Department of Computer Science
Series of Publications A
Report A-2019-7

Congestion Control and Active Queue Management During Flow Startup

Ilpo Järvinen

Doctoral dissertation, to be presented for public examination with the permission of the Faculty of Science of the University of Helsinki, in Auditorium A129, Chemicum building, Kumpula, Helsinki, on the 14th of November, 2019 at 12 o'clock.

University of Helsinki
Finland

Supervisor
Markku Kojo, University of Helsinki, Finland
Sasu Tarkoma, University of Helsinki, Finland

Pre-examiners
Pasi Sarolahti, Aalto University, Finland
Michael Welzl, University of Oslo, Norway

Opponent
Anna Brunström, Karlstad University, Sweden

Custos
Sasu Tarkoma, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/
Telephone: +358 2941 911

ISSN 1238-8645
ISBN 978-951-51-5585-6 (paperback)
ISBN 978-951-51-5586-3 (PDF)
Helsinki 2019
Unigrafia

Congestion Control and Active Queue Management During Flow Startup

Ilpo Järvinen

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
ilpo.jarvinen@cs.helsinki.fi

PhD Thesis, Series of Publications A, Report A-2019-7
Helsinki, November 2019, 87+48 pages
ISSN 1238-8645
ISBN 978-951-51-5585-6 (paperback)
ISBN 978-951-51-5586-3 (PDF)

Abstract

Transmission Control Protocol (TCP) has served as the workhorse to transmit Internet traffic for several decades already. Its built-in congestion control mechanism has proved reliable to ensure the stability of the Internet, and congestion control algorithms borrowed from TCP are being applied largely also by other transport protocols. TCP congestion control has two main phases for increasing sending rate. Slow Start is responsible for starting up a flow by seeking the sending rate the flow should use. Congestion Avoidance then takes over to manage the sending rate for flows that last long enough. In addition, the flow is booted up by sending the Initial Window of packets prior to Slow Start.

There is a large difference in the magnitude of sending rate increase during Slow Start and Congestion Avoidance. Slow Start increases the sending rate exponentially, whereas with Congestion Avoidance the increase is linear. If congestion is detected, a standard TCP sender reduces the sending rate heavily. It is well known that most of the Internet flows are short. It implies that flow startup is a rather frequent phenomenon.Â Also, many traffic types exhibit an ON-OFF pattern with senders remaining idle for varying periods of time. As the flow startup under Slow Start causes exponential sending rate increase, the link load is often subject to exponential load transients that escalate in a few round trips into overload, if not controlled properly. It is true especially near the network edge where traffic aggregation is limited to a few users.

Traditionally much of the congestion control research has focused on behavior during Congestion Avoidance and uses large aggregates during testing. To control router load, Active Queue Management (AQM) is recommended. The state-of-the-art AQM algorithms, however, are designed with little attention to Slow Start. This thesis focuses on congestion control and AQM during the flow startup. We explore what effect the Initial Window has to competing latency-sensitive traffic during a flow startup consisting of multiple parallel flows typical to Web traffic and investigate the impact of increasing Initial Window from three to ten TCP segments. We also highlight what the shortcomings are in the state-of-the-art AQM algorithms and formulate the challenges AQM algorithms must address to properly handle flow startup and exponential load transients. These challenges include the horizon problem, RTT (round-trip time) uncertainty and rapidly changing load. None of the existing AQM algorithms are prepared to handle these challenges. Therefore we explore whether an existing AQM algorithm called Random Early Detection (RED) can be altered to control exponential load transients effectively and propose necessary changes to RED. We also propose an entirely new AQM algorithm called Predict. It is the first AQM algorithm designed primarily for handling exponential load transients.

Our evaluation shows that because of shortcomings in handling exponential load transients, the state-of-the-art AQM algorithms often respond too slowly or too fast depending on the actual RTT of the traffic. In contrast, the Predict AQM algorithm performs timely congestion indication without compromising throughput or latency unnecessarily, yielding low latency over a large range of RTTs. In addition, the load estimation in Predict is designed to be fully compatible with pacing and the timely congestion indication allows relaxing the large sending rate reduction on congestion detection.

Computing Reviews (2012) Categories and Subject Descriptors:
C.2.1 Network Architecture and Design
C.2.2 Network Protocols
C.2.3 Network Operations
C.2.6 Internetworking
C.4 Performance of Systems

General Terms:
Active Queue Management, Congestion Control, Access Networks, TCP

Additional Key Words and Phrases:
Load Transients, Flow Startup

Acknowledgements

I am very thankful and lucky to have Markku Kojo as my closest PhD supervisor who has tirelessly used time on reviewing both the papers and thesis manuscript. His suggestions to improve the contents have been invaluable. The second special mention goes to Laila Daniel who encouraged and persuaded me to continue on solving the remaining accuracy stumbling blocks for Predict that had been, at that point, put aside for years as unworkable. Without it, the thesis might have needed to take another course.

Besides the coauthors in the papers, a number of other people have had a significant impact on making this thesis to happen. My second PhD supervisor Sasu Tarkoma has friendly kept "bugging" me on the progress of the research and thesis. Without the excellent support of our PhD coordinator Pirjo Moen, the steps when nearing the finish line would have been much more painful to understand. The language in the thesis owes much to Marina Kurtén who provided a long list of corrections to improve the grammar and style. I am also thankful of the useful feedback from the pre-examiners Michael Welzl and Pasi Sarolahti.

Part of this work has been carried out in Wireless Broadband Access (WiBrA) project funded by TEKES Finland, Nokia, Nokia Siemens Networks, and TeliaSonera Finland.

Helsinki, October 2019
Ilpo Järvinen

List of Reprinted Publications

Flow Initiation

Research paper I: I. Järvinen, B. Chemmagate, A. Y. Ding, L. Daniel, M. Isomäki, J. Korhonen, and M. Kojo. "Effect of Competing TCP Traffic on Interactive Real-Time Communication". In: *Proc. 14th Passive and Active Measurement Conference (PAM 2013), Lecture Notes in Computer Science.* (Hong Kong). Vol. 7799. Springer Berlin Heidelberg, March 2013, pp. 94–103. DOI: 10.1007/978-3-642-36516-4_10

Active Queue Management During Flow Startup

Research paper II: I. Järvinen, Y. Ding, A. Nyrhinen, and M. Kojo. "Harsh RED: Improving RED for Limited Aggregate Traffic". In: *Proc. 26th IEEE International Conference on Advanced Information Networking and Applications (AINA-2012).* (Fukuoka, Japan). IEEE, March 2012, pp. 832–840. DOI: 10.1109/AINA.2012.103

Research paper III: I. Järvinen and M. Kojo. "Evaluating CoDel, PIE, and HRED AQM Techniques with Load Transients". In: *Proc. 39th IEEE Conference on Local Computer Networks (LCN 2014).* (Edmonton, Alberta, Canada). IEEE, September 2014, pp. 159–167. DOI: 10.1109/LCN.2014.6925768

Research paper IV: I. Järvinen and M. Kojo. "Gazing Beyond Horizon: The Predict Active Queue Management for Controlling Load Transients". In: *Proc. 31st IEEE International Conference on Advanced Information Networking and Applications (AINA-2017).* (Taipei, Taiwan (ROC)). IEEE, March 2017, pp. 126–135. DOI: 10.1109/AINA.2017.110

List of Abbreviations

3G	Third Generation Cellular Technology
ABC	Appropriate Byte Counting
ABE	Alternative Backoff with ECN
ACK	Acknowledgement
AQM	Active Queue Management
ARED	Adaptive RED
AVQ	Adaptive Virtual Queue
BBR	Bottleneck Bandwidth and Round-trip Propagation Time
BDP	Bandwidth Delay Product
CBI	Control Block Interdependence
CBR	Constant Bit Rate
CDF	Cumulative Distribution Function
CE	Congestion Experienced
CoDel	Controlled Delay
CWR	Congestion Window Reduced
CWV	Congestion Window Validation
DAASS	Delayed ACK After Slow Start
DASH	Dynamic Adaptive Streaming over HTTP
DCH	Decicated Channel
DCTCP	Data Center TCP
DNS	Domain Name System
DOCSIS	Data Over Cable Service Interface Specification
DRR	Deficit Round Robin
DSL	Digital Subscriber Line
DualQ	Coupled Dual Queue AQM
ECE	ECN-Echo
ECN	Explicit Congestion Notification
ECT	ECN Capable Transport
EDGE	Enhanced Data Rates for GSM Evolution
EWMA	Exponentially Weighted Moving Average
FCT	Flow Completion Time

FIFO	First-in, First-out
FIFO	First In First Out
FPGA	Field-Programmable Gate Array
FQ-CoDel	FlowQueue-CoDel
FTP	File Transfer Protocol
GRED	Gentle RED
GSM	Global System for Mobile Communications
HAS	HTTP Adaptive Streaming
HRED	Harsh Random Early Detection
HSPA	High-Speed Packet Access
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HULL	High-bandwidth Ultra Low Latency
IETF	Internet Engineering Task Force
IPDV	IP Packet Delay Variation
IP	Internet Protocol
IQR	Interquartile Range
IW	Initial Window
L4S	Low Latency, Low Loss, Scalable Throughput
LEDBAT	Low Extra Delay Background Transport
LPF	Low-Pass Filter
MADPIE	Maximum and Average Queuing Delay with PIE
MD	Multiplicative Decrease
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NTP	Network Time Protocol
NVP	Non-Validated Period
P2P	Peer-to-Peer
PIE	Proportional Integral Controller Enhanced
PI	Proportional Integral Controller
PQ	Phanthom Queue
QUIC	Quick UDP Internet Connections
RED	Random Early Detection
RFC	Request For Comment
RTO	Retransmission Timeout
RTP	Real-time Transport Protocol
RTT	Round-Trip Time
SACK	Selective Acknowledgement
SFQ	Stochastic Fair Queuing
SRED	Stabilized RED

SSL	Secure Socket Layer
SYN	Synchronize
TCP	Transmission Control Protocol
TFO	TCP Fast Open
TLS	Transport Layer Security
UDP	User Datagram Protocol
VCP	Variable-structure Congestion Control Protocol
WLAN	Wireless Local Area Network
WRED	Weighted RED
WWW	World Wide Web
XCP	Explicit Control Protocol

Contents

Chapter 1

Introduction

Congestion control is essential to ensure proper functioning of the Internet. Without congestion control, the Internet would become unusable due to overload. The congestion control mechanism of Transmission Control Protocol (TCP) [13, 151] has proved reliable to ensure the stability of the Internet, and congestion control algorithms borrowed from TCP are being applied largely also by other transport protocol.

1.1 Background and Motivation

Internet traffic patterns have changed over the years. One major reason is the World Wide Web (WWW). Soon after its invention in 1989, the web became very popular and since then a large portion of Internet traffic has been web traffic [185]. Web pages are transferred using Hypertext Transfer Protocol (HTTP) [23, 26, 73–75] that is layered on top of Transmission Control Protocol (TCP) [151]. Web traffic consists largely of short TCP flows and alternation between active and inactive transmission periods is frequent [1, 49, 56, 63]. Web browsers typically use parallel connections to reduce latency in fetching a web page that consists of many HTTP objects.

TCP has a built-in congestion control mechanism to ensure the stability of the Internet by preventing congestion collapse [102, 140] and to prevent the use of excessive sending rate over a prolonged period of time. The TCP congestion control mechanism is self-clocked using acknowledgement packets (ACKs) as feedback and tries to follow a "packet conservation principle" that allows sending a new data packet to the network only when another packet has left it [102]. TCP congestion control has two main phases: Slow Start and Congestion Avoidance [13]. A TCP sender first uses Slow Start to probe for the available capacity on the end-to-end path that is unknown

to the sender. The ACK clock is booted up for Slow Start by sending up to the TCP Initial Window (IW) worth of packets to the network. Then the sender increases the sending rate exponentially during Slow Start. The sender has to violate the packet conservation principle every time it increases the sending rate because obeying the principle would only allow maintaining the same sending rate. Slow Start continues until the TCP flow hits "the ceiling" that the sender discerns through packets dropped in the network due to congestion. Once the proper sending rate is acquired, the TCP sender continues in Congestion Avoidance, which is much less aggressive compared with Slow Start. We can compare these two congestion control phases to a storm and calm waters, the storm being Slow Start and calm waters matching Congestion Avoidance. To make things worse, the storms caused by Slow Start do not settle until the sender finds the ceiling through packet drops (assuming there is enough payload to be transmitted in the TCP flow to feed the storm).

The use of short TCP flows with web traffic emphasizes the importance of the Slow-Start phase that is now the norm rather than an infrequent occurrence. It also makes the Initial Window important because web traffic often uses parallel flows to transfer a web page. Slow Start produces exponential load transients. The effect of exponential load transients on the load of a link is most significant close to the network edge where traffic aggregation is limited to only a single or a few users sharing the link. Such an access network or access link is often also the bottleneck link, that is, the narrowest link on the whole end-to-end path. When the bottleneck link is close to an end-user, it is very likely also saturable by one end-user alone, in contrast to the network core where the traffic aggregation level is higher than near the network edge.

When the sending rate is higher than the rate of the bottleneck link, a queue forms at the router before the bottleneck link. The router needs to somehow manage the queue. Traditionally, a simple First-in, First-out (FIFO) policy together with drop from the tail when the buffer is full (Taildrop) has been used. Due to issues with persistently full or nearly full buffers, Active Queue Management (AQM) was introduced to proactively drop packets before the buffer becomes full to leave buffer space for transient bursts. On the early round-trips (RTT) of Slow Start, the queue is typically transient as the link is not yet saturated. The link drains the queue during the intermediate idle periods of each RTT. However, Slow Start rapidly escalates from a low load to the full load and overload in just a few RTTs because of the exponential nature of the sending rate increase. Once the end-to-end path becomes fully utilized by Slow Start, a standing queue

forms in front of the bottleneck link. If the AQM algorithm does not react in time, the standing queue keeps growing with a rapid rate leading to a large delay spike that may harm competing traffic, for example, a competing interactive media flow or a flow in the same web transaction. Because AQM algorithms have been designed only for coping with Congestion Avoidance, the rapid load growth during exponential load transients frequently takes them by surprise. Slow Start easily overpowers the control authority in AQM algorithms leading to suboptimal performance.

Designing AQM algorithms with awareness for exponential load transients has serious challenges. This thesis explores those challenges, namely the horizon problem and RTT uncertainty. Many AQM algorithms base their control decisions on queue visible at the router, either measuring queue length or queuing delay. The horizon problem, however, prevents the router from acquiring a complete picture of the traffic load by simply measuring its current queuing, because the distribution of the load over the end-to-end path keeps some of the load-inducing packets out of the view of the router. At the same time, inability to know the RTTs of the flows going through the router also makes it challenging for the AQM algorithm to use a proper measurement interval for load calculation as it does not know what that interval should be.

Even though the experiments in this thesis are carried out using TCP, we believe it does not significantly limit the generality of the findings. The generality comes from other protocols running on top of another transport protocol, mainly User Datagram Protocol (UDP) [152], implementing congestion control features that are heavily borrowed from TCP algorithms (e.g., QUIC [100, 101, 124]). In addition, more and more traffic that has traditionally transferred over UDP is being moved to work on top of HTTP which is especially true with streaming traffic. It is now largely transmitted using HTTP adaptive streaming (HAS) [182] instead of, for example, Real-time Transport Protocol (RTP) [174] on top of UDP. As HTTP traffic is transmitted using either TCP itself or using QUIC that implements TCP-like congestion control, our focus on TCP is very valid in the Internet of today.

Given the frequent occurrence of exponential load transients in normal network traffic, considering them during AQM algorithm design seems a natural design choice. To ignore them is like designing a ship (AQM algorithm) only for calm waters. Unfortunately, exponential load transients are rather nasty storms that occur very frequently shaking the unequipped ship over and over again. As the storms caused by exponential load tran-

sients are such that they are only quenched after the AQM algorithm takes serious actions, the AQM designs ignore them only at their own peril.

This thesis focuses on the behavior during a flow startup and exponential load transients that occur during the bandwidth-probing phase. We cover (a) the issues that occur due to the unpreparedness of the congestion control and AQM algorithms to the rapid changes in the load and (b) propose solutions that are aware of exponential load transients.

1.2 Problem Statement and Scope

This thesis is composed of two research themes related to flow startup. The first research theme focuses on **flow initiation**, that is, to the very first round trip (RTT) during which data is being transmitted when a flow starts up. In addition to existing aggressive behavior in the Internet with web traffic using parallel TCP flows, there is a proposal on making the TCP Initial Window larger [50]. This relatively recent proposal and web traffic using parallel TCP flows need to be evaluated together. The related research questions are as follows:

RQ1 Does parallel flow initiation introduce load transients with delays that are large enough to be harmful to competing delay-sensitive traffic?

RQ2 What is the impact of the proposed increased TCP Initial Window?

RQ1 addresses the effect of existing behavior in web browsers that typically open a large number of parallel TCP flows. RQ2 aims to give insights into how the aggressiveness develops if the proposed TCP modifications are widely deployed.

The second research theme focuses on **Active Queue Management (AQM) during flow startup**. Our initial research questions related to existing work on this theme are:

RQ3 Do the state-of-the-art AQM algorithms [80, 94, 142, 143, 145, 146] work properly during exponential load transients that occur due to flow startup?

RQ4 What are the issues AQM algorithms need to solve to work properly during exponential load transients?

RQ3 seeks for an answer to whether the existing AQM algorithms are adequate for handling flow startup-related exponential load transients. In addition, a negative answer to RQ3 gives insights into what the problems

with the state-of-the-art AQM algorithms are, leading us to RQ4. To answer RQ4, we were able to crystallize the AQM problems with exponential load transients to two root problems that we labelled as the horizon problem and RTT uncertainty. Together those two problems make the load estimation on a router challenging. The issues caused by these problems are somewhat known beforehand to congestion control research and listed as open challenges by RFC 6077 [147]. This thesis refines the understanding about these challenges.

Since none of the existing AQM algorithms properly addresses the issues that occur during exponential load transients, we realized that we need a new AQM algorithm that is designed primarily with the exponential load transients in mind. The remaining research questions are related to such an AQM algorithm:

RQ5 Can an AQM algorithm measure link load properly if it solves the horizon problem and RTT uncertainty?

RQ6 Can the known characteristics of the exponential load transient be used by an AQM algorithm to predict the load into the near future?

RQ7 Can an AQM algorithm send a congestion signal at the right point of time to terminate the bandwidth-probing phase of a connection when the bottleneck link becomes fully utilized?

RQ5 seeks for an answer to whether the more accurate understanding on why AQM algorithms have a hard time to correctly manage the exponential load transients helps in constructing an AQM algorithm that properly measures the current link load. If the answer to RQ5 is yes, the acquired knowledge about the parameters of the exponential load transient may allow further refinement of the AQM algorithm. The refinement is formalized in RQ6 aiming to give predictive power to the AQM algorithm. The prediction allows the AQM algorithm to circumvent a third problem that occurs during exponential load transients due to rapidly changing load. The high rate of change in the load makes the current link load estimate already stale once the router is able to enforce a sender response that occurs only after one additional RTT. The prediction powers the AQM algorithm to take into account the RTT-long delay in feedback allowing the router to send timely congestion signals to the senders, which could enable solving RQ7 about timely feedback to an ongoing bandwidth-probing phase.

This thesis focuses on load transients that occur during flow startup. The behavior and problems that occur after the flow startup during TCP Congestion Avoidance are beyond the scope of this thesis. Thus, advanced

TCP algorithms that only start working during the Congestion Avoidance phase of the flow such as CUBIC [88, 159] and Compound TCP [180, 184] fall outside the scope.

Load transients have a big impact on the load of the routers at the network edge or near it. Such routers typically serve only one to a few users with limited flow aggregate and low overall utilization. This thesis focuses on the effects of load transients to such routers. In many carrier-grade routers on the Internet core the level of traffic aggregation is very high and the effect of a single flow or an end host on their load is miniscule. AQM on such a router is outside the scope of this thesis, however, as long as any single sender may cause an overload situation, the work in this thesis is relevant also on such routers. The overload condition can occur if a single sender has enough capacity to saturate the outgoing link or can cause a significant load swing on top of the other traffic.

The scope of this thesis is further narrowed by excluding flow handshake-related problems and improvements. Therefore TCP three-way handshake [151] improvements (e.g., TCP Fast Open [47]) and possible use of Transport Layer Security (TLS) [64–66, 157, 188] (formerly Secure Socket Layer (SSL) [83]) are outside the scope. While the number of RTTs needed during the handshake increases flow completion times, which is a problem from the end-user perspective, it is less of a problem from the congestion control point of view. It is the actual payloads that are more problematic for the congestion control and therefore this thesis focuses on the behavior after the completion of the initial handshakes.

1.3 Methodology

In this thesis, we perform experimental network measurements to evaluate performance during a flow startup. The experiments are conducted either with real cellular 3G hardware or as simulations using the ns2 network simulator [99]. We use workloads that mimic mainly web traffic object response with a rough level of abstraction excluding the effects of Domain Name System (DNS) [138, 139] queries, web object requests, and delays due to structural dependencies within web page, which all are relevant to real web transactions. To validate the measurement toolset, a smaller set of preliminary tests is first run to identify problems. If unwanted behavior due to bugs, misconfigurations, misimplementations, etc. are found during analysis, the issues are fixed before running the full set of tests. During the work, we identified and fixed a few issues in the ns2 RED implementation. The results from the experiments are carefully analyzed in order to confirm

the validity of the results and to locate the root cause for each phenomenon. If a validity problem is found only at this late stage, the problems are fixed and the affected experiments are rerun.

In this thesis we use mainly workloads and metrics that try to minimize the impact of samples from the flows that have transitioned to Congestion Avoidance. This selection is based on our observation that often the transient problems are short in duration compared with the overall lifetime of many experiments by other researchers. Therefore, the transient problems often fail to show up clearly, if at all, in statistics that include a vast number of samples from non-transient phases of the connections. In our measurements interest is quite often placed on the worst-case behavior because it sets a clear performance bound that is independent of the aspects we excluded during our web traffic abstraction. While the aspects abstracted away would also affect user perceived performance, in practice it would explode the required test space to cover a comprehensive set of real web pages.

In this thesis, we use mainly TCP to provoke congestion. However, the flow startup-related problems are generalizable to other transport protocols. Often such protocols even borrow heavily from the vast library of TCP algorithms developed to handle various scenarios. Even if the algorithms used by the other protocol are not exactly the same, they often still include elements similar to those of TCP, which stems from the TCP-friendly concept [79]. As the TCP-friendly concept dictates that the other transport protocols cannot be too aggressive compared with a competing TCP flow, it has led to adoption of similar behavior in practice. Furthermore, there is only quite a limited set of ways to do the flow startup anyway, which forces adoption of similar behavior for the flow startup regardless of the protocol. In addition, measurements show that when the link capacity is highly utilized, HTTP running on top of TCP is by volume responsible for most of the traffic [166].

1.4 Thesis Contributions

This thesis contributes to the knowledge on how aggressiveness during a flow startup affects performance of the flow and the co-existing flows that share the same bottleneck link, especially the peak queuing delay. The first research theme about flow initiation explores how the end-host congestion control needs to initiate flows in order to not produce delay spikes harmful to concurrent traffic that may be sensitive to latency.

The second research theme identifies and proposes solutions to the interactions between flow startup and AQM algorithms. To the best of our knowledge, we are the first to develop AQM algorithms based primarily on behavior during exponential load transients. We measure how the existing mainstream AQM algorithms perform during exponential load transients, identify their key shortcomings, and propose improved AQM algorithms that are prepared to handle exponential load transients.

The research in this thesis is presented in four original publications that are referred to as Pub. I, II, III, and IV. As the research conducted in this thesis also covers proposals active in the Internet Engineering Task Force (IETF), an important part of the research is contributing the evaluation results of the active proposals to IETF. The contribution of each publication is described below.

Pub. I: In this paper we evaluate the impact of aggressive flow startup. The evaluation covers the impact of parallel connections typical to web traffic combined with the proposal to increase TCP Initial Window (IW) [50] from three to ten TCP segments [1]. The experiments for this paper are conducted in a real high-speed cellular network and the impact of the aggressive flow startup is quantified using the delay imposed on a concurrent interactive media flow that is latency sensitive. The work in this paper answers RQ1 and RQ2 giving insight into how the end hosts should handle flow initiation in order to not harm latency-sensitive traffic that may be competing. We also confirm in this paper that a long-lived bulk TCP connection in the particular cellular network cause excessive buffering known as bufferbloat [84] that has a very devastating impact on interactive traffic. The performance impacts due to the long-lived bulk TCP connection, however, are beyond the scope of this thesis.

The author is the major contributor to the measurement toolset and to the result analysis. Some parts of the toolset originate from earlier tests by Aki Nyrhinen and Binoy Chemmagate but they were improved by the author. Effectively, most of the toolset was rewritten by the improvements from the author and by a limited amount also the new work from Binoy Chemmagate. The author is the main contributor of the paper content. Binoy Chemmagate was responsible for running the experiments and did small parts of the result analysis. Laila Daniel provided useful advice. Aaron Yi Ding assisted with related work and together with Markku Kojo gave useful suggestions. Markus Isomäki and Jouni Korhonen worked as technical advisers.

[1] We also evaluated the impact of reduced initial RTO [148] but found out its impact small and due to space limitation results were not included into Pub. I.

Pub. II: This paper highlights how the Random Early Detection (RED) AQM algorithm [80] responds too slowly when it encounters exponential load transients that are typical to occur with limited aggregate traffic and proposes the HRED (Harsh RED) AQM algorithm that properly responds to exponential load transients. The paper compares HRED performance to a FIFO queue with the simple Taildrop and RED with recommended parameters using the ns2 network simulator [99]. For this paper, we developed a simulation model for exponential load transients that mimics behavior that, for example, a web transaction using parallel connections to transmit HTTP objects would cause when aggregate traffic over a router is limited. Such condition commonly occurs at the network edge where the traffic for a router originates from one or a few users. This ns2 model is then used in all our subsequent work. While analyzing the results from the simulations we also discovered issues in the ns2 RED implementation. We fixed the issues before rerunning the affected simulations.

The results in this paper show how RED with the recommended parameters leads to anomalous behaviors during exponential load transients which causes drastic performance reduction. HRED, on the other hand, is shown to properly control the queue without anomalies. Based on the HRED experience we discover that contrary to the conventional wisdom on setting the RED parameters to respond "slowly" to congestion, it is safer to operate RED "too fast" to prevent anomalous behavior in the RED algorithm. This paper also shows that an equal response from an AQM algorithm to TCP Slow Start and Congestion Avoidance [13, 102] causes performance issues because they have a very different level of aggressiveness.

The author developed HRED, the measurement tools, and performed the result analysis. Discussion about exponential load transients and TCP Slow Start nuances with Aki Nyrhinen were very helpful while designing HRED. The author is the main contributor of the paper content. Figure 2 and Figure 1 were made by Aaron Yi Ding, the latter according to the instructions of the author. Aaron Yi Ding helped with the related work and together with Markku Kojo helped in writing and finalizing the paper.

Pub. III: In this paper we evaluate how AQM algorithms perform during exponential load transients. The paper compares HRED proposed in Pub. II to the state-of-the-art AQM algorithms CoDel (Controlled Delay) [142, 143] and PIE (Proportional Integral controller Enhanced) [145, 146] using the ns2 simulation model we developed in Pub. II. This paper together with Pub. II answers RQ3. The results show that the state-of-the-art AQM algorithms handle exponential load transients inadequately. We also discovered an issue in the departure rate estimator of PIE that

effectively disabled the whole PIE algorithm when the link bandwidth is low. We have contributed to the PIE specification [145] to ensure it does not leave the particular part of the algorithm unclear.

The author is the sole toolset contributor and performed the result analysis. The author is the main contributor of the paper content but has received valuable feedback and suggestions from the coauthor.

Pub. IV: This paper explores the issues AQM algorithms face during exponential load transients and proposes a new AQM algorithm called Predict. The paper refines the understanding about the limitations in the existing AQM algorithms by defining the horizon problem and RTT uncertainty that make load estimation challenging for routers. The HRED algorithm proposed in Pub. II went as far as the state-of-the-art AQM algorithms seem to be able to in responding to exponential load transients. However, HRED is still seriously limited by the requirement for a known, constant end-to-end RTT, that is, it would need to solve the RTT uncertainty. Instead of trying to improve HRED, this paper proposes the Predict AQM algorithm that not only solves the horizon problem and RTT uncertainty but is also able to predict the load into the near future, effectively answering RQ5, RQ6, and RQ7 positively. Pub. II found out that it is necessary to differentiate responses depending on whether there is Slow Start in progress or not, Predict solves this differentiation by detecting whether exponential load transient is in fact taking place and only then responding aggressively.

For this paper, we run simulation experiments using an extended version of the exponential load transient model that was used in Pub. II and III. The experiments cover a wide set of network paths with different RTTs to validate the Predict AQM algorithm and compare it against PIE, CoDel, and SFQ-CoDel. The results show that there is room for significant peak delay improvements if the design of an AQM algorithm properly addresses exponential load transients and that the current AQM algorithms have a hard time to correctly manage exponential load transients. The additional results acquired in this paper further reinforce that the state-of-the-art AQM algorithms do not adequately solve exponential load transients and may even do significant harm to performance (RQ3).

The Predict AQM algorithm we developed is able to control the fastest and therefore the most dangerous form of load growth transients that a congestion-controlled load can experience. Therefore, we believe we may have solved one of the open questions in congestion control research related to information acquisition [147].

The author crystallized the horizon problem and RTT uncertainty, and the limitations they impose on AQM. The author developed the Predict AQM algorithm and alone extended the toolset from the previous two papers to be suitable for the measurements done in this paper. The author did the entire result analysis and is the main contributor of the paper content. Again, the author has received valuable feedback and suggestions from the coauthor.

1.5 Structure of the Thesis

The rest of this thesis is organized as follows. Chapter 2 provides the background to flow startup first highlighting how flow startup relevance has increased due to developments in the Internet traffic. Then it explains various components related to the flow startup process and covers the flow initiation research theme explored by Pub. I. Chapter 3 focuses on Active Queue Management (AQM) during flow startup that is the second research theme in this thesis. It first introduces state of the AQM art, then explains how these AQM algorithms behave during load transients that are caused by flow startup, and finally discusses solutions that address the shortcomings in the state-of-the-art AQM algorithms. Table 1.1 summarizes the mapping between the research questions and the content of this thesis. Chapter 4 concludes this thesis, presents the areas that need future work, and highlights how the discoveries and insights gained in this thesis may turn out useful for a future Internet congestion control architecture. The four original publications describing the research conducted in this thesis are included at the end of this thesis.

Table 1.1: Summary of the thesis structure, research themes, and questions.

Theme	Research Question	Main Sections	Original Publications
Flow Initialization	RQ1	2.2	Pub. I
	RQ2	2.2	Pub. I
Active Queue Management During Flow Startup	RQ3	3.2, 3.3	Pub. II, Pub. III, Pub. IV
	RQ4	3.2, 3.3	Pub. II, Pub. III, Pub. IV
	RQ5	3.3, 3.4	Pub. II, Pub. IV
	RQ6	3.3, 3.4	Pub. IV
	RQ7	3.3, 3.4	Pub. IV

Chapter 2

Flow Startup and Congestion Control

The lifetime of any flow in the Internet begins with flow startup. With a large number of flows coming and going in common Internet traffic, it is important to understand how flow startup and congestion control interact. This chapter focuses on topics related to flow startup. First in Section 2.1 we highlight the developments in the Internet traffic that has led to a significant increase in relevance of the flow startup for good performance of flows in the Internet. In the next two sections we discuss the flow startup phases: flow initiation that relates to RQ1, RQ2, and Pub. I is discussed in Section 2.2; and the following bandwidth-probing phase is discussed in Section 2.3. Section 2.5 discusses optional mechanisms for exiting the bandwidth-probing phase before a congestion signal. In Section 2.4 we discuss congestion that often ends the bandwidth-probing phase, associated actions related to sender response, and possibly needed loss recovery. Section 2.6 concentrates on restarting flows after periods of inactivity or low activity. Finally, a summary is provided in Section 2.7.

2.1 Importance of Flow Startup due to Internet Traffic Developments

TCP congestion control [13] is based on a "packet conservation principle" that is enacted when the TCP flows in the network operate "in equilibrium" [102]. The main purpose of the TCP congestion control was initially to prevent "congestion collapse" that caused abysmal performance due to unnecessary retransmissions [102, 140] by operating TCP flows in a manner that ensures a stable network [102]. The TCP congestion control also aims

to utilize the network capacity efficiently but within the constraints imposed by its primary goal of avoiding the congestion collapse. Slow Start used for probing available capacity was seen only as a mandatory violation of the packet conservation principle until packet conservation can be achieved [102]. This means that the major focus of the congestion control work has been on Congestion Avoidance in equilibrium and that is reflected in a rather large number of congestion control-related models, proofs, etc. that are based on the "steady-state" operation. The steady-state operation implies expectation of large, long-running bulk TCP flows. In general, the progress of a long-running bulk TCP flow is at its best when congestion control does not reduce the throughput of the flow which was long used as the main or sole metric to gauge congestion control performance.

It was soon realized that congestion control needs to do more than prevent the congestion collapse. TCP congestion control aims for a condition where the network cannot hold even a single packet more. A TCP sender keeps increasing its sending rate on each RTT even in Congestion Avoidance to probe the network for higher available capacity as long as there is payload to transmit. Only after a dropped packet is detected, the sender backs off temporarily and then again continues trying to drive the network to the maximal level of utilization and buffer usage. TCP congestion control together with a simple drop-from-tail-when-buffer-is-full policy, known as Taildrop, leads to full or close to full buffers. To prevent problems associated with full buffers, Active Queue Management (AQM), such as Random Early Detection (RED) [80], was introduced and later recommended to be deployed by Internet routers [31]. But again, the AQM research and design focused on solutions for Congestion Avoidance and steady state. Nevertheless, the need for lower delay was recognized as a target for AQM algorithms.

The World Wide Web (WWW) was invented in 1989 [27]. A web page is a collection of objects that are glued together and displayed by a web browser according to the instructions given in Hypertext Markup Language (HTML) [25, 190, 191]. In order to display a web page, the web browser performs a web transaction first fetching the main object of the web page called body object from a remote web server. Based on parsing the body object, the browser decides what additional objects need to be fetched. Those additional objects are called inline objects and may contain images, scripts, stylesheets, etc. Processing of an inline object may require further inline objects to be fetched. Once all required objects are transferred, the web transaction is complete. Some web browsers may start to display the web page while the web transaction is still in progress, however, the actual

representation of the web page to the end user and issues related to it are out of scope for this thesis.

Web objects are transferred over Hypertext Transfer Protocol (HTTP). HTTP/1.0 [26] performs fetching of each web object over a Transmission Control Protocol (TCP) [151] connection and once completed, it closes the connection. If persistent connections are enabled, it is possible to reuse a TCP connection without closing it and opening another TCP connection for the next object to be fetched. As many objects typically originate from the same remote server, HTTP/1.1 [73–75] introduces HTTP pipelining to enable re-using a TCP connection more efficiently for multiple objects as HTTP pipelining allows the next request to be made before the first object is fully transferred. Also, the TCP connections with HTTP/1.1 are persistent by default.

One characteristic of web traffic is the presence of ON and OFF periods that refer to whether an HTTP transmission is currently active or not. The OFF periods are caused by (a) the user reading and thinking (inactive OFF) and (b) the web browser processing the HTTP objects received so far before the browser is able to decide what HTTP object to request next (active OFF) [1, 56]. These ON and OFF periods are also reflected in various HTTP models [21, 49, 63, 181]. A recent study from real traffic captures confirms the presence of internal OFF periods for roughly a quarter to one third of all connections [165].

In the 1990s, the web became very popular. By 1995 web traffic made up roughly one fifth of all Internet traffic [141]. Only two years after that in 1997, the share of the HTTP traffic had grown so that more than 70% of the Internet traffic was HTTP traffic [185].

While the web traffic, HTTP models, and measurements mentioned above are old, HTTP and web traffic changed very little for many years. The most significant change after HTTP/1.1 was the complexity growth in the web pages increasing the sizes of the HTTP objects and the number of the objects that were required to complete web transactions [153, 155]. As the complexity grew, also the latency of a web transaction became larger. The latency arms race between browser makers was a result. HTTP/1.1 allows up to two parallel connections to be used for a single server to better utilize the resources during a web transaction [74] but the number of allowed parallel connections was increased from two by the browser makers to reduce the latency to win against competing browsers [169]. In addition, many complex web sites were not satisfied with the off-the-shelf parallelism and resorted to a technique called domain sharding. The domain sharding artificially inflates the number of hostnames the inline objects of a web

page appear to originate from which circumvents the per hostname limit on the number of allowed parallel connections. Therefore the number of parallel connections a browser may open during a web transaction became very large [42, 178].

Many of the HTTP objects are quite small [153, 155] and therefore the TCP flows transmitting them mostly operate in Slow Start. Many of such TCP flows may never leave Slow Start before the data of the transmitted HTTP object runs out. As web traffic forms a significant portion of Internet traffic, the short TCP flows transmitting HTTP objects play a significant role for the load at the routers. TCP senders increase sending rate exponentially per RTT in Slow Start producing exponential load transients. At this point, the situation is a network that is designed for operation in Congestion Avoidance, whereas the transfers for HTTP objects mostly operate in the much more aggressive Slow Start probing for the sending rate they should use. As the load growth during an exponential load transient is very fast, the TCP flows using Slow Start can congest links just in a few RTTs, which is much faster than the AQM algorithms expect by design.

Short TCP flows are often regarded as "mice", in contrast to "elephants" that are much larger. Unfortunately, the "mouse" notion can easily convey a false message that such flows are not important from the congestion control point of view. However, a web transaction is not a single "mouse" but an "army of mice" that together may quickly eat up all the resources because the flows of the web transaction often start up and increase sending rate simultaneously. Together the footprint of the short web traffic flow "mice" is significant and is comparable to a longer TCP flow in Slow Start. Often HTTP pipelining even naturally combines the HTTP objects to a longer "train of mice" that is transmitted over a single TCP flow. A measurement study based on real traffic captures confirms that it is reasonable to assume that for flows originating from a single endpoint, a small number of flows at each time tend to proceed past TCP Initial Window to form bandwidth probing aggregates [7].

While congestion can occur almost anywhere in the network, some links are more likely to get congested than the others. One such case is the link with the least bandwidth along an end-to-end path. The link with the least bandwidth is commonly close to the network edge, that is, the access link or a link in the access network is likely to have smaller bandwidth than the links in the network core [67]. The links close to the network edge have only a limited level of traffic aggregation serving only one to a few end users. Typically, the access links also have low overall utilization [134, 166, 177]. Therefore, the impact of a single user or a single web transaction on

the load of the link is large. As OFF periods natural to the web traffic are frequent, the new connections often start their Slow Start when the link at the network edge is idle [34] or has low load. An exponential load transient then rapidly increases the load but after a relatively short ON period, the link becomes idle again and the process soon repeats itself once the end user initiates the next web transaction. A study on access link performance [176, 177] confirms that (a) web traffic is likely a significant contributor to access link saturation, that (b) volume-wise the saturated periods are "almost negligible" which hints at the presence of a significant amount of OFF periods, and that (c) the link utilization over longer periods of time is very low for most of the clients. The first observation is true even though the methodology used in the analysis filters a significant portion of HTTP traffic away by limiting the analysis only to flows with at least 190kB of payload. In addition, another measurement confirms that HTTP is responsible for 91% of the data volume when the capacity is being highly utilized [166]. The low overall link utilization for most of the broadband links is likewise confirmed by a recent study [134].

On routers in the Internet core with a high level of traffic aggregation, exponential load transients are less of a problem as the effect of a single user or an end host on the load is miniscule. The network topology towards the Internet core may dictate that no end host has enough access-link capacity to saturate an outgoing link of such a router. However, as long as there is any single source with enough capacity to cause significant load swings, the work in this thesis may have some usefulness also on large routers although such routers are not the main target for this work.

While web traffic keeps increasing, other traffic types have surpassed it by volume. First, peer-to-peer (P2P) traffic with long-running downloads and later video streaming became volume-wise larger than web traffic [52, 154, 198]. Nowadays, streaming is even more important [53]. Some of the streaming traffic is interactive and therefore sensitive to latency. With interactive game traffic, likewise, low latency often improves user experience. These interactive traffic types may not be the most dominant in the overall traffic volumes and are often not the cause for high load. Despite their innocence, they commonly still experience very noticeable effects during high-load situations negatively affecting user experience.

Today HTTP is no longer limited to its initial use for web traffic but HTTP use has expanded to many things besides the web. Most of the non-interactive streaming today uses HTTP adaptive streaming (HAS) [182] (also known as dynamic adaptive streaming over HTTP (DASH)) to deliver content to the end-users (services such as YouTube, Netflix, Spotify, etc.).

As HTTP adaptive streaming runs on top of TCP, TCP still is the most used protocol in the Internet. A HAS stream is split into chunks and each chunk is encoded with the multiple advertized rates [17, 125]. A client decides which of the advertized rates it uses in fetching each chunk. The fetched chunks are stored into a playback buffer. Typically the HAS client keeps making chunk requests in advance until the playback buffer has a good level of occupancy. Then it needs to wait until some of the playback buffer is consumed before resuming the advance chunk fetching. Thus, despite the playback itself being continuous, the HAS client has an internal tendency to produce ON-OFF periods. Therefore, it is hardly surprising that Slow Start is shown to occur often with HAS [17].

In order to reduce latency, new proposals on how to transfer web objects have been made, which also address the use of excessive number of parallel connections. First, SPDY [22, 149, 179] that uses a single connection to transmit many HTTP objects by multiplexing them. Then, the follow up for SPDY work in the context of HTTP/2 [23] that supports many of the SPDY features, including the multiplexing of HTTP objects into a single TCP connection. Further improvements are made in QUIC [29, 101] that removes the TCP protocol-level head-of-line blocking issue. The head-of-line blocking occurs with SPDY and HTTP/2 when a TCP packet is lost and the TCP receiver has to wait until a retransmitted copy of the packet arrives because TCP supports only in-order delivery of the data. With the head-of-line blocking issue resolved, there is even less need for using parallel connections to quickly deliver HTTP objects across the network. The QUIC congestion control [100] is heavily based on the existing TCP congestion control algorithms. The exponential load transients, however, are not removed by any of the techniques as probing for available capacity is needed independent of the number of flows. QUIC may eventually become "the next TCP" that is used to transport the majority of the Internet content.

Besides latency in transmitting the actual HTTP payload, the user-visible latency includes the latency caused by the TCP three-way handshake delay. TCP Fast Open (TFO) [47] optimizes the TCP handshake by allowing subsequent connections to the same destination to embed data already to the TCP SYN packet. QUIC [101, 124] as an UDP-based protocol goes even further removing the entire three-way handshake for connections to the same destination and also combines TLS handshake to further reduce connection setup latency for encrypted connections. These handshake latency-related optimizations, however, are out of scope for this thesis as

they are quite unimportant from the congestion control point of view we focus on.

2.2 Flow Initiation

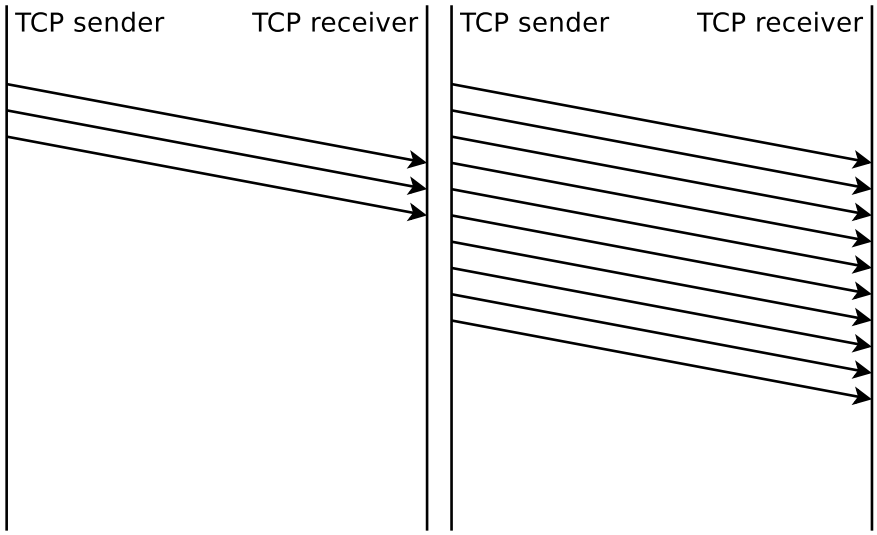
At the beginning of a TCP connection after the three-way handshake, a TCP sender starts the data transmission by sending up to the number of segments allowed by the Initial Window (IW) [12]. With the typical maximum transmission unit (MTU) of 1500 bytes, up to three segments are sent in the Initial Window. The segments in the Initial Window are used to boot up the ACK clock, a constant stream of ACK packets flowing in the reverse direction, that is used for sending packets after the Initial Window. The Initial Window is "uncontrolled" traffic in the sense that it is sent without knowing that packets are leaving the network.

Relatively recently, a larger Initial Window has been proposed for experimental use to reduce latency [50]. The larger Initial Window reduces the number of RTTs needed to complete short transfers that are typical, for example, with web objects. The larger Initial Window allows up to 10 segments to be sent without verifying that there is capacity available for them. Figure 2.1 shows the flow startup with the Initial Window of three and ten segments. A study with web search traffic indicates that the Initial Window of ten segments instead of three improves average latency by roughly 10% [68]. In addition, the latency improvement increases with higher quantiles. A measurement study indicates that one fourth to one fifth of the TCP connections become more aggressive because of the Initial Window larger than three segments [7].

Combining the larger Initial Window with parallel flows starting up at the same point of time can result in a large number of segments injected into the network. Such a large uncontrolled burst of packets may be harmful to other flows in the network. When aggregated effect is considered, measurements in the residential access network indicate that for 5%-12% of the cases[1] there is amplified effect due to combined larger Initial Windows from multiple flows [7]. Pub. I focuses on measuring the effect of the larger Initial Window and the combined effects of the parallel flows and larger Initial Window to competing latency-sensitive traffic over a high-speed cellular network.

Pub. I shows that the impact of sending the Initial Window is very significant for the delay experienced by the packets of the competing traffic.

[1] Aggregate grouping in [7] is worst-case analysis with one second time window to form an aggregated group and therefore likely includes some flows that do not group for real.



(a) IW3 (b) IW10

Figure 2.1: TCP Initial Window (IW).

With one or two flows, the quality impact is only small but issues become more severe as the number of flows starting at the same time increases. With the proposed larger Initial Window of ten segments, even the impact of a single flow exceeds the impact of six parallel flows using the Initial Window of three segments. The measurements made in Pub. I also show that the bad quality the large number of parallel flows or the proposed larger Initial Window cause is not limited to the Initial Window RTT but prolongs over the duration of the TCP Slow Start bandwidth-probing phase.

In order to alleviate the problems caused by excessive bursts due to the larger Initial Window, initial spreading has been proposed [163, 164]. Initial spreading separates the Initial Window packets to individual, smaller transients over the expected RTT that was measured during the three-way handshake. A similar proposal but with an additional removal of the Initial Window altogether has also been made in [9]. Long before the larger Initial Window proposal, JumpStart [127] was proposed effectively removing the concept of Initial Window completely but without any spreading.

The removal of the Initial Window is very harmful to other flows in the light of Pub. I as the number of uncontrolled packets allowed to be sent will be much larger. We believe that even spreading the packets over the initial RTT is unable to solve the problems caused by an excessive initial sending rate. Whenever the sender violates the packet conservation principle, there

is a possibility to do significant harm to competing traffic. As the Initial Window sender cannot even know about the harm, not to mention react to it, before the harm has already happened, the only way to avoid harming other traffic is to retain sending huge uncontrolled initial bursts. If a huge Initial Window is used, the only hope is that statistical multiplexing is lucky enough to cover for it, that is, that there is no other traffic present which could be harmed. It is somewhat unfortunate that the sender is likely to be lucky very often because of low overall utilization near the network edge. Thus, measuring or extracting the harm caused from a large set of real world data is challenging, and it is too easy and tempting to falsely conclude that no harm is caused by the larger Initial Window. The use of Slow Start for bandwidth probing with a small Initial Window is a good way to limit the extent of the harm caused by the Initial Window as it gives every sender time to respond. A claim has been made that it is in the best interest of the sender to limit the Initial Window to avoid issues because of local resource exhaustion [9] but it is likely not enough as the competing traffic may not originate from the same source. In such case, the resources exhausted may not be local but near the network edge at the other end of the connection. Therefore, the harm caused to the competing traffic may not be measurable at all by the sender using an excessive sending rate. In the worst case, the sender might not care about the competing traffic but is willing to harm other traffic to get "better service" for its own traffic and might see the opportunity to monopolize the remote resources as a competitive advantage.

An alternative to uncontrolled sending at the beginning of data transmission is given by an old solution known as Quick-Start [78, 167, 173]. A TCP sender using Quick-Start acquires information about available capacity from the routers on the end-to-end path using IP and TCP options. In practice, however, Quick-Start requires support from every router on the end-to-end path and is therefore extremely challenging to deploy.

Various alternatives to the Initial Window of three segments are compared in [170–172].

2.3 Bandwidth-Probing Phase

Bandwidth probing is a congestion control phase at the early round-trips of the connection. The purpose of probing is to determine the capacity of the end-to-end path that is usually unknown to the end host. Figure 2.2 shows the taxonomy of possible bandwidth probing approaches.

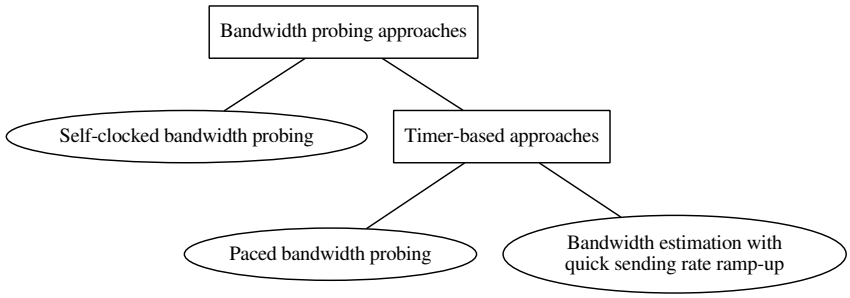


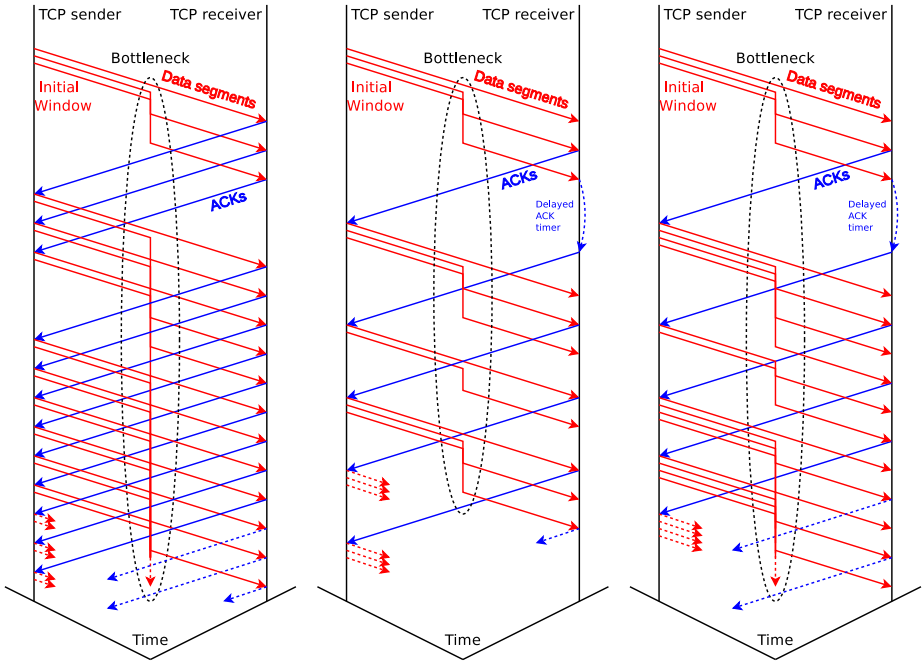
Figure 2.2: Taxonomy of bandwidth-probing approaches.

Self-Clocked Bandwidth Probing

A sender using self-clocked bandwidth probing relies on incoming acknowledgements (ACKs) for timing the outgoing packets. In addition to probing for the available capacity, the bandwidth-probing phase is used by the sender to boot up an ACK clock that is a constant stream of ACKs flowing in the reverse direction [102]. The ACK clock carries credits to the sender from which the sender can infer how many packets have left the network. The credits tell the sender how many new packets can be injected into to the network "safely", that is, without increasing the load in the network. In addition to the "safe" packets, the sender needs to inject more packets than what has left the network in order to probe for a higher sending rate than is currently being used. The sender terminates the bandwidth-probing phase once a congestion signal is detected or a high enough sending rate is attained.

In TCP the self-clocked bandwidth probing process is performed by Slow Start [13, 102]. Figure 2.3a shows the Slow-Start process. The left-hand side is the TCP sender and the right-hand side is the TCP receiver. First, the sender sends three segments as per the standard Initial Window. After the Initial Window, the sender waits for ACKs to arrive. In Slow Start, the sender sends an additional packet on each arrival of ACK besides the packets that according to the credits have left the network. The additional packet probes for more available capacity. The number of packets that can be sent is tracked by the sender using a congestion window (cwnd) variable [13]. The Slow-Start process results in doubling the congestion window and thus the sending rate per each RTT, in other words, the sending rate and the load imposed by the flow is increasing exponentially. The exponential increase in the sending rate results in exponential load transients also at the network routers. With the self-clocked bandwidth

probing, the sender has to send packets in bursts. The burst is followed by an idle period as long as the sending rate has not yet used up all the available network capacity.



(a) Slow Start

(b) Slow Start with Delayed ACK

(c) ABC corrected Slow Start with Delayed ACK

Figure 2.3: TCP Slow-Start variants.

The Slow-Start process allows up to doubling the sending rate per RTT. However, the sending rate growth may be less because of Delayed ACK [54] that was implemented decades ago to roughly halve the number of ACK packets. A smaller rate for ACKs was necessary to reduce the rate of interrupts from the network adapter and the associated cost of processing ACKs. Slow Start with Delayed ACK is shown in Figure 2.3b. When a TCP receiver is using Delayed ACK, it sends one ACK after receiving two segments instead of after every segment. If no second segment is received, the receiver waits instead until a Delayed ACK timer expires to send out the pending ACK for the single segment. Because of Delayed ACKs, instead of doubling, as low as 1.5 factor growth is possible with a typical Slow Start but the actual factor also depends on how RTT and delayed ACK timers are sized with respect to the other [14]. If the receiver does not implement

or use Delayed ACKs [2], the growth factor in Slow Start is exactly two. Also the sender TCP implementation can correct the smaller congestion window increase due to Delayed ACKs during Congestion Avoidance using Appropriate Byte Counting (ABC) [10, 13]. Use of ABC during Slow Start is allowed for experimental use [10], however, standard implementations are recommended to not increase the congestion window by more than one segment per ACK [13]. In practice, some TCP implementations have chosen to use ABC with the larger congestion window increase also during Slow Start (e.g., the Linux TCP implementation includes an ABC-based mechanism that is adapted to the packet-based TCP implementation [48]). Slow Start with Delayed ACKs and ABC is shown in Figure 2.3c. ABC restores the growth factor to two while the TCP sender is in Slow Start. For simplicity, we assume from this point onward a factor of two growth rate in this thesis unless otherwise stated.

During self-clocked bandwidth probing, the ACKs tend to arrive roughly at the rate of the narrowest link on the end-to-end path called bottleneck link because the data segments are spaced out by the bottleneck link [102] (see Figure 2.3). This implies that the sending rate of the additional packets sent during the bandwidth-probing phase exceed the rate of the bottleneck link. As the packets come in faster than can be transmitted to the bottleneck link, a queue forms at the router in front of the bottleneck link. Figure 2.4 shows an example for TCP Slow Start-induced transient queue spikes occurring at the router in front the bottleneck link. If the link is not yet saturated by the current load, the queue is only transient and the queue will be drained to the outgoing link once the short burst of packets ends. These transient queue spikes grow in amplitude as the number of packets grow exponentially, and on each RTT, half of the packets are sent with a higher rate than the bottleneck link can immediately forward. Eventually the load becomes large enough that the queue can no longer drain before the next burst of packets comes in. The bottleneck link is saturated and a standing queue with rapidly increasing length is formed. Assuming infinite flow lengths, the queue grows until congestion is signalled by the router which results in the sender discontinuing the bandwidth-probing phase.

Alternatively, the bandwidth probing may end without a congestion signal from the network. If the congestion window (cwnd) reaches the Slow-Start threshold (ssthresh) [13], the sender continues in Congestion Avoidance but typically TCP stacks set ssthresh initially to a very large value. In addition, the receiver advertized window (rwnd) may impose a

[2] E.g., Linux TCP receiver implements heuristics called quick ACKs to only use Delayed ACKs after Slow Start (DAASS) [8, 11].

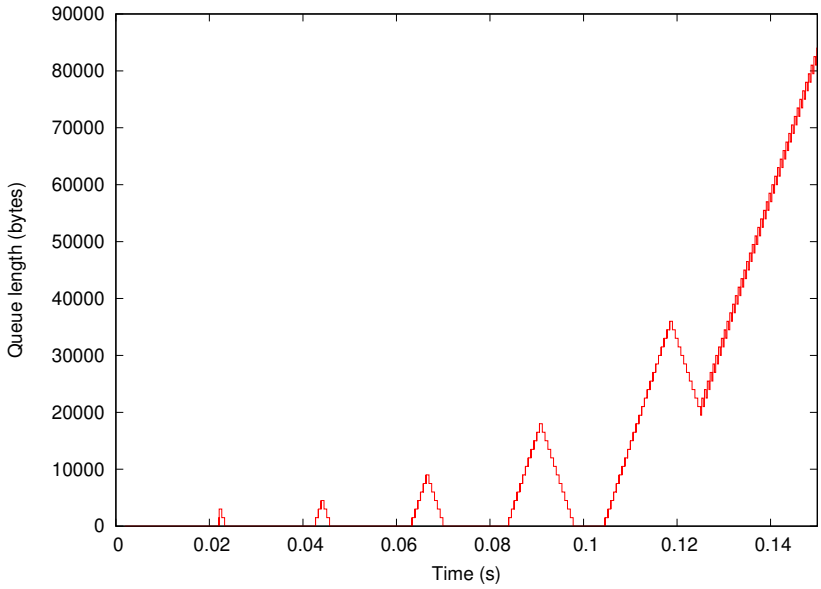


Figure 2.4: Queue transients during self-clocked bandwidth probing.

limiting upper-bound for the congestion window even if the sender has not received any congestion signal.

Paced Bandwidth Probing

The second bandwidth probing category is based on sending the packets using a timer. This timer-based sending is known as pacing [3, 16, 119]. The goal of pacing is to reduce bursts by spreading the sent packets evenly over the measured RTT. The packets are sent using a frequently expiring timer rather than when ACKs arrive. If the bandwidth probing packets are retimed to such portion of RTT that the bottleneck link rate is not exceeded due to a too nearby transmission of another packet, the probe packets do not form a queue at the bottleneck. Retiming of the transmission is possible as long as the bottleneck link is not yet saturated. After saturation, there are no more available slots to be filled using the timer as there always is the previous packet sent so recently that the sender will exceed the bottleneck link rate even with the timer.

One recent example of paced bandwidth probing is Bottleneck bandwidth and RTT (TCP BBR) [45, 46] congestion control for TCP. TCP BBR implements paced sending of all packets after the Initial Window, including pacing during bandwidth probing. Like with Slow Start, the

sending rate with TCP BBR effectively roughly doubles in one RTT but no queue is formed until the bottleneck becomes saturated (see the green line in Figure 4 in [45]). Another recent example is QUIC that recommends using pacing [100].

It is possible to use pacing for all congestion control phases but pacing is most useful during bandwidth probing as it allows maintaining a sending rate that is less than the rate of the bottleneck link until the bottleneck link becomes saturated. Thus, before saturation no transient queue is formed with paced bandwidth probing. If the bandwidth-probing phase is allowed to continue after the bottleneck link becomes saturated, however, pacing will not reduce the peak queue length because the peak queue length will occur after the bottleneck link saturation. Therefore, it would be very useful to be able to terminate the bandwidth probing exactly at the right point of time to reap queuing delay reduction benefits from the use of pacing. Section 2.5 describes possible mechanisms for exiting the bandwidth-probing phase at the right point of time.

Bandwidth Estimation with Quick Sending Rate Ramp-up

The third main bandwidth probing category is the bandwidth estimation with quick sending rate ramp-up to the estimated available capacity. The key goal in this category is to avoid extra RTTs needed to increase the sending rate "slowly" and instead quickly jump to the estimated available capacity by increasing the sending rate directly to the value derived from the estimated capacity.

One such approach is Quick-Start [78, 167, 173] that performs a negotiation of the available capacity with the routers on the end-to-end path. Quick-Start may, however, perform the sending rate ramp-up already during flow initialization and is therefore included in Section 2.2.

Other approaches such as TCP RAPID [112] estimate the available bandwidth without help from the on-path routers. The bandwidth estimation in TCP RAPID is based on the pathChirp [160] algorithm and aims to try a large sending rate range in a very short period of time. Typically the probing uses the previous estimate of the available capacity as the RTT-long average for the sending rate but sending rates both lower and higher than the average sending rate are tried in the shorter term. That is, rather than sending the packets as evenly spread as possible like with pacing, intentional unevenness is introduced by the timer. As the time to send with a high sending rate is much shorter than RTT, the high sending rate looks just like a short burst to the network without building a large standing queue unlike the other two bandwidth probing mechanisms. Packet timing

measurements are then used to determine the sending rate that exceeded the rate of the bottleneck link. Only after the sender has received a positive confirmation from the bandwidth estimator that the capacity to sustain a particular sending rate is likely available it does truly load the network with that rate. [57] amends pathChirp-based bandwidth estimation with cross-layer information about the access link characteristics.

Packet timing measurements are challenging in general due to sensitivity to small measurement errors and noisy environments [199]. Some access link environments include behavior that breaks the assumptions made by the packet-timing approaches [113, 123]. A sender using bandwidth estimation should really ensure the validity of the estimation and if the validity is even in the slightest of doubt, fall back to safer self-clocked or paced bandwidth probing to avoid too high sending rates due to mismeasured available capacity. To ensure scalability to high-rate environments, the timer used for the packets in a probe must have sufficiently small granularity [117, 199]. Instead of easily measurable RTT, the mechanism may need to use a more challenging metric based on one-way delay or one-way delay variation. While those metrics are more robust to noise than RTT, measuring them occurs at the receiving end requiring additional signalling between the end hosts [117]. In addition, the sender should also avoid overshoots due to ramp-up collisions [33, 147].

2.4 Congestion and Response to Congestion Detection

Congestion occurs at the intermediate nodes of the end-to-end path in buffers that are used to queue the packets that are waiting to be transmitted. As the outgoing link where a packet is supposed to be sent has only limited capacity, it often occurs that the transmission of the packet may not begin immediately on arrival to the intermediate node because the previously arrived packets have not yet been transmitted fully. The waiting packets form a queue. The queue is managed by some algorithm. Traditionally a very simple Taildrop algorithm is used by most of the routers to manage the queue. With Taildrop, congestion is signalled only by dropping an incoming packet when the router runs out of buffer space for the packet. The sender interprets the loss as a signal about congestion somewhere along the end-to-end path and responds with a reduction in the sending rate. Later, it was realized that more control is necessary to keep some of the buffer unoccupied for short-term transients and thus Active Queue Management (AQM) was recommended [31]. An AQM algorithm

tries to keep long-term queuing low by pro-actively signalling congestion before the buffer space on the router runs out.

Pro-active congestion signalling that is enabled by AQM algorithms allows two options for handling a packet at a router. The router can either drop the incoming packet or it can preserve the packet but still signal congestion. When the router preserves the packet, it sends the congestion signal using an Explicit Congestion Notification (ECN) flag [156]. The ECN-based congestion signal is carried in-band. ECN allows the original packet to reach the receiver which may improve latency because the sender does not need to later retransmit the payload that is lost if the original packet is dropped.

Once a congestion signal is sent by the router either using drop or marking with ECN, Slow Start is coming to its end. However, the congestion signal is sent in-band and it takes one full RTT until the effects of response to the congestion become visible at the router[3]. While Slow Start is about to finish, the last RTT of Slow Start sends roughly as many additional packets to the network as was sent during all of the earlier RTTs combined because of the exponential nature of Slow Start. This huge load spike is called Slow-Start overshoot. In order to successfully queue the packets during the last Slow-Start RTT, a larger buffer is needed. The load spike is obviously smaller if Slow Start is interrupted early, for example, due to insufficient buffer size or pro-active AQM response. However, the flow might then continue with an incorrect estimate about the available bandwidth and it may take a while before the sending rate slowly increases to the correct level using Congestion Avoidance.

When the TCP sender detects a congestion signal, it responds by reducing the sending rate. The reduction is called Multiplicative Decrease (MD). Factor of 0.5 is used for the MD [13], that is, the sending rate is halved when a congestion signal comes in. The sender performs up to one reduction per RTT. Use of 0.5 as the factor causes the resulting sending rate to match the sending rate before the Slow-Start overshoot began assuming the factor of two exponential growth during the bandwidth-probing phase. In order to not harm utilization, the buffer in front of the bottleneck link needs to be sized large enough to allow the transient with the doubled sending rate before the MD sending rate reduction takes effect. With one flow the needed buffer space is roughly the bandwidth-delay product of the end-to-end path [20, 189]. If there is always more than one flow over the bottleneck at any point of time and the losses are not synchronized between

[3]With drops the RTT actually begins only after a subsequent packet of the flow is not dropped by the router because TCP loss detection depends on duplicate ACKs

flows, a smaller buffer may be used [15]. If synchronization for congestion signals to all or many flows occurs, the net effect of the sending rate reductions is still a halved total sending rate and underutilization may occur if the buffer is too small.

Recent work proposes a sending rate reduction that is less than halving the sending rate. Alternative Backoff with ECN (ABE) [109–111] proposes to change the MD factor to 0.8 for ECN-marked congestion signals only, which results in a less aggressive sending rate reduction for flows using ECN. A similar MD factor but also for loss-based congestion signals has already been deployed to the Internet by the Linux Cubic [88, 159] implementation that is the default TCP variant used by off-the-shelf Linux kernel (its MD factor is roughly 0.7 [28]). Also Data Center TCP (DCTCP) [5, 24] that aims to replace TCP inside datacenters achieves its latency benefits largely by tweaking the sending rate reduction factor. DCTCP uses a modified ECN feedback definition allowing fine-grained congestion indication. DCTCP calculates the fraction of packets over RTT that indicated congestion to set the MD factor dynamically. Effectively, it often results in less than halving the sending rate when congestion occurs as the DCTCP sender detects that the congestion is not very heavy.

If the congestion is signalled using packet drops, the packet losses need to be recovered. The Slow-Start overshoot typically causes many drops if the router buffer is full or nearly full as happens with Taildrop. The loss recovery used to have a clear impact on performance because TCP Reno recovers only a single lost segment per window of data and requires an RTO if more packets are dropped. Also TCP NewReno [92] allows retransmitting only one lost segment per RTT. Selective ACK (SACK) [30, 130] typically allows all losses to be discovered and retransmitted within one or a few RTTs. Nowadays, wide penetration of SACK [115] makes the loss recovery much more efficient. Use of Retransmission timeout (RTO) [148] to recover the losses which initially was the only mechanism for loss recovery is used only as the last resort. If RTO expires, the sender assumes its sending rate estimate is compromised and falls back to using Slow Start. In RTO-triggered Slow Start, only one packet is allowed to be sent on the first RTT after the RTO which is typically a large reduction in sending rate. Therefore, RTO often has a larger impact on performance than the loss recovery using the other mechanisms. In case of genuine loss of many packets, Slow Start after RTO is also needed for restarting the ACK clock without huge bursts. The main focus of this thesis is not on the behavior during the loss recovery or on its effects but we cannot entirely avoid it

either as naturally a congestion signal from an AQM algorithm using losses results in the need to retransmit the lost payload on the TCP layer.

If deployed, pacing packets during the bandwidth-probing phase to remove transient queue spikes introduces additional challenge to congestion detection at the router. In case of perfect or near-perfect pacing, the traffic load grows but no transient queue forms until the link becomes saturated [3, 135, 164]. Therefore any queue length or delay-based AQM algorithm will not see any signs of congestion until the link is saturated. As no transient queue spike gives an early warning, the congestion signal and response will be delayed compared with self-clocked bandwidth probing which leads to a large Slow-Start overshoot.

2.5 Mechanisms for Exiting the Bandwidth-Probing Phase Before a Congestion Signal

Typically, the bandwidth-probing phase continues clearly beyond the saturation point of the bottleneck link because congestion signals are generated and delivered only later when the queue before the bottleneck link starts to build up rapidly. As the flow keeps increasing its sending rate exponentially, the continued bandwidth probing leads to the Slow-Start overshoot. The sender may optionally try to terminate the bandwidth-probing phase before it receives any congestion signals to avoid the overshoot. These mechanisms need to somehow estimate the rate of the bottleneck link and discontinue the bandwidth probing beyond that rate. Alternatively, they may infer that persistent queue is growing at the router in front of the bottleneck link. The latter approach, while better than nothing, is less interesting as it discontinues bandwidth probing too late because it first needs to build the persistent queue in order to then measure it. For the mechanisms that estimate the rate of the bottleneck link, additional difficulty is knowing the available capacity that may be less than the rate of the bottleneck link. As it is challenging to know the available capacity of the bottleneck link at the end hosts, many mechanisms only offer an upper bound for the sending rate rather than the actual sending rate that matches the current state of the network.

The simplest technique for estimating the rate of the bottleneck link is to use a static value that is manually configured. The most well-known approach for this is Limited Slow Start [76]. Unfortunately manual values that need to be inputted by end users are impractical and thus undeployable on large scale. That problem also sealed the fate of Limited Slow Start that was once included in the Linux kernel [91] but later removed from the

kernel [48] as hard to configure and unnecessary complexity in the code. Obviously the end user cannot be expected to update the value when the available capacity on the bottleneck link changes so a properly set value for Limited Slow Start provides only an upper bound for the sender.

If the point when the bandwidth-probing phase should be exited is known, it is also possible to somewhat limit the size of transient queuing by smoothing the sending rate. Instead of continuing according to Slow Start near the end of the bandwidth-probing phase, the growth of the sending rate is smoothly reduced when approaching the sending rate at which the sender will exit the bandwidth-probing phase. The growth rate reduction in such a technique is achieved at the cost of RTTs needed to reach the final sending rate. This kind of smoothing is used in Limited Slow Start [76] and Smooth Start [192]. A very recent take in this direction is among the requirements for TCP Prague [35, 37] in the form of "gradual exit from Slow Start" that depends on congestion feedback before the end-to-end path is saturated. To us gradual exit seems a futile attempt as the smooth exit alone cannot remove transient queue spikes but only reduce them. The cost in the number of RTTs and the still remaining transient queue spikes make deploying smooth exit approaches unattractive.

The second approach for determining the bottleneck link rate is based on the local link characteristics. In them, cross-layer information is used as an upper bound for the rate of the bottleneck link. However, if the link rate varies or the bottleneck link is not directly attached to the end host the value may be off by a very large margin. These techniques also provide no way to estimate the available capacity but instead use the nominal capacity which may be shared by more than a single flow. In addition, if the bottleneck link is at the other end of the path, signalling of the local link characteristics to the remote end needs to deployed.

Control Block Interdependence (CBI) [186] is a TCP mechanism where different flows to the same destination address share part of the TCP variables. Among them is the Slow-Start threshold (ssthresh) that is used by the sender to terminate Slow Start when its congestion window has reached it, which combined with RTT determine the used sending rate. CBI is problematic because connections to new peers typically open in bursts making ssthresh sharing occur too late for them unless there was a preceding connection to the same destination. Also, volatility of the available capacity near the network edge makes the shared ssthresh value often stale which is ironic as the value measured from the other connections reflect the available capacity to at least some extent. The gravest problem with CBI sharing for ssthresh, however, are carrier-grade middleboxes implementing network

address translation (NAT) as the same public IP address connects behind the scenes to a very large set of hosts with non-shared bottleneck links.

Quick-Start [78] or a similar negotiation mechanism could be used for setting the Slow-Start threshold based on the rate approved by Quick-Start (similar to the approach used in [168]). When using this approach, the sender Slow Starts only up to the rate matching the approved rate that hopefully roughly matches the available capacity on the bottleneck link. However, the accuracy of this approach is limited by the granularity of the rate request [78]. In contrast to ramping up the sending rate immediately as occurs with the normal Quick-Start procedure, setting only the Slow-Start threshold should be reasonably safe even if some of the routers did not understood the Quick-Start request.

HyStart [87] has two independent sender-side mechanisms for finding an exit point for Slow Start, one based on detecting build-up of persistent queue and the other on estimating the bottleneck link rate. HyStart is enabled by default in the Linux TCP implementation when the default Cubic [88, 159] congestion control is in use. As such, there is wide deployment experience already about it. This first mechanism of HyStart measures increase in the delay at the beginning of a Slow-Start round and if the increase is above a threshold, the sender assumes that a persistent queue is building up into the network and discontinues Slow Start. However, too low a threshold may give false signals if a link technology introduces delays of varying length that are unrelated to queuing. The problems with too early termination of Slow Start has resulted in some operators disabling HyStart [135] and there have been multiple issues due to the sensitivity of the HyStart delay mechanism [69, 89, 114]. It seems that there are no known good parameters for HyStart with the current link technologies [44] and a more relaxed parametrization would be needed [107, 144]. Lack of good parameters is not surprising. It is difficult, if not impossible, to find a good global delay threshold without compromising either low standing queue build-up or utilization depending on the link technology. Also, this mechanism is not related to the true link load as tightly as the second, bottleneck link rate estimation mechanism of HyStart.

The second mechanism in Hybrid Slow Start (HyStart) [87] detects when to exit from the bandwidth-probing phase using ACK timing measurements. The self-clocked bandwidth probing during Slow Start naturally produces packet trains which then trigger trains of ACKs. A sender using HyStart measures whether the time elapsed for the train of ACKs for the current RTT is longer than half of RTT, which is used as the proxy for half of the capacity. Due to exponential sending rate growth, the bottleneck

link will become saturated on the next RTT following the half-RTT-long ACK train. Therefore, HyStart discontinues the bandwidth-probing phase at that point with a rate roughly matching the rate of the bottleneck link. The sender assumes that the train of ACKs is spaced out by the rate of the bottleneck link. The spacing at the bottleneck link is typically guaranteed as long as the sender sent the packets on the previous RTT with at least the bottleneck link rate. If the ACK clock retains at least roughly the rate of the bottleneck link, the sender assumption is valid, however, some link technologies may introduce systematic discrepancies to the ACK clock. HyStart has been shown to miscalculate the exit point when the delay varies due to other than congestion-caused sources [43, 114, 144, 161] as HyStart assumes that any delay in ACK timing is due to congestion. These noise sources independent of the current load are, in general, challenges to any ACK timing measurement-based bandwidth estimators.

Packet timing measurements, in general, work better when done over long packet trains rather than for individual samples [128]. As the second mechanism in HyStart uses a long train of packets to trigger the ACKs for the timing measurements, the errors in individual samples are hidden better but it also imposes a requirement to create large queue transients for the mechanism to work properly. Unfortunately, pacing packets evenly and sending such a long train of packets are at odds with each other and cannot be deployed at the same point of time by the same sender. For the bottleneck link rate estimation to work, the sender needs to send with at least the bottleneck link rate to make sure the ACK spacing will reveal the bottleneck link rate. Sending at higher than the bottleneck link rate creates a queue before the bottleneck link and as the packet train needs to be long enough, significant queue build-up is to be expected during ACK train measurements. While the queue may be transient, it is forming such a queue that pacing packets evenly tries to avoid. As such, pacing and reliable ACK timing measurements are inherently incompatible with each other. Because of the incompatibility problem with pacing, the entire ACK train-based mechanism in HyStart is disabled by some operators [44]. Therefore, to achieve low queuing latency, we believe sender-side ACK-train-based bandwidth estimation such as in HyStart cannot be used because pacing is required to remove transient queue spikes.

To work around the problems pacing evenly introduces to determining the exit point, [135] suggests using chirping and determining the exit point with mechanisms similar to bandwidth estimation algorithms. With chirps, the frequency in sending a train of bandwidth probe packets increases rapidly while the number of packets in the train can be kept small

by using multiple chirps that are spaced apart in time. A small chirp causes only some queuing but no high queue build-up that would come with the inherent delay increase. But as noted above, all these packet-timing techniques work better with longer trains of packets. Limiting the size of the train to small, while good for delay, will likely in the general Internet run into similar issues in defining a good global threshold for the delay as the first mechanism in HyStart did. To avoid measuring noise instead of the actual signal, in the global deployment scenario the packet train size must be long enough to overpower noise in any link technology. Thus, the delay cost is likely to become prohibitive as it is likely impractical to expect that the noise level of the current path could be acquired so a high global default value must be used instead.

Delay-based congestion control algorithms are a full class of algorithms that attempt to terminate the bandwidth-probing phase based on increase in the persistent queuing. Like the first mechanism in HyStart, these mechanisms are based on measuring persistent queue and do not aim to find the right exit point but only that the bandwidth probing is already clearly past the correct exit point. That is, in order to build a measurable persistent queue, the sending rate must already clearly exceed the rate of the bottleneck link. Delay-based algorithms include for example TCP Vegas [32, 33] and FAST TCP [106, 193]. The FAST TCP window function is based directly on RTT, whereas TCP Vegas uses delay indirectly by estimating the actual sending rate that will stop growing once the persistent queue is building up as RTT increases together with the bytes transmitted over that RTT. Delay-based congestion control, in general, has starvation issues when operated together with flows that use loss-based congestion control [137]. A few other delay-based congestion estimators are listed in [158].

As an alternative to the sender-side exit from the bandwidth-probing phase, the TCP receiver advertized window or ECN marking by the receiver may be used to limit sending rate growth beyond the estimated rate of the bottleneck link. The main difference is that most calculation can then be done by the receiving host, however, the mechanisms themselves have to be based on similar fundamental principles as with the sender-side approaches explained above.

To recap the approaches listed above and issues with them if deployed to the general Internet: Most approaches likely have unsatisfactory accuracy for operation in the general Internet or are impractical to operate. The HyStart algorithm, on the other hand, is well-tested in the general Internet. Its ACK train mechanism often has acceptable performance and does not require end user input for operation which has allowed it to be accepted and

remain in the mainline Linux kernel this far but it is also known to have some issues in handling noise. However, ACK train-based approaches are inherently incompatible with pacing packets evenly which may mean not so bright a future even for the HyStart one because there is some interest in enabling pacing to smooth out burstiness that otherwise occurs during the bandwidth-probing phase.

An entirely different approach for exiting the bandwidth-probing phase at the right point of time would be to use explicit feedback from the router in front of the bottleneck link. Such a signal itself is similar to a congestion signal, however, the signal needs to be sent slightly before actual congestion takes place to reach the sender in time. Unfortunately, no solution currently exists for a router to provide timely feedback about soon to be occurring congestion and such a router algorithm is the research question RQ7 in this thesis. To eliminate the transient queuing during the bandwidth-probing phase, however, the algorithm on the router needs not only to solve RQ7 but also has to be compatible with pacing. Effectively, the congestion signal must be sent before the saturation of the bottleneck link when, because of pacing, there is not yet a queue present.

2.6 On Restarting Flows After Low-Activity Periods

When a TCP connection becomes idle or is sending with a much lower rate than allowed by the congestion control because of an application limit, the sending rate estimate of the congestion control becomes compromised. As the sending rate is not "tested" fully by sending at that rate, resuming transmission suddenly with the compromised sending rate may be problematic. Onset of congestion that would occur and be detected if the full sending rate would have been used may not be noticed by the sender because the network is less loaded during the low activity period. When the sender then resumes with the full rate, the network immediately experiences a sudden load spike that may cause excessive latency to occur. Many traffic types are suspectable to these low activity periods such as HTTP persistent connections during OFF periods, HTTP adaptive streaming (HAS) when there is a stable scene without much frame-to-frame variation, etc. According to measurements internal idle periods alone occur at least in one fourth of all connections [165].

As the outstanding data packets are few or completely drained from the network during the low activity periods, the ACK clock also has less incoming ACKs to trigger sending new data packets. In order to restart the

flow safely, Slow Start from the Restart Window that is the minimum of the Initial Window or the current congestion window is recommended after an idle period longer than RTO [13]. The Congestion Window Validation (CWV) [90] algorithm improves the decay of the congestion window to be less dramatic. As time passes on from the last validated sending rate sample, CWV gradually reduces the allowed rate with which the transmission can be resumed. Once the transmission is resumed, Slow Start [13] is used to quickly increase the sending rate up to the higher sending rate that was used earlier. The use of Slow Start allows feedback from the network to occur before the sender accelerates to the earlier, higher sending rate in case the network has become congested in the meantime.

Slow Start, and especially the gradual reduction of the sending rate all the way back to the Initial Window if the low activity period is long, has made some senders unhappy about the CWV performance. In order to remove the need for hacks around the old CWV, a new CWV algorithm was specified [70]. The new CWV is much more aggressive compared with the old CWV. Instead of Slow Start, the new CWV uses pacing to send immediately with the original rate and only validates whether there was capacity in the network for that rate afterwards.

The load increase due to the new CWV is comparable to extreme violations of the packet conservation principle similar to what a very large Initial Window are causing. The new CWV hopes opportunistically that statistical multiplexing covers for the induced bursts [70]. If no other traffic has occupied the bottleneck link in the meantime, the sender is lucky and can continue with the sending rate that has now been tested. However, if competing traffic has appeared, it will be harmed before the sender can even know about it like with a large Initial Window. In addition, the use of pacing will not remove the violation of the packet conservation principle that increases the queuing delay if the bottleneck link becomes fully utilized. In fact, in this case pacing is more a solution to self-congestion than to protect other users of the network and the included responsiveness comes too late. Whether one or a few RTTs worth of harm is acceptable to competing traffic can of course be debated but this new CWV is by no means conservative in sending that has traditionally been one of the key guidelines in operating a network endpoint.

With the new CWV, the sending rate is held for up to a Non-validated Period (NVP) that is chosen to be up to five minutes [70]. Five minutes may be enough for path stability, however, on links near the network edge it is ages from the congestion control point of view [135]. That is, the congestion level on an edge link or nearby it is hardly stable that long (unless the users

are inactive, of course). Even the new CWV proposal itself admits that "the opportunity for congestion-free statistical multiplexing" is "reduced" [70] with the hacks around the old CWV implying there are other load sources with which the restarting source may collide. To us this seems just to be a circumvention of Slow Start by replacing it with an extreme Initial Window similar to what is proposed in [9]. The only difference is a more valid estimate about the maximal capacity of the end-to-end path but as with the Initial Window, there is no recent enough information about the currently available capacity. Therefore similar considerations apply as to the Initial Window in Section 2.2.

2.7 Summary

Congestion control is traditionally based on TCP Congestion Avoidance and operating in steady state. However, the explosion of web usage caused short, non-steady state flows to increase in number and to become a very significant contributor to congestion. Early, often still in-use HTTP versions depend on parallel flows to reduce web transaction latency which led to a large default setting for the number of parallel flows allowed by web browsers. As the parallel flows in a web transaction are closely packed in time, in the worst case starting exactly at the same point of time, the Initial Window of the flows may cause significant congestion build-up. Pub. I measures the effect that parallel flows and the proposed increase of Initial Window to ten TCP segments have on competing latency-sensitive traffic. It shows that with one or two flows, the impact of competing TCP flows on delay is small when using Initial Window of three segments. If the number of flows is higher or Initial Window of ten is used regardless of the number of flows, the delay likely reduces the quality of the competing interactive media flow.

During flow startup, a bandwidth-probing phase is used by the sender to find out the sending rate it should use later in the steady state. During the bandwidth-probing phase, TCP uses Slow Start that often causes exponential load transients to occur at the bottleneck router. The exponential load transients have a significant impact especially close to the network edge where the bottleneck typically is and where the traffic aggregation level is less than in the core network routers. As web traffic has ON and OFF periods, these exponential load transients occur frequently and lead to rapid load fluctuations between idle or low utilization and overload.

HTTP adaptive streaming (HAS) is one of the recent trends on the Internet and is already a significant portion of all traffic on the Internet. HAS

runs on top of HTTP/TCP and has been shown also to cause transients on the network level even if the playback itself is continuous. As it runs on top of TCP, TCP Slow Start is very much a factor during such transients.

TCP Slow Start causes transient queue spikes at the router in front the bottleneck link because bandwidth probing based solely on the ACK clock cannot achieve a smooth sending rate. This limit is inherent to any self-clocked sending rate increase. The only way around this limitation is to use external triggers outside of ACK clock ticks, that is, to pace packets out using a timer. Pacing during the bandwidth-probing phase allows packet sending times to be spread to inactivity parts of RTT which avoids queue build-up before the bottleneck link. In the ideal case, no queue is formed until the TCP sending rate matches the rate of the bottleneck link. Thus, pacing may be used to solve the transient queue spikes associated to self-clocked TCP Slow Start but the bandwidth-probing phase must then be terminated very precisely to avoid building a persistent queue once the bottleneck link is saturated. However, precise exit from the bandwidth-probing phase is challenging as the ACK train-based mechanisms are incompatible with pacing, while the other mechanisms are inaccurate or impractical. A precise exit from the bandwidth-probing phase based on a completely new approach that would be fully compatible with pacing is the ultimate research question RQ7 in this thesis.

Chapter 3

Active Queue Management During Flow Startup

Active Queue Management (AQM) is an algorithm that runs on an Internet router to manage its queue before the router is forced to drop packets as it runs out of buffer space. AQM has long been recommended to be deployed by the Internet routers [31]. In practice, the recommendation has had little effect and if AQM capability exists in the actual router hardware, it tends to be disabled. Recently, however, new interest has revived the AQM discussion and sped up the deployment of the AQM algorithms. The AQM working group in IETF [2] and its update on the earlier AQM recommendations [19] are just a few fruits of the effort. As such, AQM is very timely and relevant today.

As a part of the recent AQM efforts, a bufferbloat problem [84] has been identified. Bufferbloat refers to oversized, unmanaged buffers in the routers which typically yield no improvement in performance but only tend to increase latency when the buffers fill. Many measurements on how bad problems the bufferbloat causes to real networks have been made [4, 7, 67, 84, 105, 183]. Several new AQM algorithms that could be used to combat the bufferbloat have been proposed [142, 146] and the IETF AQM working group has been formed to come up with specifications for the algorithms [94, 143, 145].

Unfortunately most of the AQM interest still ignores the issues that occur because of exponential load transients and focus only on steady-state behavior [93]. Yet, several studies point out that the Internet traffic exhibit ON-OFF characteristics which implies that load transients are quite a common occurrence. In this chapter, we focus solely on exponential load transients during flow startup and their effect on AQM performance.

Section 3.1 introduces the current state of the AQM art. Section 3.2 explains the AQM behavior during exponential load transients, which we analyzed in Pub. II, Pub. III, and Pub. IV. It also explains how current AQM algorithms are inadequately solving the issues during exponential load transients; this relates to the research questions RQ3 and RQ4. In Section 3.3 we cover Pub. II that aims to push a state-of-the-art AQM algorithm as far as we could to solve the issues with exponential load transients. As the solution in Pub. II turns inadequate for solving the general case, we next discuss in Section 3.4 a completely new AQM algorithm covering Pub. IV to truly solve the issues identified in Section 3.2. In Section 3.5 we apply the insights our AQM algorithms to handle exponential bandwidth probing have given us also on flow initiation and restart.

3.1 State of the AQM Art

Router Governed AQM Algorithms

Random Early Detection (RED) [80] is a decades old AQM algorithm that tracks average queue length to allow short-term bursts to pass through without reaction while responding to long-term congestion. RED updates the average queue length when a packet arrives. The average queue length (avg) is calculated from an instantaneous queue length (q_{len}) using exponentially weighted moving average (EWMA) with a weight (w_q) as follows:

$$avg = (1 - w_q)avg + w_q \cdot q_{len} \tag{1}$$

Figure 3.1 shows how RED compares the average queue length to two thresholds to decide on which operating region the current congestion level is: (a) when below the minimum threshold (th_{min}), RED does not drop any traffic, (b) when the queue average is between the minimum and maximum (th_{max}) thresholds, RED performs probabilistic dropping, and (c) when the queue average is above the maximum threshold RED invokes a safety valve that causes it to drop all traffic. The main operating region between th_{min} and th_{max} with probabilistic dropping calculates an initial dropping probability ($p_{initial}$) as a linearly increasing function of the average queue length:

$$p_{initial} = p_{max} \cdot \frac{avg - th_{min}}{th_{max} - th_{min}} \tag{2}$$

The slope of the linear function is controlled by the maximum dropping probability (p_{max}) variable. A second stage to calculate final drop

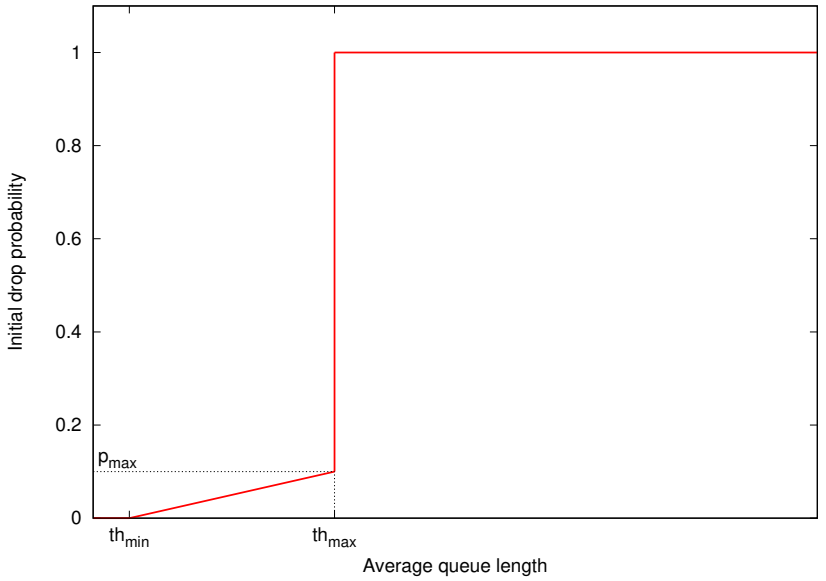


Figure 3.1: The initial dropping probability with RED.

probability (p_{final}) aims to produce a desired distribution for the drops. Typically, a uniform distribution is chosen. With it, RED calculates the final dropping probability from the initial dropping probability as follows:

$$p_{final} = \frac{p_{initial}}{1 - count * p_{initial}} \tag{3}$$

The *count* variable indicates how many packets have arrived on the router since the average queue length exceeded the minimum threshold or since the last drop. That is, *count* is reset on each entry to the main operating region and on each drop made by it. The above second-stage formula ensures that the final dropping probability rapidly approaches one as *count* approaches $1/p_{initial} - 1$.

RED calculations typically take place when packets arrive at the router. After an idle period on the outgoing link, an additional formula is needed to reduce the average queue length by feeding the queue average with phantom packets. The size of the phantom packets may be based on assumed or measured average size of the real packets to approximate how many packets could have been sent during the idle period. The idle time is accounted according to the formula:

$$avg = (1 - w_q)^{\frac{(time_{now} - time_{idle\_start})}{xmittime_{phantom}}} \cdot avg \tag{4}$$

Variables $time_{now}$ and $time_{idle\_start}$ indicate the current time and the time when the outgoing link became idle, respectively, and $xmittime_{phantom}$ is the assumed transmission time of a phantom packet.

The RED algorithm is already more than two decades old and there are multiple variations to the basic algorithm. RED and some of its variants have been deployed on many Internet routers, however, they are rarely enabled. Next we will look at the most notable RED variants.

Weighted RED (WRED) [51] allows differentiation according to a traffic class by having separate RED thresholds and weight for each class. By configuring a traffic class with larger thresholds, WRED allows that traffic class to be prioritized over the other traffic classes.

Adaptive RED (ARED) [71, 72, 82] dynamically adjusts the maximum dropping probability in order to adapt traffic load variations. ARED periodically observes the average queue length. If the average queue length is outside of the target region, the maximum dropping probability is adjusted if possible by its allowed range. This makes ARED more aggressive when congestion persists long enough for the maximum dropping probability to grow. While we have not run any tests with ARED in the experiments done in this thesis, the periodic readjustments seem to occur too infrequently to have any meaningful impact during exponential load transients. The target region definition varies between ARED proposals. In [71, 72], the target region is defined as the full region between the minimum and maximum threshold, whereas in [82] the target region is only a small part in the middle of the main operating region between the minimum and maximum thresholds:

$$target = [0.4 \cdot (th_{max} - th_{min}) + th_{min}, 0.6 \cdot (th_{max} - th_{min}) + th_{min}] \quad (5)$$

Gentle RED (GRED) [162] splits the linear function for the initial dropping probability into two parts. First the slope for the initial dropping probability is similar to that of RED and is typically kept small. Once the queue average is beyond the maximum threshold, the dropping probability grows linearly towards dropping probability of one that is reached at twice the maximum threshold.

Figure 3.2 summarizes the differences in calculation of the initial dropping probability for RED, WRED (with two traffic classes), GRED, and ARED. Other variations with different kinds of functions for the initial dropping probability have also been proposed [40, 98, 150].

PIE (Proportional Integral controller Enhanced) [145, 146] is based on PI (Proportional Integral controller) [96, 97]. To improve stability and

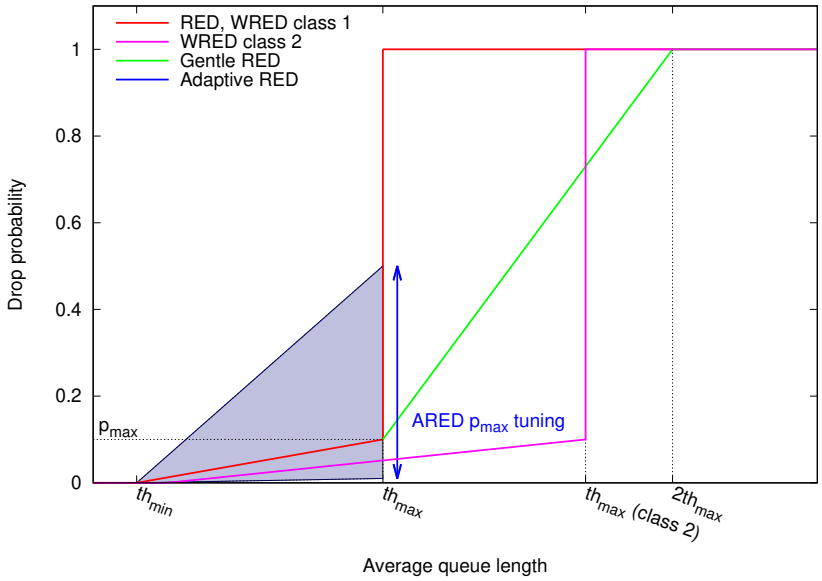


Figure 3.2: Drop probabilities with different RED variants.

response speed, PIE adds auto-scaling for PI control law and also solves some implementation challenges related to queuing delay estimation. In addition, PIE adds various conditions that result in non-continuous response instead of using the plain PI formula [61]. The most prominent feature with non-continuous response is burst allowance quota that is optional but enabled with default parameters [145]. PIE is mandatory for DOCSIS 3.1 compliant cable modems [194–196].

PIE tracks queuing delay and the trend of the queuing delay aiming to keep the queuing delay at the target delay ($delay_{target}$). At the end of each interval, PIE updates drop probability that is applied for the duration of the next interval on incoming packets. The new drop probability (p) is calculated according to equations:

$$\Delta_p = \alpha(delay_{now} - delay_{target}) + \beta(delay_{now} - delay_{prev}) \quad (6)$$

$$p = p + \Delta_p \quad (7)$$

$delay_{now}$ and $delay_{prev}$ are the current delay and the delay on the previous invocation of the drop probability calculation, respectively, and α and β are control parameters. PIE tracks not only the current queuing delay but also the trend and is able to remove steady-state error that is inherent to averaging only controllers. PIE has an auto-scaling element to control

the aggressiveness of PI control law and to ensure the stability of the algorithm. The auto-scaling alters the control parameters α and β, however, it can be implemented by scaling the delta term of the drop probability (Δ_p) in Equation 7 as follows:

$$p = p + scaling\_factor \cdot \Delta_p \tag{8}$$

The *scaling_factor* is derived from the previous drop probability as shown in Table 3.1.

Drop probability	Scaling factor
$p < 0.000001$	1/2048
$0.000001 \leq p < 0.00001$	1/512
$0.00001 \leq p < 0.0001$	1/128
$0.0001 \leq p < 0.001$	1/32
$0.001 \leq p < 0.01$	1/8
$0.01 \leq p < 0.1$	1/2
$0.1 \leq p$	1

Table 3.1: PIE drop probability auto-scaling [145].

PI^2 [61, 62] replaces the drop probability scaling table with an automatic adjustment to the right sensitivity level by squaring a PI controlled pseudo-probability to obtain the final drop probability. Effectively, PI^2 calculates the pseudo-probability in a domain linear to load but then adjusts the result by the square to take into account the throughput response of TCP Congestion Avoidance that is proportional to the square root of the drop probability [131]. In addition, PI^2 simplifies PIE by removing the non-continuous elements.

MADPIE (Maximum and Average queuing Delay with PIE) [118] attempts to mitigate the oscillations PIE experiences with large RTTs. MADPIE modifies PIE by adding deterministic dropping instead of random drop. MADPIE controls large queuing delay spikes better than PIE by avoiding alternation of full and empty queue period better than with PIE.

P^2I [200] is a two-state AQM algorithm. PI controller is used for the steady state and a proportional controller to handle load transients. The proportional controller derives the final drop probability from an instantaneous queue length without any averaging or filtering. Instead, the instantaneous queue length is directly multiplying with a drop probability factor making the resulting drop probability to increase whenever the queue length increases. The proportional controller is shown to improve transient response time over a PI controller.

CoDel (Controlled Delay) [142, 143] is a three-state AQM algorithm to address a long-standing queue that is termed "bad queue" by CoDel. CoDel maintains sojourn time (i.e., queuing delay) for each packet by timestamping them at the enqueue time of the packet and calculating the time elapsed when dequeuing the packet. The long-standing queue is detected when the sojourn time remains above the target delay (5 msecs by default) for at least an interval (100 msecs by default).

CoDel selects its state based on the measured sojourn times. The three states of CoDel are as follows 1) non-dropping state when the queuing delay is below the target delay, 2) pre-dropping state while the sojourn times are above the target delay but less than one interval has elapsed, and 3) the dropping state where the drops are scheduled with increasing frequency. Whenever the sojourn time falls below the target delay, CoDel immediately returns into the non-dropping state. When CoDel measures a sojourn time above the target in non-dropping state, it enters to the pre-dropping state and the switch to the dropping state is scheduled to occur after one interval. At the entry to the dropping state, CoDel drops one packet and initializes a *count* variable that indicates how many drops have occurred since the entry to the dropping state. CoDel then schedules the next drops by dividing the interval by the square root of the *count*:

$$time\_to\_next\_drop = \frac{interval}{\sqrt{count}} \tag{9}$$

As the *interval* is constant and the *count* is incremented by one after each drop, the *time_to_next_drop* series is not probabilistic but deadline based.

The entry to the dropping state in CoDel may trigger a count recall instead of resetting the *count* to one if the dropping state was recently active. The count recall has many variants and to our best knowledge, there has not been a comprehensive study on the performance with different variants of the count recall in CoDel. We next describe the approach used in [143]. The recall is called if the previous entry to the dropping state was within 16 intervals from the previous visit into the dropping state [1]. At recall, CoDel calculates the *count* as the delta between the *count* at the entry and exit from the previous visit to the dropping state (i.e., the number of drops made minus one). In addition, the recall must always result in a *count* of at least one.

[1] The previous dropping state is considered to be terminated at the point when it would have scheduled the next drop which never took place as sojourn time fell below the target.

CoDel, while specified for single queue use, is recommended to be deployed with fair queuing [103]. A router may use a fair-queuing mechanism to provide flow isolation. Technically fair queuing can be used orthogonally to an AQM algorithm but its presence will affect packet dynamics in significant ways.

Stochastic Fair Queuing (SFQ) [133] provides flow isolation by hashing each packet into one of the queues. The hash should place packets of a single flow in the same queue. A round-robin algorithm is then used to dequeue from those queues. As a result, the outgoing link is shared between active flows such that each will get equal share [2]. The hashing, however, may be subject to collisions resulting in putting more than one flow into the same queue. If a collision occurs, the equal share is not allocated for a flow but shared between the flows in that same queue. To prevent collisions from persisting for a long time, the hash is frequently perturbed [133] but that increases complexity. Another approach is to partition queues for n-way set associativity [55, 95] which virtually eliminates the hash collisions.

FQ-CoDel (FlowQueue-CoDel) [94] is a specifically tailored approach for CoDel to provide flow isolation. By default, FQ-CoDel provides 1024 queues each with an individual CoDel instance and state variables. When a packet arrives, it is classified into one queue using a hash. The queues are scheduled by a modified Deficit Round Robin (DRR) [175] scheduler. The DRR modifications allow FQ-CoDel to distinguish sparse flows that constantly transmit no more than their DRR quantum. Such flows are given priority over the flows that keep building the queue. The prioritization is achieved using *new* and *old* lists for queues storing currently active sparse and queue-building flows, respectively. The empty queues that are currently inactive are in neither of the lists. When a packet arrives on an inactive queue, the queue is added to the *new* list. To prevent starvation of the flows in the *old* list, whenever a queue is visited in the *new* list by the scheduler, it is moved to the tail of the *old* list even if it became empty. If a queue becomes empty during dequeue while it is in the *old* list, it is removed from the *old* list and becomes inactive. The aim of the modification is similar to the contract of DRR++ [132].

AQM algorithm cannot measure the current load of the router directly but a proxy variable must be used. As a keen reader may have noticed, one noteworthy feature of these AQM algorithms is that they tend to use queue length or a dual of it, queuing delay, as the proxy for load esti-

[2]Some call the equal share the fair share, however, usually with lots of speculation that "fair" may mean also something else. Therefore we prefer to refer it only as equal share which does not carry such philosophical burden.

mation. Queue length, however, provides a robust signal only when the bottleneck link is saturated. If the bottleneck link is not saturated, the signal tends to somewhat correlate to the load growth only if the sender is sending with a rate higher than the bottleneck link rate. With self-clocked bandwidth probing the sending rate is inherently higher due to transient queuing. Sending rates lower than the bottleneck link rate, however, cause no queuing and therefore no early signal from the proxy about the onset of congestion [3, 135, 164]. Thus, if pacing is used during the bandwidth-probing phase, queue as the proxy for load estimation gives no warning to the AQM algorithm until the link becomes saturated. As such, enabling pacing during the bandwidth-probing phase is a two-edged sword to many AQM algorithms. While intuitively it would seem that removal of the queue and therefore latency would be all good, the "too good" pacing removes the early transient queue artifacts that would benefit many AQM algorithms that include too simplistic queue length-based load estimation logic. As a result, the onset of congestion after the link becomes saturated takes the AQM algorithms by surprise. Hence, the congestion at the end of the bandwidth-probing phase will be even heavier than without paced bandwidth probing. To mitigate this problem, it would be desirable for the load estimation in AQM algorithms to not depend on the transient queuing during self-clocked bandwidth probing.

Explicit Congestion Notification with AQM Algorithms

Modern AQM algorithms aim to keep queuing delay at a low level and if the AQM algorithm is operating correctly, it should not normally run out of buffer space. Having a less than full buffer opens up a new possibility for a router as it may opt to send a congestion signal by marking a packet using Explicit Congestion Notification (ECN) [156] instead of the traditional drop. With full buffers, the router would not have had such an option as it lacks buffer space for the incoming packet.

The ECN signal is transmitted in-band when a router marks the ECN field of the IP header [156] with the Congestion Experience (CE) value. The receiver then relays the CE indications back to the sender so that the sender can adjust its sending rate. The feedback channel for CE indications is transport protocol dependent. With TCP the feedback is achieved by asserting the ECN-Echo (ECE) bit in the TCP header for all ACKs until an incoming segment with the Congestion Window Reduced (CWR) bit arrives. The sender sets the CWR bit for the next new packet it sends after it has performed a reduction to the sending rate regardless of the reason for the reduction.

The router may set the CE value to the ECN field only for those packets that do indicate ECN capability by having the ECN field set to either ECT(0) or ECT(1). If the ECN field already contains CE, marking the packet again simply retains the value as CE. If the packet became marked with CE before it was lost, the reliable loss detection ensures that the sender will reduce its congestion window even if CE never reached the receiver. Likewise, the CWR bit is set by the sender even if the window reduction was triggered because of a packet loss rather than CE marking to stop the receiver from sending more ECE-flagged ACKs. This way of signalling ensures that the sender performs one congestion window reduction for each RTT with a CE mark or loss but should not reduce the congestion window more than once per RTT.

If the packet contains the Non-ECT value in the ECN field, a router must drop it even if the router itself supports marking using ECN. The ECN specification [156] requires using Non-ECT for TCP retransmissions and control packets as their losses cannot be reliably detected by the TCP sender to ensure the congestion window is reliably reduced on a packet loss. The specification also recommends against setting the CWR bit in a retransmission. There is currently ongoing effort to relax these requirements [18, 38].

Most modern AQM algorithms discussed in Section 3.1 can be configured by the operator to use either drops or ECN marks for congestion signalling. In such setups, ECN is used in the standardized form [156]. In addition, there are a few more advanced congestion control architecture proposals that redefine the meaning of the ECN signals requiring changes to both the end hosts and the network routers.

Datacenter TCP (DCTCP) [5, 24] is an AQM approach combining a router AQM algorithm, modified ECN signalling, and end-host modifications to the TCP implementation. A DCTCP-aware router runs the RED algorithm in a degenerated mode with the queue averaging exponent set to one, that is, the average queue length equals the instantaneous queue length. In addition, the thresholds are set to the same value causing RED to implement a step function to mark packets with CE when the queue length is above the set threshold. Whereas an ordinary ECN receiver keeps sending ACKs with the ECE flag throughout the whole RTT until a CWR-flagged TCP segment arrives, a DCTCP receiver echoes each received CE indication using a single ECE-flagged ACK.

As CE marks from the DCTCP AQM controller are more frequent than from an ordinary, probabilistic AQM controller, a DCTCP sender responds to the ECE-flagged ACKs proportionally to the severity of the perceived

congestion rather than by a constant reduction to the sending rate. For each window of data, a DCTCP sender calculates the fraction of packets (F) that were congestion marked. An EWMA with weight g is then applied on it after each window of data to produce an averaged fraction of marked packets (α):

$$\alpha = (1 - g)\alpha + g \cdot F \tag{10}$$

The DCTCP congestion window (*cwnd*) is adjusted according to the severity of congestion using the averaged fraction of marked packets as follows:

$$cwnd = \left(1 - \frac{\alpha}{2}\right) cwnd \tag{11}$$

If α is small, only a small correction is applied to the congestion window, whereas with larger α the congestion window is more rapidly reduced. With the maximum α of one, the DCTCP congestion window reduction is equal to that of TCP with the MD factor of 0.5. The DCTCP *cwnd* formula is only applied during Congestion Avoidance and no modification is proposed to TCP Slow Start that is kept identical to standard TCP. As such, the end-host DCTCP logic will not play a significant role during an exponential load transient that relates to flow startup using Slow Start. The AQM component in DCTCP, however, is based on pure instantaneous queue length that is subject to transient queue spikes during the bandwidth-probing phase. The spikes, despite being only transient, may lead to crossing the DCTCP AQM step threshold resulting in a premature exit from Slow Start with a small congestion window, which causes suboptimal flow performance.

DCTCP is designed for well-controlled datacenter environments. Its main goal is to keep queuing persistently on a low level to reserve most of the buffer space for a burst of short flows [5]. DCTCP is incompatible with standard TCP congestion control and therefore cannot be readily deployed safely in the general Internet. There is ongoing work to construct a slightly modified DCTCP variant with a similar scalable congestion control but such that the obstacles to Internet-wide deployment are removed [35–37, 58–60, 116]. It is a multifaceted problem requiring congestion control advancements both in AQM and end-host TCP algorithm fronts.

DCTCP [5, 24] uses a fine-grained ECN signalling but the feedback channel is vulnerable to ACK losses [116]. Accurate ECN [38] specifies a robust feedback alternative for such fine-grained ECN signals.

Coupled Dual Queue AQM (DualQ) [58, 60] attempts to enable Internet deployment of a slightly modified DCTCP variant by segregating standard TCP congestion control and DCTCP traffic into their own queues.

The traffic type can be selected using Low Latency, Low Loss, Scalable throughput (L4S) [36] service identifier. The queue for scalable traffic has a strict priority but the probability for congestion signal is coupled between the queues to avoid starvation of the classic queue. The coupling ensures that the higher priority scalable queue is often enough without a packet giving an opportunity for the scheduler to dequeue a packet from the classic queue. Coupling between the drop probabilities is calculated according to the formula:

$$p_{classic} = \left(\frac{p_{scalable}}{k}\right)^2 \tag{12}$$

Where $p_{classic}$ and $p_{scalable}$ are the drop probabilities for the classic and scalable queues, respectively, and constant k determines how the achieved rates are coupled with equal RTTs [3].

Exponential load transients due to flow startup are known to cause premature exits from the bandwidth-probing phase with DCTCP [58]. Two main directions to work around this problem have been suggested: 1) gradual exit from the bandwidth-probing phase [35, 37, 58] and 2) bandwidth estimation approaches with quick sending rate ramp-up [59, 135, 136]. The third alternative could, instead of noise challenged bandwidth estimation, use pacing together with a timely exit from the bandwidth-probing phase. The timely exit avoids the overshoot a too late AQM response would cause. This third alternative requires finding the correct exit point which in turn requires a pacing compatible AQM algorithm that is aware of the current load during flow startup-induced exponential load transients. Such an AQM algorithm is a focus area of this thesis.

High-bandwidth Ultra-Low Latency (HULL) [6] combines sender-side pacing, DCTCP, and Phantom Queues (PQ) that are very similar to Virtual Queue AQMs [86, 120]. PQ limits the bottleneck link utilization by sending congestion feedback using ECN the way that matches a slower link. Sender-side pacing is employed to smooth out burstiness that easily triggers spurious congestion feedback from the instantaneous queue-based threshold used in PQ as with DCTCP in general. Sender-side pacing is selectively employed to avoid introducing latency to short flows. Only after enough congestion feedback from the AQM algorithm has arrived for a particular flow, pacing is applied to its packets. However, the pacer module of HULL architecture selects the flows to be paced solely based on congestion feedback observed on the flows themselves. Therefore, the pacing will not play any role

[3] To achieve an equal rate with homogeneous RTTs, k must be selected based on the congestion control used by the non-scalable traffic, see [58] for more details.

whatsoever during the bandwidth-probing phase as a DCTCP sender, like an ordinary TCP sender, terminates the bandwidth-probing phase when the first congestion indication arrives. As the bandwidth-probing phase is not paced by a HULL sender, it cannot mitigate queue transients during the bandwidth-probing phase and results in "triggering spurious congestion signals, leading to reduced throughput" [6][4].

3.2 Analyzing AQM Behavior During Load Transients

TCP Slow Start [13, 102] forms a big challenge to AQM algorithms because of its exponential nature. The exponential increase in the sending rate during Slow Start may increase also the load on a router exponentially. As the load grows exponentially, the transient from idle or low load to overload takes only a few RTTs. In the research question RQ3 we want to know how the existing AQM algorithms cope when they are subject to exponential load transients and in the research question RQ4 we seek to locate what are the main issues AQM algorithms encounter when handling exponential load transients. To answer RQ3, we show in Pub. II, Pub. III, and Pub. IV that all state-of-the-art AQM algorithms with default parametrization are not properly responding to rapidly occurring exponential load transients. This failure to work properly during transients is also confirmed by others [93].

In Pub. II we show how the RED algorithm with default parameters [77] leads to two load transient-related issues. First, a Taildrop fallback takes place because the queue average and drop probability rise too slowly for RED to respond pro-actively while there is still room in the buffer. As a result, the physical buffer space runs out and the router has no other choice but to reactively drop packets. Once the RED algorithm finally "pro-actively" reacts, it is too late and it drops retransmitted packets, which cannot be resent by TCP before RTO expires. The second issue with RED is a maximum threshold cutter that is even worse than the Taildrop fallback because it activates the safety valve of RED[5]. Even if there is room in the physical buffer, the safety valve forces the router to drop all packets until the queue average slowly falls back below the maximum

[4] [6] labels queue transients that occur due to Slow Start as "TCP artifact", however, as discussed in Section 2.3, such queuing is inherent to any self-clocked bandwidth probing and requires timers, that is, pacing to mitigate.

[5] After completing Pub. II we have been made aware of a potential improvement to RED [41] that could make the effects of the maximum threshold cutter less severe compared to the results in Pub. II.

threshold. Following the first, earlier drops, the TCP flows already reacted and reduced their sending rate, however, they cannot make any progress as now the retransmissions are dropped. Therefore RTOs are needed to restart the flows like with the Taildrop fallback. Because the RED dropping probability still remains high, more RTOs are likely needed, but now with an exponentially backed-off timeout value, until the drops become infrequent enough to allow TCP flows to maintain reasonable sending rates. Both of these RED issues trace back to too slow a reaction to an ongoing exponential load transient. We also conclude that RED is very sensitive to parametrization and a serious performance degradation may occur during exponential load transients if the parameters are incorrectly set.

In Pub. IV we show that to work properly, "an AQM algorithm must discern between transient queuing that is natural to heterogeneous packet-switched networks and persistent queuing that is caused by too high sending rate". This leads us to one of the open questions in congestion control research related to information acquisition about the "current link load" which "requires defining the right measurement interval / sampling interval" [147]. As the description of this problem in [147] is quite terse, to answer RQ4 we had to first crystallize the actual issues involved in Pub. IV to two distinct, although inter-related problems we named horizon problem and RTT uncertainty.

Pub. IV defines the horizon problem and RTT uncertainty as follows. The horizon problem prevents a router from acquiring a complete picture of the traffic load by simply measuring its current queuing because the distribution of the load over the end-to-end path keeps some of the load-inducing packets out of the view of the router. RTT uncertainty stems from the inability of the router to know the RTTs of the flows going through the router which also makes it challenging for the router to use a proper measurement interval for load calculation as the router does not know what that interval should be. Together these two problems complicate load estimation for AQM algorithms.

The horizon problem and RTT uncertainty manifest in the state-of-the-art AQM algorithms as too slow response that is shown by Pub. II, Pub. III, and Pub. IV. Another form of manifestation is found in Pub. IV that observes too aggressive response with long RTTs. Long RTTs are shown to be problematic for PIE also in [118]. These sub-optimal AQM behavioral patterns can be explained by the use of arbitrary values as load measurement intervals by those AQM algorithms. Such a tuning constant may be explicit, or it can also be implicitly built into other parameters or constants that affect the convergence rate of the formulas used by the AQM algorithm.

Quite often something around 100 msecs RTT is assumed by default (e.g., the weight calculation for ARED [82], the interval in CoDel [142, 143], and 150 msecs burst allowance with PIE [145, 146]). This tuning RTT constant acts as a dividing line for the behavior, if the actual RTT is less than the tuning RTT the AQM response is too slow, and in the opposite case, the response is too fast.

Pub. IV shows that while the AQM algorithm waits, the exponential load transient overloads the bottleneck link with RTTs smaller than the tuning RTT of the AQM algorithm, which leads to long queuing delay. With small RTTs and short flows, the flows tend to end before any congestion control action from the AQM algorithm takes place. However, even with "optimal" RTTs close to the tuning RTT, the delay performance during exponential load transients is far from good according to the results in Pub. IV. Quite contrary actually, Pub. IV shows that both PIE and CoDel-based AQM algorithms experience the longest queuing delays when the RTT is at or slightly above the tuning RTT.

With long RTTs, the tuning RTT starts to cause performance degradation because TCP Slow Start is cut short by the AQM response to a transient queue spike. As discussed in Pub. IV, this behavior is worrisome because it hurts flow completion times for intercontinental flows, users with extra terrestrial access, and other flows that have a long RTT. As both PIE and FQ-CoDel are aiming for real, wide-scale deployment in the general Internet, it may eventually cause issues for such use cases when the penetration of those AQM algorithms becomes widespread enough.

Some recent AQM proposals remove the measurement interval completely. As discussed earlier, DCTCP uses the degenerated case of the RED formula effectively eliminating the average that substitutes for a measurement interval. We believe this is a turn in a completely wrong direction when designing an AQM algorithm. Instead, one important design principle of RED is to allow transient queue during exponential load transients [80, 81]. We believe this is an approach that has a solid foundation in the light of the horizon problem. Discarding the measurement interval subjects the AQM algorithm to the mercy of the horizon problem resulting in flow completion time issues during transients that are cut short. Such problems have already been reported to occur with DCTCP [58] and requires a band-aid solution to allow the bandwidth-probing phase at the sender side to power through congestion signals [35, 37], all because of a flawed load estimate in the AQM algorithm at the router. In case the AQM algorithm is using sampling rather than a measurement interval such as the use of interval by

CoDel, the horizon problem may also mislead the algorithm to believe that the load during an exponential load transient is much less than it really is.

Pub. III highlights another pitfall for AQM algorithms. If the AQM algorithm is sending congestion feedback too infrequently, it cannot scale well to exponential load transients when there are multiple flows present. The results with CoDel clearly show that there is very little scaling when the actual load increases. Instead, the congestion signalling follows exactly the same pattern except that CoDel needs to stay in the congestion signalling state longer because unsignalled flows still keep increasing their sending rate exponentially until finally signalled. Making transient queue spikes of the other flows transparent is one of the reasons why CoDel is recommended always to be deployed with fair queuing [103, 104]. However, Pub. IV shows that at least for the flows participating in the transient, the delay is not transparent and keeps growing also for the best flow with SFQ-CoDel.

Unfortunately our results in Pub. IV do not allow drawing conclusions on how interactive traffic that is not the cause for the exponential load transient would perform with fair queuing because none of our workloads included interactive traffic. We suspect, however, that it depends heavily on the actual sending rate of the interactive traffic. If the sending rate with fair queuing is below the equal share, the performance of the interactive traffic is likely good but with rates higher than the equal share or with bursts, the exponential load transient would likely have a significant impact also on the delay of the interactive traffic. That is, it would likely experience the delay increase even if it is not itself the cause for the exponential load transient. Simply assuming the sending rate of the interactive traffic to be small enough to be below equal share is hazardous at best. On the contrary, Internet traffic patterns are known to be bursty and the interactive traffic itself has no control over how many flows it is competing with. The rigid boundary fair queuing enforces has been shown to be a problem to HTTP adaptive streaming traffic [187]. Therefore it likely makes sense to have well working AQM control also with fair queuing rather than trying to offload the transient handling from AQM to fair queuing.

Pub. IV introduces yet another challenge with exponential load transients that occur because of a high rate of load growth during the late phase of a transient. As the rate of change is very fast, also the load estimate becomes stale with a similar rate. Because of various practical obstacles that are outside the scope of this thesis, the router can send a congestion signal only in-band with a packet of a flow. Therefore the delivery of the congestion signal only occurs, at earliest, when the sender receives back the first ACK that was sent by the receiver after delivery of the packet con-

taining the in-band congestion signal. The effect of the congestion response then becomes visible at the bottleneck router only after the first packet (or possibly a hypothetical packet that would have been sent otherwise but the response prevented sending it) after the response reaches the router. In total, this feedback loop takes one RTT for the effects of the congestion response to finally reach back to the signalling router. Therefore, any AQM decision based on the estimate about the "current link load" is already using a stale value for the load. In reality during an exponential load transient, the load has already increased and the router is already committed beyond remorse to receiving that load increase. The increase of the load will become visible only later because the horizon problem still prevents the visibility during the congestion signalling RTT, yet the router can no longer signal in time to have any effect on that load increase.

The research question RQ5 looks for a solution to the "current link load" problem by taking the horizon problem and RTT uncertainty into account. The research questions RQ6 and RQ7 go even beyond RQ5 and attempts to rectify the RTT-long feedback latency and calculate the correct time for sending the congestion signal.

3.3 Alleviating AQM Problems with Load Transients

In order to avoid most of the problems the exponential load transients cause to AQM algorithms, we believe a paradigm shift in the AQM research is needed. To the best of our knowledge, handling the exponential load transients has never before been a focus during the design phase of the AQM algorithms. Instead, the main approach is to focus on Congestion Avoidance and if any attention is paid to the exponential load transients due to bandwidth probing, all some AQM algorithms aim to do is to not react to them (e.g., [81, 103]). Therefore, there simply are no AQM algorithms that handle exponential load transients well. Because of ignoring exponential load transients, the designed AQM algorithms have a hard time to correctly manage them as they are much more rapid than the algorithms expect by design, which results in delay spikes. Others measuring AQM performance during exponential load transients echo this observation [93].

In order to resolve the issues with the exponential load transients, we believe AQM algorithms must be designed primarily for handling exponential load transients that are much more aggressive than the behavior during Congestion Avoidance. The difference between exponential load transient and Congestion Avoidance is like the difference between a nasty storm and

calm waters. As exponential load transients are frequent in the general Internet due to flows constantly starting up using TCP Slow Start, ignoring the effects of exponential load transients does not seem a wise design decision. We also believe that if an AQM algorithm is able to handle exponential load transients well, it will be much easier to handle also Congestion Avoidance that is much less aggressive compared with the exponential load transients.

Our first attempt to solve the issues with exponential load transients is based on the existing RED algorithm and could therefore possibly exploit the existing real hardware that is already deployed with the RED capability. We independently understood that RED is based on a sane design principle in determining the link load in transient-aware manner before finding it already mentioned [81]. Effectively, a properly configured RED offers a solution to the horizon problem. Such a solid foundation inspires us to use it as a basis for our work rather than some other AQM algorithms that do not have the same design principle. Unfortunately, the transient awareness is not reflected in the RED parametrization guidelines [77, 80].

In Pub. II we present HRED (Harsh RED) that takes exponential load transients into account by reversing the common AQM control reasoning "when at the earliest to drop" into "when at the latest must drop" to produce parametrization that puts a deadline on arresting the rapid load growth during transients. As a result, HRED achieves reasonable performance during exponential load transients stopping them in time. HRED successfully stays in the pro-active dropping mode spacing losses out in contrast to Taildrop or RED with recommended parameters that tend to drop many packets in a burst or even consecutive packets. However, there is a serious limitation with HRED as it needs known end-to-end parameters, that is, even though HRED addresses the horizon problem during the exponential load transients, it is seriously affected by RTT uncertainty. The other limitations of HRED are its use of stale load estimate during a rapidly developing exponential load transient and its inability to adapt to the less aggressive Congestion Avoidance operating mode of TCP leading easily to underutilization when no flow is undergoing an exponential load transient. The latter of them leads us to believe that also the AQM algorithm should have one operating mode responding to Slow Start and another for Congestion Avoidance.

3.4 The Predict AQM Algorithm

In the general Internet, RTTs are not constant nor known to the router that runs an AQM algorithm. As HRED would require reconfiguration whenever its tuning RTT is not correct, it is not practical from the deployment point of view. Armed with the knowledge about what is still missing from HRED, we turn our attention to RTT uncertainty and how to address it on a router that does not have knowledge about the RTTs of the flows going through the router. We also realized that once the horizon problem and RTT uncertainty are resolved, we can also address the stale load estimates as a free bonus because the key parameters affecting the load development during exponential load transients are then known to the router. Therefore, an AQM algorithm solving the horizon problem and RTT uncertainty is not only able to estimate the "current link load" but also able to predict the load into the near future.

In Pub. IV we introduce the Predict AQM algorithm that addresses the horizon problem and RTT uncertainty. In addition, its capabilities include the ability to cast short-term predictions on how the load will develop whenever the heuristics used in the algorithm detects an exponential load transient. Pub. IV compares the performance of the Predict AQM algorithm to that of PIE, CoDel, and SFQ-CoDel over a large RTT range and finds that all other AQM algorithms have serious issues described in Section 3.2, whereas the Predict AQM algorithm scales over the entire tested RTT range as expected. One interesting detail is that both flow completion time and queuing delay with the Predict AQM algorithm are managed nearly optimally. That is, the Predict AQM algorithm does not trade off any extra from low queuing delay in an attempt to improve flow completion times when no improvement is possible nor harms flow completion times to slightly lower the queuing delay. Almost all queuing delay that is imposed during an exponential load transient with the Predict AQM algorithm is simply to avoid hurting flow completion times badly. The other AQM algorithms do harm flow completion times with large RTTs to keep the queuing delays within the configured limits because they are subject to RTT uncertainty.

One might argue that it is also wise to limit the amount of transient queuing with the larger RTTs. We agree on principle, however, such limitation of transient queuing with CoDel is an unintentional side-effect and was not the original intention of the algorithm that aims to let a transient queue pass unharmed [103]. PIE [146] offers a configurable parameter to decide how large bursts are allowed but unfortunately the effectiveness of such a constant parameter is limited by RTT uncertainty. DCTCP takes the lim-

iting of transient bursts to the very extreme at the cost of flow completion times because of premature exit from TCP Slow Start [58]. The Predict AQM algorithm, on the other hand, can really discern transient queuing from a persistent one, which truly enables a policy decision on how much transient queuing should be allowed with larger RTTs (this feature is not implemented currently by the Predict algorithm given in Pub. IV). The transient allowance parameters used to limit queuing in the state-of-the-art AQM algorithms are constants that prevent timely response with short RTTs, whereas Predict can react before such a transient queuing bound is exceeded when the real RTTs are short. Therefore, enabling such a limit for transient queuing with the Predict AQM algorithm would not impact its ability to manage queue with smaller RTTs. Furthermore, Pub. IV shows that Predict already as is outperforms the other AQM algorithms by a large margin in latency also when the real RTT is close to the constant tuning RTTs used by the other AQM algorithms.

During development of the Predict AQM algorithm we made an interesting observation on head or tail drop dilemma [122, 143]. The load estimate in the state-of-the-art AQM algorithms is always badly behind during exponential load transients and the use of head drop is simply an attempt to improve the situation slightly compared with the tail drop that would deliver the congestion signal even later. The correct solution, of course, is to address the root problem with stale load estimates instead of trying to work around the issue using the head drop that has only limited effectiveness. In contrast, the Predict AQM algorithm is well ahead of the traffic with its predictions about the future load and it does not need to act hastily. Therefore, we realized that sending congestion signals from the head of the queue would make them occur too early rather than in a timely manner as the Predict AQM algorithm optimizes the load for its vantage point at the tail of the queue, not at the head of the queue.

In addition to solving the horizon problem, RTT uncertainty, and stale load estimates, our fourth design objective with the Predict AQM algorithm was to make it truly pacing compatible. We believe pacing is ultimately the only way around the large transient queue spikes during the bandwidth-probing phase, which occur due to the inherent limitations of the self-clocked bandwidth probing. However, realizing pacing together with the state-of-the-art AQM algorithms is challenging because the better pacing gets, the less queue there is for the AQM algorithm to measure. In the extreme case with ideal pacing, there will not be a queue until the link is fully utilized [3, 135, 164]. With no queue, there is no indication of congestion for the queue length or delay-based AQM algorithms until too

late for the router to react in time and a Slow-Start overshoot occurs, which results in the sender exceeding the targeted sending rate. The Predict AQM algorithm, on the other hand, is not based on measuring the queue only but has a more complete picture of the load development even when no queue is visible at the router. Therefore, we believe it is much more compatible with pacing. However, we have not yet fully proven this property of the Predict AQM algorithm through experiments.

We believe that the Predict AQM algorithm introduced in Pub. IV is a major advancement in solving the open question in congestion control research about determining the "current link load" [147]. Our solution with the Predict AQM algorithm, however, not only involves solving both the horizon problem and RTT uncertainty but also redefines the question by taking into account that the "current link load" is already stale when measured. We believe those are the root issues behind determining the "current link load" challenge. Unfortunately RFC 6077 is rather terse in describing the issue [147] and in our opinion fails to describe the problem in a sufficient manner. We believe that any solution trying to determine the "current link load" needs to solve the above-mentioned horizon problem, RTT uncertainty, and also has to consider the effect of stale load estimate.

We built the Predict AQM algorithm as a proof-of-concept for testing whether an AQM algorithm can detect the signal given by TCP Slow Start successfully in time to produce a correctly timed congestion signal. With the timely congestion signal, Predict allows relaxing the multiplicative decrease (MD) response to TCP Slow Start because the Slow Start overshoot occurring with the other AQM algorithms that respond too slowly is avoided. In the current state, the Predict AQM algorithm has shown that the timely detection of TCP Slow Start is possible and achieves significant performance benefits, however, it is not ready for deployment to the Internet in this limited form. In order to make deployment a practical possibility, the relaxed multiplicative decrease compatibility with the current congestion signalling, Congestion Avoidance handling, and fairness aspects need to be solved, which likely must happen together as they seem to depend on each other. In addition, developing an improved version of Predict that would have a smaller memory footprint with the same or better accuracy would make deploying Predict less challenging. It would mitigate the current trade-off in Predict to use a large memory footprint to reduce the computational cost of the algorithm. Nevertheless, even the current, large memory requirement is calculated in Pub. IV to be less than the amount of memory the router needs for storing the huge burst of packets that TCP Slow Start may send with long RTTs.

3.5 Reflections on Flow Initiation and Restart

In Sections 3.3 and 3.4 we created two AQM algorithms that take the aggressive exponential load transients that occur during the bandwidth-probing phase of a flow properly into account. However, there is still the flow initiation before the flow even starts its bandwidth-probing phase, which is not covered by our algorithms. As mentioned in Chapter 2, there are proposals to make the flow initiation much more aggressive than the standardized Initial Window of three TCP segments and flow restart to use a much more aggressive sending rate. We will now see how the insights from our AQM work reflect on flow initiation and restart.

There is an important difference between packets in the Initial Window and packets from subsequent RTTs of a flow. The former is unresponsive traffic in that there is only very limited control possible (through TCP SYNs), whereas the latter is fully controllable through the ACKs for the packets of the previous RTT. This key difference in controllability is what our AQM algorithms exploit to tame the exponential load transients. If, however, the Initial Window burst is enlarged, there will be more queuing at the buffer in front of the bottleneck link.

The proposal to remove the Initial Window completely puts trust on AQM algorithms on solving the issues the extra queuing will cause [9]. However, AQM algorithms do not help as they cannot pre-emptively prevent sending the Initial Window. Also, they will likely require more than one RTT worth of tracking for the traffic like our algorithm in Pub. IV (or in general, any measurement or sampling interval-based algorithm) does, to produce a meaningful response. The most intuitive way to work around this AQM shortcoming is then to propose fair queuing instead. However, the SFQ-CoDel results in Pub. IV indicate that it might turn out to be a false hope but more work is needed to confirm it is indeed the case. As such, the increase of the Initial Window is likely to further complicate AQM design as it reduces the wiggle room an algorithm designer has to timely respond to load excursions.

3.6 Summary

In this chapter we have discussed how the state-of-the-art AQM algorithms have not been designed with exponential load transients in mind but instead base their control systems on much slower changing Congestion Avoidance behavior. Exponential load transients, however, are a frequent phenomenon in the network. As the effect of exponential load transients have been

ignored during the AQM algorithm design phase, the AQM algorithms have a hard time to correctly manage them. Exponential load growth during a transient can easily overpower the control authority of an AQM algorithm because the load is changing much faster than the algorithm expects by design, leading to delay spikes during exponential load transients.

Lack of true knowledge about exponential transient parameters makes it hard to react correctly. Depending on the RTT of the exponential traffic load, the state-of-the-art AQM algorithms react either too slow or too fast. The former lead to excessive delay and the latter sacrifices throughput. In this thesis those problems are shown to originate from two main issues we named the horizon problem and RTT uncertainty. In addition, the rapidly growing load causes AQM algorithms to use stale load estimates. There is no state-of-the-art AQM algorithm that solves these three issues adequately.

Our first attempt in this thesis to build an exponential load transient-aware AQM algorithm resulted in the HRED AQM algorithm. The HRED algorithm could have exploited the RED algorithm support on deployed hardware but unfortunately it only solves the horizon problem. As such, an unacceptable restriction requiring a known end-to-end RTT with HRED remains which stems from its inability to address RTT uncertainty.

We then construct a new AQM algorithm called Predict AQM algorithm whose primary design goal is properly handling exponential load transients. The Predict AQM algorithm achieves superb response to exponential load transients without the need to trade off latency for throughput nor vice-versa. Even with RTTs that are "optimal" compared with the tuning RTTs used in the state-of-the-art AQM algorithms, the Predict AQM algorithm wins by a huge margin in latency.

The Predict AQM algorithm is also designed to be fully compatible with pacing. Pacing would be very useful in removing the transient queue spikes that occur during exponential load transients. However, pacing has been problematic for state-of-the-art AQM algorithms because load estimation when the bottleneck link is not fully utilized is even more challenging with pacing than without it. The challenges are due to the horizon problem and RTT uncertainty. Therefore, the Predict AQM algorithm that solves both of those problems offers potential for a near-zero queue without falling victim to a surprise Slow-Start overshoot once the link becomes fully utilized.

Measuring the "current link load" has been an open question in congestion control research this far [147]. However, the Predict AQM algorithm presented in this thesis is not only able to determine the "current link load" but is also able to predict the load into the near future whenever an

exponential load transient is detected. As such, the work done in this thesis gives significant insight into how to solve the "current link load" open question. Notably, the Predict AQM algorithm has possibly solved it already for the most challenging part, the exponential load transients, which is much more challenging to control than less aggressive behavior during Congestion Avoidance.

Chapter 4

Conclusions

This thesis contributes to the area of Active Queue Management (AQM), end-host congestion control, and their interaction during flow startup.

4.1 Summary of Contributions

The particular contribution in this thesis is the Predict AQM algorithm that is the first exponential load transient-aware AQM algorithm. The Predict AQM algorithm not only detects exponential load transients but determines the parameters of the exponential load growth from the traffic itself. Based on those parameters, it can predict how the link load likely develops in the near-term future by assuming the exponential trend still keeps continuing. Because of its prediction capability, the Predict AQM algorithm is able to give very timely congestion feedback that also takes into account the inherent RTT-long latency when the congestion feedback is signalled in-band.

This thesis also describes the challenges any AQM algorithm willing to properly address exponential load transients needs to solve, namely the horizon problem, RTT uncertainty, and stale load estimates. The first two challenges stem from the limitations imposed by the placement of the AQM algorithm on an Internet router which typically offers only a limited view about the traffic going through the router. Because the current link load is rapidly changing during exponential load transients, the second and third challenges are important to address. If the AQM algorithm has no understanding about the timescale affecting load growth, which for a Slow-Starting flow is coupled to the RTT of the flow, it easily ends up responding to congestion too early or too late, which both come with negative consequences as highlighted in this thesis.

With the understanding given in this thesis about the challenges encountered during exponential load transients, we conclude that AQM algorithms, in general, should consider exponential load transients already during their design phase. If exponential load transients are not considered in the design phase, an AQM algorithm is very likely to exhibit sub-par performance during exponential load transients. As exponential load transients are bound to occur whenever long enough flows are starting and probing for the bandwidth to use, ignoring exponential load transients in AQM algorithm design is likely to have unfortunate consequences during actual operation of the algorithm.

Another interesting insight from the Predict AQM algorithm is in how it tackles the current load estimation challenge that is listed as an open challenge in congestion control research [147]. Rather than simply giving an answer to the question "what is the current link load?", the Predict AQM algorithm takes one step further and answers when the AQM algorithm needs to start reacting in order to meet the desired link load target. This approach solves the problem with the current link load that is stale already at the time it is measured, that is, the current load may be much less than the targeted load at the time when the AQM algorithm must already react. As such, this thesis may contribute significantly on resolving the challenge about estimating the current link load in AQM algorithms.

This thesis also describes the HRED AQM algorithm that worked as an intermediate step towards the Predict AQM algorithm. HRED is based on the RED AQM algorithm that allows using "a measurement interval" for the current link load estimation [147] which is necessary because of the horizon problem. HRED addresses exponential load transients reasonably well for a known end-to-end RTT if the algorithm is parametrized according to the RTT. However, as the RED algorithm is still a conventional AQM algorithm, it offers no solution to RTT uncertainty nor stale load estimate challenges. As such, it does not scale to the Internet environment with large variation in RTTs of the flows as continuous manual reconfiguration is not a realistic option. Nevertheless, it serves as a good example about the expected limitations of a conventional AQM design that does not take exponential load transients specifically into account. In addition, HRED overcontrols TCP flows that are in Congestion Avoidance highlighting the need for differentiated control depending on whether all flows operate in Congestion Avoidance or at least one of them operates in the much more aggressive Slow Start.

The performance results in this thesis evaluate how the state-of-the-art AQM algorithms PIE, CoDel, and SFQ-CoDel behave during exponential

load transients. To the best of our knowledge, only little research specifically related to AQM behavior during exponential load transients has been conducted by others [93]. Contrary to findings in the earlier studies, our results also hint that offloading transient handling to fair queuing may not be enough to address the queue delay spikes that may occur during exponential load transients. Instead, also an exponential load transient-aware AQM algorithm may be needed.

This thesis also gives insights into how the end hosts should manage flow initiation. In particular, combining the proposed larger Initial Window (IW) of ten segments with parallel flow initiation or continuation after a low activity period easily imposes delays exceeding what interactive traffic can tolerate. As the sender has no way to measure the impact until feedback arrives after one RTT, the harm is already done before the sender can react. In contrast, more careful initiation together with exponential ramp up allows congestion control reaction to take place from properly designed AQM algorithms such as Predict avoiding excessive delay spikes.

4.2 Impact for Active Queue Management Research

At minimum, the research questions answered in this thesis deepen the understanding of the current link load estimation challenge by refining the actual question to a form that is more useful for AQM algorithms. The partial answer offered to the link load estimation challenge by this work alone solves a significant portion of the operating envelope of routers near the network edge where exponential load transients are very frequent.

Our angle on the design of AQM algorithms is a new way to look at load estimation in AQM algorithms. We believe that a paradigm shift is needed in the AQM algorithm research which has previously focused almost solely on solving the Congestion Avoidance mode of operation. We highlight the importance of considering exponential load transients already during AQM algorithm development. If, however, the exponential load transients keep getting ignored during the design phase, the performance is likely not going to get very nice during them. Designing only for Congestion Avoidance is like designing a ship only for calm waters. This work attests that exponential load transients are rather nasty storms and argues that they also are frequent enough that they must be considered by a prudent AQM design. As the congestion control is built such that these exponential load transient storms only settle once the AQM algorithm reacts and meanwhile

grow intensity rapidly, the AQM algorithm must take proper action rather than hope for the best.

4.3 Future Work

The Predict AQM algorithm proposed in this thesis was built as proof of concept for detecting and responding properly to exponential load transients. As such, it is not ready for the general Internet. In particular, we did not yet address deployability and handling for long-running flows that operate long in Congestion Avoidance mode after the initial bandwidth-probing phase has completed. We believe these problems are highly intertwined. To make the algorithm deployable, an approach for sending the congestion feedback has to be selected and it must be able to inter-operate with the existing Internet traffic. The selected feedback approach then may have significant implications on how flows behave after the bandwidth-probing phase. As such, these problems likely need to be solved together, which we leave to future work.

Other development areas with the Predict AQM algorithm include potential improvements to its heuristics for smaller memory footprint, and combining Predict with sender-side pacing to ultimately provide close to zero queue during the bandwidth-probing phase. For the latter, the approach used by Predict seems much more robust than the existing AQM algorithms that use queue length or a dual of it, queuing delay, as the proxy for their load estimation. The use of sender-side pacing invalidates queue as the proxy variable for the load until the bottleneck link is saturated. Once the saturation takes place, the congestion feedback in an exponential load transient is already late by at least RTT leading to an unavoidable Slow-Start overshoot. Thus, better pacing works to mitigate queuing delay spikes during bandwidth probing, the longer the latency spikes become during the Slow-Start overshoot because pacing invalidates the assumptions the AQM algorithms using queue length depend on. Hence, also the response from such AQM algorithms will be delayed if pacing is enabled. In contrast, we would like to show that the Predict AQM algorithm is not similarly vulnerable and allows sender-side pacing to be enabled with significant latency gains.

It would also be highly interesting to proceed with a real implementation of the Predict AQM algorithm with, for example, a Linux or NetFPGA [85] platform. A real implementation would facilitate testing the performance of Predict with real web pages in a compatibility mode that detects a load of 200%, which together with the standard TCP performing multiplicative

decrease (MD) leads to the correct sending rate. Such testing could confirm the latency benefits of Predict for short RTTs and verify that the page load time remains unharmed with long RTTs.

The insight into how to design AQM algorithms for exponential load transient awareness gained in this thesis may also enable designing other exponential load transient-aware AQM algorithms besides Predict. In particular, curve fitting-based approaches seem appealing especially from a robustness perspective, however, the computational complexity of the fit may make such approaches impractical. If such an algorithm is devised, it should also properly address the pacing compatibility so that the algorithm is able to provide congestion feedback when no perceivable queue exists.

4.4 Thoughts on Future Internet Congestion Control Architecture

In the long term, we believe that sender-side pacing will need to be introduced to reduce burstiness in transmission. Pacing will help in particular during the bandwidth-probing phase as it allows eliminating transient queue spikes before the bottleneck link is saturated. The reduced queuing offers two major advantages, it reduces the probability of drops due to queue overflow and improves latency. We already see significant interest in this direction in datacenter-centric congestion control proposals such as HULL [6] and more importantly in TCP BBR [45, 46] that aims not only for datacenters but also for the general Internet. Also QUIC senders are recommended to enable sender-side pacing [100].

Once the sender-side pacing gets enabled, it will become evident that the approaches used for load estimation in many current AQM algorithms are insufficient to address the horizon problem. Therefore AQM algorithms need to become fully compatible with pacing as otherwise the latency benefits pacing offers are nullified by the rapidly increasing load after the bottleneck link becomes saturated. The pacing compatibility and exponential load transient aspects explored in this thesis are therefore significant building blocks needed to realize congestion control in the future Internet according to our understanding.

Another important development aspect is multiplicative decrease (MD) that is too dramatic to result in good utilization if AQM successfully keeps queuing at a low level. The two main approaches currently proposed for it are the TCP Alternative Backoff (ABE) [109–111] that simply alters the MD factor from 0.5 and enhanced congestion feedback such as the fine-grained *reduce slightly* congestion indication in DCTCP or its successors [5,

24, 38]. The *keep current rate* indication we used for Predict in this thesis is also one potential form of enhanced congestion feedback. At the other end of spectrum, there is the need to rapidly increase the sending rate when surplus capacity is available but the active flows are already past the bandwidth-probing phase, which may require introducing, for example, an *exponential increase allowed* indication such as in the variable-structure congestion control protocol (VCP) [126, 197] and AntiECN [121]. Together, we get *exponential increase allowed*, *keep current rate*, and the fine-grained *reduce slightly* indications. The *reduce slightly* indication allows fine-grained control but a larger reduction can still be produced by sending an appropriate number of them making the currently used MD indication unnecessary. By sending both the *reduce slightly* and *exponential increase allowed* indications, it may be possible to handle also the *keep current rate* indication without an explicit flag. Alternation between those two indications would also add some natural shuffling to the traffic during steady periods, hopefully improving fairness similar to the fairness controller in XCP [108]. In the end of this road, looms the elimination of the entire Congestion Avoidance from the end-host congestion control. It seems to us useful as its increase is highly dependent on RTT and hard to estimate at the router-side algorithms. The *exponential increase allowed* indication, on the other hand, offers a very deterministic increase, that is, the load doubles exactly for each packet flagged with the *exponential increase allowed* indication.

In general, enhanced congestion feedback will be challenging to deploy as taking advantage of them requires both router and end-host support, and the new architecture must also inter-operate with the existing traffic. One take into this direction is the DualQ coupled congestion control [58, 60] which circumvents the compatibility issues with the existing traffic by differentiating the legacy traffic from the next generation feedback-aware traffic that uses a congestion control derived from DCTCP. As DCTCP-based congestion control is known to cause a premature exit from the bandwidth-probing phase, the Predict AQM algorithm together with pacing could offer a solution to the bandwidth-probing phase within a DualQ-like approach.

Besides the congestion control machinery, the future Internet may incorporate Congestion Exposure (ConEx) [39, 129] to make the sources accountable for the congestion they cause downstream in the network. In the context of network congestion control, ConEx offers protection from misbehaving sources for congestion control algorithms that, in general, assume honest and responsive behavior when the network sends congestion indications.

References

[1] H. Abrahamsson and B. Ahlgren. "Using Empirical Distributions to Characterize Web Client Traffic and to Generate Synthetic Traffic". In: *Proc. IEEE Global Telecommunications Conference (GLOBECOM '00)*. (San Francisco, California, USA). Vol. 1. November 2000, pp. 428–433. DOI: 10.1109/GLOCOM.2000.892041.

[2] *Active Queue Management and Packet Scheduling (aqm) – Charter for Working Group*. URL: https://datatracker.ietf.org/wg/aqm/charter.

[3] A. Aggarwal, S. Savage, and T. Anderson. "Understanding the Performance of TCP Pacing". In: *Proc. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM 2000)*. (Tel Aviv, Israel). March 2000, pp. 1157–1165. DOI: 10.1109/INFCOM.2000.832483.

[4] S. Alfredsson, G. Del Giudice, J. Garcia, A. Brunström, L. De Cicco, and S. Mascolo. "Impact of TCP Congestion Control on Bufferbloat in Cellular Networks". In: *Proc. 14th International Symposium and Workshops on a World of Wireless, Mobile and Multimedia Networks (IEEE WoWMoM 2013)*. (Madrid, Spain). June 2013, pp. 1–7. DOI: 10.1109/WoWMoM.2013.6583408.

[5] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. "Data center TCP (DCTCP)". In: *Proc. SIGCOMM '10*. (New Delhi, India). August 2010. DOI: 10.1145/1851182.1851192.

[6] 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. 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*. (San Jose, California, USA). April 2012.

[7] M. Allman. "Comments on Bufferbloat". In: *ACM SIGCOMM Computer Communications Review* 43.1 (January 2012), pp. 30–37. DOI: 10.1145/2427036.2427041.

[8] M. Allman. "On the Generation and Use of TCP Acknowledgements". In: *ACM SIGCOMM Computer Communications Review* 28.5 (October 1998), pp. 4–21. DOI: 10.1145/303297.303301.

[9] M. Allman. *Removing TCP's Initial Congestion Window*. Internet Draft. Work in progress. Internet Society, November 2015.

[10] M. Allman. *TCP Congestion Control with Appropriate Byte Counting (ABC)*. RFC 3465. February 2003.

[11] M. Allman, S. Dawkins, D. Glover, J. Griner, D. Tran, T. Henderson, J. Heidemann, J. Touch, H. Kruse, S. Ostermann, K. Scott, and J. Semke. *Ongoing TCP Research Related to Satellites*. RFC 2760. February 2000.

[12] M. Allman, S. Floyd, and C. Partridge. *Increasing TCP's Initial Window*. RFC 3390. October 2002.

[13] M. Allman, V. Paxson, and E. Blanton. *TCP Congestion Control*. RFC 5681. September 2009.

[14] M. Allman and E. Blanton. "Notes on Burst Mitigation for Transport Protocols". In: *ACM SIGCOMM Computer Communications Review* 35.2 (April 2005), pp. 53–60. DOI: 10.1145/1064413.1064419.

[15] G. Appenzeller, I. Keslassy, and N. McKeown. "Sizing Router Buffers". In: *Proc. 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '04)*. (Portland, Oregon, USA). August 2004, pp. 281–292. DOI: 10.1145/1015467.1015499.

[16] M. Aron and P. Druschel. *TCP: Improving Startup Dynamics by Adaptive Timers and Congestion Control*. Tech. rep. TR98-318. Rice University, June 1998.

[17] S. Bae, D. Jang, and K. Park. "Why Is HTTP Adaptive Streaming So Hard?" In: *Proc. 6th Asia-Pacific Workshop on Systems (APSys '15)*. (Tokyo, Japan). July 2015, 12:1–12:8. DOI: 10.1145/2797022.2797031.

[18] M. Bagnulo and B. Briscoe. *ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets*. Internet Draft. Work in progress. Internet Society, October 2018.

[19] F. Baker and G. Fairhurst. *IETF Recommendations Regarding Active Queue Management*. RFC 7567. July 2015.

[20] C. Barakat. "Performance Evaluation of TCP Congestion Control". PhD thesis. University of Nice - Sophia Antipolis, April 2001.

[21] P. Barford and M. Crovella. "Generating Representative Web Workloads for Network and Server Performance Evaluation". In: *Proc. 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '98/SIGPERFORMANCE '98)*. (Madison, Wisconsin, USA). June 1998, pp. 151–160. DOI: 10.1145/277851.277897.

[22] M. Belshe and R. Peon. *SPDY Protocol*. Internet Draft. Work in progress. Internet Society, February 2012.

[23] M. Belshe, R. Peon, and M. Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540. May 2015.

[24] S. Bensley, D. Thaler, P. Balasubramanian, L. Eggert, and G. Judd. *Data Center TCP (DCTCP): TCP Congestion Control for Data Centers*. RFC 8257. October 2017.

[25] T. Berners-Lee and D. Connolly. *Hypertext Markup Language - 2.0*. RFC 1866. November 1995.

[26] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945. May 1996.

[27] T. Berners-Lee. *Information Management: A Proposal*. March 1989.

[28] *Beta for multiplicative decrease*. Linux kernel variable. URL: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/net/ipv4/tcp_cubic.c?h=v5.0#n47.

[29] M. Bishop. *Hypertext Transfer Protocol (HTTP) over QUIC*. Internet Draft. Work in progress. Internet Society, October 2018.

[30] E. Blanton, M. Allman, L. Wang, I. Järvinen, M. Kojo, and Y. Nishida. *A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP*. RFC 6675. August 2012.

[31] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. *Recommendations on Queue Management and Congestion Avoidance in the Internet*. RFC 2309. April 1998.

[32] L. Brakmo, S. O'Malley, and L. Peterson. "TCP Vegas: New Techniques for Congestion Detection and Avoidance". In: *Proc. Conference on Communications Architectures, Protocols and Applications (SIGCOMM '94)*. (London, United Kingdom). August 1994, pp. 24–35. DOI: 10.1145/190314.190317.

[33] L. Brakmo and L. Peterson. "TCP Vegas: End to End Congestion Avoidance on a Global Internet". In: *IEEE Journal on Selected Areas in Communications* 13.8 (September 1995), pp. 1465–1480. DOI: 10.1109/49.464716.

[34] B. Briscoe. *Review: Proportional Integral controller Enhanced (PIE) Active Queue Management (AQM)*. Tech. rep. TR-TUB8-2015-001. BT, May 2015.

[35] B. Briscoe and K. De Schepper. "Ultra-Low Queuing Delay For All". In: *The 95th Internet Engineering Task Force (IETF-95) meeting (l4s Bar BoF)*. (Buenos Aires, Argentina). April 2016. URL: http://bobbriscoe.net/presents/1604ietf/1604-l4s-bar-bof.pdf.

[36] B. Briscoe, K. De Schepper, and M. Bagnulo Braun. *Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture*. Internet Draft. Work in progress. Internet Society, October 2018.

[37] B. Briscoe, K. De Schepper, and M. Bagnulo Braun. *Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Problem Statement*. Internet Draft. Work in progress. Internet Society, June 2016.

[38] B. Briscoe, M. Kühlewind, and R. Scheffenegger. *More Accurate ECN Feedback in TCP*. Internet Draft. Work in progress. Internet Society, July 2018.

[39] B. Briscoe, R. Woundy, and A. Cooper. *Congestion Exposure (ConEx) Concepts and Use Cases*. RFC 6789. December 2012.

[40] B. Briscoe. *Insights from Curvy RED (Random Early Detection)*. Tech. rep. TR-TUB8-2015-003. BT, July 2015.

[41] R. B. Briscoe. "Re-feedback: Freedom with Accountability for Causing Congestion in a Connectionless Internetwork". PhD thesis. UC London, May 2009.

[42] Browserscope. URL: http://www.browserscope.org/?category=network&v=1.

[43] N. Cardwell. *TCP HyStart patch deployment*. Post on IETF tcpm Working Group Mailing List. May 2015. URL: https://www.ietf.org/mail-archive/web/tcpm/current/msg09676.html.

[44] N. Cardwell. *TCP HyStart patch deployment*. Post on IETF tcpm Working Group Mailing List. August 2016. URL: https://www.ietf.org/mail-archive/web/tcpm/current/msg10524.html.

[45] N. Cardwell, Y. Cheng, C. Gunn, S. Yeganeh, and V. Jacobson. "BBR: Congestion-Based Congestion Control". In: *ACM Queue* 14.5 (October 2016), 50:20–50:53. DOI: 10.1145/3012426.3022184.

[46] N. Cardwell, Y. Cheng, C. Gunn, S. Yeganeh, and V. Jacobson. "BBR Congestion Control". In: *Proc. 97th Internet Engineering Task Force (IETF-97) meeting (iccrg)*. (Seoul, South Korea). November 2016.

[47] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. *TCP Fast Open*. RFC 7413. December 2014.

[48] Y. Cheng. *tcp: properly handle stretch acks in slow start*. Linux kernel modifcation. Commit ID 9f9843a751d0a2057f9f3d313886e7e5e6ebaac9. October 2013.

[49] H. Choi and J. Limb. "A Behavioral Model of Web Traffic". In: *Proc. Seventh International Conference on Network Protocols (ICNP '99)*. (Toronto, Ontario, Canada). October 1999, pp. 327–334. DOI: 10.1109/ICNP.1999.801961.

[50] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis. *Increasing TCP's Initial Window*. RFC 6928. April 2013.

[51] Cisco Systems. *Configuring Weighted Random Early Detection*. Manual. URL: https://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfwred.pdf.

[52] Cisco. *Approaching the Zettabyte Era*. Whitepaper. June 2008. URL: https://web.archive.org/web/20090129225202/http://cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481374_ns827_Networking_Solutions_White_Paper.html.

[53] Cisco. *The Zettabyte Era: Trends and Analysis*. Whitepaper. June 2016.

[54] D. Clark. *Window and Acknowledgement Strategy in TCP*. RFC 813. July 1982.

[55] T. Cloonan, J. Allen, T. Cotter, and B. Widrevitz. "Minimizing Bufferbloat and Optimizing Packet Stream Performance in DOCSIS 3.0 CMs and CMTSs". In: *SCTE Cable-tec EXPO '13*. (Atlanta, Georgia, USA). October 2013.

[56] M. Crovella and A. Bestavros. "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes". In: *IEEE/ACM Transactions on Networking* 5.6 (December 1997), pp. 835–846. DOI: 10.1109/90.650143.

[57] L. Daniel and M. Kojo. "Enhancing TCP with Cross-Layer Notifications and Capacity Estimation in Heterogeneous Access Networks". In: *Proc. 37th Annual IEEE Conference on Local Computer Networks (LCN2012)*. (Sydney, New South Wales, Australia). October 2012, pp. 392–400. DOI: 10.1109/LCN.2012.6423653.

[58] K. De Schepper, O. Bondarenko, I. Tsang, and B. Briscoe. *'Data Centre to the Home': Ultra-Low Latency for All*. Under submission. July 2015.

[59] K. De Schepper and B. Briscoe. *Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay (L4S)*. Internet Draft. Work in progress. Internet Society, November 2018.

[60] K. De Schepper, B. Briscoe, O. Bondarenko, and I. Tsang. *DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)*. Internet Draft. Work in progress. Internet Society, July 2018.

[61] K. De Schepper, O. Bondarenko, I. Tsang, and B. Briscoe. "PI^2: A Linearized AQM for both Classic and Scalable TCP". In: *Proc. 12th International on Conference on Emerging Networking EXperiments and Technologies (ACM CoNEXT'16)*. (Irvine, California, USA). December 2016, pp. 105–119. DOI: 10.1145/2999572.2999578.

[62] K. De Schepper, B. Briscoe, O. Bondarenko, and I. Tsang. "PI2: AQM for Classic and Scalable Congestion Control". In: *Proc. 95th Internet Engineering Task Force (IETF-95) meeting (iccrg)*. (Buenos Aires, Argentina). April 2016.

[63] S. Deng. "Empirical Model of WWW Document Arrivals at Access Link". In: *Proc. International Conference on Communications (ICC/SUPERCOMM '96)*. (Dallas, Texas, USA). Vol. 3. June 1996, pp. 1797–1802. DOI: 10.1109/ICC.1996.535600.

[64] T. Dierks and C. Allen. *The TLS Protocol Version 1.0*. RFC 2246. January 1999.

[65] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.1*. RFC 4346. April 2006.

[66] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. August 2008.

[67] M. Dischinger, A. Haeberlen, K. Gummadi, and S. Saroiu. "Characterizing Residential Broadband Networks". In: *Proc. 7th ACM SIGCOMM Conference on Internet Measurement (IMC '07)*. (San Diego, California, USA). October 2007, pp. 43–56. DOI: 10.1145/1298306.1298313.

[68] 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". In: *ACM SIGCOMM Computer Communications Review* 40.3 (July 2010), pp. 26–33. DOI: 10.1145/1823844.1823848.

[69] E. Dumazet. *tcp_cubic: refine Hystart delay threshold*. Linux kernel modification. Commit ID 42eef7a0bb0989cd50d74e673422ff98a0ce4d7b. December 2014.

[70] G. Fairhurst, A. Sathiaseelan, and R. Secchi. *Updating TCP to Support Rate-Limited Traffic*. RFC 7661. October 2015.

[71] W. Feng, D. Kandlur, D. Saha, and K. Shin. "A Self-Configuring RED Gateway". In: *Proc. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM '99)*. (New York, New York, USA). March 1999. DOI: 10.1109/INFCOM.1999.752150.

[72] W. Feng, D. Kandlur, D. Saha, and K. Shin. *Techniques for Eliminating Packet Loss in Congested TCP/IP Networks*. Tech. rep. CSE-TR-349-97. Computer Science and Engineering University of Michigan, November 1997.

[73] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2068. January 1997.

[74] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. June 1999.

[75] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230. June 2014.

[76] S. Floyd. *Limited Slow-Start for TCP with Large Congestion Windows*. RFC 3742. March 2004.

[77] S. Floyd. *RED: Discussions of Setting Parameters*. November 1997. URL: http://www.icir.org/floyd/REDparameters.txt.

[78] S. Floyd, M. Allman, A. Jain, and P. Sarolahti. *Quick-Start for TCP and IP*. RFC 4782. January 2007.

[79] S. Floyd and K. Fall. "Promoting the Use of End-to-End Congestion Control in the Internet". In: *IEEE/ACM Transactions on Networking* 7.4 (August 1999), pp. 458–472. DOI: 10.1109/90.793002.

[80] S. Floyd and V. Jacobson. "Random Early Detection Gateways for Congestion Avoidance". In: *IEEE/ACM Transactions on Networking* 1.4 (August 1993), pp. 397–413. DOI: 10.1109/90.251892.

[81] S. Floyd. *Average queue length vs instantaneous queue length.* November 1997. URL: http://www.icir.org/floyd/REDqueue.txt.

[82] S. Floyd, R. Gummadi, and S. Shenker. *Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management.* Tech. rep. International Computer Science Institute (ICSI), August 2001.

[83] A. Freier, P. Karlton, and P. Kocher. *The Secure Sockets Layer (SSL) Protocol Version 3.0.* RFC 6101. August 2011.

[84] J. Gettys and K. Nichols. "Bufferbloat: Dark Buffers in the Internet". In: *ACM Queue* 9.11 (November 2011), 40:40–40:54. DOI: 10.1145/2063166.2071893.

[85] G. Gibb, J. Lockwood, J. Naous, P. Hartke, and N. McKeown. "NetFPGA – An Open Platform for Teaching How to Build Gigabit-Rate Network Switches and Routers". In: *IEEE Transactions on Education* 51.3 (August 2008), pp. 364–369. DOI: 10.1109/TE.2008.919664.

[86] R. Gibbens and F. Kelly. "Distributed Connection Acceptance Control for a Connectionless Network". In: *Proc. 16th International Teletraffic Congress (ITC 16).* (Edinburgh, United Kingdom). June 1999, pp. 941–952.

[87] S. Ha and I. Rhee. "Hybrid Slow Start for High-Bandwidth and Long-Distance Networks". In: *Proc. Sixth International Workshop on Protocols for FAST Long-Distance Networks (PFLDNeT 2008).* (Manchester, United Kingdom). March 2008.

[88] S. Ha, I. Rhee, and L. Xu. "CUBIC: a new TCP-friendly high-speed TCP variant". In: *SIGOPS Operating Systems Review* 42.5 (July 2008), pp. 64–74. DOI: 10.1145/1400097.1400105.

[89] S. Ha. *tcp_cubic: make the delay threshold of HyStart less sensitive.* Linux kernel modification. Commit ID 2b4636a5f8ca547000f6aba24ec1c58f31f4a91d. March 2011.

[90] M. Handley, J. Padhye, and S. Floyd. *TCP Congestion Window Validation.* RFC 2861. June 2000.

[91] J. Heffner. *[TCP]: Add RFC3742 Limited Slow-Start, controlled by variable sysctl_tcp_max_ssthresh*. Linux kernel modification. Commit ID 886236c1247ab5e2ad9c73f6e9a652e3ae3c8b07. March 2007.

[92] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. *The NewReno Modification to TCP's Fast Recovery Algorithm*. RFC 6582. April 2012.

[93] T. Høiland-Jørgensen, P. Hurtig, and A. Brunström. "The Good, the Bad and the WiFi: Modern AQMs in a Residential Setting". In: *Computer Networks* 89 (October 2015), pp. 90–106. DOI: 10.1016/j.comnet.2015.07.014.

[94] T. Høiland-Jørgensen, P. McKenney, D. Täht, J. Gettys, and E. Dumazet. *The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm*. RFC 8290. January 2018.

[95] T. Høiland-Jørgensen, D. Täht, and J. Morton. "Piece of CAKE: A Comprehensive Queue Management Solution for Home Gateways". In: *Proc. IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN 2018)*. (Washington, District of Columbia, USA). June 2018, pp. 37–42. DOI: 10.1109/LANMAN.2018.8475045.

[96] C. Hollot, V. Misra, D. Towsley, and W. Gong. "Analysis and Design of Controllers for AQM Routers Supporting TCP Flows". In: *IEEE Transactions on Automatic Control* 47.6 (June 2002), pp. 945–959. DOI: 10.1109/TAC.2002.1008360.

[97] C. Hollot, V. Misra, D. Towsley, and W. Gong. "On Designing Improved Controllers for AQM Routers Supporting TCP Flows". In: *Proc. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society Conference on Computer Communications (INFOCOM 2001)*. (Anchorage, Alaska, USA). Vol. 3. April 2001, pp. 1726–1734. DOI: 10.1109/INFCOM.2001.916670.

[98] L. Hu and A. Kshemkalyani. "HRED: a Simple and Efficient Active Queue Management Algorithm". In: *Proc. 13th International Conference on Computer Communications and Networks*. (Chicago, Illinois, USA). October 2004, pp. 387–393. DOI: 10.1109/ICCCN.2004.1401681.

[99] Information Sciences Institute (ISI), University of South California. *The Network Simulator – ns-2*. URL: http://www.isi.edu/nsnam/ns.

[100] J. Iyengar and I. Swett. *QUIC Loss Detection and Congestion Control*. Internet Draft. Work in progress. Internet Society, October 2018.

[101] J. Iyengar and M. Thompson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet Draft. Work in progress. Internet Society, October 2018.

[102] V. Jacobson. "Congestion Avoidance and Control". In: *Proc. ACM Symposium on Communications Architectures and Protocols (SIGCOMM '88)*. (Stanford, California, USA). August 1988, pp. 314–329. DOI: 10.1145/52324.52356.

[103] V. Jacobson. "Kathie Nichols' CoDel". In: *Proc. 84th Internet Engineering Task Force (IETF-84) meeting (tsvarea)*. (Vancouver, British Columbia, Canada). July 2012.

[104] V. Jacobson. "Kathie Nichols' CoDel". In: *The 84th Internet Engineering Task Force (IETF-84) meeting (tsvarea)*. (Vancouver, British Columbia, Canada). Audio Recording. July 2012. URL: http://www.ietf.org/audio/ietf84/ietf84-regencyd-20120730-1540-pm2.mp3.

[105] H. Jiang, Z. Liu, Y. Wang, K. Lee, and I. Rhee. "Understanding Bufferbloat in Cellular Networks". In: *Proc. 2012 ACM SIGCOMM Workshop on Cellular Networks: Operations, Challenges, and Future Design (CellNet '12)*. (Helsinki, Finland). August 2012, pp. 1–6. DOI: 10.1145/2342468.2342470.

[106] C. Jin, D. Wei, S. Low, J. Bunn, H. Choe, J. Doylle, H. Newman, S. Ravot, S. Singh, F. Paganini, G. Buhrmaster, L. Cottrell, O. Martin, and W. Feng. "FAST TCP: From Theory to Experiments". In: *IEEE Network* 19.1 (January 2005), pp. 4–11. DOI: 10.1109/MNET.2005.1383434.

[107] I. Johansson. *TCP HyStart patch deployment*. Post on IETF tcpm Working Group Mailing List. May 2015. URL: https://www.ietf.org/mail-archive/web/tcpm/current/msg09675.html.

[108] D. Katabi, M. Handley, and C. Rohrs. "Congestion Control for High Bandwidth-Delay Product Networks". In: *Proc. 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '02)*. (Pittsburgh, Pennsylvania, USA). August 2002, pp. 89–102. DOI: 10.1145/633025.633035.

[109] N. Khademi, G. Armitage, M. Welzl, S. Zander, G. Fairhurst, and D. Ros. "Alternative Backoff: Achieving Low Latency and High Throughput with ECN and AQM". In: *Proc. IFIP Networking Conference (IFIP Networking 2017)*. (Stockholm, Sweden). June 2017, pp. 1–9. DOI: 10.23919/IFIPNetworking.2017.8264863.

[110] 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*. Tech. rep. CAIA-TR-150710A. Swinburne University of Technology, July 2015.

[111] N. Khademi, M. Welzl, G. Armitage, and G. Fairhurst. *TCP Alternative Backoff with ECN (ABE)*. RFC 8511. December 2018.

[112] V. Konda and J. Kaur. "RAPID: Shrinking the Congestion Control Timescale". In: *Proc. INFOCOM 2009*. (New York, New York, USA). April 2009, pp. 1–9. DOI: 10.1109/INFCOM.2009.5061900.

[113] D. Koutsonikolas and Y. Hu. "On the Feasibility of Bandwidth Estimation in Wireless Access Networks". In: *Wireless Networks* 17.6 (August 2011), pp. 1561–1580. DOI: 10.1007/s11276-011-0364-5.

[114] T. Koyama and K. Aoki. "Slow Start Algorithm for Mobile Broadband Networks Including Delay Unrelated to Network Congestion". In: *Proc. International Conference on Computing, Networking and Communications (ICNC 2015)*. (Garden Grove, California, USA). February 2015, pp. 148–152. DOI: 10.1109/ICCNC.2015.7069332.

[115] M. Kühlewind, S. Neuner, and B. Trammell. "On the State of ECN and TCP Options on the Internet". In: *Proc. 14th Passive and Active Measurement Conference (PAM 2013)*. (Hong Kong). March 2013, pp. 135–144. DOI: 10.1007/978-3-642-36516-4_14.

[116] M. Kühlewind, R. Scheffenegger, and B. Briscoe. *Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback*. RFC 7560. August 2015.

[117] M. Kühlewind and B. Briscoe. "Chirping for Congestion Control - Implementation Feasibility". In: *Proc. 8th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT 2010)*. (Lancaster, Pennsylvania, USA). November 2010.

[118] N. Kuhn and D. Ros. "Improving PIE's Performance over High-Delay Paths". In: *CoRR* abs/1602.00569 (February 2016).

[119] J. Kulik, R. Coutler, D. Rockwell, and C. Partridge. *A Simulation Study of Paced TCP*. Tech. rep. NASA CR-2000-209416 / E-12041. NASA / BBN Technologies, January 2000.

[120] S. Kunniyur and R. Srikant. "Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management". In: *ACM SIGCOMM Computer Communications Review* 31.4 (October 2001), pp. 123–134. DOI: 10.1145/964723.383069.

[121] S. Kunniyur. "AntiECN Marking: A Marking Scheme for High Bandwidth Delay Connections". In: *Proc. IEEE International Conference on Communications (ICC '03)*. (Anchorage, Alaska, USA). Vol. 1. May 2003, pp. 647–651. DOI: 10.1109/ICC.2003.1204255.

[122] T. Lakshman, A. Neidhardt, and T. Ott. "The Drop from Front Strategy in TCP and in TCP over ATM". In: *Proc. Conference on Computer Communications (INFOCOM '96)*. (San Francisco, California, USA). Vol. 3. March 1996, pp. 1242–1250. DOI: 10.1109/INFCOM.1996.493070.

[123] K. Lakshminarayanan, V. Padmanabhan, and J. Padhye. "Bandwidth Estimation in Broadband Access Networks". In: *Proc. 4th ACM SIGCOMM Conference on Internet Measurement (IMC '04)*. (Taormina, Sicily, Italy). October 2004, pp. 314–321. DOI: 10.1145/1028788.1028832.

[124] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W. Chang, and Z. Shi. "The QUIC Transport Protocol: Design and Internet-Scale Deployment". In: *Proc. Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. (Los Angeles, California, USA). August 2017, pp. 183–196. DOI: 10.1145/3098822.3098842.

[125] S. Lederer, C. Müller, and C. Timmerer. "Dynamic Adaptive Streaming over HTTP Dataset". In: *Proc. 3rd Multimedia Systems Conference (MMSys '12)*. (Chapel Hill, North Carolina, USA). February 2012, pp. 89–94. DOI: 10.1145/2155555.2155570.

[126] X. Li and H. Yousefi'zadeh. "An Implementation and Experimental Study of the Variable-Structure Congestion Control Protocol (VCP)". In: *Proc. IEEE Military Communications Conference (MILCOM 2007)*. (Orlando, Florida, USA). October 2007, pp. 1–7. DOI: 10.1109/MILCOM.2007.4455186.

[127] D. Liu, M. Allman, S. Jin, and L. Wang. "Congestion Control Without a Startup Phase". In: *Proc. Fifth International Workshop on Protocols for FAST Long-Distance Networks*. (Marina Del Rey (Los Angeles), California, USA). February 2007.

[128] X. Liu, K. Ravindran, and D. Loguinov. "Multi-Hop Probing Asymptotics in Available Bandwidth Estimation: Stochastic Analysis". In: *Proc. 5th ACM SIGCOMM Conference on Internet Measurement (IMC' 05)*. (Berkeley, California, USA). October 2005, pp. 173–186.

[129] M. Mathis and B. Briscoe. *Congestion Exposure (ConEx) Concepts, Abstract Mechanism, and Requirements*. RFC 7713. December 2015.

[130] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. *TCP Selective Acknowledgment Options*. RFC 2018. October 1996.

[131] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm". In: *ACM SIGCOMM Computer Communications Review* 27.3 (July 1997), pp. 67–82. DOI: 10.1145/263932.264023.

[132] M. McGregor and W. Shi. "Deficits for Bursty Latency-Critical Flows: DRR++". In: *Proc. IEEE International Conference on Networks (ICON 2000)*. (Singapore). September 2000, pp. 287–293. DOI: 10.1109/ICON.2000.875803.

[133] P. McKenney. "Stochastic Fairness Queueing". In: *Proc. Conference on Computer Communications, Ninth Annual Joint Conference of the IEEE Computer and Communications Societies, The Multiple Facets of Integration (INFOCOM '90)*. (San Francisco, California, USA). Vol. 2. June 1990, pp. 733–740.

[134] D. Mendes, G. Senges, G. Santos, G. Mendonça, R. Leão, and E. Silva. "A Preliminary Performance Measurement Study of Residential Broadband Services in Brazil". In: *Proc. 2016 Workshop on Fostering Latin-American Research in Data Communication Networks (LANCOMM '16)*. (Florianópolis, Brazil). August 2016, pp. 16–18. DOI: 2940116.2940135.

[135] J. Misund. "Rapid Acceleration in TCP Prague". MA thesis. University of Oslo, Department of Informatics, May 2018.

[136] J. Misund and B. Briscoe. "Flow-start: Faster and Less Overshoot with Paced Chirping". In: *Proc. 102nd Internet Engineering Task Force (IETF-102) meeting (iccrg)*. (Montreal, Quebec, Canada). July 2018.

[137] J. Mo, R. La, V. Anantharam, and J. Walrand. "Analysis and Comparison of TCP Reno and Vegas". In: *Proc. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM '99)*. (New York, New York, USA). March 1999, pp. 1556–1563. DOI: 10.1109/INFCOM.1999.752178.

[138] P. Mockapetris. *Domain names - concepts and facilities*. RFC 1034. November 1987.

[139] P. Mockapetris. *Domain names - implementation and specification*. RFC 1035. November 1987.

[140] J. Nagle. *Congestion Control in IP/TCP Internetworks*. RFC 896. January 1984.

[141] *NFSNET: A Partnership for High-Speed Networking*. Final Report. 1995. URL: https://web.archive.org/web/20100528000953/http://www.merit.edu/about/history/pdf/NSFNET_final.pdf.

[142] K. Nichols and V. Jacobson. "Controlling Queue Delay". In: *ACM Queue* 10.5 (May 2012), 20:20–20:34. DOI: 10.1145/2208917.2209336.

[143] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar. *Controlled Delay Active Queue Management*. RFC 8289. January 2018.

[144] L. Nussbaum. *Make CUBIC Hystart more robust to RTT variations*. Proposed Linux kernel modification (not mainlined). March 2011. URL: http://patchwork.ozlabs.org/patch/85945/.

[145] R. Pan, P. Natarajan, F. Baker, and G. White. *Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem*. RFC 8033. February 2017.

[146] 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: *Proc. 2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR 2013)*. (Taipei, Taiwan (ROC)). July 2013. DOI: 10.1109/HPSR.2013.6602305.

[147] D. Papadimitriou, M. Welzl, M. Scharf, and B. Briscoe. *Open Research Issues in Internet Congestion Control*. RFC 6077. February 2011.

[148] V. Paxson, M. Allman, J. Chu, and M. Sargent. *Computing TCP's Retransmission Timer*. RFC 6298. June 2011.

[149] R. Peon and W. Chan. *SPDY Essentials*. Google Tech Talk. December 2011.

[150] E. Plasser, T. Ziegler, and P. Reichl. "On the Non-Linearity of the RED Drop Function". In: *Proc. 15th International Conference on Computer Communication (ICCC '02)*. (Mumbai, Maharashtra, India). August 2002, pp. 515–534.

[151] J. Postel. *Transmission Control Protocol*. RFC 793. September 1981.

[152] J. Postel. *User Datagram Protocol*. RFC 768. August 1980.

[153] R. Pries, Z. Magyari, and P. Tran-Gia. "An HTTP Web Traffic Model Based on the Top One Million Visited Web Pages". In: *Proc. 8th EURO-NGI Conference on Next Generation Internet (NGI 2012)*. (Karlskrona, Sweden). June 2012, pp. 133–139. DOI: 10.1109/NGI.2012.6252145.

[154] R. Pries, F. Wamser, D. Staehle, K. Heck, and P. Tran-Gia. "On Traffic Characteristics of a Broadband Wireless Internet Access". In: *Proc. 5th Euro-NGI Conference on Next Generation Internet Networks (NGI 2009)*. (Aveiro, Portugal). July 2009, pp. 184–190. DOI: 10.1109/NGI.2009.5175772.

[155] S. Ramachandran. *Web metrics: Size and number of resources*. May 2010. URL: http://web.archive.org/web/20140625155405/https://developers.google.com/speed/articles/web-metrics.

[156] K. Ramakrishnan, S. Floyd, and D. Black. *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168. September 2001.

[157] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. August 2018.

[158] S. Rewaskar, J. Kaur, and D. Smith. *Why Don't Delay-based Congestion Estimartors Work in the Real-world?* Tech. rep. TR06-001. Department of Computer Science, University of North Carolina at Chapel Hill, January 2006.

[159] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger. *CUBIC for Fast Long-Distance Networks*. RFC 8312. February 2018.

[160] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. "pathChirp: Efficient Available Bandwidth Estimation for Network Paths". In: *Passive and Active Measurement Workshop (PAM 2003)*. (San Diego, California, USA). April 2003.

[161] V. Roman. *TCP HyStart patch deployment*. Post on IETF tcpm Working Group Mailing List. October 2015. URL: https://www.ietf.org/mail-archive/web/tcpm/current/msg09877.html.

[162] V. Rosolen, O. Bonaventure, and G. Leduc. "A RED Discard Strategy for ATM Networks and Its Performance Evaluation with TCP/IP Traffic". In: *ACM SIGCOMM Computer Communications Review* 29.3 (July 1999). DOI: 10.1145/505724.505728.

[163] R. Sallantin, C. Baudoin, F. Arnal, E. Dubois, E. Chaput, and A. Beylot. *Safe Increase of the TCP's Initial Window Using Initial Spreading*. Internet Draft. Work in progress. Internet Society, January 2014.

[164] R. Sallantin, C. Baudoin, E. Chaput, F. Arnal, E. Dubois, and A. Beylot. "Initial Spreading: A Fast Start-Up TCP Mechanism". In: *Proc. 38th Annual IEEE Conference on Local Computer Networks (LCN 2013)*. (Sydney, New South Wales, Australia). October 2013, pp. 492–499. DOI: 10.1109/LCN.2013.6761283.

[165] M. Sargent, E. Blanton, and M. Allman. "Modern Application Layer Transmission Patterns from a Transport Perspective". In: *Proc. Passive and Active Measurement: 15th International Conference (PAM 2014)*. (Los Angeles, California, USA). March 2014, pp. 141–150. DOI: 10.1007/978-3-319-04918-2_14.

[166] M. Sargent, B. Stack, T. Dooner, and M. Allman. *A First Look at 1 Gbps Fiber-To-The-Home Traffic*. Tech. rep. 12-009. International Computer Science Institute, August 2012.

[167] P. Sarolahti, M. Allman, and S. Floyd. "Determining an Appropriate Sending Rate over an Underutilized Network Path". In: *Computer Networks* 51.7 (May 2007), pp. 1815–1832. DOI: 10.1016/j.comnet.2006.11.006.

[168] P. Sarolahti, J. Korhonen, L. Daniel, and M. Kojo. "Using Quick-Start to Improve TCP Performance with Vertical Hand-offs". In: *Proc. 31st IEEE Conference on Local Computer Networks (LCN 2006)*. (Tampa, Florida, USA). November 2006, pp. 897–904. DOI: 10.1109/LCN.2006.322197.

[169] R. Sayre. *Change max-persistent-connections-per-server to 6 (wedidntstartthefire)*. Feature Request. March 2008. URL: https://bugzilla.mozilla.org/show_bug.cgi?id=423377.

[170] M. Scharf. "Fast Startup Internet Congestion Control Mechanisms for Broadband Interactive Applications". PhD thesis. Institut für Kommunikationsnetze und Rechnersysteme der Universität Stuttgart, April 2011.

[171] M. Scharf. "Quick-Start, Jump-Start, and Other Fast Startup Approaches". In: *Proc. 73rd Internet Engineering Task Force (IETF-73) meeting (iccrg)*. November 2008.

[172] M. Scharf. "Comparison of End-to-end and Network-supported Fast Startup Congestion Control Schemes". In: *Computer Networks* 55.8 (June 2011), pp. 1921–1940. DOI: 10.1016/j.comnet.2011.02.002.

[173] M. Scharf. "Performance analysis of the Quick-Start TCP extension". In: *Proc. Fourth International Conference on Broadband Communications, Networks, and Systems (IEEE BROADNETS 2007)*. (Raleigh, North Carolina, USA). September 2007, pp. 942–951. DOI: 10.1109/BROADNETS.2007.4550538.

[174] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550. July 2003.

[175] M. Shreedhar and G. Varghese. "Efficient Fair Queueing Using Deficit Round Robin". In: *IEEE/ACM Transactions on Networking* 4.3 (June 1996), pp. 375–385. DOI: 10.1109/90.502236.

[176] M. Siekkinen, D. Collange, G. Urvoy-Keller, and E. Biersack. *Application-Level Performance of ADSL Users*. Research Report RR-06-178. Institut Eurécom, October 2006.

[177] M. Siekkinen, D. Collange, G. Urvoy-Keller, and E. Biersack. "Performance Limitations of ADSL Users: A Case Study". In: *Proc. 8th International Conference on Passive and Active Network Measurement (PAM '07)*. (Louvain-la-Neuve, Belgium). April 2007, pp. 145–154. DOI: 10.1007/978-3-540-71617-4_15.

[178] S. Souders. *Roundup on Parallel Connections*. March 2008. URL: http://www.stevesouders.com/blog/2008/03/20/roundup-on-parallel-connections/.

[179] *SPDY: An Experimental Protocol for a Faster Web*. URL: http://www.chromium.org/spdy/spdy-whitepaper.

[180] M. Sridharan, K. Tan, D. Bansal, and D. Thaler. *Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks*. Internet Draft. Work in progress. Internet Society, November 2008.

[181] R. Srinivasan, J. Zhuang, L. Jalloul, R. Novak, and J. Park. *IEEE 802.16m Evaluation Methodology Document (EMD)*. July 2008.

[182] T. Stockhammer. "Dynamic Adaptive Streaming over HTTP — Standards and Design Principles". In: *Proc. Second Annual ACM Conference on Multimedia Systems (MMSys '11)*. (San Jose, California, USA). February 2011, pp. 133–144. DOI: 10.1145/1943552.1943572.

[183] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. "Measuring Home Broadband Performance". In: *Communications of the ACM* 55.11 (November 2012), pp. 100–109. DOI: 10.1145/2366316.2366337.

[184] K. Tan, J. Song, Q. Zhang, and M. Sridharan. "A Compound TCP Approach for High-Speed and Long Distance Networks". In: *Proc. 25th International Conference on Computer Communications (IEEE INFOCOM 2006)*. (Barcelona, Spain). April 2006, pp. 1–12. DOI: 10.1109/INFOCOM.2006.188.

[185] K. Thompson, G. J. Miller, and R. Wilder. "Wide-Area Internet Traffic Patterns and Characteristics". In: *IEEE Network* 11.6 (November 1997), pp. 10–23. DOI: 10.1109/65.642356.

[186] J. Touch. *TCP Control Block Interdependence*. RFC 2140. April 1997.

[187] I. Tsang, M. Rajiullah, G. Fairhurst, A. Petlund, R. Secchi, B. Briscoe, A. Petras, K. De Schepper, and O. Bondarenko. *Deployment of RITE Mechanisms in Use-Case Trial Testbeds Report*. Project Report. January 2016.

[188] S. Turner and T. Polk. *Prohibiting Secure Sockets Layer (SSL) Version 2.0*. RFC 6176. March 2011.

[189] C. Villamizar and C. Song. "High Performance TCP in ANSNET". In: *ACM SIGCOMM Computer Communications Review* 24.5 (October 1994), pp. 45–60. DOI: 10.1145/205511.205520.

[190] W3C. *HTML 4.01 Specification*. December 1999.

[191] W3C. *HTML5.1*. November 2016.

[192] H. Wang, H. Xin, D. Reeves, and K. Shin. "A Simple Refinement of Slow-Start of TCP Congestion Control". In: *Proc. Fifth IEEE Symposium on Computers and Communications (ISCC 2000)*. (Antibes-Juan Les Pins, France). July 2000, pp. 98–105. DOI: 10.1109/ISCC.2000.860615.

[193] D. Wei, C. Jin, S. Low, and S. Hegde. "FAST TCP: Motivation, Architecture, Algorithms, Performance". In: *IEEE/ACM Transactions on Networking* 14.6 (December 2006), pp. 1246–1259. DOI: 10.1109/TNET.2006.886335.

[194] G. White. "Active Queue Management in DOCSIS 3.1 Networks". In: *IEEE Communications Magazine* 53.3 (March 2015), pp. 126–132. DOI: 10.1109/MCOM.2015.7060493.

[195] G. White and R. Pan. *Active Queue Management (AQM) Based on Proportional Integral Controller Enhanced PIE) for Data-Over-Cable Service Interface Specifications (DOCSIS) Cable Modems*. RFC 8034. February 2017.

[196] G. White. *Active Queue Management in DOCSIS 3.x Cable Modems*. White paper. CableLabs, May 2014.

[197] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman. "One More Bit Is Enough". In: *ACM SIGCOMM Computer Communications Review* 35.4 (December 2005), pp. 37–48. DOI: 10.1145/1090191.1080098.

[198] J. Yang, L. Yuan, C. Dong, G. Cheng, N. Ansari, and N. Kato. "On Characterizing Peer-to-Peer Streaming Traffic". In: *IEEE Journal on Selected Areas in Communications* 31.9 (September 2013), pp. 175–188. DOI: 10.1109/JSAC.2013.SUP.0513016.

[199] Q. Yin, J. Kaur, and F. Smith. "TCP Rapid: From Theory to Practice". In: *Proc. IEEE Conference on Computer Communications (INFOCOM 2017)*. (Atlanta, Georgia, USA). May 2017, pp. 1–9. DOI: 10.1109/INFOCOM.2017.8057179.

[200] M. Zhang, J. Wu, C. Lin, and K. Xu. "Rethink the Tradeoff Between Proportional Controller and PI Controller". In: *Proc. Seventh International Symposium on Computers and Communications (ISCC 2002)*. (Taormina-Giardini Naxos, Italy). July 2002, pp. 57–62. DOI: 10.1109/ISCC.2002.1021658.