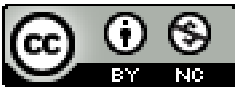
PACKET TRANSPORT MECHANISMS FOR DATA CENTER NETWORKS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Mohammadreza Alizadeh Attar
September 2013

© 2013 by Mohammadreza Alizadeh Attar. All Rights Reserved.
Re-distributed by Stanford University under license with the author.



This work is licensed under a Creative Commons Attribution-Noncommercial 3.0 United States License.
http://creativecommons.org/licenses/by-nc/3.0/us/

This dissertation is online at: http://purl.stanford.edu/wz416gf3809

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Balaji Prabhakar, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Nick McKeown, Co-Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Sachin Katti

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

Data centers are large facilities that house tens of thousands of interconnected computing and storage servers. In recent years, large investments have been made in massive data centers to support services such as search, social networking, electronic commerce, and cloud computing. This has spurred a lot of interest in the industry and the research community in innovation for reducing costs and improving the performance of data center networks.

One of the most crucial components of a data center network (indeed, of any network) is its transport mechanism — the method by which data is transferred from one server to another. The major goals of a transport mechanism are to transfer data at the highest possible rate and with the lowest latency. In this dissertation, we present measurements from three production clusters with 6000 servers that reveal impairments with today's state-of-the-art Transmission Control Protocol (TCP) in data center environments. The impairments are rooted in TCP's demands on the limited buffer space available in commodity data center switches. We then propose two novel transport mechanisms designed specifically for the data center: DCTCP and HULL. These mechanisms enable the construction of large-scale, high-performance data center networks with very low latency and high bandwidth using commodity switches. DCTCP (for Data Center TCP) uses a simple modification to the TCP congestion control algorithm that allows it to maintain very low queue occupancy (and latency) in data center switches while simultaneously providing high throughput. HULL (for High-bandwidth Ultra-Low Latency) is an architecture that builds on DCTCP to deliver baseline fabric latency (only propagation and switching) by nearly eliminating all queueing from the data center network. We also present a stability analysis of the Quantized Congestion Notification (QCN) algorithm that has been standardized as the IEEE802.1Qau standard for Layer 2 (Ethernet) congestion control.

To my parents, Marzieh and Alireza

Acknowledgements

Working towards a Ph.D. has been a deeply enriching experience; at times it's been elating, at times depressing, but it has always been very rewarding. Looking back, many people have helped shape this journey. I would like to extend them my thanks.

First and foremost, my advisor, Balaji Prabhakar. I don't know where to start and how to thank him. My work would not have been possible without his constant guidance — like when he nudged me into considering systems research more seriously — his unwavering encouragement, his many insights and ideas, and his exceptional resourcefulness. And most importantly, his friendship. I have been very fortunate to have an advisor who has also been a close friend. For all of this, Balaji, thank you.

I would also like to thank Nick McKeown and Sachin Katti. Nick has always had his door open to me throughout the years and I have benefited greatly from observing how he dissects research problems. My collaboration with Sachin on the pFabric project has been one of the most fruitful and fun engagements I have experienced. I have learned a lot writing two papers with him, witnessing how he navigates between high level concepts and low level details, and is unfazed by time pressure.

I am also grateful to Mendel Rosenblum and Mohsen Bayati for agreeing to be on my defense committee on short notice.

I have been very fortunate to work closely with some wonderful colleagues and peers at Stanford. I thank my dear friends, Abdul Kabbani and Adel Javanmard, whom I've enjoyed collaborating with on multiple projects. I also wish to thank Vimalkumar Jeyakumar, Berk Atikoglu, Shuang Yang, and Tom Yue from the Balagroup, and Masato Yasuda, who put in a tremendous effort to make the HULL paper a reality.

I have also been fortunate to have many fantastic collaborators outside Stanford. I

would like to thank Albert Greenberg, Murari Sridharan, Jitu Padhye, Dave Maltz, and Parveen Patel for all their support and guidance during my two internships at Microsoft. Without their help, the DCTCP project would not have materialized, and much of this dissertation would not have been possible. I would also like to thank Amin Vahdat for all his help during the HULL project, and his advice and counsel since. I am very grateful to Tom Edsall, Sundar Iyer, and Yi Lu for the always stimulating conversations and their encouragement and support.

I am deeply indebted to Mr. Riazi, my physics teacher in the Dr. Hesabi High School in Ahvaz, Iran. His passion for science and scholarly pursuit and his dedication to his craft has been an inspiration to me (and surly, many of his other students) and helped set me on the path on which I find myself today.

Last, but not least, I would like to thank my family. My brother, Mehdi, my sister-in-law, Narges, and my mother and father in-laws, I am very blessed to have you in my life. My beloved Tahereh, we began this journey together, two kids; we have come a long way. Words cannot express how grateful I am for your selfless love and the joy you bring to my life. And my parents, Marzieh and Alireza, you raised me, loved me, taught me right from wrong, and gave me all that I needed. This is dedicated to you.

Contents

Abstract **iv**

Acknowledgements **vi**

1 Introduction **1**

1.1 Overview & Organization 5
1.1.1 Data Center TCP (DCTCP) 5
1.1.2 Ultra-Low Latency Packet Transport (HULL) 6
1.1.3 Stability Analysis of QCN 7
1.2 Contributions 8
1.3 Previously Published Material 9

2 Data Center TCP (DCTCP) **11**

2.1 Communications in Data Centers 14
2.1.1 Partition/Aggregate Application Structure 14
2.1.2 Workload Characterization 16
2.1.3 Understanding Performance Impairments 19
2.2 The DCTCP Algorithm 23
2.2.1 Algorithm 25
2.2.2 Benefits 27
2.2.3 Preliminary Analysis 28
2.2.4 Discussion 33
2.3 Results 36
2.3.1 DCTCP Performance 36

2.3.2 Impairment Microbenchmarks 40
2.3.3 Benchmark Traffic 46
2.4 Related Work 52
2.5 Final Remarks 54

3 Analysis of DCTCP 55
3.1 DCTCP Fluid Model 58
3.1.1 Comparison with Packet-level Simulations 59
3.2 Steady State Analysis 62
3.2.1 The Normalized Fluid Model 62
3.2.2 Stability of Limit Cycle 63
3.2.3 Steady State Throughput & Delay 67
3.3 Convergence Analysis 73
3.3.1 The Hybrid Model 74
3.3.2 Rate of Convergence 75
3.3.3 Simulations 78
3.3.4 Proof of Theorem 3.2 79
3.4 RTT-fairness 81
3.4.1 Simulations 82
3.4.2 Improving DCTCP's RTT-fairness 83
3.5 Final Remarks 85

4 Ultra-Low Latency Packet Transport 87
4.1 Design Overview 91
4.1.1 Phantom Queues: Detecting and Signaling Congestion 91
4.1.2 DCTCP: Adaptive Reaction to ECN 92
4.1.3 Packet Pacing 93
4.1.4 The HULL Architecture 94
4.1.5 QoS as the Benchmark 95
4.2 Bandwidth Headroom 95
4.2.1 Importance of Stable Rate Control 95
4.2.2 Slowdown Due to Bandwidth Headroom 97

4.3 Packet Pacing . . . 98
4.3.1 Burstiness of Traffic . . . 98
4.3.2 Hardware Pacer Module . . . 101
4.3.3 Effectiveness of the Pacer . . . 102
4.3.4 Tradeoff Between Pacer Delay and Effectiveness . . . 104
4.4 Experimental Setup . . . 105
4.4.1 Implementation . . . 105
4.4.2 Testbed . . . 106
4.4.3 Parameter Choices . . . 107
4.5 Results . . . 108
4.5.1 Static Flow Experiments . . . 109
4.5.2 Dynamic Flow Experiments . . . 112
4.5.3 Large-scale ns-2 Simulation . . . 117
4.6 Related Work . . . 120
4.7 Final Remarks . . . 122

5 Stability of Quantized Congestion Notification 123
5.1 Layer 2 Congestion Control . . . 125
5.2 The QCN Algorithm . . . 128
5.2.1 The CP Algorithm . . . 128
5.2.2 The RP Algorithm . . . 129
5.3 The QCN Fluid Model . . . 131
5.3.1 Model validation . . . 134
5.4 Stability Analysis of the Fluid Model . . . 134
5.4.1 Averaging in QCN . . . 137
5.4.2 Simulations . . . 139
5.5 The Averaging Principle . . . 141
5.5.1 Description of the AP . . . 142
5.5.2 Analysis . . . 145
5.5.3 The AP applied to another congestion control algorithm: RCP . . . 150
5.6 Buffer Sizing . . . 152

5.7 Related Work 154
5.8 Final Remarks 156

6 Conclusion **157**
6.1 Summary 157
6.2 Future Directions 159

A Proof of Theorem 3.1 **161**

B Proof of Proposition 3.1 **164**

C Bounds on α^* **168**

D Characteristic Equation of Linearized QCN Fluid Model **170**

E Stability Analysis of the QCN-AIMD Fluid Model **171**

F Proof of Theorem 5.3 **173**

Bibliography **176**

List of Tables

2.1 Switches in our testbed. 36
2.2 95th percentile of query completion time with and without buffer pressure from background traffic. DCTCP prevents background traffic from affecting performance of query traffic. $RTO_{min} = 10ms$, $K = 20$. 46

4.1 Baseline throughput and latency for two long-lived flows with TCP and DCTCP-30K (30KB marking threshold). 96
4.2 The impact of interrupt coalescing. Note that 'adaptive' is the default setting for interrupt coalescing. 100
4.3 Baseline parameter settings for testbed experiments. 107
4.4 Baseline dynamic flow experiment. The average, 90th percentile, and 99th percentile switch latency and flow completion times are shown. The results are the average of 10 trials. In each case, the best scheme is shown in red. . . 113
4.5 HULL vs QoS with two priorities. The switch buffer is configured to use either dynamic buffer allocation (Large Buffer) or a fixed buffer of 10pkts = 15KB (Small Buffer). In all tests, the total load is 40%. In the case of QoS, the switch latency reported is that of the high priority queue. The results are the average of 10 trials. 116
4.6 Breakdown of flows and bytes according to flow size. 118

5.1 Comparison of the standard deviation of the sending rate of a single 10Gbps TCP and QCN source, for different RTTs. The standard deviation is an order of magnitude smaller with QCN because of its better stability. 154

List of Figures

1.1 Breakdown of global data center traffic by destination. The data is from the Cisco Global Cloud Index [2] and is based on projections for 2011–2016. . 3

1.2 Queue length measured on a Broadcom Triumph switch. Two long flows are launched from distinct 1Gbps ports to a common 1Gbps port. The switch has dynamic memory management enabled, allowing the congested port to dynamically grab up to $\sim$600KB of buffer. The experiment shows that DCTCP maintains a very small buffer occupancy compared to TCP, without compromising throughput. Note that both schemes achieve full throughput since the buffer never underflows. 6

2.1 The Partition/Aggregate application structure. Requests from higher layers of the application are broken into pieces and farmed out to workers in the lower layers, whose responses are aggregated to produce the result. 14

2.2 Time between arrival of new work for the Aggregator (queries) and between background flows between servers (update and short messages). . . . 17

2.3 The PDF and CDF of flow size distribution for background traffic. The Total Bytes represents the probability a randomly selected byte belongs to a flow of given size. 17

2.4 Distribution of number of concurrent connections, across all flows (on the left) and across the flows that are larger than 1MB (on the right). 18

2.5 Three ways in which flows interact at a shared memory switch that result in performance problems. (a) Incast: many synchronized flows converge on the same interface leading to packet loss and retransmission timeouts; (b) Queue buildup: long and short flows traverse the same port with the long flows building up a large queue backlog, causing latency to the short flows; and (c) Buffer pressure: long flows on one port use up the shared buffer pool, reducing the available buffer to absorb bursts on the other ports. . . . 19
2.6 A real incast event measured in a production environment. Timeline shows queries forwarded over 0.8ms, with all but one response returning by 13.5ms. That response is lost, and is retransmitted after $RTO_{\min}$ (300ms). RTT+Queue estimates queue length on the port to the aggregator. 20
2.7 Response time percentiles for a production application having the incast traffic pattern. Forwarded requests were jittered (deliberately delayed) over a 10ms window until 8:30am, when jittering was switched off. The 95th and lower percentiles drop 10x, while the 99.9th percentile doubles. 21
2.8 CDF of RTT to the aggregator. 10% of responses see an unacceptable queuing delay of 1 to 14ms caused by long flows sharing the queue. 23
2.9 DCTCP's AQM scheme is a variant of RED: Low and High marking thresholds are equal, and marking is based on the instantaneous queue length. . . 25
2.10 Two state ACK generation state machine used by the DCTCP receiver. . . . 26
2.11 Sawtooth window size process for a single DCTCP sender, and the associated queue size process. 29
2.12 Comparison between the queue size process predicted by the analysis with ns2 simulations. The DCTCP parameters are set to $K = 40$ packets, and $g = 1/16$. 31
2.13 DCTCP and TCP/PI with 2 and 20 long-lived flows. The flows share a 10Gbps link and have a RTT of 500μs. The switch buffer size is 100 packets. The DCTCP marking threshold (K) and the PI reference queue length (q_{ref}) are both set to 40 packets. 34

2.14 Throughput and queue length for TCP and DCTCP with long-lived traffic. (a) CDF of queue length for 2 and 20 flows at 1Gbps. (b) Throughput with different marking thresholds, K, for 2 flows at 10Gbps. 37
2.15 DCTCP versus RED at 10Gbps. RED causes much larger queue length oscillations and requires more than twice the buffer of DCTCP to achieve the same throughput. 38
2.16 Convergence experiment. Five flows sequentially start and the stop, spaced by 30 seconds. 39
2.17 Multi-hop topology. The 10Gbps link between Triumph 1 and Scorpion and the 1Gbps link between Triumph 2 and R1 are both bottlenecks. 40
2.18 Incast performance with a static 100 packet buffer at each switch port. DCTCP performs significantly better than TCP and only begins suffering from timeouts when the number of senders exceeds 35. The values shown are the mean for 1000 queries, with each query totaling 1MB of data. The 90% confidence intervals are too small to be visible. 41
2.19 Incast performance with with dynamic buffer allocation at the switch. DCTCP does not suffer problems, even with high degrees of incast. 42
2.20 All-to-all incast. 41 servers simultaneously participate in 40-to-1 incast patterns. DCTCP along with dynamic buffer allocation work well. 43
2.21 All-to-all incast with deep-buffered (CAT4948) and shallow-buffered (Triumph) switches. Deep buffers prevent incast induced timeouts for the 1MB response size, but the problem reappears with 10MB responses. DCTCP handles the incast well in both cases even with the shallow-buffered switch. All schemes use RTO_{min} = 10ms. 44
2.22 Queue buildup caused by large flows significantly increases the completion time of short transfers for TCP. With DCTCP, the short transfers complete quickly nonetheless because the queue occupancy in the switch is kept small. 45
2.23 Completion time of background traffic in cluster benchmark. Note the log scale on the Y axis. 47
2.24 Completion time of query traffic in cluster benchmark. 48

2.25 95th percentile completion time for short messages and query traffic in the scaled cluster benchmark with 10x background and 10x query traffic. 49
2.26 95th percentile of completion time for the cluster benchmark scaled by (a) increasing the background flow size by 10x; (b) increasing the query response size by 10x. 51
2.27 Fraction of queries that suffer at least one timeout with (a) increasing background flow size; (b) increasing query response size. Note the different Y axis ranges for the two plots. 51

3.1 Comparison between the DCTCP fluid model and ns2 simulations. The parameters are: $C = 10$Gbps, $d = 100\mu$s, $K = 65$ packets, and $g = 1/16$. The fluid model results are shown in solid black. 60
3.2 Comparison of the fluid and Sawtooth models with ns2 simulations for $N = 2$ sources and $g = 0.4$. The fluid model is much more accurate when g is not very small. 61
3.3 Phase diagram showing occurrence of limit cycles in the (normalized) DCTCP fluid model for $\bar{w} = 10$, $g = 1/16$ (the projection on the $\tilde{W}$–$\tilde{q}$ plane is shown). 64
3.4 Periodic system trajectory and the Poincaré map. 65
3.5 Limit cycle stability (Theorem 3.1). Stability is verified numerically for $g \in [0.001, 1]$, and $\bar{w} \in [2.01, 1000]$. 67
3.6 Queue undershoot and overshoot. The values are found by numerically computing the limit cycles of the (normalized) DCTCP fluid model for a range of $\bar{w}$ and g. 68
3.7 Period of the DCTCP limit cycle. For small g, the period grows with $\sqrt{\bar{w}}$. . 70
3.8 Worst case throughput vs delay. DCTCP achieves more than 94% throughput for $g < 0.1$ even as $\psi \to 0$ (equivalently, $K \to 0$). 71
3.9 Throughput vs marking threshold, K, in ns2 simulations. K is varied from 4 to 100 packets (1–25% of the bandwidth-delay product). As predicted by the analysis, the throughput with DCTCP remains higher than 94% even for K as small as 1% of the bandwidth-delay product. In contrast, the throughput for TCP with ECN marking at low thresholds approaches 75%. 72

3.10 Hybrid Model. The window sizes are shown for 2 flows. Note that $\sum W_i(T_k) = W_{max}$ for all k. 74
3.11 ns2 simulation of convergence time for 2 flows. The chosen g values for DCTCP correspond to: (b) $1/\sqrt{W_{max}}$, (c) $5/W_{max}$, and (d) $1/W_{max}$. 78
3.12 RTT-fairness in ns2 simulation. Two groups each with two flows are activated. Flows in group 1 have $RTT_1 = 100\mu$s, while the RTT for flows in group 2 (RTT_2) is varied from 100μs to 1ms. Note the log-log scale. The plot confirms that with the simple change to the DCTCP window update rule in Section 3.4.2, it achieves linear RTT-fairness. 83

4.1 The HULL architecture consists of phantom queues at switch egress ports and DCTCP congestion control and packet pacers at end-hosts. 94
4.2 Throughput (left) and average switch latency (right) for TCP-ECN and DCTCP with a PQ, as drain rate varies. The vertical bars in the right plot indicate the 99th percentile. 96
4.3 Burstiness with 10Gbps NICs. Every ∼0.5ms, a burst of packets totaling 65KB is sent out at line rate to obtain an average rate of ∼1Gbps. 99
4.4 Block diagram of the Pacer module. 101
4.5 Throughput and switch latency as PQ drain rate varies, with and without pacing. 103
4.6 The end-to-end RTT for paced flows, average switch latency, and overall throughput, as the parameter β is varied. The vertical bars indicate the 99th percentile. β controls how aggressively the Pacer reacts to queue buildup in the rate-limiter, with larger values reducing the backlog but also causing more bursting. The experiment shows that there is a tradeoff between the latency induced by the Pacer and its effectiveness. 104
4.7 Experimental HULL testbed. 10 servers are connected to a Broadcom Triumph switch through intermediate NetFPGA devices that implement Pacers and PQs. An additional NetFPGA (NF6) implements a Latency Measurement Module for measuring the switch latency. 106

4.8 Switch latency (left) and throughput (right) as the number of long-lived flows varies. The latency plot uses a logarithmic scale and shows the average latency, with the vertical bars showing the 99th percentile. 110
4.9 The impact of increasing the number of flows on average and 99th percentile switch latency with PQ, for MTU = 1500 bytes and MTU = 300 bytes. The smaller MTU is used to emulate a link running at 5Gbps. The PQ drain rate is set to 800Mbps and the marking threshold is 6KB. 111
4.10 Slowdown for 10MB flows: Theory vs Experiment. 114
4.11 Average switch latency (left) and 10MB FCT (right), with and without pacing. 115
4.12 Topology for ns-2 simulations: 10Gbps three-tier fat-tree with 192 servers organized in 8 pods. The topology has a 3:1 over-subscription at the TOR. . 118
4.13 Average and high percentile FCT for small flows. The statistics for the (0,10KB] and [10KB, 100KB) are shown separately. 119
4.14 Average FCT for large flows. The statistics for the [100KB,10MB) and [10MB, ∞) are shown separately. 120

5.1 The QCN CP randomly samples incoming packets and (if needed) sends a feedback message with a congestion metric to the source of the sampled packet in order to maintain the buffer occupancy near a desired operating point, Q_{eq}. 128
5.2 QCN RP operation. The current rate, R_C, is reduced multiplicatively upon a congestion message. The rate then increases in Fast Recovery in a series of jumps back towards the target rate, R_T (the rate before the decrease), and subsequently, undergoes Active Increase to probe for additional bandwidth. 130
5.3 Markov chain corresponding to current rate (R_C) increases at the QCN RP. 133
5.4 Comparison of QCN fluid model and ns2 simulation. The model matches the simulations well and accurately predicts rate and queue occupancy oscillations. 134
5.5 Rate dynamics of QCN and QCN-AIMD. QCN-AIMD does not do the averaging steps in Fast Recovery; it immediately enters Active Increase following a rate decrease. 138

5.6 Queue occupancy for QCN and QCN-AIMD with baseline parameters and RTT of (a) 50μs, (b) 200μs, and (c) 350μs. $N = 10$ sources share a single 10Gbps bottleneck. The desired operating point at the switch buffer, Q_{eq}, is set to 22 packets. The results validate the stability margins predicted for the two schemes by Theorems 5.1 and 5.2, and confirm that QCN has better stability than QCN-AIMD (Theorem 5.3). 140
5.7 A generic sampled control system. 142
5.8 Plant input signal generated by the standard controller and the AP controller. 143
5.9 Unit step responses for the standard and AP controllers with: (a) zero delay and (b) 8 second delay. The AP controller is more robust and keeps the system stable for larger feedback delays. 144
5.10 Equivalent PD control scheme to the AP controller. 146
5.11 Step response for AP and PD controllers for different feedback delays. . . 146
5.12 The block diagrams of two equivalent controllers. Controller 1 is the AP controller and Controller 2 is the PD controller (with an additional filter on the lower branch). 147
5.13 Total sending rate for RCP, AP-RCP and AP2-RCP with RTT of (a) 110ms (b) 115ms (c) 240ms (d) 250ms (e) 480ms (f) 490ms. 151
5.14 Comparison of AP-RCP and PD-RCP for (a) RTT = 240ms (b) 250ms. The two schemes are nearly identical. 152
5.15 Link utilization and queue occupancy for TCP and QCN, as RTT increases. In each case, a single 10Gbps source traverses the bottleneck switch which has a total queue size of 100 packets (150KB). 155

B.1 Example: convergence of $\bar{\alpha}(n+1) = f(\bar{\alpha}(n))$. The sequence starts with $\bar{\alpha}_0 = 1$ and follows the dashed line to $\alpha^* \approx 0.2$. 165

Chapter 1

Introduction

In the past two decades, we have witnessed the explosive growth of the Internet. Today, the Internet is essential to our way of life and has revolutionized everything from the way we consume information, communicate, and conduct business to how we socialize, access entertainment, and shop. The growth of the Internet has been driven by a series of technological waves. The rise of near-instant communication by electronic mail, instant messaging, and Voice over IP [157], and the tremendous growth of the World Wide Web [167] cemented the Internet's role as *the* medium for communication and information dissemination. The next wave witnessed the arrival of online retailers and marketplaces [16, 45] and, subsequently, the advent of platforms for streaming music [81, 130, 150, 151] and video [171, 121] which ushered online entertainment. More recently, social networking sites [46, 61] have changed the way people stay in touch with friends and relatives. Another powerful trend is now sweeping through these industries due the rapid adoption of smartphones and mobile computing devices.

These new services need a combination of large-scale computing, vast amounts of high-speed storage, and a high bandwidth interconnect network. Data centers, which combine computing, storage, and networking have arisen to meet this need. For the present purposes, we distinguish two types of data centers: (i) specialized data centers, and (ii) cloud data centers. Specialized data centers are purpose-built to support large-scale applications such as web search, electronic commerce, and social networking. Companies like Amazon, Microsoft, Google, and Facebook invest 100s of millions of dollars per

year [63, 119, 57, 47, 91] to build and operate such data centers to support their mission critical applications. Cloud data centers aim to offer computing and storage as utilities [20] to corporations, governments, and universities using an "on-demand, pay-as-you-go" service model. Amazon Elastic Compute Cloud [17], Windows Azure [165], and Google App Engine [62] are example cloud computing platforms and run in cloud data centers.

A consistent theme in data center design has been to build highly available, high performance computing and storage infrastructure using low cost, commodity components [72, 63]. A corresponding trend has also emerged in data center networks. In particular, low-cost commodity switches are commonly deployed, providing up to 48 ports at 1Gbps, at a price point under $2000 — roughly the price of one data center server. Several recent research proposals envision creating economical, easy-to-manage data centers using novel network architectures built atop these commodity switches [4, 64, 68]. Thus, while applications that run in data centers require very high performance, a high degree of cost-consciousness pervades the design of the data centers themselves.

Is this vision realistic?

The answer depends in large part on how well the commodity switches handle the traffic of real data center applications — this is the central motivation of this dissertation. Specifically, we are concerned with the design of efficient packet transport mechanisms that meet the demands of data center applications with commodity switches. For easy deployability and cost considerations, we shall strive to get substantial performance gains with the least amount of change to existing packet transport mechanisms, algorithms, and protocols which were developed in the context of the Internet.

In order to understand the issues we must focus on, it is worthwhile to understand the differences in traffic and its requirements in data center networks and the Internet. Since 2008, most Internet traffic has originated or terminated in a data center, and the amount of traffic *inside* data centers is outstripping the traffic on the Internet. Indeed, according to one recent forecast [3], the total global traffic over the Internet (IP Wide Area Networks) is projected to reach 1.3 zettabytes per year in 2016. On the other hand the volume of global data center traffic was already 1.8 zettabytes per year in 2011 and is expected to quadruple to 6.6 zettabytes per year by 2016 [2]. Peering further into data center traffic, Figure 1.1 shows that the majority of data center traffic never leaves the

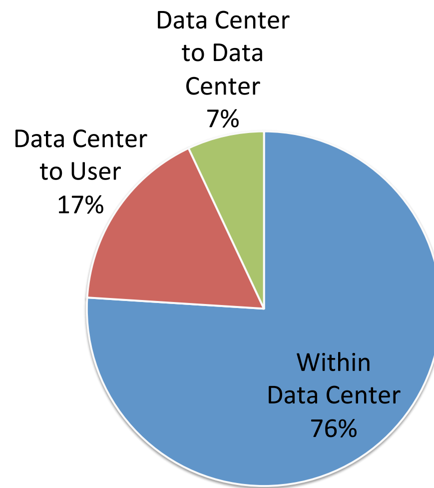


Figure 1.1: Breakdown of global data center traffic by destination. The data is from the Cisco Global Cloud Index [2] and is based on projections for 2011–2016.

data center. This traffic is generated by the "back-end" of applications. Web applications are usually deployed with a front-end tier of web servers and a back-end tier of storage servers, and have particular workflow structures (e.g., the *Partition/Aggregate* workflow described in detail in Chapter 2) that require the applications to comb through terabytes of data across many servers for each user query. This results in significant read/write, backup, and replication traffic within the data center, often on a per request basis.

The traffic within the data center has very stringent latency and throughput requirements. Low latency communication is particularly critical to overall user experience. The near real-time deadline for end results[1] (e.g., responding to a web search query) places strict deadlines for individual tasks in the back-end. Tasks not completed before their deadlines are cancelled, affecting the final result. Thus, *application requirements for low latency directly impact the quality of the result returned and hence revenue.* At the same time, latency-sensitive traffic must share the network with bulk workloads (e.g., backup traffic) that require high throughput.

In this environment, as we show via an extensive measurement study from real production clusters in Chapter 2, TCP (for Transmission Control Protocol) [84] — the state-of-the-art transport mechanism in the Internet — falls short in various ways. Essentially, TCP has

[1] For example, Amazon has reported that a 100ms increase in latency resulted in 1% less sales [107].

been designed to operate in *any* environment, across networks with very diverse speed and latency properties, but its "one-size-fits-all" approach is not adequate for the demanding performance requirements of the data center. The problems are rooted in TCP's demands on the limited buffer space available in commodity data center switches. For example, with TCP, a few bandwidth hungry "background" flows buildup large queue at the switches, and thus impact the performance of latency-sensitive "foreground" traffic.

We present new transport mechanisms for simultaneously achieving low latency and high bandwidth in a data center network. Our approach is to devise congestion control algorithms that can operate with very small buffer occupancies (and hence achieve low latency), while also maintaing high throughput. The key idea is to design control loops that react proportionately to the extent congestion in the network and avoid unnecessary rate fluctuations; i.e., congestion control loops with very good rate stability. This allows us to very proactively signal congestion in the network to prevent queue buildup in switches.

Our work builds on the vast literature on the design and analysis of congestion control algorithms for the Internet. Since Van Jacobson's seminal work [84] on the TCP algorithm in the late 1980s, the networking research community has made significant progress in understanding congestion control in the Internet, both from the practical [31, 96, 50, 154, 170, 70, 169, 44, 163, 105, 19] and theoretical [98, 112, 117, 60, 74, 116, 152] viewpoints. We advance this work by developing solutions tailored to the unique constraints of the data center — particularly, the very short round-trip latency of the data center (100s of microseconds compared to 100s of milliseconds in the Internet), the shallow buffers in commodity data center switches (10s–100s of KBytes per port compared to 100s of MBytes in Internet routers), and the specific traffic patterns that are prevalent in data centers (e.g., bursty synchronized short flows [158, 35] and small numbers of long high-bandwidth flows).

As previously mentioned, a major goal of our work is to devise solutions that are easily deployable. Hence, we strive to get substantial performance gains with the least amount of change, and reuse mechanisms and standards already available in existing hardware and networking software stacks. For instance, much of our work leverages the Explicit Congestion Notification (ECN) [138] feature widely supported in data center switches.

1.1 Overview & Organization

We now briefly overview the dissertation and its organization.

1.1.1 Data Center TCP (DCTCP)

In Chapter 2, we propose DCTCP, a TCP-like protocol for data center networks. We motivate DCTCP's design with measurements from $\sim$6000 servers in three production clusters (supporting web search and related services at Microsoft), extracting application patterns and requirements (in particular, low latency requirements), from data centers whose networks are comprised of commodity switches. We identify impairments with today's state-of-the-art TCP [84] protocol that hurt application performance, and link these impairments to the properties of the traffic and the switches. In particular, we find that TCP suffers from the incast [158, 35] problem and cannot handle synchronized bursty traffic patterns that arise in web applications.[2] Also, long TCP flows consume significant buffer space in the switches, increasing latency and causing packet drops to latency-sensitive traffic.

To address these problems, we design DCTCP to keep the switch buffer occupancies persistently low, while maintaing high throughput for the long flows. DCTCP combines Explicit Congestion Notification (ECN) [138] — a widely available feature that allows a switch to set a bit in packets traversing it to signal congestion — with a novel control scheme at the sources. It extracts multibit feedback regarding congestion in the network from the single bit stream of ECN marks. Sources estimate the fraction of marked packets, and use that estimate as a signal for the extent of congestion. We evaluate DCTCP at 1 and 10Gbps speeds using commodity, shallow buffered switches, and find that DCTCP delivers the same or better throughput than TCP, while using $\sim$90% less buffer space. Thus DCTCP significantly lowers the latency for short flows and improves the burst tolerance of the network (because it frees switch buffers). Figure 1.2 illustrates the effectiveness of DCTCP in achieving full throughput while taking up a very small footprint in the switch packet buffer, as compared to TCP.

[2] The TCP Incast problem has been discussed previously in the context of storage systems [158, 35]. We find that it occurs in large-scale web applications as well because of their Partition/Aggregate workflow structure (Section 2.1).

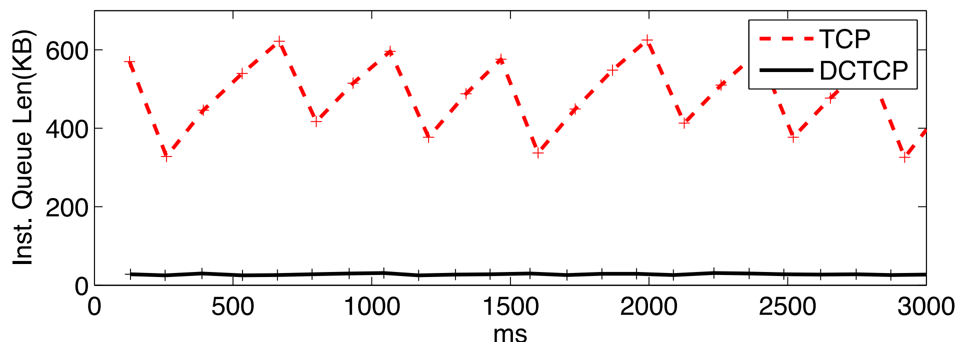


Figure 1.2: Queue length measured on a Broadcom Triumph switch. Two long flows are launched from distinct 1Gbps ports to a common 1Gbps port. The switch has dynamic memory management enabled, allowing the congested port to dynamically grab up to ∼600KB of buffer. The experiment shows that DCTCP maintains a very small buffer occupancy compared to TCP, without compromising throughput. Note that both schemes achieve full throughput since the buffer never underflows.

In Chapter 3, we conduct a mathematical analysis of DCTCP. We develop a fluid model for the DCTCP control loop and use it to analyze the throughput and delay performance of the algorithm as a function of the design parameters and of network conditions like link speeds, round-trip times, and the number of active flows. Unlike fluid model representations of standard congestion control loops [76, 116, 152], the DCTCP fluid model exhibits limit cycle behavior. Therefore, it is not amenable to analysis by linearization around a fixed point and we undertake a direct analysis of the limit cycles, proving their stability. We also analyze the rate of convergence of DCTCP sources to their fair share and their "RTT-fairness" — the rate obtained by DCTCP sources as a function of their round-trip times. Our results precisely characterize DCTCP's steady state and transient behavior and provide guidelines for setting its parameters.

1.1.2 Ultra-Low Latency Packet Transport (HULL)

In Chapter 4, we present HULL (for High-bandwidth Ultra-Low Latency), an architecture that takes low latency a step further and drives buffers completely empty, so that short flows experience the lowest possible latency (only propagation and switching delays) through the network. HULL leaves "bandwidth headroom" using Phantom Queues in the network

that deliver congestion signals before network links are fully utilized and queues form at switches. By capping utilization at less than link capacity, we leave room for latency sensitive traffic to avoid buffering and the associated large delays. At the same time, HULL uses DCTCP to mitigate the bandwidth penalties which arise from operating in a bufferless fashion, and employs hardware packet pacing to counter burstiness caused by widely used offload features in modern Network Interface Cards (NICs) such as interrupt coalescing and Large Send Offloading. Our implementation and simulation results show that by sacrificing a small amount (e.g., 10%) of bandwidth, HULL can dramatically reduce average and tail latencies in the data center, achieving about a 10X lower latency than DCTCP (which itself has a 4–6x lower latency than TCP).

1.1.3 Stability Analysis of QCN

In Chapter 5, we present a control-theoretic analysis of the stability of the Quantized Congestion Notification (QCN) algorithm that was standardized in March 2010 by the Data Center Bridging Task group in the IEEE802.1 standards body as the IEEE802.1Qau Congestion Notification [136] standard. QCN is a Layer 2 congestion control mechanism that allows a congested Ethernet switch to directly control the rates of Ethernet NICs. The standard essentially specifies a control loop at Layer 2, implemented entirely in the switch and NIC hardware, similar to the TCP (and DCTCP) control loops at Layer 3.

We analyze QCN using classical techniques from linear control theory to derive the stability margins of the control loop as a function of its design parameters and network conditions. We also dig deeper into the QCN algorithm, and illustrate that underlying QCN's good stability properties is its use of a general method we call the Averaging Principle (AP). The AP can be applied to *any* control loop (not just congestion control loops) to improve stability, especially as the feedback delay increases. We demonstrate the generality of the AP with a number of examples, and mathematically analyze it establishing a connection between the AP and the well-known Proportional-Derivative (PD) controller [54] in control systems.

1.2 Contributions

This dissertation makes the following major contributions:

- **Measurement study:** We perform the first systematic measurement study focused on the requirements and problems of packet transport in data center networks. Our measurements were collected over the course of a month in December 2009 from three production clusters with over ∼6000 servers, supporting web search and related services at Microsoft. Altogether, we gathered more than 150TB of compressed data. Our study shows that with the state-of-the-art TCP, the interaction of data center traffic patterns with commodity switches leads to impairments that hurt application performance. The impairments are rooted in TCP's wasteful use of limited buffering resources in commodity switches and illustrate the need for congestion management schemes that operate efficiently with small buffer occupancies in the data center.

- **Novel data center transport mechanisms:** We design new data center transport mechanisms that meet the demands of data center applications with commodity switches. Our mechanisms rely on widely available switch features, packet formats, and protocols and hence are immediately deployable in today's data centers. Yet, they provide significant value. DCTCP operates with ∼90% less buffer footprint in switches compared to TCP, providing low latency, high burst tolerance, and full throughput. HULL drives buffers nearly empty to provide near baseline fabric latency (only switching and propagation delays) with a configurable giveaway of bandwidth.

- **Experimental evaluation:** We implement our systems and extensively evaluate them in real hardware testbeds, demonstrating their performance gains. Our evaluations of DCTCP were conducted in a testbed running the Windows operating system, and used actual data center switches with ECN support. For HULL, we used the programmable NetFPGA [59, 122] platform to prototype our Phantom Queue and Pacer designs and implemented DCTCP in the Linux kernel. Our experimental results show that DCTCP and HULL significantly outperform TCP. In handling workloads derived from our measurement study, we find that DCTCP enables the applications to handle 10X the current background traffic without impacting the latency of the

foreground traffic. Also, a 10X increase in foreground traffic does not degrade performance. HULL further reduces both average and 99th percentile packet latency by more than a factor of 10 compared to DCTCP.

- **Control-theoretic analysis:** The analysis of the DCTCP control loop follows standard techniques for the most part, including the analysis of the limit cycles of switched dynamical systems [99, 162]. The analysis of the QCN algorithm highlights the key role played by the Averaging Principle in lending QCN stability as the round trip time of its signaling mechanism (the delay of the feedback control loop) increases. The Averaging Principle is also employed by BIC-TCP [170], an algorithm widely deployed in Linux networking stacks; our work, therefore, also provides a theoretical understanding of BIC-TCP. We also believe the Averaging Principle may be of interest in stabilizing general control loops, not just congestion control loops.

- **Deployment:** The DCTCP algorithm developed in this dissertation has been integrated into the Windows Server 2012 operating system [40]. The QCN algorithm analyzed in this dissertation has been standardized as the IEEE 802.1Qau Congestion Notification [136] standard under the aegis of the Data Center Bridging Task Group of IEEE 802.1.

1.3 Previously Published Material

The chapters of this dissertation are based on previous publications:

- Chapter 2 revises the paper [7]: M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In Proc. of ACM SIGCOMM, 2010.

- Chapter 3 revises the paper [8]: M. Alizadeh, A. Javanmard, and B. Prabhakar. Analysis of DCTCP: stability, convergence, and fairness. In Proc. of ACM SIGMETRICS, 2011.

- Chapter 4 revises the paper [10]: 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. of USENIX NSDI, 2012.

- Chapter 5 revises the two papers [6, 9]: M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar, and M. Seaman. Data center transport mechanisms: Congestion control theory and IEEE standardization. In Proc. of Allerton, 2008, and M. Alizadeh, A. Kabbani, B. Atikoglu, and B. Prabhakar. Stability analysis of QCN: the averaging principle. In Proc. of ACM SIGMETRICS, 2011.

Chapter 2

Data Center TCP (DCTCP)

In this chapter, we begin our exploration of packet transport in data centers. We focus on soft real-time applications, supporting web search, retail, advertising, and recommendation systems that have driven much data center construction. These applications generate a diverse mix of short and long flows, and require three things from the data center network: *low latency for short flows, high burst tolerance, and high utilization for long flows*.

The first two requirements stem from the *Partition/Aggregate* (described in Section 2.1.1) workflow structure that many of these applications use. The near real-time deadlines for end results (e.g., responding to a web search query) translate into latency targets for the individual tasks in the workflow. These targets vary from $\sim$10ms to $\sim$100ms, and tasks not completed before their deadline are cancelled, affecting the final result. Thus, the low latency requirements directly impact the quality of the results. Reducing network latency allows application developers to invest more cycles in the algorithms that improve relevance and end user experience.

The third requirement, high utilization for large flows, stems from the need to continuously update the internal data structures of these applications, as the freshness of the data also affects the quality of the results. Thus, high throughput for these long flows is as essential as low latency and burst tolerance.

We make two main contributions:

- First, we measure and analyze production traffic ($>$150TB of compressed data) collected over the course of a month from $\sim$6000 servers supporting web search and

related services (Section 2.1). We extract application patterns and requirements, and identify impairments with today's state-of-the-art TCP [84] protocol that hurt application performance. The impairments are linked to properties of the traffic and the commodity switches used in these data centers (see below).

- Second, we propose Data Center TCP (DCTCP), which addresses these impairments to meet the needs of applications (Section 2.2). DCTCP uses Explicit Congestion Notification (ECN) [138], a feature already available in modern commodity switches. We evaluate DCTCP at 1 and 10Gbps speeds on ECN-capable commodity switches (Section 2.3). We find that DCTCP delivers the same or better throughput than TCP, while using $\sim$90% less buffer space, and unlike TCP, DCTCP provides high burst tolerance and low latency for short flows. In handling workloads derived from operational measurements, we find that DCTCP successfully supports 10X increases in application foreground and background traffic.

Our measurements show that 99.91% of traffic in the data centers we considered uses TCP. The traffic consists of query traffic (2KB to 20KB in size), delay-sensitive short messages (100KB to 1MB), and throughput-intensive long flows (1MB to 100MB). The query traffic experiences the incast impairment, discussed previously [158, 35] in the context of storage networks. However, the data also reveals new impairments unrelated to incast. Query and delay-sensitive short messages experience long latencies due to long flows consuming some or all of the available buffer in the switches. Our key learning from these measurements is that to meet the requirements of such a diverse mix of short and long flows, *switch buffer occupancies need to be persistently low, while maintaining high throughput for the long flows.* DCTCP is designed to do exactly this.

DCTCP leverages Explicit Congestion Notification (ECN) [138] in data center switches, and uses a novel control scheme at the sources to extract multi-bit feedback regarding network congestion from the single bit stream of ECN marks. Sources use the fraction of marked packets as an estimate for the extent of congestion and react proportionately. This allows DCTCP to operate with very low buffer occupancies while still achieving high throughput (see Figure 1.2 for an illustration). DCTCP is also very simple: it requires only $\sim$30 lines of code change to TCP, and the setting of a single parameter on the switches.

While designing DCTCP, a key requirement was that it be implementable with mechanisms in existing hardware — meaning our evaluation can be conducted on physical hardware, and the solution can be deployed in today's data centers. The industry reality is that after years of debate and consensus building, a very small number of mechanisms, such as basic Random Early Discard (RED) [51] and ECN, are realized in hardware. Thus, we did not consider solutions such as XCP [96] and RCP [44], which are not implemented in any commercially-available switches (and are possibly difficult to implement).

We stress that DCTCP is designed for the data center environment. We make no claims about suitability of DCTCP for wide area networks. As previously discussed, the data center environment is significantly different from wide area networks [95, 28]. For example, round trip times (RTTs) can be less than 250μs in absence of queuing. The commodity switches typically used in data center networks have very shallow buffers (10s–100s of KBytes per port). Applications simultaneously require extremely high bandwidths and very low latencies and, often, there is little statistical multiplexing: a single flow can dominate a particular path.

At the same time, the data center environment offers certain luxuries. The network is largely homogeneous and under a single administrative control. Thus, backward compatibility, incremental deployment and fairness to legacy protocols are not major concerns. Connectivity to the external Internet is typically managed through load balancers and application proxies that effectively separate internal from external (at least one endpoint outside the data center) traffic, so issues of fairness with conventional TCP are irrelevant. In particular, we do not address the question of how to apportion data center bandwidth between internal and external traffic. The simplest class of solutions involve using Ethernet priorities (Class of Service) to keep internal and external flows separate at the switches, with ECN marking in the data center carried out strictly for internal flows.

The TCP literature is vast, and there are two large families of congestion control protocols that attempt to control queue lengths: (i) Delay-based protocols use increases in RTT measurements as a sign of growing queueing delay, and hence of congestion. These protocols rely heavily on accurate RTT measurement, which is susceptible to noise in the very low latency environment of data centers. Small noisy fluctuations of latency (e.g., microsecond variations due to process scheduling in the kernel) become indistinguishable

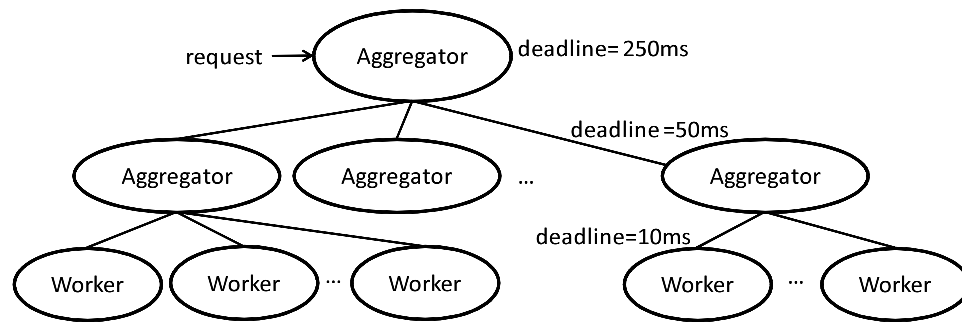


Figure 2.1: The Partition/Aggregate application structure. Requests from higher layers of the application are broken into pieces and farmed out to workers in the lower layers, whose responses are aggregated to produce the result.

from congestion and the algorithm may over-react. (ii) Active Queue Management (AQM) approaches use explicit feedback from congested switches. The algorithm we propose is in this family. Other approaches for obtaining short latencies include QoS and dividing network traffic into classes. However, QoS requires application developers to agree on how traffic is prioritized in a dynamic multi-application environment.

2.1 Communications in Data Centers

To understand the challenges facing data center transport protocols, we first describe a common application structure called Partition/Aggregate, that motivates why latency is a critical metric in data centers. We measure the synchronized and bursty traffic patterns that result from this application structure, and identify three performance impairments these patterns cause.

2.1.1 Partition/Aggregate Application Structure

The Partition/Aggregate application structure (shown in Figure 2.1) is the foundation of many large scale web applications. Requests from higher layers of the application are broken into pieces and farmed out to workers in lower layers. The responses of these workers are aggregated to produce the result. Web search, social network content composition, and

advertisement selection are all based around this application structure. For interactive, soft-real-time applications like these, latency is the key metric, with total permissible latency being determined by factors including customer impact studies [101]. After subtracting typical Internet and rendering delays, the "backend" part of the application is typically allocated between 200-300ms. This limit is called an all-up SLA (Service Level Agreement).

Many applications have a multi-layer partition/aggregate pattern workflow, with lags at one layer delaying the initiation of others. Further, answering a request may require iteratively invoking the pattern, with an aggregator making serial requests to the workers below it to prepare a response (1 to 4 iterations are typical, though as many as 20 may occur). For example, in web search, a query might be sent to many aggregators and workers, each responsible for a different part of the index. Based on the replies, an aggregator might refine the query and send it out again to improve the relevance of the result. Lagging instances of partition/aggregate can thus add up to threaten the all-up SLAs for queries. Indeed, we found that latencies run close to SLA targets, as developers exploit all of the available time budget to compute the best result possible.

To prevent the all-up SLA from being violated, worker nodes are assigned tight deadlines, usually on the order of 10-100ms. *When a node misses its deadline, the computation continues without that response, lowering the quality of the result.* Further, high percentiles for worker latencies matter. For example, since a request may rely on 100s of workers before it is satisfied, even if 1 in 1000 responses is slow, a large number of users could be affected. Therefore, latencies are typically tracked to 99.9^{th} percentiles, and deadlines are associated with high percentiles.

With such tight deadlines, network delays within the data center play a significant role in application design. Many applications find it difficult to meet these deadlines using state-of-the-art TCP, so developers often resort to complex, ad-hoc solutions. For example, our application carefully controls the amount of data each worker sends and adds jitter to requests to prevent synchronized bursty traffic patterns (Section 2.1.3). Facebook, reportedly, has gone to the extent of developing their own UDP-based congestion control [142].

2.1.2 Workload Characterization

We next measure the attributes of workloads in three production clusters related to web search and other services at Microsoft. The measurements serve to illuminate the nature of data center traffic, and they provide the basis for understanding why TCP behaves poorly and for the creation of benchmarks for evaluating DCTCP.

We instrumented a total of over 6000 servers in over 150 racks. The three clusters support soft real-time *query traffic*, integrated with urgent *short message traffic* that coordinates the activities in the cluster and continuous *background traffic* that ingests and organizes the massive data needed to sustain the quality of the query responses. We use these terms for ease of explanation and for analysis, the developers do not separate flows in simple sets of classes. The instrumentation passively collects socket level logs, selected packet-level logs, and app-level logs describing latencies – a total of about 150TB of compressed data over the course of a month.

Each rack in the clusters holds 44 servers. Each server connects to a Top of Rack switch (ToR) via 1Gbps Ethernet. The ToRs are shallow buffered, shared-memory switches; each with 4MB of buffer shared among 48 1Gbps ports and two 10Gbps ports.

Query traffic: *Query traffic* in the clusters follows the Partition/Aggregate pattern. The query traffic consists of very short, latency-critical flows, with the following pattern. A high-level aggregator (HLA) partitions queries to a large number of mid-level aggregators (MLAs) that in turn partition each query over the 43 other servers in the same rack as the MLA. Servers act as both MLAs and workers, so each server will be acting as an aggregator for some queries at the same time it is acting as a worker for other queries. Figure 2.2(a) shows the CDF of time between arrivals of queries at mid-level aggregators. The size of the query flows is extremely regular, with queries from MLAs to workers being 1.6KB and responses from workers to MLAs being 1.6–2KB.

Background traffic: Concurrent with the query traffic is a complex mix of *background traffic*, consisting of both large and small flows. Figure 2.3 presents the PDF and CDF of background flow size, illustrating how most background flows are small, but most of the bytes in background traffic are part of large flows. Key among background flows are large, 1MB to 50MB, *update* flows that copy fresh data to the workers and time-sensitive *short*

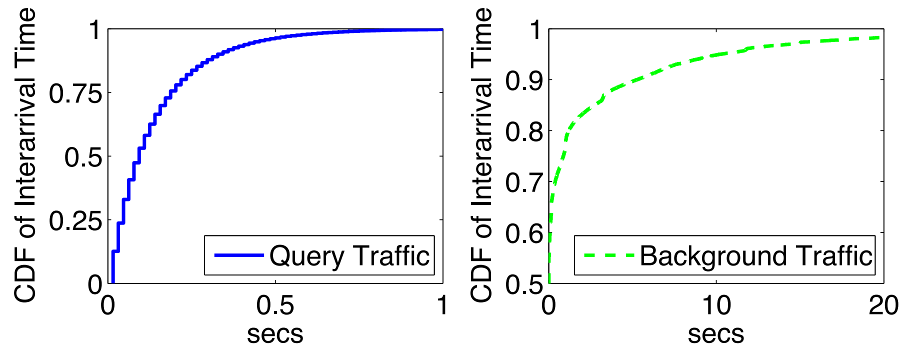


Figure 2.2: Time between arrival of new work for the Aggregator (queries) and between background flows between servers (update and short messages).

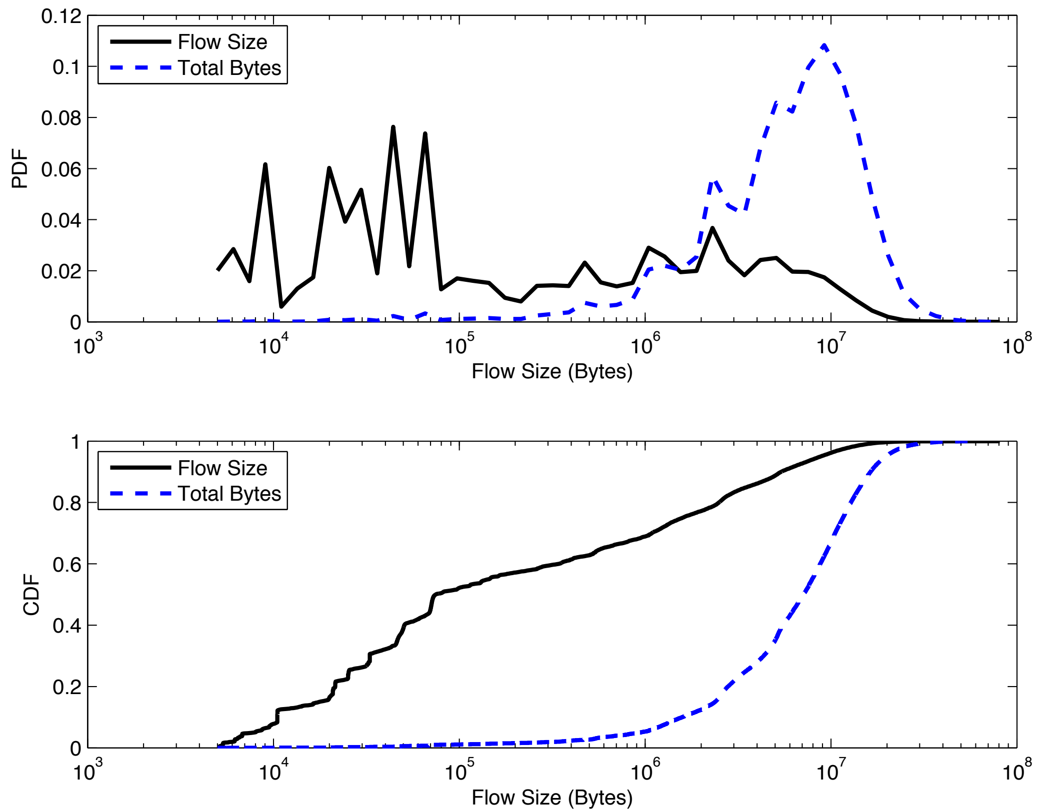


Figure 2.3: The PDF and CDF of flow size distribution for background traffic. The Total Bytes represents the probability a randomly selected byte belongs to a flow of given size.

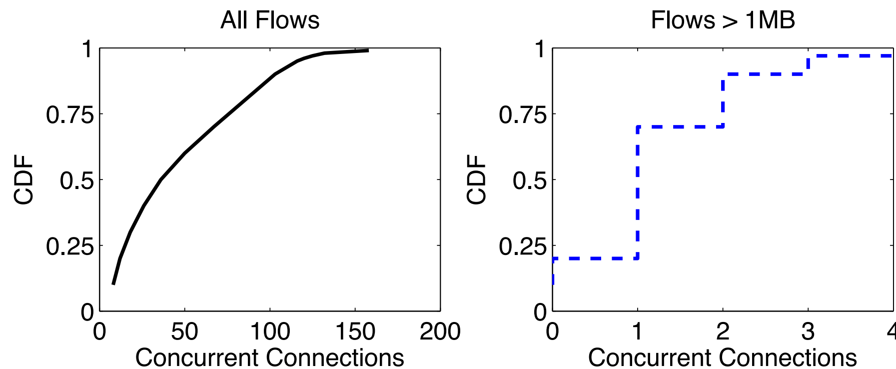


Figure 2.4: Distribution of number of concurrent connections, across all flows (on the left) and across the flows that are larger than 1MB (on the right).

message flows, 50KB to 1MB in size, that update control state on the workers. Figure 2.2(b) shows the time between arrival of new background flows. The inter-arrival time between background flows reflects the superposition and diversity of the many different services supporting the application: (i) the variance in interarrival time is very high, with a very heavy tail; (ii) embedded spikes occur, for example the 0ms inter-arrivals that explain the CDF hugging the y-axis up to the 50th percentile; and (iii) relatively large numbers of outgoing flows occur periodically, resulting from workers periodically polling a number of peers looking for updated files.

Flow concurrency: Figure 2.4 presents the CDF of the number of flows a MLA or worker node participates in concurrently (defined as the number of flows active during a 50ms window). When all flows are considered, the median number of concurrent flows is 36, which results from the breadth of the Partition/Aggregate traffic pattern in which each server talks to 43 other servers. The 99.99th percentile is over 1,600, and there is one server with a *median* of 1,200 connections. When only large flows ($>$ 1MB) are considered, the degree of statistical multiplexing is very low — the median number of concurrent large flows is 1, and the 75th percentile is 2. Yet, these flows are large enough that they last several RTTs, and can consume significant buffer space by causing queue buildup.

In summary, throughput-sensitive large flows, delay sensitive short flows and bursty query traffic, co-exist in a data center network. In the next section, we will see how TCP fails to satisfy the performance requirements of these flows.

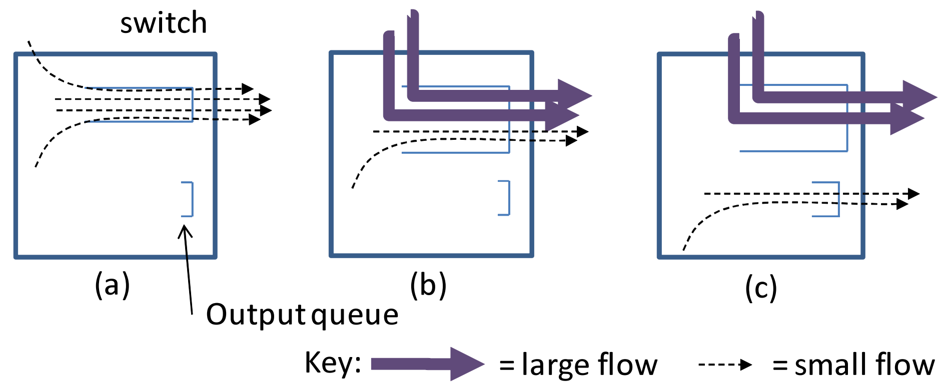


Figure 2.5: Three ways in which flows interact at a shared memory switch that result in performance problems. (a) Incast: many synchronized flows converge on the same interface leading to packet loss and retransmission timeouts; (b) Queue buildup: long and short flows traverse the same port with the long flows building up a large queue backlog, causing latency to the short flows; and (c) Buffer pressure: long flows on one port use up the shared buffer pool, reducing the available buffer to absorb bursts on the other ports.

2.1.3 Understanding Performance Impairments

To explain the performance issues seen in the production cluster, we need to study the interaction between the long and short flows in the cluster and the ways flows interact with the switches that carry the traffic.

Switches: Like most commodity switches, the switches in these clusters are *shared memory switches* that aim to exploit statistical multiplexing gain through use of logically common packet buffers available to all switch ports. Packets arriving on an interface are stored into a high speed multi-ported memory shared by all the interfaces. Memory from the shared pool is dynamically allocated to a packet by a Memory Management Unit (MMU). The MMU attempts to give each interface as much memory as it needs while preventing unfairness [110] by dynamically adjusting the maximum amount of memory any one interface can take. If a packet must be queued for an outgoing interface, but the interface has hit its maximum memory allocation or the shared pool itself is depleted, then the packet is dropped. Most cheap commodity switches are single-chip and have *shallow buffers*, with the size dictated by what fits on chip.

The shallow packet buffers cause three performance impairments shown in Figure 2.5.

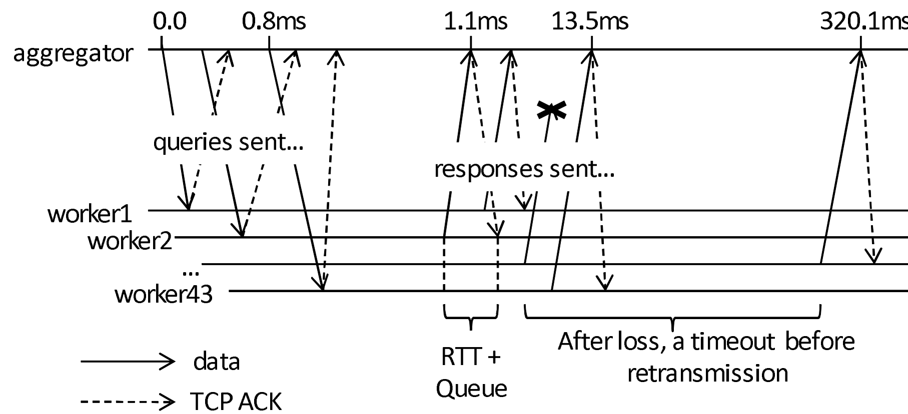


Figure 2.6: A real incast event measured in a production environment. Timeline shows queries forwarded over 0.8ms, with all but one response returning by 13.5ms. That response is lost, and is retransmitted after RTO_{min} (300ms). RTT+Queue estimates queue length on the port to the aggregator.

We now describe these impairments in detail.

Incast

As illustrated in Figure 2.5(a), if many flows converge on the same interface of a switch over a short period of time, the packets may exhaust either the switch memory or the maximum permitted buffer for that interface, resulting in packet losses. This can occur even if the flow sizes are small. This traffic pattern arises naturally from use of the Partition/Aggregate design pattern, as the request for data synchronizes the workers' responses and creates *incast* [158] at the queue of the switch port connected to the aggregator.

The previous incast research [158, 35] involves carefully constructed test lab scenarios. We find that incast-like problems do happen in production environments and they matter — degrading both performance and, more importantly, user experience. The problem is that a response that incurs incast will almost certainly miss the aggregator deadline and be left out of the final results.

We capture incast instances via packet-level monitoring. Figure 2.6 shows timeline of an observed instance. Since the size of each individual response in this application is

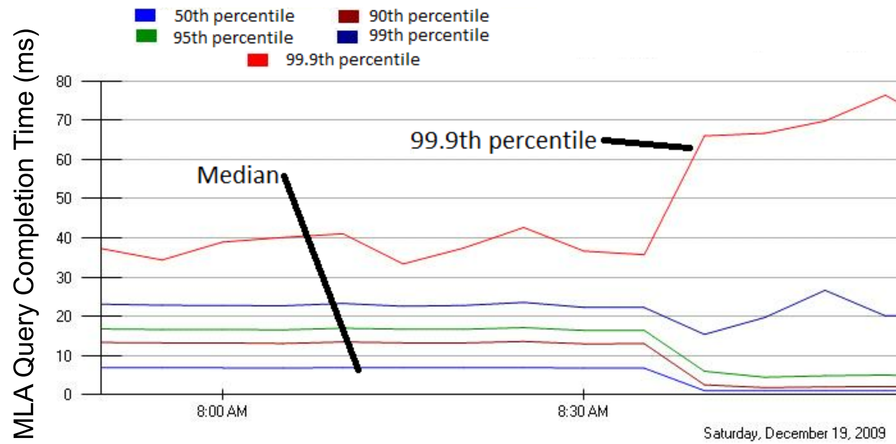


Figure 2.7: Response time percentiles for a production application having the incast traffic pattern. Forwarded requests were jittered (deliberately delayed) over a 10ms window until 8:30am, when jittering was switched off. The 95th and lower percentiles drop 10x, while the 99.9th percentile doubles.

only 2KB (2 packets),[1] loss of a packet almost invariably results in a TCP time out. In our network stack, the minimum retransmission timeout ($RTO_{\min}$) is set to 300ms. Thus, whenever a timeout occurs, that response almost always misses the aggregator's deadline.

The developers of this application have made two major changes to the application code to avoid timeouts on worker responses. First, they deliberately limited the size of the response to 2KB to improve the odds that all the responses will fit in the memory of the switch. Second, the developers added application-level jittering [52] to desynchronize the responses by deliberating delaying them by a random amount of time. The problem with jittering is that it reduces the response time at higher percentiles (by avoiding timeouts) at the cost of increasing the median response time (due to added delay). This is vividly illustrated in Figure 2.7 which shows a screen capture of a monitoring tool in one of the production clusters. The tool shows the median and higher percentiles for the query completion time at the MLAs. Forwarded requests were jittered over a 10ms window until 8:30am, when jittering was switched off. Without jittering, the 95th and lower percentiles

[1] Each query goes to all machines in the rack and each machine responds with 2KB, so the total response size is over 86KB.

improve by $\sim$10x, while the 99.9th percentile doubles.

Proposals to decrease $RTO_{\min}$ reduce the impact of timeouts [158], but, as we show next, these proposals do not address other important sources of latency.

Queue buildup

Long-lived, greedy TCP flows will cause the length of the bottleneck queue to grow until packets are dropped, resulting in the familiar sawtooth pattern (Figure 1.2). When long and short flows traverse the same queue, as shown in Figure 2.5(b), two impairments occur. First, packet loss on the short flows can cause incast problems as described above. Second, there is a *queue buildup* impairment: even when no packets are lost, the short flows experience increased latency as they are in queue behind packets from the large flows. Since every worker in the cluster handles both query traffic and background traffic (large flows needed to update the data structures on the workers), this traffic pattern occurs very frequently.

A closer look at Figure 2.6 shows that arrivals of the responses are distributed over $\sim$12ms. Since the total size of all responses is only 43 $\times$ 2KB = 86KB — roughly 1ms of transfer time at 1Gbps — it is surprising that there would be any incast losses in such transfers. However, the key issue is the occupancy of the queue caused by other flows (the background traffic) with losses occurring when the long flows and short flows coincide.

To establish that long flows impact the latency of query responses, we measured the RTT between the worker and the aggregator: this is the time between the worker sending its response and receiving a TCP ACK from the aggregator labeled as "RTT+Queue" in Figure 2.6. We measured the intra-rack RTT to approximately 100μs in absence of queuing, while inter-rack RTTs are under 250μs. This means "RTT+queue" is a good measure of the length of the packet queue headed to the aggregator during the times at which the aggregator is collecting responses. The CDF in Figure 2.8 is the distribution of the RTT for 19K measurements. It shows that 90% of the time a response packet sees $<$ 1ms of queueing, and 10% of the time it sees between 1 and 14ms of queuing (14ms is the maximum amount of dynamic buffer). This indicates that query flows are indeed experiencing queuing delays. Further, recall that answering a request can require multiple iterations, which magnifies the impact of this delay.

Note that this delay is unrelated to incast. No packets are being lost, so reducing

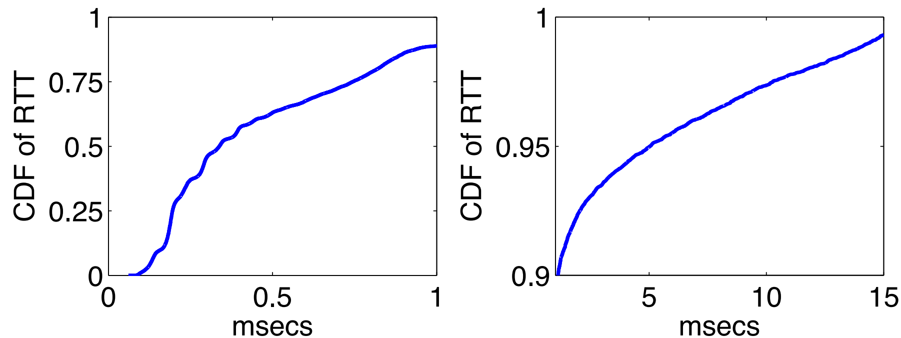


Figure 2.8: CDF of RTT to the aggregator. 10% of responses see an unacceptable queuing delay of 1 to 14ms caused by long flows sharing the queue.

RTO_{min} will not help. Further, there need not even be many synchronized short flows. *Since the latency is caused by queueing, the only solution is to reduce the size of the queues.*

Buffer pressure

Given the mix of long and short flows in our data center, it is very common for short flows on one port to be impacted by activity on any of the many other ports, as depicted in Figure 2.5(c). Indeed, the loss rate of short flows in this traffic pattern depends on the number of long flows *traversing other ports*. The explanation is that activity on the different ports is coupled by the shared memory pool.

The long, greedy TCP flows build up queues on their interfaces. Since buffer space is a shared resource, the queue build up reduces the amount of buffer space available to absorb bursts of traffic from Partition/Aggregate traffic. We term this impairment *buffer pressure*. The result is packet loss and timeouts, as in incast, but without requiring synchronized flows, or even competing long flows on the same port as the affected traffic.

2.2 The DCTCP Algorithm

The design of DCTCP is motivated by the performance impairments described in the previous section. The goal of DCTCP is to achieve high burst tolerance, low latency, and high throughput, with commodity shallow buffered switches. To this end, DCTCP is designed

to operate with small queue occupancies, without loss of throughput.

DCTCP achieves these goals primarily by reacting to congestion in proportion to the extent of congestion. DCTCP uses a simple marking scheme at switches that sets the Congestion Experienced (CE) codepoint [138] of packets as soon as the buffer occupancy exceeds a fixed small threshold. The DCTCP source reacts by reducing the window size by a factor that depends on the *fraction* of marked packets: the larger the fraction, the bigger the decrease factor.

It is important to note that the key contribution here is not the control law itself. It is the act of *deriving multi-bit feedback from the information present in the single-bit sequence of marks*. Other control laws that act upon this information can be derived as well. Since DCTCP only requires the network to provide single-bit feedback, we are able to re-use much of the ECN machinery that is already available in modern TCP stacks and switches.

The idea of reacting in proportion to the extent of congestion is also used by delay-based congestion control algorithms [31, 154]. Indeed, one can view path delay information as implicit multi-bit feedback. However, at very high data rates and with low-latency network fabrics, sensing the queue buildup in shallow-buffered switches can be extremely noisy. For example, a 10 packet backlog constitutes $120\mu s$ of queuing delay at 1Gbps, and only $12\mu s$ at 10Gbps. The accurate measurement of such small increases in queueing delay is a difficult task in today's servers given the noise induced by other factors such as process scheduling in the kernel and interrupt coalescing by the NIC (interrupt coalescing is discussed in detail in Chapter 4).

The need for reacting in proportion to the extent of congestion is especially acute in the absence of large-scale statistical multiplexing. Standard TCP cuts its window size by a factor of 2 when it receives an ECN notification. In effect, TCP reacts to the presence of congestion, not to its extent. Cutting the window size in half causes a large mismatch between the input rate to the link and the available capacity. In the high speed data center environment where only a small number of (large) high bandwidth flows are active at each port (Section 2.1.2), this leads to buffer underflows and loss of throughput.

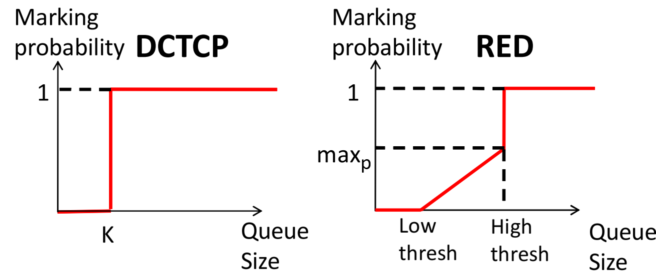


Figure 2.9: DCTCP's AQM scheme is a variant of RED: Low and High marking thresholds are equal, and marking is based on the instantaneous queue length.

2.2.1 Algorithm

The DCTCP algorithm has three main components:

1. Simple ECN marking at the switch: DCTCP employs a very simple active queue management scheme. There is only a single parameter, the marking threshold, K. An arriving packet is marked with the CE codepoint if the queue occupancy is greater than K upon its arrival. Otherwise, it is not marked. This scheme ensures that sources are quickly notified of the queue overshoot. The RED [51] marking scheme implemented by most modern switches can be re-purposed for DCTCP as shown in Figure 2.9. We simply need to set both the low and high thresholds to K, and mark based on instantaneous, instead of average queue length.

2. ECN-Echo at the receiver: The only difference between a DCTCP receiver and a TCP receiver is the way ECN marks are conveyed back to the sender. RFC 3168 [138] states that a receiver sets the ECN-Echo flag in a series of ACK packets until it receives confirmation from the sender (through the CWR flag) that the congestion notification has been received. A DCTCP receiver, however, tries to accurately convey the exact sequence of marked packets back to the sender. The simplest way to do this is to ACK every packet, setting the ECN-Echo flag if and only if the packet has a marked CE codepoint.

However, TCP stacks usually use delayed ACKs [13] where the receiver sends one cumulative ACK for every m consecutively received packets (typically, one ACK every 2 packets). This is important for a variety of reasons, including reducing the CPU load on the sender and receiver. To use delayed ACKs, the DCTCP receiver uses the simple two

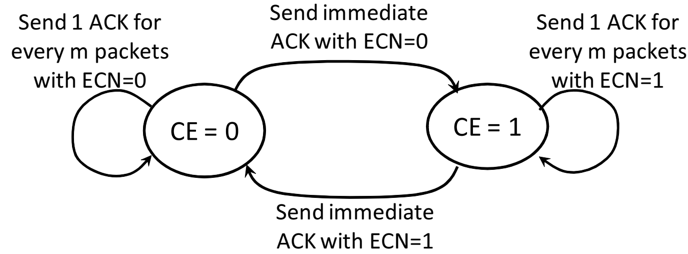


Figure 2.10: Two state ACK generation state machine used by the DCTCP receiver.

state state-machine shown in Figure 2.10 to determine whether to set the ECN-Echo bit on an ACK. The states correspond to whether the last received packet was marked with the CE codepoint or not. The receiver immediately sends an ACK anytime there is a transition from state $(CE = 0)$ to $(CE = 1)$ or vice-versa, and uses delayed ACKs when there is no transition (which is most of the time, since ECN marks at the switch typically occur in bursts). Since the sender knows how many packets each ACK covers, it can exactly reconstruct the runs of marks seen by the receiver.

3. Adaptive congestion control at the sender: The sender maintains an estimate of the fraction of packets that are marked, called α, which is updated once for every window of data (roughly one RTT) as follows:

$$\alpha \leftarrow (1 - g) \times \alpha + g \times F, \tag{2.1}$$

where F is the *fraction* of packets that were marked in the last window of data, and $0 < g < 1$ is the weight given to new samples against the past in the estimation of α. Given that the sender receives marks for *every packet* when the queue length is higher than K and does not receive *any marks* when the queue length is below K, Equation (2.1) implies that α estimates the probability that the queue size is greater than K. Essentially, α close to 0 indicates low, and α close to 1 indicates high levels of congestion.

Prior work [137, 97] on congestion control in the small buffer regime has observed that at high line rates, queue size fluctuations become so fast that you cannot control the queue size, only its *distribution*. The physical significance of α is aligned with this observation: it represents a single point of the queue size distribution at the bottleneck link.

The only difference between a DCTCP sender and a TCP sender is in how each reacts to receiving an ACK with the ECN-Echo flag set. Other features of TCP such as slow start, additive increase in congestion avoidance, or recovery from packet lost are left unchanged. While TCP always cuts its window size by a factor of 2 in response to a marked ACK, DCTCP uses α:

$$cwnd \leftarrow cwnd \times (1 - \frac{\alpha}{2}). \tag{2.2}$$

Thus, when α is near 0 (low congestion), the window is only slightly reduced. In other words, DCTCP senders start gently reducing their window as soon as the queue exceeds K. This is how DCTCP maintains low queue length, while still ensuring high throughput. When congestion is high ($\alpha = 1$), DCTCP cuts window in half, just like TCP. Also similar to TCP [138], DCTCP reduces the window size according to (2.2) at most once per window of data.

2.2.2 Benefits

DCTCP alleviates the three performance impairments discussed in Section 2.1.3 (shown in Figure 2.5) as follows.

Queue buildup: DCTCP senders begin to react as soon as the queue length on an interface exceeds the marking threshold, K. This reduces queueing delays on congested switch ports, which minimizes the impact of long flows on the completion time of small flows. Also, more buffer space is available as *headroom* to absorb transient micro-bursts, greatly mitigating costly packet losses that can lead to timeouts.

Buffer pressure: DCTCP also solves the buffer pressure problem because a congested port's queue length does not grow exceedingly large. Therefore, in shared memory switches, a few congested ports will not exhaust the buffer resources harming flows passing through other ports.

Incast: The incast scenario, where a large number of synchronized small flows hit the same queue, is the most difficult to handle. If the number of small flows is so high that even 1 packet from each flow is sufficient to overwhelm the buffer on a synchronized burst, then there isn't much DCTCP — or any congestion control scheme that does not attempt to schedule traffic — can do to avoid packet drops. However, in practice, each flow has

several packets to transmit, and their windows build up over multiple RTTs. It is often bursts in subsequent RTTs that lead to drops. Because DCTCP starts marking early (and aggressively, based on instantaneous queue length), DCTCP sources receive enough marks during the first one or two RTTs to tame the size of follow up bursts. This prevents buffer overflows and resulting timeouts.

2.2.3 Preliminary Analysis

We now conduct a preliminary analysis of the steady state behavior of the DCTCP control loop in a simplified setting. The purpose of this analysis is to provide intuition and some rough guidelines regarding how to set the DCTCP parameters. A more comprehensive mathematic analysis of DCTCP is presented in Chapter 3.

We consider N infinitely long-lived flows with identical round-trip times RTT, sharing a single bottleneck link of capacity C. We further assume that the N flows are synchronized; i.e., their "sawtooth" window size dynamics are in-phase. Of course, this assumption is only realistic when N is small. However, this is the case we care about most in data centers (Section 2.1.2). Because the N window sizes are synchronized, they follow identical sawtooths, and the queue size at time t is given by:

$$Q(t) = NW(t) - C \times RTT, \tag{2.3}$$

where $W(t)$ is the window size of a single source [19]. Therefore the queue size process is also a sawtooth. We are interested in computing the following quantities which completely specify the sawtooth (see Figure 2.11): the maximum queue size (Q_{max}), the amplitude of queue oscillations (A), and the period of oscillations (T_C). The most important of these is the amplitude of oscillations, which quantifies how 'steady' DCTCP is able to maintain the queue occupancies, due to its gentle proportionate reaction to congestion indications.

We proceed to compute these quantities. The key observation is that with synchronized senders, the queue size exceeds the marking threshold K for exactly one RTT in each period of the sawtooth, before the sources receive ECN marks and reduce their window sizes accordingly. Therefore, we can compute the fraction of marked packets, α, by simply dividing the number of packets sent during the last RTT of the period by the total number

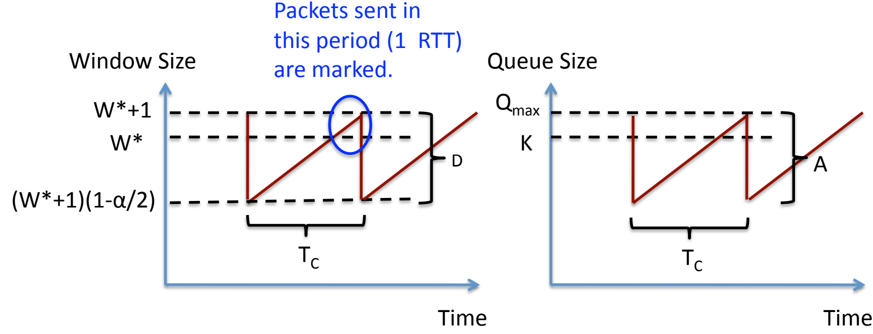


Figure 2.11: Sawtooth window size process for a single DCTCP sender, and the associated queue size process.

of packets sent during a full period of the sawtooth, T_C.

Let's consider one of the senders. Let $S(W_1, W_2)$ denote the number of packets sent by the sender, while its window size increases from W_1 to $W_2 > W_1$. Since this takes $W_2 - W_1$ round-trip times, during which the average window size is $(W_1 + W_2)/2$,

$$S(W_1, W_2) = \frac{W_2^2 - W_1^2}{2}. \tag{2.4}$$

Let $W^* = (C \times RTT + K)/N$. This is the critical window size at which the queue size reaches K and the switch starts marking packets with the CE codepoint. During the RTT it takes for the sender to react to these marks, its window size increases by one more packet, reaching $W^* + 1$. Hence,

$$\alpha = \frac{S(W^*, W^* + 1)}{S((W^* + 1)(1 - \alpha/2), W^* + 1)}. \tag{2.5}$$

Plugging (2.4) into (2.5) and rearranging, we get:

$$\alpha^2(1 - \frac{\alpha}{4}) = \frac{2W^* + 1}{(W^* + 1)^2} \approx \frac{2}{W^*}, \tag{2.6}$$

where the approximation in (2.6) is valid when $W^* >> 1$. Equation (2.6) can be used to compute α as a function of the network parameters C, RTT, N, and K. Assuming α is

small, this can further be simplified as:

$$\alpha \approx \sqrt{\frac{2}{W^*}}.$$

We can now compute A and T_C in Figure 2.11 as follows. Note that the amplitude of oscillation in window size of a single flow, D, (see Figure 2.11) is given by:

$$D = (W^* + 1) - (W^* + 1)(1 - \frac{\alpha}{2}). \tag{2.7}$$

Since there are N flows in total,

$$\begin{aligned} A = ND = N(W^* + 1)\frac{\alpha}{2} &\approx \frac{N}{2}\sqrt{2W^*} \\ &= \frac{1}{2}\sqrt{2N(C \times RTT + K)}, \end{aligned} \tag{2.8}$$

$$T_C = D = \frac{1}{2}\sqrt{\frac{2(C \times RTT + K)}{N}} \text{ (in RTTs)}. \tag{2.9}$$

Finally, using (2.3), we have:

$$Q_{max} = N(W^* + 1) - C \times RTT = K + N. \tag{2.10}$$

We have evaluated the accuracy of the above results using ns2 simulations in a variety of scenarios. Figure 2.12 shows the results for $N = 2, 10,$ and 40 long-lived DCTCP flows sharing a 10Gbps bottleneck, with a $100\mu s$ round-trip time. As seen, the analysis is indeed a fairly accurate prediction of the actual dynamics, especially when N is small (less than 10). For large N, as in the $N = 40$ case, de-synchronization of the flows leads to smaller queue variations than predicted by the analysis.

Equation (2.8) reveals an important property of DCTCP; when N is small, the amplitude of queue size oscillations with DCTCP is $O(\sqrt{C \times RTT})$, and is therefore much smaller than the $O(C \times RTT)$ oscillations of TCP. This allows for a very small marking threshold, K, without loss of throughput in the low statistical multiplexing regime seen in data centers. In fact, as we show below, even with the *worst* case assumption of synchronized flows used in this analysis, DCTCP can begin marking packets at $(1/7)^{th}$ of the

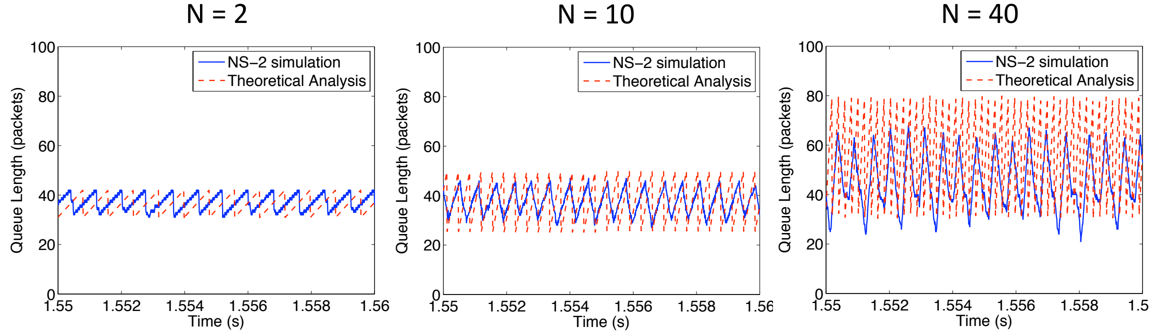


Figure 2.12: Comparison between the queue size process predicted by the analysis with ns2 simulations. The DCTCP parameters are set to $K = 40$ packets, and $g = 1/16$.

bandwidth-delay product without losing throughput.

Guidelines for choosing parameters

We now use our analysis to derive guidelines for settings the two DCTCP parameters: the marking threshold (K) at the switch, and the gain (g) for estimating the fraction of marked packets at the source. In what follows, C is in packets/second, RTT is in seconds, and K is in packets.

Marking threshold: The minimum value of the queue occupancy sawtooth (Figure 2.11) is given by:

$$\begin{aligned} Q_{min} &= Q_{max} - A \\ &= K + N - \frac{1}{2}\sqrt{2N(C \times RTT + K)}. \end{aligned} \tag{2.11}$$

To obtain a lower bound for K, we minimize Q_{min} over N, and choose K so that this minimum is larger than zero. This ensures that the queue does not underflow and throughput is not lost. Equation (2.11) is minimized for $N = (C \times RTT + K)/8$ and the minimum is given by:

$$Q^*_{min} = \frac{7K - C \times RTT}{8}.$$

Hence, the queue occupancy remains positive so long as:

$$K > \frac{C \times RTT}{7}. \tag{2.12}$$

Estimation gain: The estimation gain g must be chosen small enough to ensure that the exponential moving average in equation (2.1) spans at least one congestion event. Since a congestion event occurs every T_C round-trip times, we choose g such that:

$$(1-g)^{T_C} > \frac{1}{2}.$$

Taking the $\log(\cdot)$ of both sides and rearranging, this is equivalent to:

$$g < 1 - \exp\left(-\frac{\log 2}{T_C}\right) < \frac{\log 2}{T_C}.$$

Plugging in (2.9) with the worst case value $N = 1$ (that results in the largest value of T_C), leads to the following guideline:

$$g < \frac{2 \log 2}{\sqrt{2(C \times RTT + K)}} \approx \frac{1}{\sqrt{C \times RTT + K}}. \tag{2.13}$$

Remark 2.1. The guidelines in equations (2.12) and (2.13) are based on a simplistic model that assumes the sources have perfectly synchronized sawtooth window size processes and know the exact fraction of marked packets. Further, this model does not capture some important aspects of the DCTCP algorithm such as its convergence behavior during transience and how different round-trip times affect flow throughput. We revisit these issues in Chapter 3 where we conduct a comprehensive mathematical analysis of the DCTCP algorithm and revise the parameter guidelines accordingly.

2.2.4 Discussion

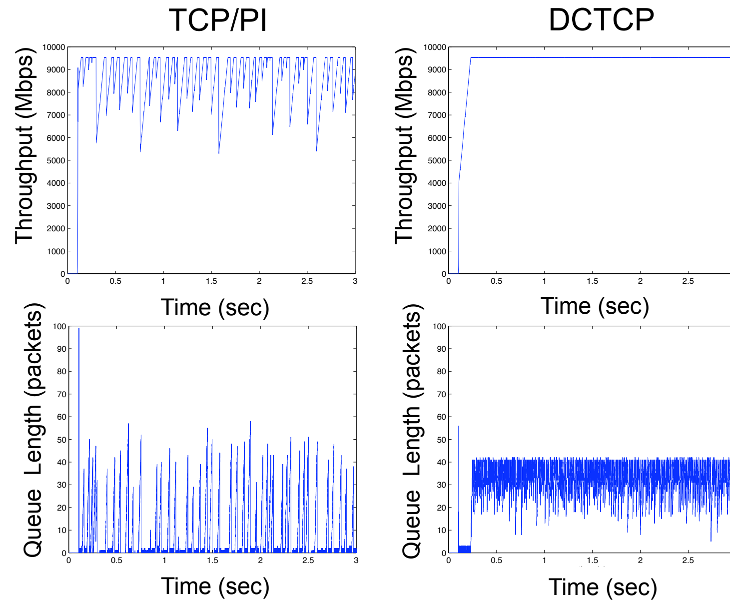
AQM is not enough: Before designing DCTCP, we evaluated Active Queue Management (AQM) schemes like RED[2] [51] and PI [76] that do not modify TCP's congestion control mechanism. We found they do not work well when there is low statistical multiplexing and traffic is bursty — both of which are true in the data center. Essentially, because of TCP's conservative reaction to congestion indications, any AQM scheme operating with TCP in a data-center-like environment requires making a tradeoff between throughput and delay: either accept large queue occupancies (and hence delay), or accept loss of throughput.

We will examine performance of RED (with ECN) in some detail in Section 2.3, since our testbed switches are RED/ECN-capable. We have evaluated other AQM schemes such as PI extensively using ns2 simulations. PI tries to regulate queue size near a desired reference value q_{ref}, which makes it easy to compare to DCTCP. Figure 2.13(a) shows the total throughput achieved by two flows on a 10Gbps bottleneck with a 500μs round-trip time. The buffer size and DCTCP marking threshold are respectively 100 and 40 packets. For PI, we set q_{ref} to 40 packets and choose the rest of the parameters based on Proposition 2 in Hollet *et al.* [76]. We see that TCP/PI suffers from queue underflows and a loss of utilization. We repeat the experiment for $N = 20$ flows in Figure 2.13(b), and find that while utilization improved for TCP/PI, queue oscillations get worse, which can hurt the latency of time-critical flows. DCTCP, however, is able to control the queue occupancy well and sustain high throughput in both scenarios.

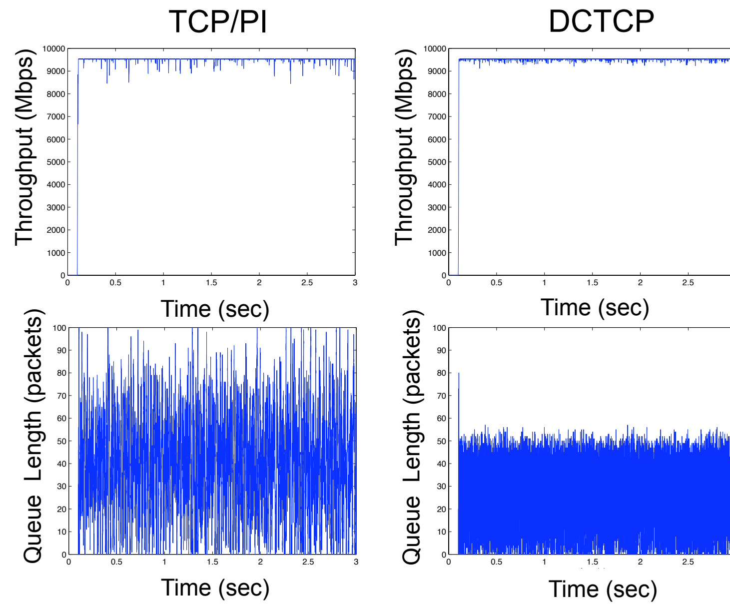
Convergence and synchronization: In both analysis and experimentation, we have found that DCTCP achieves both high throughput and low delay, all in an environment with low statistical multiplexing. In achieving this, DCTCP trades off convergence time; the time required for a new flow to grab its share of the bandwidth from an existing flow with a large window size. This is expected since a DCTCP source must make incremental adjustments to its window size based on the accumulated multi-bit feedback in α. The same tradeoff is also made by a number of TCP variants [108].

We posit that this is not a major concern in data centers. First, data center round-trip times are only a few 100μs, 2 orders of magnitudes less than RTTs in the Internet.

[2] We *always* use RED with ECN: i.e. random early *marking*, not random early drop. We call it RED simply to follow convention.



(a) $N = 2$ flows



(b) $N = 20$ flows

Figure 2.13: DCTCP and TCP/PI with 2 and 20 long-lived flows. The flows share a 10Gbps link and have a RTT of 500μs. The switch buffer size is 100 packets. The DCTCP marking threshold (K) and the PI reference queue length (q_{ref}) are both set to 40 packets.

Since convergence time for a window based protocol like DCTCP is proportional to the RTT, the actual difference in time caused by DCTCP's slower convergence compared to TCP is not substantial. Simulations show that for RTTs ranging between 100–300μs, the convergence times for DCTCP is on the order of 20–30ms at 1Gbps, and 80–150ms at 10Gbps, and is a factor of $\sim$2 slower than TCP (see Section 3.3 for an extensive analysis of DCTCP's convergence behavior). Second, in a data center dominated by microbursts, which by definition are too small to converge, and big flows, which can tolerate a small convergence delay over their long lifetimes, convergence time is the right metric to yield.

Another concern with DCTCP is that the "on-off" style marking can cause synchronization between flows. However, DCTCP's reaction to congestion is not severe, so it is less critical to avoid synchronization [51].

Practical considerations: While the recommendations of Section 2.2.3 work well in simulations, some care is required before applying these recommendations in real networks. The analysis assumes idealized DCTCP sources and does not capture any of the burstiness inherent to actual implementations of window-based congestion control protocols in the network stack. For example, we found that at 10G line rates, hosts tend to send bursts of as many as 30–40 packets when the window size permitted them to do so. While a myriad of system details (quirks in TCP stack implementations, MTU settings, and network adapter configurations) can cause burstiness, optimizations such as Large Send Offloading (LSO), and interrupt coalescing increase burstiness noticeably.[3] One must make allowances for such bursts when selecting the value of K. For example, while based on (2.12), a marking threshold as low as 20 packets can be used for 10Gbps, we found that a more conservative marking threshold larger than 60 packets is required to avoid loss of throughput. This excess is in line with the burst sizes of 30–40 packets observed at 10Gbps.

Based on our experience with the intrinsic burstiness seen at 1Gbps and 10Gbps, and the total amount of available buffering in our switches, we use the marking thresholds of $K = 20$ packets (30KB, with 1500B packets) for 1Gbps and $K = 65$ ($\sim$100KB) packets for 10Gbps ports in our experiments, unless otherwise noted. The parameter g is set to 1/16 in all experiments.

[3] Of course, such implementation issues are not specific to DCTCP and affect any protocol implemented in the stack. We explore the sources of burstiness in modern TCP stacks in depth in Section 4.3.1.

	Ports	Buffer	ECN
Triumph	48 1Gbps, 4 10Gbps	4MB	Y
Scorpion	24 10Gbps	4MB	Y
CAT4948	48 1Gbps, 2 10Gbps	16MB	N

Table 2.1: Switches in our testbed.

2.3 Results

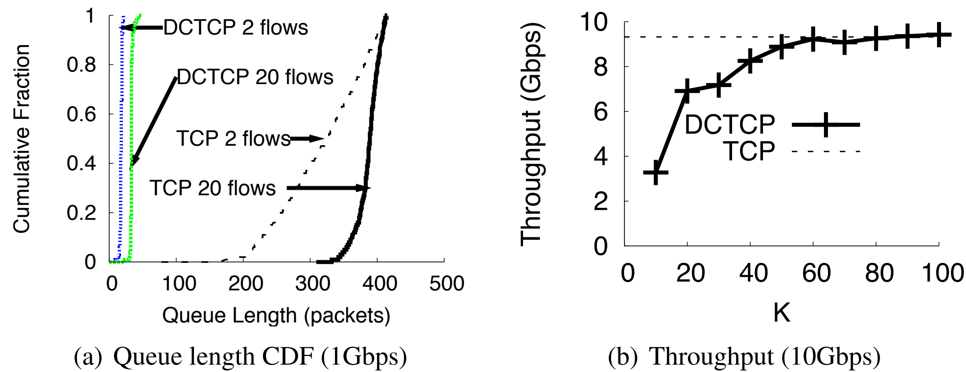
This section is divided in three parts. First, using carefully designed traffic patterns, we examine the basic properties of the DCTCP algorithm, such as convergence, fairness, and behavior in a multi-hop environment. Second, we show a series of microbenchmarks that explain how DCTCP ameliorates the specific performance impairments described in Section 2.1.3. Finally, we evaluate DCTCP using a benchmark generated from our traffic measurements. *No simulations are used in this section.* All comparisons are between a full implementation of DCTCP and a state-of-the-art TCP New Reno (w/ SACK) implementation. Unless otherwise noted, we use the parameter settings discussed at the end of Section 2.2.

Our testbed consists of 94 machines in three racks. 80 of these machines have 1Gbps NICs, and the remaining 14 have 10Gbps NICs. The CPU and memory of these machines were never a bottleneck in any of our experiments. We connect the machines together in a variety of configurations using the set of switches shown in Table 2.1. CAT4948 is a Cisco product, the rest are from Broadcom. The Triumph and Scorpion are shallow buffered, while the CAT4948 is a deep-buffered switch. Except where noted, we use the default dynamic buffer allocation policy for the switches.

2.3.1 DCTCP Performance

The experiments in this subsection are microbenchmarks, with traffic patterns specifically designed to evaluate particular aspects of DCTCP's performance.

Throughput and queue length: We begin by evaluating whether DCTCP achieves the same throughput as TCP on long-lived flows when recommended values of K are used. To determine this, we use machines connected to the Triumph switch with 1Gbps links. One



(a) Queue length CDF (1Gbps)

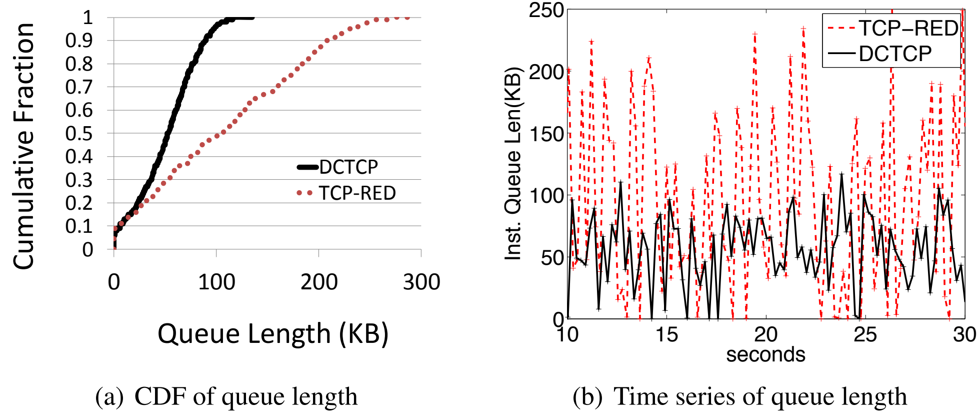
(b) Throughput (10Gbps)

Figure 2.14: Throughput and queue length for TCP and DCTCP with long-lived traffic. (a) CDF of queue length for 2 and 20 flows at 1Gbps. (b) Throughput with different marking thresholds, K, for 2 flows at 10Gbps.

host is a receiver; the others are senders. The senders establish long-lived connections to the receiver and send data as fast as they can. During the transfer, we sample the instantaneous queue length at the receiver's switch port every 125ms. We repeat the experiment for both DCTCP and TCP. For DCTCP, we set $K = 20$, as recommended before. For TCP, the switch operates in standard, drop-tail mode.

We find that both TCP and DCTCP achieve the maximum application throughput of 0.95Gbps and the receiver's link utilization is nearly 100%. The key difference is queue length at the receiver interface. The CDF in Figure 2.14(a) shows that for DCTCP the queue length is stable around 20 packets (the marking threshold) and increases gradually with more flows (i.e., is equal to $K + N$, as predicted in Section 2.2.3), while the TCP queue length is 10X larger and varies widely. In fact, Figure 1.2 is based on the data from this experiment. It shows the time series of queue length (with 2 flows) for a representative period. The sawtooth behavior of TCP is clearly visible. The upper limit for TCP's queue length is dictated by the switch's dynamic buffer allocation policy, which will allocate up to $\sim$600KB of buffer to a single busy interface if no other port is receiving traffic. In contrast, DCTCP maintains a stable, low queue length.

We also tried various other values of K, and found that the performance is fairly insensitive to value of K at 1Gbps, even for values as low as $K = 5$. We then repeated the experiment with 10Gbps link. Figure 2.14(b) shows the throughput results. Once the value



(a) CDF of queue length (b) Time series of queue length

Figure 2.15: DCTCP versus RED at 10Gbps. RED causes much larger queue length oscillations and requires more than twice the buffer of DCTCP to achieve the same throughput.

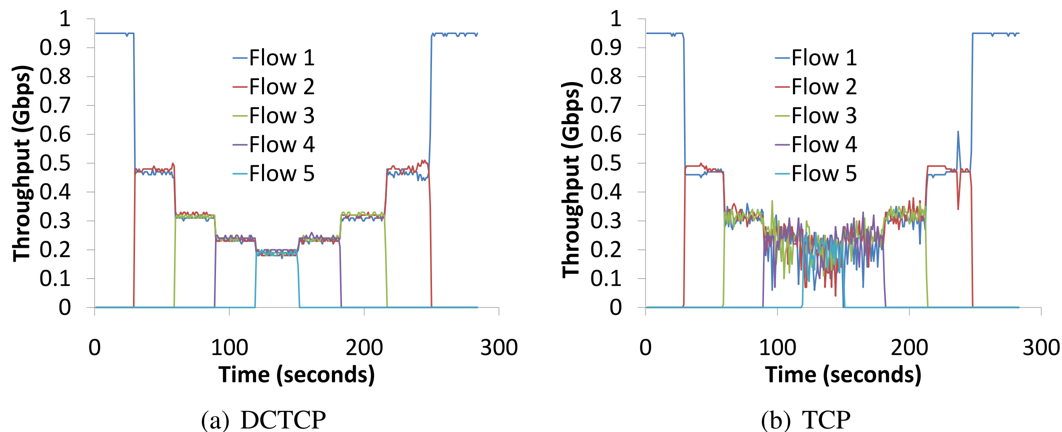
of K exceeds the recommended value of 65, DCTCP achieves full throughput (same as TCP), and is no longer sensitive to the value of K.

RED: In Section 2.2.4, we argued that AQM schemes like RED would show queue length oscillations, as these oscillations stem from the way TCP adjusts its window in response to congestion indications. We also argued that DCTCP does not suffer from this problem.

To verify this, we repeated the 10Gbps experiment. We again ran DCTCP with $K = 65$. For TCP, the switch marked packets according to RED.[4] It was difficult to set RED parameters correctly: following the guidelines in [49] (max_p=0.1, weight=9, min_th=50, max_th=150) caused TCP throughput to drop to 7.8Gbps. To get a fair comparison with DCTCP, we increased the value of the min_th and max_th RED parameters until TCP achieved 9.2Gbps at min_th = 150.

Figure 2.15 shows the distribution of queue length observed at the receiver's switch port. We see that RED causes wide oscillations in queue length, often requiring twice as much buffer to achieve the same throughput as DCTCP. This transient queue buildup means there is less room available to absorb microbursts, and we will see the impact of this on our real benchmark in Section 2.3.3. However, given the difficulty of setting RED parameters, we use TCP with drop tail as the baseline for all other experiments. The baseline also

[4]Our switches do not support any other AQM scheme. RED is implemented by setting the ECN bit, not dropping.



(a) DCTCP (b) TCP

Figure 2.16: Convergence experiment. Five flows sequentially start and the stop, spaced by 30 seconds.

reflects what our production clusters actually implement.

The main takeaway from these experiments is that *with K set according to the guidelines in Section 2.2.3, DCTCP achieves full throughput, even at very small queue length. TCP with drop tail or RED cause queue lengths to oscillate widely.*

Fairness and convergence: To show that DCTCP flows quickly converge to their fair share, we set up 6 hosts connected via 1Gbps links to the Triumph switch. K was set to 20. One of the hosts acts as a receiver, while the others act as senders. We start a single long-lived flow, and then we sequentially start and then stop the other senders, spaced by 30 seconds. The timeseries depicting the overall flow throughput is shown in Figure 2.16(a). As DCTCP flows come and go, they quickly converge to their fair share. For comparison, the corresponding timeseries for TCP is shown in Figure 2.16(b). TCP throughput is fair on average, but has much higher variation. We have repeated this test with up to 90 flows and we find that DCTCP converges quickly (nearly as quickly as TCP), and all flows achieve their fair share (the Jain's fairness index [85] is 0.99).

Multi-hop networks: To evaluate DCTCP's performance in a multi-hop, multi-bottleneck network, we created the topology in Figure 2.17. The senders in the S1 and S3 groups, totaling 20 machines, all send to receiver R1. The 20 senders in S2 each send to an assigned receiver in group R2. There are two bottlenecks in this topology: both the 10Gbps link

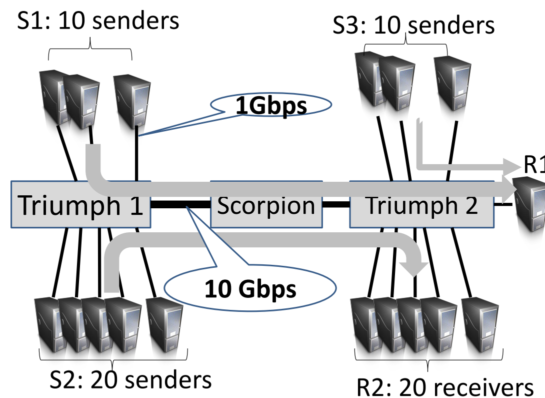


Figure 2.17: Multi-hop topology. The 10Gbps link between Triumph 1 and Scorpion and the 1Gbps link between Triumph 2 and R1 are both bottlenecks.

between Triumph 1 and the Scorpion and the 1Gbps link connecting Triumph 2 to R1 are oversubscribed. The flows from group S1 encounter both these bottlenecks. We find that with DCTCP, each sender in S1 gets 46Mbps and in S3 gets 54Mbps throughput, while the senders in S2 get approximately 475Mbps — these are within 10% of their fair-share throughputs. TCP does slightly worse: the queue length fluctuations at the two bottlenecks cause timeouts for some of the TCP connections. This experiment shows that DCTCP can cope with the multiple bottlenecks and differing RTTs that will be found in data centers.

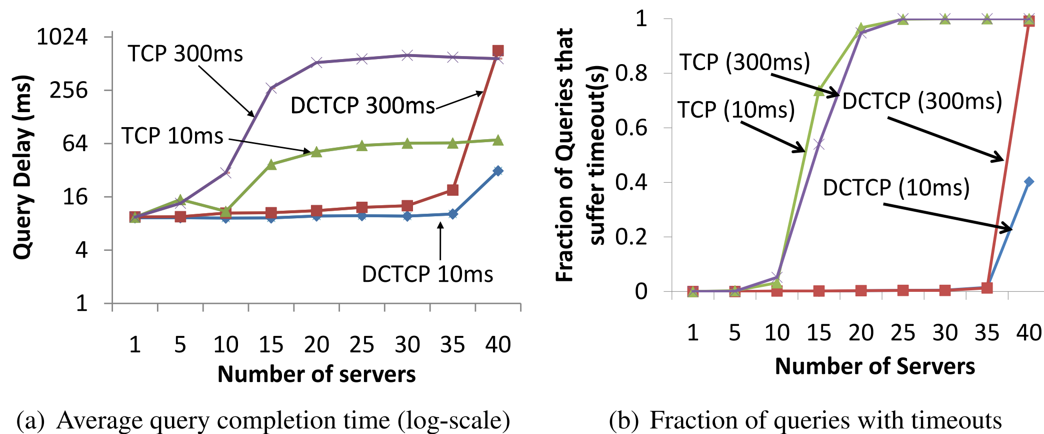
2.3.2 Impairment Microbenchmarks

We now present a series of microbenchmarks that show how DCTCP addresses the impairments described in Section 2.1.3.

Incast

In this section, we examine the *incast* impairment. Vasudevan *et al.* [158] proposed reducing the value of RTO_{min} to address the incast problem. We compare this approach with DCTCP's approach of *avoiding* timeouts, as explained in Section 2.2.2.

Basic incast: We start with an experiment that repeats the scenario considered by Vasudevan *et al.* [158]. Forty-one machines are connected to the Triumph switch with 1Gbps links,



(a) Average query completion time (log-scale) (b) Fraction of queries with timeouts

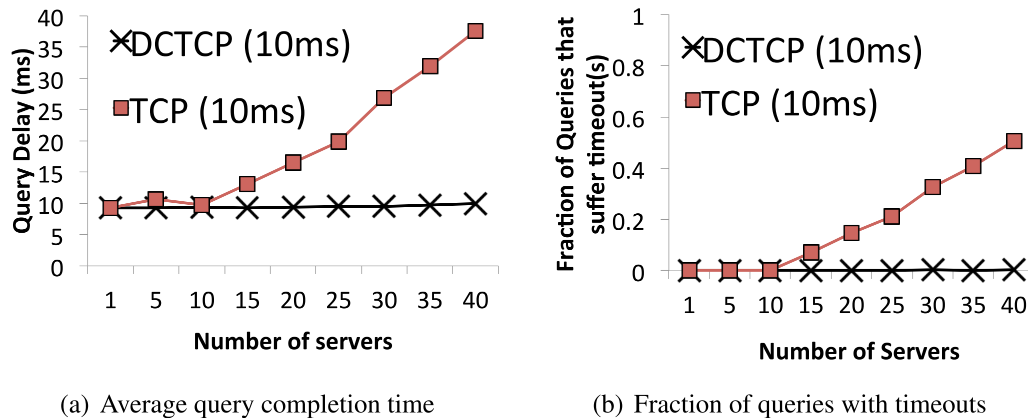
Figure 2.18: Incast performance with a static 100 packet buffer at each switch port. DCTCP performs significantly better than TCP and only begins suffering from timeouts when the number of senders exceeds 35. The values shown are the mean for 1000 queries, with each query totaling 1MB of data. The 90% confidence intervals are too small to be visible.

and, to duplicate the prior work, for this one experiment, the dynamic memory allocation policy in the switch was replaced with a static allocation of 100 packets to each port.

One machine acts as a client, others act as servers. The client requests ("queries") 1MB/N bytes from N different servers, and each server responds immediately with the requested amount of data. The client waits until it receives all the responses, and then issues another, similar query. This pattern is repeated 1000 times. The metric of interest is the completion time of the queries. The minimum query completion time is around 8ms: the incoming link at the receiver is the bottleneck, and it takes 8ms to deliver 1MB of data over a 1Gbps link. We carry out the experiment for both TCP and DCTCP, and with two timeout values: the default 300ms, and 10ms. The latter value is the tick granularity of our operating system, so it is the smallest timeout value we can use without major changes to timer handling.

Figure 2.18(a) shows the mean query completion time. DCTCP performs much better than both TCP variants — eventually converging to equivalent performance as incast degree increases to 40 senders. Figure 2.18(b) makes the reason evident by showing the fraction of queries that suffered at least one timeout.[5] TCP begins to suffer timeouts when the number

[5]Each query elicits a response from several servers, any of which can suffer a timeout. However, if multiple responses suffer timeouts, the delay does not increase proportionately, since the responses are delivered



(a) Average query completion time (b) Fraction of queries with timeouts

Figure 2.19: Incast performance with with dynamic buffer allocation at the switch. DCTCP does not suffer problems, even with high degrees of incast.

of servers exceeds 10, as no sender receives a signal to lower its sending rate until enough packets are in flight to cause a full window loss. DCTCP senders receive ECN marks, slow their rate, and only suffer timeouts once the number of senders is large enough so that each sending (around) 2 packets exceeds the static buffer size ($35 \times 2 \times 1.5\text{KB} > 100\text{KB}$).

Importance of dynamic buffering: Recall that the above experiment used a static buffer of 100 packets at each port. Would using the switch's default dynamic buffer allocation algorithm (Section 2.1.3) solve the incast problem? To answer this question, we repeated the above experiment with dynamic buffering enabled. Given the poor performance of TCP with 300ms timeout (the top line in Figure 2.18(a)), we use $RTO_{min} = 10$ms here and for the rest of the chapter.

Figure 2.19 shows that DCTCP no longer suffers incast timeouts even when the number of senders grows to 40. On the other hand, TCP continues to suffer from incast timeouts, although the dynamic buffering algorithm mitigates the impact by allocating as much as $\sim$600KB of buffer to the receiver's port (it does not allocate all 4MB for fairness). The allocated buffer is sufficient for DCTCP to avoid timeouts even with a large number of sending servers.

All-to-all incast: In the previous experiments, there was only a single receiver. This is

in parallel.

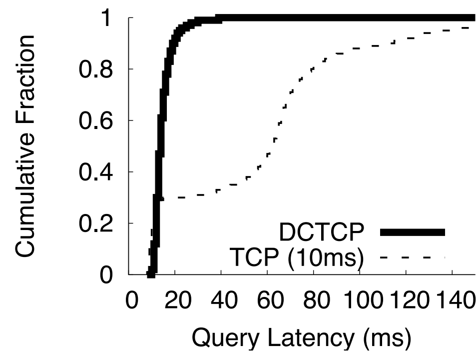
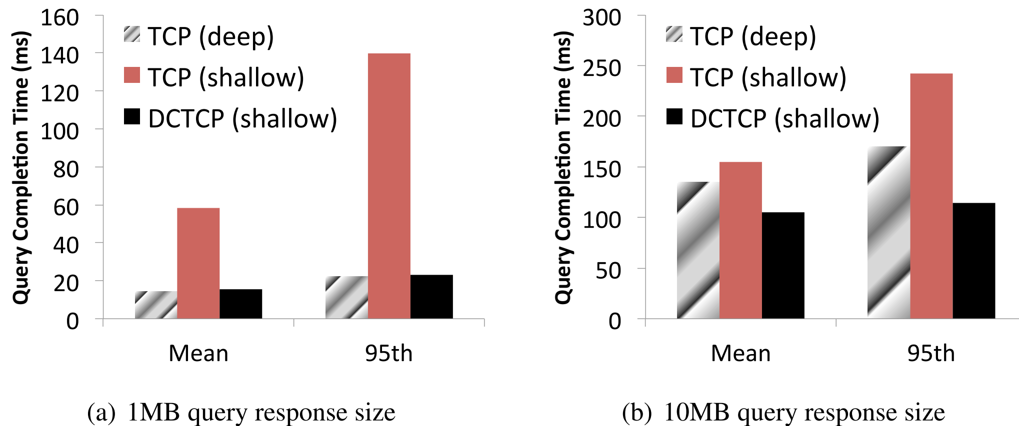


Figure 2.20: All-to-all incast. 41 servers simultaneously participate in 40-to-1 incast patterns. DCTCP along with dynamic buffer allocation work well.

an easy case for the dynamic buffer management to handle, as there is only one receiver, so only one port gets congested. But what happens when there are multiple simultaneous incasts going on, as occurs in the production clusters? To investigate, we use the same setup as before, except all 41 machines participate. Each machine requests 25KB of data from the remaining 40 machines (total 1MB). In other words, 41 incast experiments happen all at once. The CDF of the response time is shown in Figure 2.20. We see that DCTCP keeps the demand on buffer space low enough that the dynamic buffering is able to cover all requests for memory and DCTCP suffers no timeouts at all. TCP, on the other hand, performs poorly, because over 55% of the queries suffer from at least one timeout.

Other settings: We also investigated TCP and DCTCP's performance with incast traffic patterns in scenarios including 10Gbps links and larger (10MB) and smaller response sizes (100KB). The results are qualitatively similar to those presented here. As shown in Figure 2.21, repeating the experiments with our CAT4948 deep-buffered switch, we found a reduction in TCP's incast problem for small response sizes ($<$ 1MB), but the problem resurfaces for larger response sizes (10MB). DCTCP performs well at all response sizes. Besides, the deep buffers allow for larger queue buildups, which hurts performance of other traffic — we examine this in detail in Section 2.3.3.

In summary, *DCTCP handles incast without timeouts until there are so many senders that the traffic sent in the first RTT overflows the buffers.* Its performance is further enhanced by the dynamic buffering offered by modern switches.



(a) 1MB query response size (b) 10MB query response size

Figure 2.21: All-to-all incast with deep-buffered (CAT4948) and shallow-buffered (Triumph) switches. Deep buffers prevent incast induced timeouts for the 1MB response size, but the problem reappears with 10MB responses. DCTCP handles the incast well in both cases even with the shallow-buffered switch. All schemes use RTO_{min} = 10ms.

Queue buildup

The second impairment scenario in Section 2.1.3 involves big and small flows mixing in the same queue, with the queue build-up caused by the big flow increasing the latency of the small flows. To measure DCTCP's impact on this impairment, we connect 4 machines to the Triumph switch with 1Gbps links. One machine is a receiver and the other three are senders. We start one large TCP flow each from the two senders to the receiver (recall that the 75th percentile of concurrent connections measured in our data center was 2 (Figure 2.4). The receiver then requests 20KB chunks of data from the third sender. The sender responds immediately, and the receiver sends the next request as soon as it finishes receiving data. All communication is over long-lived connections, so there is no three-way handshake for each request.

Figure 2.22 shows the CDF of request completion times for 1000 20KB transfers. DCTCP's completion time (median delay $<$ 1 ms) is much lower than TCP (median delay 19ms). No flows suffered timeouts in this scenario, so reducing RTO_{min} would not reduce the delay. Since the amount of data transferred is small, the completion time is dominated by the round trip time, which is dominated by the queue length at the switch.

Thus, *DCTCP improves latency for small flows by reducing queue lengths, something*

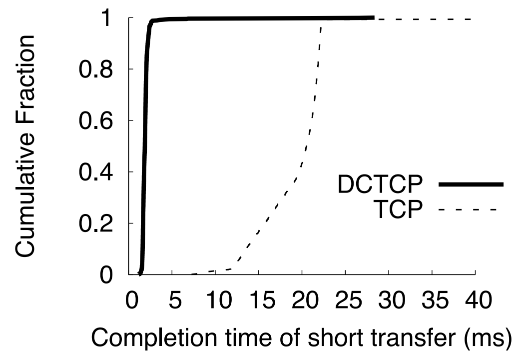


Figure 2.22: Queue buildup caused by large flows significantly increases the completion time of short transfers for TCP. With DCTCP, the short transfers complete quickly nonetheless because the queue occupancy in the switch is kept small.

reducing RTO_{min} does not affect.

Buffer pressure

The third impairment scenario in Section 2.1.3 involves flows on one set of interfaces increasing the rate of packet loss on other interfaces. Recall that these are shared-memory switches. With TCP, long flows use up the shared buffer space, which leaves less headroom to absorb incast bursts. DCTCP should alleviate this problem by limiting the queue build up on the interfaces used by long flows. To evaluate this, we connect 44 hosts to a Triumph switch with 1Gbps links. 11 of these hosts participate in a 10:1 incast pattern, with 1 host acting as a client (receiver), and 10 hosts acting as servers (senders). The client requests a total of 1MB data from the servers (100KB from each), repeating the request 10,000 times. We have seen (Figure 2.19), that the switch can easily handle 10:1 incast with both TCP and DCTCP, without inducing any timeouts.

Next, we use the remaining 33 hosts to start "background" traffic of long-lived flows to consume the shared buffer. We start a total of 66 large flows between the 33 hosts, with each host sending data to two other hosts. Recall that queue buildup only occurs when there is more than 1 flow, which measurements of our cluster show happens 25% of the time (Section 2.1). Table 2.2 shows the 95th percentile of query completion times.

We see that with TCP, query completion time is substantially worse in the presence of

	Without background traffic	With background traffic
TCP	9.87ms	46.94ms
DCTCP	9.17ms	9.09ms

Table 2.2: 95th percentile of query completion time with and without buffer pressure from background traffic. DCTCP prevents background traffic from affecting performance of query traffic. $RTO_{min} = 10ms$, $K = 20$.

long-lived flows, while DCTCP's performance is unchanged. This performance difference is due to timeouts: about 7% of queries suffer from timeouts with TCP, while only 0.08% do so under DCTCP. Long-lived flows get the same throughput under both TCP and DCTCP.

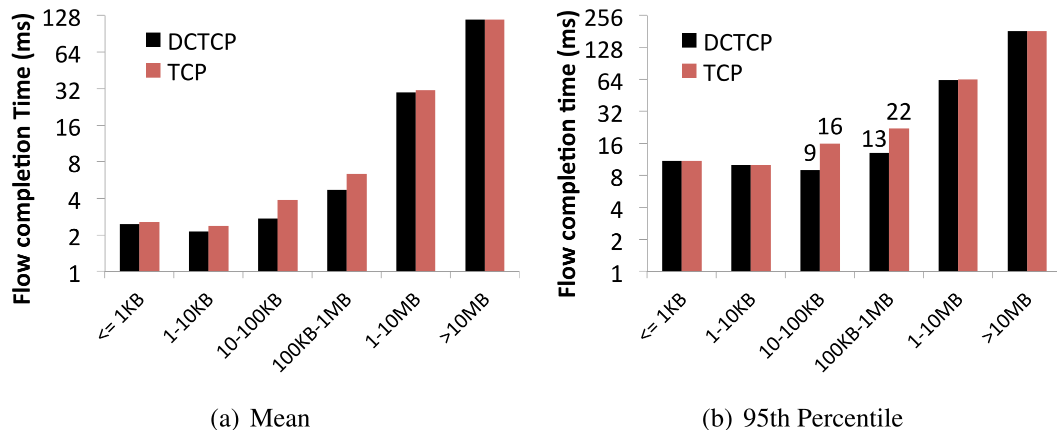
This experiment shows that *DCTCP improves the performance isolation between flows by reducing the buffer pressure that would otherwise couple them.*

In summary, these microbenchmarks allow us to individually test DCTCP for fairness, high throughput, high burst tolerance, low latency, and high performance isolation while running at 1 and 10Gbps across shared-buffer switches. On all these metrics, DCTCP significantly outperforms TCP.

2.3.3 Benchmark Traffic

We now evaluate how DCTCP would perform under the traffic patterns found in production clusters (Section 2.1.2). For this test, we use 45 servers connected to a Triumph top of rack switch by 1Gbps links. An additional server is connected to a 10Gbps port of the Triumph to act as a stand-in for the rest of the data center, and all inter-rack traffic is directed to/from this machine. This aligns with the actual data center, where each rack connects to the aggregation switch with a 10Gbps link.

We generate all three types of traffic found in the cluster: query, and short and long background traffic. Query traffic is created following the Partition/Aggregate structure of the real application by having each server draw from the query interarrival time distribution (Figure 2.2(a)) and send a query to *all* other servers in the rack, each of which then send back a 2KB response ($44 \times$ 2KB $\approx$ 90KB total response size). For the background traffic (short-messages and long flows), each server draws independently from the background interarrival time distribution (Figure 2.2(b)) and the flow size distribution (Figure 2.3), and



(a) Mean (b) 95th Percentile

Figure 2.23: Completion time of background traffic in cluster benchmark. Note the log scale on the Y axis.

chooses an endpoint so the ratio of inter-rack to intra-rack flows is the same as measured in the cluster.[6] We carry out these experiments using TCP and DCTCP, with RTO_{min} set to 10ms for both. For DCTCP experiments, K was set to 20 on 1Gbps links and to 65 on the 10Gbps link. Dynamic buffer allocation was used in all cases. We generate traffic for 10 minutes, comprising over 200,000 background flows and over 188,000 queries.

Both query and short-message flows are time critical, their metric of interest is completion time. The RTT (i.e., queue length) and timeouts affect this delay the most. For large flows in the background traffic (e.g., data updates), the throughput of the network is the main consideration.

Figure 2.23 shows the mean and 95th percentile of completion delay for background traffic, classified by flow sizes. The 90% confidence intervals of the mean are too tight to be shown. Short-messages benefit significantly from DCTCP, as flows from 100KB-1MB see a 3ms benefit at the mean and a 9ms benefit at 95th percentile. The background traffic did not suffer any timeouts with either protocol in this experiment. Thus, the *lower latency for short-messages is due to DCTCP's amelioration of queue buildup.*

Figure 2.24 shows query completion time statistics. DCTCP performs better than TCP, especially at the tail of the distribution. The reason is a combination of timeouts and high

[6] Background flows have some structure (e.g., pattern of polling other workers for updates), so using two independent distributions instead of a joint distribution is an approximation.

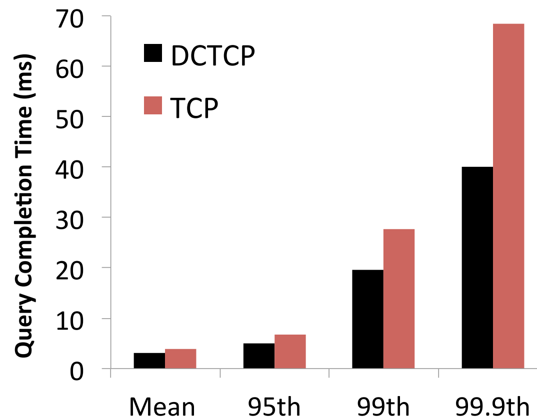


Figure 2.24: Completion time of query traffic in cluster benchmark.

queueing delay. With TCP, 1.15% of the queries suffer retransmission timeouts. No queries suffer from timeouts with DCTCP.

Scaled traffic: The previous benchmark shows how DCTCP performs on today's workloads. However, as explained in Section 2.1.3, the traffic parameters we measured reflect extensive optimization conducted by the developers to get the existing system into the tight SLA bounds on response time. For example, they restrict the size of query responses and update frequency, thereby trading off response quality for response latency. This naturally leads to a series of "what if" questions: how would DCTCP perform if query response sizes were larger? Or how would DCTCP perform if background traffic characteristics were different? We explore these questions by scaling the traffic in the benchmark.

We begin by asking if using DCTCP instead of TCP would allow a 10X increase in both query response size and background flow size without sacrificing performance. We use the same testbed as before. We generate traffic using the benchmark, except we increase the size of update flows larger than 1MB by a factor of 10 (most bytes are in these flows, so this effectively increases the volume of background traffic by a factor of 10). Similarly, we generate queries as before, except that the total size of each response is 1MB (with 44 servers, each individual response is just under 25KB). We conduct the experiment for both TCP and DCTCP.

To see whether other solutions, besides DCTCP, may fix TCP's performance problems, we tried two variations. First, we replaced the shallow-buffered Triumph switch with the

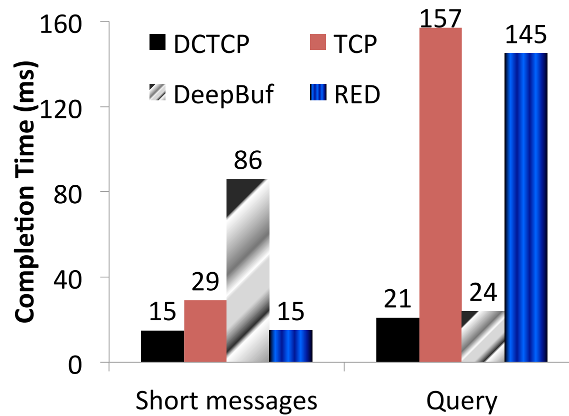


Figure 2.25: 95th percentile completion time for short messages and query traffic in the scaled cluster benchmark with 10x background and 10x query traffic.

deep-buffered CAT4948 switch.[7] Second, instead of drop tail queues, we used RED with ECN. It was as difficult to tune RED parameters at 1Gbps as it was previously at 10Gbps: after experimentation, we found that setting min_th = 20, max_th = 60, and using the recommendations of Floyd [49] for the remaining parameters gave the best performance.

Figure 2.25 shows the 95th percentile of response times for the short messages (100KB-1MB) and the query traffic (the mean and other percentiles are qualitatively similar). The results show DCTCP performs significantly better than TCP for both update and query traffic. The 95th percentile of completion time for short-message traffic improves by 14ms, while query traffic improves by 136ms. With TCP, over 92% of the queries suffer from timeouts, while only 0.3% suffer from timeouts with DCTCP.

In fact, short message completion time for DCTCP is essentially unchanged from the baseline (Figure 2.23(b)) and, even at 10X larger size, only 0.3% of queries experience timeouts under DCTCP; in contrast, TCP suffered 1.15% timeouts even for the less demanding baseline traffic. Thus, DCTCP can handle substantially more traffic without any adverse impact on performance.

deep-buffered switches have been suggested as a fix for TCP's incast problem, and we indeed see that with the deep-buffered CAT4948 switch, TCP's query completion time is comparable to DCTCP, since less than 1% of queries suffer from timeout. However, if

[7] CAT4948 does not support ECN, so we can't run DCTCP with it.

deep buffers are used, the short-message traffic is penalized: their completion times are over 80ms, which is substantially higher than TCP without deep buffers (DCTCP is even better). The reason is that deep buffers allow more queue buildup.

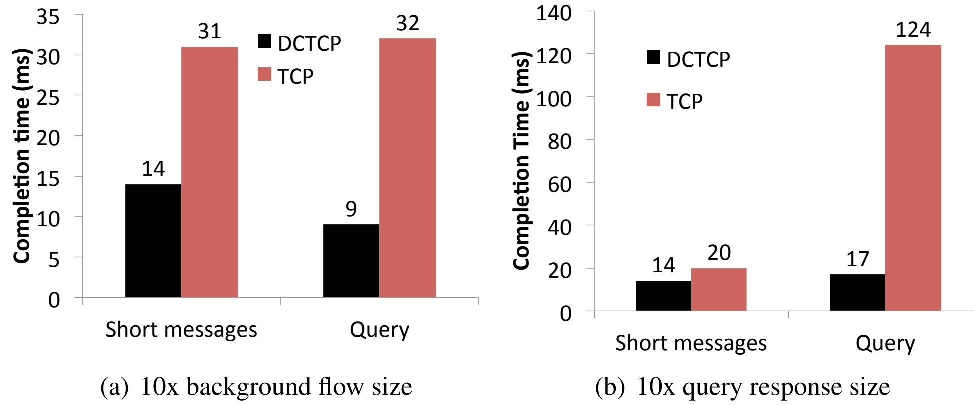
We also see that RED (with ECN marking) alone does not solve TCP's problems: while RED improves performance of short transfers by keeping average queue length low, the high variability of the queue length (illustrated in Figure 2.15) leads to poor performance for the query traffic (95% of queries experience a timeout). Another possible factor is that RED marks packets based on average queue length, and is thus slower to react to bursts of traffic caused by query incast.

These results make three key points: *First, if our data center used DCTCP it could handle 10X larger query responses and 10X larger background flows while performing better than it does with TCP today. Second, while using deep-buffered switches (without ECN) improves performance of query traffic, it makes performance of short transfers worse, due to queue build up. Third, while RED improves performance of short transfers, it does not improve the performance of query traffic, due to queue length variability.*

Benchmark variations: The intensity of our benchmark traffic can be varied either by increasing the arrival rate of the flows or by increasing their sizes. We have explored both dimensions, but the results are similar, so we report primarily on increases in flow sizes. Specifically, we now present the results for the two corners: scaling the background traffic while holding query traffic to the original benchmark size, and vice versa.

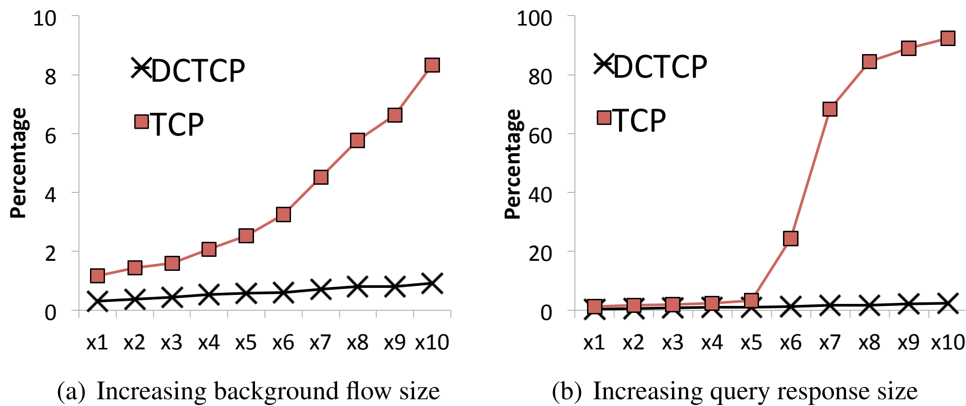
Figure 2.26(a) shows that increasing the size of background traffic hurts the performance of both short messages and query traffic. As described in Section 2.1.3, large flows cause both queue buildup delays and buffer pressure, which DCTCP mitigates. Figure 2.26(b) shows that increasing the size of the query responses by a factor of 10 severely degrades the completion time of queries with TCP. However, DCTCP handles the increased traffic without significant impact on the performance.

The reason is that DCTCP reduces incast timeouts. Figure 2.27 shows the fraction of queries that suffer timeouts with increasing background flow size and increasing query response size. We observe that increasing background traffic gradually increases buffer pressure and causes timeouts (Figure 2.27(a)), but the impact on TCP is greater than DCTCP. Also, for TCP, the fraction of queries that suffer timeouts rises sharply with increasing



(a) 10x background flow size (b) 10x query response size

Figure 2.26: 95th percentile of completion time for the cluster benchmark scaled by (a) increasing the background flow size by 10x; (b) increasing the query response size by 10x.



(a) Increasing background flow size (b) Increasing query response size

Figure 2.27: Fraction of queries that suffer at least one timeout with (a) increasing background flow size; (b) increasing query response size. Note the different Y axis ranges for the two plots.

query response size (Figure 2.27(b)). After the response size exceeds 800KB, almost all queries suffer timeouts. DCTCP handles the intensifying incast much more gracefully.

2.4 Related Work

The literature on congestion control is vast. We focus on recent developments most directly related to our work.

Incast: Incast problems are well-known to data center application developers, who typically resort to solutions that increase the median response time while curbing the higher percentiles; for example, jittering (as in Section 2.1.3) or batching (pulling responses in small batches) [103]. Recently, researchers have shown [158, 35] that lowering the $RTO_{\min}$ to 1ms (from today's values of 200–300ms) and using high-resolution retransmission timers alleviates the impact of incast induced timeouts. While not decreasing the number of packet drops, this measure reduces the time required by TCP to recover and retransmit the lost packets. However, it does not prevent queue buildup, and hence does not address latency issues due to queueing delay (Section 2.1.3). Alternatively, deep-buffered switches can avoid packet drops, but also dramatically increase the latency for short flows as these encounter the very long queues created by long flows (Section 2.3.3). Further, deep-buffered switches increase the cost of switches, as they require off-chip memory for packet buffers.

DCTCP avoids queue buildup and prevents timeouts — even with shallow-buffered switches — and hence does not have the problems of these approaches. However, we stress that DCTCP *can* be used with deep-buffered switches and/or small retransmission timeouts. In fact, because DCTCP maintains small steady-state queue occupancy, a deep-buffered switch provides larger buffer headroom to absorb bursts. Also, as we observed in our experiments (Section 2.3.2), in extreme cases with small static switch buffers, DCTCP flows may experience timeouts. In these situations, a small retransmission timeout helps.

ICTCP [166] is another recent proposal that, like DCTCP, proactively avoids packet drops and timeouts. ICTCP uses a receiver side congestion control algorithm which adjusts the TCP receiver window based on the available bandwidth at the last-hop to the receiver. ICTCP is effective in preventing packet drops at the last-hop, but cannot easily handle incast at links other than the last-hop, and is not designed to avoid queue buildup in switches.

Congestion control: Several TCP variants aim to reduce queue length at routers, including: delay based congestion control (e.g., Vegas [31] and high speed variants such as CTCP [154]), explicit feedback schemes (e.g., RCP [44], XCP [96], and the ECN-based VCP [169] and BMCC [133]), and AQM schemes (e.g., RED [51] and PI [76]). In data centers, increases in queuing delay must be measured in microseconds to avoid queue buildup effectively. This is difficult to achieve at the server because of the noise induced by factors like process scheduling in the kernel and interrupt coalescing in the NIC. Hence, end-to-end measurements of queueing delay may not provide a reliable congestion signal, and an explicit congestion signal like ECN is more appropriate. AQM schemes are difficult to tune and, as we have shown, do not perform well in the absence of statistical multiplexing with unmodified TCP sources (Section 2.2.4). Schemes like RCP and XCP require switches to do more complex operations and are not commercially available.

Much research has been devoted, with success, to improving TCP performance in networks with high bandwidth-delay product, including High-speed TCP [50], BIC-TCP [170], CUBIC [70], and FAST [163]. While many of these variants respond less drastically to packet loss, just like DCTCP does, they do not seek to maintain small queue length and their analysis often assumes a high degree of statistical multiplexing, which is not the norm in a data center environment. Other special transport schemes for networks with very small buffers (e.g., optical networks) like E-TCP [66] seek to maintain high utilization by deliberately inducing loss and not reacting to it. E-TCP does not address retransmission delays experienced by short flows or incast.

QCN [6] is a Layer 2 congestion control algorithm that has been developed by the Data Center Bridging (DCB) Task Group as the IEEE 802.1Qau standard [136] for data center Ethernet. Chapter 5 is devoted to QCN, so we only briefly contrast QCN with DCTCP here. QCN requires hardware rate limiters (implemented on the NICs) and also hardware support from switches. To reduce cost, QCN must lump flows into a single or a few rate-limiters, which then share fate, leading to possible collateral damage to flows that do not share bottleneck links with congested flows. Moreover, since QCN is a Layer 2 scheme, it cannot cross Layer 3 boundaries, which are common in today's data centers.

DCTCP's control scheme was influenced by an earlier algorithm, ECN-hat [93], and has similarities with one of the earliest ECN schemes, DECbit [139]. It differs from DECbit in

the way the AQM feedback is smoothed (filtered) across time. For instance, in DECbit, the router averages the queue length over recent cycles, while DCTCP uses a simple threshold and delegates the smoothing of the feedback across time to the host (sender). An important benefit of this is that the senders can each smooth the feedback over their respective RTTs, whereas at the router, the smoothing must occur over a conservative timescale that is larger than the worst-case RTT of all flows; hence, making the control loop more sluggish.

Deadline-aware schemes: A number of recent proposals [164, 77, 156, 11] attempt to leverage flow deadline information to partition bandwidth among flows in accordance with their deadlines. These schemes are intrinsically more sophisticated and can provide better performance than the fair bandwidth sharing provided by TCP and DCTCP because they essentially use extra information to *schedule* flows. While deadline information may be available in some scenarios, it is challenging to obtain in general. Hence, a generic and efficient congestion control scheme such as DCTCP that provides high throughput and low latency is indispensable in most data centers. Moreover, these proposals require fairly complex operations at the switch that are not available in switching silicon today.

2.5 Final Remarks

In this chapter, we proposed a new variant of TCP, called Data Center TCP (DCTCP). Our work was motivated by detailed traffic measurements from a $\sim$6000 server data center cluster, running production soft real time applications. We observed several performance impairments, and linked these to the behavior of the commodity switches used in the cluster. We found that to meet the needs of the observed diverse mix of short and long flows, switch buffer occupancies need to be persistently low, while maintaining high throughput for the long flows. We designed DCTCP to meet these needs. DCTCP relies on Explicit Congestion Notification (ECN), a feature now available on commodity switches. DCTCP succeeds through use of the multi-bit feedback derived from the series of ECN marks, allowing it to react early to congestion. A wide set of detailed experiments at 1 and 10Gbps speeds showed that DCTCP meets its design goals.

Chapter 3

Analysis of DCTCP

In the previous chapter, we presented the DCTCP congestion control algorithm and showed through extensive experimental evaluation and simulations that DCTCP achieves full throughput while maintaining a very low buffer occupancy compared to TCP. This allowed DCTCP to simultaneously provide low latency and good burst tolerance for the short flows, and high throughput for the long flows.

The key mechanisms used by DCTCP are a simple active queue management scheme at the switch, based on Explicit Congestion Notification (ECN) [138], and a window control scheme at the source which reacts to ECN marks by reducing the window size in proportion to the *fraction* of packets that are marked. The performance of DCTCP is determined by two parameters: (i) K, the marking threshold for queue occupancy at the switch above which all packets are marked; and (ii) g, the weight used for exponentially averaging ECN marks at the source. See Section 2.2 for details.

In this chapter, we undertake a comprehensive mathematical analysis of the DCTCP algorithm and make the following major contributions:

- **Fluid model:** We derive a fluid model for the DCTCP control loop in Section 3.1. The model comprises of a system of nonlinear delay-differential equations. Using ns2 [126] simulations, we find that the fluid model is very accurate and that it is more accurate across a wider range of parameters than the "Sawtooth model" presented in the previous chapter (Section 2.2.3). A key step in developing the fluid model is accurately capturing the bursty "on-off" style marking at the switch which DCTCP

employs. This is similar to the difficulty of modeling TCP–Drop-tail using fluid models [152].

- **Steady state:** We analyze the steady state behavior of DCTCP using the fluid model in Section 3.2. Due to its on-off style marking function, the fluid model does not have a fixed point. Instead, it exhibits a (periodic) limit cycle behavior in steady state. Theorem 3.1 provides a necessary and sufficient condition for local stability of the limit cycles using the so-called *Poincaré map* technique [22, 99, 120, 162]. We verify this condition numerically for a wide range of parameter values. We then explore the throughput and delay performance of DCTCP by explicitly evaluating the limit cycle solutions. Here we find that for DCTCP to achieve 100% throughput, the marking threshold, K, needs to be $\sim$17% of the bandwidth-delay product. For smaller values of K, we determine the throughput loss; i.e., we obtain the throughput-delay tradeoff curve. A key result is that DCTCP's throughput remains higher than 94% even as $K \to 0$. This is much higher than the limiting throughput of 75% for a TCP source as the buffer size goes to zero [160].

- **Convergence:** We analyze how quickly DCTCP flows converge to their fair equilibrium window sizes (equivalently, sending rates) in Section 3.3. This is important to determine since DCTCP reduces its sending rate by factors much smaller than TCP; therefore, DCTCP sources may take much longer to converge. Theorem 3.2 gives the following explicit characterization: For N flows (with identical RTTs) sharing a single bottleneck link, the window sizes at the nth congestion episode, $W_i(T_n)$, converge to the fair share, W^*, as follows:

  $$|W_i(T_n) - W^*| < O(n^2) \exp\left(-\beta_{DCTCP} T_n\right),$$

  where an explicit expression is given for β_{DCTCP}. Using this, we compare the rate of convergence of TCP and DCTCP and obtain the following bounds:

  $$\beta_{DCTCP} < \beta_{TCP} < 1.4 \times \beta_{DCTCP}.$$

  Thus, even though DCTCP converges slower than TCP, its rate of convergence is at

most 1.4 times slower.

We note that the convergence results are obtained using a different model from the fluid model, since the fluid model is not suitable for conducting transient analyses. This new model of DCTCP, which we call the Hybrid Model, employs continuous- and discrete-time variables and is similar to the AIMD models used in the analysis of TCP–Drop-tail [24, 146, 147]. While the AIMD models are linear and can be used to determine convergence rates via an analysis of the eigenvalues of linear operators [146, 147], the Hybrid Model is more challenging because it is nonlinear.

- **RTT-fairness:** We investigate the fairness properties of DCTCP for flows with diverse RTTs in Section 3.4. RTT-fairness is defined as the ratio of the throughput achieved by two groups of flows as a function of the ratio of their RTTs [170, 106, 32, 14]. Using ns2 simulations, we find that DCTCP's RTT-fairness is better than TCP–Drop-tail but worse than TCP with RED [51] marking. We identify a very simple change to the DCTCP algorithm, suggested by the fluid model, which considerably improves its RTT-fairness. The modified DCTCP is shown to have linear RTT-fairness ($Throughput \propto RTT^{-1}$) and achieve a better fairness than TCP–RED.

- **Parameter guidelines:** Our analysis of DCTCP's steady state and convergence properties yields guidelines for choosing the algorithm parameters which we summarize here. Let C and d respectively denote the bottleneck capacity (in packets/sec) and propagation delay (in seconds). Then,

$$\mathbf{K \approx 0.17Cd}, \tag{3.1}$$

$$\frac{\mathbf{5}}{\mathbf{Cd+K}} \lessapprox \mathbf{g} \lessapprox \frac{\mathbf{1}}{\sqrt{\mathbf{Cd+K}}}. \tag{3.2}$$

For example, when $C = 10$Gbps and $d = 300\mu$s, assuming 1500Byte packets, K needs to be about 42 packets and $0.02 \lessapprox g \lessapprox 0.06$.

These guidelines supersede the recommendations of equations (2.12) and (2.13) in Section 2.2.3 that were based on the simplistic Sawtooth model of DCTCP.

3.1 DCTCP Fluid Model

We now develop a fluid model of DCTCP for N long-lived flows traversing a single bottleneck switch port with capacity C. The following non-linear, delay-differential equations describe the dynamics of the window size, $W(t)$, the source's estimate of the fraction of marked packets, $\alpha(t)$, and the queue occupancy at the switch, $q(t)$:

$$\frac{dW}{dt} = \frac{1}{R(t)} - \frac{W(t)\alpha(t)}{2R(t)} p(t - R^*), \tag{3.3}$$

$$\frac{d\alpha}{dt} = \frac{g}{R(t)} \left(p(t - R^*) - \alpha(t) \right), \tag{3.4}$$

$$\frac{dq}{dt} = N\frac{W(t)}{R(t)} - C. \tag{3.5}$$

Here $p(t)$ indicates the packet marking process at the switch and is given by

$$p(t) = \mathbb{I}_{\{q(t)>K\}}, \tag{3.6}$$

and $R(t) = d + q(t)/C$ is the round-trip time (RTT), where d is the propagation delay (assumed to be equal for all flows), and $q(t)/C$ is the queueing delay.

Equations (3.3) and (3.4) describe the DCTCP source, while (3.5) and (3.6) describe the queue occupancy process at the switch and the DCTCP marking scheme. The source equations are coupled with the switch equations through the packet marking process $p(t)$ which gets fed back to the source with some delay. The feedback delay is the round-trip time $R(t)$, and varies with $q(t)$. However, as a simplification and similar to prior work [74, 113], we use the approximate fixed value $R^* = d + K/C$ for the feedback delay. The approximation aligns well with DCTCP's attempt to strictly hold the queue occupancy at around K packets.

Equation (3.3) models the window evolution and consists of the standard additive increase term, $1/R(t)$, and a multiplicative decrease term, $-W(t)\alpha(t)/2R(t)$. The latter term models the source's reduction of its window size by a factor $\alpha(t)/2$ when packets are marked (i.e., $p(t - R^*) = 1$), and this occurs once per RTT. Equation (3.4) is a continuous-time approximation of the exponentially weighted moving average used by DCTCP sources

to compute α (equation (2.1)). Finally, equation (3.5) models the queue occupancy evolution: $N \cdot W(t)/R(t)$ is the net input rate and C is the service rate.

Remark 3.1. In standard TCP fluid models [152, 116, 75, 113], the multiplicative decrease term is given by

$$-\frac{W(t)}{2}\frac{W(t-R^*)}{R(t-R^*)}p(t-R^*), \tag{3.7}$$

with the interpretation that $W(t-R^*)/R(t-R^*) \times p(t-R^*)$ is the rate at which marked ACKs arrive at the source, each causing a window reduction of $W(t)/2$. As Misra *et al.* [116] explains, this model is accurate when the packet marks can be assumed to occur as a Poisson process, for example, when the RED [51] algorithm marks packets. In DCTCP, the marking process is bursty, as captured by the function $p(t)$ at equation (3.6). Hence, the natural adaptation of (3.7):

$$-\frac{W(t)\alpha(t)}{2}\frac{W(t-R^*)}{R(t-R^*)}p(t-R^*),$$

is not valid for DCTCP.

3.1.1 Comparison with Packet-level Simulations

Figure 3.1 compares the fluid model with packet-level simulations using the ns2 [126] simulator. The parameters used are: $C = 10$Gbps, $d = 100\mu$s, $K = 65$ packets, and $g = 1/16$ (K and g are chosen to match those of the 10Gbps experiments in Section 2.3 of Chapter 2). The fluid model plots correspond to the evolution of $W(\cdot)$ and $\alpha(\cdot)$, whereas the ns2 plots show the the window size and α of *each* of the N sources, for $N = 2, 10, 100$. In the case $N = 100$, the queue size climbs to about 120 packets, even though the marking threshold, K, is 65. Thus *every* arriving packet is marked at the switch buffer (captured by $\alpha = 1$) to signal congestion to the 100 sources. The window size at each source is reduced to the minimum, equal to 2 packets, corresponding to a sending rate of 100 Mbps per source.[1]

[1] Note that, when $N = 100$, the RTT equals 240 microseconds, since $d = 100$ microseconds and $q/C = 140$ microseconds. With these numbers, the window size of 2 packets implies that the sending *rate* of each source, W/RTT, equals 100Mbps, as required.

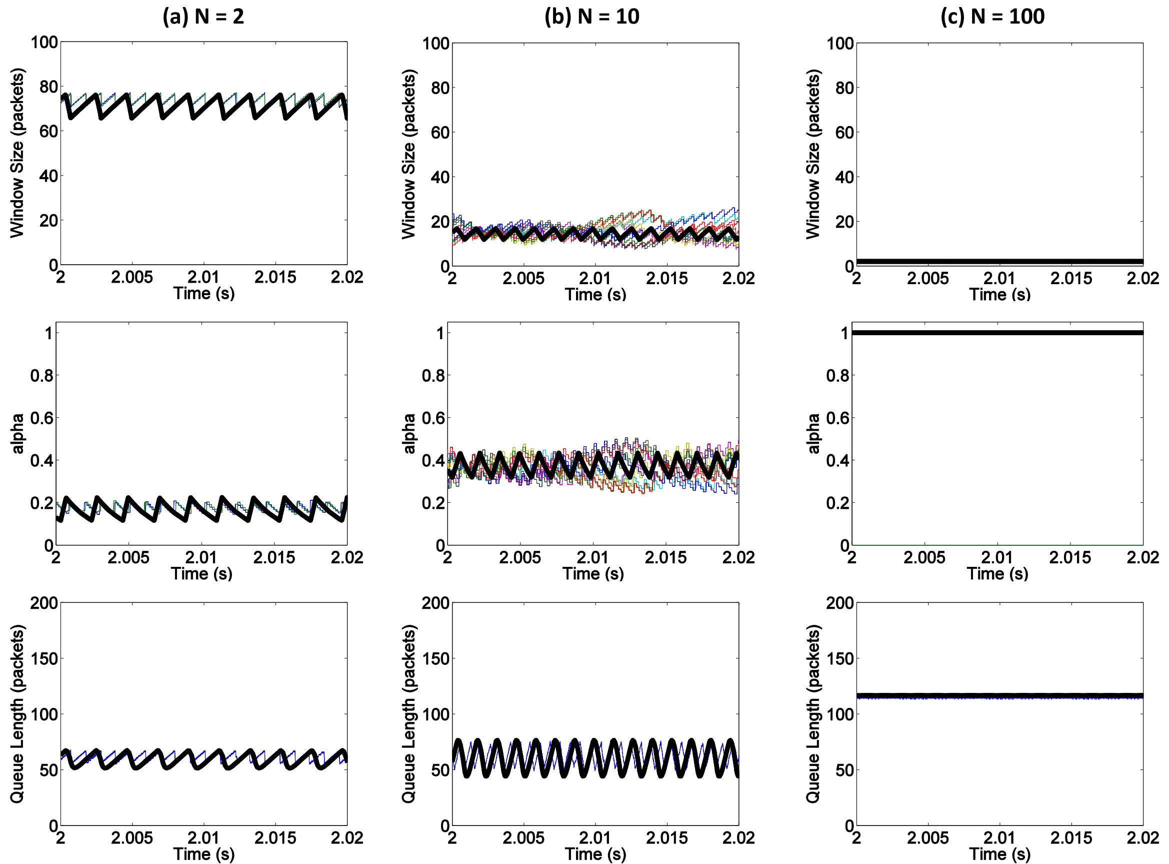


Figure 3.1: Comparison between the DCTCP fluid model and ns2 simulations. The parameters are: $C = 10$Gbps, $d = 100\mu$s, $K = 65$ packets, and $g = 1/16$. The fluid model results are shown in solid black.

As can be seen, the fluid model matches the simulations quite well. We have also compared the model with simulations for a wide range of line speeds, propagation delays, number of sources, and DCTCP parameters. Our findings are that the model has very good fidelity and that, as is typical for this type of model, the fidelity improves with the number of sources.

Comparison with the Sawtooth model: In Chapter 2, we presented a simplified model of the steady state behavior of long-lived DCTCP flows sharing a single bottleneck. We call this the Sawtooth model. It is based on the assumption that the window sizes of the N flows and the queue size are perfectly synchronized, and can be described by periodic

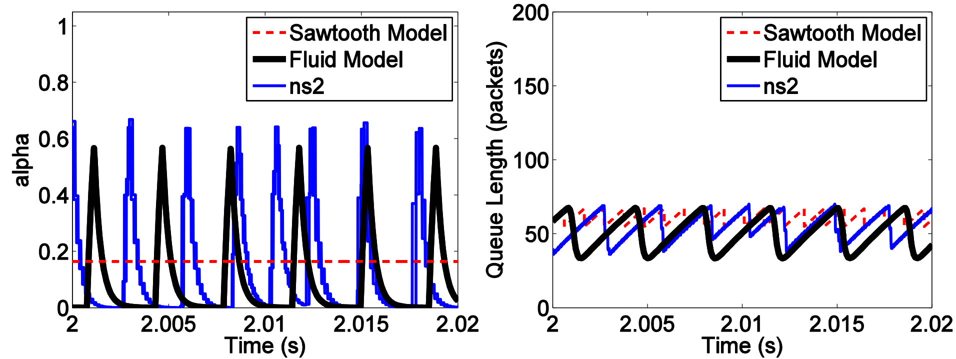


Figure 3.2: Comparison of the fluid and Sawtooth models with ns2 simulations for $N = 2$ sources and $g = 0.4$. The fluid model is much more accurate when g is not very small.

sawtooth processes. This makes it possible to compute the steady state fraction of packets marked at the switch, which can be used to completely specify the sawtooth processes in the model.

The Sawtooth model provides simple closed form approximations to quantities of interest, such as the amplitude of queue oscillations, leading to some guidelines on how to set the DCTCP parameters (Section 2.2.3). But the accuracy of the Sawtooth model rests heavily on the validity of the synchronization assumption. As mentioned in Section 2.2.3, this holds when N is small (e.g., less than 10), but is not valid for large N. Further, the Sawtooth model assumes that sources know the exact fraction of marked packets and does not include the variations in α resulting from the exponentially weighted moving average at the source (equation (2.1)). In particular, the model is only accurate for small values of g and does not capture the effect of g on system dynamics.

This is demonstrated in Figure 3.2 which repeats the previous simulation with $N = 2$, this time setting $g = 0.4$. Because the Sawtooth model assumes α is a constant, its prediction of the queue size evolution is very poor. In contrast, the fluid model captures the fluctuations in α and is, therefore, much more accurate.

3.2 Steady State Analysis

In this section, we use the fluid model to analyze the steady sate behavior of the DCTCP control loop. We first normalize the fluid model and show that its dynamics is completely determined by two quantities. It is then shown that the model exhibits two distinct equilibrium behaviors depending on these quantities. In the typical regime, it converges to periodic limit cycles in equilibrium. We prove the local stability of the limit cycles, and explicitly evaluate them to characterize the throughput and delay performance of DCTCP.

3.2.1 The Normalized Fluid Model

We change variables

$$\tilde{W}(t) = W(R^*t),\ \tilde{\alpha}(t) = \alpha(R^*t),\ \tilde{q}(t) = \frac{q(R^*t) - K}{CR^*}, \tag{3.8}$$

and rewrite the fluid model equations (3.3)–(3.6):

$$\frac{d\tilde{W}}{dt} = \frac{1}{1+\tilde{q}(t)}(1 - \frac{\tilde{W}(t)\tilde{\alpha}(t)}{2}\tilde{p}(t-1)), \tag{3.9}$$

$$\frac{d\tilde{\alpha}}{dt} = \frac{g}{1+\tilde{q}(t)}(\tilde{p}(t-1) - \tilde{\alpha}(t)), \tag{3.10}$$

$$\frac{d\tilde{q}}{dt} = \frac{1}{\bar{w}}\frac{\tilde{W}(t)}{(1+\tilde{q}(t))} - 1. \tag{3.11}$$

Here

$$\tilde{p}(t) = \mathbb{I}_{\{\tilde{q}(t)>0\}}, \tag{3.12}$$

and $\bar{w} = (Cd + K)/N$ is the average per-flow window size. Henceforth, we refer to this as the normalized fluid model. The normalized fluid model immediately reveals that while the DCTCP fluid model has 5 parameters (C, N, d, K, and g), the system dynamics is completely determined by just two quantities: (i) the average per-flow window size $\bar{w}$, and (ii) the parameter g.

We now discuss the existence of possible fixed points for the normalized fluid model.

A fixed point of the system, $(\tilde{W}, \tilde{\alpha}, \tilde{q})$, must satisfy the following equations:

$$1 - \frac{\tilde{W}\tilde{\alpha}}{2}\tilde{p} = 0, \quad \tilde{p} - \tilde{\alpha} = 0, \quad \frac{1}{\bar{w}}\frac{\tilde{W}}{(1+\tilde{q})} - 1 = 0, \tag{3.13}$$

where $\tilde{p} = \mathbb{I}\{\tilde{q} > 0\}$. The above equations have a solution only if $\bar{w} \leq 2$. Therefore, we have the following two operating regimes:

(i) $\bar{\mathbf{w}} \leq \mathbf{2}$: In this regime, the normalized fluid model has a unique fixed point, namely $(\tilde{W}, \tilde{\alpha}, \tilde{q}) = (2, 1, \frac{2}{\bar{w}} - 1)$. Equivalently, (3.3)–(3.6) has the fixed point $(W, \alpha, q) = (2, 1, 2N - Cd)$. This regime corresponds to the large N case, where the system has a very simple steady state behavior: each source transmits two packets per RTT, of which Cd fill the link capacity, and the remaining $2N - Cd$ build up a queue. All packets are marked as the queue constantly remains larger than K. The $N = 100$ case of Figure 3.1 is in this regime.

(ii) $\bar{\mathbf{w}} > \mathbf{2}$: In this regime, the system does not have a fixed point. Rather, it has a periodic solution or limit cycle,[2] characterized by a closed trajectory in state space. Figure 3.3 shows a sample phase diagram of the limit cycle projected onto the $\tilde{W}$–$\tilde{q}$ plane for $\bar{w} = 10$ and $g = 1/16$. As shown, all trajectories evolve toward the orbit of the limit cycle. Both the $N = 2$ and $N = 10$ cases of Figure 3.1 are in this regime.

The regime $\bar{w} \leq 2$ will no longer be discussed since it is trivial to show that the system converges to the unique fixed point. In the next two sections, we study the stability and structure of the limit cycle solution when $\bar{w} > 2$.

3.2.2 Stability of Limit Cycle

The analysis of the stability of limit cycles is complicated and few analytical tools exist. Fortunately, a wealth of computational tools are available [120], and it is possible to compute the periodic solution of the system and determine its stability properties numerically.

[2]Formally, a periodic solution is said to be a limit cycle if there are no other periodic solutions sufficiently close to it. In other words, a limit cycle corresponds to an isolated periodic orbit in the state space [120].

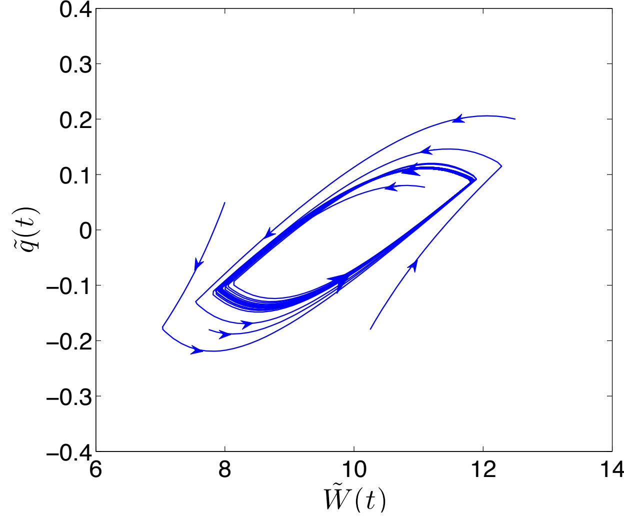


Figure 3.3: Phase diagram showing occurrence of limit cycles in the (normalized) DCTCP fluid model for $\bar{w} = 10$, $g = 1/16$ (the projection on the $\tilde{W}$–$\tilde{q}$ plane is shown).

A limitation of the computational approach is that it requires sweeping the different parameters of the model to completely characterize the dynamics. However, our task is greatly simplified by the fact that the dynamics of the normalized model is completely determined by just two parameters: $\bar{w}$, and g. We proceed by defining stability of limit cycles.

Let X^* denote the set of points in the state space belonging to the limit cycle. We define an ϵ-neighborhood of X^* by

$$U_\epsilon = \{x \in \mathbb{R}^n \mid \mathrm{dist}(x, X^*) < \epsilon\},$$

where $\mathrm{dist}(x, X^*)$ is the minimum distance from x to a point in X^*; i.e.,

$$\mathrm{dist}(x, X^*) = \inf_{y \in X^*} ||x - y||.$$

Definition 3.1. *The limit cycle X^* is*

(i) ***stable*** *if for any $\epsilon > 0$, there exists a $\delta > 0$ such that*

$$x(0) \in U_\delta \Rightarrow x(t) \in U_\epsilon, \ \forall t \geq 0.$$

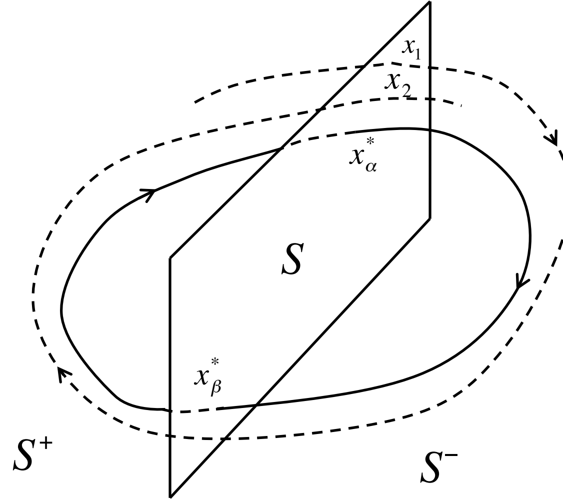


Figure 3.4: Periodic system trajectory and the Poincaré map.

(ii) ***locally asymptotically stable*** *if it is stable and δ can be chosen such that*

$$x(0) \in U_\delta \Rightarrow \lim_{t\to\infty} \operatorname{dist}(x(t), X^*) = 0.$$

Figure 3.3 shows an example of local asymptotical stability for our system: orbits initiated from points close enough to the limit cycle are attracted to it.

To proceed, let $x(t) = (\tilde{W}(t), \tilde{\alpha}(t), \tilde{q}(t))^T$ denote the trajectory in the state space for the normalized fluid model equations (3.9)–(3.12). It is convenient to represent the fluid model as:

$$\dot{x}(t) = F(x(t), u(t-1)), \quad u(t) = \mathbb{I}_{\{cx(t)>0\}}, \tag{3.14}$$

where $c = [0, 0, 1]$ and $u(t) = \tilde{p}(t)$ is the system feedback.

Poincaré map: We analyze the stability of the limit cycle via the 'Poincaré map', which we introduce after making some definitions. Define the *switching plane* as $S = \{x \in \mathbb{R}^3 : cx = 0\}$ and let $S^+ = \{x \in \mathbb{R}^3 : cx > 0\}$ and $S^- = \{x \in \mathbb{R}^3 : cx < 0\}$ (see Figure 3.4 for an illustration). Note that the switching plane is the $\tilde{q} = 0$ (equivalently, $q = K$) plane and corresponds to the DCTCP marking threshold. The limit cycle crosses the switching plane twice in each period, once from S^+ (at the point x_α^*) and once from S^- (at the point x_β^*).

The Poincaré map traces the evolution of the system when the trajectory crosses the

switching plane in a given direction. More precisely, let the successive intersections of the trajectory $x(t)$ with S in direction S^+ to S^- be denoted by x_i. The Poincaré map

$$x_{i+1} = P(x_i),$$

maps the ith intersection to the subsequent one. Note that the Poincaré map is well-defined for our system, because equations (3.9)–(3.12) guarantee that starting from any point in S^+ or S^-, the trajectory will eventually intersect with S.

The fixed point of the Poincaré map corresponds to the intersection of the limit cycle with the switching plane, say at x^*_α. Therefore, local stability of the Poincaré map at x^*_α implies local stability of the limit cycle. We refer to [22, 99, 120, 162] for more details on the Poincaré map technique.

Next, there are two main steps to establish:

Step 1: Show that the Poincaré map is locally stable. Theorem 3.1, which is an adaptation of Theorem 3.3.1 in [162] to our setting, provides a necessary and sufficient condition for local stability of the Poincaré map, and consequently, the limit cycle.

Step 2: Verify that our system satisfies the condition of Theorem 3.1 for a wide range of $\bar{w}$ and g.

Before proceeding with the statement of Theorem 3.1, we need the following definitions. Let $x^*(t)$ denote the trajectory of the limit cycle of system (3.14). Assume that $x^*(t)$ traverses the switching plane from S^+ to S^- at time $t_0 = 0$, i.e. $x^*(0) = x^*_\alpha$, and that the period of the limit cycle is $T = (1 + h_\alpha) + (1 + h_\beta)$ with $h_\alpha > 0$ and $h_\beta > 0$, where $1 + h_\alpha$ (resp. $1 + h_\beta$) is the time taken for the trajectory to move from x^*_α to x^*_β (resp. from x^*_β to x^*_α). For notational convenience, define $u_\alpha = 0$ and $u_\beta = 1$. Let

$$Z_1 = \left(I - \frac{F(x^*_\alpha, u_\beta)c}{cF(x^*_\alpha, u_\beta)}\right) \exp\left(\int_{1+h_\alpha}^{T} J_F(x^*(s), u(s-1))ds\right),$$

$$Z_2 = \left(I - \frac{F(x^*_\beta, u_\alpha)c}{cF(x^*_\beta, u_\alpha)}\right) \exp\left(\int_{0}^{1+h_\alpha} J_F(x^*(s), u(s-1))ds\right),$$

where J_F is the Jacobian matrix of F with respect to x, and I is the identity matrix. The integral of the matrix J_F is entry wise. Note that Z_1 and Z_2 are 3×3 matrices. We assume

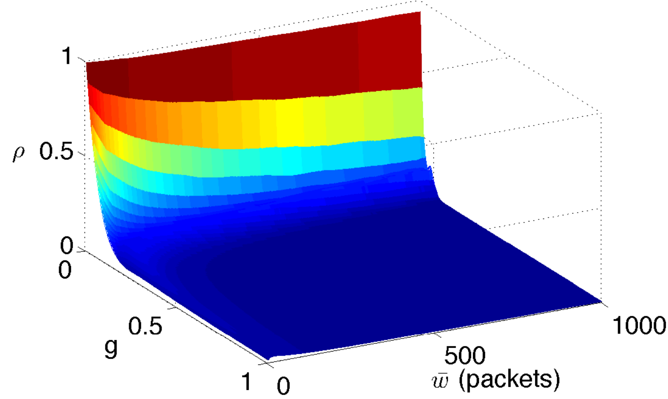


Figure 3.5: Limit cycle stability (Theorem 3.1). Stability is verified numerically for $g \in [0.001, 1]$, and $\bar{w} \in [2.01, 1000]$.

that $cF(x^*_\alpha, u_\beta) \neq 0$ and $cF(x^*_\beta, u_\alpha) \neq 0$; i.e., $x^*(t)$ is non-tangent with the switching plane at the traversing points.

Theorem 3.1. *The Poincaré map (and the limit cycle) for system* (3.14) *is locally asymptotically stable if and only if*

$$\rho(Z_1 Z_2) < 1.$$

Here, $\rho(\cdot)$ is the spectral radius.

The proof is provided in Appendix A.

Verification: We use Theorem 3.1 to verify the stability of DCTCP's limit cycle. Since Z_1 and Z_2 do not have a closed form, we sweep the parameters $\bar{w}$ and g in the ranges of interest and compute $\rho(Z_1 Z_2)$ numerically. This has been done in Figure 3.5 for the range $g \in [0.001, 1]$, and $\bar{w} \in [2.01, 1000]$. Throughout this range, $\rho < 1$, and local stability is verified. We conjecture that the limit cycle is actually globally stable for all $g \in (0, 1]$, $\bar{w} > 2$.

3.2.3 Steady State Throughput & Delay

In this section, we study the key performance metrics of throughput and delay of DCTCP, by analyzing the limit cycle solution of equations (3.9)–(3.12).

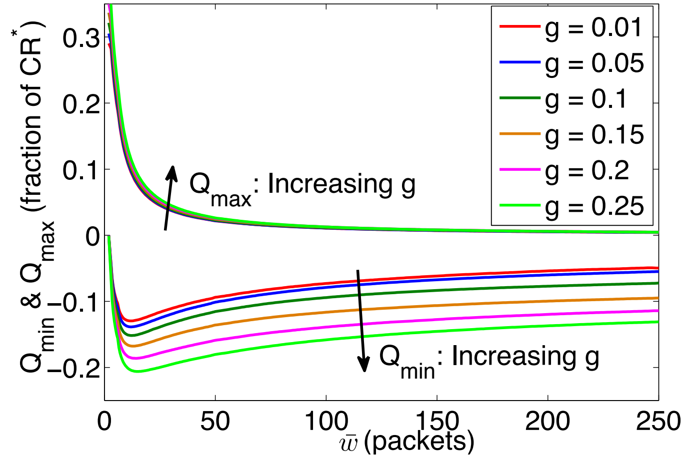


Figure 3.6: Queue undershoot and overshoot. The values are found by numerically computing the limit cycles of the (normalized) DCTCP fluid model for a range of $\bar{w}$ and g.

A standard method for approximately determining the amplitude and frequency of limit cycles is the so-called *Describing Function (DF)* [99] method. Unfortunately, the DF method applied to system (3.9)–(3.12) yields very poor results. This is because a key assumption of the DF method — that the limit cycle can be well-approximated by a single frequency sinusoid — does not hold for this system. We therefore evaluate the exact limit cycle solutions numerically.

Achieving 100% throughput

The first question we consider is: How much buffering is required for DCTCP to achieve 100% throughput? Since queue underflow must be avoided for 100% throughput, we need to determine how large the queue size oscillations are about the operating point K.

Assume $\{(\tilde{W}(t), \tilde{\alpha}(t), \tilde{q}(t))|0 \leq t < T\}$ is the limit cycle of our system (with period T). Define $Q_{min} = \min_{0 \leq t < T} \tilde{q}(t)$, and $Q_{max} = \max_{0 \leq t \leq T} \tilde{q}(t)$ to be the maximum and minimum excursions of the (normalized) queue size during a period. In Figure 3.6, we plot Q_{min} and Q_{max} against $\bar{w}$ for some values of g. We make three main observations:

(i) The queue overshoot is not sensitive to g and increases as $\bar{w}$ decreases. This follows because the queue overshoot is primarily determined by the rate at which flows increase their window size, and as $\bar{w}$ decreases, the window increase rate of 1 packet/RTT

per source becomes increasingly large compared to the bandwidth-delay product.

(ii) There is a worst case $\bar{w}$ (about 12-16 packets for the range of g values shown in Figure 3.6) at which the queue undershoot is maximized. This implies an interesting property of DCTCP: as per-flow window sizes increase — for example, due to higher and higher link speeds — DCTCP requires less buffers as a fraction of the bandwidth-delay product to achieve high utilization.

(iii) The amplitude of queue undershoot increases as g increases. This is to be expected: high values of g cause large fluctuations in α which inhibit DCTCP's ability to maintain a steady sending rate (see Figure 3.2 for an example). It is important to note that although α will cease to oscillate as $g \to 0$, queue size oscillations cannot be made arbitrary small by lowering g. In fact, in Figure 3.6, all $g \leq 0.01$ values basically produce the same curve.

Choosing K: As seen in Figure 3.6, $Q_{min} \gtrapprox -0.15$ when g is sufficiently small (the choice of g is discussed next). Therefore, to avoid queue underflow, we require:

$$K > |Q_{min}|CR^* \gtrapprox 0.15CR^*.$$

Substituting $R^* = d + K/C$ in this inequality gives the following guideline:

$$\mathbf{K \approx 0.17Cd}. \tag{3.15}$$

In words, about 17% of the bandwidth-delay product of buffering is needed for 100% throughput; any more available buffer can be used as headroom to absorb bursts. Note that this value for K is quite close to the guideline $K > (1/7)Cd$ (equation (2.12)), derived using the Sawtooth model in Section 2.2.3. This shows that the simple Sawtooth model is valid for small g.

Limit cycle period & an upper limit for g: Figure 3.7 plots the period of oscillations of the limit cycle. The figure suggests that the period grows as $\sqrt{\bar{w}}$ (for small g). Now, the marking process $\{p(t-1)\}$ is a periodic 'signal' (with period equal to the period of the limit cycle), which is input to the low-pass filter defined at equation (3.10). Since the filter

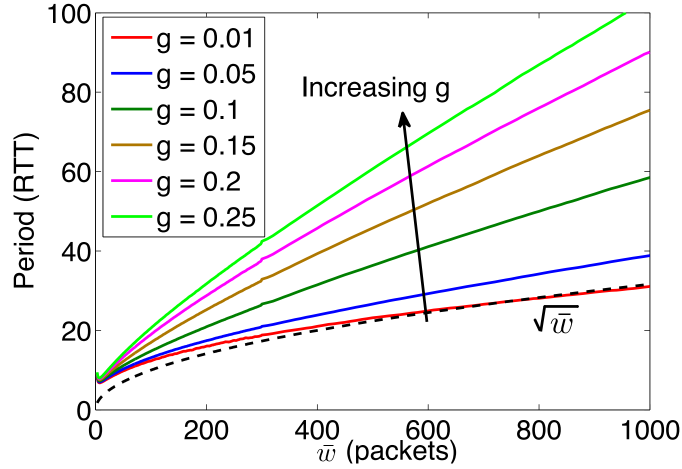


Figure 3.7: Period of the DCTCP limit cycle. For small g, the period grows with $\sqrt{\bar{w}}$.

has a cutoff frequency of about g, for it to be effective, it is necessary that g be smaller than the primary oscillation frequency of the signal, $1/\sqrt{\bar{w}}$. But $\bar{w} = (Cd + K)/N$ and is maximized for $N = 1$. Therefore, we get the following bound:

$$\mathbf{g} \lessapprox \frac{\mathbf{1}}{\sqrt{\mathbf{Cd} + \mathbf{K}}}. \tag{3.16}$$

This guideline is also in agreement with equation (2.13) that was previously derived based on the Sawtooth model. Appendix C provides yet a different justification for (3.16) based on the Hybrid Model, which we introduce in Section 3.3 to analyze the convergence rate of DCTCP. Our analysis of the convergence rate leads to a corresponding lower limit for the parameter g.

Throughput-Delay tradeoff

We have seen that if K is larger than 17% of the bandwidth-delay product, DCTCP achieves 100% throughput. However, when we require very low queueing delays, or when switch buffers are extremely shallow, it may be desirable to choose K smaller than this. Inevitably, this will result in some loss of throughput. We are interested in quantifying how much throughput is lost, and in effect, deriving a throughput-delay tradeoff curve for DCTCP.

A more accurate model of the switch queue occupancy is needed to study throughput

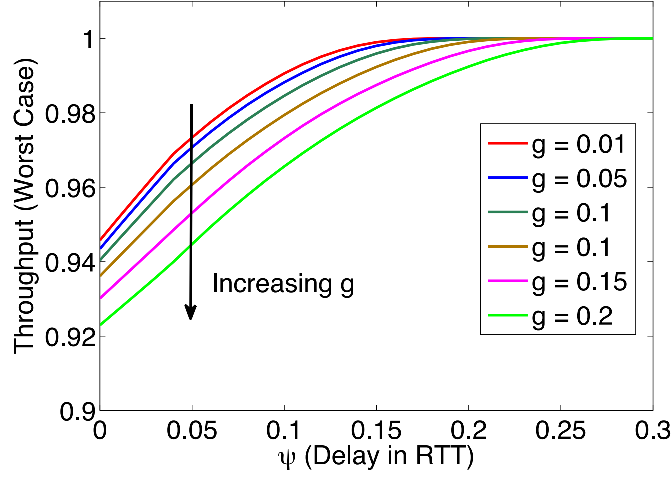


Figure 3.8: Worst case throughput vs delay. DCTCP achieves more than 94% throughput for $g < 0.1$ even as $\psi \to 0$ (equivalently, $K \to 0$).

loss. Equation (3.11) ignores the fact that a real queue will never become negative. To account for this and capture the correct behavior when the queue underflows, we define $\psi \triangleq K/(Cd)$, and replace (3.11) with:

$$\frac{d\tilde{q}}{dt} = \begin{cases} \frac{1}{\bar{w}} \frac{\tilde{W}(t)}{(1+\tilde{q}(t))} - 1 & \tilde{q}(t) > \frac{-\psi}{1+\psi}, \\ \max\left(\frac{1}{\bar{w}} \frac{\tilde{W}(t)}{(1+\tilde{q}(t))} - 1, 0\right) & \tilde{q}(t) = \frac{-\psi}{1+\psi}. \end{cases}$$

We can now explore the limit cycle solution as we vary ψ, and compute the average throughput (over period T):

$$Throughput = \frac{1}{T} \int_0^T \frac{\tilde{W}(t)}{\bar{w}(1 + \tilde{q}(t))} dt.$$

In Figure 3.8, we plot the worst case throughput as ψ is varied. At each value of ψ, the $\bar{w}$ which yields the lowest throughput is used. It should be noted that since the queue size is maintained near the marking threshold, the ψ axis also (roughly) corresponds to the average queueing delay. As expected, when $\psi \gtrapprox 0.17$, 100% throughput is achieved (for small g). As ψ is lowered, the throughput decreases, but *is always at least 94% for* $g < 0.1$. This indicates that very small marking thresholds can be used in DCTCP, with only a minor

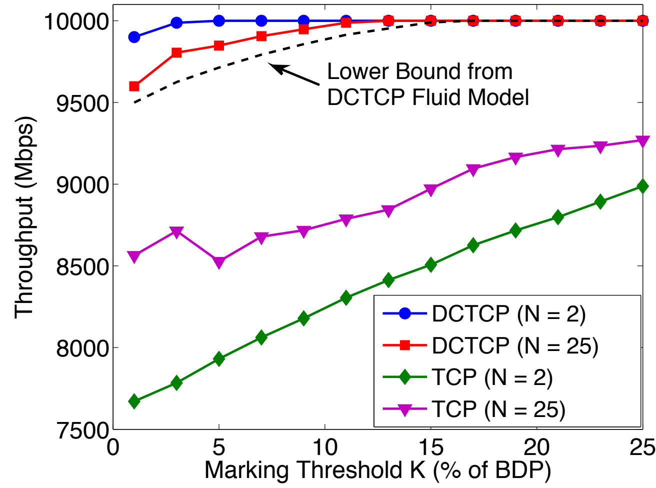


Figure 3.9: Throughput vs marking threshold, K, in ns2 simulations. K is varied from 4 to 100 packets (1–25% of the bandwidth-delay product). As predicted by the analysis, the throughput with DCTCP remains higher than 94% even for K as small as 1% of the bandwidth-delay product. In contrast, the throughput for TCP with ECN marking at low thresholds approaches 75%.

loss in throughput. We verify this next through ns2 simulations.

Simulations

We use ns2 simulations to evaluate the throughput achieved as the marking threshold K is varied. We choose $C = 10$Gbps and $d = 480\mu$s, for a bandwidth-delay product of 400 packets (each 1500 Bytes), and set $g = 0.05$. We consider two cases with $N = 2$ and $N = 25$ long-lived flows. The results are shown in Figure 3.9. For reference, we also report the results for TCP in the same scenarios.

The simulations clearly show that DCTCP achieves high throughput, even with marking threshold as low as 1% of the bandwidth-delay product. In all cases, we find the throughput is indeed higher than the worst case lower bound predicted by the fluid model. Of course, we see a much worse throughput loss for TCP with ECN marking. An interesting observation is that unlike TCP, whose throughput improves as we increase the number of flows, DCTCP gets lower throughput with $N = 25$ flows than $N = 2$ flows. This is actually predicted by our analysis, because with DCTCP, the worst case queue size fluctuations

(and throughput loss) occur when the per-flow window size is around 10-20 packets (see Figure 3.6 and observation (ii) in Section 3.2.3).

3.3 Convergence Analysis

DCTCP uses the multi-bit information derived from estimating the fraction of marked packets to reduce its window size by factors smaller than two. As we have seen in the previous section, this allows it to very efficiently utilize shallow buffers, achieving both high throughput and low queueing delays. However, the reduced multiplicative decrease factors mean slower convergence times: it takes longer for a flow with a large window size to relinquish bandwidth to a flow with a small window size.

We argued in Chapter 2 that since the convergence time is proportional to the RTT for window-based algorithms and the RTTs in a data center are only a few 100s of microseconds, the actual time to converge is not substantial relative to the transfer time of large data files. Also, based on simulations, we reported that the convergence time of DCTCP is about a factor 2 slower than TCP. The results of this section show that this is, indeed, correct in general.

Our aim in is to derive rigorous bounds for the rate of convergence of DCTCP. We consider how fast N DCTCP flows with identical RTTs, starting with arbitrary window sizes and values of α, converge to their share of the bottleneck bandwidth.[3] Since this is an analysis of the system in transience, the fluid model of the previous section is inadequate. Instead, we use a hybrid (continuous- and discrete-time) model based on the AIMD models introduced by Baccelii *et al.* [24] and Shorten *et al.*. [146]. A key difference between our model and the AIMD models is that ours is non-linear because it models the DCTCP-style multiplicative decrease, whereas the models in prior work [24, 146] are linear since they correspond to a constant decrease factor (equal to 2 for TCP).

[3]Identical RTTs ensure that in equilibrium, each flow gets $(1/N)^{th}$ of the bottleneck bandwidth. We discuss the bias against flows with longer RTTs in Section 3.4.

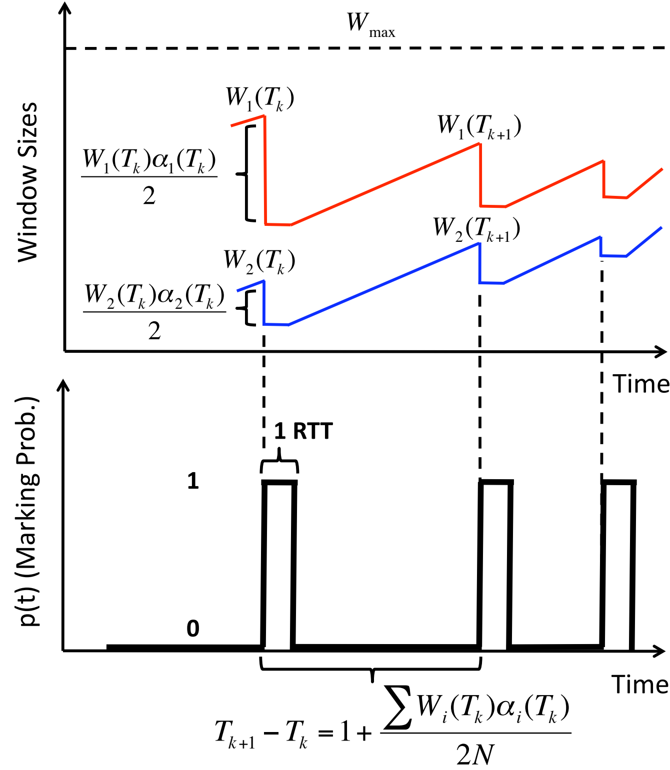


Figure 3.10: Hybrid Model. The window sizes are shown for 2 flows. Note that $\sum W_i(T_k) = W_{max}$ for all k.

3.3.1 The Hybrid Model

Consider N DCTCP flows whose window size and value of α at time t (measured in units of RTT) are denoted by $W_i(t)$ and $\alpha_i(t)$. Assume the window sizes of the N flows are synchronized; i.e., each flow reduces its window at every congestion event. See Figure 3.10 for an illustration.

Let $W_{max} = Cd + K$. When $\sum_{i=1}^{N} W_i(t) < W_{max}$, all window sizes increase linearly with slope 1 packet/RTT. Once $\sum_{i=1}^{N} W_i(t) = W_{max}$, a congestion event occurs and each flow cuts its window size according to

$$W_i(t) \leftarrow W_i(t)\left(1 - \frac{\alpha_i(t)}{2}\right).$$

Note that packets are marked ($p(t) = 1$) for 1 RTT after the window reductions because

of the feedback delay. Assume the kth congestion event occurs at time T_k. Since the total reduction in $\sum_{i=1}^{N} W_i(t)$ at T_k is equal to $\sum_{i=1}^{N} W_i(T_k)\alpha_i(T_k)/2$, which is regained at the rate of N packets/RTT, the duration of the kth "congestion epoch" is

$$\Delta T_k \triangleq T_{k+1} - T_k = 1 + \frac{\sum_{i=1}^{N} W_i(T_k)\alpha_i(T_k)}{2N}, \tag{3.17}$$

and we have:

$$W_i(T_{k+1}) = \left(1 - \frac{\alpha_i(T_k)}{2}\right) W_i(T_k) + \Delta T_k - 1. \tag{3.18}$$

It only remains to specify the evolution of $\alpha_i(t)$. This is simply given by:

$$\frac{d\alpha_i}{dt} = g(p(t) - \alpha_i(t)).$$

In particular, $\alpha_i(T_{k+1})$ is the solution of the following initial value problem at time ΔT_k:

$$\frac{dx}{dt} = g(p(t) - x(t)), \quad x(0) = \alpha_i(T_k)$$
$$p(t) = \begin{cases} 1 & 0 \le t < 1 \\ 0 & t \ge 1 \end{cases}$$

Using this, it is not difficult to show:

$$\alpha_i(T_{k+1}) = e^{-g\Delta T_k}(e^g - 1 + \alpha_i(T_k)). \tag{3.19}$$

3.3.2 Rate of Convergence

We make the following assumptions regarding the system parameters:

$$N \ge 2, \quad W_{max} \ge 2N, \quad g \le \frac{1}{\sqrt{W_{max}}}. \tag{3.20}$$

The first two assumptions are natural for studying convergence, and the third is in accordance with the guideline of the previous section (equation (3.16)) for the parameter g. The main result of this section is given by the following theorem.

Theorem 3.2. *Consider N DCTCP flows evolving according to* (3.18) *and* (3.19)*, with parameters satisfying* (3.20)*. Suppose that $W_i(0)$ and $\alpha_i(0)$ are arbitrary. Let $W^* = W_{max}/N$, and $0 < \alpha^* \leq 1$ be the unique positive solution of*

$$\alpha^* = e^{-g(1+W^*\alpha^*/2)}(e^g - 1 + \alpha^*). \tag{3.21}$$

Then $W_i(T_n) \to W^*$ and $\alpha_i(T_n) \to \alpha^*$ for all $1 \leq i \leq N$ as $n \to \infty$. Moreover:

$$|W_i(T_n) - W^*| < 2W_{max}\Big(1 + g^6 n^2\Big)e^{-\beta(T_n - T_{P2})}, \tag{3.22}$$

for all $1 \leq i \leq N$, where:

$$\beta = \min\Big(g, \frac{-\log(1-\alpha^*/2)}{1+W^*\alpha^*/2}\Big), \tag{3.23}$$

$$T_{P2} = \frac{\log(2W^*/g^6)}{g} + 2(1 + W^*/2). \tag{3.24}$$

The proof of Theorem 3.2 is given in Section 3.3.4. Here, we make a few remarks about the convergence rate. The crucial term in (3.22) is the exponential decay $e^{-\beta T_n}$. The convergence rate is therefore chiefly determined by (3.23), which has the following interpretation. Two things occur (simultaneously) during convergence of DCTCP: (i) the $\alpha_i(T_n)$ converge to α^*, and (ii) the $W_i(T_n)$ converge to W^*. It is shown in the proof of Theorem 3.2 that (i) happens with rate g, and (ii) with rate

$$\gamma = \frac{-\log(1-\alpha^*/2)}{1+W^*\alpha^*/2}.$$

The overall convergence rate is determined by the slower of the two.

Lower limit for g: We have seen that g should be smaller than $1/\sqrt{W_{max}}$ to not adversely affect steady state performance. Equation (3.23) suggests a lower bound for g, in order to not slow down convergence. Note that:

$$\frac{1}{W^*} < \gamma \approx \frac{-2\log(1-\alpha^*/2)}{W^*\alpha^*} < \frac{2\log 2}{W^*}. \tag{3.25}$$

Therefore if $g > (2\log 2)/W^*$, it will not limit the convergence rate. Of course, in practice, we don't know the number of flows N, and so don't know W^*. However, in data centers, there are typically a small number of large flows active on a path. Recall, for example, our measurements in Section 2.1.2 of Chapter 2 that showed that at most 4 flows larger than 1MB were concurrently active at a server in any 50ms period.[4] Therefore, we mainly need to focus on the small N case, for which convergence is also slowest. With these considerations, we propose the following guideline:

$$\frac{\mathbf{5}}{\mathbf{W_{max}}} \lessapprox \mathbf{g} \lessapprox \frac{\mathbf{1}}{\sqrt{\mathbf{W_{max}}}}. \tag{3.26}$$

Comparison with TCP: As mentioned in the beginning of this section, DCTCP converges slower than TCP. But how much slower is DCTCP? It is straight forward to show that the convergence rate of TCP is given by:

$$\beta_{TCP} = \frac{\log 2}{1 + W^*/2} \approx \frac{2\log 2}{W^*}.$$

Therefore, (3.23) and (3.25) imply that for g properly chosen:

$$\beta_{DCTCP} < \beta_{TCP} < (2\log 2)\beta_{DCTCP} \approx 1.4 \times \beta_{DCTCP}.$$

This means that the DCTCP convergence rate is at most 40% slower than TCP. It is important to note however that this is a statement about the *asymptotic* rate of convergence. In practice, all the terms present in (3.22) affect the convergence time. Therefore, in simulations, the actual time to converge is about a factor 2 larger for DCTCP.

Bounds on α^*: The following bounds are proven in Appendix C.

$$\frac{1}{2}\sqrt{\frac{2}{W^*}} - g < \alpha^* < \sqrt{\frac{2}{W^*}} + g.$$

[4] Note that only the large flows need to be considered, as only they can possibly converge.

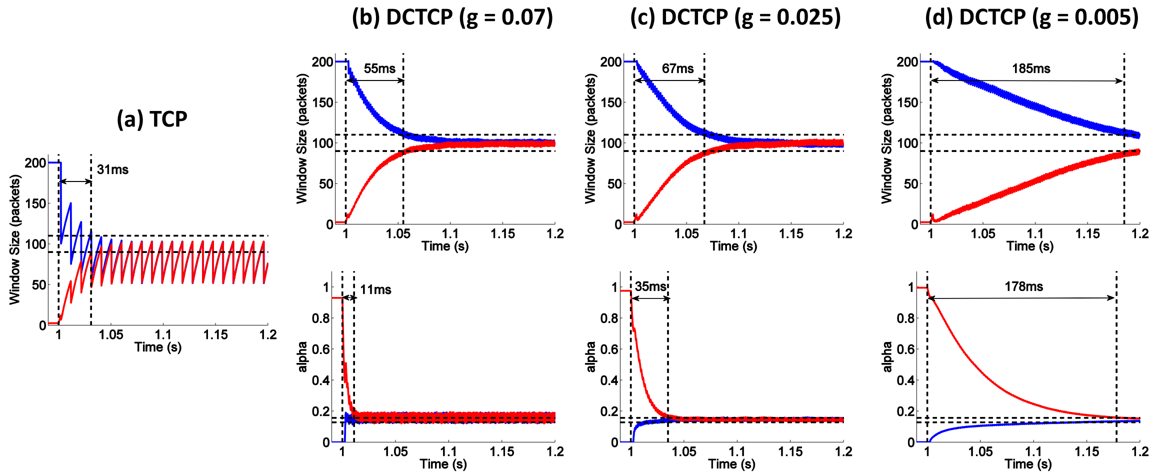


Figure 3.11: ns2 simulation of convergence time for 2 flows. The chosen g values for DCTCP correspond to: (b) $1/\sqrt{W_{max}}$, (c) $5/W_{max}$, and (d) $1/W_{max}$.

3.3.3 Simulations

We have verified the results of our analysis using extensive ns2 simulations. Figure 3.11 shows a representative example with 2 flows. The choice of parameters $C = 10$Gbps, $d = 200\mu$s, and $K = 35$ (or 17% of the BDP) gives $W_{max} \approx 200$ packets. One flow starts at the beginning of the simulation, and grabs all the available capacity (its window size reaches 200). At time 1sec, a second flow begins with a window of 1 packet, and we are interested in how long it takes for both window sizes to get within 10% of the fair share value (100 packets) for the first time. In order to have the worst case convergence time with DCTCP, we begin the second flow with $\alpha = 1$, whereas the first flow has $\alpha = 0$ before time 1sec; so the second flow actually cuts its window by much larger factors initially.

We test three values of g for DCTCP. With $g = 1/\sqrt{W_{max}}$, as expected from the analysis, the α variables converge much quicker than the window sizes (about 5x faster). Reducing g to $5/\sqrt{W_{max}}$, the convergence times of the α and window sizes get closer, with the α still converging about twice as fast. Here, the increase in convergence time is small (67ms up from 55ms). But when g is further reduced to $1/W_{max}$, the convergence of α and window sizes take about the same amount of time, showing that the limiting factor is now the convergence of α. In this case, the small value of g significantly increases the total convergence time (185ms). A final observation is that when g is appropriately chosen

according to the guideline in equation (3.26), the convergence time of DCTCP is indeed up to about a factor of 2 longer than TCP.

3.3.4 Proof of Theorem 3.2

The key idea in proving Theorem 3.2 is to consider convergence as happening in three separate phases.

Phase 1: Initially, the α_i values get close to each other. In fact, (3.19) implies that for any i and j:

$$|\alpha_i(T_n) - \alpha_j(T_n)| \leq e^{-gT_n}|\alpha_i(T_0) - \alpha_j(T_0)| \leq e^{-gT_n} \tag{3.27}$$

We have the following simple Lemma.

Lemma 3.1. *Let* $\bar{\alpha}(T_k) = \sum_{i=1}^{N} \alpha_i(T_k)/N$. *Then:*

$$|\alpha_i(T_n) - \bar{\alpha}(T_n)| \leq e^{-gT_n}$$

for all $1 \leq i \leq N$.

Proof. This follows by applying the triangle inequality and using (3.27). □

We take Phase 1 to last until time $T_{P1} \triangleq log(2W^*)/g$, so that:

$$|\alpha_i(T_k) - \bar{\alpha}(T_k)| \leq e^{-gT_k} = \frac{1}{2W^*}e^{-g(T_k - T_{P1})}. \tag{3.28}$$

Phase 2: The second phase begins with $T_k \geq T_{P1}$. In this phase, the α_i converge to a positive constant α^*. The following proposition is the main convergence result for Phase 2 and is proved in Appendix B.

Proposition 3.1. *For* $n \geq 1$, *and all* $1 \leq i \leq N$*:*

$$|\alpha_i(T_n) - \alpha^*| \leq A_0 n e^{-g(T_n - T_{P1})}, \tag{3.29}$$

where $A_0 = e^{2g(1+W^*/2)}$.

We take Phase 2 to last until time

$$T_{P2} \triangleq T_{P1} + 2(1 + W^*/2) - 6\log(g)/g,$$

so that:

$$|\alpha_i(T_k) - \alpha^*| \leq A_0 k e^{-g(T_k - T_{P1})} = g^6 k e^{-g(T_k - T_{P2})}. \quad (3.30)$$

In particular, it is easy to check that given (3.20), $g^6 T_{P2} \leq g$, which implies that for $T_k \geq T_{P2}$:

$$\zeta_k \triangleq g^6 k e^{-g(T_k - T_{P2})} \leq g^6 T_k e^{-g(T_k - T_{P2})} \leq g. \quad (3.31)$$

Phase 3: The third and final phase begins with $T_k \geq T_{P2}$. In Phase 3, the α_i values are all close to α^*, and the sources essentially perform AIMD with a decrease factor of $\alpha^*/2$. The following Lemma is the key ingredient for convergence in Phase 3.

Lemma 3.2. *For $T_k \geq T_{P2}$, and any $1 \leq i, j \leq N$:*

$$|W_i(T_{k+1}) - W_j(T_{k+1})| \leq e^{-\gamma \Delta T_k} |W_i(T_k) - W_j(T_k)| + 2W_{max}\zeta_k, \quad (3.32)$$

where

$$\gamma \triangleq \frac{-\log(1 - \alpha^*/2)}{1 + W^*\alpha^*/2} > 0.$$

Proof. Using (3.18), (3.30), and (3.31), we have:

$$\begin{aligned} |W_i(T_{k+1}) - W_j(T_{k+1})| &\leq (1 - \frac{\alpha^*}{2})|W_i(T_k) - W_j(T_k)| + \\ &\frac{|W_i(T_k)|}{2}|\alpha_i(T_k) - \alpha^*| + \frac{|W_j(T_k)|}{2}|\alpha_j(T_k) - \alpha^*|, \\ &\leq (1 - \frac{\alpha^*}{2})|W_i(T_k) - W_j(T_k)| + W_{max}\zeta_k. \end{aligned}$$

Since $|\alpha_i(T_k) - \alpha^*| \leq \zeta_k$, equation (3.17) implies:

$$\Delta T_k \leq 1 + W^*\alpha^*/2 + W^*\zeta_k/2. \quad (3.33)$$

The result follows from:

$$(1-\frac{\alpha^*}{2}) \leq e^{-\gamma(\Delta T_k - W^*\zeta_k/2)} \leq e^{-\gamma\Delta T_k}(1+\zeta_k),$$

where the last inequality is true because:

$$e^{\gamma W^*\zeta_k/2} \leq e^{-\zeta_k \log(1-\alpha^*/2)/\alpha^*} \leq e^{\zeta_k \log 2} = 2^{\zeta_k} \leq 1+\zeta_k. \quad (3.34)$$

This holds for $\zeta_k \leq 1$, which we have from (3.31). □

We can now prove Theorem 3.2. Let $r \triangleq \inf\{k | T_k \geq T_{P2}\}$ and $\beta \triangleq \min(g, \gamma)$. We iterate (3.32) backwards starting from $n \geq r$ to get:

$$|W_i(T_n) - W_j(T_n)| \leq W_{max}\Big(e^{\beta(T_r - T_{P2})} + 2g^6 \sum_{k=r}^{n-1} k e^{\beta \Delta T_k}\Big) e^{-\beta(T_n - T_{P2})}.$$

But:

$$e^{\beta(T_r - T_{P2})} \leq e^{\gamma(1+W^*/2)} \leq e^{-\log(1-\alpha^*/2)/\alpha^*} \leq e^{\log 2} = 2,$$

and using (3.33), (3.34), and (3.31):

$$\begin{aligned}\sum_{k=r}^{n-1} k e^{\beta \Delta Tk} &\leq \sum_{k=r}^{n-1} k e^{\gamma(1+W^*\alpha^*/2)} e^{\gamma W^* \zeta_k/2} \\ &\leq (1-\alpha^*/2)(1+g)\sum_{k=r}^{n-1} k \\ &< n^2.\end{aligned}$$

Noting that $|W_i(T_n) - W^*| \leq \frac{1}{N}\sum_{j=1}^{N} |W_i(T_n) - W_j(T_n)|$, we have established (3.22) for $n \geq r$. The result is trivial for $n < r$, completing the proof. □

3.4 RTT-fairness

It is well-known that TCP has a bias against flows with long round-trip times; i.e., flows with longer RTTs get a smaller share of the bottleneck bandwidth when competing with

flows with shorter RTTs [106, 32, 14, 15, 170]. This is due to the fact that the rate at which flows increase their window size is inversely proportional to their RTT (typically, the window size increases by one packet per RTT). Therefore, flows with short RTTs grab bandwidth much more quickly than flows with long RTTs and settle at a higher rate. In fact, it has been shown that the throughput achieved by a TCP flow is inversely proportional to RTT^{θ} with $1 \leq \theta \leq 2$ [106].

Another important factor which affects RTT-fairness is synchronization between flows. Higher synchronization, in terms of detecting a loss or mark event, leads to worse RTT-fairness, and it has been argued that active queue management (AQM) schemes like RED [51] which avoid synchronization by probabilistically dropping (or marking) packets improve RTT fairness compared to Drop-tail queues [14, 170].

Since DCTCP employs the same additive increase mechanism used by TCP, it is expected that DCTCP also exhibits a bias against flows with longer RTTs. Moreover, the active queue management of DCTCP is (by design) similar to Drop-tail (albeit with marking instead of dropping) and prone to causing synchronization. This makes it important to study of how well DCTCP handles RTT diversity.

3.4.1 Simulations

We consider the following ns2 simulation to investigate RTT-fairness. Four flows share a single 10Gbps bottleneck link in a "dumbbell" topology. The first two of these flows have a fixed RTT of 100μs. The RTT of the other two flows is varied from 100μs to 1ms. We measure the throughput for all flows in each test, and compute the ratio of the throughput of the first two flows to that of the second two flows, as a function of the RTT ratio.

The DCTCP parameters chosen are $K = 35$ packets and $g = 1/16$. For reference, we compare the results with TCP using DCTCP-style "on–off" ECN marking, and also TCP with RED ECN marking. For RED, the marking probability increases linearly from 0 to 10$\%$ as the *average* queue length (EWMA with weight 0.1) increases from 30 to 100 packets. These parameters are chosen to ensure stability of RED in this configuration and yield roughly the same queue lengths as DCTCP. The results are shown in Figure 3.12. The algorithm labeled "DCTCP–Improved Fairness" is described in the next section.

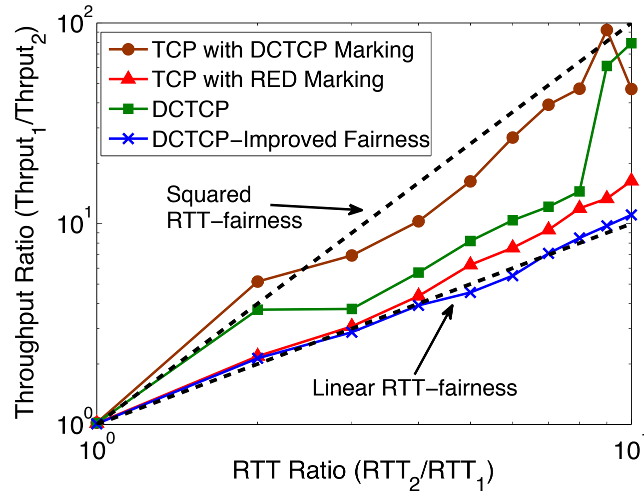


Figure 3.12: RTT-fairness in ns2 simulation. Two groups each with two flows are activated. Flows in group 1 have $RTT_1 = 100\mu$s, while the RTT for flows in group 2 (RTT_2) is varied from 100μs to 1ms. Note the log-log scale. The plot confirms that with the simple change to the DCTCP window update rule in Section 3.4.2, it achieves linear RTT-fairness.

As expected, DCTCP does exhibit a bias against flows with longer RTTs. The simulations indicate that DCTCP's RTT-fairness is better than TCP with DCTCP-style marking, which has approximately squared RTT-fairness ($Throughput \propto RTT^{-2}$), but worse than TCP–RED, which has approximately linear RTT-fairness ($Throughput \propto RTT^{-1}$). In this and other simulations not reported here, we observed that when the RTT ratio is moderate, DCTCP exhibits slightly worse than linear RTT-fairness, but tends to squared RTT-fairness as the RTT ratio becomes large. Apparently, the smooth window size adjustments made by DCTCP help alleviate some of the synchronization effects caused by the on–off marking strategy, thereby improving DCTCP's RTT-fairness.

3.4.2 Improving DCTCP's RTT-fairness

The DCTCP fluid model (Section 3.1) suggests a very simple change which considerably improves the RTT-fairness of DCTCP. We will first state this change, contrasting it with the standard DCTCP algorithm described in Section 2.2.1. We will then explain how the fluid model suggests this change.

Recall that the DCTCP algorithm reduces the window size according to

$$W \leftarrow W(1 - \alpha/2)$$

in response to a marked ACK, and that this is done at most once for each window of data. Instead, we propose subtracting $\alpha/2$ from the window size for each marked ACK, resulting in the following simple window update equation:

For each received ACK:

$$W \leftarrow W + \begin{cases} 1/W & \text{if ECN = 0} \\ 1/W - \alpha/2 & \text{if ECN = 1} \end{cases} \tag{3.35}$$

Note that a full window of marked ACKs will cause a reduction of about $W\alpha/2$, which is the same amount DCTCP would reduce the window size upon receiving a mark. A nice side-effect of this update rule is that since (3.35) applies to every ACK, it does not require maintaining extra state to prevent window reductions from happening more than once per window of data.

The simulation results in Figure 3.12 confirm that this simple change significantly improves the RTT-fairness of DCTCP (especially at high RTT ratios), and with this change, DCTCP achieves linear RTT-fairness. In fact, it even achieves a slightly better RTT-fairness than TCP–RED.

Connection with the DCTCP fluid model: Consider N flows with round-trip times RTT_i $(1 \leq i \leq N)$ in steady-state. Recall the source side equations:

$$\frac{dW_i}{dt} = \frac{1}{RTT_i} - \frac{W_i(t)\alpha_i(t)}{2RTT_i} p(t - RTT_i), \tag{3.36}$$

$$\frac{d\alpha_i}{dt} = \frac{g}{RTT_i} \left(p(t - RTT_i) - \alpha_i(t)\right), \tag{3.37}$$

where, for simplicity, we neglect the contribution of the (time-varying) queueing delay to the RTT. Note that each flow sees a time-delayed version of the common marking process $p(\cdot)$. Therefore, the $\alpha_i(\cdot)$ processes — each given by passing $p(\cdot)$ through a low pass filter — all oscillate around the same value, namely, the duty cycle of $p(\cdot)$. This leads to the

following key observation based on (3.36): the *average* window size for flow i does not depend on RTT_i; i.e., the fluid model suggests that all flows should on average have the same window size, thereby achieving linear RTT-fairness (because the ith flows's throughput is given by W_i/RTT_i).

However, as seen in Figure 3.12, simulation results indicate that the actual RTT-fairness for DCTCP is worse than linear. The key source of discrepancy between the fluid model and simulations lies in the *continuous* dynamics present in the fluid model.[5] In particular, let us consider the manner in which the window size decreases in (3.36). While $p(t - RTT_i) = 1$, the window steadily decreases with rate $W_i(t)\alpha_i(t)/(2RTT_i)$. In contrast, the packet-level algorithm reduces its window *instantaneously* upon receiving a mark. This suggests the change previously discussed to the DCTCP algorithm in equation (3.35), which bridges the gap between the packet-level and fluid dynamics.

Intuitively, this change improves the RTT-fairness by allowing flows with long RTTs to reduce their window sizes (on average) by a smaller factor compared to flows with short RTTs. This is because in a typical congestion event, flows with short RTTs receive marks earlier and reduce their window sizes, thereby relieving congestion. In such cases, less than a full window of packets from a flow with a long RTT will be marked, and therefore, the net decrease to its window size based on (3.35) will be smaller than the standard DCTCP window reduction ($W\alpha/2$).

3.5 Final Remarks

In this chapter, we mathematically analyzed the DCTCP algorithm. Our analysis showed that DCTCP can achieve very high throughput while maintaining low buffer occupancies. Specifically, we found that with a marking threshold, K, of about 17% of the bandwidth-delay product, DCTCP achieves 100% throughput, and that even for values of K as small as 1% of the bandwidth-delay product, its throughput is at least 94%. While DCTCP converges slower than TCP, we found that its convergence rate is no more than a factor 1.4 slower than TCP. We also evaluated the RTT-fairness of DCTCP, and found a simple change

[5] We emphasize that the mentioned discrepancy affects the accuracy of the fluid model only for heterogenous RTTs. As shown in Section 3.1, the fluid model is very accurate when sources have identical RTTs.

to the algorithm, which considerably improves its RTT-fairness.

In the future, it is worth mathematically analyzing the behavior of DCTCP in general networks. Our work has focused on the single bottleneck case. In particular, it would be interesting to cast the DCTCP algorithm in an optimization framework [98, 112, 104] and derive a duality model [111, 114, 163] for the algorithm. This would provide a deeper understanding of DCTCP's bandwidth allocation properties in general networks.

Chapter 4

Ultra-Low Latency Packet Transport

For decades, the primary focus of the data networking community has been on improving overall network goodput. The initial shift from circuit switching to packet switching was driven by the bandwidth inefficiencies of reserving network resources for bursty communication traffic. TCP [84] was born of the need to avoid bandwidth/congestion collapse in the network and, subsequently, to ensure bandwidth fairness [41, 131, 148] among the flows sharing a network. Discussion to add quality-of-service capability to the Internet resulted in proposals such as RSVP [172], IntServ [30] and DiffServ [125], which again focussed on bandwidth provisioning.

This focus on bandwidth efficiency has been well justified as most Internet applications typically fall into two categories. Throughput-oriented applications, such as file transfer or email, are not sensitive to the delivery times of individual packets. Even the overall completion times of individual operations can vary by multiple integer factors in the interests of increasing overall network throughput. On the other hand, latency-sensitive applications — such as web browsing and remote login — are sensitive to per-packet delivery times. However, these applications have a human in the loop and completion time variations on the order of hundreds of milliseconds or even seconds have been thought to be acceptable, especially in the interests of maintaining high average bandwidth utilization. Hence, we are left with a landscape where the network is not optimized for latency or the predictable delivery of individual packets. Consequently, the subset of applications that do require low latency communication often run on dedicated and specialized networks and protocol

stacks [80, 21, 71].

We are motivated by two recent trends that make it feasible and desirable to make low latency communication a primary metric for evaluating next-generation networks. First, a substantial amount of computing, storage, and communication is shifting to data centers. Within the confines of a single building — characterized by low propagation delays, relatively homogeneous equipment, and a single administrative entity able to modify software protocols and even influence hardware features — delivering predictable low latency appears more tractable than solving the problem in the Internet at large.

Second, and more importantly, is the rise of applications that require *ultra-low latency* communication in the data center. Besides traditional low latency applications such as high-frequency trading [5, 102] and high-performance computing [71], as discussed in Chapter 2, mainstream Internet services such as search, social networking, financial analysis, and e-commerce, must necessarily comb through terabytes of data stored across thousands of servers, all on a per-request basis. These applications are characterized by a request–response loop involving machines, not humans, and operations involving 100s (or even 1000s) of parallel requests/RPCs. Since an operation completes when all of its requests are satisfied, the *tail latency* of the individual requests are required to be in microseconds, rather than in milliseconds, to maintain quality of service and throughput targets.

The needs of these ultra-low latency applications have also given rise to new storage platforms such as RAMCloud [127, 128] that store all application data in the main memory of the servers. RAMClouds require very low latency data center networks to truly deliver on their promise of unprecedented data access performance [128]. As these platforms are integrated into mainstream applications, they must also share the network with throughput-oriented workflows consistently moving terabytes of data across a unified network fabric.

In this chapter, we build on the ideas of DCTCP in the previous chapters, and propose HULL (for High-bandwidth Ultra-Low Latency), an architecture for simultaneously delivering predictable ultra-low latency and high bandwidth utilization in a shared data center fabric, starting with existing commodity protocols and equipment.

There are several points on the path from source to destination at which packets currently experience delay: end-host stacks, network interface cards (NICs), and switches.

Techniques like kernel bypass and zero copy [145, 36] are significantly reducing the latency at the end-host and in the NICs; for example, 10Gbps NICs are currently available that achieve less than 1.5μs per-packet latency at the end-host [140]. With HULL, we focus on the latency in the network switches.

The key challenge is that high bandwidth typically requires significant in-network buffering, while predictable, ultra-low latency requires essentially no in-network buffering. Considering that modern data center fabrics can forward full-sized packets in microseconds (1.2μs for 1500 bytes at 10Gbps) and that switching latency at 10Gbps is currently 300–500ns [53, 123, 65], a one-way delivery time of 10μs (over 5 hops) is achievable across a large-scale data center, *if queueing delays can be reduced to zero*. However, given that at least 2MB of on-chip buffering is available in commodity switches [53, 123, 65] and that TCP operating on tail-drop queues attempts to fully utilize available buffers to maximize bandwidth, one-way latencies of up to a few milliseconds are quite possible — and occur in production data centers as our measurement study (Section 2.1.3) in Chapter 2 showed. This is a factor of 1,000 increase from the baseline. Since the performance of parallel, latency-sensitive applications are bound by tail latency, these applications must be provisioned for millisecond delays when, in fact, microsecond delays are achievable.

The key insight behind HULL is that it is possible to nearly eliminate network buffering by marking congestion based not on queue occupancy (or saturation), but rather based on the utilization of a link approaching its capacity. In essence, we cap the amount of bandwidth available on a link in exchange for a significant reduction in latency. The motivation is to trade the resource that is relatively plentiful in modern data centers, i.e., bandwidth, for the resource that is both expensive to deploy and results in substantial latency increase — buffer space.

Data center switches usually employ on-chip (SRAM) buffering to keep latency and pin counts low. However, in this mode, even a modest amount of buffering takes over 30% of the die area and is responsible for 30% of the power dissipation. While larger in size, off-chip buffers are both more latency intensive and incur a significantly higher pin count, increasing cost.[1] These considerations indicate that higher bandwidth switches with more

[1]The references [19, 82, 83] describe the cost of packet buffers in high bandwidth switching/routing platforms in more detail.

ports could be deployed earlier if fewer chip transistors were committed to buffers.

In summary, this chapter makes the following major contributions:

- We design the HULL architecture. HULL's design centers around *Phantom Queues*, a switch mechanism closely related to existing virtual queue-based active queue management schemes [60, 105]. Phantom queues simulate the occupancy of a queue sitting on a link that drains at *less than* the actual link's rate. Standard ECN [138] marking based on the occupancy of these phantom queues is then used to signal end hosts employing DCTCP congestion control to reduce transmission rate.

- Through our evaluation, we found that a key requirement to make this approach feasible is to employ hardware packet pacing (a feature increasingly available in NICs) to smooth the transmission rate that results from widespread network features such as Large Send Offloading (LSO) and interrupt coalescing. We introduce innovative methods for estimating the drain rate of the pacer and for adaptively detecting the flows which require pacing. Without pacing, phantom queues would be fooled into regularly marking congestion based on spurious signals causing degradation in throughput, just as spikes in queuing caused by such bursting would hurt latency.

- We extensively evaluate HULL in a hardware testbed and using simulations. We find that HULL can reduce both average and 99th percentile packet latency by more than a factor of 10 compared to DCTCP and a factor of 40 compared to TCP. For example, in one configuration, the average latency drops from 78μs for DCTCP (329μs for TCP) to 7μs and the 99th percentile drops from 556μs for DCTCP (3961μs for TCP) to 48μs, with a configurable reduction in bandwidth for throughput-oriented applications. A factor of 10 reduction in latency has the potential to substantially increase the amount of work applications such as web search perform for end-user requests — e.g., process 10 times more data with predictable completion times — though we leave such exploration of end-application benefits for future work.

4.1 Design Overview

The goal of the HULL architecture is to *simultaneously deliver near baseline fabric latency and high throughput*. In this section, we discuss the challenges involved in achieving this goal. These challenges pertain to correctly *detecting, signaling, and reacting* to impending congestion. We overview how these challenges guide our design decisions in HULL and motivate its three main components: phantom queues, DCTCP congestion control, and packet pacing.

4.1.1 Phantom Queues: Detecting and Signaling Congestion

The traditional congestion signal in TCP is the drop of packets. TCP increases its congestion window (and transmission rate) until available buffers overflow and packets are dropped. As previously discussed, given the low inherent propagation and switching times in the data center, this tail-drop behavior incurs an unacceptably large queuing latency.

Active queue management (AQM) schemes [51, 76, 23] proactively signal congestion before buffers overflow and try to regulate the queue around some target occupancy. While these methods can be quite effective in reducing queuing latency, they cannot eliminate it altogether. This is because they must observe a non-zero queue occupancy to begin signaling congestion, and sources react to these congestion signals after at least one RTT of lag, during which time the queue builds up even further. Note that the DCTCP queue management scheme has the same issue, since the switch marks packets after the queue occupancy is above the threshold K.

This leads to the following observation: Achieving predictable and low fabric latency essentially requires congestion signaling *before* any queueing occurs. That is, achieving the lowest level of queueing latency imposes a fundamental tradeoff of bandwidth — creating a "bandwidth headroom". Our experiments (Section 4.5) show that bandwidth headroom dramatically reduces average and tail queuing latencies. In particular, the reductions at the high percentiles are significant compared to queue-based AQM schemes.

We propose the Phantom Queue (PQ) as a mechanism for creating bandwidth headroom. A phantom queue is a simulated queue, associated with each switch egress port, that

sets ECN [138] marks based on link utilization rather than queue occupancy. The PQ simulates queue buildup for a virtual egress link of a configurable speed, slower than the actual physical link (e.g., running at $\gamma = 95\%$ of the line rate). *The PQ is not really a queue since it does not store packets.* It is simply a counter that is updated while packets exit the link at line rate to determine the queuing that would have been present on the slower virtual link. It then marks ECN for packets that pass through it when the counter (simulated queue) is above a fixed threshold.

The PQ explicitly attempts to set the aggregate transmission rates for congestion-controlled flows to be strictly less than the physical link capacity, thereby keeping switch buffers largely unoccupied. This bandwidth headroom allows latency sensitive flows to fly through the network at baseline transmission plus propagation rates.

Remark 4.1. The idea of using a simulated queue for signaling congestion has been used in Virtual Queue (VQ) [60, 105] AQM schemes. An important distinction is that while a VQ is typically placed in parallel to a physical queue in the switch, we propose placing the PQ in *series* with the switch egress port. This change has an important consequence: the PQ can operate independently of the internal architecture of the switch (e.g., output-queued, shared memory, or combined input-output queued) and its buffer management policies, and, therefore, work with *any* switch. In fact, we implement the PQ external to physical switches as a hardware "bump on the wire" prototyped on the NetFPGA [122] platform (Section 4.4.1). In general, of course, VQs and PQs can and have been [124] integrated into switching hardware.

4.1.2 DCTCP: Adaptive Reaction to ECN

Standard TCP reacts to ECN marks by cutting the congestion window in half. As discussed extensively in Chapters 2 and 3, without adequate buffering to keep the bottleneck link busy, this conservative back off can result in a severe loss of throughput. For instance, with zero buffering, TCP's rate fluctuates between 50% and 100% of link capacity, achieving an average throughput of only 75% [160] (see Figure 3.9 for an illustration). Hence, since the PQ aggressively marks packets to keep the buffer occupancy at zero, TCP's back off can be especially detrimental.

To mitigate this problem, HULL sources use DCTCP to adaptively react to ECN marks in proportion to the *extent* of network congestion. DCTCP is a suitable choice because, as was shown in Chapter 3, in theory, DCTCP can maintain more than 94% throughput even with zero queueing.

4.1.3 Packet Pacing

Bursty transmission occurs for multiple reasons, ranging from TCP artifacts such as ACK compression and slow start [88, 173], to various offload features in NICs such as interrupt coalescing and Large Send Offloading (LSO) designed to reduce CPU utilization [26]. With LSO for instance, hosts transfer large buffers of data to the NIC, leaving specialized hardware to segment the buffer into individual packets, which then burst out at line rate. As our experiments show (Section 4.3.1), interrupt coalescing and LSO are required to maintain acceptable CPU overhead at 10Gbps speeds.

Traffic bursts cause temporary increases in queue occupancies. This may not be a big problem if enough buffer space is available to avoid packet drops and if variable latency is not a concern. However, since the PQ aggressively attempts to keep buffers empty (to prevent variable latency), such bursts trigger spurious congestion signals, leading to reduced throughput (Section 4.3.3).

A natural technique for combating the negative effects of bursty transmission sources on network queueing is packet pacing. While earlier work [94, 161, 43, 153] introduce pacing at various points in the protocol stack, we find that, to be effective, pacing must take place in hardware after the last source of bursty transmission: NIC-based LSO. Ideally, a simple additional mechanism at the NIC itself would pace data transmission. The pacer would likely be implemented as a simple leaky bucket with a configurable exit rate. We describe a hardware design and implementation for pacing in Section 4.3.2.

It is paradoxical that the pacer must queue packets at the edge (end-hosts) so that queueing inside the network is reduced. Such edge queueing can actually increase end-to-end latency, offsetting any benefits of reduced in-network queueing. We resolve this paradox by noting that only packets that *belong to long flows* and hence are not sensitive to per-packet delivery times should be paced. Small latency-sensitive flows should not be paced,

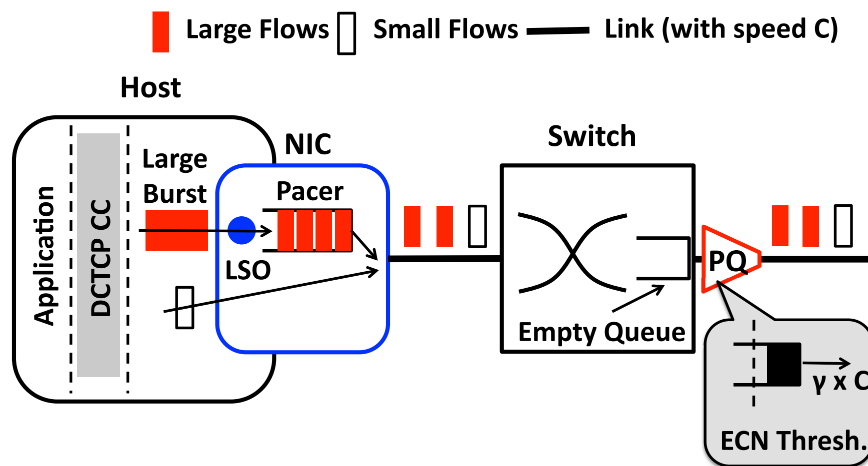


Figure 4.1: The HULL architecture consists of phantom queues at switch egress ports and DCTCP congestion control and packet pacers at end-hosts.

allowing them to exploit the lowest available fabric latency. We employ a simple adaptive end-host mechanism to determine whether a flow should be subject to pacing. This is inspired by classic work on the UNIX multi-level feedback queue [149] that attempts to classify interactive versus bulk jobs in the operating system. Newly created flows are classified as latency sensitive and initially not subjected to pacing. However, once a flow sees a sufficient number of ECN marks, it is classified as throughput-oriented and paced.

4.1.4 The HULL Architecture

The complete High-bandwidth Ultra-Low Latency (HULL) architecture is shown in Figure 4.1. Large flows at the host stack, which runs DCTCP congestion control, send large bursts to the NIC for segmentation via LSO. The Pacer captures the packets of the large flows after segmentation, and spaces them out at the correct transmission rate. The PQ uses ECN marking based on a simulated queue to create bandwidth headroom, limiting the link utilization to some factor, $\gamma < 1$, of the line rate. This ensures switch queues run (nearly) empty, which enables low latency for small flows.

4.1.5 QoS as the Benchmark

Ethernet and IP provide multiple quality of service (QoS) priorities, with packets in different priorities isolated from one another through partitioned switch buffers. One method for meeting our objective of ultra-low latency and high bandwidth is to use two priorities: an absolute priority for the flows which require very low latency and a lower priority for the bandwidth-intensive elastic flows. While this method has the potential to provide ideal performance, it may be not be very practical (and is not commonly deployed) because applications do not segregate latency-sensitive short flows and bandwidth-intensive large flows dynamically. Indeed, application developers do not consider priority classes for network transfers. It is more common to assign an entire application to a priority, and use priorities to segregate *applications*. Secondly, as we shall see in Section 4.5.2, priority-based approaches have some drawbacks. Priorities must be respected *everywhere* to be effective; in the network, the NICs and the end host software stack. Failure to do this mixes packets of different priorities into common queues, negating performance gains.

Thus, even though priorities are not commonly deployed end-to-end, due to their good performance, we consider them for benchmarking purposes. We find that HULL can nearly match the latency performance achievable with strictly prioritized traffic.

4.2 Bandwidth Headroom

This section explores the consequences of creating bandwidth headroom. We illustrate the role of the congestion control protocol in determining the amount of bandwidth headroom by comparing TCP and DCTCP. We then discuss how bandwidth headroom impacts the completion time of large flows.

4.2.1 Importance of Stable Rate Control

All congestion control algorithms cause fluctuations in rate as they probe for bandwidth and react to delayed congestion signals from the network. Typically, some amount of buffering is required to absorb rate variations and avoid throughput loss. Essentially, buffering keeps the bottleneck link busy while sources that have cut their sending rates recover. This is

	Throughput	Mean Latency	99th Percentile Latency
TCP	982Mbps	1100.6μs	4308.8μs
DCTCP-30K	975Mbps	153.9μs	305.8μs

Table 4.1: Baseline throughput and latency for two long-lived flows with TCP and DCTCP-30K (30KB marking threshold).

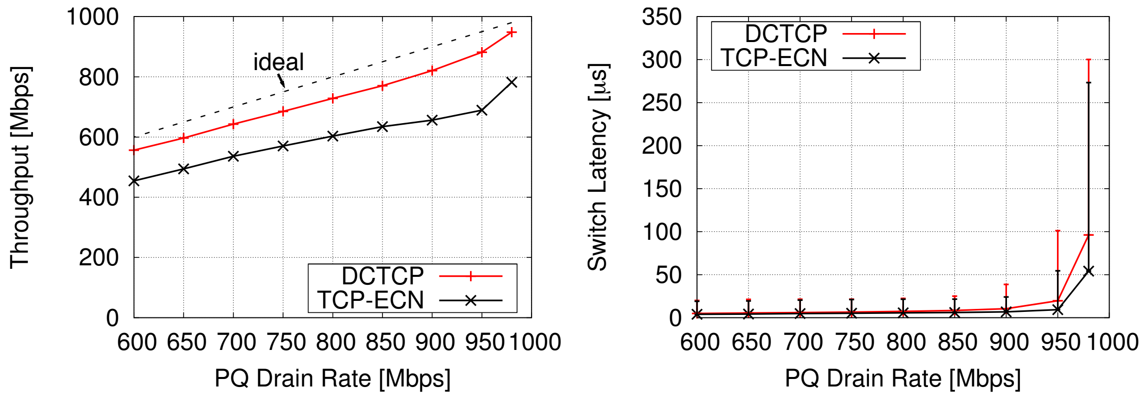


Figure 4.2: Throughput (left) and average switch latency (right) for TCP-ECN and DCTCP with a PQ, as drain rate varies. The vertical bars in the right plot indicate the 99th percentile.

especially problematic in low statistical multiplexing environments, where only a few high speed flows must sustain throughput [19].

Therefore, special care must be taken with the congestion control algorithm if we aim to reduce buffer occupancies to zero. We illustrate this using a simple experiment. We connect three servers to a single switch and initiate two long-lived flows from two of the servers to the third (details regarding our experimental setup can be found in Section 4.4). We measure the aggregate throughput and the latency due to queueing at the switch. As a baseline reference, the throughput and latency for standard TCP (with tail-drop), and DCTCP, with the default marking threshold of 30KB recommended in Chapter 2 (Section 2.2.4) are given in Table 4.1. As expected, DCTCP shows an order of magnitude improvement in latency over TCP, because it reacts to queue buildup beyond the marking threshold.

We conduct a series of experiments where we sweep the drain rate of a PQ attached to the congested port. The marking threshold at the PQ is set to 6KB and we also enable our hardware pacing module (described in Section 4.3.2). The results are shown in Figure 4.2.

Compared to the baseline, a significant latency reduction occurs for both TCP-ECN (TCP with ECN enabled) and DCTCP, when bandwidth headroom is created by the PQ. Also, for both schemes, the throughput is lower than intended by the PQ drain rate. This is because of the rate variations imposed by the congestion control dynamics. However, *TCP-ECN loses considerably more throughput than DCTCP at all PQ drain rates*. The gap between TCP-ECN's throughput and the PQ drain rate is $\sim$17–26% of the line rate, while it is $\sim$6–8% for DCTCP. These results match well with the theoretical predictions in Chapter 3.

4.2.2 Slowdown Due to Bandwidth Headroom

Bandwidth headroom created by the PQ inevitably slows down the large flows, which are bandwidth-intensive. An important question is: *How badly will the large flows be affected?*

We answer this question using a simple queuing analysis of the "slowdown", defined as the ratio of the completion times of a flow with and without the PQ. We find that, somewhat counter-intuitively, that the slowdown is not only determined by the amount of bandwidth sacrificed, but also depends on the total applied load.

Consider the well-known model of a M/G/1-Processor Sharing queue for TCP bandwidth sharing [56, 141]. Flows arrive according to a Poisson process of some rate, λ, and have sizes drawn from a general distribution, S. The flows share a link of capacity C in a fair manner; i.e., if there are n flows in the system, each gets a bandwidth of C/n. We assume the total load $\rho \triangleq \lambda \mathbb{E}(S)/C < 1$, so that the system is stable. A standard result for the M/G/1-PS queue states that in this setting, the average flow completion time for a flow of size x is given by:

$$\mathit{FCT}_{100\%} = \frac{x}{C(1-\rho)}, \tag{4.1}$$

where the '100%' indicates that this is the FCT without bandwidth headroom. Now, suppose we only allow the flows to use γC of the capacity. Noting that the load on this slower link is $\tilde{\rho} = \rho/\gamma$, and invoking (4.1) again, we find that the average completion time is:

$$\mathit{FCT}_{\gamma} = \frac{x}{\gamma C(1-\rho/\gamma)} = \frac{x}{C(\gamma-\rho)}. \tag{4.2}$$

Hence, dividing (4.2) by (4.1), the slowdown caused by the bandwidth headroom is:

$$SD \triangleq \frac{FCT_\gamma}{FCT_{100\%}} = \frac{1-\rho}{\gamma-\rho}. \tag{4.3}$$

The interesting fact is that *the slowdown gets worse as the load increases*. This is because giving bandwidth away also increases the effective load ($\hat{\rho} > \rho$). For example, using (4.3), the slowdown with 20% bandwidth headroom ($\gamma = 0.8$), at load $\rho = 0.2, 0.4, 0.6$ will be 1.33, 1.5, 2, (equivalently: 33%, 50%, 100%) respectively.

Our experiments in Section 4.5.2 confirm the validity of this model (for example, see Figure 4.10). This highlights the importance of not giving away too much bandwidth. Fortunately, as we show, *even a small amount of bandwidth headroom (e.g., 5-10%) provides a dramatic reduction in latency*.

Remark 4.2. The M/G/1-PS model provides a good approximation for large flows for which TCP has time to converge to the fair bandwidth allocation [56]. It is not, however, a good model for small flows as it does not capture latency. In fact, since the completion time for small flows is mainly determined by the latency, they are not adversely affected by bandwidth headroom (Section 4.5).

4.3 Packet Pacing

In this section, we dig deeper into packet pacing in the HULL architecture. We report experiments that show that hardware offload features used in modern networking stacks generate very bursty traffic, but are necessary to reduce CPU overhead. We then describe the design of a hardware module for packet pacing and demonstrate its effectiveness. Finally, we explore a tradeoff with the Pacer between the extra delay it induces, and how effectively it can pace.

4.3.1 Burstiness of Traffic

Modern NICs implement various offload mechanisms to reduce CPU overhead for network communication. These offloads typically result in highly bursty traffic [26]. For example,

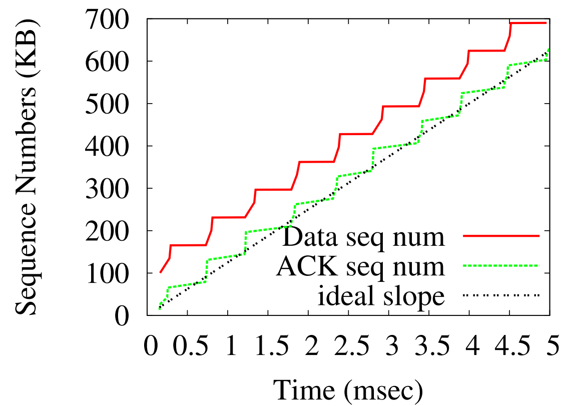


Figure 4.3: Burstiness with 10Gbps NICs. Every ∼0.5ms, a burst of packets totaling 65KB is sent out at line rate to obtain an average rate of ∼1Gbps.

interrupt coalescing is a standard feature which allows the NIC to delay interrupting the CPU and wait for large batches of packets to be processed in one SoftIrq. This disrupts the normal TCP ACK-clocking and leads to many MTUs worth of data being released by TCP in a burst. A further optimization, Large Send Offload (LSO), allows TCP to send large buffers (currently up to 64KB), delegating the segmentation into MTU-sized packets to the NIC. These packets then burst out of the NIC at line rate.

As later experiments show, this burstiness can be detrimental to our goal of ultra-low latency. However, *using hardware offloading to reduce the CPU overhead of the networking stack is unavoidable; especially, as link speeds increase to 10Gbps and beyond.*

We illustrate the burstiness of traffic using a simple experiment. We directly connect two servers with 10Gbps NICs (see Section 4.4.2 for testbed details), and enable LSO and interrupt coalescing with a MTU of 1500 bytes. We generate a single TCP flow between the two servers, and cap the window size of the flow such that its throughput is ∼1Gbps on average. Figure 4.3 shows the data and ACK sequence numbers within a 5ms window (time and sequence numbers are relative to the origin) compared to the “ideal” slope for perfectly paced transmission at 1Gbps. The sequence numbers show a step-like behavior that demonstrates the extent of burstiness. Each step, occurring roughly every 0.5ms, corresponds to a back-to-back burst of data packets totaling 65KB. Analyzing the packet trace using tcpdump [155], we find that the bursty behavior reflects the batching of ACKs at the

Interrupt Coalescing	CPU Utilization (%)	Throughput (Gbps)	Ack Ratio (KB)
adaptive	37.2	9.5	41.3
rx-frames=128	30.7	9.5	64.0
rx-frames=32	53.2	9.5	16.5
rx-frames=8	75	9.5	12.2
rx-frames=2	98.7	9.3	11.4
rx-frames=0	99	7.7	67.4

Table 4.2: The impact of interrupt coalescing. Note that 'adaptive' is the default setting for interrupt coalescing.

receiver: Every 0.5ms, 6 ACKs acknowledging 65KB in total are received within a 24–50μs interval. Whereas, ideally, for a flow at 1Gbps, the ACKs for 65KB should be evenly spread over 520μs.

The batching results from interrupt coalescing at the receiver NIC. To study this further, we repeat the experiment with no cap on the window size and different levels of interrupt coalescing by varying the value of `rx-frames`, a NIC parameter that controls the number of frames between interrupts. Table 4.2 summarizes the results. We observe a tradeoff between the CPU overhead at the receiver and the average ACK ratio — the number of data bytes acknowledged by one ACK — which is a good proxy for burstiness. Setting `rx-frames` at or below 8 heavily burdens the receiver CPU, but improves the ACK ratio. With interrupt coalescing disabled (`rx-frames` = 0), the receiver CPU is saturated and cannot keep up with the load. This further increases the ACK ratio and causes a 1.8Gbps loss in throughput.

We also found that enabling LSO is necessary to achieve the 10Gbps line rate. Without LSO, the sender's CPU is saturated and there is close to 3Gbps loss of throughput.

The takeaway is that *hardware offload mechanisms such as interrupt coalescing and LSO cause bursty packet transmissions, but are indispensable at high speeds to reduce the CPU overhead of the networking stack. Hence, packet pacing is needed to combat the burstiness of traffic.*

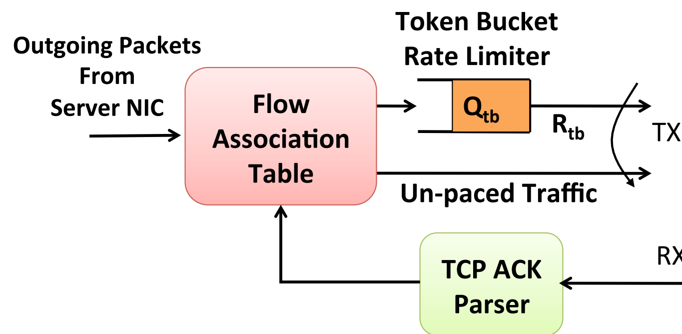


Figure 4.4: Block diagram of the Pacer module.

4.3.2 Hardware Pacer Module

We now describe a hardware module for packet pacing. The Pacer module inserts suitable spacing between the packets of flows transmitted by the server. We envision the Pacer module operating at the NIC. Pacing in hardware has a number of advantages over software pacing. A hardware pacer can easily support the sub-microsecond scheduling granularity required to pace at 10Gbps rates and beyond without stressing the host's CPU. Moreover, unlike pacing in the host stack that typically requires disabling segmentation offload, a hardware module in the NIC is oblivious to server LSO settings since it operates on the outgoing packet stream *after* segmentation takes place.

Figure 4.4 shows the block diagram of the Pacer module. The Flow Association Table is consulted to check whether an outgoing packet requires pacing (see below). If so, the packet is placed into a token bucket rate limiter with a configurable transmission rate. Otherwise, it bypasses the token bucket and is sent immediately.

The key challenges to pacing, especially in a hardware module, are: (i) determining the appropriate pacing rate, and (ii) deciding the flows that require pacing.

Dynamic pacing rate estimation: The NIC is unaware of the actual sending rate (*Cwnd*/*RTT*) of TCP sources. Therefore, we use a simple algorithm to estimate the congestion-friendly transmission rate. We assume that over a sufficiently large measurement interval (e.g., a few RTTs) each host's aggregate transmission rate will match the rate imposed by higher-level congestion control protocols such as TCP (or DCTCP). The pacer dynamically measures this rate and matches the rate of the token bucket to it. More precisely, every T_r seconds, the Pacer counts the number of bytes it receives from the host, denoted by M_r. It then

modifies the rate of the token bucket according to:

$$R_{tb} \leftarrow (1-\eta) \times R_{tb} + \eta \times \frac{M_r}{T_r} + \beta \times Q_{tb}. \tag{4.4}$$

The parameters η and β are positive constants and Q_{tb} is the current backlog of the token bucket in bytes. R_{tb} is in bytes per second.

Equation (4.4) is a first order low-pass filter on the rate samples M_r/T_r. The term $\beta \times Q_{tb}$ is necessary to prevent the Pacer backlog from becoming too large.[2] This is crucial to avoid a large buffer for the token bucket, which adds to the cost of the Pacer, and may also induce significant latency to the paced flows (see Section 4.3.4).

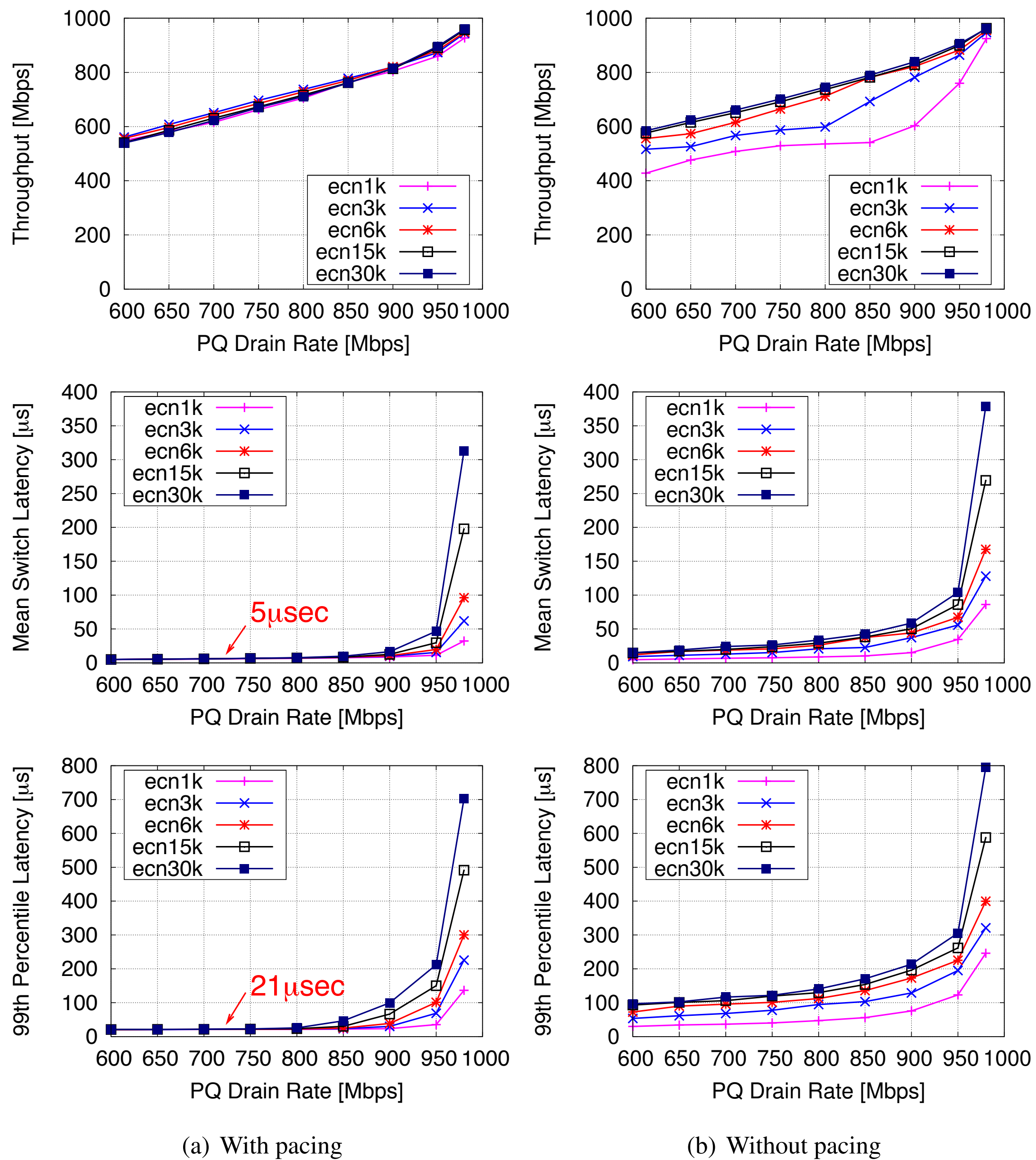
Which flows need pacing? As previously discussed, only packets belonging to long flows (that are not latency-sensitive) should be paced. Moreover, only those flows that *are causing congestion* actually warrant pacing. We employ a simple adaptive mechanism to automatically detect such flows. Newly created flows are initially not paced. For each ACK with the ECN-Echo bit set, the corresponding flow is designated for pacing with some probability, p_a (set to 1/8 in our implementation), and inserted in the Flow Association Table. This probabilistic sampling ensures that small flows are unlikely to be paced. The flow's association times out after some inactivity time, T_i, so that idle flows are eventually reclassified as latency sensitive.

Remark 4.3. We have described the pacer module with a single token bucket rate-limiter for simplicity. However, the same design can be used with multiple rate-limiters, allowing for more accuracy when pacing multiple flows. For example, a flow can be hashed to one of the rate-limiters when it is first chosen for pacing.

4.3.3 Effectiveness of the Pacer

We demonstrate the effectiveness of pacing by running an experiment with 2 long-lived DCTCP flows (similar to Section 4.2.1), with and without pacing. As before, we sweep the drain rate of the PQ. We also vary the marking threshold at the PQ from 1KB to 30KB.

[2] In fact, if the aggregate rate of paced flows is fixed at $R^* = M_r/T_r$, the only fixed point of equation (4.4) is $R_{tb} = R^*$, and $Q_{tb} = 0$.



(a) With pacing (b) Without pacing

Figure 4.5: Throughput and switch latency as PQ drain rate varies, with and without pacing.

The results are shown in Figure 4.5. We observe that pacing improves both throughput and latency. The throughput varies nearly linearly with the PQ drain rate when the Pacer is enabled. Without pacing, however, we observe reduced throughput with low marking thresholds. This is because of spurious congestion signals caused by bursty traffic. Also,

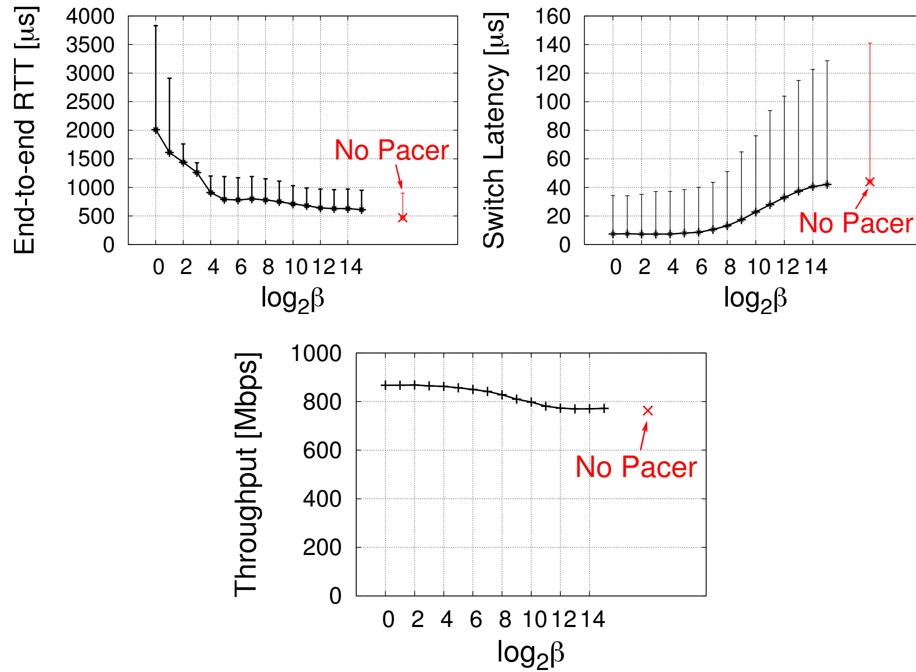


Figure 4.6: The end-to-end RTT for paced flows, average switch latency, and overall throughput, as the parameter β is varied. The vertical bars indicate the 99th percentile. β controls how aggressively the Pacer reacts to queue buildup in the rate-limiter, with larger values reducing the backlog but also causing more bursting. The experiment shows that there is a tradeoff between the latency induced by the Pacer and its effectiveness.

with pacing, the average and 99th percentile latency plummet with bandwidth headroom, quickly reaching their floor values of 5μs and 21μs respectively. In contrast, the latency decreases much more gradually without pacing, particularly at the 99th percentile.

4.3.4 Tradeoff Between Pacer Delay and Effectiveness

Pacing, *by definition*, implies delaying the transmission of packets. We find that there is a tradeoff between the delay at the Pacer and how effectively it can pace. This tradeoff is controlled by the parameter β in Equation (4.4). Higher values of β cause a more aggressive increase in the transmission rate to keep the token bucket rate-limiter backlog, Q_{tb}, small. However, this also means that the Pacer creates more bursty output when a burst of traffic hits the token bucket; basically, the Pacer does "less pacing".

The experiment in Figure 4.6 demonstrates the tradeoff. We start two long-lived DCTCP flows transmitting to a single receiver. The Pacer is enabled and we sweep β over the range $\beta = 2^0$ to $\beta = 2^{14}$ (the rest of the parameters are set as in Table 4.3). The PQ on the receiver link is configured to drain at 950Mbps and has a marking threshold of 1KB. We measure both the latency across the switch and the end-to-end RTT *for the flows being paced*, which is measured using ping from the senders to the receiver.

The Pacer induces more delay with smaller values of β (increasing the end-to-end RTT), but is also more effective in pacing. Specifically, smaller β achieve lower average switch latency and higher overall throughput. We observe a sharp increase in the Pacer's delay for values of β smaller than 2^4 without much gain in switch latency, suggesting the sweet spot for β. Nonetheless, the Pacer does add a few hundreds of microseconds of delay to the paced flows. This underscores the importance of selectively choosing the flows to pace. Only large flows which are throughput-bound and are not impacted by the increase in delay should be paced. In fact, note that the large flows also benefit from pacing since throughput increases from $\sim$770Mbps without pacing, to $\sim$870Mbps with pacing (at $\beta = 2^0$).

4.4 Experimental Setup

We now describe our implementation of HULL's components and the experimental testbed that we use for our evaluation.

4.4.1 Implementation

We use the NetFPGA [122] platform to implement the Pacer and PQ modules. NetFPGA is a PCI card with four Gigabit Ethernet ports and a Xilinx Virtex-II Pro 125MHz FPGA that has 4MB of SRAM. Each NetFPGA supports two Pacers and two PQs. The complete implementation consumes 160 block RAMs (out of 232, or 68% of the total FPGA capacity), and occupies 20,364 slices (86% of the total FPGA capacity).

The Pacer module has a single token bucket rate-limiter with a 128KB FIFO queue. The Pacer's transmission rate is controlled as described in Section 4.3.2 at a granularity of 0.5Mbps. Tokens of variable size (1-2000 bytes, proportional to the Pacer rate) are added

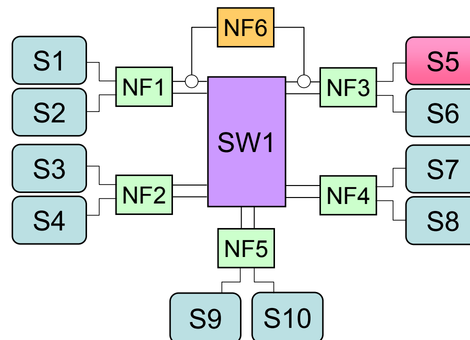


Figure 4.7: Experimental HULL testbed. 10 servers are connected to a Broadcom Triumph switch through intermediate NetFPGA devices that implement Pacers and PQs. An additional NetFPGA (NF6) implements a Latency Measurement Module for measuring the switch latency.

every 16μs and the maximum allowed outstanding tokens (the bucket depth) is 3KB. The Flow Association Table can hold 64 entries for identifying flows that require pacing. The Pacer enqueues the packets of these flows in the rate-limiter and forwards the rest in pass-through mode.

The PQ module implements the virtual queue counter. It is incremented upon receiving a packet and decremented according to the configured PQ drain rate every 800ns. If the counter exceeds the configured marking threshold, the ECN Congestion Experienced (CE) bit in the IP header of the incoming packet is set and the checksum value is recalculated. The PQ is completely pass-through and does not queue packets.

We have also introduced modifications to the TCP stack in Linux 2.6.26 for DCTCP, following the algorithm in Chapter 2. Our code is available online at [38].

4.4.2 Testbed

Our testbed consists of 10 servers and 6 NetFPGAs connected to a Broadcom Triumph2 switch as shown in Figure 4.7. Triumph2 is an ECN-capable switch with 48 1Gbps ports and 4MB of buffer memory shared across all ports.

Each server has 4-core Intel Xeon E5620 2.4GHz CPUs with Hyper-Threading and at least 16GB of RAM. The servers use Intel's 82574L 1GbE Ethernet Controller. Two of the servers, S9 and S10, also have Mellanox ConnectX-2 ENt 10Gbase-T NICs, which were

Phantom Queue	Drain Rate = 950Mbps, Marking Thresh. = 1KB
Pacer	$T_r = 64\mu$s, $\eta = 0.125$, $\beta = 16$, $p_a = 0.125$, $T_i = 10ms$

Table 4.3: Baseline parameter settings for testbed experiments.

used for the 10Gbps experiments in Section 4.3.1.

Each of the NetFPGAs NF1-NF5 implements two Pacers and two PQs: One for each of the two servers and the two switch ports connected to it. All server-to-switch traffic goes through the Pacer module and all switch-to-server traffic goes through the PQ module.

For the majority of our experiments, we use machine S5 as the receiver, and (a subset of) the rest of the machines as senders which cause congestion at the switch port connected to S5 (via NF3).

Measuring switch latency: We have also developed a Latency Measurement Module (LMM) in NetFPGA for sub-microsecond resolution measurement of the latency across the congested switch port. The LMM (NF6 in Figure 4.7) works as follows: Server S1 sends a 1500 byte ping packet to server S5 every 1ms. The ping packets are intercepted and timestamped by the LMM before entering the switch. As a ping packet leaves the switch, it is again intercepted and the previous time-stamp is subtracted from the current time to calculate the latency. Note that the pings add 12Mbps (1.2%) of throughput overhead.

Remark 4.4. The NetFPGA is not optimized for latency and uses store and forward. Because of this, the baseline latency reported by the LMM for a 1500 byte packet is 29.46μs. We subtract this constant from all our switch latency measurements and report the *latency increase* due to queueing.

4.4.3 Parameter Choices

Table 4.3 gives the baseline parameters used in the testbed experiments. The parameters are determined experimentally.

The PQ parameters are chosen based on experiments with long-lived flows, similar to the one shown in Figure 4.5. As can be seen in this figure, the PQ with 950Mbps drain rate

and 1KB marking threshold (with pacing) achieves almost the latency floor. An interesting fact is that smaller marking thresholds are required to maintain low latency as the PQ drain rate (γC) increases. This can be seen most visibly in Figure 4.5(a) for the 99th percentile latency. The reason is that since the input rate into the PQ is limited to the line rate (because it is in series), it takes longer for it to build up as the drain rate increases. Therefore, the marking threshold must also be reduced with increasing drain rate to ensure that the PQ reacts to congestion quickly.

Regarding the Pacer parameters, we find that the speed of the Pacer rate adaptation — determined by T_r/η in equation (4.4) — needs to be on the order of a few RTTs. This ensures that the aggregate host transmission rate is tracked closely by the Pacer and provides a good estimate of the rate imposed by the higher-layer DCTCP congestion control. The parameter β is chosen as described in Section 4.3.4. The parameters p_a and T_i are chosen so that small flows (e.g., smaller than 10KB) are unlikely to be paced. Overall, we do not find the Pacer to be very sensitive to these parameters.

4.5 Results

This section presents our experimental and simulation results for evaluating HULL. We use micro-benchmarks to compare the latency and throughput performance of HULL with various schemes including TCP with drop-tail, default DCTCP, DCTCP with reduced marking threshold, and TCP with an ideal two-priority QoS scheme (TCP-QoS), where small (latency-sensitive) flows are given strict priority over large flows. We also investigate the scalability of HULL using large-scale ns-2 [126] simulations. We briefly summarize our main findings:

- In micro benchmarks with both static and dynamic traffic, we find that HULL significantly reduces average and tail latencies compared to TCP and DCTCP. For example, with dynamic traffic (Section 4.5.2) HULL provides a more than 40x reduction in average latency compared to TCP (more than 10x compared to DCTCP), with bigger reductions at the high percentiles. Compared to an optimized DCTCP with low marking threshold and pacing, HULL achieves a 46–58% lower average latency,

and a 69–78% lower 99th percentile latency. The bandwidth traded for this latency reduction increases the completion-time of large flows by 17–55%, depending on the load, in good agreement with the theoretical prediction in Section 4.2.2.

- HULL achieves comparable latency to TCP-QoS with two priorities, but a lower throughput since QoS does not leave bandwidth headroom. Further, unlike TCP-QoS which loses a lot of throughput due to packet drops if buffers are shallow (more than 58% in one experiment), HULL is much less sensitive to the size of switch buffers, as it (mostly) keeps them unoccupied.

- Our large-scale ns-2 simulations confirm that HULL scales well and achieves significant latency reductions in large multi-tier topologies.

4.5.1 Static Flow Experiments

We begin with an experiment that evaluates throughput and latency in the presence of long-lived flows. We call this the *static* setting since the number of flows is constant during the experiment and the flows always have data to send. Each flow is from a different server sending to the receiver S5 (see Figure 4.7). We sweep the number of flows from 2 to 8. (Note that at least 2 servers must send concurrently to cause congestion at the switch.)

Schemes: We compare four schemes: (i) standard TCP (with drop-tail), (ii) DCTCP with 30KB marking threshold, (iii) DCTCP with 6KB marking threshold and pacing enabled, and (iv) DCTCP with a PQ (950Mbps with 1KB marking threshold). For schemes (ii) and (iii), ECN marking is enabled at the switch and is based on the physical queue occupancy, while for (iv), marking is only done by the PQ.
Note. We use the recommended marking threshold of 30KB for DCTCP (Chapter 2). We also lower the marking threshold to 6KB to evaluate how much latency can be improved with pure queue-based congestion signaling. Experiments show that with this low marking threshold, pacing is required to avoid excessive loss in throughput (Section 4.5.2). Reducing the marking threshold below 6KB severely degrades throughput, even with pacing.

Analysis: The results are shown in Figure 4.8. We observe more than an order of magnitude (about 20x) reduction in average latency with DCTCP compared to TCP. Reducing

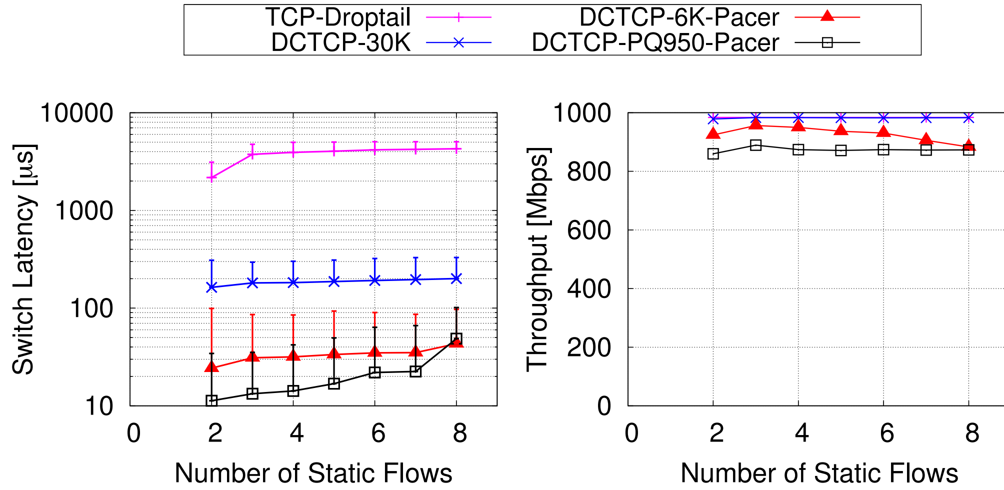


Figure 4.8: Switch latency (left) and throughput (right) as the number of long-lived flows varies. The latency plot uses a logarithmic scale and shows the average latency, with the vertical bars showing the 99th percentile.

the marking threshold to 6KB gives a further 6x reduction, bringing the average latency down from ∼200μs to ∼30μs. When there are a few static flows (e.g., less than 4), the PQ reduces the average latency by another factor 2–3 compared to DCTCP with 6KB marking threshold. Moreover, it also significantly lowers the jitter, achieving a 99th percentile of ∼30μs compared to ∼100μs for DCTCP-6K-Pacer, and more than 300μs for standard DCTCP. The PQ's lower latency is because of the bandwidth headroom it creates: The throughput for the PQ is about 870Mbps. The 8% loss compared to the PQ's 950Mbps drain rate is due to the rate fluctuations of DCTCP, as explained in Section 4.2.1.

Behavior with increasing flows: Figure 4.8 shows that bandwidth headroom becomes gradually less effective with increasing the number long-lived flows. This is because of the way TCP (and DCTCP) sources increase their window size to probe for additional bandwidth. As is well-known, during Congestion Avoidance, a TCP source increases its window size by one packet every round-trip time. This is equivalent to an increase in sending rate of $1/RTT$ (in pkts/sec) each round-trip-time. Now, with N flows all increasing their rates at this slope, more bandwidth headroom is required to prevent the aggregate rate from exceeding the link capacity and causing queuing. More precisely, because of the one RTT of delay in receiving ECN marks from the PQ, the sources' aggregate rate overshoots

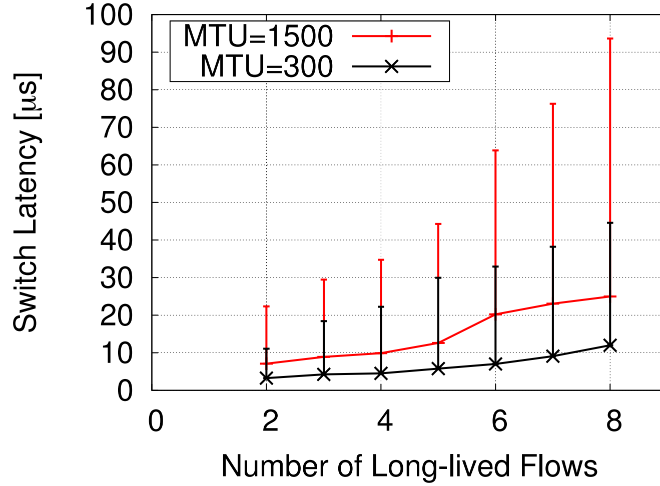


Figure 4.9: The impact of increasing the number of flows on average and 99th percentile switch latency with PQ, for MTU = 1500 bytes and MTU = 300 bytes. The smaller MTU is used to emulate a link running at 5Gbps. The PQ drain rate is set to 800Mbps and the marking threshold is 6KB.

the PQ's target drain rate by N/RTT (in pkts/sec). Hence, we require:

$$(1-\gamma)C > \frac{N}{RTT} \Longrightarrow 1-\gamma > \frac{N}{C \times RTT}, \tag{4.5}$$

where $C \times RTT$ is the bandwidth-delay product in units of packets. Equation (4.5) indicates that *the bandwidth headroom required (as a percentage of capacity) to prevent queuing increases with more flows and decreases with larger bandwidth-delay product.*

An important consequence of equation (4.5) is that for a fixed RTT, less bandwidth headroom is needed as the link capacity increases (e.g., from 1Gbps to 10Gbps). We demonstrate this using a simple experiment in Figure 4.9. In the absence of an experimental setup faster than 1Gbps, we emulate what happens at higher link speeds by decreasing the MTU (packet size) to 300 bytes. Since the default MTU is 1500 bytes, this increases the bandwidth-delay product by a factor of 5 in units of packets, effectively emulating a 5Gbps link. As can be seen in the figure, the latency with the PQ is much lower with the smaller packet size at all number of flows. This confirms that the *sensitivity to the number of flows decreases with increasing link speed. Essentially, the same amount of bandwidth headroom (as a percentage of the link speed) is more effective for faster links.*

Remark 4.5. We found that when the MTU is reduced to 300 bytes, the receiver NIC cannot keep up with the higher packets/sec and starts dropping packets. To avoid this artifact, we had to reduce the PQ drain rate to 800Mbps for the tests in Figure 4.9.

4.5.2 Dynamic Flow Experiments

In this section, we present the results of a micro-benchmark which creates a *dynamic* workload. We develop a simple client/server application to generate traffic based on patterns seen in storage systems like memcached [168]. The client application, running on server S5 (Figure 4.7) opens 16 permanent TCP connections with each of the other 9 servers. During the test, the client repeatedly chooses a random connection among the pool of connections and makes a request for a file on that connection. The server application responds with the requested file. The requests are generated as a Poisson process in an open loop fashion [143]; that is, new requests are triggered independently of prior outstanding requests. The request rate is chosen such that the average RX throughput at the client is at a desired level of load. For example, if the average file size is 100KB, and the desired load is 40% (400Mbps), the client makes 500 requests per second on average. We conduct experiments at low (20%), medium (40%), and high (60%) levels of load. During the experiments, we measure the switch latency (using the NetFPGA Latency Measurement Module), as well as the application level flow completion times (FCT).

Baseline

For the baseline experiment, we use a workload where 80% of all client requests are for a 1KB file and 20% are for a 10MB file. Of course, this is not meant to be a realistic workload. Rather, it allows a clear comparison of how different schemes impact the small (latency-sensitive) and large (bandwidth-sensitive) flows.

Note. The 1KB flows are just a single packet and can complete in one RTT. Such single packet flows are very common in data center networks; for example, measurements in a production data center of a large cloud service provider [64] have revealed that more than 50% of flows are smaller than 1KB.

		Switch Latency (μs)			1KB FCT (μs)			10MB FCT (ms)		
		Avg	**90th**	**99th**	**Avg**	**90th**	**99th**	**Avg**	**90th**	**99th**
20% Load	**TCP-DropTail**	111.5	450.3	1224.8	1047	1136	10533	110.2	162.3	349.6
	DCTCP-30K	38.4	176.4	295.2	475	638	2838	**106.8**	**155.2**	**301.7**
	DCTCP-6K-Pacer	6.6	13.0	59.7	389	531	888	111.8	168.5	320.0
	DCTCP-PQ950-Pacer	**2.8**	**7.6**	**18.6**	**380**	**529**	**756**	125.4	188.3	359.9
40% Load	**TCP-DropTail**	329.3	892.7	3960.8	1537	3387	5475	**151.3**	**275.3**	575.0
	DCTCP-30K	78.3	225.0	556.0	495	720	1794	155.1	281.5	**503.3**
	DCTCP-6K-Pacer	15.1	35.5	213.4	403	560	933	168.7	309.3	567.5
	DCTCP-PQ950-Pacer	**7.0**	**13.5**	**48.2**	**382**	**536**	**808**	198.8	370.5	654.7
60% Load	**TCP-DropTail**	720.5	2796.1	4656.1	2103	4423	5425	**250.0**	**514.6**	1007.4
	DCTCP-30K	119.1	247.2	604.9	511	740	1268	267.6	538.4	**907.3**
	DCTCP-6K-Pacer	24.8	52.9	311.7	403	563	923	320.9	632.6	1245.6
	DCTCP-PQ950-Pacer	**13.5**	**29.3**	**99.2**	**386**	**530**	**782**	389.4	801.3	1309.9

Table 4.4: Baseline dynamic flow experiment. The average, 90th percentile, and 99th percentile switch latency and flow completion times are shown. The results are the average of 10 trials. In each case, the best scheme is shown in red.

Table 4.4 gives the results for the same four schemes that were used in the static flow experiments (Section 4.5.1).

Analysis: Switch latency. The switch latency is very high with TCP compared to the other three schemes since it completely fills available buffers. With DCTCP, the latency is 3–6 times lower on average. Reducing the marking threshold to 6KB gives another factor of 5 reduction in average latency. However, some baseline level of queueing delay and significant *jitter* remains, with hundreds of microseconds of latency at the 99th percentile. As explained in Section 4.1.1, this is because queue-based congestion signaling (even with pacing) is too late to react; by the time congestion is detected, a queue has already built up and is increasing. The lowest latency is achieved by the PQ which creates bandwidth headroom. Compared to DCTCP-6K-Pacer, the PQ reduces the average latency by 46–58% and the 99th percentile by 69–78%. The latency with the PQ is especially impressive considering just a single 1500 byte packet adds 12μs of queueing latency at 1Gbps.

Analysis: 1KB FCT & end-host latency. The 1KB FCT is rather high for all schemes. It is evident from the switch latency measurements that the high 1KB FCTs are due to the delays incurred by packets at the end-hosts (in the software stack, PCI-E bus, and network adapters). The host-side latency is particularly large when the servers are actively transmitting/receiving data at high load. As a reference, the minimum 1KB FCT we have observed is about 160μs when the servers are idle. Interestingly, the latency at the end-host (and the

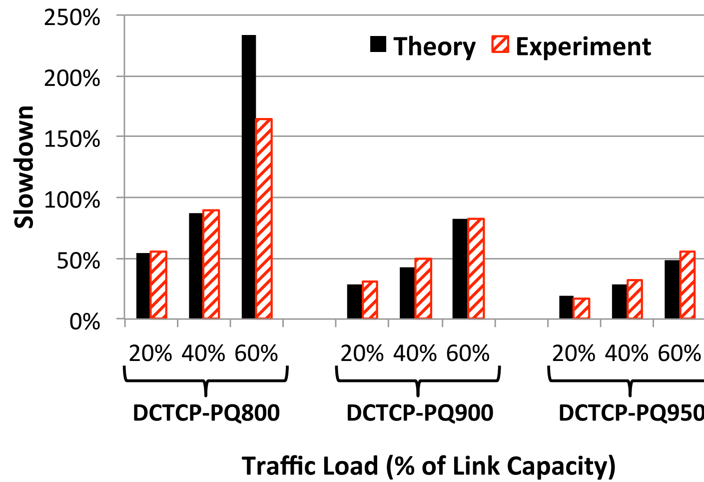


Figure 4.10: Slowdown for 10MB flows: Theory vs Experiment.

1KB FCT) improves with more aggressive signaling of congestion in the network, especially, compared to TCP-DropTail. This suggests that the un-checked increase in window size with TCP-DropTail (and to a lesser extent with DCTCP-30K) causes queuing at both the switches *and* the end-hosts. Essentially, flows with large windows deposit large chunks of data into NIC buffers, which adds delay for the small flows. This also explains why for TCP (and DCTCP-30K), the 99th percentile of the 1KB FCT actually decreases with more load: at higher load, it is less likely for a flow's window size to become very large and cause severe interference in the NIC.

The main takeaway is that *bandwidth headroom significantly reduces the average and tail switch latency under load, even compared to optimized queue-based AQM with a low marking threshold and pacing. However, to take full advantage of this reduction, the latency of the software stack and network adapters must also improve.*

Analysis: Slowdown for 10MB flows. The bandwidth given away by the PQ increases the flow completion of the 10MB flows, which are throughput-limited. As predicted by the theoretical model in Section 4.2.2, the slowdown is worse at higher load. Compared to the lowest achieved value (shown in red in Table 4.4), with the PQ, the average 10MB FCT is 17% longer at 20% load, 31% longer at 40% load, and 55% longer at 60% load.

Figure 4.10 compares the slowdown predicted by theory with that observed in experiments. The comparison includes the results for PQ with 950Mbps drain rate, given in

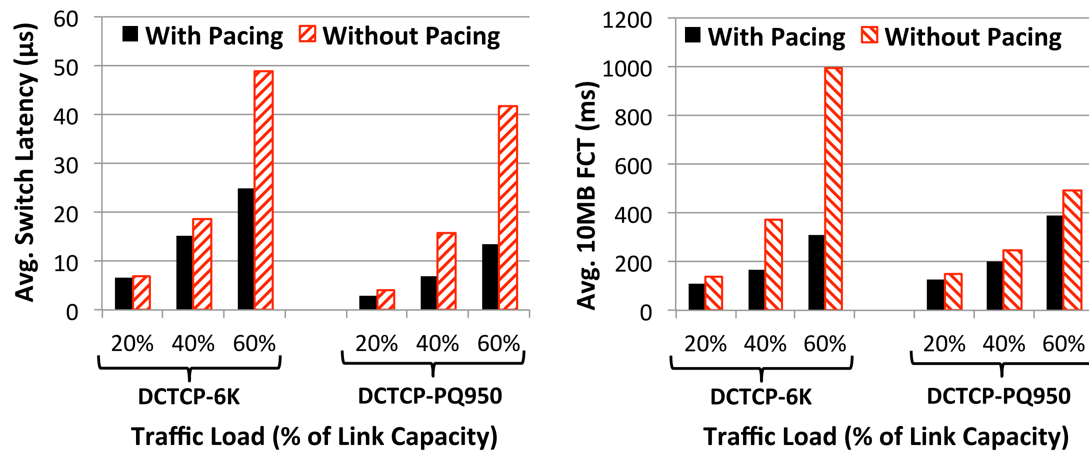


Figure 4.11: Average switch latency (left) and 10MB FCT (right), with and without pacing.

Table 4.4, as well as PQ with 900Mbps and 800Mbps for which the detailed numbers are omitted. The theoretical slowdown is computed using equation (4.3). To invoke the equation, we use $\rho = 0.2, 0.4$, and 0.6 corresponding to the load. We also account for the additional throughput loss due to DCTCP rate fluctuations (Section 4.2.1), by subtracting 8% from the PQ drain rate to get γ. That is, we use $\gamma = 0.72, 0.82$, and 0.87 corresponding to the drain rates 800, 900, and 950Mbps.

Overall, the theory and experiments match very well, except when the traffic load and the PQ drain rate are very close, as in the case of 60% load with DCTCP-PQ800 in Figure 4.10. This is because in this case, the denominator in Equation (4.3), $\gamma - \rho$, is very small and minor changes in γ make a big difference. Therefore, artifacts like the exact amount of throughput lost because of DCTCP's rate fluctuations become important. For example, just changing gamma from 0.72 to 0.75 changes the predicted slowdown at 60% load from 233% to 166%, nearly the same value observed in experiments.

Impact of pacing: Figure 4.11 compares the average switch latency and 10MB FCT, with and without pacing. The comparison is shown for DCTCP with the marking threshold at the switch set to 6KB, as well as for DCTCP with the PQ draining at 950Mbps. In all cases, pacing lowers both the switch latency and the FCT of large flows, improving the latency *and* the throughput.

		Switch Latency (μs)			1KB FCT (μs)			10MB FCT (ms)		
		Avg	**90th**	**99th**	**Avg**	**90th**	**99th**	**Avg**	**90th**	**99th**
Large Buffer	**TCP-QoS**	**6.4**	17.2	**20.2**	565	570	2308	**152.6**	**275.4**	**585.1**
	DCTCP-PQ950-Pacer	6.9	**13.5**	47.8	**381**	**538**	**810**	199.0	370.2	658.9
Small Buffer	**TCP-QoS**	5.3	15.5	**19.7**	**371**	**519**	**729**	362.3	811.9	1924.3
	DCTCP-PQ950-Pacer	**5.0**	**13.2**	35.1	378	521	759	**199.4**	**367.0**	**654.9**

Table 4.5: HULL vs QoS with two priorities. The switch buffer is configured to use either dynamic buffer allocation (Large Buffer) or a fixed buffer of 10pkts = 15KB (Small Buffer). In all tests, the total load is 40%. In the case of QoS, the switch latency reported is that of the high priority queue. The results are the average of 10 trials.

Remark 4.6. Most data center networks operate at loads less than 30% [28], so a load of 60% with Poisson/bursty traffic is highly unlikely — the performance degradation would be too severe. The results at 20% and 40% load are more indicative of actual performance.

Comparison with two-priority QoS scheme

We now compare HULL with TCP using an ideal two-priority QoS scheme. We repeat the baseline experiment from the previous section, but for QoS, we modify our application to classify the 1KB flows as high-priority using the Type of Service (TOS) field in the IP header. The switch uses a separate queue for these flows which is given strict priority over the other, best-effort, traffic. As described in Section 4.1.5, we use QoS in this way for benchmarking purposes, even though, in practice, QoS is not commonly used to segregate latency-sensitive shot flows and bandwidth-intensive large flows dynamically.

Scenarios: We consider two settings for the switch buffer size: (i) Dynamic buffer allocation (the default switch settings), and (ii) a fixed buffer of 10 packets (15KB) per port per priority. Note that in the latter setting TCP-QoS gets 30KB in total per port, whereas HULL gets just 15KB since it only uses one priority. The second setting is used to evaluate how switches with very shallow buffers impact performance, since dynamic buffer allocation allows a congested port to grab up to $\sim$700KB of the total 4MB of buffer in the switch.

Analysis: Table 4.5 gives the results. We make three observations:

- HULL and QoS achieve roughly the same average switch latency. HULL is slightly better at the 90th percentile, but worse at the 99th percentile.

- When the switch buffer is large, TCP-QoS achieves a better FCT for the 10MB flows than HULL as it does not sacrifice any throughput. However, with small buffers, there is about a 2.4x increase in the FCT with TCP-QoS (equivalent to a 58% reduction in throughput). HULL achieves basically the same throughput in both cases because it does not need the buffers in the first place.

- In the large buffer setting, the 1KB flows complete significantly faster with HULL — more than 33% faster on average and 65% faster at the 99th percentile. This is because the best-effort flows (which have large window sizes) interfere with high-priority flows at the end-hosts, similar to what was observed for TCP-DropTail in the baseline experiment (Table 4.4). This shows that all points of contention (including the end-hosts, PCI-E bus, and NICs) must respect priorities for QoS to be effective.

Overall, this experiment shows that *HULL achieves nearly as low a latency as the ideal QoS scheme, but gets lower throughput. Also, unlike QoS, HULL can cope with switches with very shallow buffers because it avoids queue buildup altogether.*

4.5.3 Large-scale ns-2 Simulation

Due to the small size of our testbed, we cannot verify in hardware that HULL scales to the multi-tier topologies common in data centers. Therefore, we complement our hardware evaluation with large-scale ns-2 simulations targeting a multi-tier topology and workload.

Topology: We simulate a three-tier fat-tree topology shown in Figure 4.12 based on recently proposed [4, 64] scalable data center architectures . The network consists of 56 8-port switches that connect 192 servers organized in 8 pods. There is a 3:1 over-subscription at the top-of-the-rack (TOR) level. The switches have 250 packets worth of buffering. All links are 10Gbps and have 200ns of delay, with 1μs of additional delay at the end-hosts. This, along with the fact that ns-2 simulates *store-and-forward* switches, implies that the end-to-end round-trip latency for a 1500 byte packet and a 40 byte ACK across the entire network (6 hops) is 11.8μs.

Routing: We use standard Equal-Cost Multi-Path (ECMP) [78] routing which hashes each flow to one of the shortest paths between the source and destination nodes. All packets of

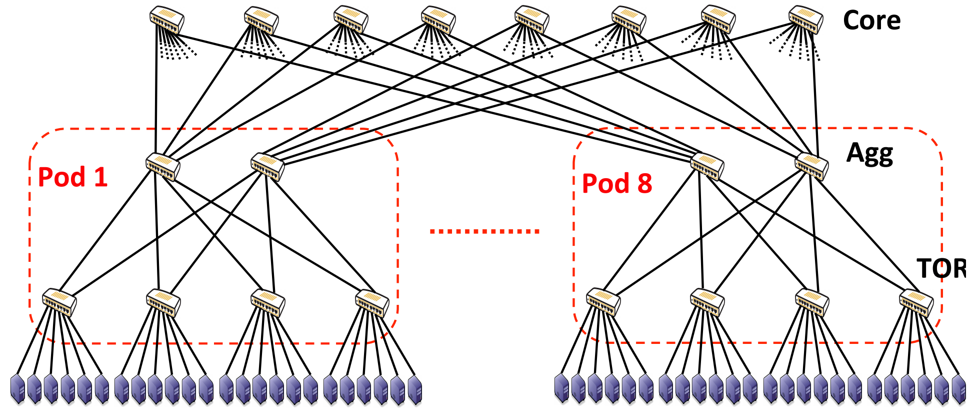


Figure 4.12: Topology for ns-2 simulations: 10Gbps three-tier fat-tree with 192 servers organized in 8 pods. The topology has a 3:1 over-subscription at the TOR.

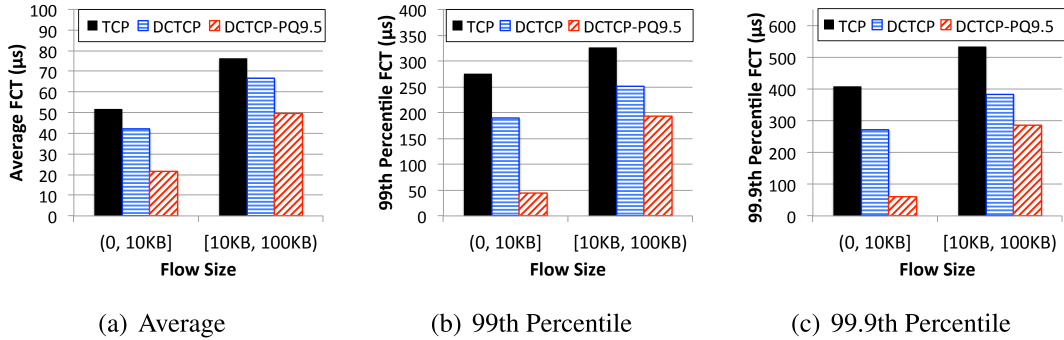
Flow Size	Percentage of Flows	Percentage of Bytes
(0, 10KB]	53.04%	6.51%
(10KB, 100KB]	42.78%	19.25%
(100KB, 10MB]	4.14%	32.46%
(10MB, ∞)	0.04%	41.78%

Table 4.6: Breakdown of flows and bytes according to flow size.

the flow take the same path. This avoids the case where packet reordering is misinterpreted as a sign of congestion by TCP (or DCTCP).

Workload: The workload is generated similarly to our dynamic flow experiments in hardware. We open permanent connections between each pair of servers. Flows arrive according to a Poisson process and are sent from a random source to a random destination server. The flow sizes are drawn from a Pareto distribution with shape parameter 1.05 and mean 100KB. This distribution creates a heavy-tailed workload where the majority of flows are small, but the majority of traffic is from large flows, as is commonly observed in real networks. The breakdown is given in Table 4.6: 95% of the flows are less than 100KB and contribute a little over 25% of all data bytes; while 0.03% of the flows that are larger than 10MB contribute over 40% of all bytes.

Simulation settings: We compare standard TCP, DCTCP, and DCTCP with a PQ draining at 9.5Gbps (HULL). The Pacer is also enabled for HULL, with parameters: $T_r = 7.2\mu$s,



(a) Average (b) 99th Percentile (c) 99.9th Percentile

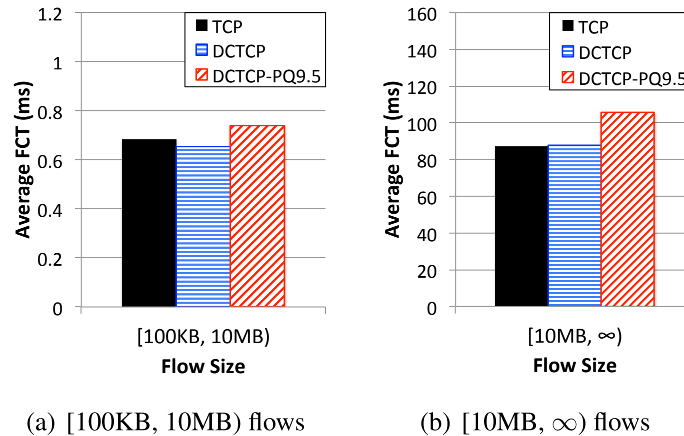
Figure 4.13: Average and high percentile FCT for small flows. The statistics for the (0,10KB] and [10KB, 100KB) are shown separately.

$\eta = 0.125$, $\beta = 375$, $p_a = 0.125$, and $T_i = 1$ms. The changes to the parameters compared to the ones used in our hardware testbed (Table 4.3) are because of the difference in link speed (10Gbps vs 1Gbps) and the much lower RTT in the simulations. We set the flow arrival rate so the load at the server-to-TOR links is 15% (We have also run many simulations with other levels of load, with qualitatively similar results). All simulations last for at least 5 million flows.

Note. Because the topology has 3:1 over-subscription at the TOR, the load is approximately 3 times higher at the TOR-to-Agg and Agg-to-Core links. A precise calculation shows that the load is 43.8% at the TOR-to-Agg links, and 39.6% at the Agg-to-Core links.

Analysis: Small flows. Figure 4.13 shows the average, 99th percentile, and 99.9th percentile of the FCT for small flows. The statistics are shown separately for flows smaller than 10KB, and flows with size between 10KB and 100KB. We observe a significant improvement in the FCT of these flows with HULL; especially for flows smaller than 10KB, there is more than 50% reduction in the average FCT compared to DCTCP, and more than 80% reduction at the 99th and 99.9th percentiles.

It is important to note that the 20μs average FCT for flows smaller than 10KB achieved by HULL is near ideal given that the simulation uses store-and-forward switching. In fact, the average size for flows smaller than 10KB is 6.8KB. Because of store-and-forward, the FCT for a flow of this size is at least $\sim$16.2μs. This implies that with HULL, across the 5 switches end-to-end between source and destination, there are, *in total*, only 3 packets



(a) [100KB, 10MB) flows (b) [10MB, ∞) flows

Figure 4.14: Average FCT for large flows. The statistics for the [100KB,10MB) and [10MB, ∞) are shown separately.

being queued on average (each adding 1.2μs of delay).

Analysis: Large flows. Figure 4.14 shows the average FCT for flows between 100KB and 10MB in size, and for those that are larger than 10MB. As expected, these flows are slower with HULL: up to 24% slower for flows larger than 10MB, which is approximately the slowdown predicted by theory (Section 4.2.2) at this fairly high level of load.

Overall, the ns-2 simulations confirm that *bandwidth headroom created by HULL is effective in large multi-tier networks and can significantly reduce the latency and jitter for small flows.*

4.6 Related Work

We briefly summarize the literature closely related to our work.

Active queue management (AQM): AQM has been an active area of research ever since RED [51] was proposed. Subsequent work [109, 48, 76, 23] refined the concept and introduced enhancements for stability and fairness. While these schemes reduce queueing delay relative to tail-drop queues, they still induce too large a queueing latency from the ultra-low latency perspective because they are, fundamentally, queue-based congestion signaling mechanisms.

Virtual-queue mechanisms [60, 105] consider signaling congestion based on link utilization. The Phantom Queue is inspired by this work with the difference that PQs are not adjacent to physical switch queues. Instead, they operate on network links (in series with switch ports). This makes the PQ completely agnostic to the internal switch architecture and allows it to be deployed with any switch as a “bump-on-the-wire”.

Transport layer: A large body of research relevant to our ultra-low latency objective includes congestion control algorithms that introduce various changes to TCP or are TCP substitutes. We provided a brief review in Section 2.4 in the context of DCTCP. Similar to DCTCP, the majority of these algorithms, by design, require the queue to build up to a certain level and, therefore, do not provide ultra-low latency. However, any congestion control algorithm can be complimented with PQs in the network to deliver ultra-low latency, as long as it exhibits good rate stability when operating with near-zero buffer occupancies so that it doesn’t lose too much throughput (Section 4.2.1). We have used DCTCP for congestion control in HULL because of its good stability, and because it can be deployed with existing hardware.

Pacing: TCP pacing was suggested for alleviating burstiness due to ACK compression [173]. Support for pacing has not been unanimous. For instance, Aggarwal *et al.* [18] show that paced flows are negatively impacted when competing with non-paced flows because their packets are deliberately delayed. With increasing line rates and the adoption of TCP segmentation offloading, the impact of burstiness has been getting worse [26] and the need for pacing is becoming more evident.

One way pacing has been implemented is using software timers to determine when a packet should be transmitted [94, 161, 43]. However, when the ticks of the pacer are very frequent, as happens at high line rates, such software-based schemes often lack access to accurate timers or heavily burden the CPU with significant interrupt processing. Further, a software pacer prior to the NIC cannot offset the effects of offloading functions like LSO which occur in the NIC.

PSPacer [153] is a tool that paces data packets by inserting 802.3x PAUSE frames between them. 802.3x PAUSE frames are then dropped at the ingress of the first switch or router encountered. Besides requiring that regular 802.3x is disabled in the network,

this approach burdens the CPU by continuously sending traffic at line rate regardless of the actual data rate.

4.7 Final Remarks

In emerging data center environments, a single shared network fabric is multiplexed across hundreds of individual applications. Many of these applications are throughput-oriented and well suited to commodity TCP/IP/Ethernet stacks. However, an increasing array of critical services require predictable low latency communication among thousands of servers on the critical path of individual operations.

In this chapter, we presented a framework for delivering baseline fabric latency for latency-sensitive applications while simultaneously supporting high fabric throughput for bandwidth oriented applications. Through a combination of Phantom Queues to run the network with near zero queueing, adaptive response to ECN marks using DCTCP, and packet pacing to smooth bursts induced by hardware offload mechanisms like LSO, we showed a factor of up to 10–40 reduction in average and tail latency with a configurable sacrifice of overall throughput. Our work makes another case for a time when aggregate bandwidth may no longer be the ultimate evaluation criterion for large-scale fabrics, but a tool in support of other high-level goals such as predictable low latency.

Chapter 5

Stability of Quantized Congestion Notification

In the past few years, the Data Center Bridging (DCB) Task Group in the IEEE 802.1 Ethernet standards body has specified several enhancements to existing Ethernet specifications to satisfy the requirements of protocols and applications in the data center. These enhancements include end-to-end congestion management in Layer 2 (IEEE 802.1Qau [136]), priority-based link-level flow control (IEEE 802.1Qbb [135]), and a common management framework for assigning bandwidth to traffic classes (IEEE 802.1Qaz [134]). Collectively, the enhancements constitute *Converged Enhanced Ethernet (CEE)* which enables consolidation of disparate link layer technologies used in existing data centers (such as Fibre Channel [27] for storage, Infiniband [21] for high performance computing, and Ethernet for LAN connectivity) in a unified fabric. We refer the reader to the DCB Task Group's website [37] for more details on the group's activities.

This chapter is concerned with the Quantized Congestion Notification (QCN) algorithm that was standardized by the DCB Task Group in March 2010 as the IEEE 802.1Qau Congestion Notification [136] standard. QCN is a Layer 2 congestion control mechanism in which a congested switch can control the rates of Layer 2 sources (Ethernet Network Interface Cards) whose packets are passing through the switch. The algorithm essentially specifies a congestion control loop at Layer 2 similar to the TCP/RED (or DCTCP) control loops at Layer 3. However, as we discuss in Section 5.1, the operating conditions in

switched Ethernet are vastly different than those in the Internet and this has necessitated the development of a new congestion control scheme.

One of the main challenges in designing QCN has been to ensure that the control loop has good stability. As we have seen in our study of DCTCP, the stability of rate and buffer occupancies is critical for the performance of congestion control algorithms, especially in data center networks because of the shallow buffered switches (e.g., 10s–100s of KBytes per port) and the small number of large (high-bandwidth) flows that are typically active on each path. Furthermore, stability is more challenging for a Layer 2 algorithm like QCN than its Layer 3 counterparts. Indeed, as we explain in Section 5.1, since there are no per-packet ACKs in Ethernet, QCN cannot benefit from "self-clocking" [84] (using ACKs to pace transmissions). Self-clocking essentially automatically tunes a congestion control algorithm to the bandwidth and delay of the operating environment and is very useful for stabilizing TCP (and TCP-like algorithms).

Our goal in this chapter is to provide an analysis of the QCN control loop's stability properties. Our results complement the extensive investigations of the algorithm that were conducted via simulation and experimentation during the standardization process [79] and provide a theoretical underpinning for understanding QCN's performance. Also, since the QCN algorithm shares commonalities with the BIC-TCP [170] and CUBIC-TCP [70] algorithms, our analysis is useful for understanding these algorithms as well (CUBIC-TCP is used by default in Linux kernels 2.6.19 and above).

We analyze QCN from two distinct viewpoints:

- First, we conduct a classical analysis of the control loop using a delay-differential fluid model of the QCN algorithm that we develop. We analyze this model using standard techniques from linear control theory and obtain the stability margins of the control loop as a function of its design parameters and network conditions such as link speeds, number of flows, and round-trip time.

- Next, we turn to the question of *why* QCN exhibits good stability. We demonstrate using analysis and simulations that QCN's good stability, especially in face of increasing feedback delay, stems from the use of a particular averaging behavior in its rate dynamics. This finding suggests a novel method, that we call the *Averaging Principle*

(AP), for stabilizing *any* control loop as feedback delays increase. We demonstrate the generality of the AP by applying it to various other feedback systems. We also analyze the AP and show that for linear control systems, it is *algebraically equivalent* to a Proportional-Derivative (PD) controller; that is, the AP controller and the PD controller are input-output equivalent. Since the PD controller is well-known to stabilize control loops when lags increase [54], this equivalence precisely characterizes the stability benefits of the AP. This result is also useful in practice because the PD controller requires the switch to compute an additional derivative of the congestion state, which can be difficult to achieve since switches implement QCN functionality in hardware. The AP provides an equivalent benefit without switch modifications.

The rest of this chapter is organized as follows. We briefly discuss the motivations for designing a new congestion control mechanism at Layer 2 in Section 5.1 and review the QCN algorithm in Section 5.2. We then present the delay-differential equation fluid model of QCN in Section 5.3. In Section 5.4, we analyze a linearized approximation of the fluid model to find the stability margins of the algorithm. We also compare the stability of the linearized model with and without the mentioned averaging behavior (AP) and show that it the AP increases the stability margin. We formally describe the AP in Section 5.5 for a generic control system and prove that for linear control systems, it is algebraically equivalent to a PD controller. Finally, we explore the impact of QCN's stability on switch buffer sizing in Section 5.6, and show that by reducing the variance in the sending rates, QCN requires significantly less buffers than TCP for good performance.

5.1 Layer 2 Congestion Control

The QCN algorithm has been developed to provide congestion control at the Ethernet layer, or at Layer 2 (L2). A related effort is the Priority Flow Control standard (IEEE 802.1Qbb [135]), for enabling hop-by-hop, per-priority pausing of traffic at congested links. Thus, when the buffer at a congested link fills up, it issues a PAUSE message to upstream buffers to ensure packets are not dropped due to congestion. A consequence of link-level pausing is the phenomenon of "congestion spreading"; the domino effect of congestion propagating upstream causing secondary bottlenecks. Secondary bottlenecks

are highly undesirable as they affect sources whose packets do not pass through the primary bottleneck. An L2 congestion control scheme allows a primary bottleneck to directly reduce the rates of those sources whose packets pass through it, thereby preventing (or reducing the instances of) secondary bottlenecks. The L2 congestion control algorithm is expected to operate well regardless of whether link-level pause exists or not (i.e. packets may be dropped), and regardless of the higher level transport protocol (e.g., TCP or UDP) used by the traffic sources.

Layer 2 vs. Layer 3. There are many differences between the operating environments of switched Ethernet (Layer 2) and TCP/IP (Layer 3), which we list below.

1. *No per-packet ACKs in Ethernet.* This has several consequences for congestion control mechanisms: (i) packet transmission is not self-clocked as with TCP, (ii) path delays (round trip times) are not knowable, and (iii) congestion must be signaled by switches directly to sources. The last point makes it difficult to know *path* congestion; one only knows about *node* congestion.

2. *Links may be paused.* As mentioned, Ethernet links may be paused to prevent packet drops. A significant side-effect of this is that congestion spreading can occur, causing spurious secondary bottlenecks.

3. *No packet sequence numbers.* L2 packets do not have sequence numbers from which RTTs, or the "length" of the control loop in terms of number of packets in flight may be inferred.

4. *Sources start at the line rate.* Unlike the slow-start mechanism in TCP, L2 sources may start transmission at the full line rate of 10Gbps. This is because L2 sources are implemented in hardware, and installing rate limiters is the only way to have a source send at less than the line rate. But since rate limiters are typically few in number, it is preferable to install them only when a source is known to be causing congestion at a switch.

5. *Very shallow buffers.* As previously discussed, Ethernet switch buffers are typically 100s of KBytes per-port, as opposed to Internet router buffers which are 100s of

MBytes per-port. Even though in terms of bandwidth-delay product the difference is about right (Ethernet RTTs are a few 100 microseconds, as opposed Internet RTTs which are a few 100 milliseconds), the transfer of a single file of, say, 1MB length can overwhelm an Ethernet buffer. This is especially true when L2 sources come on at the line rate.

6. *Small number-of-sources regime is typical.* In the Internet literature on congestion control, one usually studies the system when the number of sources is large (which is typical in the Internet). However, in data center Ethernet, there are typically a small number of sources active on each path. For instance, in our measurement study in Chapter 2, we found less than 5 concurrent flows larger than 1MB at each server (Figure 2.4).

7. *Multipathing.* Traditionally, forwarding in Ethernet is done on spanning trees. While this avoids loops, it is both fragile (there is only one path on a tree between any pair of nodes) and leads to an underutilization of network capacity. For these reasons, equal cost multipathing (ECMP) is typically used in data center Ethernet to spread traffic on more than one path between L2 sources and destinations. However, congestion levels on the different paths may be vastly different. Hence, the L2 congestion control algorithm must ensure that congestion experienced by the traffic of an L2 source on one path does not negatively impact the traffic on other paths.

These differences and challenges place some unique restrictions on the type of L2 congestion control that is suitable for the data center environment and have necessitated the development of QCN. Explaining how QCN addresses all these challenges is not in the scope of the current thesis and we refer the interested reader to the IEEE 802.1Qau standard [136] and the numerous presentations on the work group's website [79] for more details. Our focus is the QCN control loop's *stability*. Specifically, since data center Ethernet switches typically have small buffers, it is imperative that buffer occupancies do not fluctuate wildly, causing overflows and dropped packets or link under-utilization. This is particularly important when trying to control a small number of high rate sources with a shallow buffer, whose depth is a fraction of the bandwidth-delay product. For example, QCN is typically configured to operate switch buffers at 30KB occupancy when a single

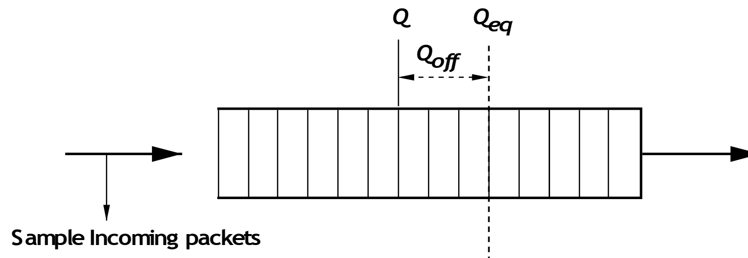


Figure 5.1: The QCN CP randomly samples incoming packets and (if needed) sends a feedback message with a congestion metric to the source of the sampled packet in order to maintain the buffer occupancy near a desired operating point, Q_{eq}.

10Gbps source with an overall round-trip time (RTT) of 500μs traverses it. That is, it keeps the buffer occupancy at less than 5% of the bandwidth-delay product. We revisit buffer sizing in Section 5.6.

5.2 The QCN Algorithm

We now provide a brief overview of the QCN algorithm, focusing on those aspects which are relevant for the mathematical model. We omit a number of details that are important for an exact implementation. Those interested are referred to the IEEE 802.1Qau standard [136].

The QCN algorithm has two components: (i) the switch, or *Congestion Point (CP)* mechanism, and (ii) the source, or *Reaction Point (RP)* mechanism. The CP mechanism is concerned with measuring the extent of congestion at the switch buffer, and signaling this information back to the source(s). The RP mechanism is concerned with the actions that need to be taken when a congestion signal is received, and how sources must probe for available bandwidth when there is no congestion.

5.2.1 The CP Algorithm

The CP buffer is shown in Figure 5.1. The goal of the CP is to maintain the buffer occupancy at a desired operating point, Q_{eq}. The CP computes a congestion metric F_b (defined below). With a probability p_s (1% by default), it randomly samples an incoming packet

and, if the buffer is congested, sends the value of F_b in a feedback message to the source of the sampled packet.[1] Let Q denote the instantaneous queue occupancy and Q_{old} denote the queue occupancy when the last packet was sampled. Let $Q_{off} = Q - Q_{eq}$ and $Q_\delta = Q - Q_{old}$. Then F_b is given by the formula:

$$F_b = Q_{off} + wQ_\delta,$$

where w is a positive constant (set to 2 for the baseline implementation).

The interpretation is that F_b captures a combination of queue occupancy excess (Q_{off}) and rate excess (Q_δ). Thus, when $F_b > 0$, either the buffer or the link or both are oversubscribed. A feedback message containing F_b, quantized to 6 bits, is sent to the source of the sampled packet *only* when $F_b > 0$; nothing is signaled when $F_b \leq 0$.

5.2.2 The RP Algorithm

The basic RP behavior is shown in Figure 5.2. The RP algorithm maintains the following two quantities:

- *Current Rate* (R_C): The sending rate at any time.
- *Target Rate* (R_T): The sending rate *just before* the arrival of the last feedback message.

Rate decrease. This occurs only when a feedback message is received, in which case R_C and R_T are updated as follows:

$$R_T \leftarrow R_C, \tag{5.1}$$

$$R_C \leftarrow R_C(1 - G_d F_b), \tag{5.2}$$

where the constant G_d is chosen so that $G_d F_{bmax} = \frac{1}{2}$; i.e. the sending rate can decrease by at most 50%.

[1] In the actual implementation, the sampling probability varies between 1-10% depending on the severity of congestion. We neglect this feature here to keep the model tractable; refer to [6] and [136] for details.

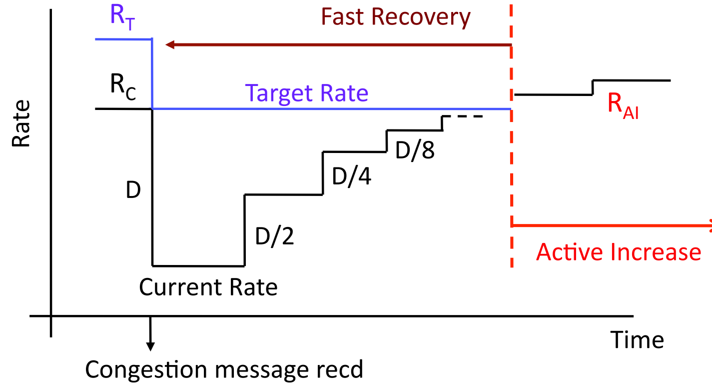


Figure 5.2: QCN RP operation. The current rate, R_C, is reduced multiplicatively upon a congestion message. The rate then increases in Fast Recovery in a series of jumps back towards the target rate, R_T (the rate before the decrease), and subsequently, undergoes Active Increase to probe for additional bandwidth.

Rate increase. Since the RP is not given positive rate-increase signals by the network, it needs a mechanism for increasing its sending rate on its own. This is achieved by using a *Byte Counter*, which counts the number of bytes transmitted by the RP. Recall that due to the absence of ACKs in Ethernet, packet transmission cannot be self-clocked like in TCP. Rate increase occurs in two phases: Fast Recovery and Active Increase.

Fast Recovery (*FR*). Immediately following a rate decrease episode, the Byte Counter is reset, and the RP enters the FR state. FR consists of 5 cycles. In each cycle, 150KB (100 packets, each 1500 bytes) of data are transmitted, as counted by the Byte Counter. At the end of each cycle, R_T remains unchanged while R_C is updated as follows:

$$R_C \leftarrow \frac{1}{2}(R_C + R_T).$$

Active Increase (*AI*). After 5 cycles of FR have completed, the RP enters the AI state where it probes for extra bandwidth on the path. In this phase, the RP increases its sending rate

by updating R_T and R_C at the end of each Byte Counter cycle as follows:

$$R_T \leftarrow R_T + R_{AI},$$
$$R_C \leftarrow \frac{1}{2}(R_C + R_T),$$

where R_{AI} is a constant (5Mbps by default).

Remark 5.1. Note that during the FR phase QCN performs *averaging* steps with R_C set to the average of R_C and R_T. The BIC-TCP algorithm [170] is the first to use averaging. It is instructive to understand the motives that led to BIC-TCP and QCN employing averaging. As Xu et al. [170] explain, the additive increase portion of the TCP algorithm can be viewed as determining the correct window size through a *linear* search process, whereas the BIC-TCP (for Binary Increase TCP) algorithm performs a more efficient *binary* search. QCN takes a control-theoretic view: a congestion control algorithm is zero-delay stable (i.e., stable without feedback delay) if the amount of rate increase after a drop is less than the amount of decrease during the drop. Since the rate before the last decrease is check-pointed as R_T and averaging ensures that $R_C < R_T$ throughout the FR phase, QCN is zero-delay stable (Section 5.4). In fact, as we show in Section 5.4.1, averaging (or binary increase) is much more stable than simple additive increase even with large feedback delays.

Remark 5.2. The duration of Byte Counter cycles measured in seconds depends on the current sending rate, and can therefore become unacceptably large when R_C is small, jeopardizing the speed of bandwidth recovery (or responsiveness). Therefore, a *Timer* is also included in the actual implementation of QCN. The Byte Counter and Timer jointly determine rate increase times (refer to the standard [136] for details). We do not consider the Timer since it is primarily used during transience, and we are mainly interested in the steady state stability behavior of QCN.

5.3 The QCN Fluid Model

The fluid model presented below corresponds with the simplified version of QCN from the previous section. The derivation of the equations, for the most part, is part of the research

literature [152]. The main distinction with typical fluid models of congestion control loops is in our use of two variables, R_C and R_T, to represent source behavior. This is a necessary step, since although R_C and R_T are inter-dependent variables, neither can be derived from the other.

We consider a 'dumb-bell' topology with N sources sharing a single link of capacity C. The RTT is assumed to be the same for all sources and equal to τ seconds. The source variables evolve according to the following differential equations:

$$\begin{aligned}\frac{dR_C}{dt} &= -G_d F_b(t-\tau) R_C(t) R_C(t-\tau) p_r(t-\tau) \\ &\quad + \left(\frac{R_T(t) - R_C(t)}{2}\right) \frac{R_C(t-\tau) p_r(t-\tau)}{(1-p_r(t-\tau))^{-100} - 1},\end{aligned} \tag{5.3}$$

$$\begin{aligned}\frac{dR_T}{dt} &= -\left(R_T(t) - R_C(t)\right) R_C(t-\tau) p_r(t-\tau) \\ &\quad + R_{AI} R_C(t-\tau) \frac{(1-p_r(t-\tau))^{500} p_r(t-\tau)}{(1-p_r(t-\tau))^{-100} - 1},\end{aligned} \tag{5.4}$$

where $p_r(t)$ is the *reflection probability* at the switch, and $F_b(t)$ is the congestion metric. These quantities are related to the queue occupancy, $Q(\cdot)$, at the switch and evolve as follows:

$$\frac{dQ}{dt} = \begin{cases} NR_C(t) - C & \text{if } q(t) > 0, \\ \max\left(NR_C(t) - C, 0\right) & \text{if } q(t) = 0, \end{cases} \tag{5.5}$$

$$F_b(t) = Q(t) - Q_{eq} + \frac{w}{Cp_s}(NR_C(t) - C), \tag{5.6}$$

$$p_r(t) = p_s \mathbb{I}_{[F_b(t)>0]}, \tag{5.7}$$

where p_s is the sampling probability. Equations (5.5) and (5.6) are self-explanatory. Equation (5.7) captures the fact that sampled packets result in a congestion feedback message being reflected back to the source only if $F_b > 0$.[2]

Equations (5.3) and (5.4) each consist of a negative term (representing rate decrease), and a positive term (representing rate increase). Let us first consider the simpler negative

[2]It must be noted that when $p_r = 0$, the right-hand sides of Equations (5.3) and (5.4) are to be interpreted as the resulting limits as $p_r \to 0$.

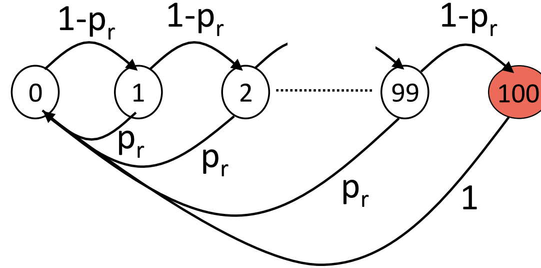


Figure 5.3: Markov chain corresponding to current rate (R_C) increases at the QCN RP.

terms. These terms model the decrease in R_C and R_T due to congestion feedback messages and correspond to equations (5.1) and (5.2). Observing that congestion feedback messages arrive to each source at rate $R_C(t-\tau)p_r(t-\tau)$, we obtain the negative terms.

Now consider the positive term in (5.3). A rate increase occurs each time 100 packets are sent and no congestion feedback message is received. The change in $R_C(t)$ at each such event is given by:

$$\Delta R_C(t) = \frac{R_C(t) + R_T(t)}{2} - R_C(t) = \frac{R_T(t) - R_C(t)}{2}. \tag{5.8}$$

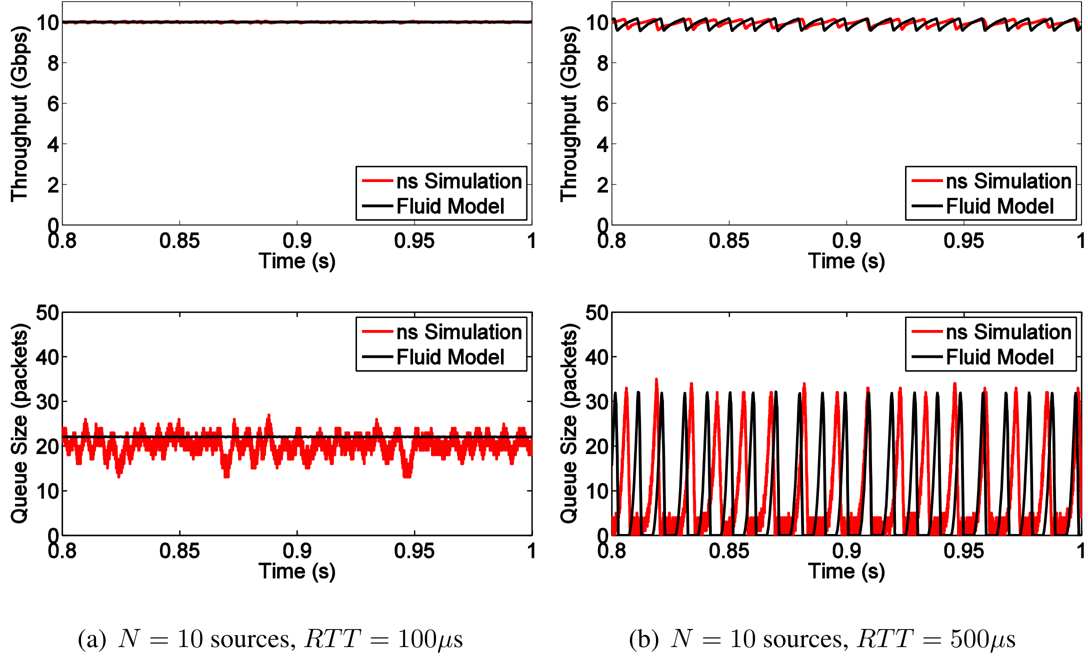
To compute the rate at which increase events occur, we consider the Markov Chain shown in Figure 5.3. It is not difficult to see that if each packet is reflected with probability p_r, then the average number of packets that must be sent before an increase event occurs is precisely equal to the expected number of steps it takes to hit state 100 starting from state 0 in this Markov Chain. This is given by:

$$\mathbb{E}_0(T_{100}) = \frac{(1-p_r)^{-100} - 1}{p_r}.$$

Therefore, because packets from a source arrive at the switch with rate $R_C(t-\tau)$ at time t, the average time between increase events is:

$$\Delta T = \frac{(1-p_r(t-\tau))^{-100} - 1}{R_C(t-\tau)p_r(t-\tau)}. \tag{5.9}$$

Dividing (5.8) by (5.9), we obtain the positive term in (5.3). The positive term in (5.4) is derived similarly.



(a) $N = 10$ sources, $RTT = 100\mu$s (b) $N = 10$ sources, $RTT = 500\mu$s

Figure 5.4: Comparison of QCN fluid model and ns2 simulation. The model matches the simulations well and accurately predicts rate and queue occupancy oscillations.

5.3.1 Model validation

We have verified the fidelity of the fluid model against packet-level simulations of QCN using the ns2 [126] network simulator. A representative example with $N = 10$ sources at two different RTTs is shown in Figure 5.4. As can be seen, the model and simulations match quite well. In particular, the model accurately predicts the amplitude and frequency of the rate and queue occupancy oscillations in different scenarios. Hence, the fluid model can be used to study the QCN control loop's stability.

5.4 Stability Analysis of the Fluid Model

We now use the fluid model to analyze the stability of the QCN control loop in the presence of feedback delay. Define:

$$\eta(p_s) \triangleq \frac{p_s}{(1-p_s)^{-100}-1}, \quad \zeta(p_s) \triangleq \frac{(1-p_s)^{500}p_s}{(1-p_s)^{-100}-1}.$$

It is easily verified that the fluid model (5.3)–(5.7) has the following unique fixed point:

$$
\begin{aligned}
R_C^* &= \frac{C}{N}, \\
R_T^* &= \frac{C}{N} + \frac{\zeta(p_s)R_{AI}}{p_s}, \\
Q^* &= Q_{eq} + \frac{\eta(p_s)\zeta(p_s)NR_{AI}}{2p_s^2 G_d C}.
\end{aligned}
$$

We are interested in understanding if, and under what conditions, is this fixed point locally stable. The standard approach we undertake is to linearize the system around the fixed point, and use tools from Linear Control Theory [54] to study its stability.

The linearization of the differential equations is straightforward. Omitting the algebra, the linearized system describing the evolution of $\delta R_C(t) \triangleq R_C(t) - R_C^*$, $\delta R_T(t) \triangleq R_T(t) - R_T^*$, and $\delta Q(t) \triangleq Q(t) - Q^*$ is given by:

$$\frac{d\delta R_C}{dt} = -a_1\delta R_C(t) + a_2\delta R_T(t) - a_3\delta R_C(t-\tau) - a_4\delta Q(t-\tau), \tag{5.10}$$

$$\frac{d\delta R_T}{dt} = b\delta R_C(t) - b\delta R_T(t), \tag{5.11}$$

$$\frac{d\delta Q}{dt} = N\delta R_C(t), \tag{5.12}$$

where:

$$
\begin{aligned}
a_1 &= \frac{\eta(p_s)}{2}R_C^* + \frac{\eta(p_s)\zeta(p_s)}{2p_s}R_{AI}, \quad a_2 = \frac{\eta(p_s)}{2}R_C^*, \\
a_3 &= G_d w R_C^*, \quad a_4 = p_s G_d {R_C^*}^2, \quad b = p_s R_C^*.
\end{aligned}
$$

We can now analyze the stability of this linear time-delayed system from its characteristic equation whose roots constitute the poles of the system. The characteristic equation of (5.10)–(5.12), derived in Appendix D, is given by:

$$1 + G(s) = 0, \tag{5.13}$$

where

$$G(s) = e^{-s\tau}\frac{a_3(s+b)(s+\gamma)}{s(s^2+\beta s+\alpha)}, \tag{5.14}$$

with $\gamma = Cp/w$, $\beta = b + a_1$, and $\alpha = b(a_1 - a_2)$.

The following theorem provides the largest feedback delay for which the linearized QCN fluid model is stable. The proof is based on a standard Bode Analysis [54] of the characteristic equation.

Theorem 5.1. *Let*

$$\tau^* = \frac{1}{\omega^*}\left(\arctan(\frac{\omega^*}{b}) - \arctan(\frac{\omega^*}{\beta}) + \arctan(\frac{\omega^*}{\gamma})\right), \tag{5.15}$$

where

$$\omega^* = \sqrt{\frac{a_3^2}{2} + \sqrt{\frac{a_3^4}{4} + \gamma^2 a_3^2}}. \tag{5.16}$$

Then $\tau^* > 0$, and the system (5.10)–(5.12) *is stable for all $\tau \le \tau^*$.*

Proof. Since $\beta > b$, we have:

$$\arctan(\frac{\omega^*}{\beta}) < \arctan(\frac{\omega^*}{b}),$$

which implies $\tau^* > 0$. The proof of stability follows by applying the Bode stability criterion [54] to $G(s)$. Define

$$r(\omega) = |G(j\omega)|, \quad \theta(\omega) = -\angle G(j\omega),$$

so that $G(j\omega) = r(\omega)e^{-j\theta(\omega)}$. We upper bound $r(\omega)$ as follows:

$$\begin{aligned} r(\omega)^2 &= \frac{a_3^2(\omega^2+b^2)(\omega^2+\gamma^2)}{\omega^2((\omega^2-\alpha)^2+\beta^2\omega^2)}, \\ &< \frac{a_3^2(\omega^2+b^2)(\omega^2+\gamma^2)}{\omega^4(\omega^2+\beta^2-2\alpha)}, \\ &< \frac{a_3^2(\omega^2+\gamma^2)}{\omega^4}. \end{aligned} \tag{5.17}$$

The last inequality holds because $\beta^2 - 2\alpha > b^2$, which can be checked by plugging in

$\beta = b + a_1$, $\alpha = b(a_1 - a_2)$. Now with ω^* given by (5.16), the bound (5.17) implies $r(\omega^*) < 1$. In particular, the 0-dB crossover frequency (ω_c such that $r(\omega_c) = 1$) occurs at some $\omega < \omega^*$. Hence, the Bode stability criterion implies that if $\theta(\omega) < \pi$ for all $0 \leq \omega < \omega^*$, the system is stable. But for $0 \leq \omega < \omega^*$, we have:

$$\begin{aligned}\theta(\omega) &= \pi + \omega\tau + \arctan(\frac{\omega^2 - \alpha}{\beta\omega}) - \arctan(\frac{\omega}{b}) - \arctan(\frac{\omega}{\gamma}),\\ &< \pi + \omega\tau + \arctan(\frac{\omega}{\beta}) - \arctan(\frac{\omega}{b}) - \arctan(\frac{\omega}{\gamma}),\\ &= \pi + \omega\tau - \arctan\left(\frac{(\beta - b)\omega}{\beta b + \omega^2}\right) - \arctan(\frac{\omega}{\gamma}),\\ &\leq \pi + \omega\tau - \arctan\left(\frac{(\beta - b)\omega}{\beta b + \omega^{*2}}\right) - \arctan(\frac{\omega}{\gamma}). \end{aligned} \tag{5.18}$$

The first inequality holds because $\alpha > 0$ and the last inequality holds because $\arctan(.)$ is an increasing function. Now let:

$$\Psi(\omega) = \pi + \omega\tau - \arctan\left(\frac{(\beta - b)\omega}{\beta b + \omega^{*2}}\right) - \arctan(\frac{\omega}{\gamma}).$$

Note that $\Psi(0) = \pi$ and for $\tau \leq \tau^*$, $\Psi(\omega^*) \leq \pi$. Moreover, since $\arctan(x)$ is concave for $x \geq 0$, $\Psi(\omega)$ is convex on $\omega \in [0, \omega^*]$. Therefore $\Psi(\omega) \leq \pi$ for all $0 \leq \omega \leq \omega^*$, and (5.18) implies $\theta(\omega) < \pi$, completing the proof. □

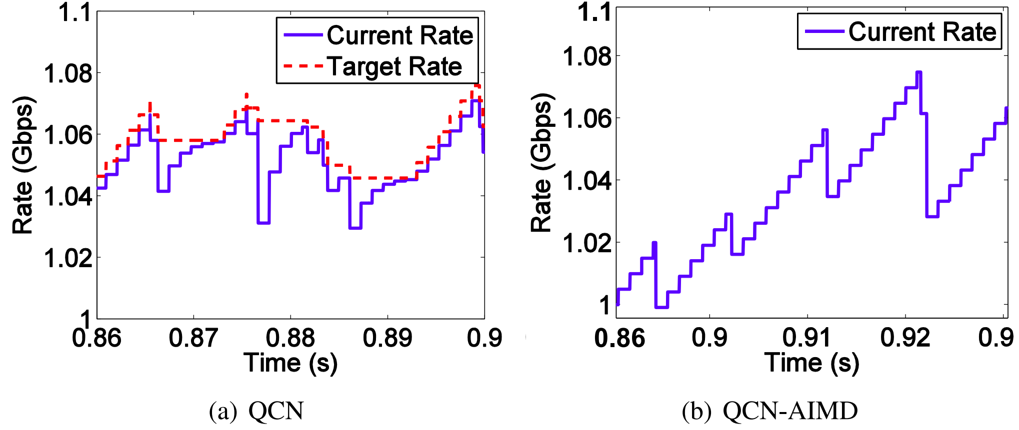
An immediate corollary of Theorem 5.1 is that the linearized fluid model is zero-delay stable. This confirms the intuitive argument given for zero-delay stability of QCN in Remark 5.1 of Section 5.2.

Corollary 5.1. *If* $\tau = 0$*, system* (5.10)–(5.12) *is stable.*

Proof. This follows because $\tau^* > 0$ in Theorem 5.1. □

5.4.1 Averaging in QCN

As previously discussed, the QCN Reaction Point averages R_C and R_T during the Fast Recovery phase. We now use fluid models to show that this averaging improves the stability of QCN as the lag in the control loop increases.



(a) QCN (b) QCN-AIMD

Figure 5.5: Rate dynamics of QCN and QCN-AIMD. QCN-AIMD does not do the averaging steps in Fast Recovery; it immediately enters Active Increase following a rate decrease.

Consider a modified QCN RP algorithm, henceforth called QCN-AIMD, where Active Increase begins *immediately* following a rate decrease; i.e. there is no Fast Recovery (see Figure 5.5 for an illustration). In Active Increase, the QCN-AIMD RP increases its sending rate each time the Byte Counter counts out 100 packets:

$$R_C \leftarrow R_C + R_{AI}.$$

The CP algorithm remains the same as QCN.[3]

A fluid model for QCN-AIMD can be derived similarly as for QCN. In fact, since QCN-AIMD has no R_T variable, we only need to change the R_C equation (5.3) to the following:

$$\frac{dR_C}{dt} = -G_d F_b(t-\tau) R_C(t) R_C(t-\tau) p_r(t-\tau) + R_{AI} \frac{R_C(t-\tau) p_r(t-\tau)}{(1-p_r(t-\tau))^{-100} - 1}. \quad (5.19)$$

This, along with equations (5.5)–(5.7), constitute the QCN-AIMD fluid model. We can now linearize the QCN-AIMD model around its fixed point and analyze its stability. This is done in Appendix E, where we prove the following theorem:

[3]QCN-AIMD and QCN are analogous to TCP and BIC-TCP respectively. However, an important distinction is that QCN-AIMD and QCN get multi-bit feedback from the network, allowing them to cut their rates by different factors corresponding to the extent of congestion.

Theorem 5.2. *Let*

$$\hat{\tau} = \frac{1}{\hat{\omega}} \left(\arctan(\frac{\hat{\omega}}{\gamma}) + \arctan(\frac{\hat{a}}{\hat{\omega}}) \right), \tag{5.20}$$

where

$$\hat{\omega} = \sqrt{\frac{a_3^2 - \hat{a}^2}{2} + \sqrt{\frac{(a_3^2 - \hat{a}^2)^2}{4} + \gamma^2 a_3^2}}. \tag{5.21}$$

Here $a_3 = G_d w R_C^*$ *and* $\gamma = Cp/w$ *are the same constants found in the QCN model, and* $\hat{a} = \eta(p_s) R_{AI}$*. The linearized QCN-AIMD fluid model, given by equations* (E.1)–(E.2)*, is stable if and only if* $\tau < \hat{\tau}$*.*

Theorems 5.1 and 5.2 provide the largest feedback delay for which the linear models of the QCN and QCN-AIMD control loops retain stability. The following theorem compares these two and proves that under mild conditions, the QCN system — with its use of averaging — is stable for larger lags in the control loop.

Theorem 5.3. *Let* τ^* *and* $\hat{\tau}$ *be given by* (5.15) *and* (5.20) *respectively. If*

$$\frac{R_{AI}}{C} \max \left(\frac{\eta(p_s)^2/p_s}{G_d}, \frac{2\eta(p_s) + 4p_s}{G_d}, \frac{\eta(p_s)w}{p_s} \right) < 0.1, \tag{5.22}$$

$$\frac{N R_{AI}}{C} < 0.2, \tag{5.23}$$

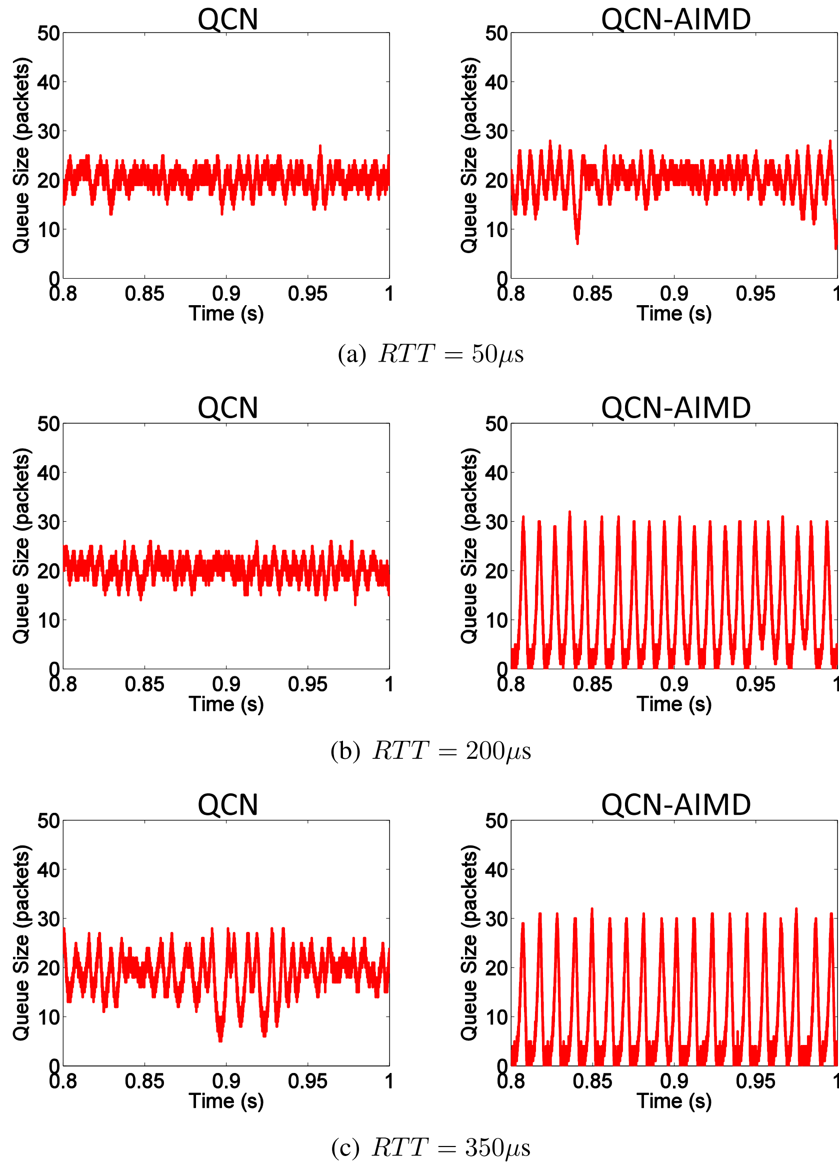
then $\tau^* > \hat{\tau}$*.*

Proof. See Appendix F. □

The conditions of Theorem 5.3 are easily satisfied in practice. For instance, for the baseline QCN parameters $C = 10$Gbps, $R_{AI} = 5$Mbps, $G_d = 1/128$, $w = 2$, $p_s = 0.01$, (5.22) is satisfied and (5.23) is equivalent to $N < 400$. As previously explained, there are typically a small number (e.g., less than 10) of sources active on each data center path. Hence, this condition is also satisfied in practice.

5.4.2 Simulations

We now verify the theoretical predictions of this section using ns2 simulations. We compare the stability of QCN and QCN-AIMD as the RTT (and hence, the feedback delay) increases.



(a) $RTT = 50\mu$s

(b) $RTT = 200\mu$s

(c) $RTT = 350\mu$s

Figure 5.6: Queue occupancy for QCN and QCN-AIMD with baseline parameters and RTT of (a) 50μs, (b) 200μs, and (c) 350μs. $N = 10$ sources share a single 10Gbps bottleneck. The desired operating point at the switch buffer, Q_{eq}, is set to 22 packets. The results validate the stability margins predicted for the two schemes by Theorems 5.1 and 5.2, and confirm that QCN has better stability than QCN-AIMD (Theorem 5.3).

Figure 5.6 shows the queue occupancy with the two schemes when 10 sources contend for a single 10Gbps bottleneck link, for three different RTTs. We use the baseline QCN

parameters mentioned above. In these scenarios, according to Theorems 5.1 and 5.2, the *linearized* QCN and QCN-AIMD control loops are stable for RTTs less than $\tau^* = 249\mu$s and $\hat{\tau} = 189\mu$s respectively.

As shown, when the RTT is small (50μs), both schemes are able to keep the queue occupancy stable around $Q_{eq} = 22$ packets. But when the RTT is increased to 200μs, QCN-AIMD can no longer control the oscillations and the queue underflows, while the queue occupancy for QCN continues to be stable as predicted. Even with RTT equal to 350μs, which is beyond the stability margin, τ^*, of the linearized model, QCN gracefully keeps the queue occupancy closely hovering around 22 packets (albeit with an increase in the amplitude of oscillations). It is only after the RTT increases beyond 500μs that the queue occupancy with QCN begins to underflow.

The results confirm that our analysis can predict the stability margins of QCN fairly accurately for a given set of parameters. They also show that QCN has better stability than QCN-AIMD and is more robust to feedback delay due to its use of averaging in the Fast Recovery phase. As we explore in depth in the next section, this suggests a general method for improving the robustness of *any* control loop to increasing lags in the control loop.

5.5 The Averaging Principle

In this section, we articulate the Averaging Principle (AP) which is a simple and general method for improving the stability of a control loop in the presence of increasing feedback delay. The QCN algorithm employs the AP during its Fast Recovery phase and, as was shown via fluid model analysis and simulations in the previous section, it is this phase that underlies QCN's good stability.

Control theory prescribes two methods of feedback compensation that have both been used to design stable congestion control algorithms as lags (round-trip times) increase. In one approach, an estimate of the RTT is used to find the correct "gains" for the loop to be stable. For example, this is the approach taken by FAST [163], XCP [96], RCP [44] and HSTCP [50].[4] The second approach improves stability by increasing the order of

[4]HSTCP does not explicitly use RTT estimates to adjust gains. Rather, it varies gains based on current window size, which implicitly depends on the RTT: the larger the RTT, the larger the current window.

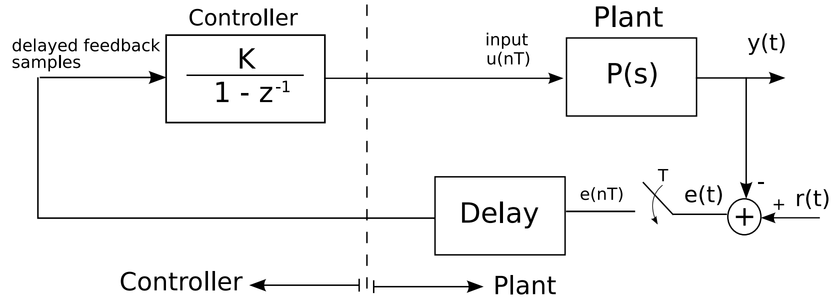


Figure 5.7: A generic sampled control system.

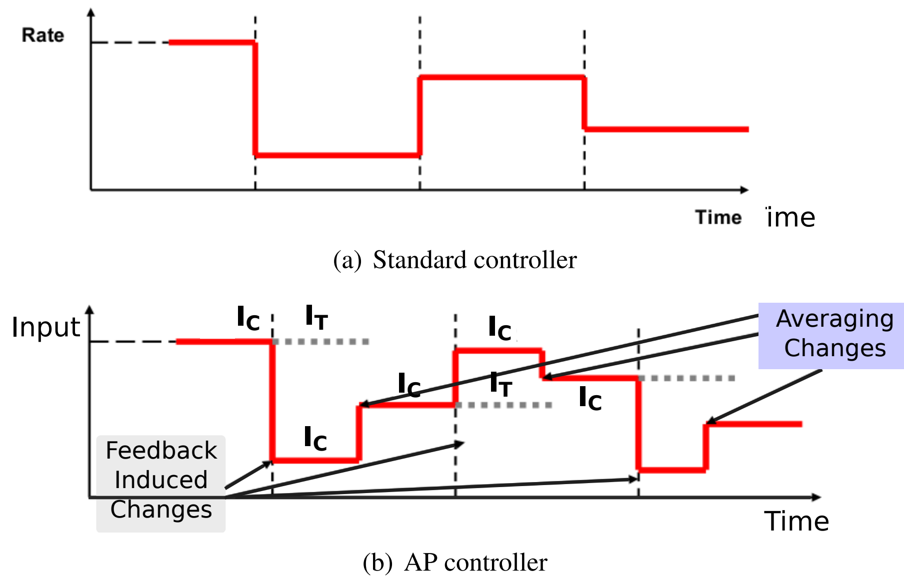
the feedback signal by considering higher order derivatives of the queue occupancy. For instance, the active queue management schemes REM [23] and PI [76] compute a weighted sum of the queue size and its derivative (which equals input rate less output rate) as the congestion signal. This is also used in XCP and RCP.

On the other hand, BIC-TC [170] operates stably in high bandwidth-delay product networks, even though it neither changes loop gains based on RTT nor uses higher order feedback. But it operates in the self-clocked universe of Internet congestion control schemes, where window size changes are made once every RTT. So it is possible that it implicitly exploits knowledge of RTTs to derive stability. However, as we have seen, the QCN algorithm has no notion of RTTs and yet achieves very good stability. This and the similarity of operation of the BIC-TCP and QCN algorithms suggest there may be a more fundamental reason for their good stability. Our attempt to understand this reason has led us to the AP, which we now describe.

5.5.1 Description of the AP

We explain the AP in the context of a generic control system such as the one shown in Figure 5.7. The output of the system, $y(t)$, tracks a reference signal, $r(t)$, which is often a constant. The error, $e(t) = r(t) - y(t)$, is sampled with period T and fed back to the controller with some delay. The controller incrementally adjusts the input to the plant as follows:

$$u((n+1)T) = u(nT) + K\,e(nT), \tag{5.24}$$



(a) Standard controller

(b) AP controller

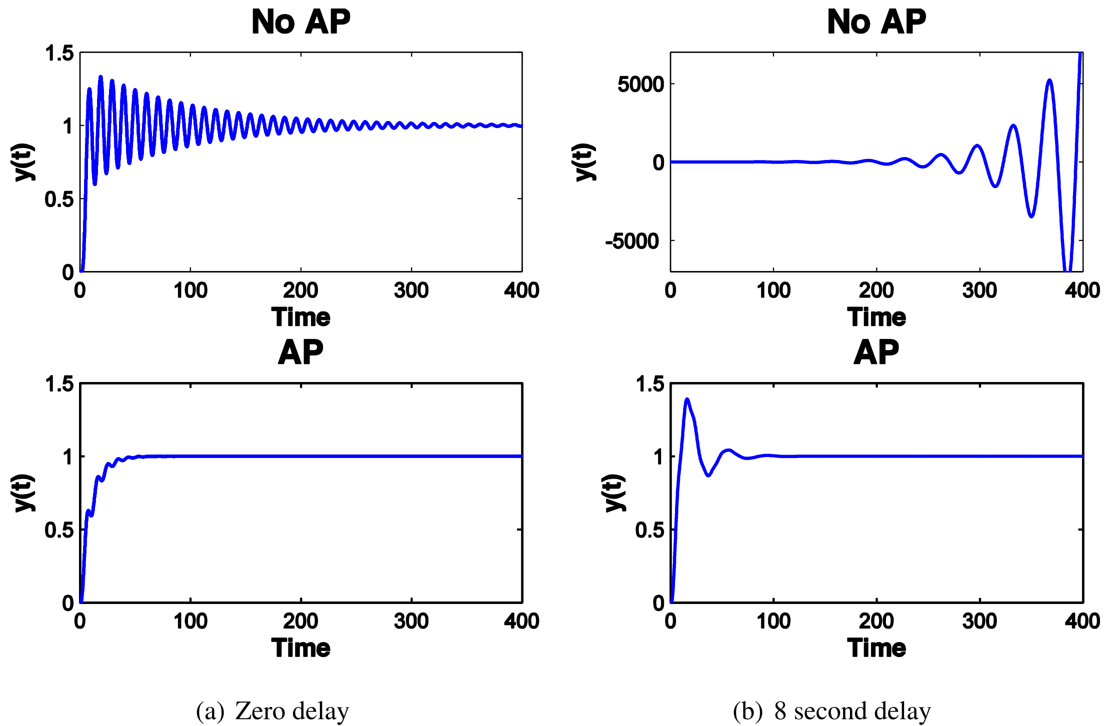
Figure 5.8: Plant input signal generated by the standard controller and the AP controller.

where K is the controller gain. The input is held constant between sampling times; i.e., $u(t) = u(nT)$ for $nT \leq t < (n+1)T$. Figure 5.8(a) illustrates the action of the controller just described. To translate this to the congestion control setting, the source (the controller) chooses a packet sending rate (i.e., the input $u(t)$). The output $y(t)$ is the queue size and rate information at a router or a switch. The error $e(t)$ is a deviation of the current queue size and rate from target values. The sampling period T is a function of the packet arrival rate, since most congestion control algorithms sample packets.

Figure 5.8(b) shows the AP controller for the same generic system. This controller reacts to feedback messages it receives from the plant exactly as in (5.24), at times which are labelled "feedback-induced changes" in the figure. Note that feedback-induced changes occur every T units of time.

At any time, let I_C (for current input) denote the value of the input, and let I_T (for target input) denote the value of the input *before* the last feedback-induced change. Precisely $T/2$ time units after every feedback-induced change, the AP controller performs an "averaging change", where the controller changes I_C as follows:

$$I_C \leftarrow \frac{I_C + I_T}{2}.$$



(a) Zero delay

(b) 8 second delay

Figure 5.9: Unit step responses for the standard and AP controllers with: (a) zero delay and (b) 8 second delay. The AP controller is more robust and keeps the system stable for larger feedback delays.

The term averaging comes from the above equation: I_C moves to the average of its value before and after the last feedback-induced change. Note that the QCN (and BIC-TCP) algorithms perform averaging several times after receiving a congestion signal.

To illustrate the effectiveness of the AP, let us consider an example linear, zero-delay stable control system with plant transfer function

$$P(s) = \frac{s+1}{s^3 + 1.6s^2 + 0.8s + 0.6},$$

controller gain $K = 1/8$, and sampling period $T = 1$s. We use a unit step function as the reference signal, and compare the stability of the control loop with and without AP as the delay increases. As shown in Figure 5.9, when there is no delay, both schemes are stable. However, the AP controller induces less oscillations and settles faster. As the delay increases to $\tau = 8$s, the standard controller becomes unstable, while the AP controller is

more robust and continues to be stable. In fact, in this example, the AP controller is stable for delays up to $\tau = 15$s.

Remark 5.3. In the generic setting of Figure 5.8, the feedback signal can take both positive and negative values. However, in some cases of interest, the feedback might be restricted to negative values only; QCN is such an example. The definition of AP is the same in either case: following every feedback-induced change, make (one or more) averaging changes.

5.5.2 Analysis

We now address the question: Why does the AP improve the stability of the basic controller? To anticipate the answer, we find that the AP controller behaves like a Proportional-Derivative (PD) controller. In fact, the AP is *algebraically equivalent* to a PD controller. Since the PD controller is well-known to stabilize control loops when lags increase [54], this equivalence explains the reason for the improved stability of the AP.

Hence, the AP is an alternative to PD control that does not require explicitly computing a derivative. This feature can be very useful in practice. For instance, using a PD controller for congestion control requires that the switches compute a derivative of the congestion state, which can be cumbersome to do in hardware. The AP achieves the same benefit at the source without modifying switches.

Before proving the claimed equivalence of the AP and PD controllers, we demonstrate it using our example. Recall the AP for the sampled control system in Figure 5.7, composed of periodic feedback-induced changes given by:

$$I_T \leftarrow I_C,$$
$$I_C \leftarrow I_C + Ke(nT),$$

and averaging changes at the midpoints of the sampling periods:

$$I_C \leftarrow \frac{I_C + I_T}{2}.$$

Now consider the PD control scheme of Figure 5.10. Here the error samples are not directly

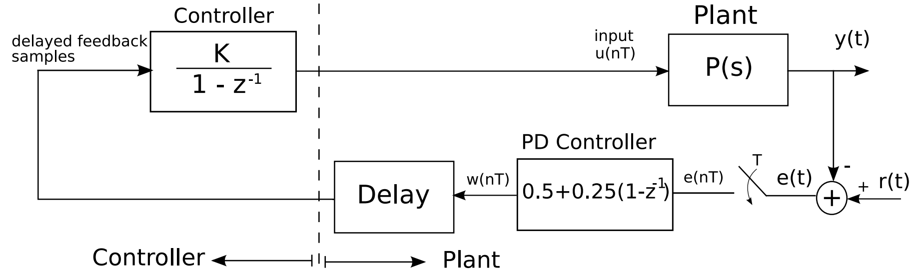
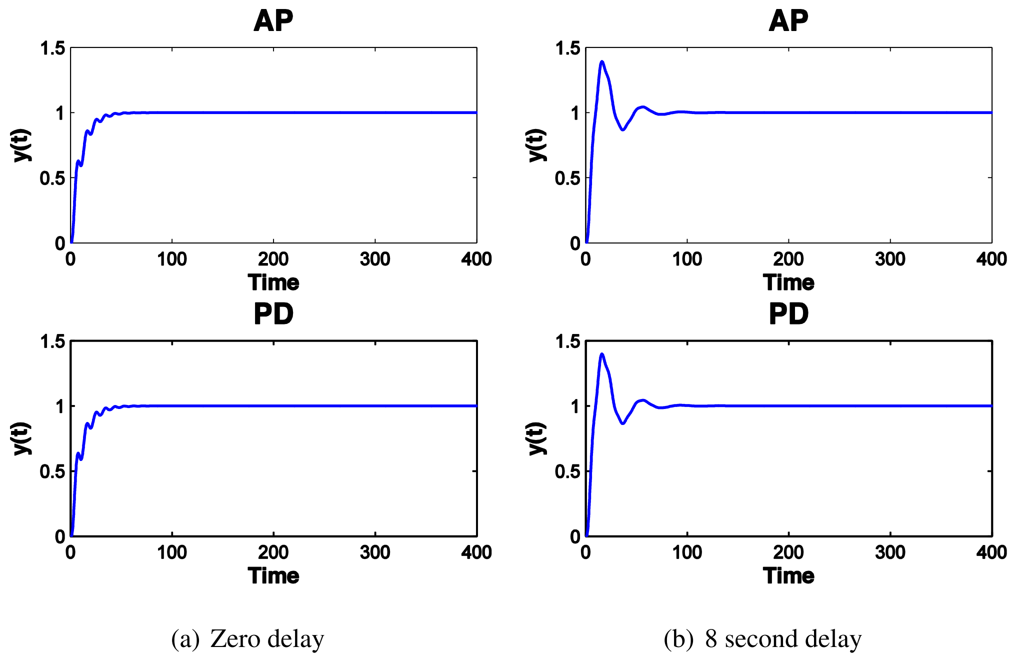


Figure 5.10: Equivalent PD control scheme to the AP controller.



(a) Zero delay (b) 8 second delay

Figure 5.11: Step response for AP and PD controllers for different feedback delays.

fed back. Instead, the feedback samples are computed as:

$$
\begin{aligned}
w(nT) &= \frac{1}{2}e(nT) + \frac{1}{4}(e(nT) - e((n-1)T)), \\
&\approx \frac{1}{2}e(nT) + \frac{T}{4}\frac{d}{dt}e(nT).
\end{aligned}
$$

Here, $w(nT)$ is a weighted sum of two terms: a *proportional* term, and a (discrete) *derivative* term. Hence, this is a discrete-time PD controller. The step response of this control loop is compared to the output of the AP controller in Figure 5.11. As shown in the

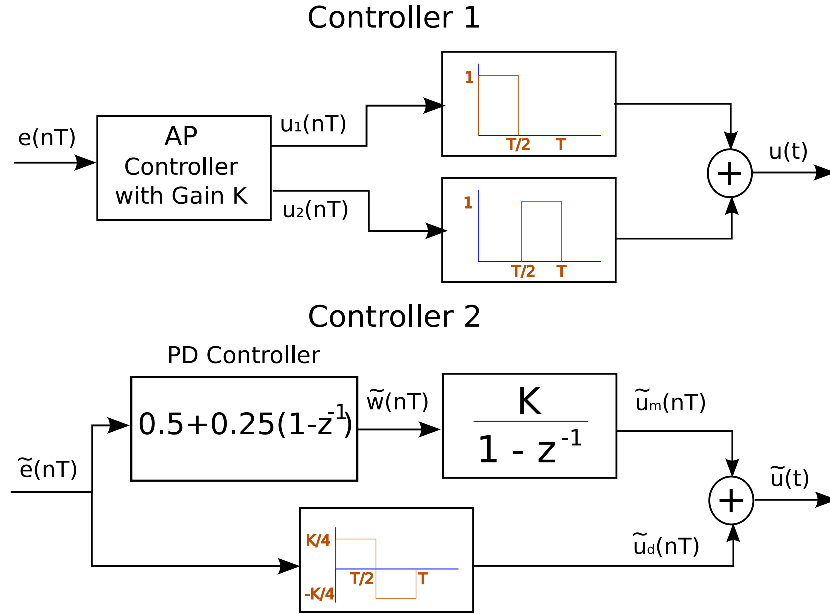


Figure 5.12: The block diagrams of two equivalent controllers. Controller 1 is the AP controller and Controller 2 is the PD controller (with an additional filter on the lower branch).

figure, the outputs of the AP and PD controllers are essentially identical.

Formally, consider the two controllers shown in Figure 5.12. Both controllers map a sequence of error samples to an input signal driving the plant. Controller 1 is the AP controller, and Controller 2 is (essentially) the PD controller. The input-output relationships of the two controllers are given by the following equations:

Controller 1:

$$u_1(nT) = u_2((n-1)T) + Ke(nT) \tag{5.25}$$

$$u_2(nT) = \frac{u_1(nT) + u_2((n-1)T)}{2} \tag{5.26}$$

$$u(t) = \begin{cases} u_1(nT) & nT \leq t < nT + \frac{T}{2} \\ u_2(nT) & nT + \frac{T}{2} \leq t < nT + T \end{cases} \tag{5.27}$$

Controller 2:

$$\tilde{w}(nT) = \frac{1}{2}\tilde{e}(nT) + \frac{1}{4}(\tilde{e}(nT) - \tilde{e}((n-1)T)) \tag{5.28}$$

$$\tilde{u}_m(nT) = \tilde{u}_m((n-1)T) + K\tilde{w}(nT) \tag{5.29}$$

$$\tilde{u}_d(t) = \begin{cases} \frac{K}{4}\tilde{e}(nT) & nT \leq t < nT + \frac{T}{2} \\ -\frac{K}{4}\tilde{e}(nT) & nT + \frac{T}{2} \leq t < nT + T \end{cases} \tag{5.30}$$

$$\tilde{u}(t) = \tilde{u}_m(t) + \tilde{u}_d(t) \tag{5.31}$$

It should be noted $\tilde{u}_m(t) = \tilde{u}_m([\frac{t}{T}]T)$ in (5.31). For brevity, we will not explicitly point out this conversion between discrete and continuous time signals each time.

Theorem 5.4. *Controllers 1 and 2 are algebraically equivalent; i.e., if they are given the same input sequences $e(nT) = \tilde{e}(nT)$ for all $n \geq 0$, and have the same initial condition $u(0) = \tilde{u}(0)$, then $u(t) = \tilde{u}(t)$ for all $t \geq 0$.*

Proof. For $n \geq 0$, let $u_m(nT) = (u_1(nT) + u_2(nT))/2$ be the average of the output of Controller 1 in each sample interval. From equations (5.25)–(5.26), we get that:

$$u_m(nT) = u_2((n-1)T) + \frac{3K}{4}e(nT), \tag{5.32}$$

and, in turn, that:

$$u_1(nT) = u_m(nT) + \frac{K}{4}e(nT), \tag{5.33}$$

$$u_2(nT) = u_m(nT) - \frac{K}{4}e(nT). \tag{5.34}$$

Comparing (5.33) and (5.34) with (5.30) and (5.31), it suffices to establish: $u_m(nT) = \tilde{u}_m(nT)$ for all $n \geq 0$. We do this by showing that $u_m(\cdot)$ and $\tilde{u}_m(\cdot)$ satisfy the same

recursion. Equations (5.32) and (5.34) give:

$$\begin{aligned} u_m(nT) &= u_m((n-1)T) - \frac{K}{4}e((n-1)T) + \frac{3K}{4}e(nT) \\ &= u_m((n-1)T) + Kw(nT), \end{aligned} \tag{5.35}$$

where:

$$w(nT) = \frac{1}{2}e(nT) + \frac{1}{4}(e(nT) - e((n-1)T)). \tag{5.36}$$

The recursion defined by (5.35) and (5.36) for $u_m(\cdot)$ in terms of $e(\cdot)$, is identical to the recursion defined by (5.28) and (5.29) for $\tilde{u}_m(\cdot)$ in terms of $\tilde{e}(\cdot)$. But, by hypothesis, $e(nT) = \tilde{e}(nT)$ for all $n \geq 0$ and $u_m(0) = \tilde{u}_m(0)$, and the proof is complete. □

Theorem 5.4 states that the AP controller is exactly equivalent to Controller 2. Since the effect of $\tilde{u}_d(t)$ is typically negligible (see below), the AP controller is essentially equivalent to the top path of Controller 2, which is the same as the PD controller in Figure 5.10.

The effect of $\tilde{u}_d(t)$ is negligible if the sampling time T is small compared to the rise time of the plant. This is a standard design criterion in digital control systems. For example, a simple rule of thumb is that the sampling time should be smaller than 10% of the dominant time constant of the plant [55]. Intuitively, if this condition is met, the plant does not 'see' the variations of the input within a sampling interval and only reacts to the *mean* value of the input. Since the mean of $\tilde{u}_d(t)$ in a sampling interval is zero, it does not affect the output of the plant. Thus, the AP and PD controllers are essentially identical, as we observed for our example in Figure 5.11.

Remark 5.4. There are a few possible extensions to basic AP scheme, including: averaging by factors other than 1/2 (e.g., $I_C \leftarrow (1-\alpha)I_C + \alpha I_T$ for $0 < \alpha < 1$), applying averaging changes at points other than the midpoint of the sampling interval, applying averaging changes more than once in the sampling interval, etc. All these cases can be analyzed in an identical manner to the basic form.

Remark 5.5. The common tradeoff in control theory between stability and responsiveness exists with the AP as well: an AP-enhanced control scheme is more stable than the original, but it is also more sluggish. This is evident from the form of the feedback signal in the

equivalent PD controller (see Figure 5.10):

$$w(nT) \approx \frac{1}{2}e(nT) + \frac{T}{4}\frac{d}{dt}e(nT).$$

Since the AP reduces the gain on the proportional term to $1/2$, it slows down the control loop. In practice, various techniques are used to improve the transient behavior of a system. Typically, these take the form of a temporary deviation from the normal controller behavior during times of transience. Some examples of this are BIC-TCP's *Fast Convergence* [170] and QCN's *Timer*, *Extra Fast Recovery*, and *Target Rate Reduction* mechanisms [136].

5.5.3 The AP applied to another congestion control algorithm: RCP

As further demonstration of the generality of the AP, we now apply it to the RCP [44] congestion control scheme. In RCP, each router offers a rate, $R(t)$, to flows passing through it. $R(t)$ is updated every T seconds according to:

$$R(t) = R(t-T)(1 + \frac{T}{d_0}F_b(t)), \tag{5.37}$$

where d_0 is a moving average of the RTT measured across all packets, and:

$$F_b(t) = \frac{\alpha(C - y(t)) - \beta\frac{q(t)}{d_0}}{C} \approx \frac{-\alpha\dot{q}(t) - \beta\frac{q(t)}{d_0}}{C}. \tag{5.38}$$

Here $y(t)$ is the measured input traffic rate during the last update interval, $q(t)$ is the instantaneous buffer size, C is the link capacity, and α and β are non-negative constants. We refer the reader to Dukkipati et al. [44] for details of RCP.

As seen in equation (5.37), RCP automatically adjusts the loop gain, d_0, based on the average RTT in the control loop. This automatic gain adjustment stabilizes RCP as the RTT varies. To demonstrate the effect of the AP on RCP's stability, we disable the automatic gain adjustment of RCP; i.e., we consider the case where d_0 is held constant at the router while the actual RTT increases.[5]

[5] It must be noted that this is only intended as a demonstration of how the AP can improve stability without gain adjustments; we are not suggesting a change to the RCP algorithm.

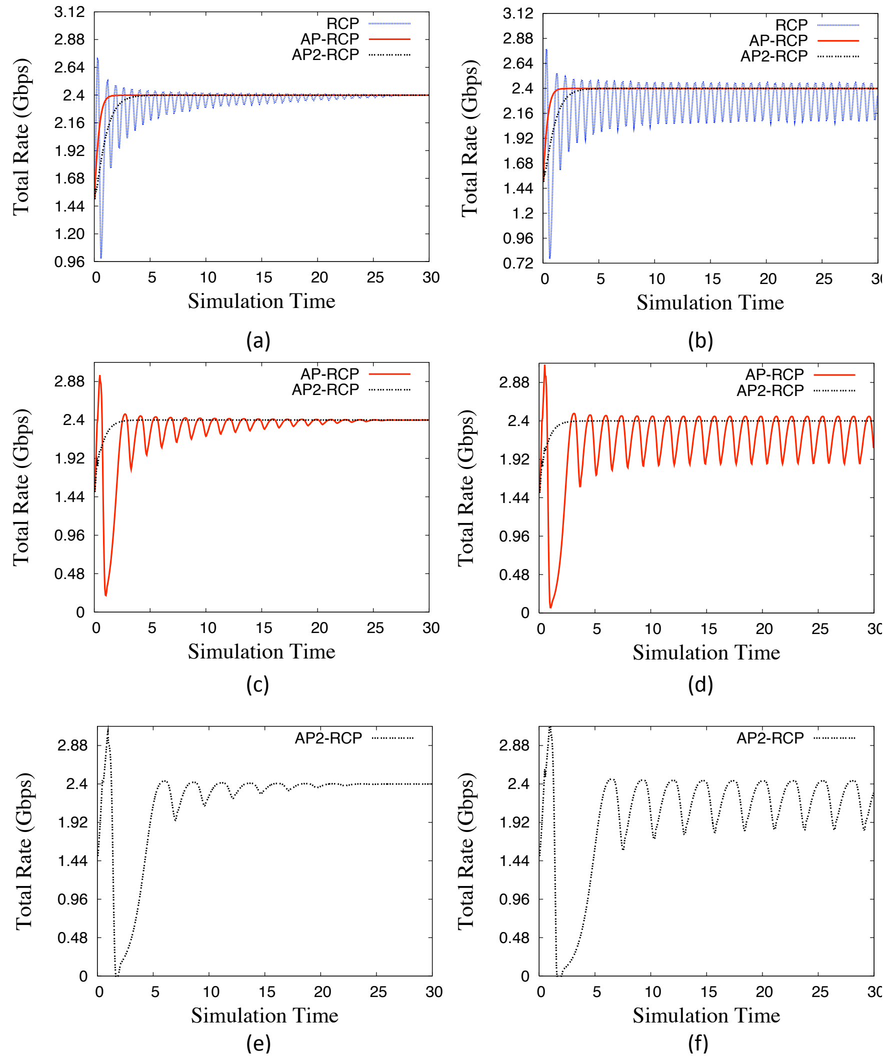


Figure 5.13: Total sending rate for RCP, AP-RCP and AP2-RCP with RTT of (a) 110ms (b) 115ms (c) 240ms (d) 250ms (e) 480ms (f) 490ms.

In what follows, we compare RCP with two AP-enhanced versions of the algorithm: AP-RCP and AP2-RCP. AP-RCP is just the application of the basic form of AP to RCP. AP2-TCP makes two averaging changes within each update interval: at $T/3$ and $2T/3$ after the start of interval.

Stability. Using ns2 [126], we simulate 10 long-lived RCP flows passing through a 2.4Gbps link. The RCP parameters are set to $\alpha = 0.1$, $\beta = 1$, the sampling period $T = 50$ms, and d_0 is fixed at 20ms. The RTT is varied and the results are shown in Figure 5.13. When the

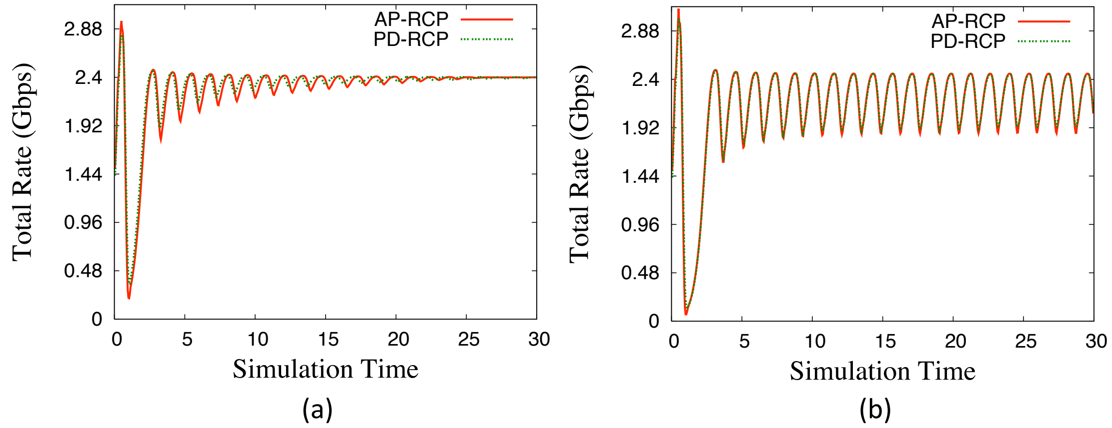


Figure 5.14: Comparison of AP-RCP and PD-RCP for (a) RTT = 240ms (b) 250ms. The two schemes are nearly identical.

RTT is smaller than 110ms, both RCP and its AP modifications are stable. RCP becomes unstable when the RTT increases to 115ms, but AP-RCP and AP2-RCP both continue to be stable for RTTs up to 240ms. AP-RCP becomes unstable at an RTT of 250ms, but AP2-RCP remains stable even when the RTT equals 480 ms.

PD Equivalence to AP. We now apply a PD controller to RCP (PD-RCP), and compare it with AP-RCP. The rate update equation for PD-RCP is given by:

$$R(t) = R(t-T)\left(1 + \frac{T}{d_0}\left(\frac{1}{2}F_b(t) + \frac{1}{4}(F_b(t) - F_b(t-T))\right)\right),$$

where $F_b(t)$ is still computed according to (5.38). Figure 5.14 shows that AP-RCP and PD-RCP are nearly identical: both are stable at RTT = 240ms, are unstable at RTT = 250ms, and exhibit very similar rate oscillations.

5.6 Buffer Sizing

The performance of congestion control algorithms crucially depends on the size of buffers at switches. If the buffers are small relative to the bandwidth-delay product of the flows they support, buffer occupancies will oscillate and either cause packet drops or under-utilization

of links. A consistent theme of this dissertation has been to use congestion control loops with good rate stability to reduce buffering requirements. We now explore this for QCN.

The TCP buffer sizing "rule of thumb" states that a single TCP flow requires a bandwidth-delay product ($C \times RTT$) amount of buffering to fully utilize a link of capacity C [159]. For example, a 10Gbps network with a 500μs round-trip time necessitates that switches have 625KB of buffering per-priority per-port. However, most commodity data center Ethernet switches have 2-10MB of buffering for 24-48 ports [53, 123, 65]. As discussed previously, this is dictated by what can fit on chip in commodity switches (Section 2.1.3) and is only likely to worsen as link speeds increase. Data center networks are already rolling out with 40Gbps Ethernet and the industry road map calls for the deployment of 100Gbps Ethernet in the near future.

The buffering requirement at a switch can be smaller if several flows are simultaneously traversing it. The paper by Appenzeller *et al.* [19] shows that when N flows share the link, the required buffer size for TCP goes down by a factor of $\sqrt{N}$ to $C \times RTT/\sqrt{N}$. This is because the variance of the total arrival rate to the buffer is reduced due to statistical multiplexing, and this leads to a reduction in the required buffering. More precisely, if N flows are multiplexed, the *standard deviation* of their aggregate sending rate equals:

$$\text{stdev}(\sum_{i=1}^{N} R_i) = \frac{\text{stdev}(\widetilde{R})}{\sqrt{N}},$$

where stdev($\widetilde{R}$) is the standard deviation in the sending rate when only one flow traverses the link. Note that the mean value of each R_i is C/N, but the mean value of $\tilde{R}$ equals C. However, the number of simultaneously active flows at a link in a data center network is typically small (fewer than 10). Therefore, it is hard to obtain the benefit of statistical multiplexing in the data center environment.

However, a congestion control algorithm with good stability has a similar "variance reduction" effect and hence also reduces buffering requirements, *even with a few flows*. Table 5.1 shows that over a 10Gbps link the standard deviation in the sending rate of a QCN source is significantly smaller than that of a TCP source as the RTT increases. QCN reduces the variance of the sending rate in two ways: (i) using multi-bit feedback allows

RTT	120μs	250μs	500 μs
TCP	265 Mbps	783 Mbps	1250 Mbps
QCN	14 Mbps	33 Mbps	95 Mbps

Table 5.1: Comparison of the standard deviation of the sending rate of a single 10Gbps TCP and QCN source, for different RTTs. The standard deviation is an order of magnitude smaller with QCN because of its better stability.

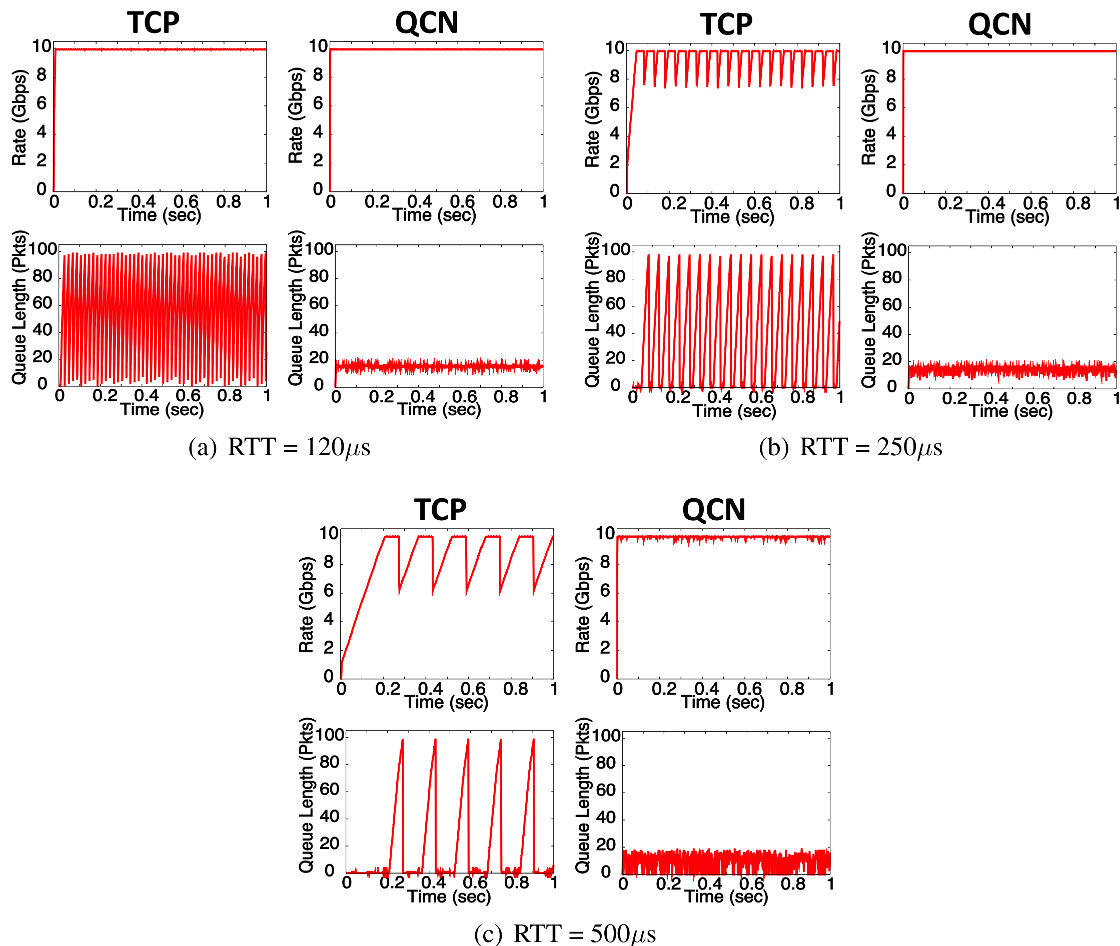
sources to cut their sending rates by factors smaller than 1/2 (as small as 1/128), and (ii) employing averaging further reduces the sending rate variance and improves rate stability.

To illustrate this further, we show the throughput and queue occupancy for one TCP and one QCN flow traversing a 10Gbps link in Figure 5.15. The buffer size at the bottleneck switch is 150KB which exactly equals the bandwidth-delay product with an RTT of 120μs. Hence, as seen in Figure 5.15(a), both algorithms achieve full throughout for this RTT. At higher RTTs of 250μs and 500μs, large queue occupancy oscillations cause TCP to lose throughput, whereas QCN remains stable.

Remark 5.6. The BIC-TCP [170] and CUBIC [70] algorithms also derive the benefit of averaging, but are constrained to cut window sizes (and rates) by a constant factor of 0.125 during congestion. So their rate variations, while smaller than TCP's, will be larger than QCN's. See also Cai *et al.* [33] for an analytical method of comparing the rate variance of congestion control algorithm via a *convex ordering* of their window size (or rate) growth functions.

5.7 Related Work

There have been a few studies on various aspects of the QCN algorithm in the literature. Jiang *et al.* [90] analyze QCN using the phase plane analysis method and show that QCN approaches its equilibrium mainly through the "sliding mode" motion, and may become unstable in certain conditions depending on the system parameters. Devkota *et al.* [42] consider the TCP incast problem in a QCN-enabled (but lossy) network and demonstrate some impairments. They propose modifications to QCN's sampling mechanism at the CP and rate increase mechanism at the RP for improving its performance in incast scenarios.



(a) RTT = 120μs

(b) RTT = 250μs

(c) RTT = 500μs

Figure 5.15: Link utilization and queue occupancy for TCP and QCN, as RTT increases. In each case, a single 10Gbps source traverses the bottleneck switch which has a total queue size of 100 packets (150KB).

Other work has considered the fairness among QCN flows. Kabbani *et al.* [92] propose Approximate Fair QCN (AF-QCN), an enhancement to QCN for programmable bandwidth partitioning among flows or traffic classes (tenants) in multi-tenant data centers. AF-QCN introduces a light-weight addition to the QCN CP based on the Approximate Fair Dropping [129] algorithm, which allows the CP to differentially send feedback messages to QCN RPs based on their deviation from the fair shares. Hayashi *et al.* [73] propose a modification to QCN's rate increase mechanism to improve fairness.

5.8 Final Remarks

Data center networks present new opportunities and challenges for developing new networking technologies. In this chapter, we studied one such technology: the L2 congestion control algorithm, QCN, developed for the IEEE 802.1Qau Congestion Notification [136] standard. We described the salient features of QCN and developed a high-fidelity fluid model corresponding to it. We determined the stability margins of the QCN algorithm using a linearization of the fluid model. We articulated the Averaging Principle (AP) and showed that it is the underlying reason for QCN's good stability properties. The AP applies to general control systems, not just congestion control systems. This aspect is worth pursuing further. We analyzed the AP controller from a control-theoretic point-of-view and showed that it is equivalent to a Proportional-Derivative (PD) controller in a strong algebraic sense. Finally, we showed that QCN's use of the AP and of a multi-bit feedback signal allows it to use much smaller buffers without loss of throughput compared to TCP.

Chapter 6

Conclusion

In this dissertation, we set out to understand the issues of packet transport in data center networks. We were motivated by the significant interest in building large-scale, high performance data center networks with low cost, commodity networking gear. Ultimately, we argued that the transport mechanisms designed for the Internet are inadequate for data centers, and presented new designs tailored to the unique constraints of the data center environment. We briefly summarize our main findings.

6.1 Summary

We showed via measurements in real production data centers that TCP — the dominant transport protocol in the Internet *and* data centers — does not meet the demands of data center applications. Specifically, data center applications have a diverse mix of short and long flows that simultaneously require very low latency (for the short flows) and high throughput (for the long flows). With TCP, long flows build up large queues in data center switches, which on the one hand increases the latency for short flows, and on the other hand imposes pressure on the limited buffer space in commodity switches. Also, TCP suffers from the Incast [158, 35] problem caused by synchronized bursty traffic patterns that are common in data centers. These problems are rooted in the interaction between TCP's buffer hungry congestion control algorithm, the shallow buffers in commodity switches, and the particular traffic patterns that data center applications generate.

We proposed the DCTCP congestion control algorithm and the HULL architecture for data center transport. DCTCP is an enhancement to TCP that combines an aggressive congestion signaling mechanism based on ECN at the switches with a novel control scheme at the sources. The algorithm essentially creates a very stable congestion control loop by having sources derive multi-bit information regarding the extent of congestion from the stream of ECN marks, and then react proportionately. This allows DCTCP to maintain very low queue occupancy in switches without loss of throughput. HULL builds on this idea by employing Phantom Queues that signal congestion (via ECN) even before queues form in the switches. In effect, Phantom Queues leave a small amount of bandwidth (e.g., 5–10% of the link capacity) as "headroom" for mice flows to fly through the network unhindered by queueing delay. HULL also benefits from DCTCP and hardware packet pacing to alleviate the bandwidth penalties associated with operating in a bufferless fashion.

We presented the design, analysis, and experimental evaluation of DCTCP and HULL. Our results show that DCTCP significantly reduces the buffer footprint in switches compared to TCP, providing high burst tolerance, low latency, and full throughput. HULL drives buffers nearly empty, reducing latency even further compared to DCTCP, particularly at the tail, with a configurable reduction in total throughput. Overall, in many scenarios, DCTCP provides a 4–10x reduction in switch latency compared to TCP, and HULL reduces latency by another order of magnitude compared to DCTCP.

We also presented a stability analysis of the QCN algorithm that has recently been standardized as the IEEE802.1Qau Congestion Notification [136] standard for Ethernet (Layer 2) congestion control. Our analysis of QCN compliments the extensive work of the standardization committee and provides a theoretical basis for understanding the QCN algorithm and the related Internet protocol, BIC-TCP. Further, through our study of QCN, we uncovered the Averaging Principle, which is a general method for stabilizing control loops as the feedback delay increases. We believe the Averaging Principle may be of interest in settings beyond congestion control.

6.2 Future Directions

The work in this dissertation fits in the broader context of the new challenges and opportunities that have arisen due to the significant interest and investment in large-scale data centers. Data centers provide an opportunity to revisit some of the fundamental issues of packet switched networks, such as congestion control, forwarding/routing, quality of service, and traffic engineering, as well as to explore new network architectures such as Software-Defined Networks (SDNs) [115, 67, 34]. They have the scale of a large subnetwork of the Internet, operate under new and unique constraints, and are controlled by a single administrative entity with the means and incentives to deploy new technologies. This makes data centers a very rich environment for the networking researcher.

We conclude the dissertation by discussing some of the directions in which our work can be pursued further.

Incremental deployability is not as large a concern in data centers as it is in the Internet. Alas, it *is* still a concern. Though we have been able to integrate the DCTCP algorithm into actual networking stacks [40] in a fairly short window of time — which gives us hope that these mechanisms will be deployed in production data centers in the near future — we expect that there will be mix of DCTCP and legacy TCP traffic (from the external WAN or between servers that haven't been upgraded) in most environments. These will need to be isolated to avoid unfairness to the TCP flows. This can be done effectively with existing Class of Service (priority) capabilities in data center switches, but further investigation is needed to determine best practices, configurations, etc.

Our approach has been to drive buffer occupancies in data center switches as low as possible in order to reduce latency and increase burst tolerance. There are other points in the design space that are also worth exploring. Specifically, as recent work [164, 77, 156, 11] has shown, transport mechanisms that partition bandwidth in accordance with detailed knowledge of individual flow requirements (e.g., the flow sizes or deadlines) can achieve better performance. We believe this direction holds a lot of promise, but will need to pay special attention to deployability considerations to ultimately bear fruition.

One of the important challenges in cloud data centers is to provide performance guarantees for the tenants that share the infrastructure [118]. The lack of performance guarantees,

particularly for network resources, leads to unpredictable application performance and limits the cloud's applicability, ultimately hurting revenue [25, 86]. This has spurred a number of proposals [144, 69, 25, 132, 87] for network performance isolation in cloud data centers. Most work thus far has focused on making *bandwidth* guarantees. It would be interesting to consider applying our techniques for controlling latency, especially those of HULL, in this setting to provide guarantees for network latency (in addition to bandwidth guarantees).

Finally, it is worthwhile to explore whether the techniques of this dissertation have use beyond data centers. In particular, in recent years, the *bufferbloat* problem in the Internet — the existence of "excessively large and frequently full" [58] buffers that cause high latency and jitter — has been getting widespread attention [12, 1, 89, 29]. Active queue management solutions such as DCTCP's aggressive ECN-based congestion signaling seem a natural fit for this problem and are being considered by the Internet Engineering Task Force (IETF) [39].

Appendix A

Proof of Theorem 3.1

The proof proceeds by computing the Jacobian of the Poincaré map for system (3.14) at its fixed point x_α^* and requiring it to be stable. Define $\mathcal{B}_\epsilon(x_0^*) \triangleq \{\xi \in \mathbb{R}^3 : ||\xi - x_0^*|| \leq \epsilon\}$. We need the following lemma that states that starting from a neighborhood of x_α^*, successive traversals of the switching plane occur and thereby the Poincaré map is well-defined.

Lemma A.1. *Assume that $F(\mathrm{x}, u)$ is infinitely differentiable with respect to* x*. For any given $\delta > 0$, there exists $\epsilon_\delta > 0$ such that the trajectory of $x(t)$ starting from any traversing point in $B_{\epsilon_\delta}(x_\alpha^*) \cap S$ (resp. $B_{\epsilon_\delta}(x_\beta^*) \cap S$) at time t will traverse the plane S again and the traversing time $t+1+t_{trav}$ satisfies $h_\alpha - \delta < t_{trav} < h_\alpha + \delta$ (resp. $h_\beta - \delta < t_{trav} < h_\beta + \delta$).*

Intuitively, this lemma is true because of continuity and its proof follows the same argument as the proof of Lemma 3.3.3 by Wang *et al.* [162]. In the next lemma, we characterize the trajectory of $x(t)$ starting from a region close to x_α^*.

Lemma A.2. *There exists $\epsilon > 0$ such that any trajectory of the system described by* (3.14)*, starting from $x_\alpha = x_\alpha^* + \delta x_\alpha \in B_\epsilon(x_\alpha^*) \cap S$ will intersect and traverse S, and the traversing point $x(t_1)$ satisfies:*

$$x(t_1) - x_\beta^* = Z_2 \delta x_\alpha + O(\delta^2).$$

Proof. From lemma A.1, there exists some $\epsilon > 0$ such that the trajectory of $x(t)$ starting from $B_\epsilon(x_\alpha^*) \cap S$ will traverse S at some instant $t_1 = 1 + h_\alpha + \delta h_\alpha$. Define $\delta x(t) =$

$x(t) - x^*(t)$. Then,

$$\begin{aligned} \dot{\delta x}(t) &= F(x(t), u(t-1)) - F(x^*(t), u(t-1)) \\ &= F(x^*(t) + \delta x(t), u(t-1)) - F(x^*(t), u(t-1)) \\ &= J_F(x^*(t), u(t-1))\ \delta x(t) + O(\delta^2). \end{aligned} \tag{A.1}$$

where we have used the series expansion of $F(\mathrm{x}^* + \delta \mathrm{x}, u)$ around (x^*, u). Since $\delta x(0) = \delta x_\alpha$, from (A.1) we get:

$$\begin{aligned} & x(1 + h_\alpha + \delta h_\alpha) - x^*(1 + h_\alpha + \delta h_\alpha) \\ &= \exp\left(\int_0^{1+h_\alpha+\delta h_\alpha} {\scriptstyle J_F(x^*(s), u(s-1)) ds}\right) \delta x_\alpha + O(\delta^2) \\ &= \exp\left(\int_0^{1+h_\alpha} {\scriptstyle J_F(x^*(s), u(s-1)) ds}\right) \delta x_\alpha + O\left({\scriptstyle \delta h_\alpha \delta x_\alpha}\right) + O(\delta^2) \\ &= \exp\left(\int_0^{1+h_\alpha} {\scriptstyle J_F(x^*(s), u(s-1)) ds}\right) \delta x_\alpha + O(\delta^2) \end{aligned} \tag{A.2}$$

Furthermore, making a series expansion in δh_α, we get:

$$x^*(1 + h_\alpha + \delta h_\alpha) - x^*_\beta = F(x^*_\beta, u_\alpha)\ \delta h_\alpha + O(\delta^2), \tag{A.3}$$

where we use the fact that $x^*(1 + h_\alpha) = x^*_\beta$. Using (A.2) and (A.3), we have:

$$x(1 + h_\alpha + \delta h_\alpha) - x^*_\beta = \exp\left(\int_0^{1+h_\alpha} {\scriptstyle J_F(x^*(s), u(s-1)) ds}\right) \delta x_\alpha + F(x^*_\beta, u_\alpha)\delta h_\alpha + O(\delta^2). \tag{A.4}$$

Since $x(1 + h_\alpha + \delta h_\alpha)$ and x^*_β are on the switching plane S, we have $cx(1 + h_\alpha + \delta h_\alpha) = cx^*_\beta = 0$. After some manipulations, we get:

$$\delta h_\alpha = -\frac{c\ \exp\left(\int_0^{1+h_\alpha} {\scriptstyle J_F(x^*(s), u(s-1)) ds}\right) \delta x_\alpha}{c\ F(x^*_\beta, u_\alpha)} + O(\delta^2). \tag{A.5}$$

Plugging (A.5) in (A.4) gives the desired result. $\square$

Similarly, if we define t_2 to be the time taken for trajectory of $x(t)$ to traverse S again (at a traversing point close to x^*_α), we have:

$$x(t_2) = x^*_\alpha + Z_1 \delta x_\beta + O(\delta^2)$$

with initial condition $x^*_\beta + \delta x_\beta = x^*_\beta + Z_2 \delta x_\alpha + O(\delta^2)$. Replacing δx_β with $Z_2 \delta x_\alpha + O(\delta^2)$ in the above equality yields:

$$x(t_2) = x^*_\alpha + Z_1 Z_2 \delta x_\alpha + O(\delta^2).$$

Note that $x(t_2) = P(x_\alpha)$ and $x^*_\alpha = P(x^*_\alpha)$. Thus,

$$P(x_\alpha) = P(x^*_\alpha) + Z_1 Z_2 \delta x_\alpha + O(\delta^2).$$

Consequently, the Jacobian of the Poincaré map at x^*_α is $Z_1 Z_2$. Therefore, the Poincaré map is locally stable at x^*_α (respectively, the limit cycle x^* is locally stable) if and only if all the eigenvalues of $Z_1 Z_2$ are inside the unit circle; i.e., $\rho(Z_1 Z_2) < 1$. $\square$

Appendix B

Proof of Proposition 3.1

Recall that the second phase of convergence in the proof of Theorem 3.2 begins with $T_k \geq T_{P1}$ (Section 3.3.4). Since the α_i are all close their mean $\bar{\alpha}$ after Phase I, it suffices to study the evolution of $\bar{\alpha}$. This is very useful, as it reduces an N-dimensional problem into a 1-dimensional one.

Define the function $f : [0, \infty] \longrightarrow [0, \infty]$:

$$f(\alpha) = e^{-g(1+W^*\alpha/2)}(e^g - 1 + \alpha).$$

Using (3.19) and (3.17) we have:

$$\begin{aligned}\bar{\alpha}(T_{k+1}) &= e^{-g\Delta T_k}(e^g - 1 + \bar{\alpha}(T_k)) \\ &= f(\bar{\alpha}(T_k))e^{g/2N\sum_i W_i(T_k)(\bar{\alpha}(T_k)-\alpha_i(T_k))}.\end{aligned}$$

Invoking (3.28), we can bound the exponent term

$$\Big|\frac{g}{2N}\sum_i W_i(T_k)(\bar{\alpha}(T_k) - \alpha_i(T_k))\Big| \leq \frac{g}{4}e^{-g(T_k - T_{P1})},$$

to get:

$$\bar{\alpha}(T_{k+1}) = f(\bar{\alpha}(T_k))(1 + \delta_k), \tag{B.1}$$

where $|\delta_k| \leq \epsilon_k \triangleq ge^{-g(T_k - T_{P1})}/2$. Since the δ_k vanish exponentially fast, (B.1) implies

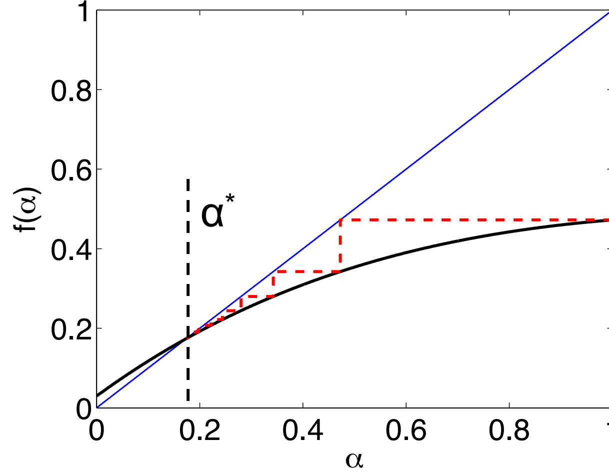


Figure B.1: Example: convergence of $\bar{\alpha}(n+1) = f(\bar{\alpha}(n))$. The sequence starts with $\bar{\alpha}_0 = 1$ and follows the dashed line to $\alpha^* \approx 0.2$.

that $\bar{\alpha}$ essentially evolves according to the iteration $\bar{\alpha}(T_{k+1}) \approx f(\bar{\alpha}(T_k))$, as illustrated in the example in Figure B.1. Before proceeding, we state a few useful properties of $f(\cdot)$.

Lemma B.1. *For parameters satisfying* (3.20)*:*

(i) *f maps* $[0,1]$ *to* $[0,1]$*; i.e.,* $f([0,1]) \subseteq [0,1]$.

(ii) f *has a global maximum at* $\alpha_{max} = 2/(gW^*) - (e^g - 1)$, *and is strictly increasing for* $\alpha < \alpha_{max}$, *and strictly decreasing for* $\alpha > \alpha_{max}$.

(iii) $f(\alpha_{max}) < 4\alpha_{max}/5$.

(iv) f *has a unique fixed point* $0 < \alpha^* < \min(\alpha_{max}, 1)$.

(v) *for any* $0 \leq \alpha \leq \alpha_{max}$,

$$|f(\alpha) - \alpha^*| \leq e^{-g(1+W^*\alpha/2)}|\alpha - \alpha^*|.$$

Proof. (i), (ii), and (iii) follow after some simple algebraic manipulations.

Proof of (iv): Since $f(0) > 0$, $f(1) < 1$, and $f(\alpha_{max}) < \alpha_{max}$, the continuity of f implies the existence of a fixed point in $(0, \min(\alpha_{max}, 1))$. The derivative of f is given by:

$$f'(\alpha) = -\frac{gW^*}{2} f(\alpha) + e^{-g(1+W^*\alpha/2)}.$$

Because $f(\alpha) \geq 0$, this implies:

$$|f'(\alpha)| \leq \max(\frac{gW^*}{2} f(\alpha_{max}), e^{-g}) \leq \max(4/5, e^{-g}) < 1.$$

Thus, f is a contraction, and the uniqueness of the fixed point follows by the contraction mapping principle [100].

Proof of (v): By the Mean-Value Theorem:

$$f(\alpha) - \alpha^* = f'(c)(\alpha - \alpha^*),$$

for some $c \in (0, \alpha_{max})$. Note that $f'(c) > 0$ by part (ii). Assume $\alpha > \alpha^*$. Then $f(\alpha) > \alpha^*$, and we have:

$$\alpha^* = f(\alpha^*) > e^{-g(1+W^*\alpha/2)}(e^g - 1 + \alpha^*),$$
$$\Longrightarrow f(\alpha) - \alpha^* < e^{-g(1+W^*\alpha/2)}(\alpha - \alpha^*).$$

An identical argument for $\alpha < \alpha^*$ completes the proof. □

Lemma B.2. *Let* $m \triangleq \inf\{k | T_k \geq T_{P1}\}$*. Then for* $k \geq m + 1$*:*

$$|\bar{\alpha}(T_{k+1}) - \alpha^*| \leq e^{-g\Delta T_k} |\bar{\alpha}(T_k) - \alpha^*| + 2\epsilon_k. \quad \text{(B.2)}$$

Proof. Using (B.1) and Lemma B.1 (iii), for $k \geq m + 1$:

$$\bar{\alpha}(T_k) < (4\alpha_{max}/5)(1 + g/2) < \alpha_{max}.$$

The last inequality uses $g < 1/2$, which follows from (3.20). Hence, using (B.1) and

Lemma B.1 (i) and (v):

$$\begin{aligned}|\bar{\alpha}(T_{k+1}) - \alpha^*| &\leq |f(\bar{\alpha}(T_k)) - \alpha^*| + f(\bar{\alpha}(T_k))\delta_k \\ &\leq e^{-g(1+W^*\bar{\alpha}(T_k)/2)}|\bar{\alpha}(T_k) - \alpha^*| + \epsilon_k.\end{aligned}$$

The result follows by noting that:

$$\begin{aligned}e^{-g(1+W^*\bar{\alpha}(T_k)/2)} &= e^{-g\Delta T_k} e^{g/2N \sum_i W_i(T_k)(\alpha_i(T_k) - \bar{\alpha}(T_k))} \\ &\leq e^{-g\Delta T_k}(1 + \epsilon_k).\end{aligned}$$

□

We are now ready to prove Proposition 3.1. Let m be defined as in Lemma B.2. Iterating (B.2) backwards from $n \geq m + 1$:

$$\begin{aligned}|\bar{\alpha}(T_n) - \alpha^*| &\leq e^{-g(T_n - T_{m+1})}|\bar{\alpha}(T_{m+1}) - \alpha^*| + 2 \sum_{k=m+1}^{n-1} \epsilon_k e^{-g(T_n - T_{k+1})}, \\ &\leq \Big(e^{g(T_{m+1} - T_{P1})} + g \sum_{k=m+1}^{n-1} e^{g(T_{k+1} - T_k)}\Big) e^{-g(T_n - T_{P1})}.\end{aligned}$$

Using $\Delta T_k \leq 1 + W^*/2$ and Lemma 3.1, we have

$$|\alpha_i(T_n) - \alpha^*| \leq \Big(1 + e^{2g(1+W^*/2)} + g e^{g(1+W^*/2)} n\Big) e^{-g(T_n - T_{P1})},$$

for all $1 \leq i \leq N$. It is easy to show that for $n \geq 2$:

$$1 + e^{2g(1+W^*/2)} + g e^{g(1+W^*/2)} n \leq e^{2g(1+W^*/2)} n,$$

and the result (equation (3.29)) follows for $n \geq m + 1$. For $1 \leq n \leq m$, the result is trivial because the right side of (3.29) is larger than 1.

Appendix C

Bounds on α^*

Theorem 3.2 showed that $\alpha_i(T_n)$ converge to the constant α^* which solves:

$$\alpha^* = e^{-g(1+W^*\alpha^*/2)}(e^g - 1 + \alpha^*). \tag{C.1}$$

Here, we provide bounds on the value of α^*. The significance of α^* is that it represents the fraction of packets that are marked in equilibrium when N long-lived DCTCP flows contend at a bottleneck link.

Theorem C.1. *Recall that $W^* = (Cd + K)/N$ is the equilibrium per-flow window size. If $W^* \geq 2$, then α^* satisfies:*

$$\frac{1}{2}\sqrt{\frac{2}{W^*}} - g < \alpha^* < \sqrt{\frac{2}{W^*}} + g. \tag{C.2}$$

Theorem C.1 implies that for very small g, $\alpha^* \approx c/\sqrt{W^*}$ for a constant $1/\sqrt{2} < c < \sqrt{2}$. As g increases, the $\alpha_i(t)$ fluctuate more widely resulting in weaker bounds on α^*. To stop the fluctuations from becoming large compared to the "ideal" value, $c/\sqrt{W^*}$, an upper limit is required for g. This provides another justification for choosing $g \lessapprox 1/\sqrt{Cd + K}$, as suggested using the simple Sawtooth model in Section 2.2.3 and using the fluid model in Section 3.2.3.

Proof. Assume that the Hybrid Model system (Section 3.3.1) starts with initial condition $W_i(0) = W^*$ and $\alpha_i(0) = \alpha^*$. Its easy to see that starting from this initial condition, the

marking process $p(t)$ is a periodic square wave with period $\Delta T = 1 + W^*\alpha^*/2$ and duty cycle $1/\Delta T$. Also, $W_i(t)$ and $\alpha_i(t)$ do not depend on i, and so dropping the subscript, the common $\alpha(t)$ process evolves according to:

$$\frac{d\alpha}{dt} = g(p(t) - \alpha(t)). \tag{C.3}$$

The low pass filter described by (C.3), has frequency response $H(s) = g/(s+g)$ and impulse response $h(t) = ge^{-gt}$. Let α_{dc} denote the DC value of the $\alpha(t)$ process. Since the DC value of $p(t)$ is equal to its duty cycle, $1/\Delta T$, $\alpha_{dc} = H(0)/\Delta T = 1/\Delta T$. Furthermore,

$$\begin{aligned}
&\alpha(t) - \alpha_{dc} \\
&= (p(t) - \frac{1}{\Delta T}) * (ge^{-gt}) \\
&= \int_0^{min\{1,t\}} (1 - \frac{1}{\Delta T}) ge^{-g(t-\tau)} d\tau + \int_{min\{1,t\}}^{t} (-\frac{1}{\Delta T}) ge^{-g(t-\tau)} d\tau,
\end{aligned}$$

where $*$ is the convolution operator. Using this, we have the following bounds:

$$\begin{aligned}
\alpha(t) - \alpha_{dc} &< g\int_0^1 (1 - \frac{1}{\Delta T}) d\tau = g(1 - \frac{1}{\Delta T}), \\
\alpha(t) - \alpha_{dc} &> g\int_1^{\Delta T} (-\frac{1}{\Delta T}) d\tau = -g(1 - \frac{1}{\Delta T}).
\end{aligned}$$

Substituting $\alpha_{dc} = 1/\Delta T$ in the above bounds, and taking t to be a congestion instant (so that $\alpha(t) = \alpha^*$) gives:

$$-g + \frac{1}{\Delta T} \leq \alpha^* \leq g + \frac{1}{\Delta T}.$$

Finally, substituting $\Delta T = 1 + W^*\alpha^*/2$, and using $W^* \geq 2$, the result follows after some basic algebra. □

Appendix D

Characteristic Equation of Linearized QCN Fluid Model

In this section we derive the characteristic equation (5.13) of the linearized QCN fluid model. Taking the Laplace transform of (5.10)–(5.12), we have:

$$\begin{aligned} sR_C(s) - \delta R_C(0) &= -a_1 R_C(s) + a_2 R_T(s) - e^{-s\tau}\left(a_3 R_C(s) + a_4 Q(s)\right), \\ sR_T(s) - \delta R_T(0) &= bR_C(s) - bR_T(s), \\ sQ(s) - \delta Q(0) &= NR_C(s). \end{aligned}$$

Solving for $R_T(s)$ and $Q(s)$ in terms of $R_C(s)$ and plugging this into the first equation, we have:

$$\Big(s + a_1 - \frac{a_2 b}{s+b} + e^{-s\tau}(a_3 + \frac{Na_4}{s})\Big) R_C(s) = \delta R_C(0) + \frac{a_2 \delta R_T(0)}{s+b} - e^{-s\tau}\frac{a_4 \delta Q(0)}{s}.$$

Multiplying both sides by $s(s+b)$, we see that the poles of the system are the roots of:

$$s\left(s^2 + \beta s + \alpha\right) + e^{-s\tau} a_3 (s+b)(s+\gamma) = 0,$$

where $\beta = b + a_1$, and $\alpha = b(a_1 - a_2)$, and $\gamma = Na_4/a_3 = Cp/w$, which can equivalently be written as (5.13).

Appendix E

Stability Analysis of the QCN-AIMD Fluid Model

Recall the QCN-AIMD fluid model given by equations (5.19) and (5.5)–(5.7). We linearize the system around it unique fixed point:

$$\hat{R_C} = \frac{C}{N}, \quad \hat{Q} = Q_{eq} + \frac{\eta(p_s) N R_{AI}}{p_s G_d C}.$$

This leads to the following linear time-delay system:

$$\frac{d\delta R_C}{dt} = -\hat{a}\delta R_C(t) - a_3 \delta R_C(t-\tau) - a_4 \delta Q(t-\tau), \tag{E.1}$$

$$\frac{d\delta Q}{dt} = N \delta R_C(t), \tag{E.2}$$

where $a_3 = G_d w \hat{R_C}$, $a_4 = p_s G_d \hat{R_C}^2$, and $\hat{a} = \eta(p_s) R_{AI}$. After taking the Laplace transform and rearranging, the characteristic equation of (E.1)–(E.2) is found to be:

$$1 + \hat{G}(s) = 0,$$

with

$$\hat{G}(s) = e^{-s\tau} \frac{a_3(s+\gamma)}{s(s+\hat{a})}.$$

We proceed to finding the stability margins of the system by applying the Bode stability criterion to $\hat{G}(s)$. Let $\hat{G}(j\omega) = \hat{r}(\omega)\exp(-j\hat{\theta}(\omega))$ with $\hat{r}(\omega) = |\hat{G}(j\omega)|, \hat{\theta}(\omega) = -\angle\hat{G}(j\omega)$. The 0-dB crossover frequency is found by solving the equation:

$$\hat{r}(\omega) = \frac{a_3\sqrt{\omega^2+\gamma^2}}{\omega\sqrt{\omega^2+\hat{a}^2}} = 1.$$

After squaring both sides, we obtain a quadratic equation in ω^2, which is easily solved to find the 0-dB crossover frequency:

$$\hat{\omega} = \sqrt{\frac{a_3^2-\hat{a}^2}{2} + \sqrt{\frac{(a_3^2-\hat{a}^2)^2}{4} + \gamma^2 a_3^2}}.$$

Therefore, we require $\hat{\theta}(\hat{\omega}) < \pi$ for the system to be stable. But:

$$\begin{aligned}\hat{\theta}(\omega) &= \frac{\pi}{2} + \omega\tau + \arctan(\frac{\omega}{\hat{a}}) - \arctan(\frac{\omega}{\gamma}),\\ &= \pi + \omega\tau - \arctan(\frac{\hat{a}}{\omega}) - \arctan(\frac{\omega}{\gamma}).\end{aligned}$$

Hence, $\hat{\theta}(\hat{\omega}) < \pi$ is equivalent to $\tau < \hat{\tau}$, with:

$$\hat{\tau} = \frac{1}{\hat{\omega}}\left(\arctan(\frac{\hat{\omega}}{\gamma}) + \arctan(\frac{\hat{a}}{\hat{\omega}})\right).$$

This completes the proof of Theorem 5.2. □

Appendix F

Proof of Theorem 5.3

To prove $\tau^* > \hat{\tau}$, we need to show:

$$\hat{\omega}\left(\arctan(\frac{\omega^*}{b}) - \arctan(\frac{\omega^*}{\beta}) + \arctan(\frac{\omega^*}{\gamma})\right) > \omega^*\left(\arctan(\frac{\hat{\omega}}{\gamma}) + \arctan(\frac{\hat{a}}{\hat{\omega}})\right), \quad \text{(F.1)}$$

where ω^* and $\hat{\omega}$ are given by (5.16) and (5.21) respectively. We begin with two simple Lemmas.

Lemma F.1. *If* (5.22) *and* (5.23) *are satisfied, then:*

$$\max\left(\frac{\hat{a}}{\gamma}, \frac{2\hat{a}b\beta}{\gamma a_3 a_1}\right) < 0.1, \quad \text{(F.2)}$$

$$\frac{\hat{a}}{a_1} < 0.4, \quad \text{(F.3)}$$

$$\frac{\hat{a}^2}{\gamma a_3} < 0.02. \quad \text{(F.4)}$$

Proof. By direct substitution for $\hat{a}$, γ, a_3, b, β, and using $a_1 > \eta(p_s)C/(2N)$, it is easy to verify that (F.2) results from (5.22), and (F.3) results from (5.23). For (F.4), note that:

$$\frac{\hat{a}^2}{\gamma a_3} = \frac{NR_{AI}}{C} \times \frac{\eta(p_s)^2 R_{AI}}{p_s G_d C} < 0.2 \times 0.1 = 0.02.$$

□

Lemma F.2. *Let ω^* and $\hat{\omega}$ be given by* (5.16) *and* (5.21) *respectively. If* (5.22) *and* (5.23) *are satisfied, then:*

$$1 < \frac{\omega^*}{\hat{\omega}} < 1 + \epsilon,$$

where

$$\epsilon = \frac{3\hat{a}^2}{4\hat{\omega}^2} < 0.015.$$

Proof. Using (5.21) we have:

$$\begin{aligned}\hat{\omega}^2 &= \frac{a_3^2 - \hat{a}^2}{2} + \sqrt{\frac{a_3^2}{4} + \gamma^2 a_3^2 + \frac{\hat{a}^4 - 2\hat{a}^2 a_3^2}{4}} \\ &> \frac{a_3^2 - \hat{a}^2}{2} + \sqrt{\frac{a_3^4}{4} + \gamma^2 a_3^2}\sqrt{1 - \frac{2\hat{a}^2 a_3^2}{a_3^4 + 4\gamma^2 a_3^2}}.\end{aligned}$$

Therefore, since $\sqrt{1-x} > 1 - x$ for $0 < x < 1$, we have:

$$\begin{aligned}\hat{\omega}^2 &> \frac{a_3^2 - \hat{a}^2}{2} + \sqrt{\frac{a_3^4}{4} + \gamma^2 a_3^2}\left(1 - \frac{2\hat{a}^2 a_3^2}{a_3^4 + 4\gamma^2 a_3^2}\right) \\ &= \frac{a_3^2}{2} + \sqrt{\frac{a_3^4}{4} + \gamma^2 a_3^2} - \frac{\hat{a}^2 a_3^2}{\sqrt{a_3^4 + 4\gamma^2 a_3^2}} - \frac{\hat{a}^2}{2} \\ &> \omega^{*2} - \frac{3\hat{a}^2}{2}.\end{aligned}$$

Hence, using $\sqrt{1+x} < 1 + x/2$:

$$\frac{\omega^*}{\hat{\omega}} < \sqrt{1 + \frac{3\hat{a}^2}{2\hat{\omega}^2}} < 1 + \frac{3\hat{a}^2}{4\hat{\omega}^2}.$$

Let $\epsilon \triangleq 3\hat{a}^2/(4\hat{\omega}^2)$. Since $\hat{\omega}^2 > \gamma a_3$, using (F.4), we have $\epsilon < 0.015$. □

We are now ready to prove the Theorem. Rearranging (F.1) and using Lemma F.2, it is enough to show:

$$\arctan(\frac{\omega^*}{b}) - \arctan(\frac{\omega^*}{\beta}) + \arctan(\frac{\omega^*}{\gamma}) > (1+\epsilon)\left(\arctan(\frac{\hat{\omega}}{\gamma}) + \arctan(\frac{\hat{a}}{\hat{\omega}})\right),$$

and since $\omega^* > \hat{\omega}$, it is enough to show:

$$\arctan(\frac{\omega^*}{b}) - \arctan(\frac{\omega^*}{\beta}) > \epsilon \arctan(\frac{\hat{\omega}}{\gamma}) + (1+\epsilon)\arctan(\frac{\hat{a}}{\hat{\omega}}).$$

Now, since $\arctan(x) \leq x$ for $x \geq 0$, and $\hat{a}/\gamma < 0.1$ (F.2):

$$\epsilon \arctan(\frac{\hat{\omega}}{\gamma}) < \frac{\hat{a}^2}{\hat{\omega}^2}\left(\frac{\hat{\omega}}{\gamma}\right) < 0.1\left(\frac{\hat{a}}{\hat{\omega}}\right) < 0.2\arctan(\frac{\hat{a}}{\hat{\omega}}),$$

where the last inequality uses the fact: $x \leq 2\arctan(x)$ for $0 \leq x \leq 1$. Therefore, it is enough to show:

$$\arctan(\frac{\omega^*}{b}) - \arctan(\frac{\omega^*}{\beta}) > 2\arctan(\frac{\hat{a}}{\hat{\omega}}).$$

After applying $\tan(\cdot)$ to both sides of this inequality and simplifying using $\omega^* > \omega$, we are left with:

$$a_1\hat{\omega}^2 - 2\hat{a}{\omega^*}^2 > a_1\hat{a}^2 + 2\hat{a}b\beta,$$

which can be further simplified using $\omega^* < (1+\epsilon)\hat{\omega}$ and $\hat{\omega} > \gamma a_3$ to arrive at:

$$1 > \frac{\hat{a}^2}{\gamma a_3} + \frac{2\hat{a}b\beta}{\gamma a_3 a_1} + \frac{2\hat{a}}{a_1}(1+\epsilon)^2.$$

This inequality holds because of (F.2)–(F.4) and $\epsilon < 0.015$. $\square$

Bibliography

[1] BufferBloat: What's Wrong with the Internet? *Queue*, 9(12):10:10–10:20, Dec. 2011.

[2] Cisco Global Cloud Index: Forecast and Methodology, 2011–2016. `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns1175/Cloud_Index_White_Paper.pdf`.

[3] Cisco Visual Networking Index: Forecast and Methodology, 2011–2016. `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf`.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.

[5] I. Aldridge. *High-Frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems*. Wiley Trading. Wiley, 2009.

[6] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar, and M. Seaman. Data center transport mechanisms: Congestion control theory and IEEE standardization. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, pages 1270 –1277, sept. 2008.

[7] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[8] M. Alizadeh, A. Javanmard, and B. Prabhakar. Analysis of DCTCP: stability, convergence, and fairness. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '11, pages 73–84, New York, NY, USA, 2011. ACM.

[9] M. Alizadeh, A. Kabbani, B. Atikoglu, and B. Prabhakar. Stability analysis of QCN: the averaging principle. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '11, pages 49–60, New York, NY, USA, 2011. ACM.

[10] 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 *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12, Berkeley, CA, USA, 2012. USENIX Association.

[11] M. Alizadeh, S. Yang, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. Deconstructing datacenter packet transport. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 133–138, New York, NY, USA, 2012. ACM.

[12] M. Allman. Comments on bufferbloat. *SIGCOMM Comput. Commun. Rev.*, 43(1):30–37, Jan. 2012.

[13] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), September 2009.

[14] E. Altman, C. Barakat, E. Laborde, P. Brown, and D. Collange. Fairness analysis of TCP/IP. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, volume 1, pages 61 –66 vol.1, 2000.

[15] E. Altman, T. Jiménez, and R. Núñez Queija. Analysis of two competing TCP/IP connections. *Perform. Eval.*, 49(1-4):43–55, Sept. 2002.

[16] Amazon. `http://www.amazon.com/`.

[17] Amazon Elastic Compute Cloud (Amazon EC2). `http://aws.amazon.com/ec2/`.

[18] A. Amit, S. Savage, and T. Anderson. Understanding the Performance of TCP Pacing. In *Proc. of INFOCOM*, 2000.

[19] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, pages 281–292, New York, NY, USA, 2004. ACM.

[20] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.

[21] I. T. Association. *InfiniBand architecture specification: release 1.0*. InfiniBand Trade Association, 2000.

[22] K. J. Astrom, G. C. Goodwin, and P. R. Kumar, editors. *Adaptive Control, Filtering, and Signal Processing*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1995.

[23] S. Athuraliya, S. Low, V. Li, and Q. Yin. REM: active queue management. *Network, IEEE*, 15(3):48 –53, May 2001.

[24] F. Baccelli and D. Hong. AIMD, fairness and fractal scaling of TCP traffic. In *INFOCOM*, 2002.

[25] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 242–253, New York, NY, USA, 2011. ACM.

[26] N. Beheshti, Y. Ganjali, M. Ghobadi, N. McKeown, and G. Salmon. Experimental study of router buffer sizing. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, IMC '08, pages 197–210, New York, NY, USA, 2008. ACM.

[27] A. Benner. *Fibre Channel for SANs*. Professional Telecom Series. McGraw-Hill, 2001.

[28] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99, Jan. 2010.

[29] Bufferbloat. `http://www.bufferbloat.net/`.

[30] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview, 1994.

[31] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications architectures, protocols and applications*, SIGCOMM '94, pages 24–35, New York, NY, USA, 1994. ACM.

[32] P. Brown. Resource sharing of TCP connections with different round trip times . In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1734 –1741 vol.3, mar 2000.

[33] H. Cai, D. Y. Eun, S. Ha, I. Rhee, and L. Xu. Stochastic Ordering for Internet Congestion Control and its Applications. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 910 –918, May 2007.

[34] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian. Fabric: a retrospective on evolving SDN. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 85–90, New York, NY, USA, 2012. ACM.

[35] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding TCP incast throughput collapse in datacenter networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, WREN '09, pages 73–82, New York, NY, USA, 2009. ACM.

[36] D. Cohen, T. Talpey, A. Kanevsky, U. Cummings, M. Krause, R. Recio, D. Crupnicoff, L. Dickman, and P. Grun. Remote Direct Memory Access over the Converged

Enhanced Ethernet Fabric: Evaluating the Options. In *Proceedings of the 2009 17th IEEE Symposium on High Performance Interconnects*, HOTI '09, pages 123–130, Washington, DC, USA, 2009. IEEE Computer Society.

[37] Data Center Bridging Task Group. `http://www.ieee802.org/1/pages/dcbridges.html`.

[38] DCTCP Linux kernel patch. `http://www.stanford.edu/~alizade/Site/DCTCP.html`.

[39] DCTCP & CoDel: The Best is the Friend of the Good. `http://www.ietf.org/proceedings/84/slides/slides-84-tsvarea-3`.

[40] Windows Server 2012 – Data Center Transmission Control Protocol (DCTCP). `http://technet.microsoft.com/en-us/library/hh997028.aspx`.

[41] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Symposium proceedings on Communications architectures & protocols*, SIGCOMM '89, pages 1–12, New York, NY, USA, 1989. ACM.

[42] P. Devkota and A. L. N. Reddy. Performance of Quantized Congestion Notification in TCP Incast Scenarios of Data Centers. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '10, pages 235–243, Washington, DC, USA, 2010. IEEE Computer Society.

[43] D.Lacamera. TCP Pacing Linux Implementation. `http://danielinux.net/index.php/TCP_Pacing`.

[44] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown. Processor sharing flows in the internet. In *Proceedings of the 13th international conference on Quality of Service*, IWQoS'05, pages 271–285, Berlin, Heidelberg, 2005. Springer-Verlag.

[45] eBay. `http://www.ebay.com/`.

[46] Facebook. `http://www.facebook.com/`.

[47] Facebooks $1 Billion Data Center Network. `http://www.datacenterknowledge.com/archives/2012/02/02/facebooks-1-billion-data-center-network/`.

[48] W.-c. Feng, K. G. Shin, D. D. Kandlur, and D. Saha. The BLUE active queue management algorithms. *IEEE/ACM Trans. Netw.*, August 2002.

[49] S. Floyd. RED: Discussions of setting parameters. `http://www.icir.org/floyd/REDparameters.txt`.

[50] S. Floyd. HighSpeed TCP for Large Congestion Windows, 2003.

[51] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, Aug. 1993.

[52] S. Floyd and V. Jacobson. The synchronization of periodic routing messages. *IEEE/ACM Trans. Netw.*, 2(2):122–136, Apr. 1994.

[53] Fulcrum FM4000 Series Ethernet Switch. `http://www.fulcrummicro.com/documents/products/FM4000_Product_Brief.pdf`.

[54] G. F. Franklin, D. J. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 4th edition, 2001.

[55] G. F. Franklin, M. L. Workman, and D. Powell. *Digital Control of Dynamic Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1997.

[56] S. B. Fred, T. Bonald, A. Proutiere, G. Régnié, and J. W. Roberts. Statistical bandwidth sharing: a study of congestion at flow level. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM ’01, pages 111–122, New York, NY, USA, 2001. ACM.

[57] Google Invests $600 Million to Expand in South Carolina. `http://www.datacenterknowledge.com/archives/2013/01/18/google-puts-another-600m-in-south-carolina/`.

[58] J. Gettys and K. Nichols. Bufferbloat: dark buffers in the internet. *Commun. ACM*, 55(1):57–65, Jan. 2012.

[59] G. Gibb, J. W. Lockwood, J. Naous, P. Hartke, and N. McKeown. NetFPGA — An open platform for teaching how to build gigabit-rate network switches and routers. *Education, IEEE Transactions on*, 51(3):364–369, 2008.

[60] R. J. Gibbens and F. Kelly. Distributed connection acceptance control for a connectionless network. In *Proc. of the International Teletraffic Congress*, pages 941–952. Elsevier, 1999.

[61] Google+. `http://plus.google.com/`.

[62] Google App Engine. `https://appengine.google.com/`.

[63] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, Dec. 2008.

[64] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM.

[65] Arista 7100 Series Switches. `http://www.aristanetworks.com/en/products/7100series`.

[66] Y. Gu, D. F. Towsley, C. V. Hollot, and H. Zhang. Congestion Control for Small Buffer High Speed Networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 6-12 May 2007, Anchorage, Alaska, USA*, pages 1037–1045. IEEE, 2007.

[67] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[68] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: a high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 63–74, New York, NY, USA, 2009. ACM.

[69] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: a data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 15:1–15:12, New York, NY, USA, 2010. ACM.

[70] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, July 2008.

[71] G. Hager and G. Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2010.

[72] J. Hamilton. On designing and deploying internet-scale services. In *Proceedings of the 21st conference on Large Installation System Administration Conference*, LISA'07, pages 18:1–18:12, Berkeley, CA, USA, 2007. USENIX Association.

[73] Y. Hayashi, H. Itsumi, and M. Yamamoto. Improving Fairness of Quantized Congestion Notification for Data Center Ethernet Networks. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems Workshops*, ICDCSW '11, pages 20–25, Washington, DC, USA, 2011. IEEE Computer Society.

[74] C. Hollot, V. Misra, D. Towsley, and W. Gong. Analysis and design of controllers for AQM routers supporting TCP flows. *Automatic Control, IEEE Transactions on*, 47(6):945 –959, jun 2002.

[75] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong. A control theoretic analysis of RED. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1510 – 1519 vol.3, 2001.

[76] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong. On designing improved controllers for AQM routers supporting TCP flows. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1726 –1734 vol.3, 2001.

[77] C.-Y. Hong, M. Caesar, and P. B. Godfrey. Finishing flows quickly with preemptive scheduling. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 127–138, New York, NY, USA, 2012. ACM.

[78] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm, 2000.

[79] IEEE 802.1Qau. `http://www.ieee802.org/1/pages/802.1au.html`.

[80] InfiniBand Trade Association. `http://www.infinibandta.org`.

[81] iTunes. `http://www.apple.com/itunes/`.

[82] S. Iyer. *Load balancing and parallelism for the internet*. PhD thesis, Stanford, CA, USA, 2008. AAI3332842.

[83] S. Iyer, R. Kompella, and N. McKeown. Designing Packet Buffers for Router Linecards. *Networking, IEEE/ACM Transactions on*, 16(3):705 –717, june 2008.

[84] V. Jacobson. Congestion avoidance and control. In *Symposium proceedings on Communications architectures and protocols*, SIGCOMM '88, pages 314–329, New York, NY, USA, 1988. ACM.

[85] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. Technical report, Digital Equipment Corporation, Sept. 1984.

[86] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, and C. Kim. Eyeq: practical network performance isolation for the multi-tenant cloud. In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, HotCloud'12, Berkeley, CA, USA, 2012. USENIX Association.

[87] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, C. Kim, and A. Greenberg. EyeQ: Practical Network Performance Isolation at the Edge. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, NSDI'13, Berkeley, CA, USA, 2013. USENIX Association.

[88] H. Jiang and C. Dovrolis. Why is the internet traffic bursty in short time scales? In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '05, pages 241–252, New York, NY, USA, 2005. ACM.

[89] H. Jiang, Z. Liu, Y. Wang, K. Lee, and I. Rhee. Understanding bufferbloat in cellular networks. In *Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design*, CellNet '12, pages 1–6, New York, NY, USA, 2012. ACM.

[90] W. Jiang, F. Ren, R. Shu, and C. Lin. Sliding Mode Congestion Control for data center Ethernet networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 1404–1412, March.

[91] Dimon: JPMorgan Spends $500 Million per Data Center. `http://www.datacenterknowledge.com/archives/2012/08/13/dimon-jpmorgan-spends-500-million-on-its-data-centers/`.

[92] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar. AF-QCN: Approximate Fairness with Quantized Congestion Notification for Multi-tenanted Data Centers. In *Proceedings of the 2010 18th IEEE Symposium on High Performance Interconnects*, HOTI '10, pages 58–65, Washington, DC, USA, 2010. IEEE Computer Society.

[93] A. Kabbani and B. Prabhakar. In Defense of TCP. In *The Future of TCP: Train-wreck or Evolution*, 2008.

[94] H. Kamezawa, M. Nakamura, J. Tamatsukuri, N. Aoshima, M. Inaba, and K. Hiraki. Inter-Layer Coordination for Parallel TCP Streams on Long Fat Pipe Networks. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, SC '04, pages 24–, Washington, DC, USA, 2004. IEEE Computer Society.

[95] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 202–208, New York, NY, USA, 2009. ACM.

[96] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '02, pages 89–102, New York, NY, USA, 2002. ACM.

[97] F. Kelly, G. Raina, and T. Voice. Stability and fairness of explicit congestion control with small buffers. *SIGCOMM Comput. Commun. Rev.*, 38(3):51–62, July 2008.

[98] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability. *The Journal of the Operational Research Society*, 49(3):pp. 237–252, 1998.

[99] H. Khalil. *Nonlinear Systems*. Prentice Hall, 2002.

[100] M. A. Khamsi and W. A. Kirk. *An Introduction to Metric Spaces and Fixed Point Theory*. Wiley, 2001.

[101] R. Kohavi, R. M. Henne, and D. Sommerfield. Practical guide to controlled experiments on the web: listen to your customers not to the hippo. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 959–967, New York, NY, USA, 2007. ACM.

[102] R. R. Kompella, K. Levchenko, A. C. Snoeren, and G. Varghese. Every microsecond counts: tracking fine-grain latencies with a lossy difference aggregator. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 255–266, New York, NY, USA, 2009. ACM.

[103] E. Krevat, V. Vasudevan, A. Phanishayee, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan. On application-level approaches to avoiding TCP throughput collapse in cluster-based storage systems. In *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing '07*, PDSW '07, pages 1–4, New York, NY, USA, 2007. ACM.

[104] S. Kunniyur and R. Srikant. End-to-end congestion control schemes: utility functions, random losses and ECN marks. *IEEE/ACM Trans. Netw.*, 11(5):689–702, Oct. 2003.

[105] S. S. Kunniyur and R. Srikant. An adaptive virtual queue (AVQ) algorithm for active queue management. *IEEE/ACM Trans. Netw.*, 12(2):286–299, Apr. 2004.

[106] T. V. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Trans. Netw.*, 5(3):336–350, June 1997.

[107] Latency Is Everywhere and It Costs You Sales. `http://highscalability.com/blog/2009/7/25/latency-is-everywhere-and-it-costs-you-sales-how-to-crush-it.html`.

[108] Y.-T. Li, D. Leith, and R. N. Shorten. Experimental evaluation of TCP protocols for high-speed networks. *IEEE/ACM Trans. Netw.*, 15(5):1109–1122, Oct. 2007.

[109] D. Lin and R. Morris. Dynamics of random early detection. In *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '97, pages 127–137, New York, NY, USA, 1997. ACM.

[110] Z.-H. Loh, M. Davies, and U. Cummings. Flow and congestion control in switch architectures for multi-hop, memory efficient fabrics, Mar. 29 2011. US Patent 7,916,718.

[111] S. H. Low. A duality model of TCP and queue management algorithms. *IEEE/ACM Trans. Netw.*, 11(4):525–536, Aug. 2003.

[112] S. H. Low and D. E. Lapsley. Optimization flow control I: basic algorithm and convergence. *IEEE/ACM Trans. Netw.*, 7(6):861–874, Dec. 1999.

[113] S. H. Low, F. Paganini, and J. C. Doyle. Internet congestion control. *Control Systems, IEEE*, 22(1):28–43, Feb. 2002.

[114] S. H. Low, L. L. Peterson, and L. Wang. Understanding TCP Vegas: a duality model. *J. ACM*, 49(2):207–235, Mar. 2002.

[115] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[116] V. Misra, W.-B. Gong, and D. Towsley. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '00, pages 151–160, New York, NY, USA, 2000. ACM.

[117] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Trans. Netw.*, 8(5):556–567, Oct. 2000.

[118] J. C. Mogul and L. Popa. What we talk about when we talk about cloud network performance. *SIGCOMM Comput. Commun. Rev.*, 42(5):44–48, Sept. 2012.

[119] Microsoft Building $500 Million Data Center In Virginia. `http://www.crn.com/news/cloud/227200088/microsoft-building-500-million-data-center-in-virginia.htm`.

[120] A. Nayfeh and B. Balachandran. *Applied Nonlinear Dynamics: Analytical, Computational, and Experimental Methods*. Wiley Series in Nonlinear Science. John Wiley & Sons, 2008.

[121] Netflix. `http://www.netflix.com/`.

[122] The NetFPGA Project. `http://netfpga.org`.

[123] Cisco Nexus 3548 Switch. `http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps11541/ps12581/data_sheet_c78-707001.pdf`.

[124] Cisco Nexus 5548P Switch. `http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9670/ps11215/white_paper_c11-622479.pdf`.

[125] K. Nichols, S. Blake, F. Baker, and D. Black. RFC 2474: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. Technical report, IETF, 1998.

[126] The Network Simulator NS-2. `http://www.isi.edu/nsnam/ns/`.

[127] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum. Fast crash recovery in RAMCloud. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 29–41, New York, NY, USA, 2011. ACM.

[128] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, D. Ongaro, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman. The case for RAMCloud. *Commun. ACM*, 54:121–130, July 2011.

[129] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate fairness through differential dropping. *SIGCOMM Comput. Commun. Rev.*, 33(2):23–39, Apr. 2003.

[130] Pandora. `http://www.pandora.com/`.

[131] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Trans. Netw.*, 1(3):344–357, June 1993.

[132] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. FairCloud: sharing the network in cloud computing. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 187–198, New York, NY, USA, 2012. ACM.

[133] I. A. Qazi, T. Znati, and L. L. H. Andrew. Congestion Control using Efficient Explicit Feedback. In *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil*, pages 10–18. IEEE, 2009.

[134] IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks–Amendment 18: Enhanced Transmission Selection for Bandwidth Sharing Between Traffic Classes. *IEEE Std 802.1Qaz-2011 (Amendment to IEEE Std 802.1Q-2011 as amended by IEEE Std 802.1Qbe-2011, IEEE Std 802.1Qbc-2011, and IEEE Std 802.1Qbb-2011)*, pages 1 –110, 30 2011.

[135] IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks–Amendment 17: Priority-based Flow Control. *IEEE Std 802.1Qbb-2011 (Amendment to IEEE Std 802.1Q-2011 as amended by IEEE Std 802.1Qbe-2011 and IEEE Std 802.1Qbc-2011)*, pages 1–40, September 2011.

[136] IEEE Standard for Local and metropolitan area networks– Virtual Bridged Local Area Networks Amendment 13: Congestion Notification. *IEEE Std 802.1Qau-2010 (Amendment to IEEE Std 802.1Q-2005)*, pages 1–119, March 2010.

[137] G. Raina, D. Towsley, and D. Wischik. Part II: control theory for buffer sizing. *SIGCOMM Comput. Commun. Rev.*, 35(3):79–82, July 2005.

[138] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP, 2001.

[139] K. K. Ramakrishnan and R. Jain. A binary feedback scheme for congestion avoidance in computer networks. *ACM Trans. Comput. Syst.*, 8(2):158–181, May 1990.

[140] Mellanox ConnectX-2 EN with RDMA over Ethernet. `http://www.mellanox.com/related-docs/prod_software/ConnectX-2_RDMA_RoCE.pdf`.

[141] J. W. Roberts. A survey on statistical bandwidth sharing. *Comput. Netw.*, 45(3):319–332, June 2004.

[142] J. Rothschild. High Performance at Massive Scale: Lessons Learned at Facebook. `mms://video-jsoe.ucsd.edu/calit2/JeffRothschildFacebook.wmv`.

[143] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open versus closed: a cautionary tale. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*, NSDI'06, Berkeley, CA, USA, 2006. USENIX Association.

[144] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, Berkeley, CA, USA, 2011. USENIX Association.

[145] P. Shivam, P. Wyckoff, and D. Panda. EMP: zero-copy OS-bypass NIC-driven gigabit ethernet message passing. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '01, New York, NY, USA, 2001. ACM.

[146] R. Shorten, D. Leith, J. Foy, and R. Kilduff. Analysis and design of AIMD congestion control algorithms in communication networks. *Automatica*, 41(4):725 – 730, 2005.

[147] R. Shorten, F. Wirth, and D. Leith. A positive systems model of TCP-like congestion control: asymptotic results. *IEEE/ACM Trans. Netw.*, 14(3):616–629, June 2006.

[148] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '95, pages 231–242, New York, NY, USA, 1995. ACM.

[149] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. Wiley Publishing, 8th edition, 2008.

[150] Slacker. `http://www.slacker.com/`.

[151] Spotigfy. `https://www.spotify.com/`.

[152] R. Srikant. *The Mathematics of Internet Congestion Control (Systems and Control: Foundations and Applications)*. SpringerVerlag, 2004.

[153] R. Takano, T. Kudoh, Y. Kodama, M. Matsuda, H. Tezuka, and Y. Ishikawa. Design and evaluation of precise software pacing mechanisms for fast long-distance networks. In *Proc. of PFLDNet*, 2005.

[154] K. Tan and J. Song. A compound TCP approach for high-speed and long distance networks. In *In Proc. IEEE INFOCOM*, 2006.

[155] The tcpdump official website. `http://www.tcpdump.org`.

[156] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter tcp (D2TCP). In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 115–126, New York, NY, USA, 2012. ACM.

[157] U. Varshney, A. Snow, M. McGivern, and C. Howard. Voice over IP. *Commun. ACM*, 45(1):89–96, Jan. 2002.

[158] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller. Safe and effective fine-grained TCP retransmissions for datacenter communication. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 303–314, New York, NY, USA, 2009. ACM.

[159] C. Villamizar and C. Song. High performance TCP in ANSNET. *SIGCOMM Comput. Commun. Rev.*, 24(5):45–60, Oct. 1994.

[160] A. Vishwanath, V. Sivaraman, and M. Thottan. Perspectives on router buffer sizing: recent results and open problems. *SIGCOMM Comput. Commun. Rev.*, 39(2):34–39, Mar. 2009.

[161] V. Visweswaraiah, J. Heidemann, et al. Improving restart of idle TCP connections. Technical report, Technical Report 97-661, University of Southern California, 1997.

[162] Q.-G. Wang, T. H. Lee, and C. Lin. *Relay Feedback: Analysis, Identification, and Control*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[163] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Trans. Netw.*, 14(6):1246–1259, Dec. 2006.

[164] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: meeting deadlines in datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 50–61, New York, NY, USA, 2011. ACM.

[165] Windows Azure: Microsoft's Cloud Platform. `http://www.windowsazure.com/`.

[166] H. Wu, Z. Feng, C. Guo, and Y. Zhang. ICTCP: Incast Congestion Control for TCP in data center networks. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 13:1–13:12, New York, NY, USA, 2010. ACM.

[167] World Wide Web Consortium (W3C). `http://www.w3.org/`.

[168] memcached - a distributed memory object caching system. http://memcached.org.

[169] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman. One more bit is enough. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '05, pages 37–48, New York, NY, USA, 2005. ACM.

[170] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control (BIC) for fast long-distance networks. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2514–2524. IEEE, 2004.

[171] Youtube. http://www.youtube.com/.

[172] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: a new resource ReSerVation Protocol. *Netwrk. Mag. of Global Internetwkg.*, 7(5):8–18, Sept. 1993.

[173] L. Zhang, S. Shenker, and D. D. Clark. Observations on the dynamics of a congestion control algorithm: the effects of two-way traffic. In *Proceedings of the conference on Communications architecture & protocols*, SIGCOMM '91, pages 133–147, New York, NY, USA, 1991. ACM.