in this section is worded in terms of framing boundaries, but it applies equally whether the protocol data units are frames, cells or packets.

Where framing boundaries are different between two layers, congestion indications SHOULD be propagated on the basis that a congestion indication on a PDU applies to all the octets in the PDU. On average, an encapsulator or decapsulator SHOULD approximately preserve the number of marked octets arriving and leaving (counting the size of inner headers, but not added encapsulating headers).

The next departing frame SHOULD be immediately marked even if only enough incoming marked octets have arrived for part of the departing frame. This ensures that any outstanding congestion marked octets are propagated immediately, rather than held back waiting for a frame no bigger than the outstanding marked octets--which might involve a long wait.

For instance, an algorithm for marking departing frames could maintain a counter representing the balance of arriving marked octets minus departing marked octets. It adds the size of every marked frame that arrives and if the counter is positive it marks the next frame to depart and subtracts its size from the counter. This will often leave a negative remainder in the counter, which is deliberate.

#### 6. Feed-Up-and-Forward Mode: Guidelines for Adding Congestion Notification

The guidance in this section is applicable when IP packets:

- o are encapsulated in Ethernet headers;
- o are forwarded by the eNode-B (base station) of a 3GPP radio access network, which is required to apply ECN marking during congestion [[LTE-RA](#)].

This guidance also generalises to encapsulation by other subnet technologies with no native support for explicit congestion notification at the lower layer, but with support for finding and processing an IP header. It is unlikely to be applicable or necessary for IP-in-IP encapsulation, where feed-forward-and-up mode based on [\[RFC6040\]](#) would be more appropriate.

Marking the IP header while switching at layer-2 (by using a layer-3 switch) or while forwarding in a radio access network seems to represent a layering violation. However, it can be considered as a benign optimisation if the guidelines below are followed. Feed-up-and-forward is certainly not a general alternative to implementing feed-forward congestion notification in the lower layer, because:

- o IPv4 and IPv6 are not the only layer-3 protocols that might be encapsulated by lower layer protocols
- o Link-layer encryption might be in use, making the layer-2 payload inaccessible
- o Many Ethernet switches do not have 'layer-3 switch' capabilities so they cannot read or modify an IP payload
- o It might be costly to find an IP header (v4 or v6) when it may be encapsulated by more than one lower layer header, e.g. Ethernet MAC in MAC [\[IEEE802.1Qah\]](#).

Nonetheless, configuring lower layer equipment to look for an ECN field in an encapsulated IP header is a useful optimisation. If the implementation follows the guidelines below, this optimisation does not have to be confined to a controlled environment such as within a data centre; it could usefully be applied on any network--even if the operator is not sure whether the above issues will never apply:

1. If a native lower-layer congestion notification mechanism exists for a subnet technology, it is safe to mix feed-up-and-forward with feed-forward-and-up on other switches in the same subnet. However, it will generally be more efficient to use the native mechanism.
2. The depth of the search for an IP header SHOULD be limited. If an IP header is not found soon enough, or an unrecognised or unreadable header is encountered, the switch SHOULD resort to an alternative means of signalling congestion (e.g. drop, or the native lower layer mechanism if available).
3. It is sufficient to use the first IP header found in the stack; the egress of the relevant tunnel can propagate congestion

notification upwards to any more deeply encapsulated IP headers later.

#### 7. Feed-Backward Mode: Guidelines for Adding Congestion Notification

It can be seen from [Section 4.3](#) that congestion notification in a subnet using feed-backward mode has generally not been designed to be directly coupled with IP layer congestion notification. The subnet attempts to minimise congestion internally, and if the incoming load at the ingress exceeds the capacity somewhere through the subnet, the layer 3 buffer into the ingress backs up. Thus, a feed-backward mode subnet is in some sense similar to a null mode subnet, in that there is no need for any direct interaction between the subnet and higher layer congestion notification. Therefore no detailed protocol design guidelines are appropriate. Nonetheless, a more general guideline is appropriate:

1. A subnetwork technology intended to eventually interface to IP SHOULD NOT be designed using only the feed-backward mode, which is certainly best for a stand-alone subnet, but would need to be modified to work efficiently as part of the wider Internet, because IP uses feed-forward-and-up mode.

The feed-backward approach at least works beneath IP, where the term 'works' is used only in a narrow functional sense because feed-backward can result in very inefficient and sluggish congestion control--except if it is confined to the subnet directly connected to the original data source, when it is faster than feed-forward. It would be valid to design a protocol that could work in feed-backward mode for paths that only cross one subnet, and in feed-forward-and-up mode for paths that cross subnets.

In the early days of TCP/IP, a similar feed-backward approach was tried for explicit congestion signalling, using source-quench (SQ) ICMP control packets. However, SQ fell out of favour and is now formally deprecated [[RFC6633](#)]. The main problem was that it is hard for a data source to tell the difference between a spoofed SQ message and a quench request from a genuine buffer on the path. It is also hard for a lower layer buffer to address an SQ message to the original source port number, which may be buried within many layers of headers, and possibly encrypted.

Quantised congestion notification (QCN--also known as backward congestion notification or BCN) [[IEEE802.1Qau](#)] uses a feed-backward mode structurally similar to ATM's relative rate mechanism. However, QCN confines its applicability to scenarios such as some data centres where all endpoints are directly attached by the same Ethernet technology. If a QCN subnet were later connected into a wider IP-

Internet-Draft

ECN Encapsulation Guidelines

March 2015

based internetwork (e.g. when attempting to interconnect multiple data centres) it would suffer the inefficiency shown Figure 3.

#### 8. IANA Considerations (to be removed by RFC Editor)

This memo includes no request to IANA.

#### 9. Security Considerations

If a lower layer wire protocol is redesigned to include explicit congestion signalling in-band in the protocol header, care SHOULD be taken to ensure that the field used is specified as mutable during transit. Otherwise interior nodes signalling congestion would invalidate any authentication protocol applied to the lower layer header--by altering a header field that had been assumed as immutable.

The redesign of protocols that encapsulate IP in order to propagate congestion signals between layers raises potential signal integrity concerns. Experimental or proposed approaches exist for assuring the end-to-end integrity of in-band congestion signals, e.g.:

- o Congestion exposure (ConEx) for networks to audit that their congestion signals are not being suppressed by other networks or by receivers, and for networks to police that senders are responding sufficiently to the signals, irrespective of the transport protocol used [[I-D.ietf-conex-abstract-mech](#)].
- o The ECN nonce [[RFC3540](#)] for a TCP sender to detect whether a network or the receiver is suppressing congestion signals.
- o A test with the same goals as the ECN nonce, but without the need for the receiver to co-operate with the protocol [[I-D.moncaster-tcpm-rcv-cheat](#)].

Given these end-to-end approaches are already being specified, it would make little sense to attempt to design hop-by-hop congestion signal integrity into a new lower layer protocol, because end-to-end integrity inherently achieves hop-by-hop integrity.

#### 10. Conclusions

Following the guidance in the document enables ECN support to be extended to numerous protocols that encapsulate IP (v4 & v6) in a consistent way, so that IP continues to fulfil its role as an end-to-end interoperability layer. This includes:

Internet-Draft

ECN Encapsulation Guidelines

March 2015

- o A wide range of tunnelling protocols with various forms of shim header between two IP headers;
- o A wide range of subnet technologies, particularly those that work in the same 'feed-forward-and-up' mode that is used to support ECN in IP and MPLS.

Guidelines have been defined for supporting propagation of ECN between Ethernet and IP on so-called Layer-3 Ethernet switches, using a 'feed-up-an-forward' mode. This approach could enable other subnet technologies to pass ECN signals into the IP layer, even if they do not support ECN natively.

Finally, attempting to add ECN to a subnet technology in feed-backward mode is deprecated except in special cases, due to its likely sluggish response to congestion.

## 11. Acknowledgements

Thanks to Gorrry Fairhurst for extensive reviews. Thanks also to the following reviewers: Ingemar Johansson and Piers O'Hanlon and Michael Welzl, who pointed out that lower layer congestion notification signals may have different semantics to those in IP.

Bob Briscoe was part-funded by the European Community under its Seventh Framework Programme through the Trilogy project (ICT-216372) for initial drafts and through the Reducing Internet Transport Latency (RITE) project (ICT-317700) subsequently. The views expressed here are solely those of the authors.

## 12. Comments Solicited

Comments and questions are encouraged and very welcome. They can be addressed to the IETF Transport Area working group mailing list <tsvwg@ietf.org>, and/or to the authors.

## 13. References

### 13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", [RFC 3168](#), September 2001.

---

Briscoe, et al.

Expires September 27, 2015

[Page 23]

Internet-Draft

ECN Encapsulation Guidelines

March 2015

- [RFC3819] Karn, P., Bormann, C., Fairhurst, G., Grossman, D., Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L. Wood, "Advice for Internet Subnetwork Designers", [BCP 89](#), [RFC 3819](#), July 2004.
- [RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", [BCP 124](#), [RFC 4774](#), November 2006.
- [RFC5129] Davie, B., Briscoe, B., and J. Tay, "Explicit Congestion Marking in MPLS", [RFC 5129](#), January 2008.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", [RFC 6040](#), November 2010.

### 13.2. Informative References

- [ATM-TM-ABR] Cisco, "Understanding the Available Bit Rate (ABR) Service Category for ATM VCs", Design Technote 10415, June 2005.
- [Buck00] Buckwalter, J., "Frame Relay: Technology and Practice", Pub. Addison Wesley ISBN-13: 978-0201485240, 2000.
- [DCTCP] Alizadeh, M., Greenberg, A., Maltz, D., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and M. Sridharan, "Data Center TCP (DCTCP)", ACM SIGCOMM CCR 40(4)63--74, October 2010, <<http://portal.acm.org/citation.cfm?id=1851192>>.
- [GTPv1] 3GPP, "GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface", Technical Specification TS 29.060.
- [GTPv1-U] 3GPP, "General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U)", Technical Specification TS 29.281.
- [GTPv2-C] 3GPP, "Evolved General Packet Radio Service (GPRS) Tunnelling Protocol for Control plane (GTPv2-C)", Technical Specification TS 29.274.
- [I-D.ietf-aqm-recommendation] Baker, F. and G. Fairhurst, "IETF Recommendations Regarding Active Queue Management", [draft-ietf-aqm-recommendation-11](#) (work in progress), February 2015.

Briscoe, et al.

Expires September 27, 2015

[Page 24]

Internet-Draft

ECN Encapsulation Guidelines

March 2015

[I-D.ietf-conex-abstract-mech]

Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism and Requirements", [draft-ietf-conex-abstract-mech-13](#) (work in progress), October 2014.

[I-D.ietf-trill-rfc7180bis]

Eastlake, D., Zhang, M., Perlman, R., Banerjee, A., Ghanwani, A., and S. Gupta, "TRILL: Clarifications, Corrections, and Updates", [draft-ietf-trill-rfc7180bis-04](#) (work in progress), March 2015.

[I-D.moncaster-tcpm-rcv-cheat]

Moncaster, T., Briscoe, B., and A. Jacquet, "A TCP Test to Allow Senders to Identify Receiver Non-Compliance", [draft-moncaster-tcpm-rcv-cheat-03](#) (work in progress), July 2014.

[IEEE802.1Qah]

IEEE, "IEEE Standard for Local and Metropolitan Area Networks--Virtual Bridged Local Area Networks--Amendment 6: Provider Backbone Bridges", IEEE Std 802.1Qah-2008, August 2008,  
<<http://www.ieee802.org/1/pages/802.1ah.html>>.

(Access Controlled link within page)

[IEEE802.1Qau]

Finn, N., Ed., "IEEE Standard for Local and Metropolitan Area Networks--Virtual Bridged Local Area Networks - Amendment 13: Congestion Notification", IEEE Std 802.1Qau-2010, March 2010, <<http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5454061>>.

(Access Controlled link within page)

[ITU-T.I.371]

ITU-T, "Traffic Control and Congestion Control in B-ISDN", ITU-T Rec. I.371 (03/04), March 2004,  
<<http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5454061>>.

[LTE-RA]

3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2", Technical Specification TS 36.300.

[RFC1323]

Jacobson, V., Braden, B., and D. Borman, "TCP Extensions for High Performance", [RFC 1323](#), May 1992.

Briscoe, et al.

Expires September 27, 2015

[Page 25]



Internet-Draft

ECN Encapsulation Guidelines

March 2015

- [RFC1701] Hanks, S., Li, T., Farinacci, D., and P. Traina, "Generic Routing Encapsulation (GRE)", [RFC 1701](#), October 1994.
- [RFC2003] Perkins, C., "IP Encapsulation within IP", [RFC 2003](#), October 1996.
- [RFC2637] Hamzeh, K., Pall, G., Verthein, W., Taarud, J., Little, W., and G. Zorn, "Point-to-Point Tunneling Protocol", [RFC 2637](#), July 1999.
- [RFC2661] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", [RFC 2661](#), August 1999.
- [RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", [RFC 2784](#), March 2000.
- [RFC2884] Hadi Salim, J. and U. Ahmed, "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks", [RFC 2884](#), July 2000.
- [RFC2983] Black, D., "Differentiated Services and Tunnels", [RFC 2983](#), October 2000.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", [RFC 3540](#), June 2003.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", [RFC 4301](#), December 2005.
- [RFC6633] Gont, F., "Deprecation of ICMP Source Quench Messages", [RFC 6633](#), May 2012.
- [RFC6660] Briscoe, B., Moncaster, T., and M. Menth, "Encoding Three Pre-Congestion Notification (PCN) States in the IP Header Using a Single Diffserv Codepoint (DSCP)", [RFC 6660](#), July 2012.
- [RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", [RFC 7348](#), August 2014.

Briscoe, et al.

Expires September 27, 2015

[Page 26]

---

Internet-Draft

ECN Encapsulation Guidelines

March 2015

#### Appendix A. Outstanding Document Issues

1. [GF] Concern that certain guidelines warrant a MUST (NOT) rather than a SHOULD (NOT). Given the guidelines say that if any SHOULD (NOT)s are not followed, a strong justification will be needed, they have been left as SHOULD (NOT) pending further list discussion. In particular:
  - \* If inner is a Not-ECN-PDU and Outer is CE (or highest severity congestion level), MUST (not SHOULD) drop?
2. Consider whether an IETF Standard Track doc will be needed to Update the IP-in-IP protocols listed in [Section 5.1](#)--at least those that the IET

#### Appendix B. Changes in This Version (to be removed by RFC Editor)

From ietf-01 to ietf-02:

- \* Added Section for guidelines that are applicable in all cases.
- \* Updated references.

From ietf-00 to ietf-01: Updated references.

From briscoe-04 to ietf-00: Changed filename following tsvwg adoption.

From briscoe-03 to 04:

- \* Re-arranged the introduction to describe the purpose of the document first before introducing ECN in more depth. And clarified the introduction throughout.
- \* Added applicability to 3GPP TS 36.300.

From briscoe-02 to 03:

- \* Scope section:
  - + Added dependence on correct propagation of traffic class information
  - + For the feed-backward mode, deemed multicast and anycast out of scope
- \* Ensured all guidelines referring to subnet technologies also refer to tunnels and vice versa by adding applicability

---

Briscoe, et al.

Expires September 27, 2015

[Page 27]

Internet-Draft

ECN Encapsulation Guidelines

March 2015

sentences at the start of sections [4.1](#), [4.2](#), [4.3](#), [4.4](#), [4.6](#) and 5.

- \* Added Security Considerations on ensuring congestion signal fields are classed as immutable and on using end-to-end congestion signal integrity technologies rather than hop-by-hop.

From briscoe-01 to 02:

- \* Added authors: JK & PT
- \* Added
  - + [Section 4.1](#) "IP-in-IP Tunnels with Tightly Coupled Shim Headers"
  - + [Section 4.5](#) "Sequences of Similar Tunnels or Subnets"
  - + roadmap at the start of [Section 4](#), given the subsections have become quite fragmented.
  - + [Section 9](#) "Conclusions"
- \* Clarified why transports are starting to be able to saturate interior links
- \* Under [Section 1.1](#), addressed the question of alternative signal semantics and included multicast & anycast.
- \* Under [Section 3.1](#), included a 3GPP example.
- \* [Section 4.2](#). "Wire Protocol Design":
  - + Altered guideline 2. to make it clear that it only applies to the immediate subnet egress, not later ones
  - + Added a reminder that it is only necessary to check that ECN propagates at the egress, not whether interior nodes mark ECN
  - + Added example of how QCN uses 802.1p to indicate support for QCN.
- \* Added references to [Appendix C of RFC6040](#), about monitoring the amount of congestion signals introduced within a tunnel

Internet-Draft

ECN Encapsulation Guidelines

March 2015

- \* [Appendix A](#): Added more issues to be addressed, including plan to produce a standards track update to IP-in-IP tunnel protocols.
- \* Updated acks and references

From briscoe-00 to 01:

- \* Intended status: BCP (was Informational) & updates 3819 added.
- \* Briefer Introduction: Introductory para justifying benefits of ECN. Moved all but a brief enumeration of modes of operation to their own new section (from both Intro & Scope). Introduced incr. deployment as most tricky part.
- \* Tightened & added to terminology section
- \* Structured with Modes of Operation, then Guidelines section for each mode.
- \* Tightened up guideline text to remove vagueness / passive voice / ambiguity and highlight main guidelines as numbered items.
- \* Added Outstanding Document Issues Appendix
- \* Updated references

#### Authors' Addresses

Bob Briscoe  
BT  
B54/77, Adastral Park  
Martlesham Heath  
Ipswich IP5 3RE  
UK

Phone: +44 1473 645196  
EMail: [bob.briscoe@bt.com](mailto:bob.briscoe@bt.com)  
URI: <http://bobbiscoe.net/>

John Kaippallimalil  
Huawei  
5340 Legacy Drive, Suite 175  
Plano, Texas 75024  
USA

EMail: [john.kaippallimalil@huawei.com](mailto:john.kaippallimalil@huawei.com)

Internet-Draft

ECN Encapsulation Guidelines

March 2015

Pat Thaler  
Broadcom Corporation  
5025 Keane Drive  
Carmichael, CA 95608  
USA

EMail: [pthaler@broadcom.com](mailto:pthaler@broadcom.com)

Briscoe, et al.

Expires September 27, 2015

[Page 30]

Network Working Group  
Internet-Draft  
Updates: 3168 (if approved)  
Intended status: Experimental  
Expires: January 1, 2016

N. Khademi  
M. Welzl  
University of Oslo  
G. Armitage  
Swinburne University of  
Technology  
G. Fairhurst  
University of Aberdeen  
June 30, 2015

TCP Alternative Backoff with ECN (ABE)  
draft-khademi-alternativebackoff-ecn-00

#### Abstract

This memo updates the TCP sender-side reaction to a congestion notification received via Explicit Congestion Notification (ECN). The updated method is less conservative than the TCP reaction in response to loss. The intention is to achieve good throughput when the queue at the bottleneck is smaller than the bandwidth-delay-product of the connection. This is more likely when an Active Queue Management (AQM) mechanism has ECN-marked a packet than when a packet was lost. Future versions of this document will discuss SCTP as well as other transports using ECN.

#### Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 1, 2016.

#### Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

Internet-Draft

ABE

June 2015

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4.e](#) of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Updating the Sender-side ECN Reaction . . . . .	3
3. Acknowledgements . . . . .	4
4. IANA Considerations . . . . .	4
5. Security Considerations . . . . .	4
6. References . . . . .	5
6.1. Normative References . . . . .	5
6.2. Informative References . . . . .	5
Authors' Addresses . . . . .	5

Internet-Draft

ABE

June 2015

## 1. Introduction

Explicit Congestion Notification (ECN) is specified in [RFC3168]. This allows a network device that uses Active Queue Management (AQM) to CE-mark rather than drop ECN-capable packets when incipient congestion is detected.

When an ECN-capable transport is used over a path that supports ECN, it provides the opportunity for flows to improve their performance in the presence of incipient congestion [I-D.AQM-ECN-benefits].

[RFC3168] not only specifies the router use of the ECN field, it also specifies a TCP procedure for using ECN. This states that a TCP sender should treat the ECN indication of congestion in the same way as that of a non-ECN-Capable TCP flow experiencing loss, by halving the congestion window "cwnd" and by reducing the slow start threshold "ssthresh". [RFC5681] stipulates that TCP congestion control sets "ssthresh" to  $\max(\text{FlightSize} / 2, 2 * \text{SMSS})$  in response to packet loss. Consequently, a non-ECN enabled standard TCP flow using this reaction needs significant queue space: it can only fully utilize a bottleneck when the length of the link queue (or the AQM dropping threshold) is at least the bandwidth-delay product of the flow. CUBIC can fully utilize a link with a smaller queue because it multiplies the current cwnd with 0.8 in response to packet loss [ID.CUBIC] (since kernel version 2.6.25 (2008), the Linux implementation uses a value of 0.7). In case of a DropTail (FIFO) queue without AQM, this increases the risk of creating a standing queue [CODEL2012].

Devices implementing AQM should be the only source of marking for packets from ECN-capable senders. AQM mechanisms typically strive to maintain a small queue length, regardless of the bandwidth-delay product of a flows passing through them. Receipt of a CE-mark therefore indicates that reacting less conservatively would be appropriate.

Results reported in [ABE-paper] show significant benefits (improved throughput, resulting in reduced completion times for short flows) when reacting to ECN-Echo by multiplying cwnd and ssthresh with a value in the range [0.7..0.85] rather than 0.5.

## 2. Updating the Sender-side ECN Reaction

This document specifies an updated TCP reaction to the receipt of CE-marked packets.

The first paragraph of Section 6.1.2, "The TCP Sender", in [RFC3168] contains the following text:



Internet-Draft

ABE

June 2015

"If the sender receives an ECN-Echo (ECE) ACK packet (that is, an ACK packet with the ECN-Echo flag set in the TCP header), then the sender knows that congestion was encountered in the network on the path from the sender to the receiver. The indication of congestion should be treated just as a congestion loss in non-ECN-Capable TCP. That is, the TCP source halves the congestion window 'cwnd' and reduces the slow start threshold 'ssthresh'."

This memo updates this text to:

"If the sender receives an ECN-Echo (ECE) ACK packet (that is, an ACK packet with the ECN-Echo flag set in the TCP header), then the sender knows that congestion was encountered in the network on the path from the sender to the receiver. The indication of congestion should induce a less conservative reaction than loss: the TCP source multiplies the congestion window 'cwnd' with 0.8 and reduces the slow start threshold 'ssthresh'."

### 3. Acknowledgements

The authors were part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the authors.

The authors would like to thank the following people for their contributions to [\[ABE-paper\]](#): Chamil Kulatunga, David Ros, Stein Gjessing, Sebastian Zander.

### 4. IANA Considerations

XX RFC ED - PLEASE REMOVE THIS SECTION XXX

This memo includes no request to IANA.

### 5. Security Considerations

This document describes a change to TCP congestion control that can make a TCP sender more aggressive than flows using [RFC 3819](#). This could lead to an unfair allocation in rates at a bottleneck. Similar unfairness is also exhibited by other congestion control mechanisms that have been in use in the Internet for many years (e.g. Cubic [\[ID.CUBIC\]](#)).

XXX This section to be completed XXX.

Internet-Draft

ABE

June 2015

## 6. References

### 6.1. Normative References

- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", [RFC 3168](#), September 2001.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", [RFC 5681](#), September 2009.

### 6.2. Informative References

- [ABE-paper]  
Khademi, N., Welzl, M., Armitage, G., Kulatunga, C., Ros, D., Fairhurst, G., Gjessing, S., and S. Zander, "Alternative Backoff: Achieving Low Latency and High Throughput with ECN and AQM", CAIA technical report CAIA-TR-150710A, July 2015 <http://caia.swin.edu.au/reports/150710A/CAIA-TR-150710A.pdf>, NOTE: this document may not be available at draft submission date. We will do our best to make it available as soon as possible, and definitely before IETF-93.
- [CODEL2012]  
Nichols, K. and V. Jacobson, "Controlling Queue Delay", July 2012, <<http://queue.acm.org/detail.cfm?id=2209336>>.
- [I-D.AQM-ECN-benefits]  
Fairhurst, G. and M. Welzl, "The Benefits of using Explicit Congestion Notification (ECN)", Internet-draft, IETF work-in-progress [draft-ietf-aqm-ecn-benefits-05](#), June 2015.
- [ID.CUBIC]  
Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", Internet-draft, IETF work-in-progress [draft-ietf-tcpm-cubic-00](#), June 2015.

Internet-Draft

ABE

June 2015

## Authors' Addresses

Naeem Khademi  
University of Oslo  
PO Box 1080 Blindern  
Oslo, N-0316  
Norway

Email: naeemk@ifi.uio.no

Michael Welzl  
University of Oslo  
PO Box 1080 Blindern  
Oslo, N-0316  
Norway

Email: michawe@ifi.uio.no

Grenville Armitage  
Centre for Advanced Internet Architectures  
Swinburne University of Technology  
PO Box 218  
John Street, Hawthorn  
Victoria, 3122  
Australia

Email: garmitage@swin.edu.au

Godred Fairhurst  
University of Aberdeen  
School of Engineering, Fraser Noble Building  
Aberdeen, AB24 3UE  
UK

Email: gorrry@erg.abdn.ac.uk

# Alternative Backoff: Achieving Low Latency and High Throughput with ECN and AQM

Naeem Khademi\*, Michael Welzl\*, Grenville Armitage†, Chamil Kulatunga‡  
 naeemk@ifi.uio.no, michawe@ifi.uio.no, garmitage@swin.edu.au, chamil@erg.abdn.ac.uk  
 David Ros§, Gorry Fairhurst‡, Stein Gjessing\*, Sebastian Zander¶  
 dros@simula.no, gorry@erg.abdn.ac.uk, steing@ifi.uio.no, s.zander@murdoch.edu.au

Centre for Advanced Internet Architectures Technical Report 150710A  
 Swinburne University of Technology  
 Melbourne, Australia

**Abstract**—CoDel and PIE are recently proposed Active Queue Management (AQM) mechanisms that minimize the time packets spend enqueued at a bottleneck, instantiating shallow, 5 ms to 20 ms buffers with short-term packet burst tolerance. However, shallow buffering causes noticeable TCP performance degradation when a path’s underlying round trip time (RTT) heads above 60 ms to 80 ms (not uncommon with cross-continental and inter-continental traffic). Using less-aggressive multiplicative backoffs is known to compensate for shallow bottleneck buffering. We propose ABE: “Alternative Backoff with ECN”, which consists of enabling Explicit Congestion Notification (ECN) and letting individual TCP senders use a larger multiplicative decrease factor in reaction to ECN-marks from AQM-enabled bottlenecks. Using a mix of experiments, theory and simulations with standard NewReno and CUBIC flows, we show significant performance gains in lightly-multiplexed scenarios, without losing the delay-reduction benefits of deploying CoDel or PIE. ABE is a sender-side-only modification that can be deployed incrementally (requiring no flag-day) and offers a compelling reason to deploy and enable ECN across the Internet.

## I. INTRODUCTION

Recent years have seen increasing mainstream awareness of how critical low latency (delay) is to today’s Internet and end users’ quality of experience. Expectations are being increasingly driven by interactive applications such as Voice over IP (VoIP), online games, online

trading [31] and even time-sensitive, transactional web-based shopping [26].

The delay experienced by any given packet is heavily influenced by routing choices (distance), link speeds (serialisation) and queuing (buffering at bottlenecks during periods of congestion). Increasing network speeds have reduced the relative contribution of serialisation, and therefore placed more focus on the size and management of bottleneck buffers.

A key influence on end user experience is the way bottleneck buffering interacts with capacity estimation techniques of common transport layers. The Internet’s reliance on statistical multiplexing requires buffering to absorb transient periods of congestion. Loss-based TCP algorithms will fill a bottleneck’s available buffer space before backing-off after the inevitable packet loss (congestion signal). When available buffering is significantly in excess of requirements the resulting queuing delay can far outweigh any delay contributed by distance [19].

Two complementary solutions have emerged—Active Queue Management (AQM) [3] and Explicit Congestion Notification (ECN) [41]. AQM schemes aim to provide earlier feedback about the onset of congestion, and thus reduce buffer filling during congestion. ECN delivers congestion feedback by adding new information to packets in transit, avoiding the detrimental side-effects of dropping packets to signal congestion [46].

The problem we face is that modern AQM schemes can interact badly with the traditional TCP response to congestion notification. A common rule-of-thumb is to allocate buffering at least equivalent to a path’s intrinsic ‘bandwidth delay product’ (BDP) enabling TCP to achieve close to 100% path utilisation. Yet the design goal of AQM schemes, such as Controlled Delay

\*Department of Informatics, University of Oslo, Norway

†Centre for Advanced Internet Architectures, Swinburne University of Technology, Australia

‡School of Engineering, University of Aberdeen, United Kingdom

§Simula Research Laboratory, Norway

¶School of Engineering and Information Technology, Murdoch University, Australia

(CoDel) [34], [35] and Proportional-Integral controller Enhanced (PIE) [37], [38], is to effectively instantiate a shallow bottleneck buffer with burst tolerance. So TCP performance suffers once a path's BDP exceeds the bottleneck AQM scheme's effective buffer size, whether congestion is signalled by packet loss or ECN.

#### A. ABE: "Alternative Backoff with ECN"

We propose a novel way of utilizing ECN that enables ECN deployment with a range of ECN marking behaviours and can be shown to continue to enable low latency and high throughput even for a High BDP path.

Alternative Backoff with ECN (ABE) can be summarised as follows:

- Upon packet loss, a TCP sender reduces its congestion window (*cwnd*) as usual (e.g., NewReno would reduce *cwnd* by 50%, CUBIC by 30%).
- Upon receipt of an ECN mark, a TCP sender reduces *cwnd* by *less* than the usual response for loss.

ABE is based on the intuition that meaningful ECN marks are generated by AQM schemes whose congestion indications are proactively aimed at keeping buffer occupancy low. An ABE sender thus compensates by backing off less, reducing the likelihood of the shallow bottleneck buffer draining to empty. Smaller backoffs will continue to occur as the AQM continues to react, ensuring the traffic does not build a long standing queue.

The idea of backing off less in response to an ECN mark is not new, as the authors of [29] proposed to use a larger multiplicative decrease factor in conjunction with a smaller additive increase factor for ECN in 2002. However [29] assumes the AQM on the bottleneck to be RED [17] (since this work was performed before the introduction of CoDel [34], [35] and PIE [37], [38]) which may not necessarily be deployed to instantiate a shallow buffer – e.g. not necessarily with low marking thresholds. The ABE mechanism differs from the work in [29] since it takes into consideration the default values of the state-of-the-art AQM mechanisms (CoDel and PIE) which aim to instantiate a shallow buffer and also solely relies on updating the multiplicative decrease factor.

ABE is a straightforward sender-side modification that may be deployed incrementally, and requires only that ECN is enabled in both end systems and AQM-managed bottlenecks along the path. An ABE sender's behaviour is simply regular TCP if either the destination or path do not support ECN. Finally, if an ABE sender's flow traverses a bottleneck whose AQM instantiates a relatively deep queue instead, then either packet losses

or successive ECN marks will still ensure the sender continues to back off appropriately.

#### B. ECN past and future

RFC 3168 [41] defined ECN in 2001. It describes a simple marking method, replacing AQM dropping with ECN marking. Since then it has been widely implemented in hosts, but is not widely used [44]. This can partly be attributed to early experiences showing incorrect behaviour by a subset of middleboxes resulted in failure of ECN-enabled TCP connections to 8% of web servers tested in 2000 [36]. This reduced to < 1% of web servers tested in 2004 [32], yet in 2011 there were still a non-trivial number of paths mangling the ECN field in the IP header [9].

In 2014, extensive active measurements showed the majority of the top million web servers provided server-side ECN negotiation [44]. Wide client-side support for ECN-fallback means only 0.5% of websites suffer additional connection setup latency when fallback per RFC 3168 is correctly implemented.

Baker and Fairhurst [8] recently recommended deployment of AQM with ECN within the Internet. ECN-marking is now supported by default in CoDel and PIE within the Linux kernel and is enabled by default in router firmware, e.g., CeroWRT [1].

In light of increased interest in AQM, ABE provides a new incentive to deploy and enable ECN more widely. ABE is a new method to replace the simple TCP response defined in [41]. ECN support has matured to the point where there is little downside in doing so.

#### C. Paper structure

The rest of this paper is structured as follows. We motivate ABE in Section II, showing the inter-relationship between TCP backoff, bottleneck buffer size and path BDP, and demonstrating how TCP degrades over CoDel and PIE bottlenecks at plausible domestic and international round-trip times (RTTs). ABE is introduced in Section III. Section IV evaluates ABE using a combination of simulations and real-world experiments; both Linux and FreeBSD systems, considering CUBIC and NewReno congestion controllers, were used, which covers the majority of TCP variants actually deployed in the Internet. Related work is reviewed in Section V. Section VI concludes with a discussion of potential issues and ideas for future work.

## II. PROBLEM AND BACKGROUND

TCP congestion control [22] is based on an Additive Increase, Multiplicative Decrease (AIMD) mechanism

for controlling the congestion window. A decrease factor  $\beta = 0.5$  has commonly been used in the congestion avoidance phase. However, the convergence of AIMD is due to its multiplicative behaviour, irrespective of the value of the backoff factor  $\beta$ ; there are no major reasons why the value should be 0.5, and [22] states this choice was a heuristic, conservative one.

The choice of  $\beta$  may have implications in terms of link utilisation. To better understand the problem with AQMs enforcing small queues on long-RTT paths, we revisit how backoff factor, path characteristics and buffer space are intertwined.

#### A. TCP backoff, path characteristics and link utilisation

Consider a bottleneck link with capacity  $C$  and Drop-Tail buffer of size  $b$ , traversed by a single long-lived TCP flow. The BDP of the path,  $C \times RTT$ , will be denoted by  $\delta$ . We assume that: (a) the TCP flow is operating in congestion avoidance, following a linear additive-increase function with parameter  $\alpha = 1$ , and multiplicative decrease factor  $\beta \in (0, 1)$ ; (b) transmission is not limited by a small TCP receiver window. Bottleneck utilisation will be denoted as  $U$ .

Using an argument similar to [6, § 2] but with an arbitrary value for  $\beta$ , the only way to ensure  $U = 1$  is with a minimum amount of buffering given by:

$$b \geq \delta \frac{1 - \beta}{\beta}. \quad (1)$$

With  $\beta = 0.5$ , (1) yields the well-known rule-of-thumb for buffer sizing:  $b \geq \delta$ . The “sweet spot” corresponds to:  $b = \delta$ ; larger  $b$  merely results in a longer queue.

Eq. (1) can be rewritten as

$$\beta \geq \frac{\delta}{b + \delta}, \quad (2)$$

where it becomes obvious that, for given path characteristics ( $\delta$ ), a shallower buffer (smaller  $b$ ) requires a larger  $\beta$  in order for (2) to hold, i.e., to sustain  $U = 1$ .

Figure 1 illustrates the  $U < 1$  case, i.e., when (1)-(2) do not hold. Buffering permits  $cwnd$  to grow up to a maximum value of  $\delta + b$  before packet loss occurs.

Since we assume that  $\alpha = 1$ ,  $cwnd$  increases by one unit per RTT from  $\beta(b + \delta)$  to  $b + \delta$ , the sawtooth has a slope of 1 and duration between two loss events of  $(1 - \beta) \cdot (b + \delta)$ . The maximum amount of data  $D$  that can be sent in one cycle over the link is given by the area below  $\delta$ :

$$D = \delta(1 - \beta)(b + \delta) \quad (3)$$

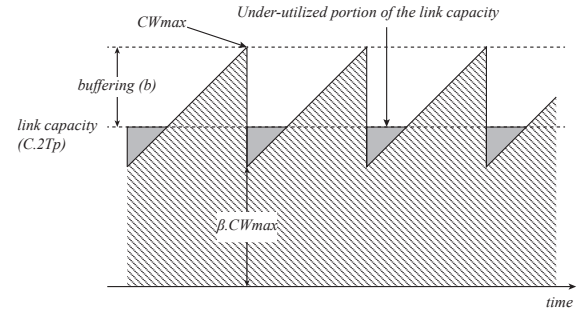


Fig. 1. TCP's sawtooth behaviour.

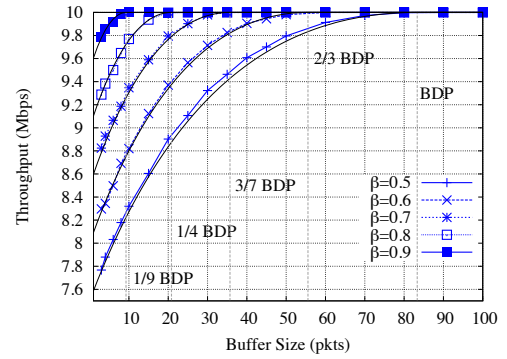


Fig. 2. Throughput (Single NewReno flow @ 10Mbps, RTT=100ms) (model and simulation).

Dark triangles correspond to periods where  $cwnd < \delta$  (i.e., the TCP sender cannot “fill the pipe”). The buffer is empty, the link is under-utilised, and the amount of data *not* sent in a cycle corresponds to the area  $\Delta_{cwnd < \delta}$  of a dark triangle:

$$\Delta_{cwnd < \delta} = \frac{(\delta - \beta(b + \delta))^2}{2} \quad (4)$$

Therefore, we can compute  $U$  as:

$$U = 1 - \frac{\Delta_{cwnd < \delta}}{D} = 1 - \frac{(\delta - \beta(b + \delta))^2}{2\delta(1 - \beta)(b + \delta)} \quad (5)$$

Considering both the cases  $U = 1$  and  $U < 1$ , the throughput, that is  $\psi = C \times U$ , is given by:

$$\psi = \begin{cases} C & \text{if } b \geq \delta \frac{1 - \beta}{\beta}, \\ C \left( 1 - \frac{(\delta - \beta(b + \delta))^2}{2\delta(1 - \beta)(b + \delta)} \right) & \text{otherwise.} \end{cases} \quad (6)$$

For  $b = 0$  and  $\beta = 0.5$ , this yields  $\psi = \frac{3}{4}C$ , in line with the model in [5]. The relation between utilisation, backoff factor and buffer size is illustrated in Figure 2, showing values given by both (6) and simulation results.

It also worth noting that (6) holds for all ranges of bottleneck link capacity ( $C$ ) since the utilised fraction



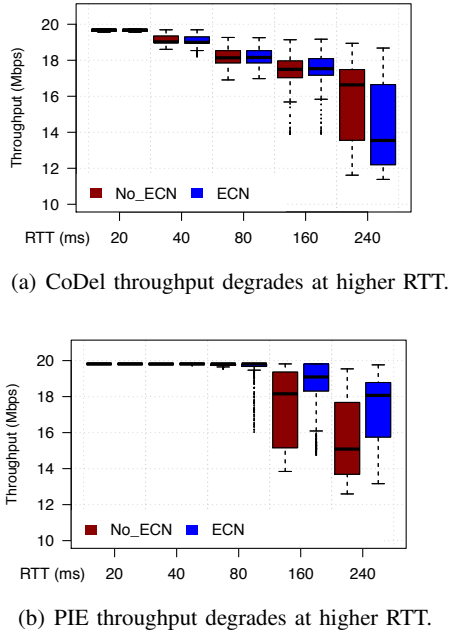


Fig. 3. Performance through CoDel and PIE bottlenecks is significantly degraded over high-RTT paths (real-life test).

of the link capacity ( $U$ ) derived in (5) is independent of  $C$ .

#### B. Using AQM with a large path RTT

AQM mechanisms try to reduce the queuing delay incurred when DropTail buffers are sized according to (1) (i.e.,  $b \geq \delta$  when  $\beta = 0.5$ ). CoDel's  $target\_delay$  and PIE's  $T_{target}$  parameters control how aggressively they mark or drop packets based on the sojourn time. Small values of  $target\_delay$  or  $T_{target}$  do reduce delay but also reduce TCP throughput when the RTT is large [24], [42].

The problem is illustrated by Figure 3, showing the throughput of a single CUBIC flow through a 20 Mbps bottleneck using CoDel or PIE, both with and without ECN, as a function of the path's intrinsic RTT<sup>1</sup>; simulated time was 90 s. Increasing path RTT has a detrimental impact on a CUBIC flow's throughput through both CoDel and PIE bottlenecks<sup>2</sup>. ECN on its own provides limited assistance; simply (naively) replacing drops by marks does not address the problem.

An explanation lies in the way CoDel and PIE both aim for a buffer of 5-20 ms (see § IV-A for the AQM parameters used). For a given bottleneck rate, this target equates to a shallow buffer ( $b$ ) relative to the BDP ( $\delta$ ) of

<sup>1</sup>CUBIC's use of  $\beta = 0.7$  does not alter the key point.

<sup>2</sup>Using DropTail, CUBIC achieves 100% throughput under the same circumstances, albeit with a well-known spike in queuing delay.

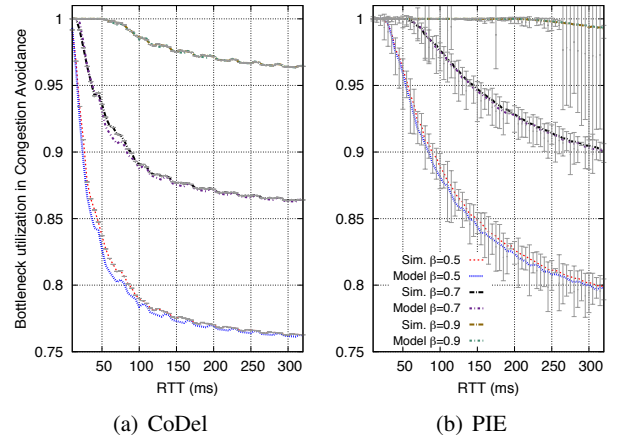


Fig. 4. Model vs. simulation (NewReno @ 20Mbps); error bars not shown for CoDel because fluctuations were negligible.

an entire path. As path RTT increases, the condition in (1) is increasingly violated and utilisation trends below 100%.

Figure 4 compares the median bottleneck utilisation generated both by simulation and Eq. (5) for a TCP NewReno flow in congestion avoidance over an AQM-controlled, 20 Mbps bottleneck with various RTTs and  $\beta$ . Error bars represent standard deviation.

The results are based on the average sending rate over five congestion cycles (saw-teeth) of a simulated flow that lasted 100 cycles. CoDel exhibited negligible variations over multiple runs (too small for error bars), while PIE's probabilistic nature created noticeable variations. At high RTTs with  $\beta = 0.9$  PIE would sometimes trigger multiple consecutive backoffs immediately after slow start (dropping  $cwnd$  to 37% of  $\delta$  when RTT=300 ms).

The model (Eq. (5)) was applied by tracking the queue length at the time of packet loss (i.e., the largest queue length that CoDel and PIE gave to the flow in each cycle) and taking the median of these values as  $b$ .

Figure 4 illustrates that the AQMs seem to roughly behave like a shallow DropTail queue: the utilisation drop with large RTTs is an expected outcome of the small queue granted to the TCP flow, and a larger  $\beta$  is needed to increase utilisation.

#### C. Selecting the right backoff factor

TCP will clearly benefit from larger  $\beta$  when confronted by shallow queues. But we cannot simply raise  $\beta$  without regard to network conditions. A TCP flow risks creating a significant standing queue if it uses high  $\beta$  on a path whose bottleneck queue is large.

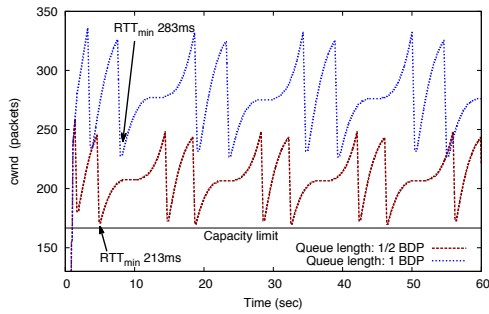


Fig. 5. CUBIC generates a standing queue.

Figure 5 illustrates the issue with Linux CUBIC (using  $\beta = 0.7$  since Linux kernel 2.6.25 in 2008) over two sizes of DropTail queue. With a bottleneck rate of 10 Mbps and base RTT of 200 ms, a queue of  $1/2 \times \text{BDP}$  never completely drains whilst a queue of  $1 \times \text{BDP}$  experiences significant and sustained extra delay.

We therefore need a way for end systems to distinguish between the cases of a potentially large DropTail queue and the often much smaller average queue that is under control of an AQM mechanism.

### III. SOLUTION

We propose “Alternative Backoff with ECN” (ABE), a mechanism in which TCP backs off with a new value  $\beta_{\text{ecn}} > \beta$  instead of the default  $\beta$  in response to an ECN mark, and the back-off in response to loss remains unchanged.

ABE differs from the recommendation in [41], which is that ECN marks should be treated the same way as congestion drops (i.e., same reaction to both at the TCP sender). An ABE sender’s behaviour is based on the assumption that an ECN mark is only generated by an AQM mechanism and therefore the queue is likely to be significantly smaller than in case of loss (if this assumption is wrong, another ECN mark or a packet loss is likely to occur, causing TCP to back off further).

The analysis in § II focused on motivating the use of ABE during congestion avoidance, however, having a different response to marks at the end of slow start can also be beneficial, as we will see next.

#### A. ABE in Slow Start

TCP doubles  $cwnd$  every RTT during the slow start phase and enters the congestion avoidance phase after the first packet loss or mark [41] by halving  $cwnd$  (using

$\beta=0.5$ ).<sup>3</sup> The rationale behind this is to fall back to the last  $cwnd$  that successfully transmitted all its data.

Depending on how a certain multiple of TCP’s initial window is aligned with the path BDP and the queue length, slow start can exceed the available capacity with an overshoot that may be as large as almost one BDP or as small as one packet. While halving the rate is the only way to reliably terminate the overshoot of one flow within one RTT, it can create significant under-utilisation, depending on how large that overshoot was. This problem is more pronounced for short flows such as common web traffic which may terminate with a  $cwnd$  of just above  $0.5 \times \text{BDP}$ , shortly after slow start. Recent trends [2] show a substantial increase in the length of short flows, increasing the probability that they terminate not during slow start but rather shortly after it, during the congestion avoidance phase, making the impact of the choice  $\beta_{\text{ecn}} = \beta$  in slow start more profound (§ IV-D). Also, SPDY and HTTP/2 [10], [43] reuse existing connections for multiple objects, further increasing the probability of web flows to leave slow start.

This issue is depicted in Figure 6. The solid lines illustrate two single NewReno flows, with intrinsic RTTs of 160 ms and 240 ms, when  $\beta_{\text{ecn}} = \beta = 0.5$  is used. The flow with RTT=240 ms suffers from under-utilisation: after slow-start overshoot, a small  $cwnd$  results in a slow increase during congestion avoidance. The flow with RTT=160 ms was luckier after slow start, but then the default  $\beta_{\text{ecn}}$  wastes capacity in congestion avoidance.

Dashed lines in Figure 6 show the same scenarios with  $\beta_{\text{ecn}} = 0.9$ . In both cases,  $cwnd$  is just below the BDP soon after slow start, which prevents the bottleneck from severe under-utilisation, reducing the completion time of short flows such as the ones common in web traffic. However, a large overshoot results in  $cwnd$  being reduced in multiple steps, which increases the latency over multiple RTTs after slow-start (however, the number of these RTTs is bounded by  $\log_{\beta_{\text{ecn}}}(0.5)$ ).

In § IV-D we evaluate the costs versus benefits of using a higher  $\beta_{\text{ecn}}$  at the end of slow start and demonstrate that its gains are significant.

### IV. EVALUATION

The experimental setup for both simulations and real-world tests is described in detail in Appendix A.

<sup>3</sup>This is the case for both NewReno and CUBIC. While the latest implementation of CUBIC includes Hystart [20] (a mechanism that tries to prevent slow start’s overshoot by examining the delays of ACK arrivals), if an overshoot happens it causes  $cwnd$  to be halved.



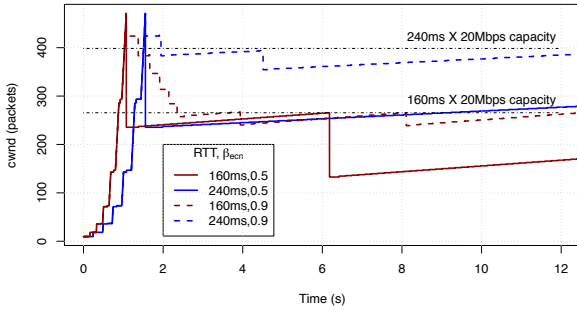


Fig. 6. The effect of overshoot at the end of slow-start for  $\beta_{ecn} = \{0.5, 0.9\}$  @ 20 Mbps (real-life test).

### A. AQM algorithms and parameters

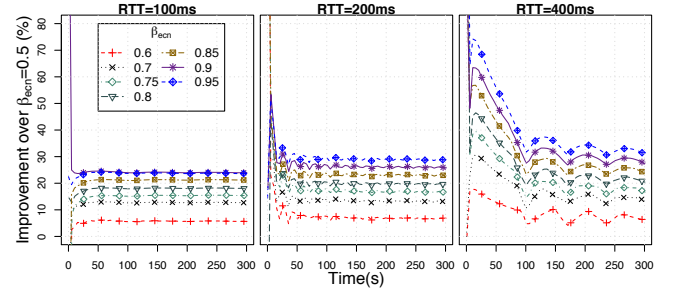
Both CoDel and PIE were used in all experiments. We did not run tests with FQ\_CoDel [21] because it shares some of CoDel's issues, and it may introduce other issues that are beyond the scope of this paper (e.g., problems with classifying flows due to use of IPsec or tunnels such as VPNs).

For CoDel and PIE, in all experiments, we used the code that was shipped with Linux 3.17.4 and the ns-2 code that is publicly available, and applied the Linux default parameters everywhere. CoDel's *interval* and *target* were set to 100 ms and 5 ms, respectively, while PIE's *target* and *tupdate* were set to 20 ms and 30 ms. PIE's *alpha* is 0.125 and *beta* is 1.25. A parameter called *max\_burst* is described in [39], with a default value of 150 ms. This parameter allows bursts of a given size to pass without being affected by PIE's operation, is 100 ms by default in the simulation code and is hard coded as such in the Linux implementation. *bytemode* was turned off in Linux and accordingly *queue\_in\_bytes\_* was disabled in ns-2.

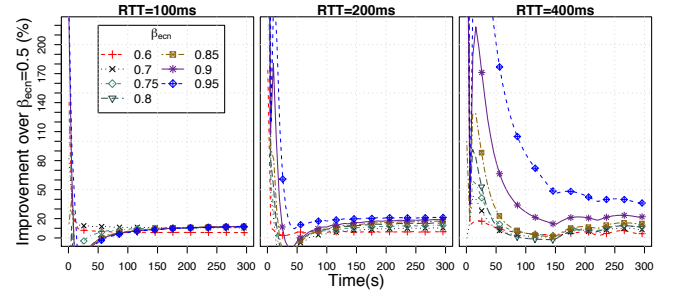
None of the specifications mention the maximum physical queue length – this is not a parameter of the AQM itself. In the Linux *man* pages, *limit* is 1000 packets.

In Linux kernel 3.17.4, packets on ECN-enabled flows are dropped rather than marked when PIE's drop/mark probability exceeds 10%, yet we could find no published literature providing a rationale for such behaviour. One might argue it provides a defense against non-responsive, ECN-enabled flows. However, here we recommend disabling it due to its detrimental impact on well-behaved ECN-enabled flows<sup>4</sup>.

<sup>4</sup>See Appendix B for experiments supporting this choice.



(a) CoDel



(b) PIE

Fig. 7. Improvement over  $\beta_{ecn}=0.5$  (bytes departed) for a single NewReno flow @ 10 Mbps.

### B. Latency vs. throughput for a single flow

Finding the right  $\beta_{ecn}$  value requires investigating a trade-off between latency and throughput. Generally, the larger the multiplication factor ( $\beta$ ) the higher the throughput, but also the latency. However, the ECN congestion signal is a notification of the presence of an AQM mechanism on the bottleneck. The state-of-the-art AQM schemes already mark at low buffering thresholds, limiting the latency increase that can be caused by a larger than standard  $\beta$ .

We investigated this trade-off for different  $\beta_{ecn}$  values for a single long-lived TCP NewReno flow using simulations with CoDel and PIE, and for a set of RTT values (100 ms, 200 ms and 400 ms) as presented in Figures 7 and 8.

Figure 7 shows that significant gains in throughput can be achieved as  $\beta_{ecn}$  increases with both PIE and CoDel, especially during the period right after slow-start and for larger RTTs. Moreover, a higher  $\beta_{ecn}$  improves the throughput of longer transfers compared to  $\beta_{ecn}=0.5$ , especially on large RTT paths—e.g. with an RTT of 400 ms, by 13%, 24% and 31% (CoDel) and 9%, 16% and 37% (PIE) for  $\beta_{ecn}$  values of 0.7, 0.85 and 0.95, respectively.

Figure 8 shows the CDF of queuing delay for the

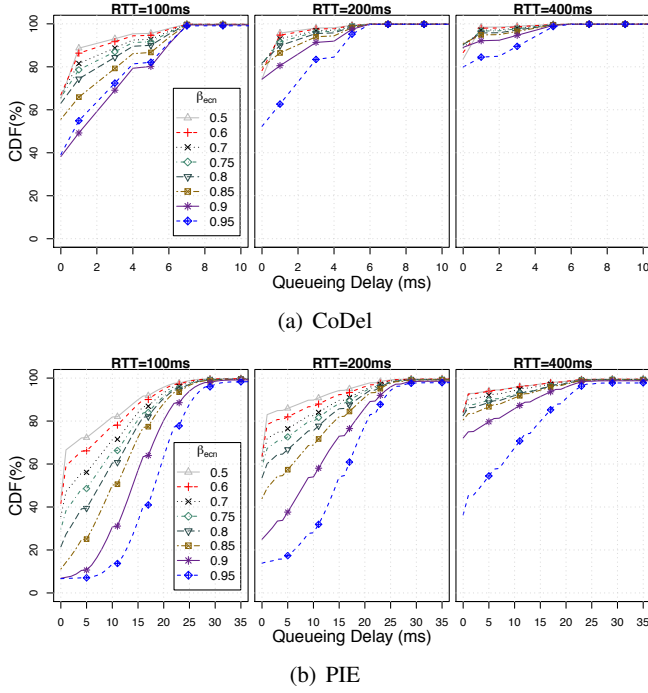


Fig. 8. CDF of queuing delay for a single NewReno flow @ 10 Mbps.

scenario of Figure 7. CoDel is able to keep queuing delay close to its default target delay parameter of 5 ms for all ranges of  $\beta_{ecn}$  (Figure 8(a)). In case of PIE (Figure 8(b)), the latency distribution becomes somewhat more heavy-tailed with increasing  $\beta_{ecn}$ , although the 90% percentile stays below PIE's default target of 20 ms for the range  $0.5 \leq \beta_{ecn} \leq 0.85$ . This is due to the burst-absorbing and random nature of PIE in contrast to the deterministic dropping policy of CoDel [25].

When  $RTT=100$  ms, while CoDel adds 1 ms/4 ms to the median/90th percentile for  $\beta_{ecn}=0.95$  compared to  $\beta_{ecn}=0.5$ , PIE adds 18 ms/11 ms to the median/90th percentile for the same scenario which is slightly significant relative to the intrinsic RTT. However, with  $\beta_{ecn}$  of 0.7 and 0.85, this can be reduced to 2 ms/3 ms and 10 ms/6 ms respectively.

We can deduce that for a NewReno flow, a  $\beta_{ecn}$  value in the range  $0.7 \leq \beta_{ecn} \leq 0.85$  would be an optimal trade-off between latency and throughput.

We also evaluated the above scenarios for a single CUBIC flow as presented in Figure 9 and Figure 10. Similar trends as for NewReno can be observed with CUBIC with regards to the achieved throughput gain for the period after slow-start and short flows for  $0.7 \leq \beta_{ecn}$  although with a less profound gain due to the already

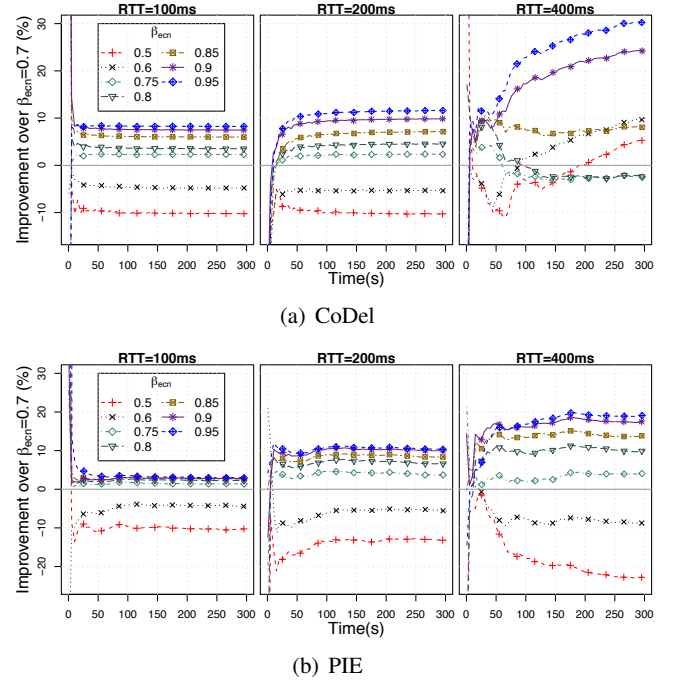


Fig. 9. Improvement over  $\beta_{ecn}=0.7$  (bytes departed) for a single CUBIC flow @ 10 Mbps.

aggressive nature of CUBIC's additive increase and  $\beta_{ecn}=0.7$  (Figure 9). The trend for the queuing delay is also similar to that of NewReno (Figure 10).

In terms of long-term throughput the exception is only for the scenario of CoDel on a path with large RTT where, surprisingly, an improvement of 5% and 10% can be gained by choosing  $\beta_{ecn}$  of 0.5 and 0.6 instead of the default 0.7 and using any value of  $0.75 \leq \beta_{ecn} \leq 0.8$  will decrease the long-term throughput. However using a higher than default  $\beta_{ecn}$  will always lead to a better gain in throughput for all other RTT scenarios with CoDel (Figure 9(a)). In case of PIE, the trend in all scenarios is consistent with NewReno gaining 14% and 19% over the default  $\beta_{ecn}$  for 0.85 and 0.95 values respectively (Figure 9(b)).

Choosing these  $\beta_{ecn}$  values comes at the cost of an increase in the median/90th percentile of queuing delay equal to 6 ms/3 ms (PIE,  $\beta_{ecn}=0.85$ ) and 12 ms/5 ms (PIE,  $\beta_{ecn}=0.95$ ) and 1 ms/1 ms (CoDel,  $\beta_{ecn}=0.85$ ) and 3 ms/2 ms (CoDel,  $\beta_{ecn}=0.95$ ) when  $RTT=100$  ms (Figure 10), which is negligible compared to the gain in throughput.

We can deduce that for a CUBIC flow, a  $\beta_{ecn}$  value in the range  $0.85 \leq \beta_{ecn} \leq 0.95$  would be an optimal trade-off between latency and throughput.

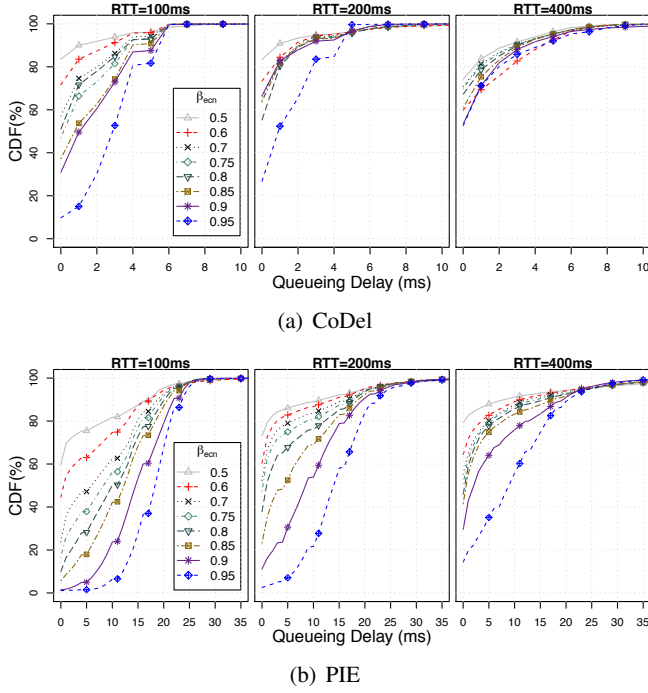


Fig. 10. CDF of queuing delay for a single CUBIC flow on @ 10 Mbps.

### C. Bulk transfers

Figure 11(a) illustrates how increasing  $\beta_{ecn}$  (0.5, 0.6, 0.7 and 0.8) allows a FreeBSD NewReno flow to recover some of the throughput otherwise lost when running through a CoDel bottleneck at a large RTT. At the same time, Figure 11(b) shows that the resulting RTT is basically unaffected.

We also investigated how  $\beta_{ecn}$  affects the time that similar flows need to converge to a fair rate allocation. Intuitively, one might believe that using a larger  $\beta_{ecn}$  increases the time needed by a new flow to reach its fair share because already converged flows would give up less of their bandwidth when they hit the capacity limit. This is not necessarily correct. Simplifying, if we call the rates (or congestion windows) of two flows  $X$  and  $Y$ , perfect fairness is achieved if their ratio  $X/Y$  is 1. Then, if a congestion event affects both flows at the same time, this ratio becomes  $\frac{X}{\beta_{ecn}} * \frac{\beta_{ecn}}{Y}$ , rendering the value of  $\beta_{ecn}$  irrelevant for convergence to fairness. The ratio will obviously become different if only one flow is affected – but by this argument, convergence to fairness should mainly depend on how often each flow is affected, and not on how often congestion happens in total. In the absence of a per-flow scheduler like FQ\_CoDel, this depends on how queuing dynamics

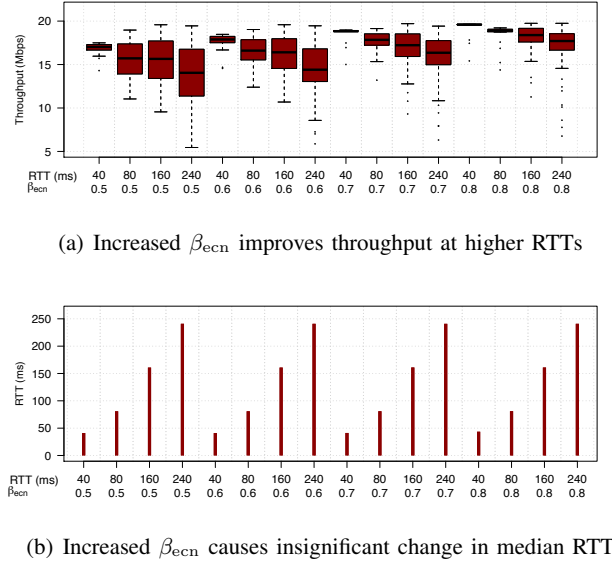


Fig. 11. Increasing  $\beta_{ecn}$  (single NewReno flow over CoDel bottleneck @ 20 Mbps).

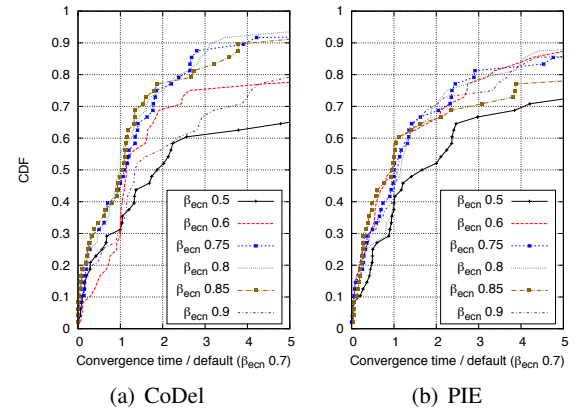


Fig. 12. Ratio of convergence time to a test with default  $\beta_{ecn}$ , CUBIC.

affect flow synchronization, which may itself not depend heavily on the value of  $\beta_{ecn}$  used by all flows.

We measured convergence by calculating Jain's Fairness Index [23] of the cumulative number of bytes transferred per flow from the time a new flow entered, and noting the time it took until this index reached a threshold (which, for the graphs presented, was set to 0.95). We tested using two, four and ten flows with equal RTTs, starting with a delay of 30 seconds in between, and did not find any consistent improvement or disadvantage from changing  $\beta_{ecn}$ . Therefore we opted to explore the parameter space with a set of more controlled experiments of only two flows, using different capacity

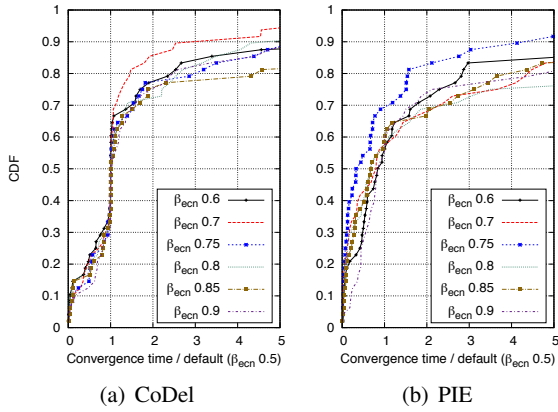


Fig. 13. Ratio of convergence time to a test with default  $\beta_{ecn}$ , NewReno.

and RTT values for every test (all combinations of 1, 5, 10, 20, 40, 100 Mbps and 10, 20, 40, 60, 80, 160, 240, 320 ms). Figures 12 and 13 show the convergence time from 485 90-second two-flow experiments (97 for every beta value, including 0.5; we also tested values 0.55, 0.65, 0.75, 0.85 and 0.95 with similar results and omit these lines for clarity) each for NewReno and CUBIC. The simulation time was long enough to ensure convergence in all experiments.

Convergence time is shown as the ratio of the time taken in an experiment with similar conditions but the default behaviour of CUBIC ( $\beta_{ecn} = 0.7$ ) and NewReno ( $\beta_{ecn} = 0.5$ ), respectively, i.e. a value of 2 means that it took twice as long for a flow to converge than the default case. The distribution is heavy-tailed because there were some extreme outliers – not only very large times for the depicted  $\beta_{ecn}$  values, but also very small ones for  $\beta_{ecn} = 0.5$ . Some were caused by the initially explained scheduling (one flow was simply “unlucky” because it was perpetually affected by a congestion mark). In other cases, the default flow saw slow start ending at just the right value, creating an outlier for all non-default values of  $\beta_{ecn}$ .

Unsurprisingly, a significantly *lower*  $\beta_{ecn}$  than CUBIC’s default consistently increases its convergence time. All  $\beta_{ecn}$  values can sometimes cause even faster convergence than the default, but for CUBIC no value of  $\beta_{ecn}$  could significantly improve it (generally, less than half of the cases converged faster than the default case). This is somewhat different with NewReno, where at least with PIE the value  $\beta_{ecn} = 0.75$  seemed to have consistently improved convergence.

The point of this evaluation was to see if convergence time would be significantly (and consistently) harmed by

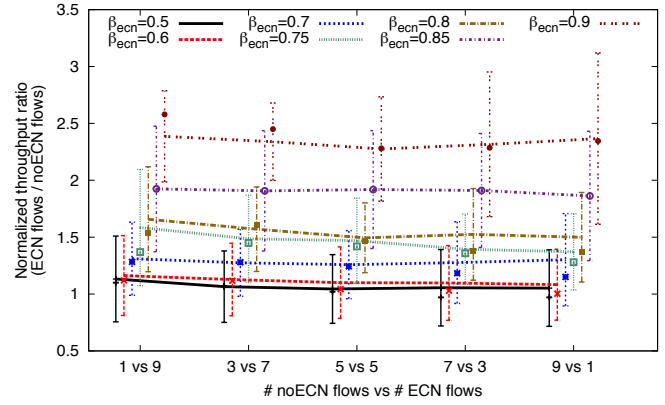


Fig. 14. Normalized throughput ratio between two groups of 10 flows @ 10 Mbps with a CoDel queue using several RTTs. Lines pass through the arithmetic mean, dots indicate the median.

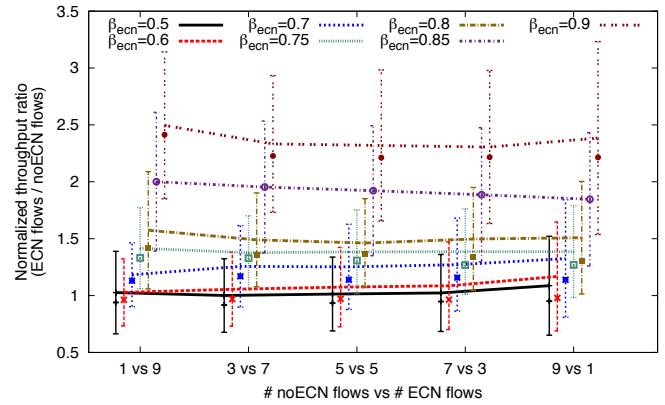


Fig. 15. Normalized throughput ratio between two groups of 10 flows @ 10 Mbps with a PIE queue using several RTTs. Lines pass through the arithmetic mean, dots indicate the median.

changing  $\beta_{ecn}$ . Our results let us conclude that this is not the case.

Next, we investigated the long-term impact that flows with a large  $\beta_{ecn}$  value can have on legacy traffic that does not support ECN. To study the worst case, we tested CUBIC flows with various  $\beta_{ecn}$  values in competition with NewReno flows that do not support ECN. Figures 14 and 15 show the throughput ratio for various combinations of 10 flows across a 10 Mbps bottleneck link using CoDel and PIE. The plotted value was normalized by weighting it with the number of flows in each group. Each test lasted five minutes, of which we cut the first two seconds to remove slow start. We carried out every test 3 times, using RTTs of 20, 80, 160 and 320 ms; with the largest RTT, this duration included 133 TCP



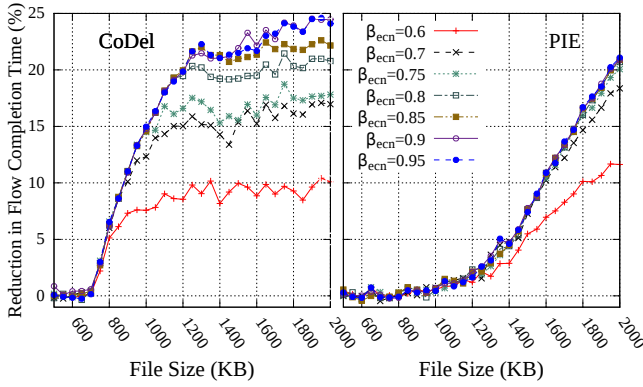


Fig. 16. Percentage reduction in FCT (compared to using  $\beta_{ecn} = 0.5$ ) with different flow sizes over a 10 Mbps link with 100 ms base RTT

sawteeth.

The difference between CoDel and PIE is marginal, and generally a larger  $\beta_{ecn}$  value increased the throughput ratio, as could be expected. Our goal was to assess the potential “danger” of upgrading  $\beta_{ecn}$  on the Internet. The increase in the throughput ratio is limited to a factor of 3 for the already quite extreme value of  $\beta_{ecn}=0.9$ . The expected throughput difference is about double for  $\beta_{ecn}=0.85$ , and smaller values seem to be even closer together. This is CUBIC, where  $\beta_{ecn}=0.8$  has been used on the Internet for several years and  $\beta_{ecn}=0.7$  is now the default value in the Linux kernel.

#### D. Short flows

Much of the Internet’s traffic consists of short flows (i.e. web traffic, e-mails etc). Recent trends show that the average web page size is increasing but most web flows still have a size of less than 2 MB [2]. So flows that do not terminate during slow-start may terminate shortly after, and the value of  $\beta_{ecn}$  at the end of slow-start will have an impact on the load time of the corresponding pages.

We simulated one TCP NewReno (Initial Window=3) flow over a 10 Mbps bottleneck with different flow sizes ranging from 500 KB to 2000 KB and a base RTT of 100 ms. Figure 16 shows the percentage reduction in Flow Completion Time (FCT) for different  $\beta_{ecn}$  values compared to using  $\beta_{ecn}=0.5$ . TCP flows started randomly within the first second of simulation time and the results were averaged over 1000 different seed values.

According to Figure 16, there is a reduction in FCT when the file size is larger than 700 KB for both PIE and CoDel. A flow with a size of less than 700 KB ends

during or just after slow-start (for the selected link speed and RTT). Therefore changing  $\beta_{ecn}$  has no impact on the performance of very small files.

However, when a file is larger than 700 KB it may experience one or more cwnd reductions after slow-start overshoot. These consecutive reductions (up to one per RTT) slow down the sending rate to match the link speed. In Figure 17, PIE (target delay=20 ms) reduces the cwnd four times (from 448 packets to 29 packets) with  $\beta_{ecn}=0.5$  (the cwnd is reduced suddenly as opposed to a smooth reduction which is shown in the figure with ns-2). This quickly drains the standing queue developed during the exponential growth. With  $\beta_{ecn}=0.9$ , a larger standing queue is created, and it takes only three reductions (up to 324 packets) until the file transfer is completed. When  $\beta_{ecn}$  is changed from 0.5 to 0.9 with CoDel (target delay=5 ms), it results a number of cwnd reductions from 2 (from 190 packets to 49 packets) to 8 (up to 84 packets).

In both schemes  $\beta_{ecn}=0.5$  brings the cwnd much below the target link rate (the BDP of 83 packets), resulting in poor link utilisation with 100 ms RTT. This is particularly significant with large RTTs where the cwnd growth rate becomes very slow. Small reductions with a large  $\beta_{ecn}$  bring the cwnd more precisely to the bottleneck rate, resulting in better bottleneck utilisation. This yields a better FCT for short flows with a moderate file size.

In the case of Figure 17, it reduces FCT by 300 ms with PIE and by 570 ms with CoDel. However, the increased number of ECN-marked packets with  $\beta_{ecn}=0.9$  due to a larger standing queue does not have a negative impact on FCT when used with an ECN-capable transport.

The improvement in FCT with CoDel is greater for file sizes below 2000 KB compared to PIE. CoDel overshoots less than PIE and marks earlier, creating a smaller standing queue. As a result, a large  $\beta_{ecn}$  causes less improvement in PIE than in CoDel.

We simulated the same scenario for different RTTs (100 ms, 200 ms and 400 ms) keeping the file size at 2000 KB. In Figure 18, PIE shows a better performance for the whole range of RTTs. PIE also randomises packet drops, making FCT improvement consistent. The improvement of PIE saturates at  $\beta_{ecn}$  near 0.75.

With CoDel, the performance is not consistent for all RTTs if flow is terminated after the slow-start. It shows the same number of cwnd reductions for all different seed values due to the deterministic behaviour of packet drops with a single flow. Because one reduction with  $\beta_{ecn}=0.5$  is more significant than with  $\beta_{ecn}=0.9$ , results

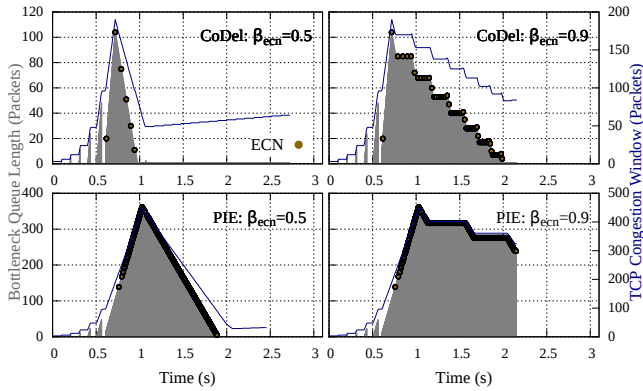


Fig. 17. Bottleneck queue and cwnd behaviour with one TCP NewReno flow. A large  $\beta_{ecn}$  results in precise cwnd adaptation (file size 2000 KB)

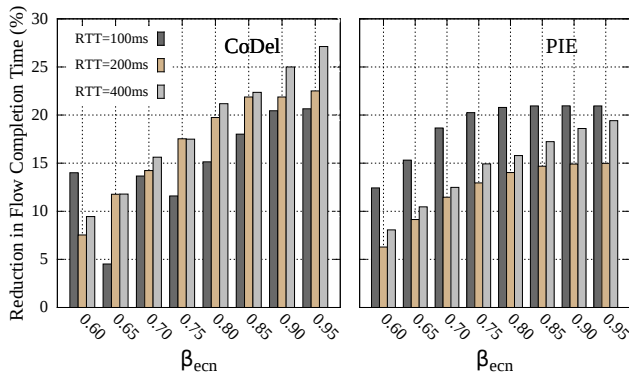


Fig. 18. FCT reduction performance with three (100 ms, 200 ms, 400 ms) different RTTs (file size 2000 KB)

are not consistent with CoDel depending on how many reductions have occurred and depending on the mismatch in cwnd under each RTT. Therefore to minimise this impact we randomized the RTT by  $\pm 10$  ms for each selected RTT in our simulations.

## V. RELATED WORK

Since its inception, ECN has attracted much interest in the research community. One obvious question that papers have tried to answer was: can we get “more bang for the bits”? Given that there are two ECN bits in the IP header, they can be re-defined to distinguish between multiple levels of congestion [15], [47], or used as defined, but marked with a different rule that must be known to the end hosts [45], [40]. LT-TCP [18] proposed using ECN to distinguish between random packet losses on wireless links and congestion; NF-TCP [7] suggests

using it as a means to detect congestion early and back off more aggressively than standard TCP, thereby reducing the delay caused for competing applications. The potential applications are broad – yet, none of these methods were designed to interoperate with the currently deployed mix of standards-compliant ECN-capable and -incapable routers and end hosts such that it could be gradually introduced in the Internet.

“Congestion Exposure” (ConEx) [13], which is based on Re-ECN [12], uses traffic shapers to hold users accountable for the congestion they cause on the whole path (as opposed to giving most capacity to the host running the most aggressive congestion control). ConEx is gradually deployable, but needs significant changes to the infrastructure (hosts must be modified to give information to the network, traffic shapers must be deployed).

Some schemes were defined for use within a pre-configured operator domain. In “Pre-Congestion Notification” (PCN) [33], the ECN bits are used to inform DiffServ ingress routers of incipient congestion to better support admission control. Data Center TCP (DCTCP) [4] is another proposal to update TCP’s ECN response. This has been shown to offer measurable benefits when network devices update their ECN marking behaviour. Only the software in hosts needs to change for DCTCP—deployment in routers can be achieved via an unusual configuration of RED to operate on the instantaneous queue length. However, in contrast to our proposal, this is only currently considered safe when all network devices along a path use the updated marking rules, and hence is only enabled by default for network paths with an RTT less than 10 ms [11], the primary reason why this has only been specified for Data Center usage.

A recent IETF presentation [28] discussed the possibility of gradually introducing a DCTCP-like scheme in the Internet; this is very close in spirit to the work presented in this paper. However, just like DCTCP, it requires a change to the receiver to more precisely feed back the number of ECN marks at a greater precision than today (i.e., more than one signal per RTT). Incorporating such feedback in TCP is ongoing IETF work [27]. Because marking the instantaneous queue length provides faster feedback, such an approach should theoretically be expected to perform better than the marking that we propose, provided it can also be combined with a sender behaviour that strikes the correct balance between efficiency across updated routers and compatibility across old routers. We regard both this paper and [28] as necessary investigations of the problem

space before a decision can be made regarding an update of the ECN standard.

Compared to DCTCP, our response to ECN is still based on a reduction each RTT, rather than per marked segment, but our method releases the network from the deployment pre-requisites of DCTCP. However, we note that a network device configured for DCTCP can also provide appropriate marking for our method, as can other router marking policies. This lack of sensitivity to marking removes a key obstacle to ECN deployment.

The idea of using values of  $\beta \neq 0.5$  is at the basis of proposals for improving performance of long-lived TCP flows in high-speed networks. CUBIC is one example of such congestion controllers tailored to high-speed links, but other similar schemes can be found in the literature. For instance, H-TCP [30] uses  $\beta \in (0.5, 0.8)$ , and the value of  $\beta$  is adapted as a function of measured minimum and maximum RTTs. In a similar vein, after loss is experienced TCP Westwood [14] sets  $cwnd$  to the product of the estimated available capacity and the lowest observed RTT—thus, the equivalent  $\beta$  is variable and dependent on network conditions. High-Speed TCP (HSTCP) [16] adapts  $\beta$  in the range  $(0.5, 1)$ , with  $\beta$  taking higher values for larger  $cwnd$  above a threshold. In these proposals, the rationale behind the choice of a larger  $\beta$  is to allow for a faster recovery of  $cwnd$  after loss<sup>5</sup>; something that, with “standard” congestion control, can take many RTTs over paths with large BDP.

## VI. CONCLUDING REMARKS

This paper proposes and motivates Alternative Backoff with ECN (ABE), a simple change in a TCP sender’s reaction to observing ECN congestion marks. We show that ABE can bring important performance improvements, with a low cost in terms of implementation effort. ABE achieves this performance using the ECN marking specified in RFC 3168 for routers/middleboxes and the TCP receiver. It also defaults to a conservative behaviour whenever ECN is not supported along the path, ensuring it is incrementally deployable.

As our results in § IV show, the choice of  $\beta_{ecn}$  is important but not overly critical, in the sense that ABE seems robust and offers performance improvements across a range of values of  $\beta_{ecn}$ . Setting  $\beta_{ecn} \in [0.7, 0.85]$  for NewReno and  $\beta_{ecn} \approx 0.85$  for CUBIC seems to provide reasonable trade-offs between latency, throughput and fairness across a wide range of scenarios.

<sup>5</sup>Adaptation of the window-increase parameter  $\alpha$  is also used by these mechanisms for such purpose.

We expect methods like ABE to encourage usage of ECN, and as this usage increases, we believe the time will become ripe to revisit proposals for alternative ECN marking, along the lines of Section V. When use becomes widespread, router/middlebox manufacturers will have an incentive to implement these improved ECN specifications to further optimise performance. Hosts using ABE will then also be able to update their  $\beta_{ecn}$ .

Our study explored two well-known auto-tuning AQM methods with their standard pre-set parameter values (§ IV-A). To limit the problem space, we did not investigate the impact of changing the CoDel and PIE parameters. We also used the same  $\beta$  value in both slow start and congestion avoidance. Intuitively, one may think that using more aggressive AQM parameters for marking should advocate a higher  $\beta$  value. However, we expect there is a limit to how aggressively an AQM scheme can react, before even packets in a natural packet burst are punished by ECN marks or drop. Such questions concerning AQM tuning should be investigated in future work.

ECN-enabled routers need to protect themselves from overload by unresponsive traffic. RFC 3168 recommended routers to avoid high levels of ECN marking, assuming this was an indication of congestion collapse, and therefore encouraged drop under these conditions, as an appropriate response to overload. However, DCTCP and other modern AQM schemes use a low marking threshold, where this assumption becomes invalid, and can significantly impact performance (§ B). We therefore agree with the recommendation in [8], which encourages new research into more appropriate overload protection methods.

The use of ECN must be initiated by the TCP client. This makes ABE immediately applicable for use cases where a host updated with ABE initiates a connection and then transmits data, such as uploads to the Cloud, Web 2.0 applications, etc. In use cases where a client initiates a connection for data sent by a server, such as web surfing, ABE requires the server to be updated. An example of the expected performance with ECN, but without ABE is shown in Figure 3. Given such results and the increasing ability of routers to correctly pass ECN-enabled packets, no significant disadvantage is to be expected in this case, and the impact of ABE will increase as servers introduce support for it.

In conclusion, results have been presented from experiments, theory and simulation that show real benefit can be derived from updating TCP’s ECN response, and we assert that this can provide a significant performance

incentive towards greater use of the ECN across the Internet.

## VII. ACKNOWLEDGEMENTS

This work was partly funded by the European Community under its 7th Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the authors.

## APPENDIX

### A. Simulations

All simulations presented in this paper are based on ns-2.35, using “Linux TCP” updated to use the pluggable congestion controls shipped in Linux kernel 3.17.4.

### B. Experimental testbed

The results in Figures 3, 5, 6, 11, 12, 13, 14 and 15 were created using TEACUP [48] driving a physical testbed of source and destination hosts in two separate subnets either side of a PC-based router (the classic dumbbell topology).

The 64-bit end hosts<sup>6</sup> are booted into FreeBSD 10.1-RELEASE or openSUSE 13.2 for NewReno and CUBIC trials respectively. Dynamic TCP state is logged with SIFTR under FreeBSD (event driven) and web10g under Linux (polled). We use iperf to source and sink traffic, and use tcpdump to capture the traffic actually seen ‘on the wire’ at each host. Modified FreeBSD NewReno and Linux CUBIC congestion control modules were implemented so that loss-triggered  $\beta$  and ECN-triggered  $\beta_{ecn}$  could be separately set and altered via sysctls.

The two subnets are linked by a 64-bit software router<sup>7</sup> based on openSUSE 13.2. The router provides a configurable rate-shaping bottleneck with CoDel or PIE AQM as required. The router also separately emulates configurable artificial delay in each direction.

Shaping, AQM and delay/loss emulation is on the router’s egress NIC in each direction. The hierarchical token bucket (HTB) queuing discipline is used for rate-shaping, with the desired AQM queuing discipline (e.g. pfifo, codel) as leaf node. After the bottleneck stage, additional (configurable) fixed delay is emulated with

netem. We loop traffic through pseudo interfaces (intermediate function block, or IFB, devices) to cleanly separate the rate-shaping/AQM from the delay/loss instantiated by netem. This ensures the bottleneck buffering occurs under the control of the selected AQM.

Linux kernel 3.17.4 is used on all hosts and the router. To get high timer precision for queuing and delay emulation the router’s kernel is patched and recompiled to tick at 10000Hz.

Each PC has an additional NIC attached to an independent control network. A separate ‘command and control’ host uses ssh over this control network to coordinate and launch repeated trials. This includes configuration of source and destination hosts (enabling or disabling ECN, disabling of hardware segmentation offloading, setting specific TCP congestion control algorithms and  $\beta_{ecn}$ , etc), configuration of the bottleneck (rates, path delays, AQM, and ECN on or off), initiation of trials (launching one or more instances of iperf at particular times with particular iperf parameters) and subsequent data retrieval from participating machines.

A key issue is that PIE’s average drop/mark probability can be driven high for reasons that do not indicate a high level of congestion or misbehavior of a non-responsive TCP flow in the short term. We explore this with a simulation of a PIE bottleneck carrying 20 long-lived TCP flows running for 300 s (all started within the first second of simulation time) and a short flow starting after 50 s with different values of  $\beta_{ecn}$ .

We turned off PIE’s default drop behaviour for Figure 19(a), which shows that PIE’s drop/mark probability can frequently rise significantly above 10% for  $\beta_{ecn} > 0.7$  flows. We further observed that PIE’s drop/mark probability would even rise above 10% for  $\beta_{ecn} = 0.5$  flows during normal slow-start. Yet queuing delays remain bounded, as expected of a PIE bottleneck.

Figure 19(b) shows the same experiments with default behaviour re-enabled. PIE’s dropping probability is controlled under 10% but the queuing delay is increased with larger  $\beta_{ecn}$ .

We experience a large number of CE packets with large  $\beta_{ecn}$  in the case of Figure 19(a) and counting them in the PIE algorithm increases the marking probability. Hence PIE creates a low queuing delay with responsive TCP flows. In the case of Figure 19(b), large  $\beta_{ecn}$  results in dropping more packets and controlling the probability. Use of large  $\beta_{ecn}$  with less than 10% marking probability increases queuing delay.

It seems clear that a drop/mark probability of 10% is far too low a threshold to begin dropping ECN-enabled

<sup>6</sup>HP Compaq dc7800, 4GB RAM, 2.33GHz Intel® Core™2 Duo CPU, Intel 82574L Gigabit NIC for test traffic and 82566DM-2 Gigabit NIC for control traffic

<sup>7</sup>Supermicro X8STi, 4GB RAM, 2.80GHz Intel® Core™i7 CPU, 2 x Intel 82576 Gigabit NICs for test traffic and a 82574L Gigabit NIC for control traffic



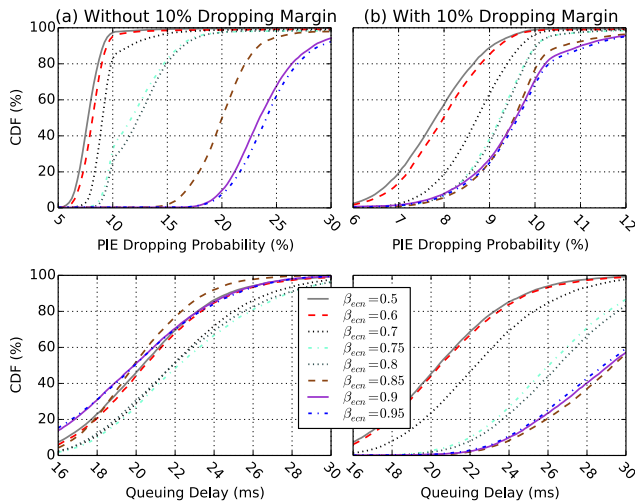


Fig. 19. PIE without and with 10% dropping threshold (20 long-lived TCP flows).

packets. It is detrimental to short flows (where slow start is a significant portion of a flow's overall lifetime) and to flows using higher  $\beta_{ecn}$  (such as flows originating from ABE-enabled senders), and appears to provide little benefit. Therefore, for the rest of our PIE experiments we disabled this default behaviour.

## REFERENCES

- [1] CeroWrt Project. <http://www.bufferbloat.net/projects/cerowrt>.
- [2] HTTP Archive. <http://httparchive.org/trends.php>.
- [3] R. Adams. Active Queue Management: A Survey. *IEEE Communications Surveys and Tutorials*, 15(3):1425–1476, 2013.
- [4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *Proc. ACM SIGCOMM*, pages 63–74, New Delhi, India, 2010.
- [5] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic. Parallel TCP Sockets: Simple Model, Throughput and Validation. In *Proc. IEEE INFOCOM*, pages 1–12, Apr 2006.
- [6] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing Router Buffers. In *Proc. ACM SIGCOMM*, pages 281–292, Portland (OR), USA, Sep 2004.
- [7] M. Arumathurai, X. Fu, and K. Ramakrishnan. NF-TCP: Network Friendly TCP. In *17th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN)*, pages 1–6, 2010.
- [8] F. Baker (ed.) and G. Fairhurst (ed.). IETF recommendations regarding Active Queue Management. Internet Draft draft-ietf-aqm-recommendation, work in progress, Jan 2015.
- [9] S. Bauer, R. Beverly, and A. Berger. Measuring the State of ECN Readiness in Servers, Clients, and Routers. In *Proc. ACM IMC*, pages 171–180, 2011.
- [10] M. Belshe and R. Peon. SPDY Protocol. Internet Draft draft-mbelshe-httpbis-spdy, work in progress, Feb 2012.
- [11] S. Bensley, L. Eggert, and D. Thaler. Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters. Internet-Draft draft-bensley-tcpm-dctcp, Feb 2014.
- [12] B. Briscoe, A. Jacquet, C. D. Cairano-Gilfedder, A. Salvatori, A. Soppera, and M. Koyabe. Policing Congestion Response in an Internetwork Using Re-Feedback. *Proc. ACM SIGCOMM, Computer Communication Review*, 35(4):277–288, Aug 2005.
- [13] B. Briscoe, R. Woundy, and A. Cooper. Congestion Exposure (ConEx) Concepts and Use Cases. RFC 6789 (Informational), Dec 2012.
- [14] C. Casetti, M. Gerla, S. Mascolo, M. Sanadidi, and R. Wang. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. In *Proceedings of ACM MOBICOM*, pages 287–297, Rome, Jul 2001.
- [15] A. Dursesi, L. Barolli, R. Jain, and M. Takizawa. Congestion Control Using Multilevel Explicit Congestion Notification. *Journal of Information Processing Society of Japan*, 48(2):514–526, Feb 2007.
- [16] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), Dec 2003.
- [17] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, Aug 1993.
- [18] B. Ganguly, B. Holzbauer, K. Kar, and K. Battle. Loss-Tolerant TCP (LT-TCP): Implementation and Experimental Evaluation. In *Military Communications Conference - MILCOM*, pages 1–6, 2012.
- [19] J. Gettys and K. Nichols. Bufferbloat: Dark Buffers in the Internet. *Queue*, 9(11):40:40–40:54, Nov 2011.
- [20] S. Ha and I. Rhee. Taming the Elephants: New TCP Slow Start. *Computer Networks*, 55(9):2092–2110, Jun 2011.
- [21] T. Hoeiland-Joergensen, P. McKenney, D. Täht, J. Gettys, and E. Dumazet. FlowQueue-Codel. Internet Draft draft-ietf-aqm-fq-codel, work in progress, Dec 2014.
- [22] V. Jacobson. Congestion Avoidance and Control. In *Symposium Proceedings on Communications Architectures and Protocols, SIGCOMM*, pages 314–329, New York, NY, USA, 1988. ACM.
- [23] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems. Technical report TR-301, DEC Research, Sep 1984.
- [24] N. Khademi, D. Ros, and M. Welzl. The New AQM Kids on the Block: Much Ado about Nothing? Technical Report 434, University of Oslo, Dept. of Informatics, Oct 2013.
- [25] N. Khademi, D. Ros, and M. Welzl. The New AQM Kids on the Block: An Experimental Evaluation of CoDel and PIE. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 85–90, April 2014.
- [26] R. Kohavi and R. Longbotham. Online experiments: Lessons learned. *IEEE Computer Magazine*, 40(9):103–105, 2007.
- [27] M. Kuehlewind and R. Scheffenecker. Problem Statement and Requirements for a More Accurate ECN Feedback. Internet-Draft (work in progress) draft-ietf-tcpm-accecn-reqs-04, Oct 2013.
- [28] M. Kuehlewind, D. Wagner, J. M. R. Espinosa, and B. Briscoe. Immediate ECN. Presentation at the 88th IETF meeting.
- [29] M. Kwon and S. Fahmy. TCP Increase/Decrease Behavior with Explicit Congestion Notification (ECN). In *IEEE ICC*, New York, New York, USA, May 2002.
- [30] D. Leith and R. Shorten. H-TCP: TCP for High-speed and Long-distance Networks. In *Proceedings of PFLDnet 2004*, Argonne (IL), USA, Feb 2004.
- [31] J. Loveless, S. Stoikov, and R. Waeber. Online Algorithms in High-frequency Trading. *Commun. ACM*, 56(10):50–56, Oct 2013.

- [32] A. Medina, M. Allman, and S. Floyd. Measuring the Evolution of Transport Protocols in the Internet. *SIGCOMM Computer Communication Review*, 35(2):37–52, Apr 2005.
- [33] M. Menth, B. Briscoe, and T. Tsou. Precongestion Notification: New QoS Support for Differentiated Services IP Networks. *Communications Magazine, IEEE*, 50(3):94–103, 2012.
- [34] K. Nichols and V. Jacobson. Controlling Queue Delay. *Queue*, 10(5):20:20–20:34, May 2012.
- [35] K. Nichols and V. Jacobson. Controlled Delay Active Queue Management. Internet-Draft draft-nichols-tsvwg-codel-01.txt, Feb 2013.
- [36] J. Padhye and S. Floyd. Identifying the TCP Behavior of Web Servers. In *ACM SIGCOMM*, 2000.
- [37] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem. In *HPSR*, pages 148–155. IEEE, 2013.
- [38] R. Pang. PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem. Internet-Draft draft-pan-tsvwg-pie.txt, Jun 2013.
- [39] R. Pang, P. Natarajan, F. Baker, B. VerSteeg, M. S. Prabhu, C. Piglione, V. Subramanian, and G. White. PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem. Internet-Draft draft-ietf-aqm-pie-00, Oct 2014.
- [40] I. A. Qazi, L. L. H. Andrew, and T. Znati. Congestion Control with Multipacket Feedback. *Networking, IEEE/ACM Transactions on*, PP(99):1, 2012.
- [41] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), Sep 2001. Updated by RFCs 4301, 6040.
- [42] J. Schwardmann, D. Wagner, and M. Kühlewind. Evaluation of ARED, CoDel and PIE. *20th Eunice Open European Summer School and Conference*, 2014.
- [43] B. Thomas, R. Jurdak, and I. Atkinson. SPDYing up the Web. *Commun. of ACM*, 55(12):64–73, Dec 2012.
- [44] B. Trammell, M. Kühlewind, D. Boppert, I. Learmonth, G. Fairhurst, and R. Scheffenegger. Enabling Internet-wide Deployment of Explicit Congestion Notification. In *Proceedings of the 2015 Passive and Active Measurement Conference*, New York, 03/2015 2015.
- [45] N. Vasic, S. Kuntimaddi, and D. Kostic. One Bit is Enough: A Framework for Deploying Explicit Feedback Congestion Control Protocols. In *First International Communication Systems and Networks and Workshops (COMSNETS)*, pages 1–9, 2009.
- [46] M. Welzl and G. Fairhurst. The Benefits and Pitfalls of Using Explicit Congestion Notification (ECN). Internet-Draft draft-ietf-aqm-ecn-benefits, Oct 2014.
- [47] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman. One More Bit is Enough. *IEEE/ACM Trans. Netw.*, 16(6):1281–1294, Dec 2008.
- [48] S. Zander and G. Armitage. TEACUP v1.0 - A System for Automated TCP Testbed Experiments. CAIA Technical Report, <http://caia.swin.edu.au/reports/150529A/CAIA-TR-150529A.pdf>, 29 May 2015.

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: December 25, 2015

G. Fairhurst  
University of Aberdeen  
M. Welzl  
University of Oslo  
June 23, 2015

The Benefits of using Explicit Congestion Notification (ECN)  
draft-ietf-aqm-ecn-benefits-05

Abstract

The goal of this document is to describe the potential benefits when applications use a transport that enables Explicit Congestion Notification (ECN). The document outlines the principal gains in terms of increased throughput, reduced delay and other benefits when ECN is used over a network path that includes equipment that supports ECN-marking. It also discusses challenges for successful deployment of ECN. It does not propose new algorithms to use ECN, nor does it describe the details of implementation of ECN in endpoint devices (Internet hosts), routers or other network devices.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 25, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4.e](#) of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1.	Introduction . . . . .	2
1.1.	Terminology . . . . .	3
2.	Benefit of using ECN to avoid Congestion Loss . . . . .	4
2.1.	Improved Throughput . . . . .	4
2.2.	Reduced Head-of-Line Blocking . . . . .	5
2.3.	Reduced Probability of RTO Expiry . . . . .	5
2.4.	Applications that do not Retransmit Lost Packets . . . . .	6
2.5.	Making Incipient Congestion Visible . . . . .	7
2.6.	Opportunities for new Transport Mechanisms . . . . .	7
3.	Network Support for ECN . . . . .	8
3.1.	The ECN Field . . . . .	9
3.2.	Forwarding ECN-Capable IP Packets . . . . .	9
3.3.	Enabling ECN in Network Devices . . . . .	9
3.4.	Co-existence of ECN and non-ECN flows . . . . .	10
3.5.	Bleaching and Middlebox Requirements to deploy ECN . . . . .	10
3.6.	Tunneling ECN and the use of ECN by Lower Layer Networks . . . . .	11
4.	Using ECN across the Internet . . . . .	11
4.1.	Partial Deployment . . . . .	11
4.2.	Detecting whether a Path Really Supports ECN . . . . .	12
4.3.	Detecting ECN Receiver Feedback Cheating . . . . .	12
5.	Summary: Enabling ECN in Network Devices and Hosts . . . . .	12
6.	Acknowledgements . . . . .	14
7.	IANA Considerations . . . . .	14
8.	Security Considerations . . . . .	14
9.	Revision Information . . . . .	15
10.	References . . . . .	16
10.1.	Normative References . . . . .	16
10.2.	Informative References . . . . .	17
	Authors' Addresses . . . . .	18

## 1. Introduction

Internet Transports (such as TCP and SCTP) are implemented in endpoints (Internet hosts) and are designed to detect and react to network congestion. Congestion may be detected by loss of an IP packet or, if Explicit Congestion Notification (ECN) [[RFC3168](#)] is enabled, by the reception of a packet with a Congestion Experienced (CE)-marking in the IP header. Both of these are treated by transports as indications of congestion. ECN may also be enabled by other transports: UDP applications that provide congestion control

Internet-Draft

Benefits of ECN

June 2015

may enable ECN when they are able to correctly process the ECN signals [ID.[RFC5405.bis](#)] (e.g., ECN with RTP [[RFC6679](#)]).

Active Queue Management (AQM) [ID.[RFC2309.bis](#)] is a class of techniques that can be used by network devices (a router, middlebox, or other device that forwards packets through the network) to manage the size of queues in network buffers. A network device that does not support AQM typically uses a drop-tail policy to drop excess IP packets when its queue becomes full. The discard of packets serves as a signal to the end-to-end transport that there may be congestion on the network path being used. This results in a congestion control reaction by the transport to reduce the maximum rate permitted by the sending endpoint.

When an application uses a transport that enables use of ECN [[RFC3168](#)], the transport layer sets the ECT(0) or ECT(1) codepoint in the IP header of packets that it sends. This indicates to network devices that they may mark, rather than drop the ECN-capable IP packets. An ECN-capable network device can then signal incipient congestion (network queueing) at a point before a transport experiences congestion loss or high queuing delay. The marking is generally performed as the result of various AQM algorithms, where the exact combination of AQM/ECN algorithms does not need to be known by the transport endpoints.

Since ECN makes it possible for the network to signal the presence of incipient congestion without incurring packet loss, it lets the network deliver some packets to an application that would otherwise have been dropped if the application or transport did not support ECN. This packet loss reduction is the most obvious benefit of ECN, but it is often relatively modest. However, enabling ECN can also result in a number of beneficial side-effects, some of which may be much more significant than the immediate packet loss reduction from ECN-marking instead of dropping packets. Several benefits reduce latency (e.g., reduced Head-of-Line Blocking).

The focus of the document is on usage of ECN by transport and application layer flows, not its implementation in endpoint hosts, or in routers and other network devices.

### 1.1. Terminology

The following terms are used:

AQM: Active Queue Management.

CE: Congestion Experienced, a codepoint value marked in the IP packet header.

Internet-Draft

Benefits of ECN

June 2015

ECN-capable: An IP packet with a non-zero ECN value (i.e., with a ECT(0), ECT(1), or the CE codepoint). An ECN-capable network device may forward, drop or queue an ECN-capable packet and may choose to CE-mark this packet when there is incipient congestion.

ECN field: A 2-bit field specified for use explicit congestion signalling in the IPv4 and IPv6 packet headers.

Endpoint: An Internet host that terminates a transport protocol connection across an Internet path.

Incipient Congestion: The detection of congestion when it is starting, perhaps by a network device noting that the arrival rate exceeds the forwarding rate.

Network device: A router, middlebox, or other device that forwards IP packets through the network.

non-ECN-capable: An IP packet with a zero value ECN codepoint. A non-ECN-capable packet may be forwarded, dropped or queued by a network device.

## 2. Benefit of using ECN to avoid Congestion Loss

An ECN-capable network device is expected to CE-mark an ECN-capable IP packet when an AQM method detects incipient congestion, rather than to drop the packet [ID.[RFC2309](#).bis]. An application can benefit from this marking in several ways:

### 2.1. Improved Throughput

ECN seeks to avoid the inefficiency of dropping data that has already made it across at least part of the network path.

ECN can improve the throughput of an application, although this increase in throughput is often not the most significant gain. When an application uses a light to moderately loaded network path, the number of packets that are dropped due to congestion is small. Using an example from Table 1 of [[RFC3649](#)], for a standard TCP sender with a Round Trip Time, RTT, of 0.1 seconds, a packet size of 1500 bytes and an average throughput of 1 Mbps, the average packet drop ratio would be 0.02 (i.e., 1 in 50 packets). This translates into an approximate 2% throughput gain if ECN is enabled. (Note that in heavy congestion, packet loss may be unavoidable with, or without, ECN.)

## 2.2. Reduced Head-of-Line Blocking

Many Internet transports provide in-order delivery of received data segments to the applications they support. For these applications, use of ECN can reduce the delay that can result when these applications experience packet loss.

Packet loss may occur for various reasons. One cause arises when an AQM scheme drops a packet as a signal of incipient congestion. Whatever the cause of loss, a missing packet needs to trigger a congestion control response. A reliable transport also triggers retransmission to recover the lost data. For a transport providing in-order delivery, this requires that the transport receiver stalls (or waits) for all data that was sent ahead of a lost segment to be correctly received before it can forward any later data to the application. A loss therefore creates a delay of at least one RTT after a loss event before data can be delivered to an application. We call this Head-of-Line (HOL) blocking. This is the usual requirement for TCP and SCTP. (PR-SCTP [RFC3758], UDP [RFC0768][ID.RFC5405.bis], and DCCP [RFC4340] provide a transport that does not provide re-ordering).

By enabling ECN, a transport continues to receive in-order data when there is incipient congestion, and can pass this data to the receiving application. Use of ECN avoids the additional reordering delay in a reliable transport. The sender still needs to make an appropriate congestion-response to reduce the maximum transmission rate for future traffic, which usually will require a reduction in the sending rate [ID.RFC5405.bis].)

## 2.3. Reduced Probability of RTO Expiry

Some patterns of packet loss can result in a retransmission time out (RTO), which causes a sudden and significant change in the allowed rate at which a transport/application can forward packets. Because ECN provides an alternative to drop for network devices to signal incipient congestion, this can reduce the probability of loss and hence reduce the likelihood of RTO expiry.

Internet transports/applications generally use a RTO timer as a last resort to detect and recover loss [ID.RFC5405.bis] [RFC5681]). Specifically, a RTO timer detects loss of a packet that is not followed by other packets, such as at the end of a burst of data segments or when an application becomes idle (either because the application has no further data to send or the network prevents sending further data, e.g., flow or congestion control at the transport layer). This loss of the last segment (or last few segments) of a traffic burst is also known as a "tail loss".



Standard transport recovery methods, such as Fast Recovery ([RFC5681](#)), are often unable to recover from a tail loss. This is because the endpoint receiver is unaware that the lost segments were actually sent, and therefore generates no feedback [[Fla13](#)]. Retransmission of these segments therefore relies on expiry of a transport retransmission timer. This timer is also used to detect a lack of forwarding along a path. Expiry of the RTO therefore results in the consequent loss of state about the network path being used. This typically includes resetting path estimates such as the RTT, re-initialising the congestion window, and possibly updates to other transport state. This can reduce the performance of the transport until it again adapts to the path.

An ECN-capable network device cannot eliminate the possibility of tail loss, because a drop may occur due to a traffic burst exceeding the instantaneous available capacity of a network buffer or as a result of the AQM algorithm (overload protection mechanisms, etc [[ID.RFC2309.bis](#)]). However, an ECN-capable network device that observes incipient congestion may be expected to buffer the IP packets of an ECN-capable flow and set a CE-mark in one or more packet(s), rather than triggering packet drop. Setting a CE-mark signals incipient congestion without forcing the transport/application to enter retransmission timeout. This reduces application-level latency and can improve the throughput for applications that send intermittent bursts of data.

The benefit of avoiding retransmission loss is expected to be significant when ECN is used on TCP SYN/ACK packets [[RFC5562](#)] where the RTO interval may be large because TCP cannot base the timeout period on prior RTT measurements from the same connection.

#### 2.4. Applications that do not Retransmit Lost Packets

A transport that enables ECN can receive timely congestion signals without the need to retransmit packets each time it receives a congestion signal.

Some latency-critical applications do not retransmit lost packets, yet may be able to adjust their sending rate following detection of incipient congestion. Examples of such applications include UDP-based services that carry Voice over IP (VoIP), interactive video, or real-time data. The performance of many such applications degrades rapidly with increasing packet loss and the transport/application may therefore employ mechanisms (e.g., packet forward error correction, data duplication, or media codec error concealment) to mitigate the immediate effect of congestion loss on the application. Some mechanisms consume additional network capacity, some require additional processing and some contribute additional path latency



when congestion is experienced. By decoupling congestion control from loss, ECN can allow transports that support these applications to reduce their rate before the application experiences loss from congestion. This can reduce the negative impact of triggering loss-hiding mechanisms with a direct positive impact on the quality experienced by the users of these applications.

## 2.5. Making Incipient Congestion Visible

A characteristic of using ECN is that it exposes the presence of congestion on a network path to the transport and network layers allowing information to be collected about the presence of incipient congestion.

Recording the presence of CE-marked packets can provide information about the current congestion level experienced on a network path. A network flow that only experiences CE-marking and no loss implies that the sending endpoint is experiencing only congestion. A network flow may also experience loss (e.g., due to queue overflow, AQM methods that protect other flows, link corruption or loss in middleboxes). When a mixture of ECN-marking and packet loss is experienced, transports and measurements need to assume there is congestion [ID.RFC2309.bis]. An absence of CE-marks therefore does not indicate a path has not experienced congestion.

The reception of CE-marked packets can be used to monitor the level of congestion by a transport/application or a network operator. For example, ECN measurements are used by Congestion Exposure (ConEx) [RFC6789]. In contrast, metering packet loss is harder.

## 2.6. Opportunities for new Transport Mechanisms

ECN can enable design and deployment of new algorithms in network devices and Internet transports. Internet transports need to regard both loss and CE-marking as an indication of congestion. However, while the amount of feedback provided by drop ought naturally to be minimized, this is not the case for ECN. In contrast, an ECN-Capable network device could provide richer (more frequent and fine-grained) indication of its congestion state to the transport.

All ECN-capable receiving endpoints need to provide feedback to the transport sender to indicate that CE-marks have been received. [RFC3168] provides one method that signals once each round trip time that CE-marked packets have been received.

A receiving endpoint may provide more detailed feedback to the congestion controller at the sender (e.g., describing the set of received ECN codepoints, or indicating each received CE-marked

packet). Precise feedback about the number of CE-marks encountered is supported by the Real Time Protocol (RTP) when used over UDP [RFC6679] and has been proposed for SCTP [ST14] and TCP [ID.Acc.ECN].

More detailed feedback is expected to enable evolution of transport protocols allowing the congestion control mechanism to make a more appropriate decision on how to react to congestion. Designers of transport protocols need to consider not only how network devices CE-mark packets, but also how the control loop in the application/transport reacts to reception of these CE-marked packets.

Benefit has been noted when packets are CE-marked early using an instantaneous queue, and if the receiving endpoint provides feedback about the number of packet marks encountered, an improved sender behavior has been shown to be possible, e.g, Datacenter TCP (DCTCP) [AL10]. DCTCP is targeted at controlled environments such as a datacenter. This is work-in-progress and it is currently unknown whether or how such behaviour could be safely introduced into the Internet. Any update to an Internet transport protocol requires careful consideration of the robustness of the behaviour when working with endpoints or network devices that were not designed for the new congestion reaction.

### 3. Network Support for ECN

For an application to use ECN requires that the endpoints first enable ECN within the transport being used, but also for all network devices along the path to at least forward IP packets that set a non-zero ECN codepoint.

ECN can be deployed both in the general Internet and in controlled environments:

- o ECN can be incrementally deployed in the general Internet. The IETF has provided guidance on configuration and usage in [ID.RFC2309.bis].
- o ECN may be deployed within a controlled environment, for example within a data centre or within a well-managed private network. This use of ECN may be tuned to the specific use-case. An example is DCTCP [AL10] [ID.DCTCP].

Early experience of using ECN across the general Internet encountered a number of operational difficulties when the network path either failed to transfer ECN-capable packets or inappropriately changed the ECN codepoints [BA11]. A recent survey reported a growing support for network paths to pass ECN codepoints [TR15].

The remainder of this section identifies what is needed for network devices to effectively support ECN.

### 3.1. The ECN Field

The current IPv4 and IPv6 specifications assign usage of 2 bits in the IP header to carry the ECN codepoint. This 2-bit field was reserved in [RFC2474] and assigned in [RFC3168].

[RFC4774] discusses some of the issues in defining alternate semantics for the ECN field, and specifies requirements for a safe coexistence in an Internet that could include routers that do not understand the defined alternate semantics.

### 3.2. Forwarding ECN-Capable IP Packets

Not all network devices along a path need to be ECN-capable (i.e., perform CE-marking). However, all network devices need to be configured not to drop packets solely because the ECT(0) or ECT(1) codepoints are used.

Any network device that does not perform CE-marking of an ECN-capable packet can be expected to drop these packets under congestion. Applications that experience congestion at these network devices do not see any benefit from enabling ECN. However, they may be expected to see benefit if the congestion were to occur within a network device that did support ECN.

### 3.3. Enabling ECN in Network Devices

Network devices should use an AQM algorithm that CE-marks ECN-capable traffic when making decisions about the response to congestion [ID.RFC2309.bis]. An ECN method should set a CE-mark on ECN-capable packets in the presence of incipient congestion. A CE-marked packet will be interpreted as an indication of incipient congestion by the transport endpoints.

There is opportunity to design an AQM method for an ECN-capable network device that differs from an AQM method designed to drop packets. [ID.RFC2309.bis] states that the network device should allow this behaviour to be configurable.

[RFC3168] describes a method in which a network device sets the CE-mark at the time that the network device would otherwise have dropped the packet. While it has often been assumed that network devices should CE-mark packets at the same level of congestion at which they would otherwise have dropped them, [ID.RFC2309.bis] recommends that network devices allow independent configuration of the settings for

AQM dropping and ECN marking. Such separate configuration of the drop and mark policies is supported in some network devices.

#### 3.4. Co-existence of ECN and non-ECN flows

Network devices need to be able to forward all IP flows and provide appropriate treatment for both ECN and non-ECN traffic.

The design considerations for an AQM scheme supporting ECN needs to consider the impact of queueing during incipient congestion. For example, a simple AQM scheme could choose to queue ECN-capable and non-ECN capable flows in the same queue with an ECN scheme that CE-mark packets during incipient congestion. The CE-marked packets that remain in the queue during congestion can continue to contribute to queueing delay. In contrast, non-ECN-capable packets would normally be dropped by an AQM scheme under incipient congestion. This difference in queueing is one motivation for consideration of more advanced AQM schemes, and may provide an incentive for enabling flow isolation using scheduling [ID.RFC2309.bis]. The IETF is defining methods to evaluate the suitability of AQM schemes for deployment in the general Internet [ID.AQM.eval].

#### 3.5. Bleaching and Middlebox Requirements to deploy ECN

Network devices should not be configured to change the ECN codepoint in the packets that they forward, except to set the CE-codepoint to signal incipient congestion.

Cases have been noted where an endpoint sends a packet with a non-zero ECN mark, but the packet is received by the remote endpoint with a zero ECN codepoint [TR15]. This could be a result of a policy that erases or "bleaches" the ECN codepoint values at a network edge (resetting the codepoint to zero). Bleaching may occur for various reasons (including normalising packets to hide which equipment supports ECN). This policy prevents use of ECN by applications.

When ECN-capable IP packets, marked as ECT(0) or ECT(1), are remarked to non-ECN-capable (i.e., the ECN field is set to zero codepoint), this could result in the packets being dropped by ECN-capable network devices further along the path. This eliminates the advantage of using of ECN.

A network device must not change a packet with a CE mark to a zero codepoint, if the network device decides not to forward the packet with the CE-mark, it has to instead drop the packet and not bleach the marking. This is because a CE-marked packet has already received ECN treatment in the network, and remarking it would then hide the congestion signal from the receiving endpoint. This eliminates the

benefits of ECN. It can also slow down the response to congestion compared to using AQM, because the transport will only react if it later discovers congestion by some other mechanism.

Prior to [RFC2474](#), a previous usage assigned the bits now forming the ECN field as a part of the now deprecated Type of Service (ToS) field [[RFC1349](#)]. A network device that conforms to this older specification was allowed to remark or erase the ECN codepoints, and such equipment needs to be updated to the current specifications to support ECN.

### 3.6. Tunneling ECN and the use of ECN by Lower Layer Networks

Some networks may use ECN internally or tunnel ECN (e.g., for traffic engineering or security). These methods need to ensure that the ECN-field of the tunnel packets is handled correctly at the ingress and egress of the tunnel. Guidance on the correct use of ECN is provided in [[RFC6040](#)].

Further guidance on the encapsulation and use of ECN by non-IP network devices is provided in [[ID.ECN-Encap](#)].

## 4. Using ECN across the Internet

A receiving endpoint needs to report the loss it experiences when it uses loss-based congestion control. So also, when ECN is enabled, a receiving endpoint must correctly report the presence of CE-marks by providing a mechanism to feed this congestion information back to the sending endpoint, [[RFC3168](#)], [[ID.RFC5405.bis](#)], enabling the sender to react to experienced congestion. This mechanism needs to be designed to operate robustly across a wide range of Internet path characteristics. This section describes partial deployment, how ECN-enabled endpoints can continue to work effectively over a path that experiences misbehaving network devices or when an endpoint does not correctly provide feedback of ECN congestion information.

### 4.1. Partial Deployment

Use of ECN is negotiated between the endpoints prior to using the mechanism. This

ECN has been designed to allow incremental partial deployment [[RFC3168](#)]. Any network device can choose to use either ECN or some other loss-based policy to manage its traffic. Similarly, transport/application negotiation allows senders and receiving endpoints to choose whether ECN is to be used to manage congestion for a particular network flow.

#### 4.2. Detecting whether a Path Really Supports ECN

Internet transport and applications need to be robust to the variety and sometimes varying path characteristics that are encountered in the general Internet. They need to monitor correct forwarding of ECN over the entire path and duration of a session.

To be robust, applications and transports need to be designed with the expectation of heterogeneous forwarding (e.g., where some IP packets are CE-marked by one network device, and some by another, possibly using a different AQM algorithm, or when a combination of CE-marking and loss-based congestion indications are used. ([ID.AQM.eval] describes methodologies for evaluating AQM schemes.)

A transport/application also needs to be robust to path changes. A change in the set of network devices along a path could impact the ability to effectively signal or use ECN across the path, e.g., when a path changes to use a middlebox that bleaches ECN codepoints (see [Section 3.5](#)).

A sending endpoint can check that any ECN-marks applied to packets received from the path are indeed delivered to the remote receiving endpoint and that appropriate feedback is provided. (This could be done by a sender setting known ECN codepoints for specific packets in a network flow and then checking whether the remote endpoint correctly reports these marks [ID.Fallback], [TR15].) If a sender detects misuse of ECN, it needs to either conservatively react to congestion or even fall back to using loss-based recovery instead of ECN.

#### 4.3. Detecting ECN Receiver Feedback Cheating

Appropriate feedback requires that the endpoint receiver does not try to conceal reception of CE-marked packets in the ECN feedback information provided to the sending endpoint [ID.RFC2309.bis]. Designers of applications/transports are therefore encouraged to include mechanisms that can detect this misbehavior. If a sending endpoint detects that a receiver is not correctly providing this feedback, it can either conservatively react to congestion or fall back to using loss-based recovery instead of ECN.

#### 5. Summary: Enabling ECN in Network Devices and Hosts

This section summarises the benefits of deploying and using ECN within the Internet. It also provides a list of summary of prerequisites to achieve ECN deployment.

Application developers should where possible use transports that enable the benefits of ECN. Applications that directly use UDP need to provide support to implement the functions required for ECN [ID.RFC5405.bis]. Once enabled, an application that uses a transport that supports ECN will experience the benefits of ECN as network deployment starts to enable ECN. The application does not need to be rewritten to gain these benefits. Table 1 summarises the key benefits.

Section	Benefit
2.1	Improved throughput
2.2	Reduced Head-of-Line blocking
2.3	Reduced probability of RTO Expiry
2.4	Applications that do not retransmit lost packets
2.5	Making incipient congestion visible
2.6	Opportunities for new transport mechanisms

Table 1: Summary of Key Benefits

Network operators and people configuring network devices should enable ECN [ID.RFC2309.bis].

Prerequisites for network devices (including IP routers) to enable use of ECN include:

- o A network device that updates the ECN field in IP packets must use IETF-specified methods (see [Section 3.1](#)).
- o A network device may support alternate ECN semantics (see [Section 3.1](#)).
- o Network devices need to be configured not to drop packets solely because the ECT(0) or ECT(1) codepoints are used (see [Section 3.2](#)).
- o A network device must not change a packet with a CE mark to a zero codepoint, if the network device decides not to forward the packet with the CE-mark, it has to instead drop the packet and not bleach the marking (see [Section 3.5](#)).
- o An ECN-capable network device should correctly update the ECN codepoint of ECN-capable packets in the presence of incipient congestion (see [Section 3.3](#)).

Internet-Draft

Benefits of ECN

June 2015

- o Network devices need to be able to forward both ECN-capable and non-ECN-capable flows (see [Section 3.4](#)).

Prerequisites for network endpoints to enable use of ECN include:

- o an application should use an Internet transport that can set and receive ECN marks (see [Section 4](#)).
- o an ECN-capable transport/application must return feedback indicating congestion to the sending endpoint and perform an appropriate congestion response (see [Section 4](#)).
- o an ECN-capable transport/application should detect paths where there is misuse of ECN and either conservatively react to congestion or even fall back to not using ECN (see [Section 4.2](#)).
- o designers of applications/transport are encouraged to include mechanisms that can detect and react appropriately to misbehaving receivers that fail to report CE-marked packets (see [Section 4.3](#)).

## 6. Acknowledgements

The authors were part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the authors.

The authors would like to thank the following people for their comments on prior versions of this document: Bob Briscoe, David Collier-Brown, Colin Perkins, Richard Scheffenegger, Dave Taht, Wes Eddy, Fred Baker, Mikael Abrahamsson, Mirja Kuehlewind, John Leslie, and other members of the AQM and TSV Working Groups.

## 7. IANA Considerations

XX RFC ED - PLEASE REMOVE THIS SECTION XXX

This memo includes no request to IANA.

## 8. Security Considerations

This document introduces no new security considerations. Each RFC listed in this document discusses the security considerations of the specification it contains.



Internet-Draft

Benefits of ECN

June 2015

## 9. Revision Information

XXX RFC-Ed please remove this section prior to publication.

Revision 00 was the first WG draft.

Revision 01 includes updates to complete all the sections and a rewrite to improve readability. Added [section 2](#). Author list reversed, since Gorry has become the lead author. Corrections following feedback from Wes Eddy upon review of an interim version of this draft.

Note: Wes Eddy raised a question about whether discussion of the ECN Pitfalls could be improved or restructured - this is expected to be addressed in the next revision.

Revision 02 updates the title, and also the description of mechanisms that help with partial ECN support.

We think this draft is ready for wider review. Comments are welcome to the authors or via the IETF AQM or TSVWG mailing lists.

Revision 03 includes updates from the mailing list and WG discussions at the Dallas IETF meeting.

The section "Avoiding Capacity Overshoot" was removed, since this refers primarily to an AQM benefit, and the additional benefits of ECN are already stated. Separated normative and informative references

Revision 04 (WG Review during WGLC)

Updated the abstract.

Added a table of contents.

Addressed various (some conflicting) comments during WGLC with new text.

The section on Network Support for ECN was moved, and some suggestions for rewording sections were implemented.

Decided not to remove section headers for 2.1 and 2.2 - to ensure the document clearly calls-out the benefits.

Updated references. Updated text to improve consistency of terms and added definitions for key terms.

Internet-Draft

Benefits of ECN

June 2015

Note: The group suggested this document should not define recommendations for end hosts or routers, but simply state the things needs to enable deployment to be successful.

Revision 05 (after WGLC comments)

Updated abstract to avoid suggesting that this describes new methods for deployment.

Added ECN-field definition, and sorted terms in order.

Added an opening para to each "benefit" to say what this is. Sought to remove redundancy between sections.

Added new section on Codepoints to avoid saying the same thing twice.

Reworked sections 3 and 4 to clarify discussion and to remove unnecessary text.

Reformatted Summary to refer to sections describing things, rather than appear as a list of new recommendations. Reordered to match the new document order.

Note: This version expects an update to [RFC5405.bis](#) that will indicate UDP ECN requirements (normative).

## 10. References

### 10.1. Normative References

[ID.[RFC2309.bis](#)]

Baker, F. and G. Fairhurst, "IETF Recommendations Regarding Active Queue Management", Internet-draft [draft-ietf-aqm-recommendation-06](#), October 2014.

[ID.[RFC5405.bis](#)]

Eggert, Lars., Fairhurst, Gorry., and Greg. Shepherd, "Unicast UDP Usage Guidelines", 2015.

[RFC2474] "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers".

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", [RFC 3168](#), September 2001.

[RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", [RFC 6040](#), November 2010.

Internet-Draft

Benefits of ECN

June 2015

## 10.2. Informative References

- [AL10] Alizadeh, M., Greenberg, A., Maltz, D., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and M. Sridharan, "Data Center TCP (DCTCP)", SIGCOMM 2010, August 2010.
- [BA11] Bauer, Steven., Beverly, Robert., and Arthur. Berger, "Measuring the State of ECN Readiness in Servers, Clients, and Routers, ACM IMC", 2011.
- [Fla13] Flach, Tobias., Dukkkipati, Nandita., Terzis, Andreas., Raghavan, Barath., Cardwell, Neal., Cheng, Yuchung., Jain, Ankur., Hao, Shuai., Katz-Bassett, Ethan., and Ramesh. Govindan, "Reducing web latency: the virtue of gentle aggression.", SIGCOMM 2013, October 2013.
- [ID.Acc.ECN] Briscoe, Bob., Scheffeneger, Richard., and Mirja. Kuehlewind, "More Accurate ECN Feedback in TCP, Work-in-Progress".
- [ID.AQM.eval] Kuhn, Nicolas., Natarajan, Preethi., Ros, David., and Naeem. Khademi, "AQM Characterization Guidelines (Work-in-progress, [draft-ietf-aqm-eval-guidelines](#))", 2015.
- [ID.DCTCP] Bensley, S., Eggert, Lars., and D. Thaler, "Microsoft's Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters (Work-in-progress, [draft-bensley-tcpm-dctcp](#))", 2015.
- [ID.ECN-Encap] Briscoe, B., Kaippallimalil, J., and P. Thaler, "Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP", Internet-draft, IETF work-in-progress [draft-ietf-tsvwg-ecn-encap-guidelines](#).
- [ID.Fallback] Kuehlewind, Mirja. and Brian. Trammell, "A Mechanism for ECN Path Probing and Fallback, [draft-kuehlewind-tcpm-ecn-fallback](#), Work-in-Progress".
- [RFC0768] Postel, J., "User Datagram Protocol", 1980.
- [RFC1349] "Type of Service in the Internet Protocol Suite".

Internet-Draft

Benefits of ECN

June 2015

- [RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", [RFC 3649](#), December 2003.
- [RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", [RFC 3758](#), May 2004.
- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", [RFC 4340](#), March 2006.
- [RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", [BCP 124](#), [RFC 4774](#), November 2006.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", [RFC 5562](#), June 2009.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", [RFC 5681](#), September 2009.
- [RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", [RFC 6679](#), August 2012.
- [RFC6789] Briscoe, B., Woundy, R., and A. Cooper, "Congestion Exposure (ConEx) Concepts and Use Cases", [RFC 6789](#), December 2012.
- [ST14] Stewart, R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", Internet-draft [draft-stewart-tsvwg-sctpecn-05.txt](#), January 2014.
- [TR15] Trammell, Brian., Kuehlewind, Mirja., Boppart, Damiano, Learmonth, Iain., and Gorry. Fairhurst, "Enabling internet-wide deployment of Explicit Congestion Notification Trammell, B., Kuehlewind, M., Boppart, D., Learmonth, I., Fairhurst, G. & Scheffnegger, Passive and Active Measurement Conference (PAM)", March 2015.

Authors' Addresses

Internet-Draft

Benefits of ECN

June 2015

Godred Fairhurst  
University of Aberdeen  
School of Engineering, Fraser Noble Building  
Aberdeen AB24 3UE  
UK

Email: [gorry@erg.abdn.ac.uk](mailto:gorry@erg.abdn.ac.uk)

Michael Welzl  
University of Oslo  
PO Box 1080 Blindern  
Oslo N-0316  
Norway

Phone: +47 22 85 24 20  
Email: [michawe@ifi.uio.no](mailto:michawe@ifi.uio.no)

# ‘Data Centre to the Home’: Ultra-Low Latency for All

Koen De Schepper<sup>†</sup> Olga Bondarenko<sup>\* ‡</sup> Ing-Jyh Tsang<sup>†</sup> Bob Briscoe<sup>§</sup>  
<sup>†</sup>Alcatel-Lucent, Belgium <sup>‡</sup>Simula Research Laboratory, Norway <sup>§</sup>BT, UK  
<sup>†</sup>{koen.de\_schepper|ing-jyh.tsang}@alcatel-lucent.com  
<sup>‡</sup>olgabo@simula.no <sup>§</sup>research@bobbriscoe.net

## ABSTRACT

Data Centre TCP (DCTCP) was designed to provide predictably low queuing latency, near-zero loss, and throughput scalability using explicit congestion notification (ECN) and an extremely simple marking behaviour on switches. However, DCTCP does not co-exist with existing TCP traffic—throughput starves. So, until now, DCTCP could only be deployed where a clean-slate environment could be arranged, such as in private data centres. This paper proposes ‘Coupled Active Queue Management (AQM)’ to allow scalable congestion controls like DCTCP to safely co-exist with classic Internet traffic. In extensive tests within the edge gateway of a realistic broadband access testbed, the Coupled AQM ensures that a flow runs at about the same rate whether it uses DCTCP or TCP Reno/Cubic, but without inspecting transport layer flow identifiers. DCTCP achieves sub-millisecond average queuing delay and zero congestion loss under a wide range of mixes of DCTCP and ‘classic’ broadband Internet traffic, without compromising the performance of the classic traffic. The solution also reduces network complexity and eliminates network configuration.

## 1. INTRODUCTION

Latency is becoming the critical performance factor for many (most?) applications on the public Internet, e.g. Web, voice, conversational video, gaming and finance apps. In the developed world, further increases in access network bit-rate offer diminishing returns, whereas latency is still a multi-faceted problem. In the last decade or so, much has been done to reduce propagation time by placing caches or servers closer to users. However, queuing remains a major component of latency.

The Diffserv architecture provides Expedited Forwarding, so that low latency traffic can jump the queue of other traffic. However, on access links dedicated to individual sites (homes, small enterprises or mobile devices), often *all* traffic at any one time will be latency-sensitive. Then Diffserv is of little use. Instead, we need to remove the causes of any unnecessary delay.

\*The first two authors contributed equally

The bufferbloat project has shown that excessively-large buffering (‘bufferbloat’) has been introducing significantly more delay than the underlying propagation time. These delays appear only intermittently—only when a capacity-seeking (e.g. TCP) flow is long enough for the queue to fill the buffer, making every packet in other flows sharing the buffer sit through the queue.

Active queue management (AQM) was originally developed to solve this problem (and others). Unlike Diffserv, AQM controls latency for *all* traffic in a class. In general, AQMs introduce an increasing level of discard from the buffer the longer the queue persists above a shallow threshold. This gives sufficient signals to capacity-seeking (aka. greedy) flows to keep the buffer empty for its intended purpose: absorbing bursts. However, RED and other algorithms from the 1990s were sensitive to their configuration and hard to set correctly. So, AQM was not widely deployed. More recent state-of-the-art AQMs, e.g. fq-CoDel [8], PIE [13], Adaptive RED, define the threshold in time not bytes, so it is invariant for different link rates.

It seems that further changes to the network alone will now yield diminishing returns. Data Centre TCP (DCTCP [1]) teaches us that a small but radical change to TCP is needed to cut two major outstanding causes of queuing delay variability:

1. the ‘sawtooth’ varying rate of TCP itself;
2. the smoothing delay deliberately introduced into

AQMs to permit bursts without triggering losses. The former causes a flow’s round trip time (RTT) to vary from about 1 to 2 times the base RTT between the machines in question. The latter delays the system’s response to change by a worst-case (transcontinental) RTT, which could be hundreds of times the actual RTT of typical traffic from localised CDNs.

Latency is not our only concern.

3. It was known when TCP was first developed that it would not scale to high bandwidth-delay products. Given regular broadband bit-rates over WAN distances are already beyond the scaling range of ‘classic’ TCP Reno, ‘less unscalable’ Cubic [7] and Compound [15] variants of TCP have been successfully deployed. How-

ever, these are now approaching their scaling limits. Unfortunately, fully scalable TCPs cause ‘classic’ TCP to starve itself, which is why they have been confined to private data centres or research testbeds (until now).

Our contribution is a ‘Coupled AQM’ that solves the problem of coexistence between DCTCP and classic flows, without having to inspect flow identifiers. It needs fewer operations per packet than RED uses. Also, no network configuration is needed for a wide range of scenarios.

It uses two queues for two services: “Classic” and “Low-Latency, Low-Loss and Scalable” (L4S), denoted resp. by subscripts ‘C’ and ‘L’. The ‘Classic’ service is intended for all the behaviours that currently co-exist with TCP Reno (TCP Cubic, Compound, SCTP, etc). The ‘L4S’ service is intended for DCTCP traffic but it is also more general—it will allow a set of congestion controls similar to DCTCP (e.g. Relentless) to evolve.

The AQM couples marking and/or dropping across the two queues such that a flow will get roughly the same throughput whichever it uses. This enables scalable congestion controls like DCTCP to give stunningly low and predictably low latency, without compromising the performance of competing ‘classic’ Internet traffic.

In our tests, we use DCTCP unmodified, despite its known deficiencies (listed as further work in Section 10). Our focus is on getting the network service in place. Then, without any management intervention, applications can exploit it by migrating to scalable controls like DCTCP, which can then evolve *while* their benefits are being enjoyed by everyone on the Internet.

We also conducted subjective testing with a demanding panoramic interactive video application run over a stack with DCTCP enabled and deployed on the testbed. Each user could pan or zoom their own HD sub-window of a larger video scene from a football match (Figure 1). Even though the user was also downloading large amounts of L4S and Classic data, latency was so low that the picture appeared to stick to their finger on the touchpad (all the L4S data achieved the same ultra-low latency). With an alternative AQM, the video noticeably lagged behind the finger gestures.

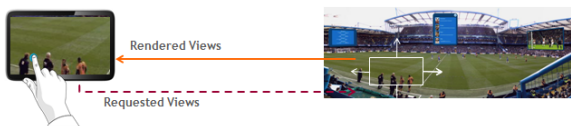


Figure 1: Panoramic interactive video application

**Document Outline.** Section 2 gives the intuition for our design choices, deferring the mathematical rationale to Section 3. We then present our two main contributions: the ‘Coupled AQM’ that addresses throughput equivalence (Section 4) then a Dual Queue structure that provides latency separation (Section 5).

To evaluate the new AQM’s performance, we have built it into the downstream of a realistic end-to-end

broadband testbed (Section 6). Section 7 gives a condensed report of thousands of experiments to quantify its performance against RED, PIE and fq\_CoDel.

The paper ends with discussion of standardisation aspects (Section 8) before the usual tailpieces.

## 2. RATIONALE

### 2.1 Intuition: Why DCTCP and ECN?

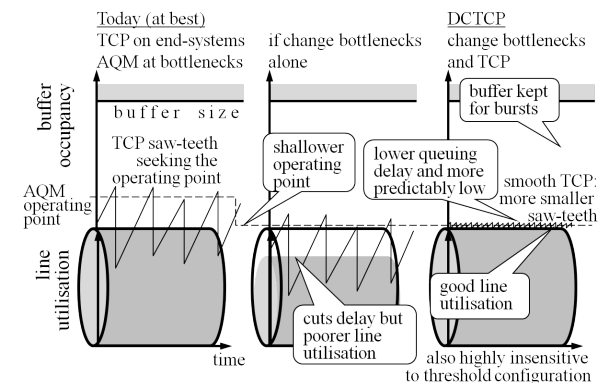


Figure 2: Data Centre TCP: Intuition

Figure 2 shows how DCTCP resolves the dilemma caused by TCP’s large sawteeth (problem #1 in the Introduction). DCTCP’s smaller sawteeth allow both maximal utilisation of the link (the grey cylinder) and minimal occupancy of the buffer (above the cylinder). The size of DCTCP’s sawteeth also remains invariant with increasing bit-rate, which addresses problem #3.

DCTCP also solves problem #2 above, by requiring the network to signal congestion immediately, without any smoothing delay (not illustrated in the figure).

The underlying reason that DCTCP can solve problems #1 and #2 is that it signals congestion with explicit congestion notification (ECN [14]). ECN is purely a signal, whereas drop is both an impairment and a signal, which compromises signalling flexibility:

1. DCTCP’s finer sawteeth imply a higher signalling rate, which would be untenable as loss;
2. If the queue grows, an AQM holds back from introducing loss in case it is just a sub-RTT burst, whereas emitting ECN immediately is harmless.

This last point is most significant, because the sender knows its own RTT, which DCTCP could use as the time constant to smooth incoming ECN marks. Whereas, with drop, the network has to smooth but it does not know each flow’s RTT, so it has to use a worst-case (transcontinental) RTT, to avoid instability if a flow does have such a long RTT.

ECN also offers the obvious latency benefit of near-zero congestion loss, of most concern to short flows. This removes retransmission and time-out delays and the head-of-line blocking that a loss can cause when a



single TCP flow carries a multiplex of streams.

To be concrete, in our testbed, we identify classic traffic by a cleared ECN field in the IP header (not ECN-capable). In Section 8, we discuss an alternative but non-preferred compromise that would allow Classic traffic to use ECN as well.

The coexistence mechanism has nothing to do with flow identifiers; it solely contrives the two aggregate signalling levels to compensate for the different responses to the signals exhibited by flows in each aggregate.

## 2.2 Delay vs. Drop

Evaluating our Coupled AQM at high load, we noticed that PIE and fq\_CoDel produced unusually high drop, which took over from queuing as the dominant cause of delay for short Web flows. Thus, the goal of AQMs like PIE and fq\_CoDel—to cap queuing delay at a fixed target—is misguided. If TCP cannot increase queuing delay, it will increase drop instead.

Our coupled AQM constrains classic traffic with a softened delay target, but not as soft as RED. We call the mechanism Curvy RED. The original RED algorithm increases drop probability linearly with queue growth. With a curviness parameter of 2, the Curvy RED algorithm increases drop with the square of the queue, so it pushes back increasingly aggressively the more the queue grows. This is implemented by simply comparing the queue to two random numbers, not one. It also uses queuing time, not queue length.

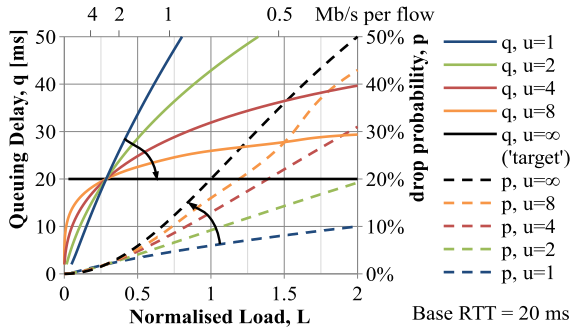


Figure 3: The Delay-Loss Dilemma: Sensitivity to Load for Curvy RED AQMs with Increasing Curviness,  $u$ .

The order of the legend follows that of the curves. Normalised load,  $L \propto 1/\text{bit-rate per Reno flow}$  (top).

Conceptually, as the curviness parameter tends to infinity (shown by the two pincer arrows in Figure 3 from [2]), it approximates the hard delay target of PIE or CoDel, shown as a flat step configured at 20 ms. To clamp delay to a hard constraint, the corresponding dashed drop curve has to increase aggressively with load. In this paper, we show that intermediate curviness can still keep queuing delay under control whatever the load, but without having to introduce so much loss that loss itself becomes the dominant cause of delay.

## 2.3 Per-Flow Queuing and Self-Harm

Superficially, it might seem that per-flow queuing (as in fq\_CoDel) would address the problems in Section 1; it is designed to isolate a latency-sensitive flow from the delays induced by other flows. However, that does not protect a latency-sensitive flow from saw-toothing, which the flow still inflicts upon *itself*.

It might seem that self-inflicted queuing delay should not count. To avoid delay in a dedicated remote queue, a sender would have to hold back the data, causing the same delay, just in a different place. It seems preferable to release the data into a dedicated network queue; then it will be ready to go as soon as the queue drains.

However, this logic applies i) if and only if the sender somehow knows that the bottleneck in question implements per-flow queuing and ii) only for non-adaptive applications. Modern Web applications multiplex streams into a single flow, e.g. SPDY or HTTP/2; then they alter which stream they prioritise, depending on the progress of each of the streams already sent. Our panoramic interactive video app is another example, where the amount of data sent (the frame rate) at any instant depends on how much prior data was delivered. For such applications, a remote queue is not useful if their self-induced queue is remote; once optional data is in flight, they cannot suppress it. To adapt how much they send, they need to maintain their send-queue locally.

Per-flow queuing was not an appropriate solution, given it i) does not solve the self-induced latency problem; ii) needs to inspect transport layer headers (preventing transport evolution); and iii) requires many more queues and supporting scheduling structures, whereas our primary goal was to *reduce* cost and complexity.

## 3. STEADY STATE RATE

An L4S congestion controller such as DCTCP achieves low latency, low loss and low throughput variations by driving the network to give it a more responsive signal with a higher resolution. Therefore, a solution must be found to reduce the congestion signal intensity for Classic congestion controllers (TCP Cubic and Reno), but not for L4S (DCTCP).

Our first concern is to support rough equality in terms of long term throughput (other factors being equal). In terms of throughput for short flows, we want to allow differentiation to support lower latency, less loss and less throughput variation for the improved class.

For our purposes, it is sufficient to ignore dynamic aspects, and apply the simplified models in (1) and (2) for TCP Reno and Cubic packet rate as in [12] and [7]:

$$r_{\text{reno}} = \frac{1.22}{p^{1/2}T} \quad (1) \quad r_{\text{cubic}} = \frac{1.17}{p^{3/4}T^{1/4}} \quad (2)$$

where  $p$  is drop probability and  $T$  is RTT.

The Cubic implementation in Linux provides a fallback to TCP Reno when RTT is small. Due to the



different decrease factor, the steady state rate will deviate from (1) with a slightly higher constant (3).

The implicit switchover RTT can be derived from (2) and (3). Pure Cubic behaviour (as defined in (2)) will become active when (4) is false.

$$r_{cubic} = \frac{1.68}{p^{1/2}T} \quad (3) \quad r * T^{5/2} < 3.5 \quad (4)$$

For typical user access from local data centres ( $T < 20$  ms), pure Cubic behavior can only be expected for flows faster than 500Mbps. Therefore, at least for AQMs in access networks, we can focus on the relationship between Reno and DCTCP and assume that Cubic will be in Reno mode, or at least not too far outside it.

We have derived a similar model for DCTCP, assuming an idealised uniform deterministic marker, which marks every  $1/p$  packets. A DCTCP congestion controller has an incremental window increase per RTT  $\alpha = 1$  and a multiplicative decrease factor  $\beta = p/2$  (with  $p$  being estimated). So, every RTT,  $W$  is increased by  $W \leftarrow W + 1$ , meaning that under steady state, this must be compensated every RTT by (5). This decrease is steered by marks, as defined in (6).

$$W \leftarrow \left(1 - \frac{1}{W}\right) W \quad (5) \quad W \leftarrow \left(1 - \frac{p}{2}\right) W \quad (6)$$

From (5) and (6), we see that to preserve this balance, (7) must be true, which determines the window and therefore the steady state packet rate in (8).

$$\frac{p}{2} = \frac{1}{W} \quad (7) \quad r_{dc} = \frac{2}{pT} \quad (8) \quad r_{dcth} = \frac{2}{p^2T} \quad (9)$$

Note that (9) derived in the DCTCP paper [1] has a different exponent of  $p$  compared to (8). The reason is that (9) is defined for a step threshold, which causes an on-off marking pattern. When DCTCP marking is coupled to the drop probability of Classic traffic, it uses fractional marking probability, so (8) will be applicable.

#### 4. COUPLED AQM FOR EQUAL RATE

Knowing the relation between network congestion signal, (mark or drop) probability and the flow rate, we can adjust the feedback from the network to each type of congestion control. For TCP Reno and DCTCP, we substitute (1) and (8) in  $r_{reno} = r_{dc}$ :

$$\frac{1.22}{p_{reno}^{1/2}T_{reno}} = \frac{2}{p_{dc}T_{dc}} \quad (10)$$

If the RTTs are equal (they may not be; see Section 5), we can arrange the rates to be equal using the simple relation between the probabilities, defined in (11). This relation could be achieved by a modified AQM in the network, shown in Figure 4.

$$p_{reno} = \left(\frac{p_{dc}}{1.63}\right)^2 \quad (11)$$

Probabilistic mark/drop is typically implemented by comparing the probability  $p$  with a pseudo-random gen-

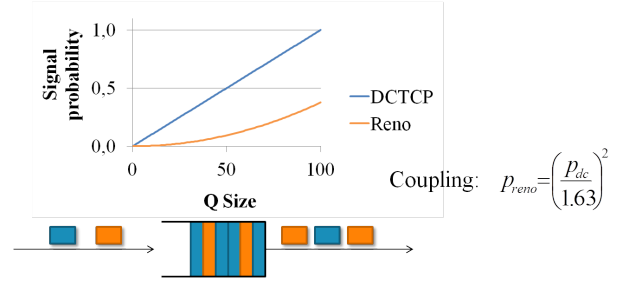


Figure 4: Equal rate AQM for TCP Reno and DCTCP

erated value  $R$  per packet. A signal is applied for a packet when  $p > R$ . The advantage of using relation (11) is that  $p^2$  can easily be acquired by comparing  $p$  with 2 pseudo-random generated values and signal only if both random values are smaller than  $p$ :  $p > \max(R_1, R_2)$ .

A square curve is a simple way to emulate the three piecewise-linear parts of RED. Drop probability hugs the zero axis, while the queue is shallow. Then, as load increases, it introduces a growing ‘barrier’ to higher delay, but with only one parameter, not three.

To configure our evaluations, we used DCTCP (8) and Reno (1) behaviour as the gold standards for the L4S and Classic classes respectively, so we used (11) as the main coupling relation.

The approach is encapsulated by the phrase “Think once to mark, twice to drop”. This concept was first verified by simulation. The results are not shown due to space limitations, but confirmed that for any number of flows with any combination of RTT’s, the steady state throughput was independent of the congestion controller (DCTCP or Reno) if the signal probability was coupled, as in Figure 4, and the window sizes of flows were above four packets.

In addition, further analysis revealed a different sensitivity to multiple bottlenecks. In worst case (all bottlenecks signalling the same probability), DCTCP’s throughput depends on the accumulated probabilities, while Reno’s by the square root of it. In a realistic situation with 2 dominant bottlenecks, DCTCP would have maximum  $\sqrt{2}$  times less throughput.

Coupling signals supports rate equalisation, but as long as there is only one queue, Classic traffic spoils the consistent low queueing latency of DCTCP traffic (unless utilization is sacrificed).

#### 5. DUAL QUEUE FOR LOW LATENCY

To achieve low latency for L4S traffic in the presence of Classic traffic, we still use a coupled AQM, but across two queues as shown in Figure 5. In our experiments, if the ECN field in the IP header is not cleared, we classify the packet into the L4S (L) queue, otherwise into Classic (C). More sophisticated classification might be necessary (see Section 8).

To minimise latency for L4S traffic, we schedule the

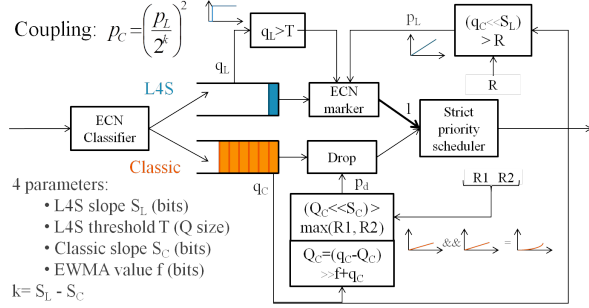


Figure 5: Dual Queue Coupled AQM

L4S queue with strict priority, and the Classic queue only when the L4S queue is empty.

The ECN signal from the L4S queue is coupled to the queuing time of the Classic queue (also known as sojourn, waiting or service time). The coupled feedback ensures that flows share capacity correctly across the two queues. The L4S queue size remains low enough to keep the Classic queue flowing. Policing unresponsive flows is a policy issue that needs to be separate from a basic AQM, but the scheme does need to handle overload<sup>1</sup>.

Whenever there are packets in the Classic queue, the coupled ECN feedback that the L4S queue emits already depends on its own utilisation (via the Classic queue). However, the L4S queue needs to be able to emit ECN signals if L4S load causes the L4S queue itself to grow, particularly if there is no Classic traffic to generate any coupled feedback. For now, in addition to any coupled feedback, the L4S queue applies a shallow step function without any smoothing delay, as used for DCTCP in data centres (see [5, §10] for other possibilities).

When there is traffic in both queues, (11) gives the desired coupling between the drop and marking probabilities in the two classes that should achieve our objective of roughly equal flow rates (other factors being equal). For implementation efficiency, we approximate the denominator by an integer power of 2, giving:

$$p_C = \left( \frac{p_L}{2^k} \right)^2. \quad (12)$$

The resulting coupled AQM needs just 2 parameters for each queue. For the Classic queue:

$S_C$  To convert the current Classic queue sojourn time into a dropping probability in the range  $[0, 1)$  requires a scaling parameter. To make multiplication efficient, we use an integer power of two, so we define the slope of the AQM’s square curve,

<sup>1</sup>A tradeoff needs to be made between complexity and the risk harming Classic flows. It is an operator policy to define what must happen if the service time of the classic queue becomes too big. Actions can include delay based scheduling, common drop, etc...

relating Classic queueing time to drop probability from the Classic queue as  $2^{S_C}$ ;

$f$  To support smoothing the sojourn time of the Classic queue and make multiplication efficient, we will use an integer power of two for the EWMA constant, which we define as  $2^{-f}$ .

For the L4S queue:

$S_L$  As for the Classic queue, we define the slope of the AQM’s marking function as  $2^{S_L}$ ;

$T$  The queue size at which step threshold marking starts in the L4S queue.

The average queue sojourn time  $Q_C$  is calculated every time a packet is dequeued from any of the queues for scheduling. The calculation is done according to (13), with  $q_C$  the sojourn time of the latest packet in the Classic queue and  $f$  the smoothing exponent.

$$Q_C \leftarrow 2^{-f} q_C + (1 - 2^{-f}) Q_C \\ \leftarrow Q_C + (q_C - Q_C) \gg f, \quad (13)$$

where  $\gg$  is a right bit shift.

The queuing time of the Classic queue then drives the marking probability of L4S packets and dropping probability of Classic packets as follows:

$$p_L = 2^{S_L} q_C, \quad (14) \quad p_C = (2^{S_C} Q_C)^2. \quad (15)$$

For either case,  $2^{-S_L}$  or  $2^{-S_C}$  represent the queuing time in the Classic queue at which marking or dropping probability reaches 100%. Assuming a steady state and no under-utilisation,  $Q_C = q_C$ . So, substituting (14) & (15) into (12) we have:

$$k = S_L - S_C. \quad (16)$$

Therefore,  $k$  represents the strength of coupling between the two queues, but it is not an additional parameter.

Using the dual queue AQM with coupled feedback will ensure that the L4S traffic leaves sufficient capacity unused as long as there are packets in the Classic queue. However, the rate ratio between DCTCP and Cubic(Reno) traffic in equation (11) was derived assuming the RTTs of the two types of flow were equivalent. Typically, there would be hardly any delay in the L4S queue, but substantial delay in the Classic queue. The resulting increase in a Classic flow’s RTT could be substantial such that (11) no longer held, then (17) would have to be used instead.

$$\frac{r_{reno}}{r_{dc}} = \frac{1.22}{2} \frac{p_{dc}}{p_{reno}^{1/2}} \frac{T_{dc}}{T_{reno}}, \quad (17)$$

with  $T_{reno}$  and  $T_{dc}$  being the total RTT composed of the delay in their respective queue and a common base RTT  $T_0$  for the rest of the network.

Some value has to be chosen for  $k$  in (12) to relate the rates of L4S and Classic flows, given the possible discrepancy between their RTTs. This is a policy decision for the network operator, which could decide to use:

- $2^k \approx 1.63$  from (11) on the basis that Classic flows have only themselves to blame if the queue they build reduces their rate;
- $2^k \approx 1.19$  on the same basis, but if it judged that most Classic traffic would be in Cubic mode.
- $2^k \approx 1.63(T_0 + q_C)/T_0$  to compensate for the extra queue delay  $q_C$  for exact throughput equality.<sup>2</sup>

For the first two policies respectively  $k = 1$  and  $k = 0$  would be the closest values. In the the last case, the Classic queue delay  $q_C$  depends on the slope of the Classic AQM  $S_C$ . For the slope used in our experiments,  $q_C \approx 4 * T_0$ , therefore  $2^k \approx 1.63 * (4 + 1) \approx 8$ , which implies  $k = 3$ . Of course, given  $T_0$  is bigger towards third party servers, the RTT ratio will become smaller, and any overcompensated factor will be to the disadvantage of L4S flows. In our testbed experiments, we adopted the throughput equality policy to allow an easy comparison of the results. However, there is no implication that this policy is recommended.

The pseudocode below summarises the implementation of the above analysis used for all experiments.

---

**Algorithm 1** Dequeue for Dual Queue Coupled AQM

---

```

1: if LQ.DEQUEUE(pkt) then
2:   if (LQ.LEN() > T) ∨ ((CQ.TIME() << SL) > RND())
   then
3:     MARK(pkt)
4:   end if
5:   RETURN(pkt)    ▷ return the packet and stop here
6: end if
7: while CQ.DEQUEUE(pkt) do
8:   QC += (PKT.TIME() - QC) >> fC    ▷ C EWMA
9:   if (QC << SC) > MAX(RND(), RND()) then
10:    DROP(pkt)    ▷ Squared drop, redo loop
11:   else
12:    RETURN(pkt) ▷ return the packet and stop here
13:   end if
14: end while

```

---

## 6. TESTBED SETUP

We have used a testbed to evaluate the proposed DualQ AQM mechanism in a realistic setting, and to run repeatable experiments in a controlled environment. The testbed was assembled from carrier grade equipment used for testing customer solutions. Figure 6 depicts the testbed, which consists of a classical residential service delivery network composed of Residential Gateway, xDSL DSLAM (DSL Access Multiplexers), BNG (Broadband Network Gateway), Service Routers (SR) and application servers. A residential user’s gateway is connected by VDSL to a DSLAM, which is connected to the BNG through an aggregation network, representing a local ISP or access wholesaler. Traffic is routed to another network representing a global ISP that hosts the application servers and offers breakout to the Internet.

<sup>2</sup>Assuming the queueing delay of the L4S queue is negligible and  $T_0$  is the same for Reno and DCTCP flows.

The client computers in the home network and the application servers at the global ISP are Linux machines, which can be configured to use any TCP variant and start applications and test traffic. The two client-server pairs (A and B) are respectively configured with the same TCP variants and applications.

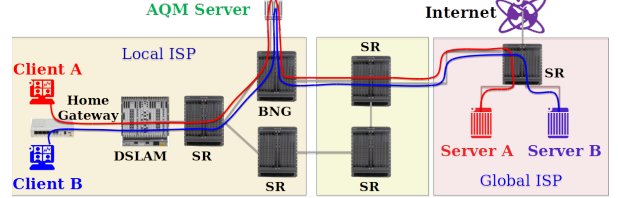


Figure 6: Testbed configuration

Within a BNG, per-customer queues form the leaves of a hierarchical scheduling tree. In a production access network, the BNG is deliberately arranged as the downstream bottleneck for the per customer queues. A Linux server (AQM server) is used to create this bottleneck and to configure the different AQMs that were evaluated. Traffic from the client-server pairs is routed from the BNG through this Linux box so as to simulate the function proposed for the BNG. This server also controls the experiments and captures and analyses the traffic. In practice it would also be important to deploy AQM in the residential gateway, but to minimise side-effects we kept upstream traffic below capacity.

The following setup was used for the evaluations (Section 7): Two client computers were connected to a modem using 100Mbps Fast Ethernet; the xDSL line was configured at 48Mbps downstream and 12Mbps upstream; the links between network elements consisted of at least 1GigE connections. The AQM server at the BNG created a 40Mbps bottleneck, before the configured AQM for the downstream traffic. No bottlenecks were explicitly configured for the upstream traffic. All Linux computers were Ubuntu 14.04 LTS with kernel 3.18.9, which contained the implementation of the TCP variants and AQMs. The base RTT ( $T_0$ ) between the clients and servers was 7 ms, which primarily originated from the xDSL interleaved FEC configuration.

The experiments used DCTCP, Cubic and Reno with their default values. For ECN-Cubic, we additionally enabled TCP ECN negotiation on the relevant client and server. The AQM configurations used the options as described in Table 1, unless otherwise stated.

All	Buffer: 1000 pkt (320ms @40Mbps), ECN
RED	Min thres: 80 pkt (24 ms @40Mbps) Max thres: 240 pkt (72 ms @40Mbps) Burst: 220 pkt (66 ms @40Mbps)
PIE	Target delay: 20 ms, Burst: 100 ms
fq_CoDel	Target delay: 5 ms, Burst: 100 ms
DualQ	$S_C = 1$ , $f = 5$ , $S_L = 4$ , $T = 5$

Table 1: Default parameters for the different AQMs.

## 7. EVALUATION

To assess our primary objective (long term throughput equivalence), we performed experiments on different AQMs with long-running flows. We used steady flows not as an example of a realistic Internet traffic mix, rather as a situation where starvation can typically occur. We also evaluated the AQMs under dynamic load to verify the impact of low latency and loss on completion time of short flows.

Each experiment (lasting 250 s) was performed between client-server pairs A and B, with a specified TCP variant configured on each client-server pair and a specified AQM on the AQM server.

For all long-running flow experiments, each client started 0 to 10 file downloads on its matching server, resulting in 120 combinations. To preserve details, including variability over time, while visualising the overall outcome, we show flow throughput, queue delay, marking and dropping probability in an overview matrix for all 120 combinations.

The row and column labels indicate the TCP variant (C:Cubic, E:ECN-Cubic, D:DCTCP) followed by the number of active flows on the respective client-server pairs. The left matrix label shows the AQM used and the X and Y ranges.

For all dynamic behaviour experiments, 25 load combinations were tested on each client-server pair. Again, the row and column labels (X0+L; X0+H, X1; X1+L; X1+H) indicate TCP variant (X=C,E,D), the number of long-running flows (0 to 1), and the dynamic load level which is an exponential arrival process with 100ms (L=low load) or 10ms (H=high load) average interarrival times (or none), requesting a Pareto distributed download size  $\alpha = 0,9$  with a minimum size of 1KB. Every request opened a new TCP connection, closed by the server after sending the data.

Plotted values of flow or class throughput and marking or dropping probabilities are always measured over each second. Flow completion time plots show a dot per download size and time between opening the connection and receiving the data. A reference line shows the completion time that a perfect lone download would have achieved at full line rate after a 2-RTT handshake. The Queue delay CDF plots the sojourn time of every packet for each traffic class. For flow throughput, the Y range was adapted to locate the expected throughput of the  $N$  dominant flows across the middle of the graph (80/ $N$  Mbps). For example, if 4 Cubic flows compete over the 40Mbps bottleneck, the number of dominant flows is 4, the expected throughput is 10Mbps and the scale 20Mbps. If Cubic flows are starved when 5 Cubic and 2 DCTCP flows compete, then only  $N = 2$  flows are dominant, and the upper limit of Y will be 40Mbps. This results in overall comparable plots, scaling the visualisation optimally.

### 7.1 Long term throughput equivalence

To demonstrate the starvation problem that the DualQ aims to solve, the first evaluation uses DCTCP and Cubic over RED, PIE, fq-CoDel and DualQ AQMs.

The results are plotted in Figures 7 & 8. As expected, the DCTCP flows take most of the available bandwidth from Cubic on the single Q AQMs (RED and PIE).

The queue delay in the RED AQM is very high, but with moderate dropping and marking probability. PIE<sup>3</sup> dropping probability exceeded the 10% limit at which it switches to dropping ECN traffic, and the DCTCP response to drop was incorrectly implemented (see [3] for details).

fq-CoDel seems to handle DCTCP quite well, providing every flow with an almost perfectly stable and equal rate, except when the statistical buffer assignment fails to use a unique buffer per flow. If flows of the same class land in the same Q, the throughput deviation from the equal rate is only 50%. If flows of different classes are assigned to the same buffer, the Cubic flow starves (as in Figure 7 D:8-C:7). This behaviour results in sporadic and hard to reproduce random failure of applications, with potential frustration for users and service support. From a queuing delay perspective, unlike PIE, fq-CoDel is not able to keep the DCTCP flows at its (smaller) target delay, but delay is still low. However, as predicted, the drop probability of fq-CoDel rises quickly with load.

Our DualQ Coupled AQM is able to guarantee equal throughput between DCTCP and Cubic flows. It deviates slightly due to the Classic Q size, which grows on higher load, resulting in less throughput for the Classic flows, and is smaller at lower load, resulting in a higher throughput for the Classic flows. The queuing delay for the L4S traffic is stunningly low—so low that the CDF plots are nearly perfect step functions. In the D:10-C:10 combination, the marking probability approaches 100%. Combinations with larger number of flows revealed a previously unnoticed limit to TCP’s ability to scale to low queuing delays, which needs to be fixed, at least in DCTCP. We have explained this in a separate technical report [3]. Essentially, DCTCP or TCP will override any AQM and increase queuing delay to keep at least 2 segments in flight. For Classic traffic, compared to fq-CoDel, loss levels are kept to reasonable levels by relaxing the delay constraint somewhat (see Section 2.2).

Further experiments with adding a 10 or 20Mbps unresponsive UDP CBR flow<sup>3</sup> also showed an important difference between fq-CoDel and DualQ. fq-CoDel assumes that capping the CBR flow rate to an equal share is always the correct policy. Whereas the DualQ AQM allows applications to determine their flow rate and the responsive flows to share the remaining

<sup>3</sup>See complementary technical report [3] for these plots.



30 or 20 Mbps evenly. For instance, DualQ would support unresponsive multicast TV up to the link capacity whereas fq\_CoDel would divide this by the current number of large flows. Further fq\_CoDel treats a VPN tunnel containing many flows as one, and can prevent background/delay-based flows from yielding their throughput to others. Worth noting that unfortunately, in our fq\_CoDel D:1-C:1 combination, the 20Mbps flow got classified in the same buffer as the CBR flow, resulting in a starving DCTCP flow. fq\_CoDel’s buffer collisions were more frequent than expected.

## 7.2 Evaluation under dynamic load

Figure 9 shows the results for 6 dynamic experiments. We did the (ECN) Cubic experiments as a benchmark to distinguish how much improvement was due to ECN, DCTCP or DualQ. RED<sup>3</sup> and PIE show similar results. fq\_CoDel approximates the perfectly equal share for completion times. The small queuing delay (5ms) just accommodates Cubic’s needs preventing underutilization.

In completion time results for Cubic, we see 2 levels of timeouts: at 1s (due to lost SYN) and 300ms (due to lost FIN). Using ECN-Cubic does not reduce the number of lost SYN/ACK/FIN packets, since they don’t carry the ECN capability in the IP header, the first two in compliance with the ECN standard (and see below regarding FINs). Interpreting the results, we found an anomaly in the Linux implementation. Flows with 1 or 2 packets of data (below 3KB of data) keep experiencing lost packet timeouts of 200ms. The reason is that 2 packets are sent without delay, and a later close connection call results in a separate FIN packet sent without ECN. The current DCTCP implementation uses ECN flags in the IP header of all packets.

In Experiment 4 (PIE), the dots at 300ms (due to drop of final packets) indicate that burst allowance is not effective if other long flows are present. The results for RED<sup>3</sup> were almost identical. Using ECN clearly has an advantage, resulting in shorter completion times as drop is partially avoided. fq\_CoDel’s burst allowance is effective as it works per flow, and ECN provides no significant improvement (Experiment 5). Sporadic occurrences can be attributed to queue collision with an ongoing flow. Also, lower completion times for ECN-Cubic due to less retransmissions of other dropped packets can be identified. ECN has no significant impact on queuing delay.

In Experiment 6 we configured the DualQ AQM with parameters adapted to ECN-Cubic. We halved the L4S slope, additionally applying a squared probability to the ECN-Cubic. As a result, we see significant improvement for the completion times, similar to fq\_CoDel. Also, we see near zero queuing delay for the L4S traffic,

but as a downside, a significant reduction in utilisation when competing with many short flows.

Comparing Experiment 5 to 7, Cubic has more completion time outliers when a long running DCTCP flow is active, probably due to fq\_CoDel’s buffer collisions. Additionally, long running DCTCP flows have a much larger queue, explaining the full utilisation.

Comparing Experiments 5, 7 with 6, 8, we can again conclude that our DualQ AQM very much approximates the fq\_CoDel AQM without the need for flow identification and more complex processing. The main advantage is DualQ’s lower queuing delay for L4S traffic. Compared to Cubic, DCTCP improves utilisation, as it reduces the throughput more appropriately on congestion signals, but can only regain the available capacity incrementally when a short flow ends. One issue with DCTCP also becomes apparent for flows bigger than the initial windows size of 10. As marking probability is much higher, slow start will get consistently prematurely interrupted. A good result is that no slow start overshoot is detected (zero Q delay), but it leads to unnecessarily longer completion times. The outcome suggests that a gradual slow start exit scheme is possible. Again, Dual Q queuing delay is nearly perfect for DCTCP traffic, and even for Classic traffic its delay is nearly as good as fq\_CoDel.

## 8. STANDARDISATION REQUIREMENTS

An identifier will need to be standardised to distinguish L4S and C packets. In our tests we used a cleared ECN field to indicate C packets and L4S otherwise.

The ECN standard [14] currently defines a mark as equivalent to a drop, but discussions have started at the IETF on changing its meaning [16, §5]. It is being questioned whether merely preventing drop offers enough performance improvement for an operator to countenance the cost and risk of deployment.

For those who have managed to get classic ECN widely deployed on servers, moving the goalposts at this stage would be harsh. However, despite widespread server deployment there is no evidence that any public network operator is considering or has deployed ECN, even though it was standardised in 2001. Whereas private data centre operators do redefine the ECN field for the predictable latency of DCTCP.

If the meaning of ECN cannot be changed from “equivalent to drop”, it would be possible to identify the L4S service in another way, e.g. a combination of ECN and Diffserv, or the ECT(1) codepoint. However, the Diffserv codepoint is not preserved end-to-end and it may be argued that the last ECN codepoint should not be burned when the current one is not being used.

The square relationship between L4S marking and Classic drop (Eqn. (12)) would need to be standardised, but it would be better to recommend rather than standardise a value for  $k$ , given differences would not prevent interoperability.

## 9. RELATED WORK

In 2002, Gibbens and Kelly [6] developed a scheme to mark ECN in a priority queue based on the combined length of both queues. However, they were not trying to serve different congestion controllers as in the present work. In 2005 Kuzmanovic [11, §5] presaged the main elements of DCTCP showing that ECN should enable a naïve unsmoothed threshold marking scheme to outperform sophisticated AQMs like the proportional integral (PI) controller. It assumed smoothing at the sender, as earlier proposed by Floyd. Wu et al. [17] investigates a way to incrementally deploy DCTCP within data centres, marking ECN when the temporal queue exceeds a shallow threshold but using standard [14] ECN on end-systems. Kuehlewind et al. [10] showed that DCTCP and Reno could co-exist in the same queue configured with a form of WRED classifying on ECN not Diff-serv. Judd [9] uses Diffserv scheduling to partition data centre switches between DCTCP and classic traffic in a financial data centre scenario. The technical report complementing this paper gives fuller reviews of each of these sources and more [5].

## 10. CONCLUSION

Extensive tests of our novel Coupled Dual Queue AQM within the edge gateway of a broadband access testbed have verified that it elegantly solves the problem of co-existence between traffic like Data Centre TCP (DCTCP) and classic TCP traffic. This means that ISPs can offer a new form of unmanaged Internet service that we call Low Latency Low Loss and Scalable (L4S) without existing ‘Classic’ traffic losing per-flow throughput. In our tests the 99th %-ile queuing delay of L4S was 1 ms; more than an order of magnitude lower than that of PIE or fq-CoDel. We identified self-delay as a new concern for a novel breed of rapidly adaptive applications. L4S will also allow TCP throughput to scale indefinitely, which will otherwise soon become problematic.

In the Linux implementation of our AQM, L4S and Classic together consume fewer instructions per packet than even the simplest form of RED. No flow-ID inspection or per-flow queuing is needed.

Further work is needed to verify the parameter insensitivity of Dual Queue in a wide range of settings and to determine the best AQM for Classic traffic that drops with the square of the probability of L4S traffic. Curvy RED shows promise, and we plan to explore it more fully, particularly with more convexity, but taking care with stability. We will also determine whether a PI controller might be better.

DCTCP needs numerous improvements: fall-back to Reno on loss; loss-resilient feedback [4]; smoothing incoming feedback over the flow’s own RTT estimate; smoothing the exit from slow-start; faster than additive increase; and pacing a fractional window [3].

## 11. REFERENCES

- [1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM ’10, pages 63–74, New York, NY, USA, 2010. ACM.
- [2] B. Briscoe. Insights from Curvy RED (Random Early Detection). Technical report TR-TUB8-2015-003, BT, May 2015. <http://riteproject.eu/publications/>.
- [3] B. Briscoe and K. de Schepper. Scaling TCP’s Congestion Window for Small Round Trip Times. Technical report TR-TUB8-2015-002, BT, May 2015. <http://riteproject.eu/publications/>.
- [4] B. Briscoe, R. Scheffenegger, and M. Kuehlewind. More accurate ECN feedback in TCP. IETF Internet Draft draft-kuehlewind-tcpm-accurate-ecn-03, work in progress, July 2014.
- [5] K. de Schepper, O. Bondarenko, I. Tsang, and B. Briscoe. Data Center to the Home. Technical report, RITE Project, June 2015. <http://riteproject.eu/publications/>.
- [6] R. J. Gibbens and F. P. Kelly. On Packet Marking at Priority Queues. *IEEE Transactions on Automatic Control*, 47(6):1016–1020, June 2002.
- [7] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Operating Systems Review*, 42(5):64–74, July 2008.
- [8] T. Hoeiland-Joergensen, P. McKenney, D. Täht, J. Gettys, and E. Dumazet. Flowqueue-codel. Internet Draft draft-hoeiland-joergensen-aqm-fq-codel, work in progress, June 2014.
- [9] G. Judd. Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 145–157, Oakland, CA, May 2015. USENIX Association.
- [10] M. Kuehlewind, D. Wagner, J. Espinosa, and B. Briscoe. Using data center tcp (dctcp) in the internet. In *Globecom Workshops (GC Wkshps), 2014*, pages 583–588, Dec 2014.
- [11] A. Kuzmanovic. The Power of Explicit Congestion Notification. *Proc. ACM SIGCOMM’05, Computer Communication Review*, 35(4), 2005.
- [12] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP Congestion Avoidance algorithm. *Computer Communication Review*, 27(3), July 1997.
- [13] R. Pan et al. PIE: A lightweight control scheme to address the bufferbloat problem. In *Proc. IEEE Int’l Conf. on High Performance Switching and Routing (HPSR)*, pages 148–155, 2013.
- [14] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN) to IP. RFC 3168 (Proposed Standard), Sept. 2001.
- [15] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP approach for high-speed and long distance networks. In *Proc. IEEE Conference on Computer Communications*, pages 1–12, 2006.
- [16] M. Welzl and G. Fairhurst. The Benefits to Applications of using Explicit Congestion Notification (ECN). Internet Draft draft-welzl-ecn-benefits-02, IETF, Mar. 2015. (Work in Progress).
- [17] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang. Tuning ECN for Data Center Networks. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT ’12*, pages 25–36, New York, NY, USA, 2012. ACM.

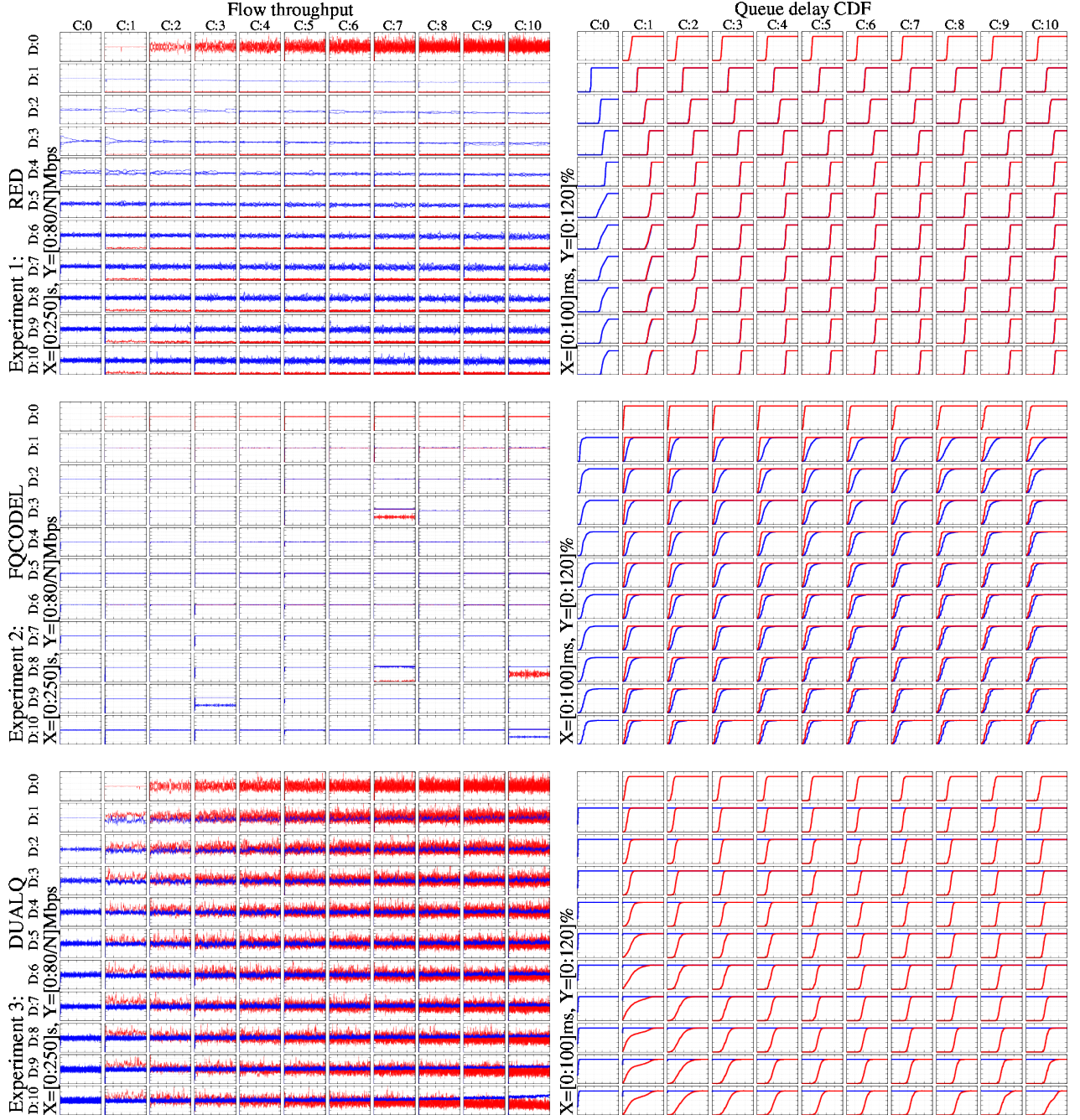


Figure 7: AQM comparison for long flows

Showing the coexistence problem (Exp 1) and potential solutions (Exps 2 &amp; 3).

D: Number of DCTCP flows (blue), C: Number of Cubic flows (red), N: the number of expected dominant flows.

Note: With Dual Q, the queue delay CDFs for DCTCP (blue) are hard to see, because they are all nearly perfect step-functions.



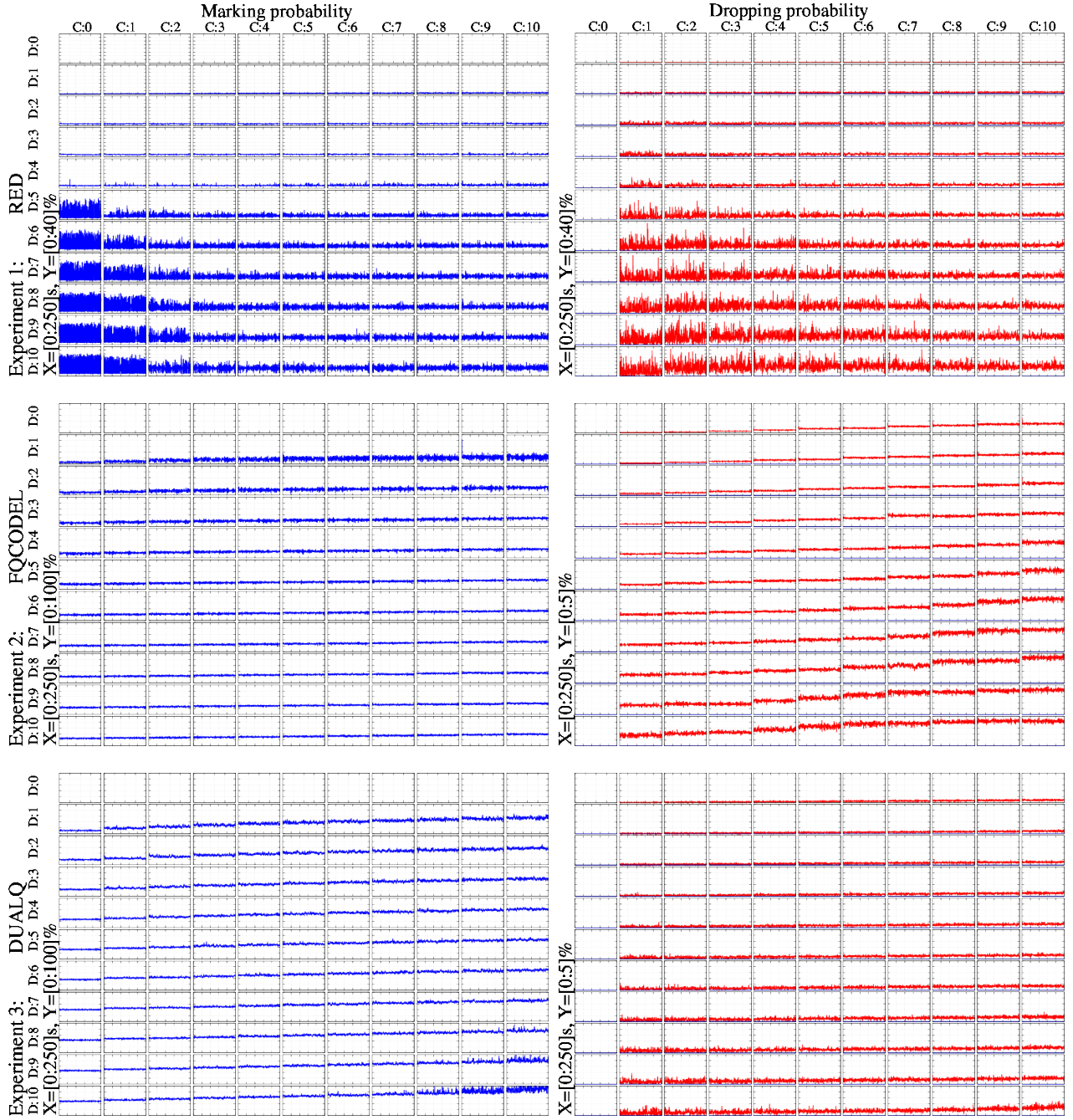


Figure 8: AQM comparison for long flows (cont.)  
 D: Number of DCTCP flows (blue), C: Number of Cubic flows (red).



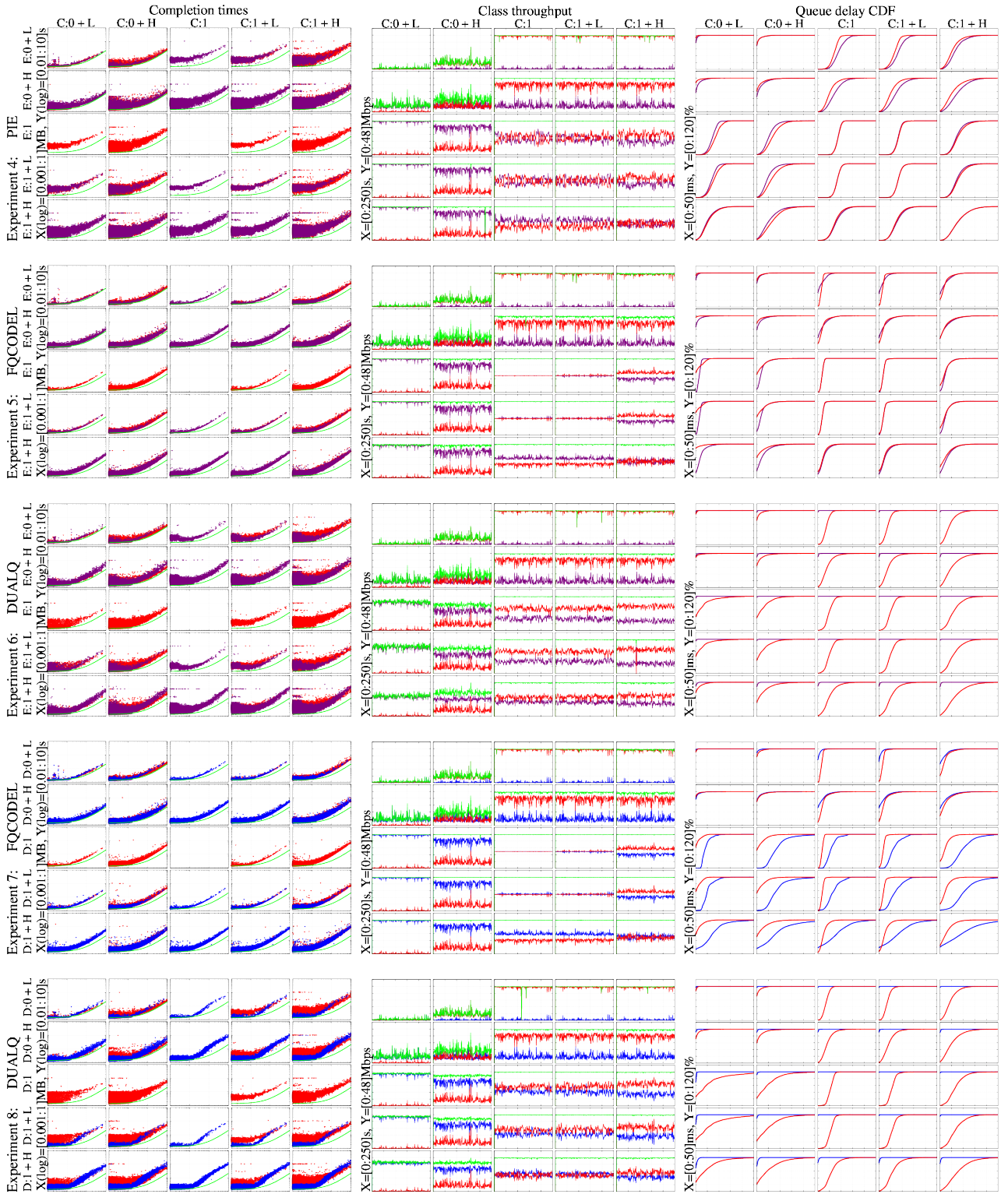


Figure 9: Dynamic load for different AQMs for ECN-Cubic & Cubic (Exps 4–6) or DCTCP & Cubic (Exps 7 & 8). E: Number of ECN-Cubic flows (purple), D: Number of DCTCP flows (blue), C: Number of Cubic flows (red), Green = total throughput, showing utilisation. L: 100 ms exponential load, H: 10 ms exponential load. Note: With Dual Q, the queue delay CDFs for DCTCP (blue) are hard to see, because they are all nearly perfect step-functions.

Active Queue Management (aqm)  
Internet-Draft  
Intended status: Standards Track  
Expires: February 8, 2016

K. De Schepper  
Bell Labs  
B. Briscoe, Ed.  
Independent  
O. Bondarenko  
Simula Research Lab  
I. Tsang  
Bell Labs  
August 07, 2015

DualQ Coupled AQM for Low Latency, Low Loss and Scalable Throughput  
draft-briscoe-aqm-dualq-coupled-00

#### Abstract

Data Centre TCP (DCTCP) was designed to provide predictably low queuing latency, near-zero loss, and throughput scalability using explicit congestion notification (ECN) and an extremely simple marking behaviour on switches. However, DCTCP does not co-exist with existing TCP traffic---throughput starves. So, until now, DCTCP could only be deployed where a clean-slate environment could be arranged, such as in private data centres. This specification defines 'DualQ Coupled Active Queue Management (AQM)' to allow scalable congestion controls like DCTCP to safely co-exist with classic Internet traffic. The Coupled AQM ensures that a flow runs at about the same rate whether it uses DCTCP or TCP Reno/Cubic, but without inspecting transport layer flow identifiers. When tested in a residential broadband setting, DCTCP achieved sub-millisecond average queuing delay and zero congestion loss under a wide range of mixes of DCTCP and 'Classic' broadband Internet traffic, without compromising the performance of the Classic traffic. The solution also reduces network complexity and eliminates network configuration.

#### Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

Internet-Draft

DualQ Coupled AQM

August 2015

This Internet-Draft will expire on February 8, 2016.

#### Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4.e](#) of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

#### Table of Contents

1. Introduction . . . . .	2
1.1. Problem and Scope . . . . .	2
1.2. Terminology . . . . .	4
1.3. Features . . . . .	5
2. DualQ Coupled AQM Algorithm . . . . .	6
2.1. Coupled AQM . . . . .	6
2.2. Dual Queue . . . . .	7
2.3. Traffic Classification . . . . .	7
2.4. Normative Requirements . . . . .	9
3. IANA Considerations . . . . .	10
4. Security Considerations . . . . .	10
4.1. Overload Handling . . . . .	10
5. Acknowledgements . . . . .	11
6. References . . . . .	11
6.1. Normative References . . . . .	11
6.2. Informative References . . . . .	11
<a href="#">Appendix A.</a> Example DualQ Coupled Algorithm . . . . .	14
<a href="#">Appendix B.</a> Guidance on Controlling Throughput Equivalence . . .	20
<a href="#">Appendix C.</a> DCTCP Safety Enhancements . . . . .	21
Authors' Addresses . . . . .	22

#### 1. Introduction

##### 1.1. Problem and Scope

Latency is becoming the critical performance factor for many (most?) applications on the public Internet, e.g. Web, voice, conversational video, gaming and finance apps. In the developed world, further increases in access network bit-rate offer diminishing returns,

Internet-Draft

DualQ Coupled AQM

August 2015

whereas latency is still a multi-faceted problem. In the last decade or so, much has been done to reduce propagation time by placing caches or servers closer to users. However, queuing remains a major component of latency.

The Diffserv architecture provides Expedited Forwarding [[RFC3246](#)], so that low latency traffic can jump the queue of other traffic. However, on access links dedicated to individual sites (homes, small enterprises or mobile devices), often all traffic at any one time will be latency-sensitive. Then Diffserv is of little use. Instead, we need to remove the causes of any unnecessary delay.

The bufferbloat project has shown that excessively-large buffering ('bufferbloat') has been introducing significantly more delay than the underlying propagation time. These delays appear only intermittently--only when a capacity-seeking (e.g. TCP) flow is long enough for the queue to fill the buffer, making every packet in other flows sharing the buffer sit through the queue.

Active queue management (AQM) was originally developed to solve this problem (and others). Unlike Diffserv, AQM controls latency for `_all_` traffic in a class. In general, AQMs introduce an increasing level of discard from the buffer the longer the queue persists above a shallow threshold. This gives sufficient signals to capacity-seeking (aka. greedy) flows to keep the buffer empty for its intended purpose: absorbing bursts. However, RED [[RFC2309](#)] and other algorithms from the 1990s were sensitive to their configuration and hard to set correctly. So, AQM was not widely deployed. More recent state-of-the-art AQMs, e.g. fq\_CoDel [[I-D.ietf-aqm-fq-codel](#)], PIE [[I-D.ietf-aqm-pie](#)], Adaptive RED [[ARED01](#)], define the threshold in time not bytes, so it is invariant for different link rates.

It seems that further changes to the network alone will now yield diminishing returns. Data Centre TCP (DCTCP [[I-D.bensley-tcpm-dctcp](#)]) teaches us that a small but radical change to TCP is needed to cut two major outstanding causes of queuing delay variability:

1. the 'sawtooth' varying rate of TCP itself;
2. the smoothing delay deliberately introduced into AQMs to permit bursts without triggering losses.

The former causes a flow's round trip time (RTT) to vary from about 1 to 2 times the base RTT between the machines in question. The latter delays the system's response to change by a worst-case (transcontinental) RTT, which could be hundreds of times the actual RTT of typical traffic from localized CDNs.

Internet-Draft

DualQ Coupled AQM

August 2015

Latency is not our only concern:

3. It was known when TCP was first developed that it would not scale to high bandwidth-delay products.

Given regular broadband bit-rates over WAN distances are already [[RFC3649](#)] beyond the scaling range of 'classic' TCP Reno, 'less unscalable' Cubic [[I-D.zimmermann-tcpm-cubic](#)] and Compound [[I-D.sridharan-tcpm-ctcp](#)] variants of TCP have been successfully deployed. However, these are now approaching their scaling limits. Unfortunately, fully scalable TCPs such as DCTCP cause 'classic' TCP to starve itself, which is why they have been confined to private data centres or research testbeds (until now).

This document specifies a 'DualQ Coupled AQM' that solves the problem of coexistence between DCTCP and classic flows, without having to inspect flow identifiers. The AQM is not like flow-queuing approaches [[I-D.ietf-aqm-fq-code1](#)] that classify packets by flow identifier into numerous separate queues in order to isolate sparse flows from the higher latency in the queues assigned to heavier flow. In contrast, the AQM exploits the behaviour of scalable congestion controls like DCTCP so that every packet in every flow sharing the queue for DCTCP-like traffic can be served with very low latency.

The AQM needs fewer operations per packet than RED uses. Also, no network configuration is needed for a wide range of scenarios where the range of RTTs is typical for the public Internet. Therefore it is believed the Coupled AQM would be applicable and easy to deploy in all types of buffers; buffers in cost-reduced mass-market residential equipment; buffers in end-system stacks; buffers in carrier-scale equipment including remote access servers, routers, firewalls and Ethernet switches; buffers in network interface cards, buffers in virtualized network appliances, hypervisors, and so on.

The supporting paper [[DCTtH15](#)] gives the full rationale for the AQM's design, both discursively and in more precise mathematical form.

## 1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [[RFC2119](#)]. In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying [RFC-2119](#) significance.

Internet-Draft

DualQ Coupled AQM

August 2015

The DualQ Coupled AQM uses two queues for two services. Each of the following terms identifies both the service and the queue that provides the service:

Classic (denoted by subscript C): The 'Classic' service is intended for all the behaviours that currently co-exist with TCP Reno (TCP Cubic, Compound, SCTP, etc).

Low-Latency, Low-Loss and Scalable (L4S, denoted by subscript L): The 'L4S' service is intended for DCTCP traffic but it is also more general--it will allow a set of congestion controls with similar scaling properties to DCTCP (e.g. Relentless [Mathis09]) to evolve.

Either service can cope with a proportion of unresponsive or less-responsive traffic as well (e.g. DNS, VoIP, etc).

### 1.3. Features

The AQM couples marking and/or dropping across the two queues such that a flow will get roughly the same throughput whichever it uses. Therefore both queues can feed into the full capacity of a link and no rates need to be configured for the queues. The L4S queue enables scalable congestion controls like DCTCP to give stunningly low and predictably low latency, without compromising the performance of competing 'Classic' Internet traffic. Thousands of tests have been conducted in a typical fixed residential broadband setting. Typical experiments used a base round trip delay of 7ms between the data centre and home network, and large amounts of background traffic in both queues. For every L4S packet, the AQM kept the 99th percentile of queuing delay to about 1ms, and no losses at all were introduced by the AQM. Details of the extensive experiments will be made available [DCTH15].

Subjective testing was also conducted using a demanding panoramic interactive video application run over a stack with DCTCP enabled and deployed on the testbed. Each user could pan or zoom their own high definition (HD) sub-window of a larger video scene from a football match. Even though the user was also downloading large amounts of L4S and Classic data, latency was so low that the picture appeared to stick to their finger on the touchpad (all the L4S data achieved the same ultra-low latency). With an alternative AQM, the video noticeably lagged behind the finger gestures.

Unlike Diffserv Expedited Forwarding, the L4S queue does not have to be limited to a small proportion of the link capacity in order to achieve low delay. The L4S queue can be filled with a heavy load of capacity-seeking flows like DCTCP and still achieve low delay. The

Internet-Draft

DualQ Coupled AQM

August 2015

L4S queue does not rely on the presence of other traffic in the Classic queue that can be 'overtaken'. It gives low latency to L4S traffic whether or not there is Classic traffic, and the latency of Classic traffic does not suffer when a proportion of the traffic is L4S. The two queues are only necessary because DCTCP-like flows cannot keep latency predictably low and keep utilization high if they are mixed with legacy TCP flows,

The experiments used the Linux implementation of DCTCP that is deployed in private data centres, without any modification despite its known deficiencies. Nonetheless, certain modifications will be necessary before DCTCP is safe to use on the Internet, which are recorded for now in [Appendix C](#). However, the focus of this specification is to get the network service in place. Then, without any management intervention, applications can exploit it by migrating to scalable controls like DCTCP, which can then evolve `_while_` their benefits are being enjoyed by everyone on the Internet.

## 2. DualQ Coupled AQM Algorithm

There are two main aspects to the algorithm:

- o the Coupled AQM that addresses throughput equivalence between Classic (e.g. Reno, Cubic) flows and L4S (e.g. DCTCP) flows
- o the Dual Queue structure that provides latency separation for L4S flows to isolate them from the typically large Classic queue.

### 2.1. Coupled AQM

In the 1990s, the 'TCP formula' was derived for the relationship between TCP's congestion window, `cwnd`, and its drop probability, `p`. To a first order approximation, `cwnd` of TCP Reno is inversely proportional to the square root of `p`. TCP Cubic implements a Reno-compatibility mode, which is the only relevant mode for typical RTTs under 20ms, while the throughput of a single flow is less than about 500Mb/s. Therefore we can assume that Cubic traffic behaves similar to Reno (but with a slightly different constant of proportionality), and we shall use the term 'Classic' for the collection of Reno and Cubic in Reno mode.

In our supporting paper [[DCTH15](#)], we derive the equivalent rate equation for DCTCP, for which `cwnd` is inversely proportional to `p` (not the square root), where in this case `p` is the ECN marking probability. DCTCP is not the only congestion control that behaves like this, so we use the term 'L4S' traffic for all similar behaviour.



Internet-Draft

DualQ Coupled AQM

August 2015

In order to make a DCTCP flow run at roughly the same rate as a Reno TCP flow (all other factors being equal), we make the drop probability for Classic traffic,  $p_C$  distinct from the marking probability for L4S traffic,  $p_L$  (in contrast to [RFC3168](#) which requires them to be the same). We make the Classic drop probability  $p_C$  proportional to the square of the L4S marking probability  $p_L$ . This is because we need to make the Reno flow rate equal the DCTCP flow rate, so we have to square the square root of  $p_C$  in the Reno rate equation to make it the same as the straight  $p_L$  in the DCTCP rate equation.

There is a really simple way to implement the square of a probability - by testing the queue against two random numbers not one. This is the approach adopted in [Appendix A](#).

Stating this as a formula, the relation between Classic drop probability,  $p_C$ , and L4S marking probability,  $p_L$  needs to take the form:

$$p_C = ( p_L / 2^k )^2 \quad (1)$$

where  $2^k$  is the constant of proportionality, which is expressed as a power of 2 so that implementations can avoid costly division by shifting  $p_L$  by  $k$  bits to the right.

## 2.2. Dual Queue

Classic traffic builds a large queue, so a separate queue is provided for L4S traffic, and it is scheduled with strict priority. Nonetheless, coupled marking ensures that giving priority to L4S traffic still leaves the right amount of spare scheduling time for Classic flows to each get equivalent throughput to DCTCP flows (all other factors such as RTT being equal). The algorithm achieves this without having to inspect flow identifiers.

## 2.3. Traffic Classification

Both the Coupled AQM and DualQ mechanisms need an identifier to distinguish L4S and C packets, which will need to be standardized. In our tests we used a cleared ECN field to indicate C packets and L4S otherwise. The ECN specification [[RFC3168](#)] currently defines a mark as equivalent to a drop. However, it says

"An environment where all end nodes were ECN-Capable could allow new criteria to be developed for setting the CE codepoint, and new congestion control mechanisms for end-node reaction to CE packets. However, this is a research issue, and as such is not addressed in this document."



Internet-Draft

DualQ Coupled AQM

August 2015

and [RFC4774]} gives valid ways to alter ECN's semantics without harming interoperability.

Since publication in 2001, deployment of RFC3168 ECN has been dogged by bugs and misunderstandings. In recent years RFC3168 ECN has been deployed quite successfully on servers [ECN\_Deploy], and until recently it was deployed but not enabled on a fair proportion of user machines. Recently one major developer of client devices has configured ECN on-by-default in its beta releases. However although some network equipment vendors and developers have implemented ECN, there is little evidence that any public network operator is considering or has deployed ECN-capable AQMs on network equipment yet.

A number of private data centre operators have deployed ECN, but not RFC3168 ECN. Instead, they are using DCTCP to get predictable ultra-low latency, and they are either ensuring that there is no non-DCTCP traffic [I-D.bensley-tcpm-dctcp], or they are segregating such traffic from DCTCP using Diffserv [DCTCP\_Pitfalls]. The RFC3168 approach merely prevents drop, whereas the DCTCP approach provides scalable throughput and ultra-low latency as well as avoiding drop. Consequently it has been questioned whether the RFC3168 approach offers enough performance improvement for an operator to countenance the cost and risk of deployment. There has been some discussions at the IETF on changing the meaning of an ECN mark to move towards the DCTCP approach. The performance results from our experiments with DCTCP for broadband residential users are certainly significant enough to warrant interest from operators.

For those who have managed to get classic ECN widely deployed on end-systems, moving the goalposts at this stage would be harsh. If the meaning of ECN cannot be changed from "equivalent to drop", it would be possible to identify the L4S service in another way, e.g. a combination of ECN and Diffserv, or using the ECT(1) codepoint. The Diffserv codepoint is not ideal, because L4S is an end-to-end service and a DSCP is not preserved end-to-end. However, combining ECN and Diffserv may be sufficient for initial deployment, while confined to controlled sets of networks, during which time any users of classic ECN can upgrade to L4S. The ECT(1) codepoint is perhaps less ideal, because two separate uses of ECN really need two codepoints each, and anyway it could be argued that the last ECN codepoint should not be burned when the current one is not being used.

This draft does not currently recommend an approach for identifying for the L4S service, which is initially left open for discussion within the IETF.

Internet-Draft

DualQ Coupled AQM

August 2015

#### 2.4. Normative Requirements

In the Dual Queue, L4S packets **MUST** be given priority over Classic, although strict priority **MAY** not be appropriate.

All L4S traffic **MUST** be ECN-capable, although some Classic traffic **MAY** also be ECN-capable.

Whatever identifier is used for L4S traffic, it will still be necessary to agree on the meaning of an ECN marking on L4S traffic, relative to a drop of Classic traffic. In order to prevent starvation of Classic traffic by scalable L4S traffic (e.g. DCTCP) the drop probability of Classic traffic **MUST** be proportional to the square of the marking probability of L4S traffic, In other words, the power to which  $p_L$  is raised in Eqn. (1) **MUST** be 2.

The constant of proportionality,  $k$ , in Eqn (1) determines the relative flow rates of Classic and L4S flows when the AQM concerned is the bottleneck (all other factors being equal).  $k$  does not have to be standardized because differences do not prevent interoperability. However,  $k$  has to take some value, and each operator can make that choice.

A value of  $k=0$  is **RECOMMENDED** as the default for public Internet access networks, assuming the DCTCP algorithm remains similar to that in [I-D.bensley-tcpm-dctcp]. Nonetheless choice of  $k$  is a matter of operator policy, and operators **MAY** choose a different value using Table 1 and the guidelines in [Appendix B](#).

Typically, access network operators isolate customers from each other with some form of layer-2 multiplexing (TDM in DOCSIS, CDMA in 3G) or L3 scheduling (WRR in broadband), rather than relying on TCP to share capacity between customers [RFC0970]. In such cases, the choice of  $k$  will solely affect relative flow rates within the customer's access capacity, not between customers. Also,  $k$  would not affect rates of small flows, nor long flows at any times when they are all Classic or all L4S.

An example DualQ Coupled AQM algorithm is given in [Appendix A](#). Marking and dropping in each queue is based on an AQM called Curvy RED, which is intended to improve on RED, PIE and CoDel. We have found that Curvy RED offers good performance, requires less operations per packet than RED and is insensitive to configuration. Nonetheless, it would be possible to control each queue with an alternative AQM, as long as the above normative requirements (those expressed in capitals) are observed, which are intended to be independent of the specific AQM.

Internet-Draft

DualQ Coupled AQM

August 2015

{ToDo: Add management and monitoring requirements}

### 3. IANA Considerations

This specification contains no IANA considerations.

### 4. Security Considerations

#### 4.1. Overload Handling

Where the interests of users or flows might conflict, it could be necessary to police traffic to isolate any harm to performance. This is a policy issue that needs to be separable from a basic AQM, but the scheme does need to handle overload. A trade-off needs to be made between complexity and the risk of either class harming the other. It is an operator policy to define what must happen if the service time of the classic queue becomes too great. In the following subsections three optional non-exclusive overload protections are defined. Their objective is for the overload behaviour of the DualQ AQM to be similar to a single queue AQM. Other overload protections can be envisaged:

**Minimum throughput service:** By replacing the priority scheduler with a weighted round robin scheduler, a minimum throughput service can be guaranteed for Classic traffic. Typically the scheduling weight of the Classic queue will be small (e.g. 5%) to avoid interference with the coupling but big enough to avoid complete starvation of Classic traffic.

**Drop on overload:** On severe overload, e.g. due to non responsive traffic, queues will typically overflow and packet drop will be unavoidable when the queues reach their limits. The drop-limit of each queue should be configured by specifying the maximum supported load and determining the expected maximum size of each queue when that load is separately applied to each queue. The Classic queue limit will typically be larger than the L4S queue limit. Overflow of one traffic type will automatically result in drop in its respective queue. Both traffic types will get a high congestion signal, due to the coupled marking, which will result in similar starvation of responsive traffic in both queues. Thus, the behaviour will be like a single queue AQM. To further improve the arrival fairness of a single queue an extra overall AQM limit can be applied, which is a limit to the sum of both queues. To be effective, it should be configured to be less than the sum of the limits of both queues, but greater than the maximum individual queue limit. It ensures that the drop probability of unresponsive traffic will be independent of its traffic type.

Internet-Draft

DualQ Coupled AQM

August 2015

Delay on overload: To control milder overload of responsive traffic, particularly when close to the maximum congestion signal, delay can be used as an alternative congestion control mechanism. The Dual Queue Coupled AQM can be made to behave like a single FIFO queue with differentiated service times by replacing the priority scheduler with a very simple "biased longest sojourn time first scheduler". The bias is defined as a maximum sojourn time difference ( $T_m$ ) between the Classic and L4S packets. The scheduler adds  $T_m$  to the sojourn time of the next L4S packet, before comparing it with the timestamp of the next Classic packet, then it selects the packet with the greater adjusted sojourn time. This time shifted FIFO queue behaves just like a single FIFO queue under moderate and high overload.

## 5. Acknowledgements

Thanks to Anil Agarwal for detailed review comments and suggestions on how to make our explanation clearer.

The authors' contributions are part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed here are solely those of the authors.

## 6. References

### 6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

### 6.2. Informative References

[ARED01] Floyd, S., Gummadi, R., and S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management", ACIRI Technical Report , August 2001, <<http://www.icir.org/floyd/red.html>>.

[CoDel] Nichols, K. and V. Jacobson, "Controlling Queue Delay", ACM Queue 10(5), May 2012, <<http://queue.acm.org/issuedetail.cfm?issue=2208917>>.

Internet-Draft

DualQ Coupled AQM

August 2015

## [CRED\_Insights]

Briscoe, B., "Insights from Curvy RED (Random Early Detection)", BT Technical Report TR-TUB8-2015-003, July 2015,  
<[http://www.bobbriscoe.net/projects/latency/credi\\_tr.pdf](http://www.bobbriscoe.net/projects/latency/credi_tr.pdf)>.

## [DCTCP\_Pitfalls]

Judd, G., "Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter", 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15) 145--157, May 2015,  
<<http://blogs.usenix.org/conference/nsdi15/technical-sessions/presentation/judd>>.

## [DCTH15]

De Schepper, K., Bondarenko, O., Briscoe, B., and I. Tsang, "'Data Centre to the Home': Ultra-Low Latency for All", 2015, <[http://www.bobbriscoe.net/projects/latency/dctth\\_preprint.pdf](http://www.bobbriscoe.net/projects/latency/dctth_preprint.pdf)>.

(Under submission)

## [ECN\_Deploy]

Trammell, B., Kuehlewind, M., Boppart, D., Learmonth, I., Fairhurst, G., and R. Scheffenegger, "Enabling Internet-Wide Deployment of Explicit Congestion Notification", Proc Passive & Active Measurement (PAM'15) Conference, 2015,  
<<http://ecn.ethz.ch/ecn-pam15.pdf>>.

## [I-D.bensley-tcpm-dctcp]

Bensley, S., Eggert, L., Thaler, D., Balasubramanian, P., and G. Judd, "Microsoft's Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters", [draft-bensley-tcpm-dctcp-05](#) (work in progress), July 2015.

## [I-D.ietf-aqm-fq-codel]

Hoeiland-Joergensen, T., McKenney, P., dave.taht@gmail.com, d., Gettys, J., and E. Dumazet, "FlowQueue-Codel", [draft-ietf-aqm-fq-codel-01](#) (work in progress), July 2015.

## [I-D.ietf-aqm-pie]

Pan, R., Natarajan, P., Baker, F., and G. White, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem", [draft-ietf-aqm-pie-01](#) (work in progress), March 2015.

Internet-Draft

DualQ Coupled AQM

August 2015

- [I-D.ietf-tcpm-accecn-reqs]  
Kuehlewind, M., Scheffenegger, R., and B. Briscoe,  
"Problem Statement and Requirements for a More Accurate  
ECN Feedback", [draft-ietf-tcpm-accecn-reqs-08](#) (work in  
progress), March 2015.
- [I-D.sridharan-tcpm-ctcp]  
Sridharan, M., Tan, K., Bansal, D., and D. Thaler,  
"Compound TCP: A New TCP Congestion Control for High-Speed  
and Long Distance Networks", [draft-sridharan-tcpm-ctcp-02](#)  
(work in progress), November 2008.
- [I-D.zimmermann-tcpm-cubic]  
Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and  
R. Scheffenegger, "CUBIC for Fast Long-Distance Networks",  
[draft-zimmermann-tcpm-cubic-01](#) (work in progress), April  
2015.
- [Mathis09]  
Mathis, M., "Relentless Congestion Control", PFLDNeT'09 ,  
May 2009, <[http://www.hpcc.jp/pfldnet2009/  
Program\\_files/1569198525.pdf](http://www.hpcc.jp/pfldnet2009/Program_files/1569198525.pdf)>.
- [RFC0970] Nagle, J., "On Packet Switches With Infinite Storage",  
[RFC 970](#), DOI 10.17487/RFC0970, December 1985,  
<<http://www.rfc-editor.org/info/rfc970>>.
- [RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering,  
S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G.,  
Partridge, C., Peterson, L., Ramakrishnan, K., Shenker,  
S., Wroclawski, J., and L. Zhang, "Recommendations on  
Queue Management and Congestion Avoidance in the  
Internet", [RFC 2309](#), DOI 10.17487/RFC2309, April 1998,  
<<http://www.rfc-editor.org/info/rfc2309>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition  
of Explicit Congestion Notification (ECN) to IP",  
[RFC 3168](#), DOI 10.17487/RFC3168, September 2001,  
<<http://www.rfc-editor.org/info/rfc3168>>.
- [RFC3246] Davie, B., Charny, A., Bennet, J., Benson, K., Le Boudec,  
J., Courtney, W., Davari, S., Firoiu, V., and D.  
Stiliadis, "An Expedited Forwarding PHB (Per-Hop  
Behavior)", [RFC 3246](#), DOI 10.17487/RFC3246, March 2002,  
<<http://www.rfc-editor.org/info/rfc3246>>.

Internet-Draft

DualQ Coupled AQM

August 2015

- [RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", [RFC 3649](#), DOI 10.17487/RFC3649, December 2003, <<http://www.rfc-editor.org/info/rfc3649>>.
- [RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", [BCP 124](#), [RFC 4774](#), DOI 10.17487/RFC4774, November 2006, <<http://www.rfc-editor.org/info/rfc4774>>.
- [TCP-sub-mss-w]  
Briscoe, B. and K. De Schepper, "Scaling TCP's Congestion Window for Small Round Trip Times", BT Technical Report TR-TUB8-2015-002, May 2015, <<http://www.bobbriscoe.net/projects/latency/sub-mss-w.pdf>>.

#### [Appendix A.](#) Example DualQ Coupled Algorithm

As a concrete example, the pseudocode below gives the DualQ Coupled AQM algorithm we used in testing. Although we designed the AQM to be efficient in integer arithmetic, to aid understanding it is first given using real-number arithmetic. Then, one possible optimization for integer arithmetic is given, also in pseudocode. To aid comparison, the line numbers are kept in step between the two by using letter suffixes where the longer code needs extra lines.

Internet-Draft

DualQ Coupled AQM

August 2015

```

1:  dualq_dequeue(lq, cq) {  % Couples L4S & Classic queues, lq & cq
2:      if ( lq.dequeue(pkt) ) {
3a:         p_L = cq.sec() / 2^S_L
3b:         if ( lq.byt() > T )
3c:             mark(pkt)
3d:         elif ( p_L > maxrand(U) )
4:             mark(pkt)
5:         return(pkt)          % return the packet and stop here
6:     }
7:     while ( cq.dequeue(pkt) ) {
8a:         alpha = 2^(-f_C)
8b:         Q_C = alpha * pkt.sec() + (1-alpha)* Q_C % Classic Q EWMA
9a:         sqrt_p_C = Q_C / 2^S_C
9b:         if ( sqrt_p_C > maxrand(2*U) )
10:             drop(pkt)          % Squared drop, redo loop
11:         else
12:             return(pkt)        % return the packet and stop here
13:     }
14:     return(NULL)             % no packet to dequeue
15: }

16: maxrand(u) {                % return the max of u random numbers
17:     maxr=0
18:     while (u-- > 0)
19:         maxr = max(maxr, rand())          % 0 <= rand() < 1
20:     return(maxr)
21: }

```

Figure 1: Example Dequeue Pseudocode for Coupled DualQ AQM

Packet classification code is not shown, as it is no different from regular packet classification. Potential classification schemes are discussed in [Section 2](#). Overload protection code will be included in a future draft {ToDo}.

At the outer level, the structure of `dualq_dequeue()` implements strict priority scheduling. The code is written assuming the AQM is applied on dequeue (Note 1). Every time `dualq_dequeue()` is called, the if-block in lines 2-6 determines whether there is an L4S packet to dequeue by calling `lq.dequeue(pkt)`, and otherwise the while-block in lines 7-13 determines whether there is a Classic packet to dequeue, by calling `cq.dequeue(pkt)`. (Note 2)

In the lower priority Classic queue, a while loop is used so that, if the AQM determines that a classic packet should be dropped, it continues to test for classic packets deciding whether to drop each until it actually forwards one. Thus, every call to `dualq_dequeue()`



Internet-Draft

DualQ Coupled AQM

August 2015

returns one packet if at least one is present in either queue, otherwise it returns NULL at line 14. (Note 3)

Within each queue, the decision whether to drop or mark is taken as follows (to simplify the explanation, it is assumed that  $U=1$ ):

**L4S:** If the test at line 2 determines there is an L4S packet to dequeue, the tests at lines 3a and 3c determine whether to mark it. The first is a simple test of whether the L4S queue (`lq.by()` in bytes) is greater than a step threshold  $T$  in bytes (Note 4). The second test is similar to the random ECN marking in RED, but with the following differences: i) the marking function does not start with a plateau of zero marking until a minimum threshold, rather the marking probability starts to increase as soon as the queue is positive; ii) marking depends on queuing time, not bytes, in order to scale for any link rate without being reconfigured; iii) marking of the L4S queue does not depend on itself, it depends on the queuing time of the `_other_` (Classic) queue, where `cq.sec()` is the queuing time of the packet at the head of the Classic queue (zero if empty); iv) marking depends on the instantaneous queuing time (of the other queue), not a smoothed average; v) the queue is compared with the maximum of  $U$  random numbers (but if  $U=1$ , this is the same as the single random number used in RED).

Specifically, in line 3a the marking probability  $p_L$  is set to the Classic queueing time `qc.sec()` in seconds divided by the L4S scaling parameter  $2^{S_L}$ , which represents the queuing time (in seconds) at which marking probability would hit 100%. Then in line 3d (if  $U=1$ ) the result is compared with a uniformly distributed random number between 0 and 1, which ensures that marking probability will linearly increase with queueing time. The scaling parameter is expressed as a power of 2 so that division can be implemented as a right bit-shift ( $>>$ ) in line 3 of the integer variant of the pseudocode (Figure 2).

**Classic:** If the test at line 7 determines that there is at least one Classic packet to dequeue, the test at line 9b determines whether to drop it. But before that, line 8b updates  $Q_C$ , which is an exponentially weighted moving average (Note 5) of the queuing time in the Classic queue, where `pkt.sec()` is the instantaneous queueing time of the current Classic packet and  $\alpha$  is the EWMA constant for the classic queue. In line 8a,  $\alpha$  is represented as an integer power of 2, so that in line 8 of the integer code the division needed to weight the moving average can be implemented by a right bit-shift ( $>> f_C$ ).

Internet-Draft

DualQ Coupled AQM

August 2015

Lines 9a and 9b implement the drop function. In line 9a the averaged queuing time  $Q_C$  is divided by the Classic scaling parameter  $2^{S_C}$ , in the same way that queuing time was scaled for L4S marking. This scaled queuing time is given the variable name  $\text{sqrt\_p\_C}$  because it will be squared to compute Classic drop probability, so before it is squared it is effectively the square root of the drop probability. The squaring is done by comparing it with the maximum out of two random numbers (assuming  $U=1$ ). Comparing it with the maximum out of two is the same as the logical 'AND' of two tests, which ensures drop probability rises with the square of queuing time (Note 6). Again, the scaling parameter is expressed as a power of 2 so that division can be implemented as a right bit-shift in line 9 of the integer pseudocode.

The marking/dropping functions in each queue (lines 3 & 9) are two cases of a new generalization of RED called Curvy RED, motivated as follows. When we compared the performance of our AQM with  $\text{fq\_CoDel}$  and PIE, we came to the conclusion that their goal of holding queuing delay to a fixed target is misguided [[CRED\\_Insights](#)]. As the number of flows increases, if the AQM does not allow TCP to increase queuing delay, it has to introduce abnormally high levels of loss. Then loss rather than queuing becomes the dominant cause of delay for short flows, due to timeouts and tail losses.

Curvy RED constrains delay with a softened target that allows some increase in delay as load increases. This is achieved by increasing drop probability on a convex curve relative to queue growth (the square curve in the Classic queue, if  $U=1$ ). Like RED, the curve hugs the zero axis while the queue is shallow. Then, as load increases, it introduces a growing barrier to higher delay. But, unlike RED, it requires only one parameter, the scaling, not three.

There follows a summary listing of the two parameters used for each of the two queues:

Classic:

$S_C$  : The scaling factor of the dropping function scales Classic queuing times in the range  $[0, 2^{S_C}]$  seconds into a dropping probability in the range  $[0,1]$ . To make division efficient, it is constrained to be an integer power of two;

$f_C$  : To smooth the queuing time of the Classic queue and make multiplication efficient, we use a negative integer power of two for the dimensionless EWMA constant, which we define as  $2^{-f_C}$ .

Internet-Draft

DualQ Coupled AQM

August 2015

L4S :

S\_L (and k): As for the Classic queue, the scaling factor of the L4S marking function scales Classic queueing times in the range  $[0, 2^{(S_L)}]$  seconds into a probability in the range  $[0,1]$ . Note that  $S_L = S_C + k$ , where  $k$  is the coupling between the queues ([Section 2.1](#)). So  $S_L$  and  $k$  count as only one parameter;

T : The queue size in bytes at which step threshold marking starts in the L4S queue.

{ToDo: These are the raw parameters used within the algorithm. A configuration front-end could accept more meaningful parameters and convert them into these raw parameters.}

From our experiments so far, recommended values for these parameters are:  $S_C = -1$ ;  $f_C = 5$ ;  $T = 5 * MTU$  for the range of base RTTs typical on the public Internet. [[CRED\\_Insights](#)] explains why these parameters are applicable whatever rate link this AQM implementation is deployed on and how the parameters would need to be adjusted for a scenario with a different range of RTTs (e.g. a data centre) {ToDo incorporate a summary of that report into this draft}. The setting of  $k$  depends on policy (see [Section 2.4](#) and [Appendix B](#) respectively for its recommended setting and guidance on alternatives).

There is also a cUrviness parameter,  $U$ , which is a small positive integer. It is likely to take the same hard-coded value for all implementations, once experiments have determined a good value. We have solely used  $U=1$  in our experiments so far, but results might be even better with  $U=2$  or higher.

Note that the dropping function at line 9 calls `maxrand(2*U)`, which gives twice as much curviness as the call to `maxrand(U)` in the marking function at line 3. This is the trick that implements the square rule in equation (1) ([Section 2.1](#)). This is based on the fact that, given a number  $X$  from 1 to 6, the probability that two dice throws will both be less than  $X$  is the square of the probability that one throw will be less than  $X$ . So, when  $U=1$ , the L4S marking function is linear and the Classic dropping function is squared. If  $U=2$ , L4S would be a square function and Classic would be quartic. And so on.

The `maxrand(u)` function in lines 16-21 simply generates  $u$  random numbers and returns the maximum (Note 7). Typically, `maxrand(u)` could be run in parallel out of band. For instance, if  $U=1$ , the Classic queue would require the maximum of two random numbers. So, instead of calling `maxrand(2*U)` in-band, the maximum of every pair of

Internet-Draft

DualQ Coupled AQM

August 2015

values from a pseudorandom number generator could be generated out-of-band, and held in a buffer ready for the Classic queue to consume.

```

1: dualq_dequeue(lq, cq) { % Couples L4S & Classic queues, lq & cq
2:   if ( lq.dequeue(pkt) ) {
3:     if ((lq.byt() > T) || ((cq.ns() >> (S_L-2)) > maxrand(U)))
4:       mark(pkt)
5:     return(pkt) % return the packet and stop here
6:   }
7:   while ( cq.dequeue(pkt) ) {
8:     Q_C += (pkt.ns() - Q_C) >> f_C % Classic Q EWMA
9:     if ( (Q_C >> (S_C-2)) > maxrand(2*U) )
10:      drop(pkt) % Squared drop, redo loop
11:     else
12:      return(pkt) % return the packet and stop here
13:   }
14:   return(NULL) % no packet to dequeue
15: }
```

Figure 2: Optimised Example Dequeue Pseudocode for Coupled DualQ AQM using Integer Arithmetic

#### Notes:

1. The drain rate of the queue can vary if it is scheduled relative to other queues, or to cater for fluctuations in a wireless medium. To auto-adjust to changes in drain rate, the queue must be measured in time, not bytes or packets [CoDel]. In our Linux implementation, it was easiest to measure queuing time at dequeue. Queuing time can be estimated when a packet is enqueued by measuring the queue length in bytes and dividing by the recent drain rate.
2. An implementation has to use priority queueing, but it need not implement strict priority.
3. If packets can be enqueued while processing dequeue code, an implementer might prefer to place the while loop around both queues so that it goes back to test again whether any L4S packets arrived while it was dropping a Classic packet.
4. In order not to change too many factors at once, for now, we keep the marking function for DCTCP-only traffic as similar as possible to DCTCP. However, unlike DCTCP, all processing is at dequeue, so we determine whether to mark a packet at the head of the queue by the byte-length of the queue `_behind_` it. We plan to test whether using queuing time will work in all circumstances, and if we find that the step can cause

oscillations, we will investigate replacing it with a steep random marking curve.

5. An EWMA is only one possible way to filter bursts; other more adaptive smoothing methods could be valid and it might be appropriate to decrease the EWMA faster than it increases.
6. In practice at line 10 the Classic queue would probably test for ECN capability on the packet to determine whether to drop or mark the packet. However, for brevity such detail is omitted. All packets classified into the L4S queue have to be ECN-capable, so no dropping logic is necessary at line 3. Nonetheless, L4S packets could be dropped by overload code (see [Section 4.1](#)).
7. In the integer variant of the pseudocode (Figure 2) real numbers are all represented as integers scaled up by  $2^{32}$ . In lines 3 & 9 the function `maxrand()` is arranged to return an integer in the range  $0 \leq \text{maxrand()} < 2^{32}$ . Queuing times are also scaled up by  $2^{32}$ , but in two stages: i) In lines 3 and 8 queuing times `cq.ns()` and `pkt.ns()` are returned in integer nanoseconds, making the values about  $2^{30}$  times larger than when the units were seconds, ii) then in lines 3 and 9 an adjustment of -2 to the right bit-shift multiplies the result by  $2^2$ , to complete the scaling by  $2^{32}$ .

#### [Appendix B](#). Guidance on Controlling Throughput Equivalence

RTT_C / RTT_L	Reno	Cubic
1	k=1	k=0
2	k=2	k=1
3	k=2	k=2
4	k=3	k=2
5	k=3	k=3

Table 1: Value of k for which DCTCP throughput is roughly the same as Reno or Cubic, for some example RTT ratios

To determine the appropriate policy, the operator first has to judge whether it wants DCTCP flows to have roughly equal throughput with Reno or with Cubic (because, even in its Reno-compatibility mode, Cubic is about 1.4 times more aggressive than Reno). Then the operator needs to decide at what ratio of RTTs it wants DCTCP and Classic flows to have roughly equal throughput. For example choosing the recommended value of k=0 will make DCTCP throughput roughly the same as Cubic, *\_if their RTTs are the same\_*.

Internet-Draft

DualQ Coupled AQM

August 2015

However, even if the base RTTs are the same, the actual RTTs are unlikely to be the same, because Classic (Cubic or Reno) traffic needs a large queue to avoid under-utilization and excess drop, whereas L4S (DCTCP) does not. The operator might still choose this policy if it judges that DCTCP throughput should be rewarded for keeping its own queue short.

On the other hand, the operator will choose one of the higher values for  $k$ , if it wants to slow DCTCP down to roughly the same throughput as Classic flows, to compensate for Classic flows slowing themselves down by causing themselves extra queuing delay.

The values for  $k$  in the table are derived from the formulae, which was developed in [DCTH15]:

$$2^k = 1.64 \text{ (RTT\_reno / RTT\_dc)} \quad (2)$$

$$2^k = 1.19 \text{ (RTT\_cubic / RTT\_dc)} \quad (3)$$

For localized traffic from a particular ISP's data centre, we used the measured RTTs to calculate that a value of  $k=3$  would achieve throughput equivalence, and our experiments verified the formula very closely.

#### Appendix C. DCTCP Safety Enhancements

This Appendix is informational not normative. It records changes needed to DCTCP implementations so they can co-exist safely alongside other traffic sources. They are recorded here until a more appropriate draft is available to hold them.

Proposed changes are listed in rough order of criticality. Therefore those later in the list may not be necessary:

- o Negotiate its altered feedback semantics, which conveys the extent of ECN marking, not just its existence, and this feedback needs to be robust to loss [I-D.ietf-tcpm-accecn-reqs];
- o fall back to Reno or Cubic behaviour on loss;
- o use a packet identifier associated with the L4S service;
- o average ECN feedback over its own RTT, not the hard-coded RTT suitable only for data-centres, perhaps like Relentless TCP [Mathis09];
- o handle a window of less than 2 when the RTT is low, rather than increase the queue [TCP-sub-mss-w].

Internet-Draft

DualQ Coupled AQM

August 2015

- o test heuristically whether ECN marking is emanating from an [RFC3168](#) AQM.

Other, non-essential enhancements to DCTCP can be envisaged.

#### Authors' Addresses

Koen De Schepper  
Bell Labs  
Antwerp  
Belgium

Email: [koen.de\\_schepper@alcatel-lucent.com](mailto:koen.de_schepper@alcatel-lucent.com)  
URI: [https://www.bell-labs.com/usr/koen.de\\_schepper](https://www.bell-labs.com/usr/koen.de_schepper)

Bob Briscoe (editor)  
Independent

Email: [ietf@bobbriscoe.net](mailto:ietf@bobbriscoe.net)  
URI: <http://bobbriscoe.net/>

Olga Bondarenko  
Simula Research Lab  
Lysaker  
Norway

Email: [olgabnd@gmail.com](mailto:olgabnd@gmail.com)  
URI: <https://www.simula.no/people/olgabo>

Ing-jyh Tsang  
Bell Labs  
Antwerp  
Belgium

Email: [ing-jyh.tsang@alcatel-lucent.com](mailto:ing-jyh.tsang@alcatel-lucent.com)

## C Getting up to Speed Fast Publications

In this appendix the publications of the Getting up to Speed Fast (GUTS) activity are included:

- B. Briscoe, M. Rajiullah, A. Brunstrom, and A. Petlund, “What Use is Top Speed without Acceleration?,” Reducing Internet Transport Latency (RITE) project, Tech. Rep., Jul. 2015

**Abstract:** This paper is a call to action. The aim of the paper is to fully characterise the slow-start problem. This involves quantifying the latency penalty of the start-up phase of typical Internet flows through evaluating a model of the state-of-the-art flow-start-up dynamics and characterising typical flow lengths. We have analysed a decade of traces from an Internet backbone as well as traces collected from a point much closer to users to account for CDN traffic. This analysis shows that an ever-growing proportion of typical user’s sessions is becoming limited by flow-start limitations, not capacity. Moreover, according to our trace analysis, in more than 50% of the cases, access links are totally empty when a new flow starts, hinting at the potential of a better flow-start to exploit the idle capacity. We also show that the impact of the flow-start problem largely depends on the judgment of values for different flow sizes to users. The paper further discusses the lack of scalability of the known solutions that would be easiest to deploy—those that only depend on unilateral deployment by one party. This leaves only solutions that require coordinated deployment. The paper avoids siding with any particular solution, but identifies the critical deployment problems that any solution will have to overcome.

- B. Briscoe and P. Hurtig, “Up to Speed with Queue View (QV),” RITE EU FP7 Project 317700, Technical Report, Aug. 2014, (Unpublished)

**Abstract:** The central idea of Queue View is for buffers on the path to continuously encode the length of their queue by setting markers in the explicit congestion notification (ECN) field in packet headers.

- M. Kühlewind, R. Scheffenegger, and B. Briscoe, “Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback,” Internet Engineering Task Force, Request for Comments rfc7560, Aug. 2015. [Online]. Available: <https://tools.ietf.org/html/rfc7560>

**Abstract:** Explicit Congestion Notification (ECN) is a mechanism where network nodes can mark IP packets, instead of dropping them, to indicate congestion to the endpoints. An ECN-capable receiver will feed this information back to the sender. ECN is specified for TCP in such a way that it can only feed back one congestion signal per Round-Trip Time (RTT). In contrast, ECN for other transport protocols, such as RTP/UDP and SCTP, is specified with more accurate ECN feedback. Recent new TCP mechanisms (like Congestion Exposure (ConEx) or Data Center TCP (DCTCP)) need more accurate ECN feedback in the case where more than one marking is received in one RTT. This document specifies requirements for an update to the TCP protocol to provide more accurate ECN feedback.

- B. Briscoe, R. Scheffenegger, and M. Kühlewind, “More Accurate ECN Feedback in TCP,” Internet Engineering Task Force, Internet Draft draft-kuehlewind-tcpm-accurate-ecn-04, Sep. 2015, (Work in Progress). [Online]. Available: <http://tools.ietf.org/html/draft-kuehlewind-tcpm-accurate-ecn>

**Abstract:** Explicit Congestion Notification (ECN) is a mechanism where network nodes can mark IP packets instead of dropping them to indicate incipient congestion to the end-points. Receivers with an ECN-capable transport protocol feed back this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recently, new TCP mechanisms like Congestion Exposure (ConEx) or Data Center TCP (DCTCP) need more accurate ECN feedback information whenever more than one marking is received in one RTT. This document specifies an experimental scheme to provide more than one feedback signal per RTT in the TCP header. Given TCP header space is scarce, it overloads the three existing ECN-related flags in the TCP header and provides additional information in a new TCP option.



# What Use is Top Speed without Acceleration?

Bob Briscoe<sup>†</sup>, Mohammad Rajiullah<sup>‡</sup>, Anna Brunstrom<sup>‡</sup>, Andreas Petlund<sup>\*</sup>

<sup>†</sup>, British Telecom, UK <sup>‡</sup>Karlstad University, Sweden,

<sup>\*</sup>Simula Research Laboratory, Norway

bob.briscoe@bt.com, {moharaji,annabrun}@kau.se,  
apetlund@simula.no

## ABSTRACT

This paper is a call to action. The aim of the paper is to fully characterise the slow-start problem. This involves quantifying the latency penalty of the start-up phase of typical Internet flows through evaluating a model of the state-of-the-art flow-start-up dynamics and characterising typical flow lengths. We have analysed a decade of traces from an Internet backbone as well as traces collected from a point much closer to users to account for CDN traffic. This analysis shows that an ever-growing proportion of typical user's sessions is becoming limited by flow-start limitations, not capacity. Moreover, according to our trace analysis, in more than 50% of the cases, access links are totally empty when a new flow starts, hinting at the potential of a better flow-start to exploit the idle capacity. We also show that the impact of the flow-start problem largely depends on the judgment of values for different flow sizes to users. The paper further discusses the lack of scalability of the known solutions that would be easiest to deploy—those that only depend on unilateral deployment by one party. This leaves only solutions that require coordinated deployment. The paper avoids siding with any particular solution, but identifies the critical deployment problems that any solution will have to overcome.

## 1. INTRODUCTION

Internet transport protocols use congestion control [23] to adapt to the available capacity in the path. A congestion control has to be designed on the basis that each time a flow starts the available capacity is unknown. Even a re-start after idling cannot assume that the capacity or other traffic has remained unchanged [46]. Path capacity changes rapidly when the physical capacity changes or as other flows arrive and depart, at least in scenarios with low numbers of flows at the bottleneck. So senders universally use some variant of the TCP slow-start algorithm [4], which sends an initial handful of packets, waits for feedback, then doubles how much it sends in each subsequent round as long as it has sensed no losses in the feedback. TCP's exponential slow-start with various optimization (that we shall model) is traditionally considered an acceptable compromise between acceleration and overshoot. How-

ever, slow-start does not scale, because every doubling of target flow rate<sup>1</sup> takes another round trip to reach, which means that more capacity only improves performance for flows larger than a limit that doubles as fast as link capacities double.

Unscalability of slow-start would not be a problem if all transfer sizes were getting larger, but they are not. As we continue to invest in greater link capacity, larger transfers become feasible. But it does not follow that all the demand for the smaller objects disappears—the range of feasible transfer sizes merely widens. We will show that an ever-growing proportion of a typical user's sessions become limited by slow-start, not capacity. And, even for flows that are large enough to exploit faster links, the amount of loss experienced during overshoot increases. This problem is not unique to TCP—in the absence of any explicit signalling all data flows face the same dilemma. In the rest of this paper, we shall loosely call this the 'slow-start' problem, without any implication it is only a problem for TCP flows. A solution for a more efficient flow-start is important for reducing latency over the Internet for a typical user session. This is also one of the remaining open issues of the Internet congestion control [46]. The longer we fail to solve this slow-start problem, investing in capacity will make less and less difference to more and more people.

This paper aims to fully characterise the slow-start problem. Some understands it intuitively, but most has slight misunderstandings. To this end, this paper attempts to quantify the proportion of a typical user's experience that is limited by slow-start rather than bottleneck link capacity as link capacities increase. The problem with slow-start is that it always has to start from the bottom but it gets no hard information from the network until it reaches the top. Over the years, as we make the top higher, a greater proportion of all transfers complete before they even discover that they didn't need to be cautious anyway. The paper presents a model of the state of the art TCP slow-start with a characterisation of typical flow length to show that only

<sup>1</sup>Bottleneck link capacities have doubled every 1.6 years over the past two decades [42].

a very small proportion of flows can now benefit from making link rates any faster unless a better approach than slow-start can be found. We use longitudinal study in the analysis, not to prove that the Internet will always be like it was but to provide a concrete data point for the model. This paper also attempts to determine whether there are inherent limits to solving this problem without introducing additional explicit signalling and to quantify those limits.

The rest of the paper is organized as follows. §2 describes a model of the long-term scaling of Internet traffic. §3 quantifies the slow-start problem. §4 discusses several existing solutions to the slow-start problem followed by related research in §5. Finally some concluding remarks and indication to future work are given in §6.

## 2. SCALING MODEL

In this section, we present a model of the long-term scaling of Internet traffic in the three dimensions of flow-size, flow-rate and number of flows. The following scaling model shows that capacity increases can be used to absorb larger flows or more simultaneous flows, but there is also an expectation that increases in capacity should make individual flows faster.

After Kelly [27] we model the change in scale of traffic using three independent scaling factors:

$a$ , the change in size (i.e. volume) of each flow;

$b$ , the change in rate of a flow of constant size;

$c$ , the change in the number of simultaneous flows.

If the change in capacity is referred to as  $X$ , then in order to absorb these three dimensions of traffic scaling,  $X$  needs to be equal to  $a.b.c$ . Note that the speed-up factor  $a$  absorbs the scaled-up volume of each flow without increasing its completion time while  $b$  is an *additional* speed-up, or equivalently the speed-up that a flow of constant size would experience. Increase factor  $c$  absorbs more flows while keeping the link utilisation unchanged. Of course, any of the factors  $a, b, c$  may take values less than one to represent decreases.

Moreover, for a given change in  $X$ , the ideal change in rate (let's call it  $b^*$ ) is  $\frac{X}{ac}$ . However, this is only achievable when flows are capacity limited. In practice the change in rate is often smaller as the flows are protocol limited. So, for an increasing capacity, if the actual change in rate is referred to as  $b$ , then  $\frac{b}{b^*} \rightarrow 0$ . In this case,  $\frac{b}{b^*}$  measures the effectiveness of capacity upgrades.

Each factor in the scaling model is a statistical quantity, with a mean and a distribution. In §3 we will use packet trace data over the last twelve years to put numbers to these factors. Nonetheless it is useful to have a generic model to be able to explore other possible ways

traffic may scale, because the past is not always a good prediction of the future.

## 3. QUANTIFYING THE PROBLEM

§3.1 presents a model for the best possible variant of TCP slow-start to show that flows smaller than a certain size can not benefit from making links any faster. §3.2 illustrates how an ever increasing proportion of Internet flows are limited by slow-start, not by capacity, based on a longitudinal study of packet traces from an Internet core. §3.3 shows how different value-judgments can influence the slow-start problem.

### 3.1 Model of Slow-Start

To analyse the slow-start problem we model the average transfer rate of a single TCP flow with a dedicated bottleneck capacity. We assume that access links are more often empty than full when a new flow starts, which we will later argue to be true based on our measurements of real links. Using the model we will show which factors influence the performance. Average transfer rate rather than completion time is chosen as the 'figure of merit'. An alternative could have been to use completion time, but average rate has the advantage that it is meaningful for different flow volumes where a bigger number represents better performance. Besides, anyone can compare it directly with well-known bottleneck access link rates. We use average rate as

$$\text{Average rate} = \text{flow volume} / \text{completion time}.$$

In order to quantify how inefficient slow-start is, we want to ensure that we criticise the most aggressive variant of slow-start. Therefore we use state of the art parameters for slow-start, that is:

- Initial window of ten Ethernet-size segments [14], as well as three for comparison;
- Base two exponential increase (i.e. doubling per round);
- A range of round trip times that are realistic for the public Internet, with and without caching;
- No use of multiple parallel flows (because a larger initial window is equivalent), see Appendix B;

Our model for the slow-start is described in Appendix A. For a wide range of sizes of each transfer (1B to 1GB), Fig. 1 shows the average rate achieved over the whole transfer, including the slow-start phase<sup>2</sup>, on a log-log scale for a few scenarios.

<sup>2</sup>But excluding the initial round of handshaking, i.e. optimistically assuming the flow is re-starting without having to hand-shake, e.g. a re-start after idling or using the experimental Fast Open enhancement to TCP [12] after an earlier connection between the same client-server pair.

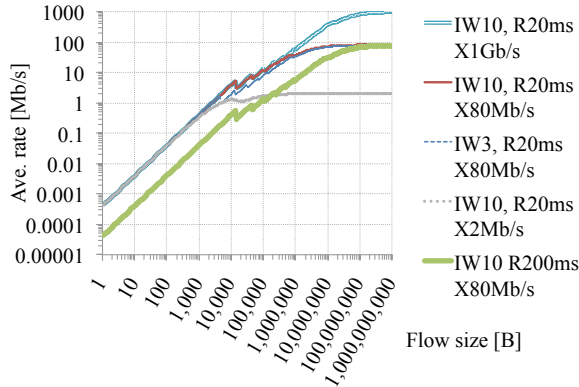


Figure 1: The problem: TCP has to limit the average transfer rate of different size flows despite dedicated capacity  $X$ . IW=initial window; R=round trip time.

Taking the second plot in the figure as an example (IW=10,  $R=20\text{ms}$ ,  $X=80\text{Mb/s}$ ), it shows that unless the transfer size is larger than around 1MB, it hardly even starts to exploit a dedicated 80Mb/s link. A 1MB transfer can only average about 40Mb/s, while a smaller 15kB transfer can only average about 3Mb/s. For transfer sizes less than 500kB, any capacity increase beyond about 80Mb/s would make no noticeable difference to average rate (illustrated by comparing the  $X=80\text{Mb/s}$  and the  $X=1\text{Gb/s}$  plot in the figure). Keeping the scaling model (discussed in §2) in mind, the lack of scalability is obvious here. A capacity increase does not give any additional rate increase,  $b$ , to fill available capacity for a wide range of increase in flow sizes,  $a$ . Again the area above the 1Gb/s plot and the 1Gb/s horizontal is inaccessible to a large number of flow sizes.

A 2Mb/s plot is also included to show that it is much more useful to many more people to increase capacity from 2Mb/s to 80Mb/s, because it considerably increases the average rate of a broad range of popular sizes of transfer (any larger than 3kB).

Furthermore, the last plot of Fig. 1 shows that the problem is significantly worse if the round trip time is longer. For an inter-continental round trip time that is ten times longer ( $R=200\text{ms}$ ) only flows ten times larger (more than 10MB) can start to make use of 80Mb/s capacity.

Comparing the second and third plots shows that increasing the initial window (IW) from 3 to 10 (as recently deployed by Google) makes a reasonable difference for flows between 4.5kB and 15kB, but for larger flows it makes increasingly less difference.

One might think that the dependency between the capacity and traffic volume is the other way round: as capacity grows new applications emerge that exploit it by transferring larger objects—“build it and they will

come”. However, in the next section we will see that the distribution of transfer sizes is not changing much as capacity grows.

### 3.2 Longitudinal Study of Packet Traces

The scaling behaviour illustrated in Fig. 1 would not be a problem if all Internet flows were getting larger (parameter  $a$  in the scaling model in §2). Next, we show that while increased capacity provides the possibility of larger transfers, the demand for small transfers continues. To establish the proportion of traffic that is limited by slow-start rather than capacity we have performed a study of Internet flow sizes based on Cooperative Association for Internet Data Analysis (CAIDA) [11] packet traces.

In the following we show the flow size distribution based on CAIDA traces. We also process the traces to identify the number of active flows that are available when a new flow starts in a user’s access link. Finally, since CDN traffic is likely to be heavily under-represented in the CAIDA traces, we do the same trace analysis on a trace that contains all the traffic from users.

#### 3.2.1 CAIDA Packet Traces

CAIDA packet traces were recorded by CAIDA’s data collection monitors at two Equinix datacenters. Each yearly dataset for the years 2008-2012 were collected at these two monitors that are connected to bi-directional backbone links of a Tier 1 ISP between Chicago and Seattle, and San Jose and Los Angeles, respectively. Each data set generally consists of twelve monthly samples, taken at the same time of the day. Each monthly sample includes separate traces for each backbone direction. The average size of a monthly dataset varies between 100 and 350 GB. Earlier data sets were obtained from the ampath-oc12 passive monitor, which tapped two directions of an OC12c link located in Miami, Florida and is no longer in use.

#### 3.2.2 Flow Size Distribution

Fig. 2 shows the prevalence of flow sizes in the CAIDA data sets over the decade, 2002-2012. As shown in Fig. 2a, the majority of the flows is short in size and has hardly changed over time. The prevalence of short flows is also consistent with earlier studies [9, 16, 49, 57]. Although large flows are rare in Fig. 2a, if we zoom in on the tail of the flow size distribution, as shown in Fig. 2b; the tail gets longer over time, showing the presence of even larger flows in recent years. Nevertheless, the distribution of typical transfer sizes has not changed (scaling parameter,  $a$  in §2) much as the links have got faster over the years.

In general, the slow-start problem does not play a role for flows that can fit in the initial congestion win-

dow [4], up to approximately 90th percentile in Fig. 2a (considering initial window of 10 segments). Similarly, for very large connections, initial delay due to the slow-start does not have a very large impact on the overall performance. The slow-start problem therefore mainly influences flows that require more than 1 RTT to complete<sup>3</sup> but not so long as the delay from the slow-start is only a small transient as compared to the overall transfer time. For example, flows above 90th percentile in the CDFs in Fig. 2a will require more than one RTT to complete and the majority of those flows are not large enough to ignore the limitation from slow-start.

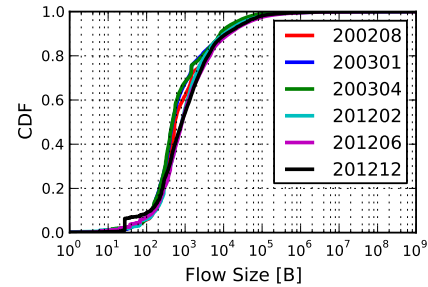
Moreover, Fig. 2a shows that the flow size distribution has not changed (scaling parameter,  $a$  in §2) much over the last decade. The capacity change during this time suggests the limited efficiency of the investment. This can be shown for a range of flow sizes in the figure with the slow-start model in Fig. 1. For example,  $b^*$  is 40 for a capacity increase from 2 to 80Mb/s. However, for a flow size of 202 kB and IW 10, based on the corresponding average rates in the figure, the actual rate increase,  $b$  is only  $\frac{18Mb/s}{2Mb/s}$  or 9, so the effectiveness from the capacity upgrade is only  $\frac{9}{40}$  or 22.5%. The effectiveness reduces even further for larger upgrade. For example, for the same flow size, if the capacity is further upgraded from 2 to 1Gbps, the effectiveness is reduced to only 2%.

### 3.2.3 Identifying Parallel Flows

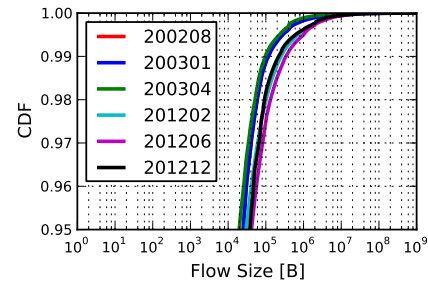
To circumvent the slow-start problem, applications some times open multiple flows together [6]. These flows are correlated and with respect to their capacity demand they should be considered as a single flow. However, it is difficult to find the correlation of a group of packets that are seen temporally together in the trace. These packets can not be considered as a single flow if the relevant flows were not started together. Nevertheless, the number of active flows gives a notion of link occupancy or utilization. For example, if there are  $n$  active flows in a link,  $n$  gives an indication of the link occupancy for a new flow coming to this link. To this end, we identify the number of active flows from each user (individual IP) from the CAIDA datasets. We look per user in order to approximate each user's access link that we assume as their bottleneck. In our analysis, an active flow means that at least a single packet<sup>4</sup> has been exchanged in the last  $x$  ms. Our choice of  $x$  comes from rather coarse assumptions of the maximum RTT (we tried with 100 and 300 ms). Fig. 3 shows the CDF of the estimated number of active flows in a user's access link when a new flow starts, and how it differs between the years. We here assume that all the packets in a user's access link contain a single IP address of a user

<sup>3</sup>Not considering the hand shaking delay.

<sup>4</sup>Excluding FIN and RST packets.



(a) Whole CDF



(b) Tail of the CDF

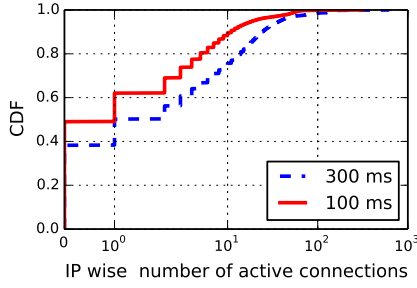
Figure 2: Prevalence of flows of different sizes on the Internet, based on traces from CAIDA.

(or of multiple users behind a NAT). This is of course not necessarily true for a user with multiple communicating devices or multiple users, not using a NAT.

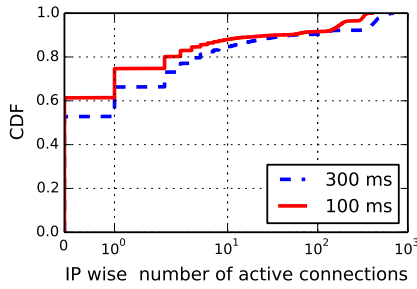
According to the graphs in Fig. 3, most of the time when a flow starts users' access links are idle, implicating no clear change of the distribution of scaling parameter  $c$  from §2. Slow-start will give poor utilization in such scenarios. Also, over the years, a new flow coming to a user's access link mostly sees a similar number of active flows in the link for approximately 80% of the cases as seen in the figure. Nevertheless, comparing the CDFs over 80th percentile in both Fig. 3a and 3b, the number of active flows or parallel flows has increased over time in user access links. Besides, the difference between the graphs in each figure is due to the fact that when we assume a higher value of the maximum RTT, 300 ms, connections with inter-arrival times longer than 100 ms are also considered as active connections. However, this analysis has a serious limitation, because we cannot assume that for any user all the flows go through the CAIDA monitors. In the next section, we use traces that were collected from points much closer to the users.

### 3.2.4 Access-link Packet Traces

In today's Internet, much traffic a typical user sees is served by CDNs. When users request resources, they are often redirected to the set of CDN caches closest to them instead of the requested servers. The involve-



(a) CAIDA:200208



(b) CAIDA:201212

Figure 3: Number of active flows seen when a new flow starts.

ment of CDNs makes a longitudinal study difficult, because we only have historic data under repeatable conditions from the Internet exchange point that CAIDA uses, which no longer may show traffic representative of that seen by end-users.

In this section, we use a different trace that mainly records the customer's traffic. The trace was collected by UNINETT<sup>5</sup> from the access link of University of Agder (UoA) in Norway<sup>6</sup>. The trace contains traffic from the university as well as from the campus where students accommodates. It is a 10 Gbps link. The trace duration was over two hours and was taken during peak hours (between 13:30 pm and 15:45 pm). We will refer to this trace as the UoA trace.

Fig. 4 shows the CDF of flow sizes for the UoA trace. Unlike CAIDA traces, the UoA trace contains all the traffic from the users. However, similar to the flow size distribution of CAIDA traces in §3.2.2, we see that the majority of user traffic is small in size. Besides, in the figure, similar to our observations for CAIDA traffic in the CDFs of Fig. 2a, the majority of flows over 90th percentile can potentially be limited by the slow-start problem.

Similar to our analysis in §3.2.3 we also look for the correlated flows in the UoA trace. Fig. 5 shows the

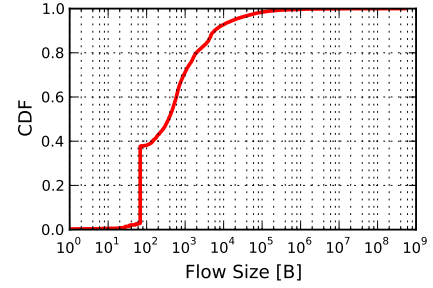


Figure 4: CDF of flow sizes for the UoA trace

number of active flows seen when a new flow starts. According to the figure, user's access link occupancy level is again low, suggesting that an end user can get potentially more speedup if the available capacity is quickly utilized. However, our correlated flow analysis is still limited to our assumption of each user containing a single IP address. To address our limitations, we also do the utilization analysis for all users together, which shows the level of utilization in the link where the trace was captured. From our trace based analysis, the link shows to be lightly utilized. However, according to the number of active flows in Fig. 6, there are some constant number of flows in the link. Our further investigation shows that these are very low intensity flows that mostly exchange one or two packets in every 100/300 ms.

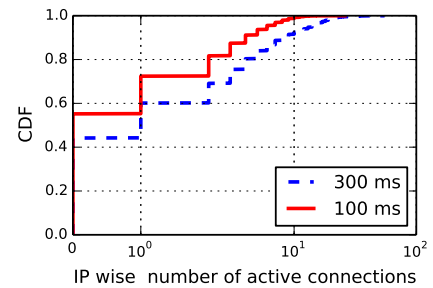


Figure 5: Number of active flows seen when a new flow starts (based on the UoA trace)

### 3.3 Impact on User Value

In the following we will show that the impact of the slow-start problem is largely influenced by either how users value flows of various sizes or users' particular quality requirements.

#### 3.3.1 Flow Sizes

The analysis in §3.1 tells us that the limitation from slow-start mostly affects the short flows and subsequently §3.2 suggests that most of the flows are short. Without saying explicitly, we so far have assumed that all

<sup>5</sup><http://uninett.no>

<sup>6</sup><http://www.uia.no/en>

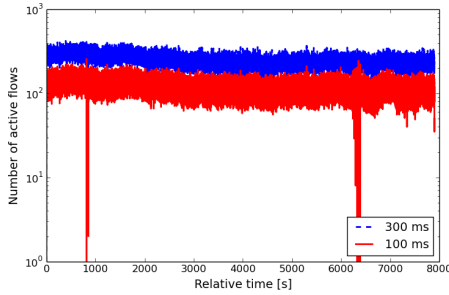


Figure 6: Number of active flows in the link (based on the UoA trace)

flows are of equal importance. However, the effect of the slow-start limitation depends on how one values a flow of a certain size. Traditionally, network engineers are mainly interested in large flows representing applications like traditional file downloading or video streaming for the purpose of capacity planning. However, users may value the short flows representing interactive applications like short messaging service, web browsing etc. the most [61]. Users may, for example, only be interested in the proportion of flows up to a certain size. Ideally, each flow could be weighted by the likely value users would gain from improving the performance of flows of that size. The CDF of flow sizes could be weighted by the value of the faster completion time for different flow sizes. In the following we will see, given different weighting approaches, how the importance of the slow-start problem can be changed under different value-judgement assumptions.

The following Fig. 7 shows how different weighting approaches affect the value for flows of different sizes. In other words, we weigh the CDF of flow sizes in different ways to get the intuition on how the size of the problem changes. For illustration, CAIDA traces (201212 dataset) have been used as a sample. In the value-judgments, for example, if the value per byte delivered is independent of flow size, then the CDF of proportion of value in flows up to a certain size would be identical to the CDF of bytes in flows up to a certain size (see the graph  $n$  in the figure). Under this assumption, the slow-start problem is marginal, since the majority of the flows users value are large enough to ignore the delays from the slow-start at the beginning. On the other hand, if the value per byte delivered is inversely proportional to flow size, then the CDF of value would be identical to the CDF of number of flows (see the graph 1) as shown in Fig. 2a. In this case, slow-start limitation affects most of the flows that users value and that do not fit in the initial congestion window. The figure also shows how other weighting approaches or value-judgments can change the distribution of value between

flows of different sizes. The first three graphs, showing weighting approaches that value mostly the bytes in short flows. In this case, the slow-start problem may affect a large number of flows that cannot be entirely transferred within the initial congestion window. On the other hand, weighing each flow size by its size divided by its log, the last graph, shows that the importance of the slow-start problem becomes extremely slim in the case where users mostly value very large flows. These flows are so large that the slow-start limitation at the beginning does not play any major role for them.

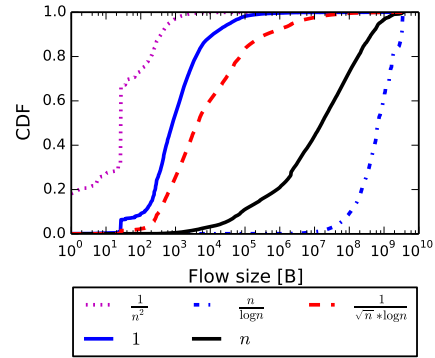


Figure 7: Value per byte analysis based on different weighting approaches.

### 3.3.2 Quality Requirements

The effect of the slow-start limitation also depends on the user's quality requirement. One central example is HTTP segment streaming that is responsible for an increasing amount of the traffic on the Internet today [34]. When a control action is performed for the video, like start, pause, forward, rewind, a full frame of video must be sent to restart the video playout. The delay experienced by the user is negligible for low-resolution video, but as the resolution increases, the slow-start limitation comes into play as long as most part of the transfer happens during slow-start. It rises to a level where the delay will easily be detected and experienced as annoying, even with a perfect link.

Table 1 shows the time it will take to download the first frame of video as the resolution increases<sup>7</sup>. The link speed is defined as unlimited with no packet loss, so the slow-start/RTT relation is the only limitation to the start-up latency. We can see that for recent Ultra High Definition TV (UHD TV) resolutions, the delay for starting the playout will be close to a second, a delay

<sup>7</sup>Since the I-frame needs to be transmitted in its entirety, the size is calculated based on the compression rate of 1:20 that is found in MPEG video. We have defined each pixel to, uncompressed, be represented by 24 bits. Thus, we use 1,2 bits per pixel to calculate the I-frame size in the table.



Format	Resolution	Bytes	Packets	Download time	Minimum bandwidth required to transfer the I-frame in slow-start
DVD	720 × 576	62,208	43	300ms	2 Mb/s
720p	1280 × 720	138,240	95	400ms	3 Mb/s
1080p	1920 × 1080	311,040	212	500ms	5 Mb/s
2160p	3840 × 2160	1,244,160	851	700ms	20 Mb/s
4320p	7680 × 4320	4,976,640	3404	900ms	77 Mb/s

Table 1: Download time for the 1st I-frame of streamed video using traditional slow-start on a link with unlimited bandwidth and RTT of 100ms.

that will be perceived as annoying to most users. There are other emerging technologies that require much more data to be streamed over the Internet, like high-quality 3D mesh streaming. In such scenarios, the start-up delay will of course be even bigger. The limitation from the slow-start will play a major role if most part of the flow is transferred while being in the slow-start.

The bandwidth column in Table 1 shows the minimum bandwidth required to transfer the whole I-frame in slow-start. Many of today’s connections already provide such bandwidths. For example, the obtainable bandwidth when using very high rate digital subscriber line (VDSL2) in Fiber to the node (FTTN) is well over 100 Mbps [45]. LTE-advanced (LTE-A) developed by 3GPP provides a speed of 300 Mbps for cellular connection [56]. Besides, the flow sizes in Table 1 fall over the 90th percentiles in the CDFs of both Fig. 2 and Fig. 4. However, since most of the video streaming are CDN traffic today [1, 39], they are probably missing in core Internet traces like CAIDA but could potentially exist in the UoA traces.

## 4. SOLUTIONS AND THEIR DEPLOYMENT PROBLEMS

The aim of this paper is not to present a solution to the slow-start problem; it is to thoroughly investigate the nature of the problem. That includes understanding why there appear to be no feasible solutions.

§4.1 shows that all the unilaterally deployable solutions<sup>8</sup> have their own scalability problems that make them little better than slow-start. §4.1 also includes solutions that have not been used due to implementation difficulties. Then §4.2 introduces proposals that seem more promising, but none are unilaterally deployable. This implies that if we are at the limit of host-only solutions, changing the network interface is an industry co-ordination problem of epic proportions.

### 4.1 Limited Scope for Unilaterally Deployable Solutions

Our aim here is to help the industry recognize that

<sup>8</sup>These solutions do not require changes to both network and end-systems.

Internet performance is becoming limited by a scaling problem for which there is no easy solution.

#### 4.1.1 Caching

The idea behind all sorts of caching technology available in the Internet is to reduce the time to access various Internet services. A typical caching mechanism reduces the RTT by keeping resources closer to the clients. However, the study in [31] found only an average of 22%–26% reduction of latency for web caching, even for cache hit ratios of 47%–52%. Although caching helps, caching itself in the network can also bring up the same unscalable transport dynamics we have shown for the end-systems. Besides, both the size and the placement of network caches can influence the performance of caching [8, 40].

Moreover, caching is not particularly suitable for a range of services dealing with real time/ dynamic content (e.g. gaming, interactive video), remote control operations or financial updates. Servers typically produce dynamic content at the time a request is made. Any legacy content caching or CDN is not suitable to cache dynamic contents [50]. For example, web pages containing stock quotations are not feasible to cache as stock prices may change in every second.

#### 4.1.2 Multiple Parallel Flows or a Large Initial Window

Developers of applications have tended to patch over the slow-start problem by opening multiple parallel flows. For instance, the Firefox Web browser opens 6 parallel connections for each domain<sup>9</sup> where today’s Web page contains components from several domains. Perhaps surprisingly, parallel flows provide exactly the same speed-up as starting with a larger IW, but with the overhead of opening all the extra flows. The total window across all flows still only doubles in the rounds after the first. For example, if four parallel flows each increase their windows in the sequence 3, 6, 12, ..., the total window across them all will start higher, but still double each round:  $4 \times 3, 4 \times 6, 4 \times 12 \dots = 12, 24, 48 \dots$

Using parallel flows is feasible for modest speed-up

<sup>9</sup><http://www.browserscope.org/?category=network>

of small flows, however, to keep up with capacity increases, the number of parallel flows would have to grow more than exponentially over time—the number of flows has to scale faster than capacity growth, which itself increases exponentially over the years [42]. This will quickly become infeasible, particularly as larger flows become constrained by slow-start rather than capacity—meaning flows that require more than half a dozen round trips to complete. For instance, over a 20 ms path, one 1MB flow can achieve an average rate of 70Mb/s. To double its average rate requires 15 flows, but to exploit four times the capacity would require 58 parallel flows<sup>10</sup>.

In summary, increasing beyond the numbers of multiple flows already in use, for example 6 in Firefox, will have limited additional benefit, unless stupidly large numbers of flows are used, which would introduce considerable other problems (e.g. flow-state memory exhaustion in servers and NATs, as well as the multiple initial windows or equivalently a single larger initial window overflowing buffers on slower lines).

#### 4.1.3 Faster Exponential

It may seem that a flow could get up to speed faster by more than doubling its rate in each round. However, in the last decade researchers at the leading edge of high-speed networking (e.g. for transferring large data sets in astronomy or physics) have found that even rate doubling leads to tremendous overshoot problems as link capacities grow [5, 20]. So far, attempts to detect overshoot early using delay measurements (see below) have produced mixed results, with worse performance as well as better even when just doubling.

More than doubling during slow-start would create the need to size buffers larger than one bandwidth-delay product, which already introduces delay problems for shorter flows and real-time applications.

#### 4.1.4 Using Queuing Delay Measurements

Superficially, a sender could send the initial window in a couple of line rate bursts, then measure how much inter-packet delays expand between sender and receiver. Then for the whole of the next round trip it could continuously pace a whole round of subsequent packets at the same average inter-arrival rate as the bursts of ACKs (or slightly more slowly, or leave some gaps to be conservative). Attempts to use similar approaches in [22, 30, 47] have made partial improvements in some scenarios but not all.

Paced Start [22] monitors the queuing delay that a buffer adds between packets when sent in trains during TCP slow-start and paces the packets sent in subsequent rounds. This avoids TCP's overshoot, but it takes even longer than TCP's slow-start to reach the avail-

able capacity. Swift-start [47] uses a packet-pair technique [29] with the first window of packets to determine the initial estimate of available bandwidth and then uses packet-pacing to spread out the estimated congestion window of packets over the next RTT. However, the paper does not discuss how good the estimation needs to be to be useful. Further, not only are more than a handful of packets typically necessary to obtain an accurate estimation [3] but also the packet-pair technique for accurate bandwidth estimation at the beginning of a TCP connection is specially found quite difficult [48]. Later a more sophisticated bandwidth estimation algorithm called RAPID was proposed [30]. RAPID is based on PathChirp [51]. In RAPID, each packet transmission is precisely timed, which produces multiple sending rates in a single RTT. The receiver then estimates the capacity from the observed inter-packet spacing. Under ideal conditions, RAPID can probe for or adapt to a large change in available bandwidth within one to four RTTs. However, precise timing of packet transmission has always been a great challenge for the implementers. The detailed implementation related challenges are reported in [32].

#### 4.1.5 Other Solutions

Liu *et al* [37] investigated what the impact would be if every flow simply tried to send all its data paced out over the first round trip time (termed Jump Start). If acknowledgements report losses or if the first acknowledgement returns while there is still data to send, the algorithm moves into TCP's standard retransmission and congestion avoidance behaviour. The authors monitored current Internet flows and found that only about 7.4% of them comprise more than the three packets that a sender would send immediately anyway under the existing standard behaviour. The paper is inconclusive on whether the edges of the Internet would cope with the very high loss rates that this 7.4% of flows would cause (because they represent a very much larger proportion of the bytes on the Internet).

Moreover, some solutions use state sharing at the sender to know the available capacity in the network. Temporal and ensemble sharing described in RFC 2140 [59], and the congestion manager framework described in RFC 3124 [7] all represent techniques for state sharing and joint congestion management at the sender. Although joint congestion management can allow a new flow to quickly find a suitable sending rate, it is only applicable when multiple flows share a common bottleneck.

## 4.2 Promising Solutions with Deployment Challenges

This section briefly surveys the landscape of solutions where deployment would be more involved. The pur-

<sup>10</sup>Appendix B derives the formula for how many flows would be needed to achieve different speed-ups.



pose is to show that, in order to solve the slow-start problem, the industry has to solve a problem of coordinated deployment, which we know from experience will be hard, e.g. IPv6, ECN.

Solutions fall into the following categories:

- new in-band per-packet signalling for all traffic using a new shim header, e.g. RCP [13], XCP [25];
- new in-band signalling for packets that choose to use it, but without changing the format of packet headers, e.g. Anti-ECN [33], VCP [62];
- new out-of-band per-flow signalling for flows that choose to use it, e.g. Quick-Start [18];

Clearly, approaches like RCP and XCP that require all traffic to use the new scheme and require a new packet header are not feasible to deploy in the general Internet. RCP and XCP today could only be deployed in private networks where there are large amounts of under-utilized bandwidth and the operator has all the control. Nonetheless, they help us understand what might be the limits of the possible, even with a clean-slate design.

One idea has been to use the ECT(1) value to signal an intermediate level of congestion between uncongested (ECT(0)) and congested (CE). This idea has been standardised in an approach termed pre-congestion notification (PCN [15]). PCN uses a virtual queue, which is not actually a queue; rather it is a number that represents the length of queue that would have formed if the buffer were drained more slowly than the real buffer drains. One variant of PCN uses two virtual queues, one configured to drain at a slower rate than the other. When the slower virtual queue fills, it marks packets with the ECT(1) codepoint and when the faster virtual queue fills it marks packets with the CE codepoint. The PCN approach is not standardised to be used as a signal to end-systems, only within the network.

Appendix D of Satoh *et al* [54] describes a mechanism for marking the proportion of packets that represents the instantaneous utilisation of a logical link by marking the ECN field of every packet that arrives when the virtual queue is non-empty (by Little's Law). The representation is strictly only precise for a Poisson distribution of inter-arrival times.

VCP [62] proposes that the ECT(1) codepoint of the ECN field should be used to signal when utilisation of a link has exceeded a set threshold, in a similar way to PCN but to address the problem of lack of information for end-systems at the start of a new data flow.

In "AntiECN Marking" [33] Kunniyur proposes that each packet should carry an Anti-ECN bit in its header. The bit is initially set to zero. Each router along the packet's path checks to see if it can allow the flow to

increase its sending rate by determining whether the packet has arrived at an empty virtual queue. If so, the router sets the bit to one, and otherwise to zero. If feedback from the receiver shows the bit was set to one, the sender increases its congestion window.

All the above proposals share the idea of starting to ECN-mark at a threshold set at some fraction of the link rate. They all require a decision on what fraction to use, but there is no natural 'best' setting, so it is unlikely that one standard will prevail. Therefore these approaches don't really solve the problem, they just move it. A source still has no information while it is searching for the lower virtual rate. Admittedly it doesn't matter if it overshoots, but once it has reached the virtual threshold, it still doesn't know how much faster it must send to reach the real available capacity, because the virtual queue doesn't tell it whether it is alone and once above the threshold the virtual queue saturates and gives no further useful information.

All the other approaches, in one way or another, involve end-systems asking network components to tell them whether it is safe to be more aggressive than slow-start. The main problem this raises is how the end-system can know whether sufficient network components along the path speak the new protocol, particularly whether the bottleneck does. This would not be a problem in green-field networks or isolated private networks under the control of a single administrator, e.g. a data center or an enterprise LAN surrounded by proxies. However, otherwise, this partial network deployment problem is the critical barrier to deployment.

Quick-Start attempts to solve the partial network deployment problem by introducing a Quick-Start time-to-live (QS TTL) field separate from the TTL field in the IP header. If there are fewer Quick-Start hops than IP hops, then end-systems know not to use Quick-Start. During early deployment this unfortunately leads to a vanishingly small probability of finding any path over which Quick-Start can be used. The problem is even worse given links within the subnets between IP hops will often have to be considered, unless the subnet is non-blocking by design.

## 5. OTHER RELATED WORK

Several existing works characterized the limitations in the slow-start algorithm. Some of these work have been covered in §4. Besides, several other works discuss the limitation with regard to parameter settings [14, 22, 52, 53, 60] or how fast it allows a flow to grab the available bandwidth [18, 26, 30, 37, 43]. In contrast, in this work we quantify the limitation of slow-start and show the proportion of flows in the Internet that are likely to be affected.

Several works [14, 22, 30, 43, 52, 53, 60] mention the influence of the size of IW on the performance of slow-

start. If IW is assigned to a very small value, slow-start may take a large number of RTTs to probe for the available bandwidth. The slow probing virtually provides no opportunity for short Internet transfers, like web flows, to reach the available capacity. According to Liu et al. [37], for a flow requiring a congestion window of  $N$  packets to fully utilize the available capacity, slow-start takes  $\log_2 N$  RTTs. In other words, the flow must be at least  $2N$  packets long to reach a congestion window of  $N$  packets. Besides, according to [26], the long probing not only causes poor bandwidth utilisation but also increases latency. Balakrishnan et al. [6] further mentioned that as a consequence of slow probing in slow-start, several applications tend to start multiple connections to probe faster. This further exacerbates the problem, for example, by competing with each other. However, unlike previous work, we thoroughly analyze the scaling limitations of long probing and then to substantiate our analytical findings, we investigate longitudinal packet traces from the Internet and show the proportion of flows likely to be affected.

Investigating Internet traffic characteristics is not new [9, 35, 44, 49, 58, 63]. These works mainly investigated several characteristics of a flow such as size, rate, duration and burstiness. On the other hand, in this paper, we present the longitudinal analysis of Internet flow sizes and its influence on the efficiency of the transport protocol used in the Internet from the view point of end-users. Similar to our work, Allman [2] grouped multiple parallel flows into a single flow to investigate the cumulative effect of a larger initial window. However, he used a crude resolution of 1 second bound for start times to group multiple flows.

In this paper we derive a TCP average rate model to show how efficiently TCP can grab the available bandwidth or the protocol efficiency for varying flow sizes, RTTs, slow-start parameters and bottleneck capacities. Similar to our work, protocol efficiency of TCP has been investigated in [19, 38] for mobile networks. These works investigated the overall efficiency of TCP as a protocol to utilize resources in different mobile networks. Liu et al. [38] quantified the aggregated cellular capacity loss due to TCP's inability to utilize the available bandwidth. Similar to our work, Garcia et al. [19] addressed the protocol efficiency from an end user perspective. However, their TCP modeling is mainly based on their measurements of loss and throughput in different mobile networks. Several earlier efforts of TCP modeling are done in [10, 24] and SCTP modeling is done in [36]. Packet loss has been assumed to be independent of the congestion window in [10, 24, 36], whereas in our model, loss is assumed to happen only once the congestion window fills up the bottleneck capacity, which is a typical scenario for end-users in wired networks.

Congestion avoidance in standard TCP has also been shown to be unscalable [41, 28, 17], especially in high speed wide area networks. In steady state, with a loss rate  $p$ , standard TCP's average congestion window is proportional to  $\frac{1}{\sqrt{p}}$ , which makes the loss recovery time extremely slow requiring very low loss rates. For example, the recovery time for standard TCP with 1500 B packets in a 100 ms RTT network is 9 min when transferring data at 1 Gb/s. In such a scenario TCP can only support a ridiculously low loss rate of  $2 \times 10^{-8}$ . Internet traffic, consequently, do not use the standard congestion control in TCP. For example, by default, Linux uses the Cubic algorithm [21] and windows server 2008 uses the Compound TCP algorithm [55]. These algorithms are not the ultimate scalable algorithm, but are less unscalable than standard TCP. Some of the scalable algorithms proposed in the literature are Scalable TCP [28] and Relentless Congestion Control [41] that try to give constant recovery time between losses by providing an average congestion window proportional to  $\frac{1}{p}$ .

## 6. CONCLUSIONS

As illustrated by the bit-rate against flow size in Fig. 1 and the CDFs of flow sizes in Fig. 2a, flows large enough to make use of tens of Mb/s of capacity are rare. Although, the CDFs of the value of the flow are largely influenced by different value-judgments, as seen in §3.3, considering the low latency requirements for short flows involving for example web data transfers or cloud based applications, our high level estimations suggest that the value is approximately inversely proportional to flow size. With the increasing capacity, an ever-growing proportion of typical user sessions are therefore becoming limited by slow-start, not capacity. Using a traffic scaling model, we have also shown that since there is no clear change of either the flow size distribution or number of parallel flows distribution, an additional speed-up of Internet flows by having a better start up mechanism is required to utilize the increasing capacity investment. A solution for a more efficient flow-start is thus important for reducing latency and for exploiting the invested capacity.

The slow-start problem is fundamental in the Internet. This is due to insufficient information passing over the standardised interface between hosts and networks (the service interface of IP). In other words the interface between IP and transport layers is deficient. In the paper, we have shown that there are some solutions to the slow-start problem that modify the slow-start parameters at the source but none of these solutions would scale into the future with the capacity - all were just stop-gaps. Several other approaches for an alternative flow-start, based on various methods to obtain information about the path, have also been proposed in the literature. Solutions that are based on measuring

delays along the path are promising. However, their performance largely depends on the accuracy of the measurements that are implementation wise challenging. We have further shown in the paper that solutions requiring changes in both network and the end nodes are hard to deploy and none of these solutions has therefore seen any real use. RFC6077 also identified the problem of an improved interface between network components and end-systems as perhaps the most critical open research issue in congestion control [46]. All the proposed solutions are attempting to solve the problem without changing this interface, but the problem will only get worse unless we grasp this nettle.

As a part of our future work, we will extend the current network/host interface in order to establish the generic information flow (in a precise and timely manner) from the network to the transport layer on which scalable solutions to the slow-start problem can be built. Given the constraints set by the existing IP protocol, we believe the updated interface will have to rely on some form of encoding of the ECN field as the information channel. This encoding will have to squeeze an extra signalling channel into the 2-bit ECN field, without interfering with the existing use of the ECN field by AQM algorithms.

## 7. ACKNOWLEDGMENTS

This work was funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the author(s). Raffaello Secchi reviewed the slow-start model. Koen De Schepper gave useful comments on a draft of the paper.

## 8. REFERENCES

- [1] V. K. Adhikari, G. Yang, H. Fang, V. Hilt, and Z. Zhang. A tale of three CDNs: An active measurement study of Hulu and its CDNs. In *Proceedings of Computer Communications Workshops (INFOCOM WKSHPS)*, pages 7–12. IEEE, March 2012.
- [2] M. Allman. Comments on bufferbloat. *ACM SIGCOMM Computer Communication Review*, 43(1):30–37, January 2012.
- [3] M. Allman and V. Paxson. On estimating end-to-end network path properties. In *Proceedings of ACM SIGCOMM*, pages 263–274, Cambridge, MA, USA, August 1999. ACM.
- [4] M. Allman, V. Paxson, and E. Blanton. TCP congestion control. Request for Comments 5681, Internet Engineering Task Force, 2009.
- [5] A. Antony, J. Blom, C. Laat, and J. Lee. Exploring practical limitations of TCP over transatlantic networks. *Future Generation Computing System*, 21(4):489–499, 2005.
- [6] H. Balakrishnan, H. S. Rahul, and S. Seshan. An integrated congestion management architecture for Internet hosts. In *Proceedings of ACM SIGCOMM*, pages 175–187, Cambridge, MA, USA, September 1999. ACM.
- [7] H. Balakrishnan and S. Seshan. The Congestion Manager. Request for Comments 3124, Internet Engineering Task Force, June 2001.
- [8] B. Bjurling and P. Kreuger. Performance of Cache Placement Policies. In *10th Swedish National Computer Networking Workshop, SNCNW*, Västerås, Sweden, June 2014.
- [9] N. Brownlee and K. C. Claffy. Internet stream size distributions. In *Proceedings of ACM SIGMETRICS*, pages 282–283, Marina Del Rey, California, USA, June 2002. ACM.
- [10] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP latency. In *Proceedings of IEEE Conference on Computer Communications*, volume 3, pages 1742–1751, Tel Aviv, Israel, March 2000. IEEE.
- [11] Center for Applied Internet Data Analysis (CAIDA). University of California, San Diego Supercomputer. <http://www.caida.org>. Accessed: 2015-01-07.
- [12] Y. Cheng, S. Radhakrishnan, and A. Jain. TCP Fast Open. Request for Comments 7413, Internet Engineering Task Force, December 2014.
- [13] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown. Processor sharing flows in the Internet. In *Proceedings of International Workshop on QoS*, pages 271–285, Passau, Germany, June 2005. Springer-verlag.
- [14] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An argument for increasing TCP’s initial congestion window. *ACM SIGCOMM Computer Communication Review*, 40:27–33, July 2010.
- [15] P. Eardley. Metering and marking behaviour of PCN-nodes. Request for Comments 5670, Internet Engineering Task Force, November 2009.
- [16] W. Fang and L. Peterson. Inter-AS traffic patterns and their implications. In *Proceedings of IEEE Global Communications Conference*, volume 3, pages 1859–1868, Rio de Janeiro, Brazil, March 1999. IEEE.
- [17] S. Floyd. HighSpeed TCP for large congestion windows. Request for Comments 3649, Internet Engineering Task Force, 2003.
- [18] S. Floyd, M. Allman, A. Jain, and P. Sarolahti. Quick-Start for TCP and IP. Request for Comments 4782, Internet Engineering Task Force, January 2007. (Status: Experimental).

- [19] J. Garcia, S. Alfredsson, and A. Brunstrom. A measurement based study of TCP protocol efficiency in cellular networks. In *Proceedings of International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, pages 131–136, Hammamet, Tunisia, May 2014. IEEE.
- [20] S. Ha and I. Rhee. Hybrid slow start for high-bandwidth and long-distance networks. In *Proceedings of International Workshop on Protocols for Future, Large-scale & Diverse Network Transports*, Manchester, UK, March 2008.
- [21] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.
- [22] N. Hu and P. Steenkiste. Improving TCP startup performance using active measurements: algorithm and evaluation. In *Proceedings of IEEE International Conference on Network Protocols*, pages 107–118, Atlanta, Georgia, USA, November 2003. IEEE.
- [23] V. Jacobson. Congestion avoidance and control. *ACM SIGCOMM Computer Communication Review*, 25(1):157–187, January 1995.
- [24] Z. Jiong, Z. Shu-jing, and Z. Qi-gang. An adapted full model for tcp latency. In *Proceedings of IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, volume 2, pages 801–804, Beijing, China, October 2002. IEEE.
- [25] D. Katabi. *Decoupling Congestion Control from the Bandwidth Allocation Policy and its Application to High Bandwidth-Delay Product Networks*. PhD thesis, MIT, March 2003.
- [26] R. H. Katz and V. N. Padmanabhan. TCP fast start: A technique for speeding up web transfers. In *Proceedings of IEEE Globecom Internet Mini-Conference*, Sydney, Australia, November 1998. IEEE.
- [27] F. P. Kelly. Models for a self-managed internet. *Philosophical Transactions of the Royal Society*, 358(1773):2335–2348, August 2000.
- [28] T. Kelly. Scalable TCP: improving performance in highspeed wide area networks. *ACM SIGCOMM Computer Communication Review*, 33(2):83–91, 2003.
- [29] S. Keshav. A control-theoretic approach to flow control. *ACM SIGCOMM Computer Communication Review*, 25(1):188–201, 1995.
- [30] V. Konda and J. Kaur. RAPID: Shrinking the congestion-control timescale. In *Proceedings of IEEE Conference on Computer Communications*, pages 1–9, Rio de Janeiro, Brazil, April 2009. IEEE.
- [31] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, pages 13–22, Anaheim, CA, USA, January 1997. USENIX Association.
- [32] M. Kühlewind and B. Briscoe. Chirping for congestion control-implementation feasibility. In *Proceedings of International Workshop on Protocols for Future, Large-scale & Diverse Network Transports*, Lancaster, PA, USA, November 2010.
- [33] S. S. Kunniyur. AntiECN marking: A marking scheme for high bandwidth delay connections. In *Proceedings of IEEE International Conference on Communications*, volume 1, pages 647–651, Anchorage, Alaska, USA, May 2003. IEEE.
- [34] T. Kupka, C. Griwodz, P. Halvorsen, D. Johansen, and T. Hovden. Analysis of a real-world http segment streaming case. In *Proceedings of the 11th European Conference on Interactive TV and Video*, pages 75–84, New York, NY, USA, 2013. ACM.
- [35] K. Lan and J. Heidemann. A measurement study of correlations of Internet flow characteristics. *Computer Networks*, 50(1):46–62, January 2006.
- [36] Y. Lee, M. Atiquzzaman, and S. Sivagurunathan. Mean response time estimation for HTTP over SCTP in wireless environment. In *Proceedings of IEEE International Conference on Communications*, pages 164–169, Istanbul, Turkey, June 2006. IEEE.
- [37] D. Liu, M. Allman, K. Fall, and L. Wang. Congestion control without a startup phase. In *Proceedings of International Workshop on Protocols for Future, Large-scale & Diverse Network Transports*, pages 61–66, Los-Angeles, CA, USA, February 2007.
- [38] K. Liu and J. Y. Lee. Impact of TCP protocol efficiency on mobile network capacity loss. In *Proceedings of International Symposium on Modeling & Optimization in Mobile, Ad Hoc & Wireless Networks*, pages 1–6, Tsukuba Science City, Japan, 2013. IEEE.
- [39] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A case for a coordinated internet video control plane. In *Proceedings of ACM SIGCOMM*, pages 359–370, New York, NY, USA, August 2012. ACM.
- [40] I. Marsh. Elastic network resources: controlled caching. In *10th Swedish National Computer Networking Workshop, SNCNW*, Västerås, Sweden, June 2014.

- [41] M. Mathis. Relentless congestion control. In *Proceedings of International Workshop on Protocols for Future, Large-scale & Diverse Network Transports*, Akihabara, Tokyo, Japan, May 2009.
- [42] M. E. McLaughlin and J. Moran. Predicting bandwidth demand and network planning implications on the internet. [http://www.merit.edu/research/pdf/2004/Predicting\\_Bandwidth\\_Demand.pdf](http://www.merit.edu/research/pdf/2004/Predicting_Bandwidth_Demand.pdf), March 2014. Last Accessed: 2015-01-07.
- [43] R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Recursively cautious congestion control. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation*, pages 373–385, Philadelphia, PA, USA, June 2014. USENIX.
- [44] S. Molnar and Z. Moczar. Three-dimensional characterization of Internet flows. In *Proceedings of IEEE International Conference on Communications*, pages 1–6, Kyoto, Japan, June 2011. IEEE.
- [45] V. Oksman, H. Schenk, A. Clausen, J.M. Cioffi, M. Mohseni, G. Ginis, C. Nuzman, J. Maes, M. Peeters, K. Fisher, and P.-E. Eriksson. The ITU-Ts new G.vector standard proliferates 100 mb/s DSL. *IEEE Communications Magazine*, 48(10):140–148, October 2010.
- [46] D. Papadimitriou, M. Welzl, M. Scharf, and B. Briscoe. Open research issues in Internet congestion control. Request for Comments 6077, Internet Engineering Task Force, February 2011.
- [47] C. Partridge, D. Rockwell, M. Allman, R. Krishnan, and J. Sterbenz. A swifter start for TCP. Technical report, BBN Technologies, Cambridge, MA, BBN Technical Report, 2002.
- [48] R. Prasad, C. Dovrolis, M. Murray, and K. C. Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Network*, 17(6):27–35, 2003.
- [49] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger. TCP revisited: a fresh look at TCP in the wild. In *Proceedings of ACM SIGCOMM Conference on Internet Measurement*, pages 76–89, Chicago, Illinois, USA, November 2009. ACM.
- [50] J. Ravi, Z. Yu, and W. Shi. A survey on dynamic web content generation and delivery techniques. *Journal of Network and Computer Applications*, 32(5):943–960, September 2009.
- [51] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell. Pathchirp: Efficient available bandwidth estimation for network paths. In *Proceedings of International Conference on Passive and Active Network Measurement*, San Diego, CA, USA, April 2003. Springer-verlag.
- [52] R. Sallantin, C. Baudoin, E. Chaput, F. Arnal, E. Dubois, and A. Beylot. Initial spreading: a fast start-up TCP mechanism. In *Proceedings of IEEE Conference on Local Computer Networks*, pages 492–499, Sydney, Australia, October 2013. IEEE.
- [53] P. Sarolahti, M. Allman, and S. Floyd. Determining an appropriate sending rate over an underutilized network path. *Computer Networks*, 51(7):1815–1832, May 2007.
- [54] D. Satoh, H. Ueno, Y. Maeda, and O. Phanachet. Single PCN Threshold Marking by using PCN baseline encoding for both admission and termination controls. Internet draft (work in progress), Internet Engineering Task Force, September 2009. draft-satoh-pcn-st-marking-02.
- [55] K. Tan, J. Song, Q. Zhang, and M. Sridharan. Compound TCP: a scalable and TCP-friendly congestion control for high-speed networks. In *Proceedings of International Workshop on Protocols for Future, Large-scale & Diverse Network Transports*, Nara, Japan, May 2006.
- [56] Agilent Technologies. Introducing LTE-Advanced. <http://cp.literature.agilent.com/litweb/pdf/5990-6706EN.pdf>. Accessed: 2015-01-07.
- [57] K. Thompson, G. J. Miller, and R. Wilder. Wide-area Internet traffic patterns and characteristics. *IEEE Network*, 11(6):10–23, 1997.
- [58] K. Thompson, G. J. Miller, and R. Wilder. Wide-area Internet traffic patterns and characteristics. *IEEE Network*, 11(6):10–23, Nov/Dec 1997.
- [59] J. Touch. TCP control block interdependence. Request for Comments 2140, Internet Engineering Task Force, April 1997.
- [60] R. Wang, G. Pau, K. Yamada, M. Sanadidi, and M. Gerla. TCP startup performance in large bandwidth networks. In *Proceedings of IEEE Conference on Computer Communications*, volume 2, pages 796–805, Hong Kong, China, March 2004. IEEE.
- [61] DJ Wischik. Short messages. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1872):1941–1953, June 2008.
- [62] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman. One more bit is enough. *ACM SIGCOMM Computer Communication Review*, 35(4):37–48, 2005.
- [63] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of Internet flow rates. In *Proceedings of ACM SIGCOMM*, pages 309–322, Pittsburgh, PA, USA, 2002. ACM.

## APPENDIX

### A. TCP AVERAGE RATE MODEL

This is a model of the average rate of a single TCP flow in dedicated bottleneck capacity, defined using the following independent variables:

**Link capacity**,  $X$  [b/s]

**Round trip time**,  $R$  [s]

**Initial window**,  $i$  [pkt]

**Packet size**,  $S$  [B] or  $s = 8S$  [b]

**Flow size**,  $F$  [B].

**Flow size**,  $f = F/S$  [pkt]

**Bandwidth-delay product**,  $W = XR/s$  [pkt].

The general approach is to find the number  $n$  of round trips in which slow-start only partially fills the bottleneck, *before* the round in which either slow-start ends or the flow ends, if sooner. Then any remaining packets will arrive at the bottleneck rate, whether they are all sent in the next round while still in slow-start phase or there are enough to progress into the fast retransmit and congestion avoidance phases. In the latter case, the number of retransmissions is added to the number of packets to be forwarded before completion. One round must be added between the receiver's request and the start of the response.<sup>11</sup> Then **completion time**  $T$  between the client's request and it receiving the last data consists of:

$$T = (n + 1)R + (m + M)s/X,$$

The analysis below finds the values of  $n, m$  &  $M$ ,

**Round index**,  $n$  [integer]

**Remainder**,  $m$  is the remaining packets of the flow that all leave the bottleneck at rate  $X$

**Retransmissions**,  $M$  due to losses during the overshoot at the end of slow-start.

We define the number of packets in slow-start as  $f_s$ . Index  $u$  [real number] and the final window  $w$  [pkt] are only used to derive formulae during slow-start, outside which they are invalid:

$$\begin{aligned} \sum_{j=0}^{n-1} i2^j &< f_s \leq \sum_{j=0}^n i2^j \\ i(2^n - 1) &< f_s \leq i(2^{(n+1)} - 1) \\ n &\geq \lg(f_s/i + 1) - 1 \\ n &= \lceil \lg(f_s/i + 1) \rceil - 1; \end{aligned}$$

<sup>11</sup> Any handshaking rounds are excluded by the assumption that either the flow is re-starting after idle or using TCP fast open after an earlier connection between the same hosts.

The critical flow size,  $f_c$  is defined as the number of packets that can be sent until buffer overflow ends slow-start, and the critical round as the real number  $u_c$ . The buffer is assumed perfectly sized for the flow, at 1BDP =  $W$ . Therefore at the critical flow size, the critical window,

$$\begin{aligned} w_c &= 2W \\ &= i2^{u_c} && \text{if } 2W \geq i \\ \text{when } f_c &= i(2^{(u_c+1)} - 1) \\ &= 2i2^{u_c} - i \\ &= 4W - i && \text{if } 2W \geq i; \\ f_c &= 2W && \text{if } 2W < i; \\ f_s &= \min(f, \max(2W, (4W - i))); \end{aligned}$$

The number of packets,  $e$ , in partially empty rounds of slow-start is the number of packets in  $n$  rounds,

$$\begin{aligned} e &= i(2^n - 1) \\ m &= f - e \\ M &= \min(\max(i, 2W), \max(0, (m - 2W))). \end{aligned}$$

Explanation of the last formula for  $M$ : The buffer always has time to empty the packets sent in the round before the round in which overflow occurs (if it occurs). The number of packets dropped is the number of packets not constrained by slow-start  $m$  less the 2 BDP of packets that the link can forward or buffer, but limited by the number that will ever be sent in one round during SS or CA, which is also  $2W$  (or the initial window  $i$  if it is larger, so it can overflow the buffer on its own).

### B. PARALLEL SLOW-STARTS

If a source opens  $m$  parallel flows all using the same slow-start with initial window  $i$ , it only gives equivalent performance to one flow in slow-start using a larger initial window  $mi$  (but without all the overhead of the extra flows). The total window across all flows still only doubles in the rounds after the first.

The increase in bit-rate from using  $m$ -times larger packets is also equivalent to using an  $m$ -times larger initial window or  $m$  parallel flows (but without so much packet overhead).

Opening parallel flows (or equivalently increasing IW or packet size) only offers a modest increase in average rate. Put the other way round, to achieve a decent change in rate or speed-up ratio  $b$  requires a stupidly large number of parallel flows  $m$ .

The following analysis proves this by approximating from the model of slow-start in A on condition that  $f \gg i$ , where  $f$  is the flow-size in packets. The notation  $\bar{x}_m$  or  $T_m$  means the average rate or completion time for  $m$  parallel flows (or equivalently IW= $mi$  or  $m$  times larger packets).

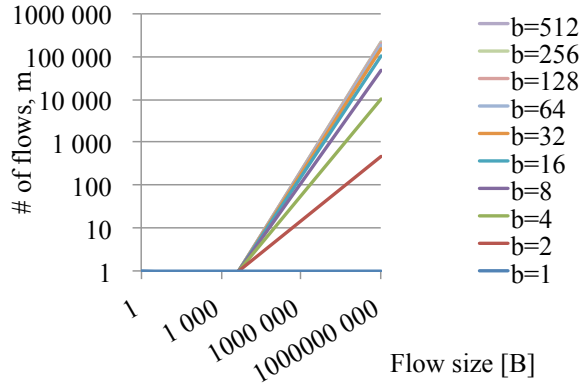


Figure 8: The number of parallel flows  $m$  required to achieve speed-up ratio  $b$  for various flow sizes  $F$

$$\begin{aligned}
 T_m &\approx \lg(f/mi); \\
 b &= \frac{\bar{x}_m}{\bar{x}_1} \\
 &= \frac{T_1}{T_m} \\
 &\approx \frac{\lg(f/i)}{\lg(f/mi)} \\
 &\approx \frac{\lg(f/i)}{\lg(f/i) - \lg m} \\
 \lg m &\approx (1 - 1/b) \lg(f/i) \\
 m &\approx 2^{((1-1/b) \lg(f/i))}.
 \end{aligned}$$

Fig. 8 illustrates this formula for the number of parallel flows  $m$  required to achieve various speed-up ratios  $b$  for a range of flow sizes  $F$  [B].

For a 100kB transfer, it can be seen that a modest speed-up ratio of  $b = 4$  can be achieved with 18 parallel flows. However, to speed up even slightly larger transfers quickly becomes stupidly infeasible. For instance, to speed-up a 1MB transfer 4 times requires 159 flows and to speed up a 10MB flow 4 times would need 1,373 flows.

Clearly it would be similarly infeasible to increase the initial window 1,373 times (to over 4000 1500B packets), because the initial window has to be set conservatively to fit easily into any size buffer that may be encountered anywhere on the Internet.

## C.2 Up to Speed with Queue View (QV)

Bob Briscoe (BT) and Per Hurtig (Karlstad University)

The central idea of Queue View is for buffers on the path to continuously encode the length of their queue by setting markers in the explicit congestion notification (ECN) field in packet headers. Figure C.1 shows packets arriving at a queue with ECT(0) set in the 2-bit ECN field of the IP header, represented by the tag ‘0’. ECT(0) is the name of the ‘10’ codepoint that a sender with an ECN-capable transport (ECT) must set by default according to RFC3168 [38]. In the remainder of this work, we will use ‘0’ to represent ECT(0) and ‘1’ to represent ECT(1), which is the term for the ‘01’ codepoint, currently set aside for experimental use. The ‘0’ & ‘1’ terminology also generalises the discussion to include any similar pair of codes in protocols other than IP.

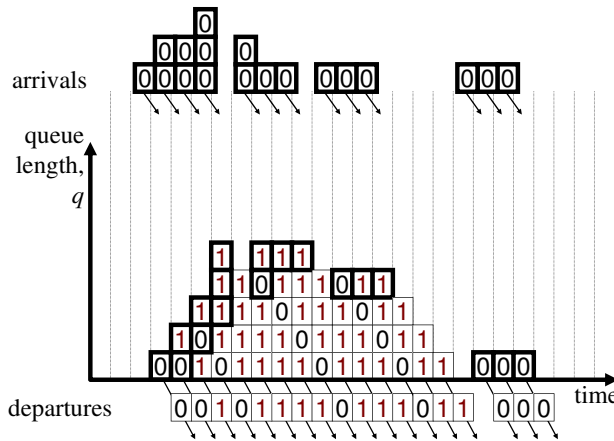


Figure C.1: The idea: Only one zero at a time in the queue makes the spacing between zeroes into a measure of queue length

Solely for illustration purposes, the figure divides time into slots along the horizontal axis and one equal sized packet leaves the buffer in each slot. Then, whenever more than one packet joins the queue within a timeslot (shown with thicker borders), the queue grows. Actually the approach does not use time-slots and works just as well with variable sized packets.

To reveal the queue length only requires a simple addition to the queue management algorithm, in which it re-marks all 0 packets to 1 as long as a 0 packet is already in the queue. Those packets that keep their 0 setting appear as diagonal stripes of 0s as they progress through the queue. Only when one 0 departs can the next stripe start, which ensures that the spacing between pairs of 0-marked packets represents a sample of the precise length of the queue at the time the second of each pair arrived. This spacing can be seen in the stream of packets departing from the queue along the bottom of the figure.

### Per-flow viewpoint

Although the process operates on the aggregate of packets in the buffer, on average the spacing between 0s within each flow still represents the queue size. This can be understood intuitively through a simple but unrealistic example: if a constant 20% of the packets belong to one flow, and the queue is a constant 10 packets long, on average 1 in 10 packets will be 0 within the 20% belonging to that flow, as well as within the aggregate.

Happily, when a flow has passed through multiple QV-enabled buffers, the resulting spacing between 0 markers will roughly represent the length of the longest queue on the path. To understand why<sup>10</sup>, consider a flow passing through

<sup>10</sup>Formally, for constant rate flows and two constant queues,  $q_1$  &  $q_2$  in series, the resulting spacing between 0 markings is:

$$\begin{array}{ll} q_1; & \text{if } q_1 \geq q_2 \\ \lceil q_2/q_1 \rceil q_1; & \text{if } q_2 \geq q_1 \end{array}$$



two constant queues of 17 and 5 packets in series:

**If the longer queue is first**, on average 1 in 17 packets within the flow arriving at the second queue will be 0, so the second buffer will rarely re-mark any more packets to 1, except in the occasional random cases when the arriving spacing is less than 5 packets.

**If the longer queue is second**, on average 1 in 5 packets arriving at the second queue will be 0, so once the second queue has allowed in one 0 it will re-mark any 0s within the next 17 packets to 1. So, about 1 in 20 (the next multiple of 5) packets in this flow will emerge still marked as 0.

## Simple implementation

Various algorithms to encode queue length into the spacing between markers can be devised; all very simple. The one in [Figure C.1](#) illustrates the idea most clearly but it may sometimes be impractical because it sets ECN codepoints on enqueue, and therefore has to share a variable with the dequeue process to flag when each 0 packet departs. The pseudocode below does the re-marking in the four lines added at the end of the regular dequeue process. It will often be preferable because it uses time<sup>11</sup> as the shared variable by solely requiring the enqueue process to add an incoming timestamp (`tsi`) to the internal representation of each packet (inspired by PDPC [65] and CoDel [7]). This is ideal when enqueueing and dequeuing occur in completely different parts of a switch or router, or potentially even in different machines.

```
enqueue(packet) {
    write(time(now), packet.tsi)
    % regular enqueueing code here
}

dequeue(packet) {
    % regular pre-dequeue code here
    if (read(packet.ECN) == ECT0) {
        if (packet.tsi > t0) {t0 = time(now)}
        else {write(packet.ECN, ECT1)}
    }
}
```

## Using QV Signals

To be worthy of deployment in general purpose network equipment, QV should be useful for multiple higher layer applications. We plan to develop some of the initial rough ideas below.

### Flow (re-)start

We assume that the receiver reflects the stream of arriving 0s and 1s back to the sender (see later). Imagine a scenario where a TCP sender has already initialised the connection with a SYN, SYN-ACK exchange, so it has an initial estimate of the round trip time (RTT).<sup>12</sup> Then imagine that the sender immediately starts one burst of five packets back-to-back and another five-packet burst half an RTT later, all ECN-capable and set to the default of 0.

If the first ACK of either burst reflected a 1, the sender would know with certainty that it was not alone in the bottleneck and it should fall-back to a more cautious start, e.g. TCP slow-start. Otherwise, having taken two samples of the path capacity spaced evenly across the round trip, the sender should be able to calculate the rate at which it can pace packets throughout the next round, with increased confidence that it will not exceed available capacity. This should result in two simultaneous improvements:

The rounding up in the second case is on the safe side and anyway, in more realistic scenarios with varying flow rates, the spacing approaches the desired length of the longer queue,  $q_2$ .

<sup>11</sup>Or any monotonically increasing counter.

<sup>12</sup>It may be using TCP Fast Open, or the connection may be re-starting after idle, in which cases the RTT estimate may have been established some considerable time earlier.

- For the flow in question it should allow start-up latency to approach a single round trip time regardless of available capacity;
- For other competing sessions and for the flow in question, it should largely eliminate buffer overshoot that otherwise does so much harm, particularly to real-time traffic

For example, let us assume that the feedback from these two bursts tells the sender that they arrived at the receiver as: 0,1,1,0,1; 0,1,1,0,1 with a longer interval before the sixth ACK than all the others.<sup>13</sup> The presence of 1s tells the sender that it has induced a queue somewhere on the path. The initial 0 implies it is likely that the buffer was empty<sup>14</sup> when the first packet arrived, and the initial 0 of the second burst after a pause increases this likelihood, so that the flow can more safely assume it is alone in the bottleneck. The fact that the two sequences of five are identical strongly implies that the available capacity at the bottleneck is constant.

If available capacity is constant and if the rate of each burst were  $n$  times faster and constant, the spacing between 0s within each burst would form the series:  $n, n^2, n^3, \dots$ . So, the spacing of 3 packets between the 0s in both bursts implies  $n > 2, n \leq 3$ . Then the sender could start continuously pacing out a full second round of packets 3 times more slowly than the back-to-back burst rate used in the first round, being the conservative end of the estimated range. It could even start this pacing after the first burst of five ACKs, then continue if feedback from the second burst concurred with the first, or modify its estimate otherwise.

One of the most elegant aspects of the original TCP is that it self-clocks packet departures without timers. Therefore a strategy for pacing will be needed that minimises the maintenance of timers, which would otherwise place a demanding load on the sending machine. Another alternative is to pace in hardware, similar to that developed for HULL [66].

It would be nice to emulate another elegant feature of slow-start; a new flow can approach parity with other traffic before it experiences any signals (losses) itself. As a new flow sends more and more, others may still be using the rest of the bottleneck. If the incumbent flows are still using more than the new flow, by the laws of probability they will be more likely to experience losses and yield capacity that the new flow can grow into, with less chance of experiencing the losses itself.

In the second round, perhaps the sender could adopt the strategy of only half filling its best estimate of the window. Then by the third round, having sampled the bottleneck for about half of a round trip, it would have the best possible estimate of the available capacity without having increased its rate any more than the largest increase that TCP slow-start might do before discovering other traffic at the bottleneck. The QV source could send the half window of packets in line-rate bursts of (say) 8 packets, pacing the start of each evenly across the round trip. Each burst would pulse the queue to test the available capacity, but leave a gap to allow the induced queue to drain. And each burst would be no worse than the initial window of a new flow with  $IW=8$ .

In the third round, having calculated a much better estimate of the congestion window from all the QV signals, the source could then use the rate of returning ACKs to pace out subsequent data, and divide the remainder of the calculated congestion window over the number of gaps, sending each burst of remaining packets out back-to-back, which would establish the ACK-clock for the fourth round without any line-rate bursts of more than 8 packets.

Returning to the very first window of packets, the best strategy for getting most information from QV at this critical stage is where further research will need to focus. For instance, consider the following dilemma:

- It may be better to send the initial window all in one burst so that the sender can estimate  $n$  more precisely. Assuming a burst of 10 packets for example, if the feedback were 0,1,1,0,1,1,1,1,1 the sender could infer from the lack of a third 0 that  $n^2 \geq 7$  (i.e.  $n \gtrapprox 2.65$ ), as well as inferring  $n > 2, n \leq 3$  from the first pair of 0s. Therefore it can narrow its estimate to  $2.65 \leq n \leq 3$ .
- However, using two bursts half an RTT apart allows the sender to test with more confidence whether it is alone on the link—whereas a leading 1 implies it is definitely not alone, a leading 0 only implies it is probably alone.<sup>15</sup>

This is a dilemma between knowing more precisely how aggressively to behave *vs.* knowing whether to take a more

<sup>13</sup>One ACK per sent packet shows the receiver is not using delayed ACKs, at least not during slow-start.

<sup>14</sup>Empty of ECN-enabled traffic, at least (see Section C.1).

<sup>15</sup>With probability  $1/q$  of being wrong, where  $q$  is the spacing between 0s. Testing twice and getting a leading zero both times reduces the probability of being wrong to  $1/(q_1 q_2)$ , where  $q_1$  &  $q_2$  are the respective spacings.

aggressive approach in the first place. To judge which is more important in order to implement an initial window behaviour for the general case will require considerable analysis and experimentation.

### Chirping

In order for the sender to extract the most information from one round of QV feedback it would be ideal to send chirps rather than back-to-back bursts. A chirp is a sequence of packets in which the inter-packet spacing is progressively reduced. This was first introduced in the PathChirp tool [67] and it is the best-known technique to test a wide range of rates with the least disruption to any existing traffic. Chirping requires a demanding number of timers to be maintained, but implementation with high resolution timers in Linux has proved feasible [68].

Chirping would not only be appropriate in the first round. Once the sender has approximated the operating rate of the path, if it sends even slightly too slowly, it will get hardly any information from QV; just a sequence of 0s. If instead it sends with the same average rate, but continuously in chirps, it will be able to quickly determine how much to increase its pace in the next round. Such continuous chirping has already been experimentally implemented in the Linux TCP stack by Kühlewind [68], but using delay to infer the queue, not QV. This strategy would be particularly useful for rapidly filling newly freed up capacity after a competing flow finishes.

### Distinguishing Congestive Loss from Other Losses

QV offers another important benefit over traditional congestion signalling protocols; it allows the source to distinguish whether losses are due to congestion or other causes such as radio interference, with near-certainty. Currently, a source has to conservatively assume any loss might be due to congestion and conservatively reduce its rate—typically by half. If a loss was actually due to a transmission error (e.g. due to radio interference), and the source had known, it would not need to have reduced its rate at all. This uncertainty currently leads to very poor utilisation of some links, particularly wireless links where radio interference is a common problem.

It has been suggested that standard explicit congestion notification (ECN) could also be used to distinguish congestion from other losses. If ECN-capable packets are being sent and a loss arises without any ECN marking on prior packets, it is argued that the source could assume the loss was non-congestive. However there are two flaws in this approach:

- A sharp rate increase from another source can cause a buffer to drop traffic from its tail well before any ECN signals, so ECN cannot be assumed to precede all congestive losses. This is because most active queue management algorithms delay ECN signals considerably by only signalling ECN when a smoothed moving average of the queue length exceeds a threshold.
- Absence of ECN signals before a loss may merely indicate that the bottleneck does not support ECN. Even ECN signals seen during the same connection do not prove the bottleneck supports ECN, because they may come from a different bottleneck.

A bottleneck sends QV signals without smoothing or delay. So if the source detects a loss or burst of losses at any stage in the sending of a flow of packets, and if the queue length both before and after the episode of loss is very short, the source can assume with high probability that the loss was not due to queue overflow. In such a case it would not need to reduce its rate at all.

This still seems to suffer from the second flaw of the ECN approach, where absence of a longer queue around a loss episode may merely indicate that the bottleneck does not support QV. Nonetheless, as we shall see later (Section C.1), a source should be able to test whether QV signals are emanating from the operative bottleneck by checking whether the increase in delay being experienced is of the same order as that expected from the QV signals. Whether such double-checking would be robust enough in practice would need to be verified experimentally.

### Self-admission control

An inelastic or semi-elastic class of applications could send a brief chirp of probe packets and use QV<sup>16</sup> to test whether there was sufficient available capacity to start a new session, e.g. in a premium traffic class. Compared to sending a burst at the desired rate or rates, this would introduce minimal disruption to existing traffic.

<sup>16</sup>Complemented by estimates of queueing delay—see Section C.1.

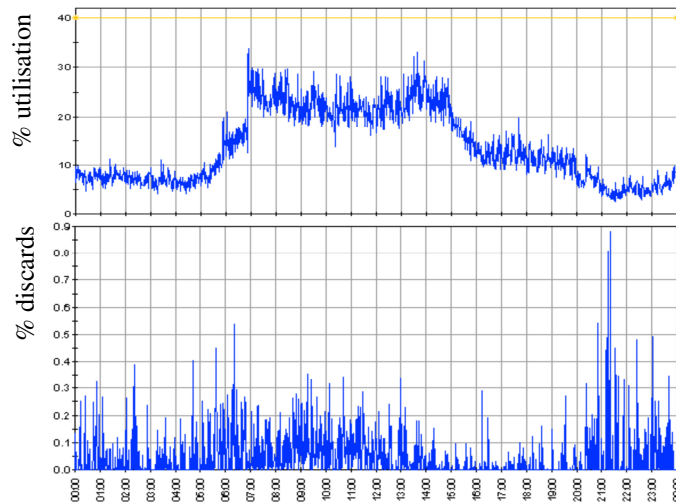


Figure C.2: 24hrs of utilisation and discard metrics averaged at 15 min granularity at an example link

### Capacity planning

Many network operators monitor and record link utilisation in order to decide when to upgrade capacity. However, utilisation is not the right metric because it has to be averaged, so the lower utilisation periods cancel out the higher ones during the measurement period. Figure C.2 illustrates this point with utilisation and discard metrics averaged at 15-minute granularity over a 24hr period at a link in a live enterprise network. The averaged utilisation metric is actually at its lowest when loss is highest.

To identify where more capacity is needed, loss is a better metric to monitor than utilisation. However, operators often want very low or zero loss—particularly where non-TCP applications may be involved—which is why they monitor utilisation, thinking that once it hits a certain threshold they will upgrade capacity to preempt any loss. In scenarios where loss is undesirable, some vendors attempt to improve the usefulness of utilisation monitoring by averaging at finer and finer granularity—even down to microseconds for demanding finance applications. It becomes extremely expensive to log all this data at every interface, then transport it all to a central store for analysis.

When in a hole, stop digging... Such ultra-fine granularity utilisation attempts to tell the operator whether and how much the traffic is instantaneously exceeding link capacity. Whenever it does, even for a brief moment, a queue will form. Monitoring average queue length via QV markings will pick up even tiny two-packet queues, and should be a better approach to predict when capacity needs to be upgraded:

- rather than loss statistics, which only reveal occasions when traffic was heavy enough to exceed both link and buffer capacity;
- rather than utilisation statistics, which allow brief low periods to cancel out periods at 100%.

A network operator could deploy QV in buffers and passively monitor ECT markings leaving the network. Even for non-ECN-capable customer traffic, the operator could turn on ECN in an outer L2 header (or a tunnel outer) and monitor QV markings per source address at the egress of the subnet (or tunnel), while forwarding the encapsulated IP packet to the customer in its original state with ECN turned off. Not only would this monitor a correct capacity planning metric, but it would carry the bulk of the monitoring in-band, then the management network would only be needed for targeted monitoring<sup>17</sup>, rather than carrying copies of nearly every header.

Logging of the spacing between 0s would also be an efficient way to record network performance, because long sequences of 0s would consume only one log entry over entire periods when no queueing was present to be recorded.

### Active performance monitoring

<sup>17</sup>Perhaps triggered by rising QV metrics.

Increasingly communications regulators and network operators are using volunteer customers to deploy probes that monitor the network's performance. QV should minimise the amount of traffic needed while maximising the information gained.

### Distinguishing base delay from queuing delay

At the receiver, the time from each 0 marker to the next gives a continual measure of queueing delay.<sup>18</sup> With QV enabled in both directions, the sender can monitor one-way queueing delay in both outward and return directions in the transport feedback and IP layer headers respectively. A sequence of 0s shows it is highly likely there was no queueing. A sequence of 0s in both directions gives the sender near-certainty that the RTT measurement is the base delay.

The network operator can also monitor QV signals leaving the network to measure queuing latency and monitor whenever queueing arises. As with utilisation monitoring above, QV can be used instead of monitoring latency at finer and finer granularity. The operator only needs to measure the base delay occasionally, and recording QV spacing will monitor additional queueing delay. This will inherently use the finest granularity necessary, e.g. a 40B packet takes 8 nanoseconds to serialise at 40Gb/s, so QV will intrinsically measure the queueing delay down to 8 nanosecond granularity.<sup>19</sup>

### Incremental Deployment

A QV-enabled buffer could safely introduce QV markings whether or not the receiver or sender would understand them or act on them, because the proposed uses of QV are optional optimisations. The proposal above to build QV into ECN unnecessarily ties the fate of QV to deployment of ECN, which has been painfully slow or non-existent. However, the alternative of making QV independent of ECN would introduce a further set of problems:

- There is no obvious alternative field in IPv6, IPv4 and link layer protocol headers;
- Any alternative field would have to be arranged to propagate upwards from lower layers and from outer tunnel headers;
- QV feedback would have to be added to all the major transport protocols, separately from ECN;

Looked at another way, the benefits of QV seem stronger than those of ECN, so it may pull ECN deployment in its wake, rather than hang waiting for ECN.

### Pre-requisite feedback

In order for the sender to exploit QV markings, it must first receive end-to-end feedback that distinguishes ECT(0) from ECT(1) markings arriving at the receiver. In the TCP protocol, as well as in more recent transport protocols (DCCP, SCTP), the experimental 1-bit nonce sum [69] is insufficient for this purpose (and anyway it has never been deployed). Nonetheless, the IETF recently adopted an activity onto its standards agenda to augment TCP with more accurate ECN feedback [70]. Each host looks at the TCP flags set by the other end during connection initiation to know whether it offers enhanced feedback. The RTCP protocol has recently been enhanced [71] to report a count of ECT(1) feedback separately from ECT(0), which would work for QV, but only if the feedback was sufficiently frequent.

### Non-QV mixed with QV buffers

In common with all known faster-start schemes, there will be a possibility that the bottleneck buffer does not support the scheme. This means a QV source should proceed with caution, which is irritating given the purpose of QV is to signal when a source can proceed *without* such caution. Quick-Start [72] introduced a Quick-Start hop counter so it could check that all the hops counted by the IP TTL were also Quick-Start-enabled. However, that would be too

<sup>18</sup>Assuming a single bottleneck on the path.

<sup>19</sup>This is not surprising when one thinks back to the implementation in Section C.1, which signals a 0 whenever the arriving timestamp exceeds the time the previous zero was signalled.

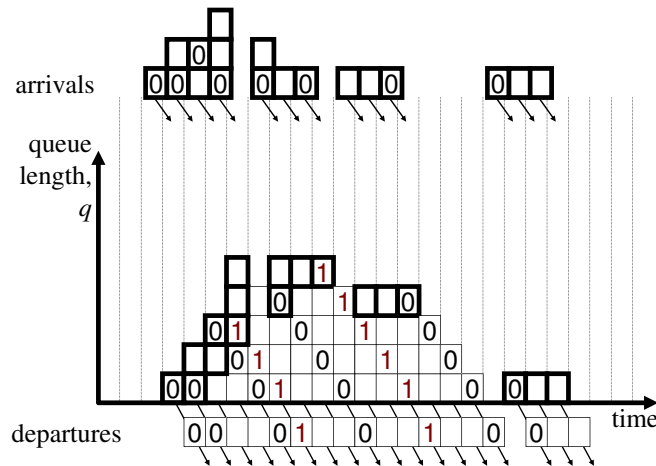


Figure C.3: QV mixed with non-QV traffic (shown untagged)

conservative for QV, given only potential bottlenecks need to be QV-enabled and the following strategies are available to allow a source to detect whether bottleneck(s) on the path support QV:

1. If the source sends a burst of 0s at its maximum line-rate and the ACKs return at less than its own line-rate but reflect no 1s at all, the source should assume that it is bottlenecked by a buffer that does not support QV and revert to a more cautious start, e.g. slow-start.
2. However, even if some 1s are returned, a source may be bottlenecked by a buffer that does support QV followed later on the path by a tighter bottleneck that does not. Therefore, if a source uses QV feedback to calculate a tentative rate for the next round that spaces packets significantly closer together than the ACK spacing from the previous round, it should disregard the QV signals.
3. A source could be configured with a policy that it is part of a private network universally configured with QV, which would override any need for the above cautionary rules.

### Non-QV mixed with QV traffic

With the proposed approach, the spacing between 0-marked packets represents the queue length solely of the subset of packets in the queue that are ECN-capable, as illustrated in Figure C.3. This causes QV to scale down the queue length it reports by the fraction of QV traffic. Therefore QV traffic is less sensitive to non-QV traffic, but not completely insensitive. For instance, if one imagined that the non-QV packets in Figure C.3 were absent, there would be no queue at all, and the QV traffic would see all 0s, whereas QV does report a queue when the non-QV packets are there.

Whether this design choice is reasonable requires further investigation, but it was deliberately chosen<sup>20</sup> to reflect the two main behaviours of QV traffic:

- a QV source is expected to start more aggressively when it believes there is no other traffic in the bottleneck;
- a QV source attempts not to overshoot, but it doesn't want to starve itself when non-QV sources overshoot.

### Design Choices

#### Dimensionless signal

<sup>20</sup>Although it would be difficult to do otherwise because non-ECN packets can neither signal a 1 nor a 0.

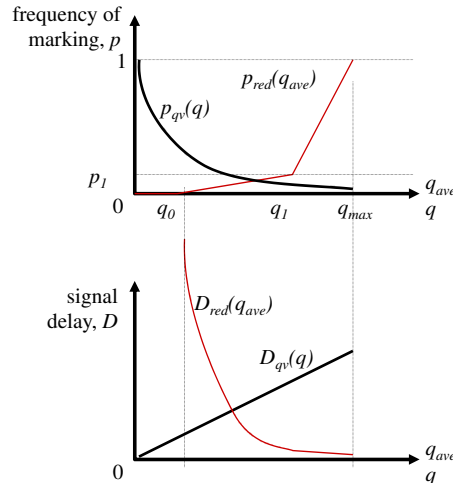


Figure C.4: The delay between signals is very low when the queue is shortest with QV, but when it is longest with RED

Importantly, an observer of a stream of QV packets can measure the queue length in packets, bytes or time, by counting the packets, bytes or time between ECT(0) markers. The signal itself does not prejudge its units, which can be chosen unilaterally by the system using the signal.

### Zero-config

QV is inherently zero-config—not only does it not require any configuration, but there is nothing that could possibly need to be configured. So, whatever QV signals are used for, they will be unambiguously understood as queue length<sup>21</sup>. QV uses no units, no arbitrary thresholds, no smoothing periods, no weights, no slopes, no delay intervals—nothing. Any filtering or thresholds are left to be introduced by the end-systems when they interpret the signals, not by the buffer when it sends the signal—in line with the end-to-end principle [73].

### Cross-layer

Although the encoding of QV signals has been described in the context of IP, it could be defined for any lower layer protocol too, e.g. MPLS, Ethernet, using similar explicit congestion notification facilities, any of which can be propagated up to the IP layer as described in a recent draft posted to the IETF [62]. The QV algorithm can change ECT(0) to ECT(1) but not the reverse, which is a required condition to work correctly through tunnels [40].

### Approaching infinite scaling

Currently the latency of TCP’s slow-start scales with the log of available capacity—every doubling of link capacity adds one more round trip to latency. It would be ideal if latency were no more than one round trip, independent of future capacity growth. QV signalling has been designed to add no delay, so as not to preclude the possibility of approaching that ideal.

Firstly, unlike most AQM schemes, QV does not attempt to filter out any queue variation that is faster than the ‘typical’ round trip time response of end-systems, because filtering implies delay for any end-systems using a shorter RTT. Instead, QV leaves end-systems to smooth the signal, because they know their own round trip time. This makes QV applicable over any RTT, whether between planets or between the cores of a processor.

Secondly, at the very first sign of queue growth, QV avoids the signal delay that a unary encoding inherently introduces.<sup>22</sup> The upper plot in Figure C.4 shows that QV departs from the traditional approach of increasing the frequency

<sup>21</sup>See Appendix C.1 for a critique of prior proposals on this score.

<sup>22</sup>A unary encoding is necessary because it allows a buffer to signal to flows without any flow state.



of marking as the queue grows; instead the QV encoding *reduces* marking frequency as the queue gets longer.<sup>23</sup> The lower plot shows why; to signal a number using a unary encoding takes proportionately longer the higher the number (for the same precision).

For instance, to signal the number 12 requires 12 packets-worth of time. As the queue grows, an AQM like RED starts to move up its well-known ramp. It signals a probability such as 0.01% or 1/10,000 as a unary encoding of the number 10,000, which takes of the order of 10,000 packets. That is reasonable if the buffer forwards of the order of 10,000 packets in the typical RTT of any of the flows involved, because at least one of them will get the signal within its RTT.<sup>24</sup> In contrast, QV aims to signal to perhaps a single flow that has not yet opened up a full window, so the algorithm may only handle a few packets in a round trip. And, even if there are many flows at the bottleneck, if one new flow arrives and starts to induce a queue, QV tells all the flows immediately, not just one randomly picked one, because only the new flow will be watching for QV signals and we don't want the buffer to have to detect which is the new flow. This is why QV's encoding is designed the other way up—to encode the signal with least delay for a short queue length—over one or two packets, rather than the tens of thousands that an AQM would use.<sup>25</sup>

### Spoof-resistant

Encoding queue length in-band (in packet headers) creates an inherent security binding between the signals and the data packets. Otherwise, if signalling were out-of-band (as in ICMP Source Quench [74] or Quick-Start [72]) it would have to be cryptographically bound to the data so that spoofed signals could be detected.

### Competing uses of ECT(1)

The ECT(1) codepoint is one of the few unused codepoints left in the IPv6 & IPv4 header, but it is not unassigned. Therefore, a cast-iron case will need to be made to standardise a new use. ECT(1) was originally proposed so that an ECN sender could encode a nonce [69] one bit at a time into a stream of packets. Then the sender could detect whether ECN receivers were suppressing congestion feedback in order to fool the sender into transferring faster than it should. The ECN nonce still has experimental status, although it has never been deployed, so there would at least be no backward compatibility problems if it were decided to reassign the ECT(1) codepoint for use by QV.

### Signal integrity

QV, like ECN more generally, depends on all the parties around the feedback loop maintaining the integrity of the signal. Network nodes signal queue growth via downstream networks to the receiver, which in turn signals feedback to the sender, which in turn is meant to respond to the signals.

The IETF is in the process of standardising an experimental approach called congestion exposure (ConEx [75]) where the sender is expected to reinsert the feedback it receives within subsequent packets it is sending. Then egress network nodes can audit the 're-echoed' signals against those they originally set, assuring the signal's integrity so that ingress nodes can use it to police the sending behaviour of sources.

The ConEx protocol could easily be extended to cover the integrity of ECT(1) signals, not just CE and discards as at present. However, such an extension should be unnecessary because CE and loss levels will strongly depend on QV signals. This is because QV is intended to allow a sender to avoid causing the massive buffer overshoots that cause huge bursts of loss or 'CE' ECN signals. Therefore simply policing CE and loss with ConEx should be sufficient to prevent anyone in the feedback loop lying about the ECT(1) signals of QV.

Alternatively, Moncaster *et al* [53] proposes that the sender can unilaterally test the receiver's feedback integrity, by occasionally setting a codepoint that only the network would normally be expected to set, then checking that the receiver faithfully reflects it back. Like the nonce, this approach only addresses the integrity of the receiver, not all

<sup>23</sup>In a unary encoding, one of the two symbols must be defined as the marker, and the other as the spacer. Here we are talking about the frequency of the markers, irrespective of whether 1 or 0 is used as the marker—in fact QV uses 1 as the marker whereas AQM schemes use 0.

<sup>24</sup>This only scales if congestion control algorithms make their window proportional to  $1/p$ , which they are evolving towards but have not quite reached; e.g. Reno, Cubic and Compound use respectively  $1/p^{0.5}$ ,  $1/p^{0.75}$  &  $1/p^{0.8}$ .

<sup>25</sup>PDPC and CoDel claim to remove the delay introduced when encoding large numbers in unary because they trigger the first signal deterministically rather than randomly. Although this approach removes the delay, it also removes the signal spacing information until the second signal arrives.



parties in the loop as ConEx does. But it can be unilaterally deployed by senders, it can test ECT(1) and CE without consuming any codepoints itself and it requires no standardisation.

## Functional Tests

To determine whether the simple idea behind QV actually would work as expected we conducted a series of functional tests where we extended ns-2's droptail queue to perform ECT marking according to the QV algorithm. We then tested how well the sequence of 0s and 1s present in packet headers corresponded to the actual queue length of the longest buffer in the path. We looked at the sequence of 0s and 1s both from an aggregated viewpoint and from the viewpoint of the individual flows traversing the network path. The reason for looking at both the aggregate and the individual flows is to see whether the queue length information is correct on both these scales.

In the coming graphs we show the behavior of QV by plotting the queue length, in packets, as seen by arrivals at the bottleneck. Over this curve, we show how incoming packets are marked 0 or 1, depending on the number of 0s already in the queue. The number of 0s in the queue, at both arrival and departure are also shown. We also illustrate how QV signals the queue lengths to the endpoints, as encoded in the spacing (in packets) between consecutive 0s, shown when the last 1 of a series leaves the bottleneck; the x-axis position of the impulse indicates the departure time, and the height of the impulse indicates the number of 1s in a row, between two successive 0s.

Figure C.5 shows the results from running a single TCP flow over a single bottleneck. For this particular experiment we forced the TCP sender to start with a very high initial window (well over the total buffering capacity of the bottleneck). This allowed us to determine whether QV works for both quickly changing queue sizes as well as very stable ones (which happens after TCP has reacted to the initial buffer overshoot). As shown in the graph, QV's estimated queue length is both accurate and, as indicated by the spacing between impulses, delivered roughly once every RTT (the RTT being approximately 20ms in this scenario).

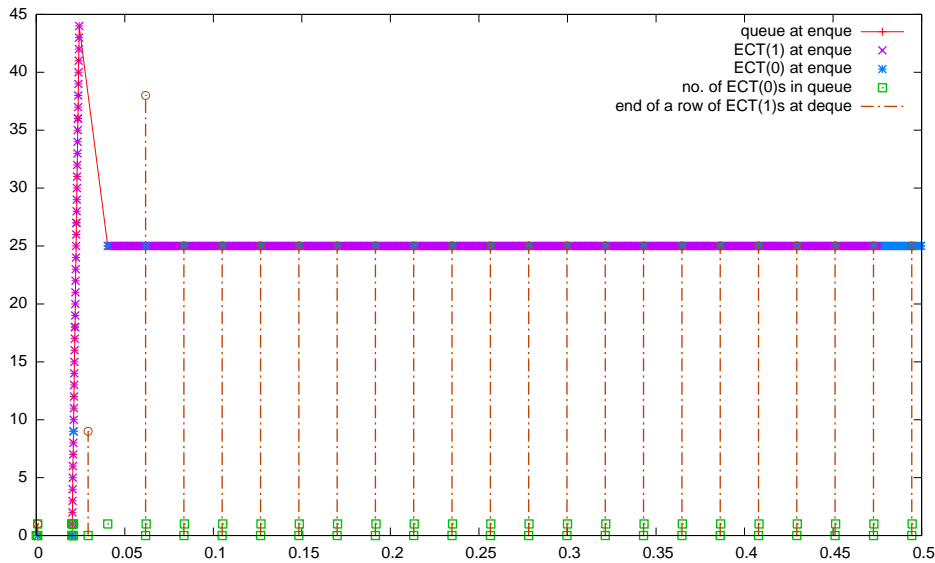


Figure C.5: A single long-lived flow traversing a QV-enabled bottleneck.

Figure C.6 shows the results in a multi-flow scenario where 10 flows share a bottleneck. For these experiments we did not force the flows to overflow the buffer artificially but rather used the default slow-start mechanism in TCP. The aggregated view of QV feedback is shown in Figure C.6a. From the graph it is evident that QV is able to correctly estimate the actual queue length at regular intervals. However, this is not the case for individual flows. Shown in Figure C.6b is the viewpoint from one of the individual flows. It is clear that the signal propagated back to the endpoints is sometimes incorrect, and also very sparse. This happens simply because of the high level of multiplexing at the bottleneck which makes the sampling intervals for individual flows too long.

We've also seen similar effects when looking at topologies with multiple bottlenecks – the periodicity and the correctness

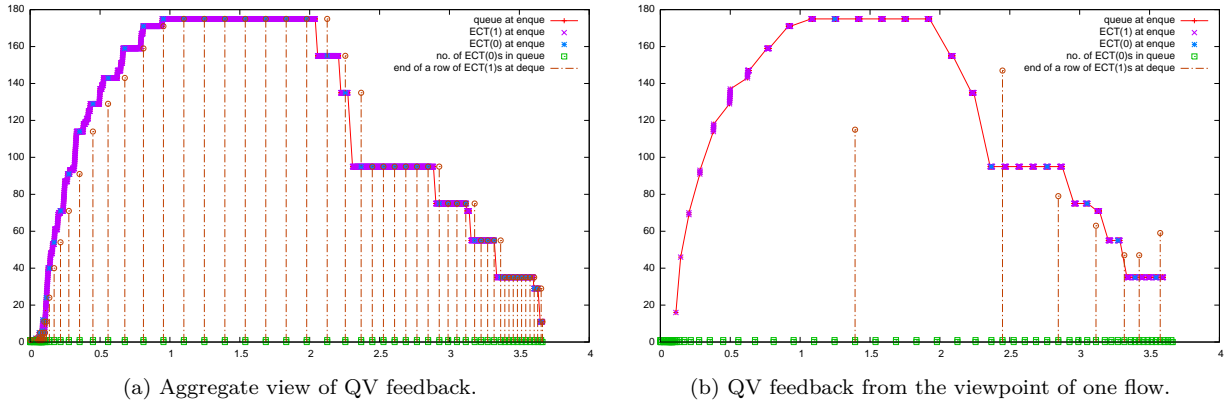


Figure C.6: 10 TCP flows successively arriving at, then successively leaving a QV-enabled bottleneck.

of the QV feedback given to individual flows become less qualitative as the number of bottlenecks increase.

Thus, so far we have been able to determine that QV works well in scenarios when a flow enters a previously empty buffer, or in scenarios where the aggregated information from QV can be used. However, we still need to run more experiments to determine whether the current QV mechanism is sufficient to give flow endpoints the required amount of information to more efficiently adjust its sending rate in a general scenario.

### Challenges and next steps

Attempts to use similar approaches to QV (see related research in Appendix C.1) have made partial improvements in some scenarios but not all. We plan to establish whether QV can do better, on the basis that QV gives critical information that delay measurements alone lack:

1. QV reveals whether the flow is alone in the bottleneck. Slow-start has to behave conservatively on the assumption that competing flows may be increasing in parallel, whereas QV can test whether this is a likely scenario, and behave much more aggressively if it is not.<sup>26</sup>
2. QV unambiguously signals the distinction between base delay and queuing delay, whereas delay measurements take considerable time to establish this distinction, and even then they can be wrong.
3. QV unambiguously signals precise queuing delay, whereas delay measurements include considerable noise, particularly when measuring queuing delay at a high speed link that contributes a very small proportion of the overall round trip delay, but a large proportion of the available buffer.

To establish whether QV can do better will require a number of items of further work, listed below with their status:

- Implement QV (done for simulation (ns2); still to be done for real (e.g. in Linux));
- Functional testing of whether QV behaves as predicted with multiple flows and multiple bottlenecks (done Section C.1);
- Developing a general purpose probing strategy for the initial window of a TCP flow that will most often extract the most QV information for the next round (done in outline, but to be developed fully and written up);
- Proving that the benefits of QV cannot largely be achieved without QV by merely measuring delay deltas (dependent on success of probing strategy item);

<sup>26</sup>In a dedicated link, one of the most common causes of competing traffic is when an application opens multiple flows in parallel. With QV, all but the first will detect the others, so no more than one flow will infer that it can behave aggressively.

- Testing that the strategies for deploying QV incrementally are safe (dependent on success of probing strategy item);
- Developing a general way for a flow in progress to use QV to quickly detect and use new capacity that has been freed up, e.g. when a competing flow finishes (only if successful in slow-start phase);
- Ensuring the standardisation work in progress on accurate ECN feedback meets QV's requirements (done Section 4.4);
- Standardisation of the QV protocol (dependent on success of probing strategy item).

The immediate next steps related to the QV mechanism are twofold.

- determine how to make QV more robust in multi-flow/multi-bottleneck scenarios;
- how to use QV-like signals at the endpoints to more efficiently adjust the sending rate of a flow that calculates that it is alone at the bottleneck.

These tasks will be conducted in parallel.

Internet Engineering Task Force (IETF)  
Request for Comments: 7560  
Category: Informational  
ISSN: 2070-1721

M. Kuehlewind, Ed.  
ETH Zurich  
R. Scheffenegger  
NetApp, Inc.  
B. Briscoe  
BT  
August 2015

Problem Statement and Requirements for Increased Accuracy  
in Explicit Congestion Notification (ECN) Feedback

Abstract

Explicit Congestion Notification (ECN) is a mechanism where network nodes can mark IP packets, instead of dropping them, to indicate congestion to the endpoints. An ECN-capable receiver will feed this information back to the sender. ECN is specified for TCP in such a way that it can only feed back one congestion signal per Round-Trip Time (RTT). In contrast, ECN for other transport protocols, such as RTP/UDP and SCTP, is specified with more accurate ECN feedback. Recent new TCP mechanisms (like Congestion Exposure (ConEx) or Data Center TCP (DCTCP)) need more accurate ECN feedback in the case where more than one marking is received in one RTT. This document specifies requirements for an update to the TCP protocol to provide more accurate ECN feedback.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <http://www.rfc-editor.org/info/rfc7560>.

RFC 7560                      Requirements for More Accurate ECN Feedback                      August 2015

## Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Terminology . . . . .	4
2. Recap of Classic ECN and ECN Nonce in IP/TCP . . . . .	5
3. Use Cases . . . . .	6
4. Requirements . . . . .	8
5. Design Approaches . . . . .	11
5.1. Redefinition of ECN/NS Header Bits . . . . .	11
5.2. Using Other Header Bits . . . . .	13
5.3. Using a TCP Option . . . . .	13
6. Security Considerations . . . . .	14
7. References . . . . .	14
7.1. Normative References . . . . .	14
7.2. Informative References . . . . .	14
Appendix A. Ambiguity of the More Accurate ECN Feedback in DCTCP	16
Acknowledgements . . . . .	17
Authors' Addresses . . . . .	17

RFC 7560                      Requirements for More Accurate ECN Feedback                      August 2015

## 1. Introduction

Explicit Congestion Notification (ECN) [RFC3168] is a mechanism where network nodes can mark IP packets instead of dropping them to indicate congestion to the endpoints. An ECN-capable receiver will feed this information back to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). This is sufficient for preexisting TCP congestion control mechanisms that perform only one reduction in sending rate per RTT, independent of the number of ECN congestion marks. But recently proposed or deployed mechanisms like Congestion Exposure (ConEx) [RFC6789] or Data Center TCP (DCTCP) [DCTCP] need more accurate ECN feedback than 'classic ECN' [RFC3168] to work correctly in the case where more than one marking is received in any one RTT.

For an in-depth discussion of the application benefits of using ECN (including with sufficiently granular feedback), see [ECN-BENEFITS].

ECN is also defined for transport protocols beside TCP. ECN feedback as defined for RTP/UDP [RFC6679] provides a very detailed level of information, delivering individual counters for all four ECN codepoints as well as lost and duplicate segments, but at the cost of high signalling overhead. ECN feedback for SCTP has been proposed in [SCTP-ECN]. This delivers a counter for the number of ECN-capable packets that were marked due to congestion (since the last sender-side window reduction), but it comes at the cost of increased overhead.

Today, implementations of DCTCP already exist that alter TCP's ECN feedback protocol in proprietary ways (DCTCP was released in Microsoft Windows 8, and implementations exist for Linux and FreeBSD). However, the changes DCTCP makes to TCP omit capability negotiation, relying instead on uniform configuration across all hosts and network devices with ECN capability. A primary motivation for this document is to intervene before each proprietary implementation invents its own non-interoperable handshake, which could lead to *de facto* consumption of the few flags or codepoints that remain available for standardizing capability negotiation.

This document lists requirements for a robust and interoperable TCP/ECN feedback protocol that is more accurate than classic ECN [RFC3168] and that all implementations of new TCP extensions, like ConEx and/or DCTCP, can use. While a new feedback scheme should still deliver as much information as classic ECN, this document also clarifies what has to be taken into consideration in addition. Thus, the listed requirements should be addressed in the specification of a more accurate ECN feedback scheme. A few solutions have already been

proposed. Section 5 demonstrates how to use the requirements to compare them, by briefly sketching their high-level design choices and discussing the benefits and drawbacks of each.

The scope of these requirements is not limited to any specific environment and is intended for general deployment over public and private IP networks. Candidate solutions should try to adhere to all these requirements, but, where this is not possible, they should justify the deviation. The ordering of the requirements listed in this document is not to be taken as an order of importance, because each requirement might have different weight in different deployment scenarios.

These requirements are only concerned with the type and quality of the ECN feedback signal. The requirements do not stipulate how a TCP sender might react to the improved ECN signal. The requirements also do not imply that any modifications to TCP senders or receivers are obligatory.

### 1.1. Terminology

We use the following terminology from [RFC3168] and [RFC3540]:

The ECN field in the IP header:

- Not-ECT: the not ECN-Capable Transport codepoint,
- CE: the Congestion Experienced codepoint,
- ECT(0): the first ECN-Capable Transport codepoint, and
- ECT(1): the second ECN-Capable Transport codepoint.

The ECN flags in the TCP header:

- CWR: the Congestion Window Reduced flag,
- ECE: the ECN-Echo flag, and
- NS: ECN Nonce Sum.

In this document, the ECN feedback scheme as specified in [RFC3168] is called 'classic ECN' and any new proposal is called a 'more accurate ECN feedback' scheme. A 'congestion mark' is defined as an IP packet where the CE codepoint is set. A 'congestion episode' refers to one or more congestion marks that belong to the same overload situation in the network (usually during one RTT). A TCP segment with the acknowledgement flag set is simply called an ACK.

## 2. Recap of Classic ECN and ECN Nonce in IP/TCP

ECN requires two bits in the IP header. The ECN capability of a packet is indicated when either one of the two bits is set. A network node can set both bits simultaneously when it experiences congestion. This leads to the four codepoints (Not-ECT, ECT(0), ECT(1), and CE) as listed above.

In the TCP header, the first two bits in byte 14 are defined as ECN feedback for each half-connection. A TCP receiver signals the reception of a congestion mark using the ECN-Echo (ECE) flag in the TCP header. For reliability, the receiver continues to set the ECE flag on every ACK. To enable the TCP receiver to determine when to stop setting the ECE flag, the sender sets the CWR flag upon reception of an ECE feedback signal. This always leads to a full RTT of ACKs with ECE set. Thus, the receiver cannot signal back any additional CE markings arriving within the same RTT.

The ECN Nonce [RFC3540] is an experimental addition to ECN that the TCP sender can use to protect itself against accidental or malicious concealment of CE-marked or dropped packets. This addition defines the last bit of byte 13 in the TCP header as the Nonce Sum (NS) flag. The receiver maintains a nonce sum that counts the occurrence of ECT(1) packets and signals the least significant bit of this sum on the NS flag. There are no known deployments of a TCP stack that makes use of the ECN Nonce extension.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Header Length				Reserved			N S	C W R	E C E	U R G E	A C K	P S H	R S T	S Y N	F I N

Figure 1: The (Post-ECN Nonce) Definition of the TCP Header Flags

An alternative for a sender to assure feedback integrity has been proposed where the sender itself occasionally inserts a CE mark or reorders packets, and checks that the receiver feeds these back faithfully [TEST-RCV]. This alternative consumes no header bits or codepoints, and it releases the ECT(1) codepoint in the IP header and the NS flag in the TCP header for other uses.



### 3. Use Cases

The following two examples serve to show where existing mechanisms would already benefit from more accurate ECN feedback information. However, as it is hard to predict the future, once a more accurate ECN feedback mechanism that adheres to the requirements stated in this document is widely deployed, it's very likely that additional uses will be found. The examples listed below are in no particular order.

ConEx is an experimental approach that allows a sender to relay congestion feedback provided by the receiver into the network along the forward data path. ConEx information can be used for traffic management to limit traffic proportionate to the actual congestion being caused, rather than limiting traffic based on rate or volume [RFC6789]. A ConEx sender uses selective acknowledgements (SACK) [RFC2018] for accurate feedback of loss signals, but until now TCP has offered no equivalent accurate feedback for ECN.

DCTCP offers very low and predictable queuing delay. DCTCP changes the reaction to congestion of a TCP sender and additionally requires switches/routers to have ECN enabled and configured with a low step threshold and no signal smoothing, so it is currently only used in private networks, e.g., internal to data centers. DCTCP was released in Microsoft Windows 8, and implementations exist for Linux and FreeBSD. To retrieve sufficient congestion information, the different DCTCP implementations use a proprietary ECN feedback protocol, but they omit capability negotiation. Moreover, the feedback protocol proposed in [DCTCP] only works if there are no losses at all, and otherwise it gets very confused (see Appendix A). Therefore, if a generic, more accurate ECN feedback scheme were available, it would solve two problems for DCTCP: i) the need for a consistent variant of DCTCP to be deployed network-wide and ii) the inability to cope with ACK loss.

Classic ECN-TCP would not benefit from more accurate ECN feedback, but it would not suffer either. The same signal that is currently conveyed with ECN following the specification given in [RFC3168] would be available.

The following scenarios should briefly show where accurate ECN feedback is needed or adds value:

A sender with standardized TCP congestion control that supports ConEx:

In this case, the ConEx mechanism uses the extra information per RTT to re-echo the precise congestion information, but the congestion control algorithm still ignores multiple marks per RTT [RFC5681].

A sender using DCTCP congestion control without ConEx:

The congestion control algorithm uses the extra info per RTT to perform its decrease depending on the number of congestion marks.

A sender using DCTCP congestion control and supporting ConEx:

Both the congestion control algorithm and ConEx use the more accurate ECN feedback mechanism.

As-yet-unspecified sender mechanisms:

The above are two examples of more general interest in sender mechanisms that respond to the extent of congestion feedback, not just its existence. It will greatly simplify incremental deployment if the sender can unilaterally deploy new behaviours and rely on the presence of generic receivers that have already implemented more accurate feedback.

A TCP sender using congestion control as specified in RFC 5681 without ConEx:

No accurate feedback is necessary here. The congestion control algorithm still reacts to only one signal per RTT. But, it is best to feed back all the information the receiver gets, whether or not the sender uses it -- at least as long as overhead is low or zero.

Using CE for checking integrity:

If a more accurate ECN feedback scheme feeds all occurrences of CE marks back, a sender could perform integrity checking by occasionally injecting CE marks itself. Specifically, a sender can send packets that it randomly marks with CE (at low frequency), then check if feedback is received for these packets. The congestion notification feedback for these self-injected markings would not require a congestion control reaction [TEST-RCV].

RFC 7560                      Requirements for More Accurate ECN Feedback                      August 2015

#### 4. Requirements

The requirements of the accurate ECN feedback protocol are to have fairly accurate (not necessarily perfect), timely, and protected signalling. This leads to the following requirements, which should be discussed for any proposed more accurate ECN feedback scheme:

##### Resilience

The ECN feedback signal is carried within the ACK. Pure TCP ACKs can get lost without recovery (not just due to congestion but also due to deliberate ACK thinning). Moreover, delayed ACKs are commonly used with TCP. Typically, an ACK is triggered after two data segments (or more, e.g., due to receive segment coalescing, ACK compression, ACK congestion control [RFC5690], or other phenomena; see [RFC3449]). In a high-congestion situation where most of the packets are marked with CE, an accurate feedback mechanism should still be able to signal sufficient congestion information. Thus, the accurate ECN feedback extension has to take delayed ACKs and ACK loss into account. Also, a more accurate feedback protocol should still provide more accurate feedback than classic ECN when delayed ACKs cover more than two segments, or when a thin stream disables Nagle's algorithm [RFC896]. Finally, the feedback mechanism should not be impacted by reordering of ACKs, even when the ACKed sequence number does not increase.

##### Timeliness

A CE mark can be induced by the sending host, or more commonly a network node on the transmission path, and is then echoed by the receiver in the TCP ACK. Thus, when this information arrives at the sender, it is naturally already about one RTT old. With a sufficient ACK rate, a further delay of a small number of packets can be tolerated. However, this information will become stale with large delays, given the dynamic nature of networks. TCP congestion control (which itself partly introduces these dynamics) operates on a time scale of one RTT. Thus, to be timely, congestion feedback information should be delivered within about one RTT.

##### Integrity

The integrity of the feedback in a more accurate ECN feedback scheme should be assured, at least as well as the ECN Nonce. Alternatively, it should at least be possible to give strong incentives for the receiver and network nodes to cooperate honestly.

Given there are known problems with ECN Nonce deployment, this document only requires that the integrity of the more accurate ECN feedback can be assured; it does not require that the ECN Nonce mechanism is employed to achieve this. Indeed, if integrity could be provided in another manner, a more accurate ECN feedback protocol might repurpose the nonce sum (NS) flag in the TCP header.

If the more accurate ECN feedback scheme provides sufficient information, the integrity check could be performed by, e.g., deterministically setting the CE in the sender and monitoring the respective feedback (similar to ECT(1) and the ECN Nonce sum). Whether a sender should enforce when it detects wrong feedback information, and what kind of enforcement it should apply, are policy issues that need not be specified as part of the more accurate ECN feedback signal scheme itself, but rather when specifying an update to core TCP mechanisms like congestion control that make use of the more accurate ECN signal.

#### Accuracy

Classic ECN feeds back one congestion notification per RTT; this is sufficient for classic TCP congestion control, which reduces the sending rate at most once per RTT. Thus, the more accurate ECN feedback scheme should ensure that, if a congestion episode occurs, at least one congestion notification is echoed and received per RTT as classic ECN would do. Of course, the goal of a more accurate ECN extension is to reconstruct the number of CE markings more accurately. In the best case, the new scheme should even allow reconstruction of the exact number of payload bytes that a CE-marked packet was carrying. However, it is accepted that it may be too complex for a sender to get the exact number of congestion markings or marked bytes in all situations. Ideally, the feedback scheme should preserve the order in which any (of the four) ECN signals were received. And, ideally, it would even be possible for the sender to determine which of the packets covered by one delayed ACK were congestion marked, e.g., if the flow consists of packets of different sizes, or to allow for future protocols where the order of the markings may be important.

In the best case, a sender that sees more accurate ECN feedback information would be able to reconstruct the occurrence of any of the four codepoints (Not-ECT, CE, ECT(0), ECT(1)). However, assuming the sender marks all data packets as ECN-capable and uses a default setting of ECT(0) (as with [RFC3168]), solely feeding back the occurrence of CE and ECT(1) might be sufficient. Because the sender can keep account of the transmitted segments with any of the three ECN codepoints, conveying any two of these back to the sender is sufficient for it to reconstruct the third as

observed by the receiver. Thus, a more accurate ECN feedback scheme should at least provide information on two of these signals, e.g., CE and ECT(1).

If a more accurate ECN scheme can reliably deliver feedback in most but not all circumstances, ideally the scheme should at least not introduce bias. In other words, undetected loss of some ACKs should be as likely to increase as decrease the sender's estimate of the probability of ECN marking.

#### Complexity

Implementation should be as simple as possible, and only a minimum of additional state information should be needed. This will enable more accurate ECN feedback to be used as the default feedback mechanism, even if only one ECN feedback signal per RTT is needed.

#### Overhead

A more accurate ECN feedback signal should limit the additional network load, because ECN feedback is ultimately not critical information (in the worst case, loss will still be available as a congestion signal of last resort). As feedback information has to be provided frequently and in a timely fashion, potentially all or a large fraction of TCP acknowledgements might carry this information. Ideally, no additional segments should be exchanged compared to a TCP session as specified in RFC 3168, and the overhead in each segment should be minimized.

#### Backward and forward compatibility

Given more accurate ECN feedback will involve a change to the TCP protocol, it should be negotiated between the two TCP endpoints. If either end does not support the more accurate feedback, they should both be able to fall back to classic ECN feedback.

A more accurate ECN feedback extension should aim to traverse most middleboxes, including firewalls and Network Address Translators (NATs). Further, a feedback mechanism should provide a method to fall back to classic ECN signalling if the new signal is suppressed by certain middleboxes.

In order to avoid a fork in the TCP protocol specifications, if experiments with the new ECN feedback protocol are successful, the intention is to eventually update RFC 3168 for any TCP/ECN sender, not just for ConEx or DCTCP senders. Then, future senders will be able to unilaterally deploy new behaviours that exploit the existence of more accurate ECN feedback in receivers (forward

compatibility). Conversely, even if another sender only needs one ECN feedback signal per RTT, it should be able to use more accurate ECN feedback and simply ignore the excess information.

Furthermore, the receiver should not make assumptions about the mechanism that was used to set the markings nor about any interpretation or reaction to the congestion signal. The receiver only needs to faithfully reflect congestion information back to the sender.

## 5. Design Approaches

This section introduces some possible design approaches for TCP ECN feedback. The purpose of this section is to give examples of how trade-offs might be needed between the requirements, as input to future IETF work to specify a protocol. The order is not significant, and there is no intention to endorse any particular approach.

All approaches presented below (and proposed so far) are able to provide accurate ECN feedback information as long as no ACK loss occurs and the congestion rate is reasonable. In the case of a high ACK loss rate or very high congestion (CE-marking) rate, the proposed schemes have different resilience characteristics depending on the number of bits used for the encoding. While classic ECN provides reliable (but inaccurate) feedback of a maximum of one congestion signal per RTT, the proposed schemes do not implement an explicit acknowledgement mechanism for the feedback (as, e.g., the ECE/CWR exchange of [RFC3168]).

### 5.1. Redefinition of ECN/NS Header Bits

Schemes in this category can additionally use the NS bit for capability negotiation during the TCP handshake exchange. Thus a more accurate ECN could be negotiated without changing the classic ECN negotiation and thus being backwards compatible.

Schemes in this category can simply redefine the ECN header flags, ECE and CWR, to encode the occurrence of a CE marking at the receiver. This approach provides very limited resilience against loss of ACK, particularly pure ACKs (no payload and therefore delivered unreliably).

A couple of schemes have been proposed so far:

- o A naive 1-bit scheme that sends one ECE for each CE received could use CWR to increase robustness against ACK loss by introducing redundant information on the next ACK, but this is still vulnerable to ACK loss.
- o The scheme defined for DCTCP [DCTCP], which toggles the ECE feedback on an immediate ACK whenever the CE marking changes, and otherwise feeds back delayed ACKs with the ECE value unchanged. Appendix A demonstrates that this scheme is still ambiguous to the sender if the ACKs are pure ACKs, and if some may have been lost.

Alternatively, the receiver uses the three ECN/NS header flags, ECE, CWR, and NS, to represent a counter that signals the accumulated number of CE markings it has received. Resilience against loss is better than the flag-based schemes but may not suffice in the presence of extended ACK loss that otherwise would not affect the TCP sender's performance.

A number of coding schemes have been proposed so far in this category:

- o A 3-bit counter scheme continuously feeds back the three least significant bits of a CE counter;
- o A scheme that defines a standardized lookup table to map the eight codepoints onto either a CE counter or an ECT(1) counter.

These proposed schemes provide accumulated information on CE marking feedback, similar to the number of acknowledged bytes in the TCP header. Due to the limited number of bits, the ECN feedback information will wrap much more often than the acknowledgement field. Thus, feedback information could be lost due to a relatively small sequence of pure-ACK losses. Resilience could be increased by introducing redundancy, e.g., send each counter increase two or more times. Of course, any of these additional mechanisms will increase the complexity. If the congestion rate is greater than the ACK rate (multiplied by the number of congestion marks that can be signaled per ACK), the congestion information cannot correctly be fed back. Covering the worst case (where every packet is CE marked) can potentially be realized by dynamically adapting the ACK rate and redundancy. This again increases complexity and perhaps the signalling overhead as well. Schemes that do not repurpose the ECN NS bit could still support the ECN Nonce.

## 5.2. Using Other Header Bits

As seen in Figure 1, there are currently three unused flags in the TCP header. The proposed 3-bit counter or codepoint schemes could be extended by one or more bits to add higher resilience against ACK loss. The relative gain would be exponentially higher resilience against ACK loss, while the respective drawbacks would remain identical.

Alternatively, a new method could standardize the use of the bits in the Urgent Pointer field (see [RFC6093]) to signal more bits of its congestion signal counter, but only whenever the Urgent Flag is not set. As this is often the case, resilience could be increased without additional header overhead.

Any proposal to use such bits would need to check the likelihood that some middleboxes might discard or 'normalize' the currently unused flag bits or a non-zero Urgent Pointer when the Urgent Flag is cleared. If during experimentation certain bits have been proven to be usable, the assignment of any of these bits would then require an IETF standards action.

## 5.3. Using a TCP Option

Alternatively, a new TCP option could be introduced, to help maintain the accuracy and integrity of ECN feedback between receiver and sender. Such an option could provide higher resilience and even more information, e.g., as much as is provided by a proposal for SCTP that counts the number of CE marked packet [SCTP-ECN] since the last CWR was observed, or by ECN for RTP/UDP [RFC6679]. The latter explicitly provides the total number of packets during a connection where the IP ECN field is set to ECT(0), ECT(1), CE, or Not-ECT, as well as the number of lost packets. However, deploying new TCP options has its own challenges. Moreover, to actually achieve high resilience, this option would need to be carried by most or all ACKs as the receiver cannot know if and when ACKs may be dropped. Thus, this approach would introduce considerable signalling overhead even though ECN feedback is not extremely critical information (in the worst case, loss will still be available to provide a strong congestion feedback signal). Nevertheless, such a TCP option could be used in addition to a more accurate ECN feedback scheme in the TCP header or in addition to classic ECN, only when needed and when space is available.



## 6. Security Considerations

ECN feedback information must only be used if the other information contained in a received TCP segment indicates that the congestion was genuinely part of the flow and not spoofed. That is, the normal TCP acceptance techniques have to be used to verify that the segment is part of the flow before returning any contained ECN information, and, similarly, ECN feedback is only accepted on valid ACKs.

Given ECN feedback is used as input for congestion control, the respective algorithm would not react appropriately if ECN feedback were lost and the resilience mechanism to recover it was inadequate. This resilience requirement is articulated in Section 4. However, it should be noted that ECN feedback is not the last resort against congestion collapse, because if there is insufficient response to ECN, loss will ensue, and TCP will still react appropriately to loss.

A receiver could suppress ECN feedback information leading to its connections consuming excess sender or network resources. This problem is similar to that seen with the classic ECN feedback scheme and should be addressed by integrity checking as required in Section 4.

## 7. References

### 7.1. Normative References

- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<http://www.rfc-editor.org/info/rfc3168>>.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <<http://www.rfc-editor.org/info/rfc3540>>.

### 7.2. Informative References

- [DCTCP] Bensley, S., Eggert, L., and D. Thaler, "Microsoft's Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters", Work in Progress, draft-bensley-tcpm-dctcp-05, July 2015.
- [ECN-BENEFITS] Fairhurst, G. and M. Welzl, "The Benefits of using Explicit Congestion Notification (ECN)", Work in Progress draft-ietf-aqm-ecn-benefits-06, July 2015.

RFC 7560                      Requirements for More Accurate ECN Feedback                      August 2015

- [RFC896] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984, <<http://www.rfc-editor.org/info/rfc896>>.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <<http://www.rfc-editor.org/info/rfc2018>>.
- [RFC3449] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, DOI 10.17487/RFC3449, December 2002, <<http://www.rfc-editor.org/info/rfc3449>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<http://www.rfc-editor.org/info/rfc5681>>.
- [RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, DOI 10.17487/RFC5690, February 2010, <<http://www.rfc-editor.org/info/rfc5690>>.
- [RFC6093] Gont, F. and A. Yourtchenko, "On the Implementation of the TCP Urgent Mechanism", RFC 6093, DOI 10.17487/RFC6093, January 2011, <<http://www.rfc-editor.org/info/rfc6093>>.
- [RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August 2012, <<http://www.rfc-editor.org/info/rfc6679>>.
- [RFC6789] Briscoe, B., Ed., Woundy, R., Ed., and A. Cooper, Ed., "Congestion Exposure (ConEx) Concepts and Use Cases", RFC 6789, DOI 10.17487/RFC6789, December 2012, <<http://www.rfc-editor.org/info/rfc6789>>.
- [SCTP-ECN] Stewart, R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", Work in Progress, draft-stewart-tsvwg-sctpecn-05, January 2014.
- [TEST-RCV] Moncaster, T., Briscoe, B., and A. Jacquet, "A TCP Test to Allow Senders to Identify Receiver Non-Compliance", Work in Progress, draft-moncaster-tcpm-rcv-cheat-03, July 2014.

## Appendix A. Ambiguity of the More Accurate ECN Feedback in DCTCP

As defined in [DCTCP], a DCTCP receiver feeds back ECE=0 on delayed ACKs as long as CE remains 0, and also immediately sends an ACK with ECE=0 when CE transitions to 1. Similarly, it continually feeds back ECE=1 on delayed ACKs while CE remains 1 and immediately feeds back ECE=1 when CE transitions to 0. A sender can unambiguously decode this scheme if there is never any ACK loss, and the sender assumes there will never be any ACK loss.

The following two examples show that the feedback sequence becomes highly ambiguous to the sender if either of these conditions is broken. Below, '0' represents ECE=0, '1' represents ECE=1, and '.' represents a gap of one segment between delayed ACKs. Now imagine that the sender receives the following sequence of feedback on three pure ACKs:

0.0.0

When the receiver sent this sequence, it could have been any of the following four sequences:

- a. 0.0.0 (0 x CE)
- b. 010.0 (1 x CE)
- c. 0.010 (1 x CE)
- d. 01010 (2 x CE)

where any of the 1s represent a possible pure ACK carrying ECE feedback that could have been lost. If the sender guesses (a), it might be correct, or it might miss 1 or 2 congestion marks over 5 packets. Therefore, when confronted with this simple sequence (that is not contrived), a sender can guess that congestion might have been 0%, 20%, or 40%, but it doesn't know which.

Sequences with a longer gap (e.g., 0...0.0) become far more ambiguous. It helps a little if the sender knows the distance the receiver uses between delayed ACKs, and it helps a lot if the distance is 1, i.e., no delayed ACKs. However, even without delayed ACKs there will still be ambiguity whenever there are pure ACK losses.

RFC 7560                      Requirements for More Accurate ECN Feedback                      August 2015

#### Acknowledgements

Thanks to Gorry Fairhurst for his review and for ideas on CE-based integrity checking and to Mohammad Alizadeh for suggesting the need to avoid bias.

Bob Briscoe was partly funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700) and through the Trilogy 2 project (ICT-317756). The views expressed here are solely those of the authors, in the context of the mentioned funding projects.

#### Authors' Addresses

Mirja Kuehlewind (editor)  
ETH Zurich  
Gloriastrasse 35  
Zurich 8092  
Switzerland

Email: [mirja.kuehlewind@tik.ee.ethz.ch](mailto:mirja.kuehlewind@tik.ee.ethz.ch)

Richard Scheffenegger  
NetApp, Inc.  
Am Euro Platz 2  
Vienna 1120  
Austria

Phone: +43 1 3676811 3146  
Email: [rs@netapp.com](mailto:rs@netapp.com)

Bob Briscoe  
BT  
B54/77, Adastral Park  
Martlesham Heath  
Ipswich IP5 3RE  
United Kingdom

Phone: +44 1473 645196  
Email: [ietf@bobbriscoe.net](mailto:ietf@bobbriscoe.net)  
URI: <http://bobbriscoe.net/>

TCP Maintenance & Minor Extensions (tcpm)  
Internet-Draft  
Intended status: Experimental  
Expires: March 9, 2016

B. Briscoe  
Simula Research Laboratory  
M. Kuehlewind  
ETH Zurich  
R. Scheffenegger  
NetApp, Inc.  
September 6, 2015

More Accurate ECN Feedback in TCP  
draft-kuehlewind-tcpm-accurate-ecn-04

## Abstract

Explicit Congestion Notification (ECN) is a mechanism where network nodes can mark IP packets instead of dropping them to indicate incipient congestion to the end-points. Receivers with an ECN-capable transport protocol feed back this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recently, new TCP mechanisms like Congestion Exposure (ConEx) or Data Center TCP (DCTCP) need more accurate ECN feedback information whenever more than one marking is received in one RTT. This document specifies an experimental scheme to provide more than one feedback signal per RTT in the TCP header. Given TCP header space is scarce, it overloads the three existing ECN-related flags in the TCP header and provides additional information in a new TCP option.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 9, 2016.

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

## Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Document Roadmap . . . . .	4
1.2. Goals . . . . .	4
1.3. Experiment Goals . . . . .	5
1.4. Terminology . . . . .	5
1.5. Recap of Existing ECN feedback in IP/TCP . . . . .	6
2. AccECN Protocol Overview and Rationale . . . . .	7
2.1. Capability Negotiation . . . . .	8
2.2. Feedback Mechanism . . . . .	8
2.3. Delayed ACKs and Resilience Against ACK Loss . . . . .	9
2.4. Feedback Metrics . . . . .	9
2.5. Generic (Dumb) Reflector . . . . .	10
3. AccECN Protocol Specification . . . . .	11
3.1. Negotiation during the TCP handshake . . . . .	11
3.2. AccECN Feedback . . . . .	13
3.2.1. The ACE Field . . . . .	14
3.2.2. Safety against Ambiguity of the ACE Field . . . . .	15
3.2.3. The AccECN Option . . . . .	16
3.2.4. Path Traversal of the AccECN Option . . . . .	17
3.2.5. Usage of the AccECN TCP Option . . . . .	18
3.3. AccECN Compliance by TCP Proxies, Offload Engines and other Middleboxes . . . . .	19
4. Interaction with Other TCP Variants . . . . .	20
4.1. Compatibility with SYN Cookies . . . . .	20
4.2. Compatibility with Other TCP Options and Experiments . . . . .	20
4.3. Compatibility with Feedback Integrity Mechanisms . . . . .	21
5. Protocol Properties . . . . .	22
6. IANA Considerations . . . . .	24
7. Security Considerations . . . . .	24
8. Acknowledgements . . . . .	25
9. Comments Solicited . . . . .	25

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

10. References . . . . .	26
10.1. Normative References . . . . .	26
10.2. Informative References . . . . .	26
Appendix A. Example Algorithms . . . . .	28
A.1. Example Algorithm to Encode/Decode the AccECN Option . . . . .	28
A.2. Example Algorithm for Safety Against Long Sequences of ACK Loss . . . . .	29
A.2.1. Safety Algorithm without the AccECN Option . . . . .	29
A.2.2. Safety Algorithm with the AccECN Option . . . . .	31
A.3. Example Algorithm to Estimate Marked Bytes from Marked Packets . . . . .	32
A.4. Example Algorithm to Beacon AccECN Options . . . . .	33
A.5. Example Algorithm to Count Not-ECT Bytes . . . . .	33
Appendix B. Alternative Design Choices (To Be Removed Before Publication) . . . . .	34
Appendix C. Open Protocol Design Issues (To Be Removed Before Publication) . . . . .	35
Appendix D. Changes in This Version (To Be Removed Before Publication) . . . . .	35
Authors' Addresses . . . . .	36

## 1. Introduction

Explicit Congestion Notification (ECN) [RFC3168] is a mechanism where network nodes can mark IP packets instead of dropping them to indicate incipient congestion to the end-points. Receivers with an ECN-capable transport protocol feed back this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recently, proposed mechanisms like Congestion Exposure (ConEx [I-D.ietf-conex-abstract-mech]) or DCTCP [I-D.bensley-tcpm-dctcp] need more accurate ECN feedback information whenever more than one marking is received in one RTT. A fuller treatment of the motivation for this specification is given in the associated requirements document [RFC7560].

This documents specifies an experimental scheme for ECN feedback in the TCP header to provide more than one feedback signal per RTT. It will be called the more accurate ECN feedback scheme, or AccECN for short. If AccECN progresses from experimental to the standards track, it is intended to be a complete replacement for classic ECN feedback, not a fork in the design of TCP. Thus, the applicability of AccECN is intended to include all public and private IP networks (and even any non-IP networks over which TCP is used today). Until the AccECN experiment succeeds, [RFC3168] will remain as the standards track specification for adding ECN to TCP. To avoid confusion, in this document we use the term 'classic ECN' for the pre-existing ECN specification [RFC3168].

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

AccECN is solely an (experimental) change to the TCP wire protocol. It is completely independent of how TCP might respond to congestion feedback. This specification overloads flags and fields in the main TCP header with new definitions, so both ends have to support the new wire protocol before it can be used. Therefore during the TCP handshake the two ends use the three ECN-related flags in the TCP header to negotiate the most advanced feedback protocol that they can both support.

It is likely (but not required) that the AccECN protocol will be implemented along with the following experimental additions to the TCP-ECN protocol: ECN-capable SYN/ACK [RFC5562], ECN path-probing and fall-back [I-D.kuehlewind-tcpm-ecn-fallback] and testing receiver non-compliance [I-D.moncaster-tcpm-rcv-cheat].

### 1.1. Document Roadmap

The following introductory sections outline the goals of AccECN (Section 1.2) and the goal of experiments with ECN (Section 1.3) so that it is clear what success would look like. Then terminology is defined (Section 1.4) and a recap of existing prerequisite technology is given (Section 1.5).

Section 2 gives an informative overview of the AccECN protocol. Then Section 3 gives the normative protocol specification. Section 4 assesses the interaction of AccECN with commonly used variants of TCP, whether standardised or not. Section 5 summarises the features and properties of AccECN.

Section 6 summarises the protocol fields and numbers that IANA will need to assign and Section 7 points to the aspects of the protocol that will be of interest to the security community.

Appendix A gives pseudocode examples for the various algorithms that AccECN uses.

### 1.2. Goals

[RFC7560] enumerates requirements that a candidate feedback scheme will need to satisfy, under the headings: resilience, timeliness, integrity, accuracy (including ordering and lack of bias), complexity, overhead and compatibility (both backward and forward). It recognises that a perfect scheme that fully satisfies all the requirements is unlikely and trade-offs between requirements are likely. Section 5 presents the properties of AccECN against these requirements and discusses the trade-offs made.



Internet-Draft

Accurate TCP-ECN Feedback

September 2015

The requirements document recognises that a protocol as ubiquitous as TCP needs to be able to serve as-yet-unspecified requirements. Therefore an AccECN receiver aims to act as a generic (dumb) reflector of congestion information so that in future new sender behaviours can be deployed unilaterally.

### 1.3. Experiment Goals

TCP is critical to the robust functioning of the Internet, therefore any proposed modifications to TCP need to be thoroughly tested. The present specification describes an experimental protocol that adds more accurate ECN feedback to the TCP protocol. The intention is to specify the protocol sufficiently so that more than one implementation can be built in order to test its function, robustness and interoperability (with itself and with previous version of ECN and TCP).

The experimental protocol will be considered successful if it satisfies the requirements of [RFC7560] in the consensus opinion of the IETF tcpm working group. In short, this requires that it improves the accuracy and timeliness of TCP's ECN feedback, as claimed in Section 5, while striking a balance between the conflicting requirements of resilience, integrity and minimisation of overhead. It also requires that it is not unduly complex, and that it is compatible with prevalent equipment behaviours in the current Internet, whether or not they comply with standards.

### 1.4. Terminology

AccECN: The more accurate ECN feedback scheme will be called AccECN for short.

Classic ECN: the ECN scheme as specified in [RFC3168].

ACK: A TCP acknowledgement, with or without a data payload.

Pure ACK: A TCP acknowledgement without a data payload.

TCP client: The TCP stack that originates a connection.

TCP server: The TCP stack that responds to a connection request.

Data Receiver: The endpoint of a TCP half-connection that receives data and sends AccECN feedback.

Data Sender: The endpoint of a TCP half-connection that sends data and receives AccECN feedback.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

### 1.5. Recap of Existing ECN feedback in IP/TCP

ECN [RFC3168] uses two bits in the IP header. Once ECN has been negotiated with the receiver at the transport layer, an ECN sender can set two possible codepoints (ECT(0) or ECT(1)) in the IP header to indicate an ECN-capable transport (ECT). If both ECN bits are zero, the packet is considered to have been sent by a Not-ECN-capable Transport (Not-ECT). When a network node experiences congestion, it will occasionally either drop or mark a packet, with the choice depending on the packet's ECN codepoint. If the codepoint is Not-ECT, only drop is appropriate. If the codepoint is ECT(0) or ECT(1), the node can mark the packet by setting both ECN bits, which is termed 'Congestion Experienced' (CE), or loosely a 'congestion mark'. Table 1 summarises these codepoints.

IP-ECN codepoint (binary)	Codepoint name	Description
00	Not-ECT	Not ECN-Capable Transport
01	ECT(1)	ECN-Capable Transport (1)
10	ECT(0)	ECN-Capable Transport (0)
11	CE	Congestion Experienced

Table 1: The ECN Field in the IP Header

In the TCP header the first two bits in byte 14 are defined as flags for the use of ECN (CWR and ECE in Figure 1 [RFC3168]). An TCP client indicates it supports ECN by setting ECE=CWR=1 in the SYN, and an ECN-enabled server confirms ECN support by setting ECE=1 and CWR=0 in the SYN/ACK. On reception of a CE-marked packet at the IP layer, the Data Receiver starts to set the Echo Congestion Experienced (ECE) flag continuously in the TCP header of ACKs, which ensures the signal is received reliably even if ACKs are lost. The TCP sender confirms that it has received at least one ECE signal by responding with the congestion window reduced (CWR) flag, which allows the TCP receiver to stop repeating the ECN-Echo flag. This always leads to a full RTT of ACKs with ECE set. Thus any additional CE markings arriving within this RTT cannot be fed back.

The ECN Nonce [RFC3540] is an optional experimental addition to ECN that the TCP sender can use to protect against accidental or malicious concealment of marked or dropped packets. The sender can

send an ECN nonce, which is a continuous pseudo-random pattern of ECT(0) and ECT(1) codepoints in the ECN field. The receiver is required to feed back a 1-bit nonce sum that counts the occurrence of ECT(1) packets using the last bit of byte 13 in the TCP header, which is defined as the Nonce Sum (NS) flag.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Header Length				Reserved			N S	C W R	E C E	U R G	A C K	P S H	R S T	S Y N	F I N

Figure 1: The (post-ECN Nonce) definition of the TCP header flags

## 2. AccECN Protocol Overview and Rationale

This section provides an informative overview of the AccECN protocol that will be normatively specified in Section 3

Like the original TCP approach, the Data Receiver of each TCP half-connection sends AccECN feedback to the Data Sender on TCP acknowledgements, reusing data packets of the other half-connection whenever possible.

The AccECN protocol has had to be designed in two parts:

- o an essential part that re-uses ECN TCP header bits to feed back the number of arriving CE marked packets. This provides more accuracy than classic ECN, but limited resilience against ACK loss;
- o a supplementary part using a new AccECN TCP Option that provides additional feedback on the number of bytes that arrive marked with each of the three ECN codepoints (not just CE marks). This provides greater resilience against ACK loss than the essential feedback, but it is more likely to suffer from middlebox interference.

The two part design was necessary, given limitations on the space available for TCP options and given the possibility that certain incorrectly designed middleboxes prevent TCP using any new options.

The essential part overloads the previous definition of the three flags in the TCP header that had been assigned for use by ECN. This design choice deliberately replaces the classic ECN feedback protocol, rather than leaving classic ECN intact and adding more accurate feedback separately because:

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

- o this efficiently reuses scarce TCP header space, given TCP option space is approaching saturation;
- o a single upgrade path for the TCP protocol is preferable to a fork in the design;
- o otherwise classic and accurate ECN feedback could give conflicting feedback on the same segment, which could open up new security concerns and make implementations unnecessarily complex;
- o middleboxes are more likely to faithfully forward the TCP ECN flags than newly defined areas of the TCP header.

AcceCN is designed to work even if the supplementary part is removed or zeroed out, as long as the essential part gets through.

## 2.1. Capability Negotiation

AcceCN is a change to the wire protocol of the main TCP header, therefore it can only be used if both endpoints have been upgraded to understand it. The TCP client signals support for AcceCN on the initial SYN of a connection and the TCP server signals whether it supports AcceCN on the SYN/ACK. The TCP flags on the SYN that the client uses to signal AcceCN support have been carefully chosen so that a TCP server will interpret them as a request to support the most recent variant of ECN that it supports. Then the client falls back to the same ECN variant.

An AcceCN TCP client does not send the new AcceCN Option on the SYN as SYN option space is limited and successful negotiation using the flags in the main header is taken as sufficient evidence that both ends also support the AcceCN Option. The TCP server sends the AcceCN Option on the SYN/ACK and the client sends it on the first ACK to test whether the network path forwards the option correctly.

## 2.2. Feedback Mechanism

A Data Receiver maintains four counters initialised at the start of the half-connection. Three count the number of arriving payload bytes marked CE, ECT(1) and ECT(0) respectively. The fourth counts the number of packets arriving marked with a CE codepoint (including control packets without payload if they are CE-marked).

The Data Sender maintains four equivalent counters for the half connection, and the AcceCN protocol is designed to ensure they will match the values in the Data Receiver's counters, albeit after a little delay.

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

Each ACK carries the three least significant bits (LSBs) of the packet-based CE counter using the ECN bits in the TCP header, now renamed the Accurate ECN (ACE) field. The LSBs of each of the three byte counters are carried in the AccECN Option.

### 2.3. Delayed ACKs and Resilience Against ACK Loss

With both the ACE and the AccECN Option mechanisms, the Data Receiver continually repeats the current LSBs of each of its respective counters. Then, even if some ACKs are lost, the other end should be able to infer how much to increment its own counters, even if the protocol field has wrapped.

The 3-bit ACE field can wrap fairly frequently. Therefore, even if it appears to have incremented by one (say), the field might have actually cycled completely then incremented by one. This is a possibility because the whole sequence of ACKs carrying the intervening values of the field might all have been lost or delayed.

The fields in the AccECN Option are larger, but they will increment in larger steps because they count bytes not packets. Nonetheless, their size has been chosen such that a whole cycle of the field would never occur between ACKs unless there had been an infeasibly long sequence of ACK losses. Therefore, as long as the AccECN Option is available, it can be treated as a dependable feedback channel.

If the AccECN Option is not available, e.g. it is being stripped by a middlebox, the AccECN protocol will only feed back information on CE markings (using the ACE field). Although not ideal, this will be sufficient, because it is envisaged that ECT(0) and ECT(1) will never indicate more severe congestion than CE, even though future uses for ECT(0) or ECT(1) are still unclear. Because the 3-bit ACE field is so small, when it is the only field available the Data Sender has to interpret it conservatively assuming the worst possible wrap.

Certain specified events trigger the Data Receiver to include an AccECN Option on an ACK. The rules are designed to ensure that the order in which different markings arrive at the receiver is communicated to the sender (as long as there is no ACK loss). Implementations are encouraged to send an AccECN Option more frequently, but this is left up to the implementer.

### 2.4. Feedback Metrics

The CE packet counter in the ACE field and the CE byte counter in the AccECN Option both provide feedback on received CE-marks. The CE packet counter includes control packets that do not have payload data, while the CE byte counter solely includes marked payload bytes.

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

If both are present, the byte counter in the option will provide the more accurate information needed for modern congestion control and policing schemes, such as DCTCP or ConEx. If the option is stripped, a simple algorithm to estimate the number of marked bytes from the ACE field is given in Appendix A.3.

Feedback in bytes is recommended in order to protect against the receiver using attacks similar to 'ACK-Division' to artificially inflate the congestion window, which is why [RFC5681] now recommends that TCP counts acknowledged bytes not packets.

## 2.5. Generic (Dumb) Reflector

The ACE field provides information about CE markings on both data and control packets. According to [RFC3168] the Data Sender is meant to set control packets to Not-ECT. However, mechanisms in certain private networks (e.g. data centres) set control packets to be ECN capable because they are precisely the packets that performance depends on most.

For this reason, AccECN is designed to be a generic reflector of whatever ECN markings it sees, whether or not they are compliant with a current standard. Then as standards evolve, Data Senders can upgrade unilaterally without any need for receivers to upgrade too. It is also useful to be able to rely on generic reflection behaviour when senders need to test for unexpected interference with markings (for instance [I-D.kuehlewind-tcpm-ecn-fallback] and [I-D.moncaster-tcpm-rcv-cheat]).

The initial SYN is the most critical control packet, so AccECN provides feedback on whether it is CE marked, even though it is not allowed to be ECN-capable according to RFC 3168. However, middleboxes have been known to overwrite the ECN IP field as if it is still part of the old Type of Service (ToS) field. If a TCP client has set the SYN to Not-ECT, but receives CE feedback, it can detect such middlebox interference and send Not-ECT for the rest of the connection (see [I-D.kuehlewind-tcpm-ecn-fallback] for the detailed fall-back behaviour).

Today, if a TCP server receives CE on a SYN, it cannot know whether it is invalid (or valid) because only the TCP client knows whether it originally marked the SYN as Not-ECT (or ECT). Therefore, the server's only safe course of action is to disable ECN for the connection. Instead, the AccECN protocol allows the server to feed back the CE marking to the client, which then has all the information to decide whether the connection has to fall-back from supporting ECN (or not).

Providing feedback of CE marking on the SYN also supports future scenarios in which SYNs might be ECN-enabled (without prejudging whether they ought to be). For instance, in certain environments such as data centres, it might be appropriate to allow ECN-capable SYNs. Then, if feedback showed the SYN had been CE marked, the TCP client could reduce its initial window (IW). It could also reduce IW conservatively if feedback showed the receiver did not support ECN (because if there had been a CE marking, the receiver would not have understood it). Note that this text merely motivates dumb reflection of CE on a SYN, it does not judge whether a SYN ought to be ECN-capable.

### 3. AccECN Protocol Specification

#### 3.1. Negotiation during the TCP handshake

During the TCP handshake at the start of a connection, to request more accurate ECN feedback the TCP client (host A) **MUST** set the TCP flags NS=1, CWR=1 and ECE=1 in the initial SYN segment.

If a TCP server (B) that is AccECN enabled receives a SYN with the above three flags set, it **MUST** set both its half connections into AccECN mode. Then it **MUST** set the flags CWR=1 and ECE=0 on its response in the SYN/ACK segment to confirm that it supports AccECN. The TCP server **MUST NOT** set this combination of flags unless the preceding SYN requested support for AccECN as above.

If the received SYN segment is CE-marked (see Section 2.5), an AccECN-enabled TCP server **MUST** set NS=1 on the SYN/ACK. If the received SYN is Not-ECT or ECT(0)/ECT(1), an AccECN-enabled server **MUST** clear NS (NS=0).

Once a TCP client (A) has sent the above SYN to declare that it supports AccECN, and once it has received the above SYN/ACK segment that confirms that the TCP server supports AccECN, the TCP client **MUST** set both its half connections into AccECN mode.

If after the normal TCP timeout the TCP client has not received a SYN/ACK to acknowledge its SYN, the SYN might just have been lost, e.g. due to congestion or a middlebox might be blocking segments with the AccECN flags. To expedite connection setup, the host **SHOULD** fall back to NS=CWR=ECE=0 on the retransmission of the SYN. It would make sense to also remove any other experimental fields or options on the SYN in case a middlebox might be blocking them, although the required behaviour will depend on the specification of the other option(s) and any attempt to co-ordinate fall-back between different modules of the stack. Implementers **MAY** use other fall-back strategies if they are found to be more effective (e.g. attempting to retransmit a second

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

AccECN segment before fall-back, falling back to classic ECN rather than non-ECN, and/or caching the result of a previous attempt to access the same host while negotiating AccECN).

The fall-back procedure If the TCP server receives no ACK to acknowledge a SYN/ACK that tried to negotiate AccECN is specified in Section 3.2.4.

The three flags set to 1 to indicate AccECN support on the SYN have been carefully chosen to enable natural fall-back to prior stages in the evolution of ECN. Table 2 tabulates all the negotiation possibilities for ECN-related capabilities that involve at least one AccECN-capable host. To compress the width of the table, the headings of the first four columns have been severely abbreviated, as follows:

Ac: More \*Ac\*curate ECN Feedback

N: ECN-\*N\*once [RFC3540]

E: \*E\*CN [RFC3168]

I: Not-ECN (\*I\*mplicit congestion notification using packet drop).

Ac	N	E	I	SYN A->B			SYN/ACK B->A			Mode
				NS	CWR	ECE	NS	CWR	ECE	
AB				1	1	1	0	1	0	AccECN
AB				1	1	1	1	1	0	AccECN (CE on SYN)
A	B			1	1	1	1	0	1	classic ECN
A		B		1	1	1	0	0	1	classic ECN
A			B	1	1	1	0	0	0	Not ECN
B	A			0	1	1	0	0	1	classic ECN
B		A		0	1	1	0	0	1	classic ECN
B			A	0	0	0	0	0	0	Not ECN
A			B	1	1	1	1	1	1	Not ECN (broken)
A				1	1	1	0	1	1	Not ECN (see Appx B)
A				1	1	1	1	0	0	Not ECN (see Appx B)

Table 2: ECN capability negotiation between Originator (A) and Responder (B)

Table 2 is divided into blocks each separated by an empty row.



Internet-Draft

Accurate TCP-ECN Feedback

September 2015

1. The top block shows the case already described where both endpoints support AccECN and the TCP server (B) might indicate congestion.
2. The second block shows the cases where the TCP client (A) supports AccECN but the TCP server (B) supports some earlier variant of TCP, indicated in its SYN/ACK. Therefore, as soon as an AccECN-capable TCP client (A) receives the SYN/ACK shown it MUST set both its half connections into the mode shown in the rightmost column.
3. The third block shows the cases where the TCP server (B) supports AccECN but the TCP client (A) supports some earlier variant of TCP, indicated in its SYN. Therefore, as soon as an AccECN-enabled TCP server (B) receives the SYN shown it MUST set both its half connections into the mode shown in the rightmost column.
4. The fourth block displays combinations that are not valid and therefore both ends MUST fall-back to Not ECN for both half connections. Especially the first case (marked 'broken') where all bits set in the SYN are reflected by the receiver in the SYN/ACK, which happens quite often if the TCP connection is proxied. {ToDo: Consider using the last two cases for AccECN f/b of ECT(0) and ECT(1) on the SYN (Appendix B)}

The following exceptional cases need some explanation:

ECN Nonce: An AccECN implementation, whether client or server, sender or receiver, does not need to implement the ECN Nonce behaviour [RFC3540]. AccECN is compatible with an alternative ECN feedback integrity approach that does not use up the ECT(1) codepoint and can be implemented solely at the sender (see Section 4.3).

Simultaneous Open: An originating AccECN Host (A), having sent a SYN with NS=1, CWR=1 and ECE=1, might receive another SYN from host B. Host A MUST then enter the same mode as it would have entered had it been a responding host and received the same SYN. Then host A MUST send the same SYN/ACK as it would have sent had it been a responding host (see the third block above).

### 3.2. AccECN Feedback

Each Data Receiver maintains four counters, `r.cep`, `r.ceb`, `r.e0b` and `r.elb`. The CE packet counter (`r.cep`), counts the number of packets the host receives with the CE code point in the IP ECN field, including CE marks on control packets without data. `r.ceb`, `r.e0b` and `r.elb` count the number of TCP payload bytes in packets marked

respectively with the CE, ECT(0) and ECT(1) codepoint in their IP-ECN field. When a host first enters AccECN mode, it initialises its counters to  $r.cep = 6$ ,  $r.e0b = 1$  and  $r.ceb = r.elb = 0$  (see Appendix A.5). Non-zero initial values are used to be distinct from cases where the fields are incorrectly zeroed (e.g. by middleboxes).

A host feeds back the CE packet counter using the Accurate ECN (ACE) field, as explained in the next section. And it feeds back all the byte counters using the AccECN TCP Option, as specified in Section 3.2.3. Whenever a host feeds back the value of any counter, it MUST report the most recent value, no matter whether it is in a pure ACK, an ACK with new payload data or a retransmission.

### 3.2.1. The ACE Field

After AccECN has been negotiated on the SYN and SYN/ACK, the hosts overload the three TCP flags ECE, CWR and NS in the main TCP header as one 3-bit field. Then the field is given a new name, ACE, as shown in Figure 2.

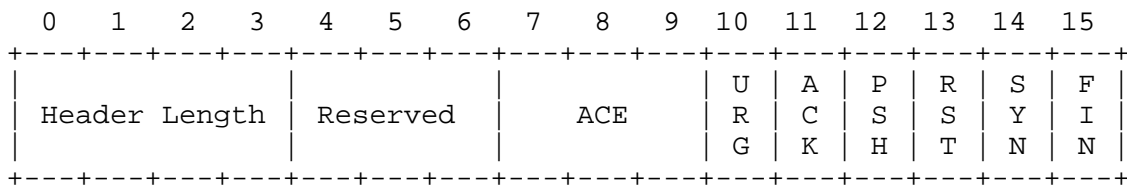


Figure 2: Definition of the ACE field within bytes 13 and 14 of the TCP Header (when AccECN has been negotiated and SYN=0).

The original definition of these three flags in the TCP header, including the addition of support for the ECN Nonce, is shown for comparison in Figure 1. This specification does not rename these three TCP flags, it merely overloads them with another name and definition once an AccECN connection has been established.

A host MUST interpret the ECE, CWR and NS flags as the 3-bit ACE counter on a segment with SYN=0 that it sends or receives if both of its half-connections are set into AccECN mode having successfully negotiated AccECN (see Section 3.1). A host MUST NOT interpret the 3 flags as a 3-bit ACE field on any segment with SYN=1 (whether ACK is 0 or 1), or if AccECN negotiation is incomplete or has not succeeded.

Both parts of each of these conditions are equally important. For instance, even if AccECN negotiation has been successful, the ACE field is not defined on any segments with SYN=1 (e.g. a retransmission of an unacknowledged SYN/ACK, or when both ends send

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

SYN/ACKs after AccECN support has been successfully negotiated during a simultaneous open).

The ACE field encodes the three least significant bits of the `r.cep` counter, therefore its initial value will be `0b110`. This non-zero initialization allows a TCP server to use a stateless handshake (see Section 4.1) but still detect from the TCP client's first ACK that the client considers it has successfully negotiated AccECN. If the SYN/ACK was CE marked, the client MUST increase its `r.cep` counter before it sends its first ACK, therefore the initial value of the ACE field will be `0b111`. These values have deliberately been chosen such that they are distinct from [RFC5562] behaviour, where the TCP client would set ECE on the first ACK as feedback for a CE mark on the SYN/ACK.

If the value of the ACE field on the first segment with SYN=0 in either direction is anything other than `0b110` or `0b111`, the Data Receiver MUST disable ECN for the remainder of the half-connection by marking all subsequent packets as Not-ECT.

### 3.2.2. Safety against Ambiguity of the ACE Field

If too many CE-marked segments are acknowledged at once, or if a long run of ACKs is lost, the 3-bit counter in the ACE field might have cycled between two ACKs arriving at the Data Sender.

Therefore an AccECN Data Receiver SHOULD immediately send an ACK once (n) CE marks have arrived since the previous ACK, where n SHOULD be 2 and MUST be no greater than 6.

If the Data Sender has not received AccECN TCP Options to give it more dependable information, and it detects that the ACE field could have cycled under the prevailing conditions, it SHOULD conservatively assume that the counter did cycle. It can detect if the counter could have cycled by using the jump in the acknowledgement number since the last ACK to calculate or estimate how many segments could have been acknowledged. An example algorithm to implement this policy is given in Appendix A.2. An implementer MAY develop an alternative algorithm as long as it satisfies these requirements.

If missing acknowledgement numbers arrive later (reordering) and prove that the counter did not cycle, the Data Sender MAY attempt to neutralise the effect of any action it took based on a conservative assumption that it later found to be incorrect.

### 3.2.3. The AccECN Option

The AccECN Option is defined as shown below in Figure 3. It consists of three 24-bit fields that provide the 24 least significant bits of the r.e0b, r.ceb and r.elb counters, respectively. The initial 'E' of each field name stands for 'Echo'.

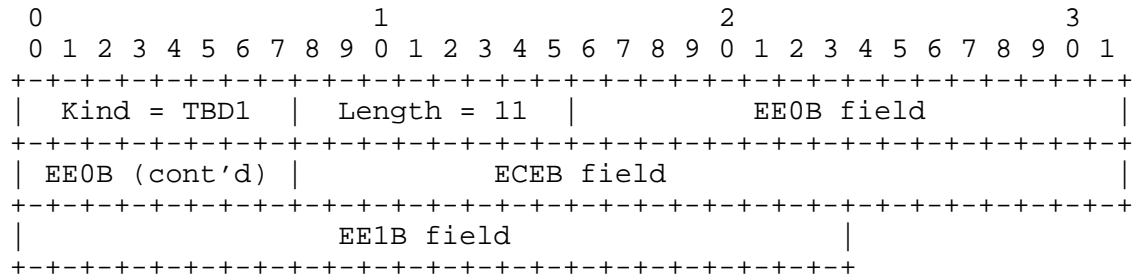


Figure 3: The AccECN Option

The Data Receiver MUST set the Kind field to TBD1, which is registered in Section 6 as a new TCP option Kind called AccECN. An experimental TCP option with Kind=254 MAY be used for initial experiments, with magic number 0xACCE.

Appendix A.1 gives an example algorithm for the Data Receiver to encode its byte counters into the AccECN Option, and for the Data Sender to decode the AccECN Option fields into its byte counters.

Note that there is no field to feedback Not-ECT bytes. Nonetheless an algorithm for the Data Sender to calculate the number of payload bytes received as Not-ECT is given in Appendix A.5.

Whenever a Data Receiver sends an AccECN Option it SHOULD always send a full-length option. To cope with option space limitations, it MAY omit unchanged fields from the tail of the option, as long as it preserves the order of the remaining fields, It MUST include any field that has changed. The length field MUST indicate which fields are present as follows:

Length=11: EE0B, ECEB, EE1B

Length=8: EE0B, ECEB

Length=5: EE0B

Length=2: (empty)

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

The empty option of Length=2 is provided to allow for a case where an AccECN Option has to be sent (e.g. on the SYN/ACK to test the path), but there is very limited space for the option. For initial experiments, the Length field MUST be 2 greater to accommodate the 16-bit magic number.

All implementations of a Data Sender MUST be able to read in AccECN Options of any of the above lengths. They MUST ignore an AccECN Option of any other length.

#### 3.2.4. Path Traversal of the AccECN Option

An AccECN host MUST NOT include the AccECN TCP Option on the SYN. Nonetheless, if the AccECN negotiation using the ECN flags in the main TCP header (Section 3.1) is successful, it implicitly declares that the endpoints also support the AccECN TCP Option.

If the TCP client indicated AccECN support, a TCP server that confirms its support for AccECN (as described in Section 3.1) MUST also include an AccECN TCP Option in the SYN/ACK. A TCP client that has successfully negotiated AccECN MUST include an AccECN Option in the first ACK at the end of the 3WSH. However, this first ACK is not delivered reliably, so the TCP client MUST also include an AccECN Option on the first data segment it sends (if it ever sends one).

If the TCP client has successfully negotiated AccECN but does not receive an AccECN Option on the SYN/ACK, it switches into a mode that assumes that the AccECN Option is not available for this half connection. Similarly, if the TCP server has successfully negotiated AccECN but does not receive an AccECN Option on the first ACK or on the first data segment, it switches into a mode that assumes that the AccECN Option is not available for this half connection.

While a host is in the mode that assumes the AccECN Option is not available, it MUST adopt the conservative interpretation of the ACE field discussed in Section 3.2.2. However, it cannot make any assumption about support of the AccECN Option on the other half connection, so it MUST continue to send the AccECN Option itself.

If after the normal TCP timeout the TCP server has not received an ACK to acknowledge its SYN/ACK, the SYN/ACK might just have been lost, e.g. due to congestion, or a middlebox might be blocking the AccECN Option. To expedite connection setup, the host SHOULD fall back to NS=CWR=ECE=0 and no AccECN Option on the retransmission of the SYN/ACK. Implementers MAY use other fall-back strategies if they are found to be more effective (e.g. retransmitting a SYN/ACK with AccECN TCP flags but not the AccECN Option; attempting to retransmit a second AccECN segment before fall-back (most appropriate during

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

high levels of congestion); or falling back to classic ECN rather than non-ECN).

Similarly, if the TCP client detects that the first data segment it sent was lost, it SHOULD fall back to no AccECN Option on the retransmission. Again, implementers MAY use other fall-back strategies such as attempting to retransmit a second segment with the AccECN Option before fall-back, and/or caching the result of previous attempts.

Either host MAY include the AccECN Option in a subsequent segment to retest whether the AccECN Option can traverse the path.

Currently the Data Sender is not required to test whether the arriving byte counters in the AccECN Option have been correctly initialised. This allows different initial values to be used as an additional signalling channel in future. If any inappropriate zeroing of these fields is discovered during testing, this approach will need to be reviewed.

### 3.2.5. Usage of the AccECN TCP Option

The following rules determine when a Data Receiver in AccECN mode sends the AccECN TCP Option, and which fields to include:

**Change-Triggered ACKs:** If an arriving packet increments a different byte counter to that incremented by the previous packet, the Data Receiver SHOULD immediately send an ACK with an AccECN Option, without waiting for the next delayed ACK. Certain offload hardware might not be able to support change-triggered ACKs, but otherwise it is important to keep exceptions to this rule to a minimum so that Data Senders can generally rely on this behaviour;

**Continual Repetition:** Otherwise, if arriving packets continue to increment the same byte counter, the Data Receiver can include an AccECN Option on most or all (delayed) ACKs, but it does not have to. If option space is limited on a particular ACK, the Data Receiver MUST give precedence to SACK information about loss. It SHOULD include an AccECN Option if the r.ceb counter has incremented and it MAY include an AccECN Option if r.ec0b or r.ec1b has incremented;

**Full-Length Options Preferred:** It SHOULD always use full-length AccECN Options, but it MAY use shorter AccECN Options as long as it includes the counter that just incremented;

**Beaconing Full-Length Options:** Nonetheless, it MUST include a full-length AccECN TCP Option on at least three ACKs per RTT, or on all

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

ACKs if there are less than three per RTT (see Appendix A.4 for an example algorithm that satisfies this requirement).

The following example series of arriving marks illustrates when a Data Receiver will emit an ACK if it is using a delayed ACK factor of 2 segments and change-triggered ACKs: 01 -> ACK, 01, 01 -> ACK, 10 -> ACK, 10, 01 -> ACK, 01, 11 -> ACK, 01 -> ACK.

For the avoidance of doubt, the change-triggered ACK mechanism ignores the arrival of a control packet with no payload, because it does not alter any byte counters. The change-triggered ACK approach will lead to some additional ACKs but it feeds back the order and timing in which ECN marks are received with minimal additional complexity.

Note that sending an AccECN Option each time a different counter changes and including a full-length AccECN Option on every delayed ACK will satisfy the requirements described above and might be the easiest implementation, as long as sufficient space is available in each ACK (in total and in the option space).

Appendix A.3 gives an example algorithm to estimate the number of marked bytes from the ACE field alone, if the AccECN Option is not available.

If a host has determined that segments with the AccECN Option always seem to be lost, it is no longer obliged to follow the above rules.

### 3.3. AccECN Compliance by TCP Proxies, Offload Engines and other Middleboxes

A large class of middleboxes split TCP connections. Such a middlebox would be compliant with the AccECN protocol if the TCP implementation on each side complied with the present AccECN specification and each side negotiated AccECN independently of the other side.

Another large class of middleboxes intervene to some degree at the transport layer, but attempts to be transparent (invisible) to the end-to-end connection. A subset of this class of middleboxes attempts to 'normalise' the TCP wire protocol by checking that all values in header fields comply with a rather narrow interpretation of the TCP specifications. To comply with the present AccECN specification, such a middlebox MUST NOT change the ACE field or the AccECN Option and it MUST attempt to preserve the timing of each ACK (for example, if it coalesced ACKs it would not be AccECN-compliant). A middlebox claiming to be transparent at the transport layer MUST forward the AccECN TCP Option unaltered, whether or not the length value matches one of those specified in Section 3.2.3, and whether or

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

not the initial values of the byte-counter fields are correct. This is because blocking apparently invalid values does not improve security (because AccECN hosts are required to ignore invalid values anyway), while it prevents the standardised set of values being extended in future (because outdated normalisers would block updated hosts from using the extended AccECN standard).

Hardware to offload certain TCP processing represents another large class of middleboxes, even though it is often a function of a host's network interface and rarely in its own 'box'. Leeway has been allowed in the present AccECN specification in the expectation that offload hardware could comply and still serve its function. Nonetheless, such hardware MUST attempt to preserve the timing of each ACK (for example, if it coalesced ACKs it would not be AccECN-compliant).

#### 4. Interaction with Other TCP Variants

This section is informative, not normative.

##### 4.1. Compatibility with SYN Cookies

A TCP server can use SYN Cookies (see Appendix A of [RFC4987]) to protect itself from SYN flooding attacks. It places minimal commonly used connection state in the SYN/ACK, and deliberately does not hold any state while waiting for the subsequent ACK (e.g. it closes the thread). Therefore it cannot record the fact that it entered AccECN mode for both half-connections. Indeed, it cannot even remember whether it negotiated the use of classic ECN [RFC3168].

Nonetheless, such a server can determine that it negotiated AccECN as follows. If a TCP server using SYN Cookies supports AccECN, if the first ACK it receives contains an ACE field with the value 0b110 or 0b111, it can assume that:

- o the TCP client must have requested AccECN support on the SYN
- o it (the server) must have confirmed that it supported AccECN

Therefore the server can switch itself into AccECN mode, and continue as if it had never forgotten that it switched itself into AccECN mode earlier.

##### 4.2. Compatibility with Other TCP Options and Experiments

AccECN is compatible (at least on paper) with the most commonly used TCP options: MSS, time-stamp, window scaling, SACK and TCP-AO. It is also compatible with the recent promising experimental TCP options



Internet-Draft

Accurate TCP-ECN Feedback

September 2015

TCP Fast Open (TFO [RFC7413]) and Multipath TCP (MPTCP [RFC6824]). AccECN is particularly friendly to all these protocols, because space for TCP options is particularly scarce on the SYN, where AccECN consumes zero additional header space.

When option space is under pressure from other options, Section 3.2.5 provides guidance on how important it is to send an AccECN Option and whether it needs to be a full-length option.

#### 4.3. Compatibility with Feedback Integrity Mechanisms

The ECN Nonce [RFC3540] is an experimental IETF specification intended to allow a sender to test whether ECN CE markings (or losses) introduced in one network are being suppressed by the receiver or anywhere else in the feedback loop, such as another network or a middlebox. The ECN nonce has not been deployed as far as can be ascertained. The nonce would now be nearly impossible to deploy retrospectively, because to catch a misbehaving receiver it relies on the receiver volunteering feedback information to incriminate itself. A receiver that has been modified to misbehave can simply claim that it does not support nonce feedback, which will seem unremarkable given so many other hosts do not support it either.

With minor changes AccECN could be optimised for the possibility that the ECT(1) codepoint might be used as a nonce. However, given the nonce is now probably undeployable, the AccECN design has been generalised so that it ought to be able to support other possible uses of the ECT(1) codepoint, such as a lower severity or a more instant congestion signal than CE.

Three alternative mechanisms are available to assure the integrity of ECN and/or loss signals. AccECN is compatible with any of these approaches:

- o The Data Sender can test the integrity of the receiver's ECN (or loss) feedback by occasionally setting the IP-ECN field to a value normally only set by the network (and/or deliberately leaving a sequence number gap). Then it can test whether the Data Receiver's feedback faithfully reports what it expects [I-D.moncaster-tcpm-rcv-cheat]. Unlike the ECN Nonce, this approach does not waste the ECT(1) codepoint in the IP header, it does not require standardisation and it does not rely on misbehaving receivers volunteering to reveal feedback information that allows them to be detected. However, setting the CE mark by the sender might conceal actual congestion feedback from the network and should therefore only be done sparsely.

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

- o Networks generate congestion signals when they are becoming congested, so they are more likely than Data Senders to be concerned about the integrity of the receiver's feedback of these signals. A network can enforce a congestion response to its ECN markings (or packet losses) using congestion exposure (ConEx) audit [I-D.ietf-conex-abstract-mech]. Whether the receiver or a downstream network is suppressing congestion feedback or the sender is unresponsive to the feedback, or both, ConEx audit can neutralise any advantage that any of these three parties would otherwise gain.

ConEx is a change to the Data Sender that is most useful when combined with AccECN. Without AccECN, the ConEx behaviour of a Data Sender would have to be more conservative than would be necessary if it had the accurate feedback of AccECN.

- o The TCP authentication option (TCP-AO [RFC5925]) can be used to detect any tampering with AccECN feedback between the Data Receiver and the Data Sender (whether malicious or accidental). The AccECN fields are immutable end-to-end, so they are amenable to TCP-AO protection, which covers TCP options by default. However, TCP-AO is often too brittle to use on many end-to-end paths, where middleboxes can make verification fail in their attempts to improve performance or security, e.g. by resegmentation or shifting the sequence space.

## 5. Protocol Properties

This section is informative not normative. It describes how well the protocol satisfies the agreed requirements for a more accurate ECN feedback protocol [RFC7560].

**Accuracy:** From each ACK, the Data Sender can infer the number of new CE marked segments since the previous ACK. This provides a better accuracy on CE feedback than classic ECN. In addition if the AccECN Option is present (not blocked by the network path) the number of bytes marked with CE, ECT(1) and ECT(0) are provided.

**Overhead:** The AccECN scheme is divided into two parts. The essential part reuses the 3 flags already assigned to ECN in the IP header. The supplementary part adds an additional TCP option consuming up to 11 bytes. However, no TCP option is consumed in the SYN.

**Ordering:** The order in which marks arrive at the Data Receiver is preserved in AccECN feedback, because the Data Receiver is expected to send an ACK immediately when a different mark arrives.

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

**Timeliness:** While the same ECN markings are arriving continually at the Data Receiver, it can defer ACKs as TCP does normally, but it will immediately send an ACK as soon as a different ECN marking arrives.

**Timeliness vs Overhead:** Change-Triggered ACKs are intended to enable latency-sensitive uses of ECN feedback by capturing the timing of transitions but not wasting resources while the state of the signalling system is stable. The receiver can control how frequently it sends the AccECN TCP Option and therefore it can control the overhead induced by AccECN.

**Resilience:** All information is provided based on counters. Therefore if ACKs are lost, the counters on the first ACK following the losses allows the Data Sender to immediately recover the number of the ECN markings that it missed.

**Resilience against Bias:** Because feedback is based on repetition of counters, random losses do not remove any information, they only delay it. Therefore, even though some ACKs are change-triggered, random losses will not alter the proportions of the different ECN markings in the feedback.

**Resilience vs Overhead:** If space is limited in some segments (e.g. because more option are need on some segments, such as the SACK option after loss), the Data Receiver can send AccECN Options less frequently or truncate fields that have not changed, usually down to as little as 5 bytes. However, it has to send a full-sized AccECN Option at least three times per RTT, which the Data Sender can rely on as a regular beacon or checkpoint.

**Resilience vs Timeliness and Ordering:** Ordering information and the timing of transitions cannot be communicated in three cases: i) during ACK loss; ii) if something on the path strips the AccECN Option; or iii) if the Data Receiver is unable to support Change-Triggered ACKs.

**Complexity:** An AccECN protocol solely involves simple counter increments, some modulo arithmetic to communicate the least significant bits and allow for wrap, and some heuristics for safety against fields cycling due to prolonged periods of ACK loss. Each host needs to maintain eight additional counters. The hosts have to apply some additional tests to detect tampering by middleboxes, but in general the protocol is simple to understand, simple to implement and requires few cycles per packet to execute.

**Integrity:** AccECN is compatible with at least three approaches that can assure the integrity of ECN feedback. If the AccECN Option is

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

stripped the resolution of the feedback is degraded, but the integrity of this degraded feedback can still be assured.

**Backward Compatibility:** If only one endpoint supports the AccECN scheme, it will fall-back to the most advanced ECN feedback scheme supported by the other end.

**Backward Compatibility:** If the AccECN Option is stripped by a middlebox, AccECN still provided basic congestion feedback in the ACE field. Further, AccECN can be used to detect mangling of the IP ECN field; mangling of the TCP ECN flags; blocking of ECT-marked segments; and blocking of segments carrying the AccECN Option. It can detect these conditions during TCP's 3WSH so that it can fall back to operation without ECN and/or operation without the AccECN Option.

**Forward Compatibility:** The behaviour of endpoints and middleboxes is carefully defined for all reserved or currently unused codepoints in the scheme, to ensure that any blocking of anomalous values is always at least under reversible policy control.

## 6. IANA Considerations

This document defines a new TCP option for AccECN, assigned a value of TBD1 (decimal) from the TCP option space. This value is defined as:

Kind	Length	Meaning	Reference
TBD1	N	Accurate ECN (AccECN)	RFC XXXX

[TO BE REMOVED: This registration should take place at the following location: <http://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1>]

Early implementation before the IANA allocation MUST follow [RFC6994] and use experimental option 254 and magic number 0xACCE (16 bits) {ToDo register this with IANA}, then migrate to the new option after the allocation.

## 7. Security Considerations

If ever the supplementary part of AccECN based on the new AccECN TCP Option is unusable (due for example to middlebox interference) the essential part of AccECN's congestion feedback offers only limited resilience to long runs of ACK loss (see Section 3.2.2). These

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

problems are unlikely to be due to malicious intervention (because if an attacker could strip a TCP option or discard a long run of ACKs it could wreak other arbitrary havoc). However, it would be of concern if AccECN's resilience could be indirectly compromised during a flooding attack. AccECN is still considered safe though, because if the option is not presented, the AccECN Data Sender is then required to switch to more conservative assumptions about wrap of congestion indication counters (see Section 3.2.2 and Appendix A.2).

Section 4.1 describes how a TCP server can negotiate AccECN and use the SYN cookie method for mitigating SYN flooding attacks.

There is concern that ECN markings could be altered or suppressed, particularly because a misbehaving Data Receiver could increase its own throughput at the expense of others. Given the experimental ECN nonce is now probably undeployable, AccECN has been generalised for other possible uses of the ECT(1) codepoint to avoid obsolescence of the codepoint even if the nonce mechanism is obsoleted. AccECN is compatible with the three other schemes known to assure the integrity of ECN feedback (see Section 4.3 for details). If the AccECN Option is stripped by an incorrectly implemented middlebox, the resolution of the feedback will be degraded, but the integrity of this degraded information can still be assured.

The AccECN protocol is not believed to introduce any new privacy concerns, because it merely counts and feeds back signals at the transport layer that had already been visible at the IP layer.

## 8. Acknowledgements

We want to thank Michael Welzl for his input and discussion. The idea of using the three ECN-related TCP flags as one field for more accurate TCP-ECN feedback was first introduced in the re-ECN protocol that was the ancestor of ConEx.

Bob Briscoe was part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700) and through the Trilogy 2 project (ICT-317756). The views expressed here are solely those of the authors.

## 9. Comments Solicited

Comments and questions are encouraged and very welcome. They can be addressed to the IETF TCP maintenance and minor modifications working group mailing list <tcpm@ietf.org>, and/or to the authors.

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

## 10. References

### 10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<http://www.rfc-editor.org/info/rfc3168>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<http://www.rfc-editor.org/info/rfc5681>>.
- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <<http://www.rfc-editor.org/info/rfc6994>>.

### 10.2. Informative References

- [I-D.bensley-tcpm-dctcp] Bensley, S., Eggert, L., Thaler, D., Balasubramanian, P., and G. Judd, "Microsoft's Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters", draft-bensley-tcpm-dctcp-05 (work in progress), July 2015.
- [I-D.ietf-conex-abstract-mech] Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism and Requirements", draft-ietf-conex-abstract-mech-13 (work in progress), October 2014.
- [I-D.kuehlewind-tcpm-ecn-fallback] Kuehlewind, M. and B. Trammell, "A Mechanism for ECN Path Probing and Fallback", draft-kuehlewind-tcpm-ecn-fallback-01 (work in progress), September 2013.
- [I-D.moncaster-tcpm-rcv-cheat] Moncaster, T., Briscoe, B., and A. Jacquet, "A TCP Test to Allow Senders to Identify Receiver Non-Compliance", draft-moncaster-tcpm-rcv-cheat-03 (work in progress), July 2014.

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <<http://www.rfc-editor.org/info/rfc3540>>.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <<http://www.rfc-editor.org/info/rfc4987>>.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <<http://www.rfc-editor.org/info/rfc5562>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<http://www.rfc-editor.org/info/rfc5925>>.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <<http://www.rfc-editor.org/info/rfc6824>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<http://www.rfc-editor.org/info/rfc7413>>.
- [RFC7560] Kuehlewind, M., Ed., Scheffenegger, R., and B. Briscoe, "Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback", RFC 7560, DOI 10.17487/RFC7560, August 2015, <<http://www.rfc-editor.org/info/rfc7560>>.

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

## Appendix A. Example Algorithms

This appendix is informative, not normative. It gives example algorithms that would satisfy the normative requirements of the AccECN protocol. However, implementers are free to choose other ways to implement the requirements.

## A.1. Example Algorithm to Encode/Decode the AccECN Option

The example algorithms below show how a Data Receiver in AccECN mode could encode its CE byte counter `r.ceb` into the ECEB field within the AccECN TCP Option, and how a Data Sender in AccECN mode could decode the ECEB field into its byte counter `s.ceb`. The other counters for bytes marked ECT(0) and ECT(1) in the AccECN Option would be similarly encoded and decoded.

It is assumed that each local byte counter is an unsigned integer greater than 24b (probably 32b), and that the following constant has been assigned:

$$\text{DIVOPT} = 2^{24}$$

Every time a CE marked data segment arrives, the Data Receiver increments its local value of `r.ceb` by the size of the TCP Data. Whenever it sends an ACK with the AccECN Option, the value it writes into the ECEB field is

$$\text{ECEB} = \text{r.ceb} \% \text{DIVOPT}$$

where `'%'` is the modulo operator.

On the arrival of an AccECN Option, the Data Sender uses the TCP acknowledgement number and any SACK options to calculate `newlyAckedB`, the amount of new data that the ACK acknowledges in bytes. If `newlyAckedB` is negative it means that a more up to date ACK has already been processed, so this ACK has been superseded and the Data Sender has to ignore the AccECN Option. Then the Data Sender calculates the minimum difference `d.ceb` between the ECEB field and its local `s.ceb` counter, using modulo arithmetic as follows:

```
if (newlyAckedB >= 0) {
    d.ceb = (ECEB + DIVOPT - (s.ceb % DIVOPT)) % DIVOPT
    s.ceb += d.ceb
}
```

For example, if `s.ceb` is 33,554,433 and ECEB is 1461 (both decimal), then



Internet-Draft

Accurate TCP-ECN Feedback

September 2015

```

s.ceb % DIVOPT = 1
d.ceb = (1461 + 2^24 - 1) % 2^24
        = 1460
s.ceb = 33,554,433 + 1460
        = 33,555,893

```

## A.2. Example Algorithm for Safety Against Long Sequences of ACK Loss

The example algorithms below show how a Data Receiver in AccECN mode could encode its CE packet counter `r.ceb` into the ACE field, and how the Data Sender in AccECN mode could decode the ACE field into its `s.ceb` counter. The Data Sender's algorithm includes code to heuristically detect a long enough unbroken string of ACK losses that could have concealed a cycle of the congestion counter in the ACE field of the next ACK to arrive.

Two variants of the algorithm are given: i) a more conservative variant for a Data Sender to use if it detects that the AccECN Option is not available (see Section 3.2.2 and Section 3.2.4); and ii) a less conservative variant that is feasible when complementary information is available from the AccECN Option.

### A.2.1. Safety Algorithm without the AccECN Option

It is assumed that each local packet counter is a sufficiently sized unsigned integer (probably 32b) and that the following constant has been assigned:

$$\text{DIVACE} = 2^3$$

Every time a CE marked packet arrives, the Data Receiver increments its local value of `r.ceb` by 1. It repeats the same value of ACE in every subsequent ACK until the next CE marking arrives, where

$$\text{ACE} = \text{r.ceb} \% \text{DIVACE}.$$

If the Data Sender received an earlier value of the counter that had been delayed due to ACK reordering, it might incorrectly calculate that the ACE field had wrapped. Therefore, on the arrival of every ACK, the Data Sender uses the TCP acknowledgement number and any SACK options to calculate `newlyAckedB`, the amount of new data that the ACK acknowledges. If `newlyAckedB` is negative it means that a more up to date ACK has already been processed, so this ACK has been superseded and the Data Sender has to ignore the AccECN Option. If `newlyAckedB` is zero, to break the tie the Data Sender could use timestamps (if present) to work out `newlyAckedT`, the amount of new time that the ACK acknowledges. Then the Data Sender calculates the minimum difference

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

d.cep between the ACE field and its local s.cep counter, using modulo arithmetic as follows:

```
if ((newlyAcedB > 0) || (newlyAcedB == 0 && newlyAcedT > 0))
    d.cep = (ACE + DIVACE - (s.cep % DIVACE)) % DIVACE
```

Section 3.2.2 requires the Data Sender to assume that the ACE field did cycle if it could have cycled under prevailing conditions. The 3-bit ACE field in an arriving ACK could have cycled and become ambiguous to the Data Sender if a row of ACKs goes missing that covers a stream of data long enough to contain 8 or more CE marks. We use the word 'missing' rather than 'lost', because some or all the missing ACKs might arrive eventually, but out of order. Even if some of the lost ACKs are piggy-backed on data (i.e. not pure ACKs) retransmissions will not repair the lost AcceCN information, because AcceCN requires retransmissions to carry the latest AcceCN counters, not the original ones.

The phrase 'under prevailing conditions' allows the Data Sender to take account of the prevailing size of data segments and the prevailing CE marking rate just before the sequence of ACK losses. However, we shall start with the simplest algorithm, which assumes segments are all full-sized and ultra-conservatively it assumes that ECN marking was 100% on the forward path when ACKs on the reverse path started to all be dropped. Specifically, if newlyAcedB is the amount of data that an ACK acknowledges since the previous ACK, then the Data Sender could assume that this acknowledges newlyAcedPkt full-sized segments, where newlyAcedPkt = newlyAcedB/MSS. Then it could assume that the ACE field incremented by

```
dSafer.cep = newlyAcedPkt - ((newlyAcedPkt - d.cep) % DIVACE),
```

For example, imagine an ACK acknowledges newlyAcedPkt=9 more full-size segments than any previous ACK, and that ACE increments by a minimum of 2 CE marks (d.cep=2). The above formula works out that it would still be safe to assume 2 CE marks (because  $9 - ((9-2) \% 8) = 2$ ). However, if ACE increases by a minimum of 2 but acknowledges 10 full-sized segments, then it would be necessary to assume that there could have been 10 CE marks (because  $10 - ((10-2) \% 8) = 10$ ).

Implementers could build in more heuristics to estimate prevailing average segment size and prevailing ECN marking. For instance, newlyAcedPkt in the above formula could be replaced with newlyAcedPktHeur = newlyAcedPkt\*p\*MSS/s, where s is the prevailing segment size and p is the prevailing ECN marking probability. However, ultimately, if TCP's ECN feedback becomes inaccurate it still has loss detection to fall back on. Therefore, it would seem safe to implement a simple algorithm, rather than a perfect one.

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

The simple algorithm for dSafer.cep above requires no monitoring of prevailing conditions and it would still be safe if, for example, segments were on average at least 5% of full-sized as long as ECN marking was 5% or less. Assuming it was used, the Data Sender would increment its packet counter as follows:

```
s.cep += dSafer.cep
```

If missing acknowledgement numbers arrive later (due to reordering), Section 3.2.2 says "the Data Sender MAY attempt to neutralise the effect of any action it took based on a conservative assumption that it later found to be incorrect". To do this, the Data Sender would have to store the values of all the relevant variables whenever it made assumptions, so that it could re-evaluate them later. Given this could become complex and it is not required, we do not attempt to provide an example of how to do this.

#### A.2.2. Safety Algorithm with the AcceECN Option

When the AcceECN Option is available on the ACKs before and after the possible sequence of ACK losses, if the Data Sender only needs CE-marked bytes, it will have sufficient information in the AcceECN Option without needing to process the ACE field. However, if for some reason it needs CE-marked packets, if dSafer.cep is different from d.ceb, it can calculate the average marked segment size that each implies to determine whether d.ceb is likely to be a safe enough estimate. Specifically, it could use the following algorithm, where d.ceb is the amount of newly CE-marked bytes (see Appendix A.1):

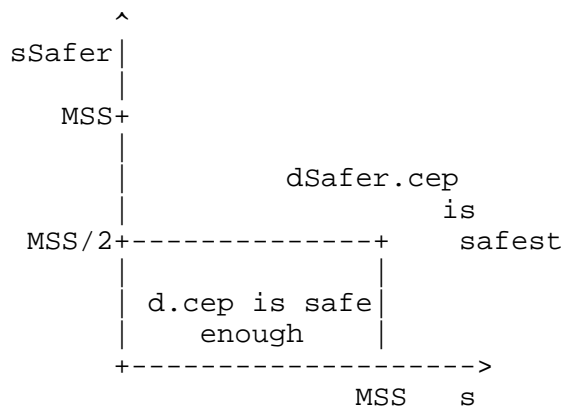
```
SAFETY_FACTOR = 2
if (dSafer.cep > d.ceb) {
    s = d.ceb/d.ceb
    if (s <= MSS) {
        sSafer = d.ceb/dSafer.cep
        if (sSafer < MSS/SAFETY_FACTOR)
            dSafer.cep = d.ceb    % d.ceb is a safe enough estimate
    } % else
        % No need for else; dSafer.cep is already correct,
        % because d.ceb must have been too small
}
```

The chart below shows when the above algorithm will consider d.ceb can replace dSafer.cep as a safe enough estimate of the number of CE-marked packets:

Internet-Draft

Accurate TCP-ECN Feedback

September 2015



The following examples give the reasoning behind the algorithm, assuming  $MSS=1,460$  [B]:

- o if  $d.cep=0$ ,  $dSafer.cep=8$  and  $d.ceb=1,460$ , then  $s=infinity$  and  $sSafer=182.5$ .  
Therefore even though the average size of 8 data segments is unlikely to have been as small as  $MSS/8$ ,  $d.cep$  cannot have been correct, because it would imply an average segment size greater than the  $MSS$ .
- o if  $d.cep=2$ ,  $dSafer.cep=10$  and  $d.ceb=1,460$ , then  $s=730$  and  $sSafer=146$ .  
Therefore  $d.cep$  is safe enough, because the average size of 10 data segments is unlikely to have been as small as  $MSS/10$ .
- o if  $d.cep=7$ ,  $dSafer.cep=15$  and  $d.ceb=10,200$ , then  $s=1,457$  and  $sSafer=680$ .  
Therefore  $d.cep$  is safe enough, because the average data segment size is more likely to have been just less than one  $MSS$ , rather than below  $MSS/2$ .

If pure ACKs were allowed to be ECN-capable, missing ACKs would be far less likely. However, because [RFC3168] currently precludes this, the above algorithm assumes that pure ACKs are not ECN-capable.

### A.3. Example Algorithm to Estimate Marked Bytes from Marked Packets

If the AccECN Option is not available, the Data Sender can only decode CE-marking from the ACE field in packets. Every time an ACK arrives, to convert this into an estimate of CE-marked bytes, it needs an average of the segment size,  $s_{ave}$ . Then it can add or subtract  $s_{ave}$  from the value of  $d.ceb$  as the value of  $d.cep$  increments or decrements.

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

To calculate `s_ave`, it could keep a record of the byte numbers of all the boundaries between packets in flight (including control packets), and recalculate `s_ave` on every ACK. However it would be simpler to merely maintain a counter `packets_in_flight` for the number of packets in flight (including control packets), which it could update once per RTT. Either way, it would estimate `s_ave` as:

```
s_ave ~= flightsize / packets_in_flight,
```

where `flightsize` is the variable that TCP already maintains for the number of bytes in flight. To avoid floating point arithmetic, it could right-bit-shift by `lg(packets_in_flight)`, where `lg()` means log base 2.

An alternative would be to maintain an exponentially weighted moving average (EWMA) of the segment size:

```
s_ave = a * s + (1-a) * s_ave,
```

where `a` is the decay constant for the EWMA. However, then it is necessary to choose a good value for this constant, which ought to depend on the number of packets in flight. Also the decay constant needs to be power of two to avoid floating point arithmetic.

#### A.4. Example Algorithm to Beacon AccECN Options

Section 3.2.5 requires a Data Receiver to beacon a full-length AccECN Option at least 3 times per RTT. This could be implemented by maintaining a variable to store the number of packets since the AccECN Option was last sent:

```
if (packets_since_last_sent > packets_in_flight / BEACON_FREQ)
    send_AccECN_Option()
```

For optimised integer arithmetic, `BEACON_FREQ = 4` could be used, rather than 3, so that the division could be implemented as an integer right bit-shift by `lg(BEACON_FREQ)`.

#### A.5. Example Algorithm to Count Not-ECT Bytes

A Data Sender in AccECN mode can infer the amount of TCP payload data arriving at the receiver marked Not-ECT from the difference between the amount of newly ACKed data and the sum of the bytes with the other three markings, `d.ceb`, `d.e0b` and `d.e1b`. Note that, because `r.e0b` is initialised to 1 and the other two counters are initialised to 0, the initial sum will be 1, which matches the initial offset of the TCP sequence number on completion of the 3WSH.

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

For this approach to be precise, it has to be assumed that spurious (unnecessary) retransmissions do not lead to double counting. This assumption is currently correct, given that RFC 3168 requires that the Data Sender marks retransmitted segments as Not-ECT. However, the converse is not true; necessary transmissions will result in under-counting.

However, such precision is unlikely to be necessary. The only known use of a count of Not-ECT marked bytes is to test whether equipment on the path is clearing the ECN field (perhaps due to an out-dated attempt to clear, or bleach, what used to be the ToS field). To detect bleaching it will be sufficient to detect whether nearly all bytes arrive marked as Not-ECT. Therefore there should be no need to keep track of the details of retransmissions.

#### Appendix B. Alternative Design Choices (To Be Removed Before Publication)

This appendix is informative, not normative. It records alternative designs that the authors chose not to include in the normative specification, but which the IETF might wish to consider for inclusion:

Feedback all four ECN codepoints on the SYN/ACK: The last two negotiation combinations in Table 2 could also be used to indicate AccECN support and to feedback that the arriving SYN was ECT(0) or ECT(1). This could be used to probe the client to server path for incorrect forwarding of the ECN field [I-D.kuehlewind-tcpm-ecn-fallback]. Note, however, that it would be unremarkable if ECN on the SYN was zeroed by security devices, given RFC 3168 prohibited ECT on SYN because it enables DoS attacks.

Feedback all four ECN codepoints on the First ACK: To probe the server to client path for incorrect ECN forwarding, it could be useful to have four feedback states on the first ACK from the TCP client. This could be achieved by assigning four combinations of the ECN flags in the main TCP header, and only initialising the ACE field on subsequent segments.

Empty AccECN Option: It might be useful to allow an empty (Length=2) AccECN Option on the SYN/ACK and first ACK. Then if a host had to omit the option because there was insufficient space for a larger option, it would not give the impression to the other end that a middlebox had stripped the option.

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

#### Appendix C. Open Protocol Design Issues (To Be Removed Before Publication)

1. Currently it is specified that the receiver 'SHOULD' use Change-Triggered ACKs. It is controversial whether this ought to be a 'MUST' instead. A 'SHOULD' would leave the Data Sender uncertain whether it can rely on the timing and ordering information in ACKs. If the sender guesses wrongly, it will probably introduce at least 1RTT of delay before it can use this timing information. Ironically it will most likely be wanting this information to reduce ramp-up delay. A 'MUST' could make it hard to implement AccECN in offload hardware. However, it is not known whether AccECN would be hard to implement in such hardware even with a 'SHOULD' here. For instance, was it hard to offload DCTCP to hardware because of change-triggered ACKs, or was this just one of many reasons? The choice between MUST and SHOULD here is critical. Before that choice is made, a clear use-case for certainty of timing and ordering information is needed, plus well-informed discussion about hardware offload constraints.
2. There is possibly a concern that a receiver could deliberately omit the AccECN Option pretending that it had been stripped by a middlebox. No known way can yet be contrived to take advantage of this downgrade attack, but it is mentioned here in case someone else can contrive one.
3. The s.cep counter might increase even if the s.ceb counter does not (e.g. due to a CE-marked control packet). The sender's response to such a situation is considered out of scope, because this ought to be dealt with in whatever future specification allows ECN-capable control packets. However, it is possible that the situation might arise even if the sender has not sent ECN-capable control packets, in which case, this draft might need to give some advice on how the sender should respond.

#### Appendix D. Changes in This Version (To Be Removed Before Publication)

The difference between any pair of versions can be displayed at <http://datatracker.ietf.org/doc/draft-kuehlewind-tcpm-accurate-ecn/history/>

From 03 to 04::

- \* Extensively rewritten. Algorithm was simplified to use an CE packet counter in the ACE (instead of codepoints) and a TCP option for additional byte counter information. No detailed summary of changes has been prepared.

Internet-Draft

Accurate TCP-ECN Feedback

September 2015

## Authors' Addresses

Bob Briscoe  
Simula Research Laboratory  
  
EMail: [ietf@bobbbriscoe.net](mailto:ietf@bobbbriscoe.net)  
URI: <http://bobbbriscoe.net/>

Mirja Kuehlewind  
ETH Zurich  
Gloriastrasse 35  
Zurich 8092  
Switzerland  
  
EMail: [mirja.kuehlewind@tik.ee.ethz.ch](mailto:mirja.kuehlewind@tik.ee.ethz.ch)

Richard Scheffenegger  
NetApp, Inc.  
Am Euro Platz 2  
Vienna 1120  
Austria  
  
Phone: +43 1 3676811 3146  
EMail: [rs@netapp.com](mailto:rs@netapp.com)