Louisiana State University
LSU Digital Commons

LSU Doctoral Dissertations | Graduate School

2014

A study on fairness and latency issues over high speed networks and data center networks

Lin Xue
Louisiana State University and Agricultural and Mechanical College, lxue2@tigers.lsu.edu

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_dissertations

 Part of the Computer Sciences Commons

Recommended Citation
Xue, Lin, "A study on fairness and latency issues over high speed networks and data center networks" (2014). *LSU Doctoral Dissertations*. 419.
https://digitalcommons.lsu.edu/gradschool_dissertations/419

This Dissertation is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Doctoral Dissertations by an authorized graduate school editor of LSU Digital Commons. For more information, please contactgradetd@lsu.edu.

A STUDY ON FAIRNESS AND LATENCY ISSUES OVER HIGH SPEED NETWORKS AND DATA CENTER NETWORKS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

Department of Computer Science and Engineering

by
Lin Xue
B.S., China Agricultural University, 2005
M.S., Beijing University of Posts and Telecommunications, 2008
May 2014

ACKNOWLEDGMENTS

My most sincere gratitude and appreciation go to Dr. Seung-Jong Park as my major professor and committee chair. I thank him for introducing me to this fantastic world of scientific research. I thank him for his help on jumpstart of knowledge, his guidance, encouragement and support throughout my Ph.D. studies. This dissertation would not have been possible without him.

My deepest appreciation also extends to Dr. Costas Busch and Dr. Feng Chen for taking their time to serve as my committee members. Their valuable feedback and comments have improved the quality of this research.

I thank Dr. Mostafa Elseifi from Department of Civil and Environmental Engineering for serving on my committee as Dean's Representative. His suggestions and comments are highly appreciated.

I also would like to thank all my lab mates and colleagues: Dr. Cheng Cui, Dr. Suman Kumar, Praveenkumar Kondikoppa, Chui-hui Chiu, Georgi Stoyanov, Richard Platania, and all people who helped me with testbed setup and paper writing and constantly support me through all the stress. I deeply appreciate their time and help during my PhD study.

Finally, I will be forever grateful to my parents, my wife Yi Liang and all my college friends. Without their support and help, this journey would not have been possible.

This work has been supported in part by the NSF MRI Grant No.0821741, NSF CC-NIE Grant No.1341008, GENI grant, and DEPSCoR project N0014-08-1-0856.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Newly emerging computer networks, such as high speed networks and data center networks, have characteristics of high bandwidth and high burstiness which make it difficult to address issues such as fairness, queuing latency and link utilization. In this study, we first conduct extensive experimental evaluation of the performance of 10Gbps high speed networks. We found inter-protocol unfairness and larger queuing latency are two outstanding issues in high speed networks and data center networks.

There have been several proposals to address fairness and latency issues at switch level via queuing schemes. These queuing schemes have been fairly successful in addressing either fairness issue or large latency but not both at the same time. We propose a new queuing scheme called Approximated-Fair and Controlled-Delay (AFCD) queuing scheme that meets following goals for high speed networks: approximated fairness, controlled low queuing delay, high link utilization and simple implementation. The design of AFCD utilizes a novel synergistic approach by forming an alliance between approximated fair queuing and controlled delay queuing. AFCD maintains very small amount of state information in sending rate estimation of flows and makes drop decision based on a target delay of individual flow.

We then present FaLL, a Fair and Low Latency queuing scheme that meets stringent performance requirements of data center networks: fair share of bandwidth, low queuing latency, high throughput, and ease of deployment. FaLL uses an efficiency module, a fairness module and a target delay based dropping scheme to meet these goals. Through rigorous experiments on real testbed, we show that FaLL outperforms various peer solutions in variety of network conditions over data center networks.

CHAPTER 1
INTRODUCTION

1.1 Background

In this section, we introduce the background of this work: high speed networks and data center networks.

1.1.1 High Speed Networks

Advances in high speed networking technology coincide with the need for network infrastructure development to support scientific computing, distant learning, e-commerce, health, and many other unforeseen future applications. Consequently, 10Gbps high speed networks, such as Internet2 [36], NRL (National LambdaRail) [51], and LONI (Louisiana Optical Network Initiative) [48], have been the first ones that were developed to connect a wide range of academic institutes. Figure 1.1 shows a 10Gbps network infrastructure encompassing the campus network of LSU (Louisiana State University) and MAX (Mid-Atlantic Crossroads). Network operators use Internet2 network stitching [27] to federate all campus networks together, and core routers are responsible for connection across various facilities located in-campus. The 10Gbps high speed network enables resource sharing among different departments at different institutes. As gigabit connectivities are easily available for gigabit-based PCs, servers, data center storage and high performance computing, gigabit networking technology is preferred by many organizations. Therefore, various organizations are increasingly migrating to gigabit links in order to grow their networks to support new applications and traffic types. It is true that the availability of multi-gigabit switches/routers provides the opportunity to build high-performance, high-reliability networks but only if correct design approaches are followed at both hardware and software level.

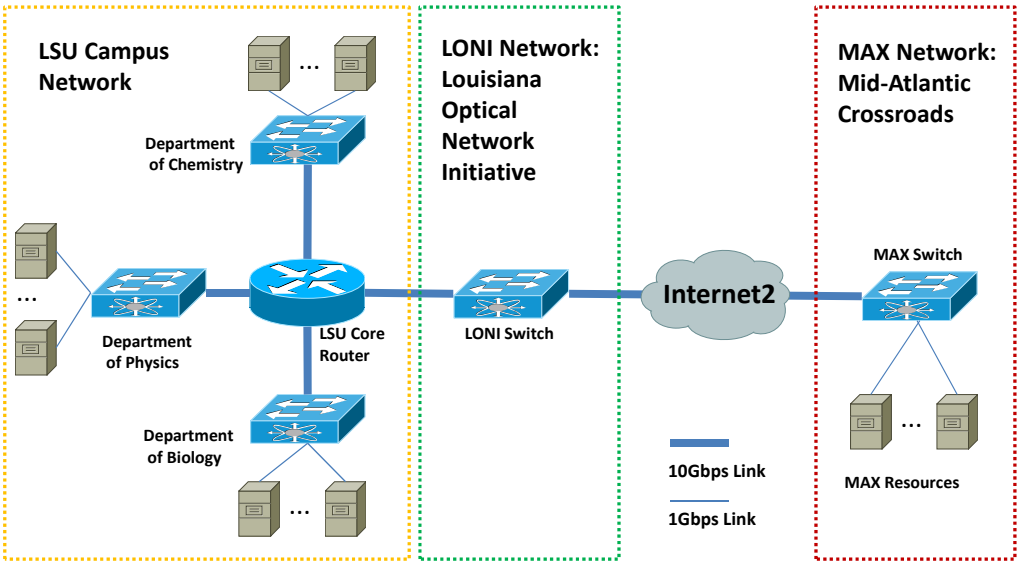


FIGURE 1.1. 10Gbps high speed campus networks connected by Internet2 for large scale scientific computing, distance learning, etc.

To meet the demand of high speed networks, several TCP variants have been proposed. Extensive performance evaluations of these protocols convinced network users to adopt several of these seemingly promising proposals. Furthermore, open sourcing of these protocols makes it flexible for user to select the choice of their protocols. Therefore, current computer networks are ruled by heterogeneous TCP variants[77] such as TCP-SACK, HighSpeed TCP (HSTCP) [22], CUBIC TCP (CUBIC) [31], etc. TCP flows are dominant in Internet2 traffic [37] accounting for around 85% flows while rest being UDP flows with most of the network usages being long-lived bulk data transfer. One of the key design goals of the proposed TCP variants is fair bandwidth sharing with other competing TCP flows. However, our study [76] discloses severe inter-protocol unfairness among heterogeneous long-lived TCP flows in high speed networks where faster TCP flows consume most of the network bandwidth, whereas slow ones starve.

Packet delay is an equally key high speed network performance measure with others being throughput and fairness. Although high speed networks do not have Bufferbloat

problem[28], queuing delay in high speed networks needs careful consideration. For example, in a typical setting of a 10Gbps router [15] in high speed networks, the output buffer size of routers could be up to 89MB, which may create large queuing delay in high speed networks. The large queuing delay may create bad user experience in live concert [35], video streaming, etc, over high speed networks like Internet2. Also, popular applications such as Online shopping, Voice over IP, HDTV, banking, and gaming, require not only high throughput but also low delay. In fact, importance of packet delay is growing with the emergence of a new class of high performance computing applications such as algorithmic trading and various data center applications with soft real time deadlines that demand extremely low end-to-end latency (ranging from microsecond to orders of milliseconds). It is clear that the large latency may result in poor performance of the networks. Therefore, predictable and controllable queuing delay is highly desired in contemporary high speed networks.

Performance of heterogeneous TCP flows depends on router parameters [66], but high bandwidth, high latency, and bursty nature of high speed networks make it a challenging task to design queue management (QM) schemes that ensure minimal latency while maintaining fair share of bandwidth among heterogeneous flows. There have been considerable efforts to address these challenges through various QM proposals [61, 63, 52, 50]. The QM scheme in [61] uses stateful information of the flows to classify the incoming packets, and puts the classified packets into different queues which is not suitable for large scale networks. The QM scheme in [63] is stateless in core layer, but it needs stateful information in edge layer. In [52], the authors proposed a stateless active queue management (AQM) solution, and it achieves approximate fairness for heterogeneous flows. Although the fairness problem has been addressed in these QM schemes, controllable and predictable queuing delay has been overlooked. Recently proposed controlled delay (CoDel) AQM scheme [50] focuses on providing

extremely low queuing delay, and CoDel works well in a wide range of scenarios. While CoDel provides extremely low queuing delay, fairness issue is at bay. Only very recently that there is an attempt to bring fair queuing into CoDel and a variation of CoDel has been implemented in Linux kernel [19] that provides fairness by classifying different flows into different CoDel queues. However, the history of research on classification based QM suggests that any approach requiring huge amount of state information is not practical for large scale networks.

1.1.2 Data Center Networks

Data centers are growing in capacity to host diverse Internet-scale web applications such as social networking, web search, video streaming, advertisement, and so on. Large IT enterprises such as Google, Facebook, Amazon, and Microsoft build their own data center networks to provide large scale online services. As more of society is relying on the growing use of pervasive devices creating new demand for storage and computing facilities, businesses are increasingly moving to wholesale data center model to take advantage of economy of scale [6]. Therefore, data center networks are becoming the cornerstone of always evolving web applications. Web technology trends suggest that low latency, fair share of bandwidth among applications, and high throughput are the major performance requirements that data center networks are expected to meet.

In fact, low latency is becoming a stringent performance requirement for data center networks because of real-time nature of many applications that have become popular, whose performance is critical to service revenue industry. For example, in algorithmic high frequency trading where market data must arrive with minimal latency (of the order of microseconds), financial service providers rely on high speed networks that must provide low latency to carry these computational transactions [67]. Likewise, in retail web services, single page request may require calling more than 100 services

and because these services can be interdependent, low latency is a critical factor for user experience [18]. Many data center applications require task completion within their deadlines. Missing the deadlines may create bad user experience or even failure of the job resulting into lost revenue [17].

Fair bandwidth sharing is another critical performance metric for data center networks. Network traffic from diverse applications are multiplexed at network switches in data center networks. These network traffic may come from different data center tenants[42]. Enforcing fair share of bandwidth among these non-cooperating applications is considered to be an issue [59] that data center networks must address. Also, multi-rooted tree (Figure 1.2), a natural topology of data center is exposed to severe flow unfairness known as TCP outcast [57]. When a large set of flows and a small set of flows come in at two input ports of a switch and come out at one common output port, the small set of flows suffer from throughput starvation significantly. TCP outcast mainly occurs at drop-tail queues with a smaller TCP minimum retransmission timeout, which is a common case in commodity data center networks.

Modern data center networks are complex network systems that contain hundreds to thousands of servers making it challenging to address above issues. DCTCP[2], an ECN (Explicit Congestion Notication) based server side congestion control mechanism has been successful to solve TCP incast, queue buildup, and buffer pressure in data center networks. Although DCTCP has been publicly available as a kernel patch for Linux 2.6.38.3, regular TCP variants are still default options in widely used open sourcing operating systems. To utilize the infrastructure of data center networks, network operators and users may have their own choice of transport protocols on servers. For example, there are heterogeneous regular TCP variants running in the Internet [77]. We argue that queuing mechanism at layer-2 switch is the key to improvement in fairness and latency while achieving high throughput. Figure 1.2 shows a data center

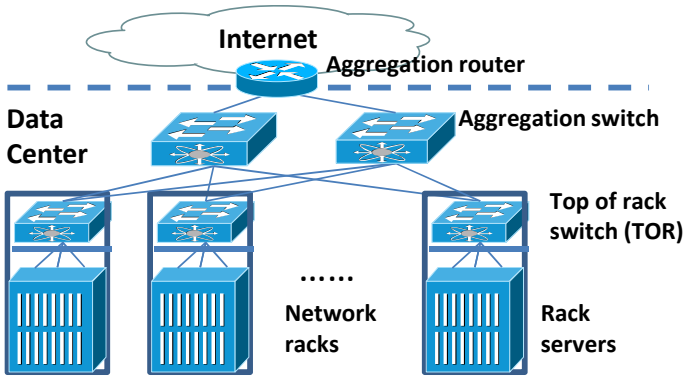


FIGURE 1.2. A data center network showing aggregation and top of rack (TOR) switches

network with switches connected to each other and to servers. Our study is based on the observation that in data center networks, aggregate queuing delays caused by layer-2 switches are a major contributor to in-network latencies [68, 2]. Although a data center usually resides in a single building with very low propagation delay, the high bandwidth and high burstiness still create high queuing delay resulting into high latency in traditional switches [2, 3]. Moreover, since non-cooperating applications and multi-tenants coexist in data center networks, fairness issue is better to be tackled at switch level through queuing scheme [66, 75].

1.2 Summary of Contribution

The main contributions of this study fall into following three parts:

1. Through extensive experimental evaluation of 10Gbps high speed networks, we present for the first time the interplay of queue management schemes, high speed TCP variants, and variety of buffer sizes over a 10Gbps high speed networking environment. We found it difficult to address issues such as fairness, low queuing delay and high link utilization. Current high speed networks carry heterogeneous TCP flows which makes it even more challenging to address these issues. We identify two critical issues high speed networks and data center networks now facing: inter-protocol unfairness and large queuing latency.

2. We propose a new queuing scheme called Approximated-Fair and Controlled-Delay (AFCD) queuing for high speed networks that aims to meet following design goals: approximated fairness, controlled low queuing delay, high link utilization and simple implementation. The design of AFCD utilizes a novel synergistic approach by forming an alliance between approximated fair queuing and controlled delay queuing. It uses very small amount of state information in sending rate estimation of flows and makes drop decision based on a target delay of individual flow. Through experimental evaluation in a 10Gbps high speed networking environment, we show AFCD meets our design goals.

3. We present FaLL, a Fair and Low Latency queuing scheme for data center networks that meets following performance requirements: fair share of bandwidth, low queuing latency, high throughput, and ease of deployment. FaLL uses an efficiency module, a fairness module and a target delay based dropping scheme to meet these goals. FaLL requires very small amount of state information. Through rigorous experiments on real testbed, we show that FaLL outperforms various peer solutions in variety of network conditions over data center networks.

1.3 Outline of Dissertation

The rest of this dissertation is organized as following parts:

Chapter 2 is the performance evaluation of 10Gbps high speed networks. We first setup a real high speed networking testbed. We then present the experimental evaluation of the interrelationship among queue management schemes, high speed TCP variants, and various buffer sizes. Fairness issue and latency issue are found to be the most outstanding issues over 10Gbps high speed networks.

In Chapter 3, we propose the AFCD (Approximated-Fair and Controlled-Delay) queuing for high speed networks. AFCD addresses the fairness issue and the latency

issue at the same time. We explain the detailed design and algorithm of AFCD. Then we conduct experimental evaluation of AFCD in a high speed networking environment.

We present the FaLL (Fair and Low Latency) queuing scheme for data center networks in Chapter 4. FaLL enforces fair share of bandwidth and low queuing latency with minimum state informations. We present the design, algorithm, and experimental evaluation of FaLL.

We draw our conclusion in Chapter 5.

CHAPTER 2
EXPERIMENTAL EVALUATION OF THE PERFORMANCE OF 10GBPS HIGH SPEED NETWORKS

We first conduct experimental evaluation the performance of 10Gbps high speed networks in this chapter. Performance results are presented for important metrics of interest such as link utilization, intro-protocol fairness, inter-protocol fairness, RTT fairness, queuing delay and computational complexity. The results of evaluation suggest two outstanding issues in contemporary networks: fairness and latency.

2.1 An Experimental Study of the Impact of Queue Management Schemes and TCP Variants on 10Gbps High Speed Networks

In this section, we conduct a comprehensive experimental study of the impact of queue management schemes and TCP variants on the performance of 10Gbps high speed networks.

2.1.1 Overview

Over the years, the Drop-tail queue mechanism has been under scrutiny and is found to be unsuitable choice to address issues such as transmission control protocols (TCP) global synchronization, underutilization of link bandwidth, high packet drop rate, high transmission delay, and high queuing delay. To address these issues, Random Early Detection (RED) [24] was proposed as an active queue management (AQM) solution in 1993. Thereafter, several AQMs, such as CHOKe(CHOose and Keep for responsive flows, CHOose and Kill for unresponsive flows) [53], SFB(Stochastic Fair Blue) [21] etc., have been proposed. Inspite of so many AQM proposals, there has been a significant lack of comparative performance evaluation studies on real production

networks to permit any conclusion on the merits of these QM schemes. There are two major consequences for the lack of comparative studies. Firstly, although these AQMs are theoretically superior to Drop-tail, these AQMs are still scarce in production networks; secondly, there is not much support from the testing to the development of future QM schemes. To address the challenges in design and development of QM schemes in future networks, our focus is to compare the performance of competing QM proposals in a systematic and repeatable manner in a 10Gbps high speed network environment.

Simulation and experiment are two main methods to perform such studies. Most of the evaluation research works on AQM schemes rely on simulation models, such as Network Simulator 2 (ns-2) or OPNET modeler. However, a linear increase in bandwidth demands for an exponential increase in CPU-time and memory usage for these discrete simulation methods [46] which makes it very difficult to finish the simulation of high speed links in a reasonable time. Besides, to address issues in design and real network deployment of QM schemes demands a real experimental network environment [25].

Since the cost could be expensive for large businesses or ISP networks to deploy the AQM schemes in the Internet, recent network research [49] tries to check if it is beneficial to deploy the AQM schemes in the core routers. The existing research focuses on the nature of the AQM scheme itself, but overlooks the effect of the AQM scheme on transport layer congestion control. Often, the core of the network is a playground for transport protocols and therefore, it is difficult for a network service provider to address issues related to performance of their network. Due to the feedback nature of AQM schemes, TCPs will behave differently according to its own congestion control algorithm. Especially in production networks (or data centers), the pairing of a TCP variant and an appropriate QM scheme to compliment that TCP is highly

desirable. In other words, it is highly desirable to know the impact of QM schemes on transport protocols. Therefore, in this study, the performance metrics for QM schemes are chosen to be very TCP specific. We also consider the metrics such as memory usage and CPU requirement which we find in a direct correlation with ease of deployment and operational costs.

We choose the popular QM schemes for evaluation, including Drop-tail, RED, CHOKe, and SFB. Among high speed TCP variants, we select CUBIC (CUBIC TCP) [31], HSTCP (HighSpeed TCP) [22], RENO (TCP-Reno), and VEGAS (TCP-Vegas) [10]. It was noted that CUBIC, HSTCP and RENO account for 2/3 of all TCP variants used on the Internet [77]. VEGAS represents the delay-based TCP, and is the only delay-based TCP supported in the current Linux kernel. We present the results in terms of link utilization, intro-protocol fairness, RTT fairness, delay, and computational complexity.

2.1.2 Related Works

To evaluate performance of TCP variants, the authors in [29] and [30] evaluated high speed TCP protocols in a realistic high speed networking environment. The high speed TCP protocols were evaluated against itself in terms of several TCP performance metrics. However, when evaluating all TCP protocols, the authors did not consider the impact of queue management schemes in the router.

In [8], the authors presented a framework to evaluate AQM schemes. Five metrics were chosen to characterize overall network performance of AQM schemes. The authors suggested simulation environments and scenarios, including ns-2 interfaces, traffic models and network topologies. As a continuing work [9], the authors gave simulation based evaluation and comparison of a subset of AQM schemes. Their framework was based on ns-2 simulation which is different than a real-world experiment, especially the 10Gbps high speed networks.

The authors in [14] evaluated new proposed AQM schemes with some specific network scenarios. They proposed a common testbed for the evaluation of AQM schemes which includes a specification of the network topology, link bandwidths and delays, traffic patterns, and metrics for the performance evaluation. Also, the authors realized that AQM schemes need to cooperate closely with TCP. However, they evaluated AQM schemes over regular Internet speed but not 10Gbps speed, and only presented the results of TCP-RENO in their evaluation.

Moreover, the authors in [40] considered router buffer sizing in evaluation of high speed TCP protocol. They conducted an experimental evaluation of CUBIC TCP in small router buffers (e.g. a few tens of packets). Their work highlighted the need for a thorough investigation on the performance of high speed TCP variants with small router buffers for newly emerging high speed networks.

In a recent work [47], the authors found the tradeoff between throughput and fairness in high speed networks. The network performance was evaluated by a model based simulation method, which shows some bottlenecks in evaluating high speed networks. In [74], the authors evaluated the impact of queue management schemes on the performance of TCP over 10Gbps high speed networks. However, the detailed setup of a 10Gbps environment was not unveiled. And some of the research results were not presented such as TCP-VEGAS result, intro-protocol fairness result, memory consumption, etc. In [76], fairness was evaluated thoroughly among heterogeneous high speed TCP variants by using different queue management schemes with varying degrees of buffer sizes, but other metrics have not been fully evaluated yet. The authors in [75] evaluated extensively both fairness and latency of AQM schemes over 10Gbps high speed networks. They proposed a new AQM schemes which works well in terms of fairness and latency over 10Gbps high speed networks.

In this work, we consider the most important metrics which need to be evaluated for the interrelationship between TCP variants and queue management schemes. Performance metrics have been defined in [23] previously. The authors discussed the metrics to be considered to evaluate congestion control mechanisms for the Internet. They brought 11 metrics in total, which could be used for evaluating new or modified transport layer protocols.

2.1.3 Network Performance Consideration

Previous experimental studies treated evaluation of TCP, evaluation of AQMs and effect of router parameters running a particular AQM on TCP separately. Also, these studies do not include 10Gbps bandwidth consideration. In this study, we propose a complementary approach of combining TCP, router parameters and a high speed network of order of 10Gbps.

The following equation summarizes the dynamics of the TCP congestion window $W(t)$ at time t:

$$W(t) = W_{max}\beta + \alpha\frac{t}{RTT} \tag{2.1}$$

W_{max} is the congestion window size just before the last window reduction, RTT is the round trip delay of this flow, and α and β are increase and decrease parameters respectively.

W_{max} depends on router parameters, such as a penalty signal P of queue management schemes , router buffer size Q, and the bottleneck capacity C:

$$W_{max} \leftarrow \frac{QC}{P} \tag{2.2}$$

From the equation above, it is clear that the performance of a network depends on the combination of TCP variants, queue management schemes and router buffer sizes. Our argument is consistent with [64], which concluded that the TCP sending rate depends on both the congestion control algorithms and the queue management

schemes in the links. A real network measurement on high speed networks shows that burstiness increases with bandwidth because packets degenerate into extremely bursty out flows with data rates going beyond the available bandwidth for short periods of time [26]. It is clear that such surges can easily generate severe penalty signals and further degrade the overall network performance.

The penalty signals for different queue management schemes are also different. When the router buffer is full, A Drop-tail queue drops all the packets. A RED queue drops packets early and randomly according to the queue length. CHOKe extends RED to compare random packets and drops packets for fast flows. SFB uses a bloom-filter to determine fast flows and drops packets from fast flows. Buffer size at the routers also impacts network performance. Router buffers cause queuing delay and delay-variance. And in the case of underflow, throughput degradation is observed. Appropriate sizing of buffers has been considered a difficult task for router or switch manufacturers.

Given the importance of router parameters, different high speed TCP variants will differ in performance for the same router parameters. We elaborate on this point by considering the impact of penalty signals on congestion window TCP variants in consideration as below:

1) Traditional TCP's AIMD algorithm has the increase parameter α and decrease parameter β to be 1 and 0.5 respectively.

2) HSTCP's increase parameter α and decrease parameter β are functions of the current window size, namely $\alpha(W)$ and $\beta(W)$. The range of $\alpha(W)$ could be from 1 to 73 packets, and $\beta(W)$ from 0.5 to 0.09.

3) CUBIC updates the congestion window according to a cubic function as shown below:

$$W_{CUBIC} \leftarrow C(t - \sqrt[3]{W_{max}\beta/C})^3 + W_{max} \tag{2.3}$$

where C is a scaling factor, t is the elapsed time since the last window reduction, W_{max} is the window size just before the last window reduction, and β is the decrease parameter.

4) VEGAS is a delay-based TCP variant. It has 2 thresholds, α and β, to control the amount of extra data, i.e $T_{extra} = T_{expected}$ - T_{actual}, where $T_{expected}$ is an estimation of expected throughput calculated by $T_{expected} = windowsize$ / $smallestmeasuredRTT$. Window size of VEGAS is updated as follows:

If $T_{expected} < \alpha$, window size increased by 1.

If $\alpha < T_{expected} < \beta$, no change in window size.

If $T_{expected} > \beta$, window size decreased by 1.

A detailed understanding of the many facts of network parameters is critical for evaluating the performance of networking protocols, for assessing the effectiveness of proposed protocols, and for developing the next generation high speed networks. In this work, we narrow down our focus to three key components that affect the performance of 10Gbps networks: TCP variants, queue management schemes, and router buffer size.

2.1.4 Experimental Preparation

1. CRON Setup

CRON [16] is an emulation-based 10Gbps high speed testbed, which is a cyberinfrastructure of reconfigurable optical networking environment that provides multiple networking testbeds operating up to 10Gbps bandwidth. As shown in Figure 2.1, CRON provides users with automatic configuration of arbitrary network topologies with 10Gbps bandwidth. Also, CRON can be federated with other 10Gbps high speed networks, such as Internet2, NLR, and LONI. Details of demonstrations of how to use the CRON testbed could be found here [1].

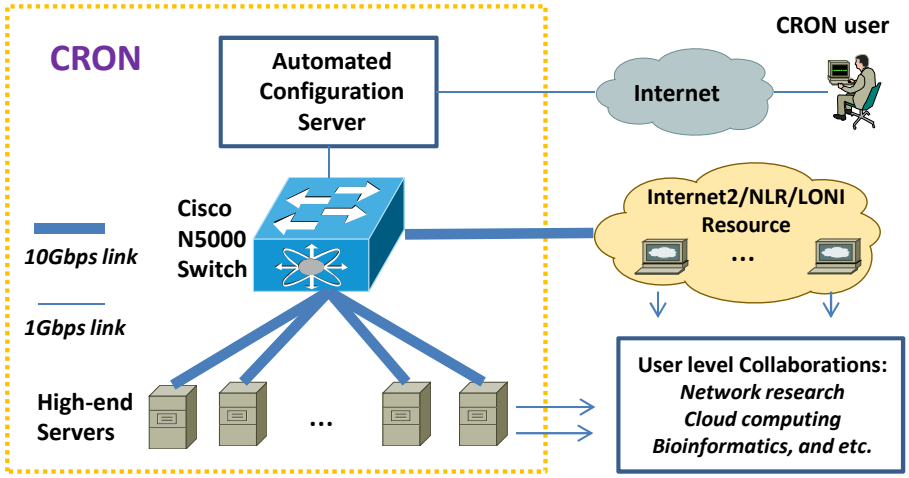


FIGURE 2.1. CRON system architecture consisting of routers, delay links, and high-end workstation operating up to 10Gbps bandwidth

As dumbbell topology is a widely accepted network topology, we create a dumbbell topology as shown in Figure 2.2 in CRON. In the topology, all nodes are Sun Firex4240 servers which have two quad core 2.7-GHz AMD Opteron 2384 processors, 8 GB/s bus, 8GB RAM and 10GE network interface cards. From a software perspective, two pairs of senders and receivers run a modified version of Linux 2.6.34 kernel, which supports TCP variants of CUBIC, HSTCP, RENO, and VEGAS. The routers run a modified version of Linux-2.6.39.3 kernel, which supports queuing disciplines of Drop-tail, RED, CHOKe, and SFB. The delay node runs a modified version of FreeBSD 8.1, which supports a 10Gbps version of Dummynet[11] with 10Gbps bandwidth and enlarged queue size.

We set the RTT on the delay node to 120ms. All the links have a 10Gbps capacity, and the bottleneck link is the one between Router1 and Router2. We set the queue disciplines at the output queue of Router1, where the congestion happens.

By default, we send 10 flows from Sender1 to Receiver1 and 10 flows from Sender2 to Receiver2. We choose the number of flows to be 10 according to recent statistics of network flows of Internet2 [37], which suggested that the number of long-lived TCP flows are always several tens of flows in 10Gbps high speed networks such as Internet2.

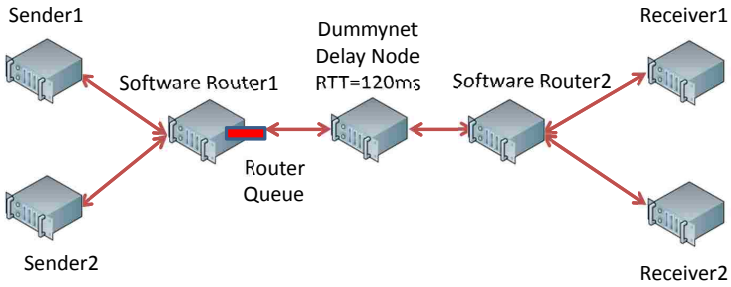


FIGURE 2.2. Experimental topology: dumbbell topology

The duration of each emulation test is 20 minutes, and all tests run for 7 to 10 times. We get the final result based on the average value.

2. System Tuning for 10Gbps

To get a systematic and repeatable 10Gbps network environment, we perform system tuning and software patching in the CRON testbed. Firstly, on the senders and receivers with Linux kernel 2.6.34, we enlarge the default TCP buffer size for high speed TCP transmit. According to [78], we implement the zerocopy Iperf to avoid the overhead of data copy from user-space to kernel space, and we enable packets large receive offload (LRO) and TCP segment offload on the NICs. We also set MTU to 9000 Bytes [72].

Secondly, on the routers with Linux kernel 2.6.39.3, the Linux default queuing discipline controller, traffic control (tc), does not support control for CHOKe and SFB in user-space. So we patch tc(8) to support CHOKe and SFB. In kernel-space, we find that for RED, the scaled parameter of maximum queue threshold only supports 24 bit value which means up to only 16MB. So we change it to a 56 bit value to support a higher maximum queue threshold. In addition, we use standard $skbuf$ to forward packets and disable LRO on the router NICs.

Thirdly, on the delay node which runs FreeBSD 8.1, we optimize memory utilization by creating a continuous memory space for received packets to overcome the drawback of the memory fragmentation of $Mbuf$ allocation in FreeBSD. In Dummynet, we

change the type of bandwidth from *int* to *long* so that its bandwidth capacity gets an improvement from 2Gbps to 10Gbps. We also increase the value of the Dummynet hardware interruption storm threshold.

3. Queue Parameter Setup

Queue management schemes in consideration along with its parameters are shown in Table 2.1.

In [4], the authors suggest that a link needs only a buffer of size O(C / $\sqrt{N}$), where C is the capacity of the link, and N is the number of flows sharing the link. In addition, we vary the router buffer size to examine all the combinations of TCP variants and queue management schemes. In [20], the authors suggest that buffers can be reduced even further to 20-50 packets. Given the significance of their role, we vary the buffer size as 1%, 5%, 10%, 20%, 40%, and 100% of BDP to find out the impact of the router buffer sizing on AQM schemes in 10Gbps high speed networks.

TABLE 2.1. Parameter setup for 4 queue management schemes

Queue	Parameter Setup
Drop-tail	queue length *limit*: from 1% to 100% BDP
RED	queue length *limit*: from 1% to 100% BDP minimum threshold qth_min: $0.1 \times limit$ maximum threshold qth_max: $0.9 \times limit$ average packet size *avpkt*: 9000 maximum probability max_P: 0.02
CHOKe	queue length *limit*: from 1% to 100% BDP minimum threshold qth_min: $0.1 \times limit$ maximum threshold qth_max: $0.9 \times limit$
SFB	queue length *limit*: from 1% to 100% BDP increment of dropping probability *increment*: 0.00050 decrement of dropping probability *decrement*: 0.00005 Bloom filter uses two 8 x 16 bins target per-flow queue size *target*: 1.5/N of total buffer size N: number of flows maximum packets queued *max*: $1.2 \times target$

4. Performance Metrics

(1) Link Utilization. Link utilization is the percentage of total bottleneck capacity utilized during an experiment run.

(2) Intra-protocol Fairness. Intra-protocol fairness represents the fairness among all TCP flows. Long term flow throughput is used for computing fairness according to Jain's fairness index [39].

(3) RTT fairness. RTT fairness is the fairness among TCP flows with different RTTs. Longer RTT TCP flows suffer from lower throughput and shorter RTT flows get more throughput.

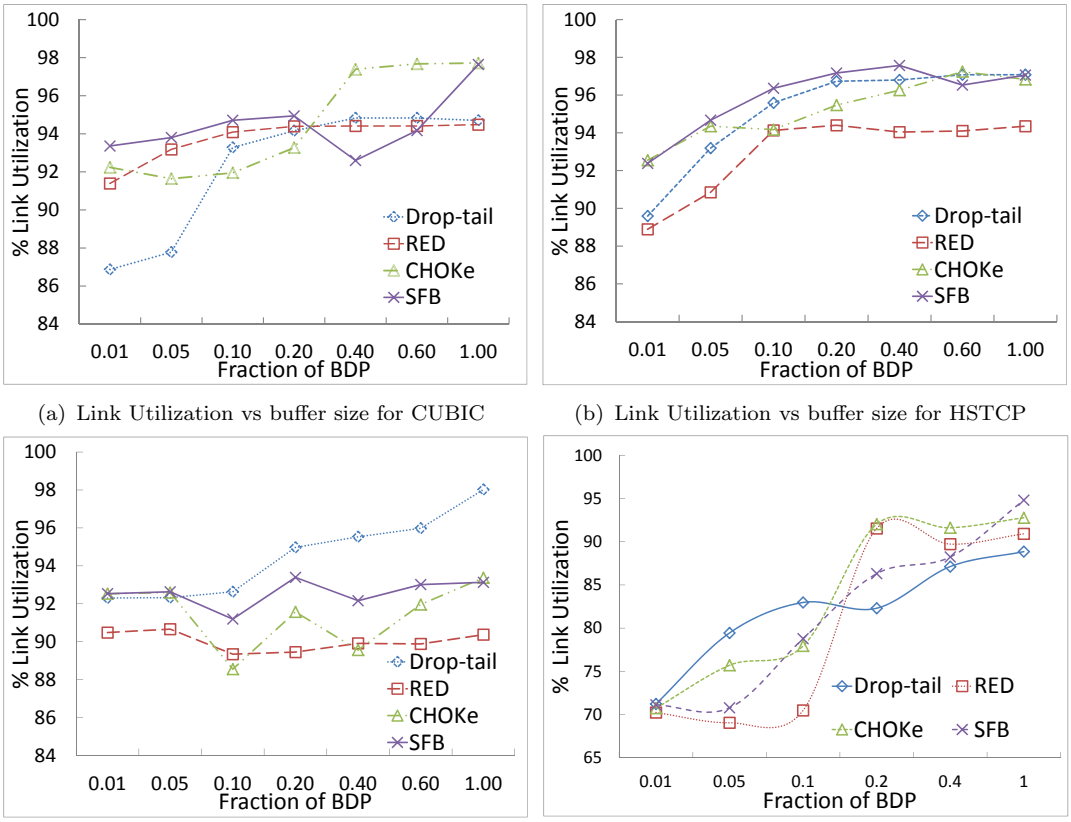
(4) Delay. Delay is the average end-to-end delay experienced by the flows, also known as Round-trip delay (RTT) of packets across the bottleneck paths. It includes the queuing delay created by the queue at router.

(5) Computational Complexity. Computational complexity is the algorithm space and time complexity of the queue management schemes, which is the memory consumption and CPU usage on the servers in our experiments.

2.1.5 Results of Experimental Evaluation

1. Link Utilization

We vary queue buffer size to observe the link utilization. Figure2.3(a), Figure2.3(b), Figure2.3(c), and Figure2.3(d) show link utilization for queue management schemes as a function of buffer size for CUBIC, HSTCP, RENO with SACK, and VEGAS respectively. With only 1% BDP buffer size on the bottleneck link; almost all of the queue management schemes for all TCP variants show more than 85% link utilization, which is close to the previous sizing router buffer researches [4], [45]. And if the buffer size reaches 10% of BDP, almost all the queue management schemes under all TCPs will get more than 90% link utilization except for TCP-VEGAS.



(a) Link Utilization vs buffer size for CUBIC

(b) Link Utilization vs buffer size for HSTCP

(c) Link Utilization vs buffer size for RENO

(d) Link Utilization vs buffer size for VEGAS

FIGURE 2.3. Link Utilization as a function of buffer size in each TCP variant

Figure2.3(a) is for link utilization of queue management schemes for CUBIC, when the buffer size is very small to 1% BDP, Drop-tail performs worst, while SFB gets highest link utilization among others. If the buffer size increases up to 100% BDP, CHOKe and SFB get higher link utilization. In Figure2.3(b), link utilization of queue management schemes is for HSTCP, and SFB almost always outperforms other queuing schemes, while RED almost always gets lowest link utilization than others. Figure2.3(c) shows the link utilization of queue management schemes for RENO with SACK, Drop-tail performs best in this case, SFB is still better than the other two, while RED almost always gets the lowest link utilization.

In Figure2.3(d), VEGAS shows different link utilization behaviors because of its delay-based nature, which depends on the queue size. In general, when the buffer size

becomes larger, all queue management schemes get higher link utilization. In the case of less than 10% BDP, Drop-tail almost always gets the highest link utilization. In the case of more than 10% BDP, AQM schemes almost always get higher link utilization. The reason is when the queue size becomes larger, AQM schemes do not have early drops because VEGAS controls the queue size in a limited range, and therefore, link utilization of the AQM scheme improves.

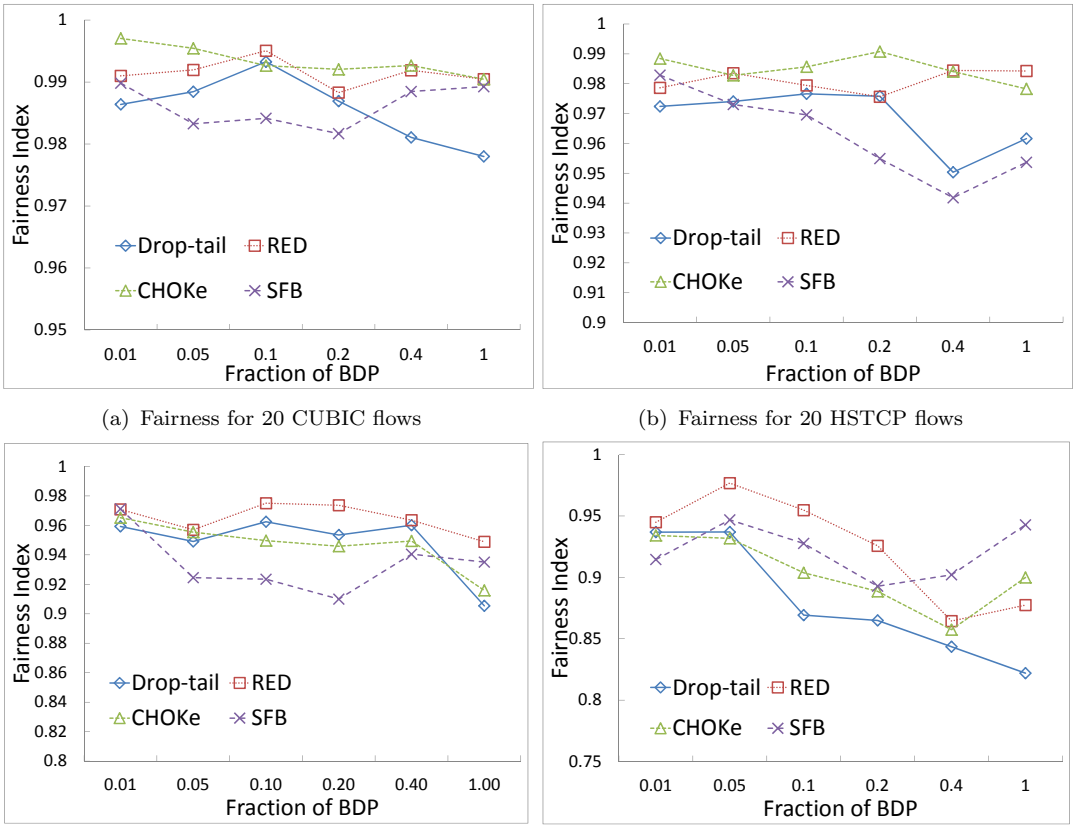
2. Intra-protocol Fairness

Intra-protocol fairness is the fairness among TCP flows with the same kind of TCP. In our evaluation, Jains fairness index is calculated for intra-protocol fairness among 20 flows with same TCP variant and same RTT of 120ms.

Figure2.4 shows the intra-protocol fairness for these 20 flows. In general, CUBIC shows the highest fairness index, which has a fairness index in the range of 0.97 to 1. HSTCP seconds with a fairness index in the range of 0.94 to 0.99. RENO is third, which has a fairness index higher than 0.89. VEGAS is last with a fairness index higher than 0.8.

Figure2.4(a) shows the case for CUBIC, RED and CHOKe have a very high intro-protocol fairness around 0.99 fairness index. SFB generally shows lower intro-protocol fairness than other queue management schemes. According to our observation, although SFB has bucket drops to limit the fast flows, SFB always has more tail drops than other queue management schemes in high speed networks because of its more complex queuing mechanism. That is the reason SFB can not perform as fairly as other AQM schemes.

Figure2.4(b) is the result for HSTCP, which is similar to the case of CUBIC. RED and CHOKe still show higher intro-protocol fairness, while Drop-tail and SFB show lower intro-protocol fairness.



(a) Fairness for 20 CUBIC flows

(b) Fairness for 20 HSTCP flows

(c) Fairness for 20 RENO flows

(d) Fairness for 20 VEGAS flows

FIGURE 2.4. Fairness among 20 flows as a function of buffer size in each TCP variant (RTT = 120ms)

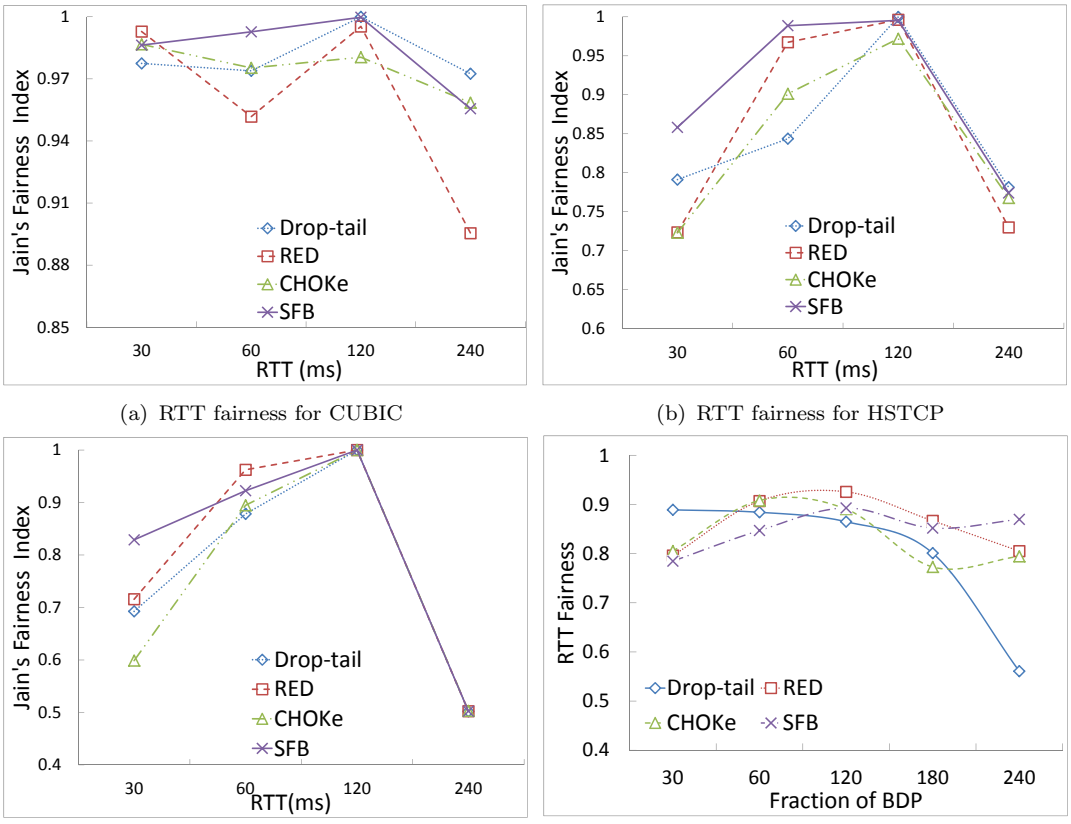
In the case of RENO, Figure2.4(c) shows that RED always gets the highest intro-protocol fairness. Drop-tail is second, and CHOKe is the third. The slow instinct of RENO makes AQM schemes have a similar performance to Drop-tail in terms of fairness. Whenever RENO flows have early drops from AQM schemes, it takes some time for the flows to recover, which in consequence degrades the fairness of AQM schemes. SFB still almost always shows the lowest intro-protocol fairness because of having more tail drops than others.

Figure2.4(d) shows the case for VEGAS. Since VEGAS is a delay-based TCP variant, it keeps the queue size as small as possible. AQM schemes all get better fairness than Drop-tail. Drop-tail makes relatively large changes on the queue size, and therefore it performs relatively unfair.

3. RTT Fairness

We measure RTT fairness by Jain's fairness index for 20 competing flows of the same TCP variant but different RTTs. In these 20 flows, 10 of them have a fixed RTT, while the other 10 flows have a different RTT. The RTT of 10 flows from Sender1 is fixed at 120ms, while the RTT of the other 10 from Sender2 is changed as 30ms, 60ms, 120ms, and 240ms respectively. We set the bottleneck buffer to 10% BDP because link utilization of the bottleneck link shows good performance with buffer size 10% BDP in general. Also, in this case queuing delay can be neglected.

Figure 2.5(a) shows CUBIC RTT fairness for four queue management schemes. In our measurement, CUBIC shows very good behavior of RTT fairness. Even under 240ms RTT, every queue management scheme shows more than 90% of RTT fairness.



(a) RTT fairness for CUBIC

(b) RTT fairness for HSTCP

(c) RTT fairness for RENO

(d) RTT fairness for VEGAS

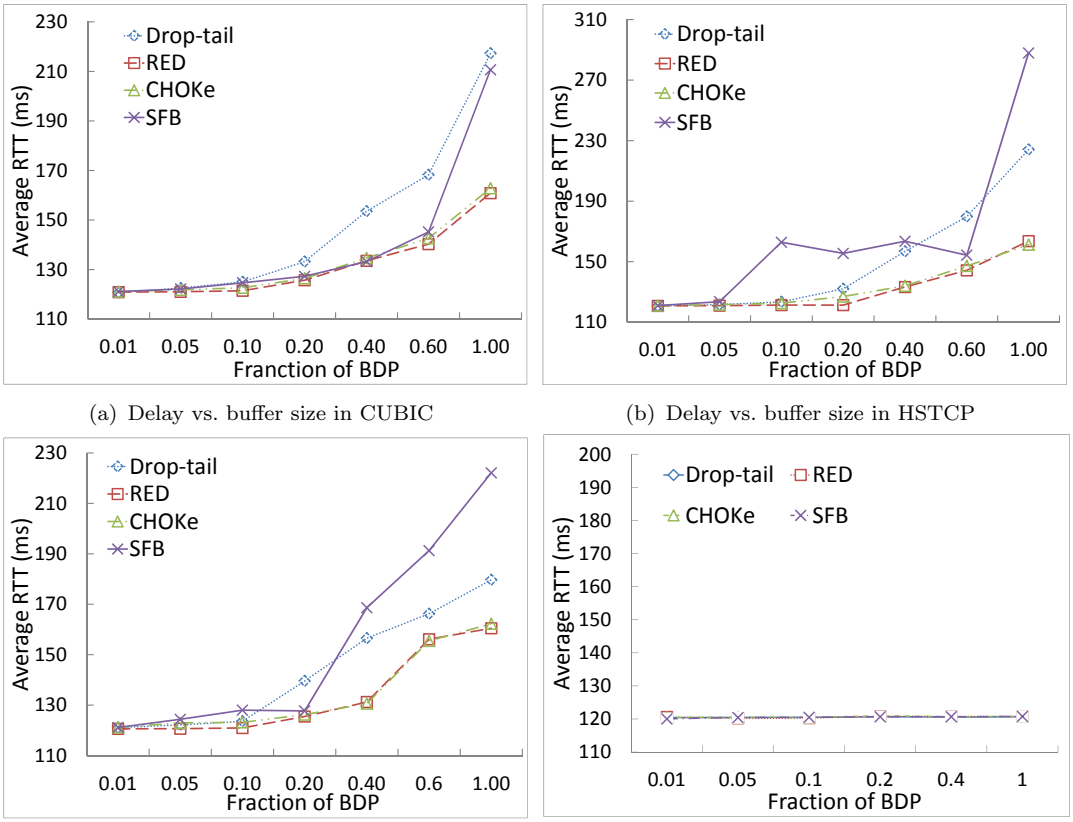
FIGURE 2.5. RTT fairness as a function of RTT in each TCP variant

SFB performs better than others; while RED gets least fairness. Figure 2.5(b) and 2.5(c) show HSTCP's and RENO's RTT fairness cases. HSTCP's RTT fairness is better than RENO's for all queue management schemes, and both HSTCP's RTT fairness and RENO's RTT fairness are quite lower than CUBIC's. We can still see SFB shows the best for all the TCP variants under different RTT scenarios, and when one RTT increases to 240ms, we get a low RTT fairness in both cases. Figure 2.5(d) shows VEGAS's RTT fairness. Every queuing scheme gets around 0.8 to 0.9 fairness index except that of Drop-tail which gets a low RTT fairness in the case of 240ms RTT.

4. Queuing Delay

We observe performance in delay by varying buffer size. Since this measurement is based on round trip propagation delay of 120ms, the average RTT will be $120 + [0, max\_queuing\_delay]$. Figure 2.6(a) shows CUBIC result, Drop-tail always has more queuing delays than others. SFB is the second with more queuing delay than RED and CHOKe. Figure 2.6(b) shows result for HSTCP, we can still see Drop-tail and SFB show more queuing delay than the others, and SFB shows an oscillation, and exhibits queue delays more than double the amount of propagation delays. Figure 2.6(c) is for RENO, Drop-tail and SFB queuing delays grow faster than the others. RED and CHOKe show nearly the same behavior and both grow more smoothly with the increase in buffer size as compared to Drop-tail and SFB.

Figure 2.6(d) shows results for VEGAS. For all queue management schemes, VEGAS almost does not create any queuing delay. In all cases, the average RTTs for VEGAS are only 120ms, which is the propagation delay we set. This is because VEGAS itself maintains the queue in a very small size, such as several packets. The results confirm that delay-based TCP variant maintains a stable and small queue size in 10Gbps high speed networks.



(a) Delay vs. buffer size in CUBIC

(b) Delay vs. buffer size in HSTCP

(c) Delay vs. buffer size in RENO

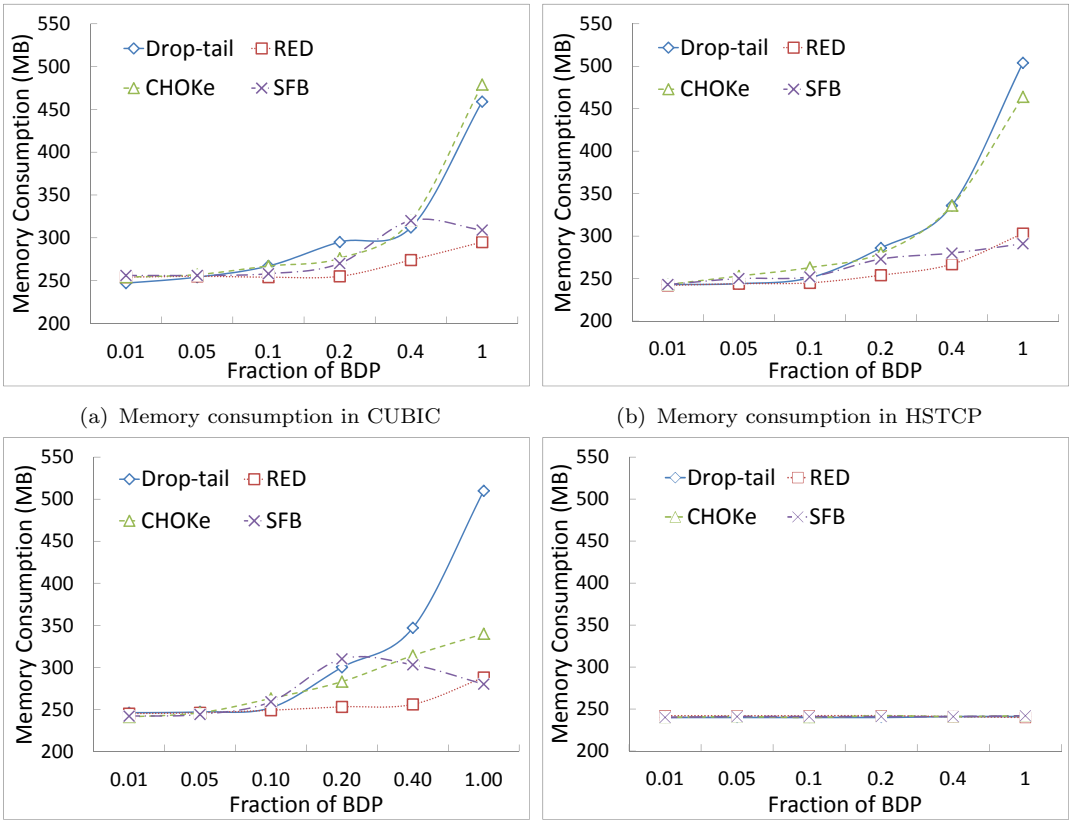
(d) Delay vs. buffer size in VEGAS

FIGURE 2.6. Delay as a function of buffer size in each TCP variant

5. Computational Complexity

Figure 2.7 shows the average memory consumption in a function of buffer size on the bottleneck router. We can see that for CUBIC, HSTCP and RENO, the general trend is that when the buffer size increases, the memory consumption increases. When the buffer size is 100% BDP, the memory consumption reaches more than 500MB. Drop-tail generally needs more memory than other AQM schemes. Figure 2.7(d) shows that VEGAS almost does not create any additional memory for queuing mechanisms, because it maintains a very small size queue.

The results of CPU usage reveal less than 10% of total CPU usage of all CPU cores in all of our experiments, and therefore we do not list the detailed CPU usage result here.



(a) Memory consumption in CUBIC

(b) Memory consumption in HSTCP

(c) Memory consumption in RENO

(d) Memory consumption in VEGAS

FIGURE 2.7. Memory consumption as a function of buffer size in each TCP variant

2.1.6 Summary

We present the experimental study of the interplay of queue management schemes and high speed TCP variants over a 10Gbps high speed networking environment. TCP specific performance metrics such as link utilization, fairness, delay, and computational complexity are chosen to compare the impact of queuing schemes on the performance of TCP-RENO, CUBIC, HSTCP and VEGAS. Our test reveals that Drop-tail is most suited for TCP-RENO and observed to be worst for CUBIC and HSTCP. In our experiment scenario, we observe at least 10% BDP of buffer size is required for more than 90% link utilization. RED exhibits higher fairness as compared to other QMs for all the TCP variants. SFB is shown to be effective in RTT fairness improvement. TCP VEGAS shows very low queuing delay and memory consumption.

In summary, we observe differences in performance of queue management schemes for different TCP variants.

We hope that even a preliminary understanding of key factors, when combined with critical performance metrics, can provide a perspective that is easily understood and can serve as guidelines for network designers of 10Gbps high speed networks. Also, the results of the study address the current need for the research on the impact of queue management schemes on the performance of the high speed TCP variants. It is also desirable to observe the same impacts on a more realistic experimental environment by considering background traffic. For our future work, it is interesting to observe the impact of these queue management schemes in a wide range of different network topologies. In this study, although our focus has been on homogeneous TCP flows, we expect a different behavior in the case of heterogeneous TCP flows. The presented work supports further research work on the design and deployment issues of queue management schemes for high speed networks.

2.2 A Study of Fairness among Heterogeneous TCP Variants over 10Gbps High-speed Optical Networks

In this section, we conduct an experimental study of fairness among heterogeneous TCP variants over 10Gbps high-speed optical networks. We choose most popular TCP variants and present the fairness behavior of these heterogeneous TCP variants.

2.2.1 Overview

High-speed optical networks such as National Lambda Rail (NLR) [51], Internet2 [36], Louisiana Optical Networks Initiative (LONI) [48], etc., with their capacity to deliver a data transfer rate in excess of 10Gbps, have been developed to serve the demand of end users. Efficiency and fairness of these high-speed optical networks among its end users have been concerned among researchers. Traditional congestion control mechanism Additive Increase Multiplicative Decrease (AIMD) has been outstanding

in fulfilling the requirements of efficiency and fairness, and has been a first choice for large networks. However, it is not scalable in high bandwidth and large delay networks. Therefore, researchers proposed several TCP variants, such as HighSpeed TCP (HSTCP) [22], Scalable TCP (STCP) [44], FAST TCP (FastTCP) [41], BIC TCP (BIC) [73], CUBIC TCP (CUBIC) [31], etc. Open sourcing of these protocols enables Linux/BSD users to choose their own transport layer protocols. Also, a recent study [77] on 5000 most popular web servers shows that AIMD, BIC/CUBIC, and HSTCP/CTCP are used by 16.85-25.58%, 44.5%, and 10.27-19% respectively among the web servers. Active use of these high-speed TCP variants by web servers suggests that our traditional homogeneous network is rapidly evolving into a heterogeneous one.

Because every TCP employs its own congestion control mechanism differently, it is hard to predict behavior of a network where these different TCP variants interact at a bottleneck link. Since these TCP variants are designed for high-speed networks, we believe they are all able to provide high throughput, and we confirm the high link utilization for heterogeneous TCP flows in Section 2.2.4. On the other hand, these TCP variants react differently to packet losses and use different mechanisms to quickly adapt to the available bandwidth. Fairness among heterogeneous TCP variants becomes hard problem, and mainly depends on router parameters such as queue management schemes and buffer size [65].

Most of TCP variants have been evaluated in fairness behavior against itself or traditional AIMD [29]. However, it is clear that future high-speed optical networks will be heterogeneous in nature, which means it will consist of flows of many different variants of TCP. Therefore, it is critical for us to evaluate fairness behavior among heterogeneous TCP flows in newly emerging high-speed optical networks.

To address the fairness behavior, it is important to understand the interaction of these TCP variants in a high-speed optical network environment. However, due to the high cost and limited availability of high-speed optical networks, it is hard for network researchers or operators to get privilege to evaluate network performance by investigating different network parameters, such as heterogeneous TCP variants, router parameters, etc. In this work, we evaluate fairness among heterogeneous TCP flows with various network parameters over a cyberinfrastructure of reconfigurable optical networking environment (CRON) [16], which is an emulation-based reconfigurable 10Gbps high-speed optical network testbed.

Moreover, this study presents fairness issues among heterogeneous TCP variants, which could exist in all-optical routers that are designed to have very limited buffer size. One of the challenges in deploying high-speed optical networks is to manage all-optical routers with very small buffer to cooperate with various TCP variants [12, 60, 70]. This study brings the fairness issues to the table and matches the needs of preliminary research of deploying all-optical routers in high-speed optical networks.

Our heterogeneous TCP flows consist of flows of TCP-SACK, HSTCP and CUBIC. It is to be noted that these three protocols have substantial presence in current Internet [77]. Experimental scenarios presented in this study have similar traffic characteristics as observed in high-speed optical networks such as Internet2 [37], where number of high speed flows (long-lived TCP flows, bulk data transfer, etc.) ranges from a few to a few tens of flows. Thus, we first show fairness behavior for single long-lived TCP flow case; then the case for many long-lived TCP flows; and finally the case with both long-lived TCP flows and short-lived TCP flows. We also show RTT fairness behavior with and without short-lived TCP flows.

Among router parameters, we choose queue management schemes such as Drop-tail, RED(Random Early Detection) [24], CHOKe(CHOose and Keep for responsive flows,

CHOose and Kill for unresponsive flows) [53], and AFD(Approximated Fair Dropping) [52]. RED, CHOKe, and AFD, as active queue management (AQM) schemes, have been studied for a long time to avoid network congestion and to improve fairness. Furthermore, fairness behavior is presented for buffer sizes ranging from 1% to 100% of Bandwidth-Delay Product (BDP), where $BDP = C \times RTT$, C is the data rate of the link and RTT is the round trip time. Nowadays, researchers are looking to decrease router buffer size to a point where network performance can be optimized. In [4], authors suggested a link with n flows requires no more than buffer size of $BDP/\sqrt{N}$ for long-lived or short-lived TCP flows. And authors in [20] argued that buffer size of 20 - 50 packets could be sufficient to serve the requirements. Thus, we examine various router buffer sizes in our experiments to see the impact of buffer sizing on fairness among heterogeneous high-speed TCP flows.

Our findings suggest that fairness becomes poor when there are heterogeneous TCP flows mixed at the bottleneck link. AQM schemes, such as RED, CHOKe, and AFD, can improve fairness to some extent when router buffer size is more than 10% of BDP. For buffer sizes less than 10% of BDP, AQM schemes lose their advantage on fairness. Furthermore, multiplexing of many long-lived flows and short-lived flows improve fairness. To the best of our knowledge, this work is the first one to present a comprehensive study on fairness behavior of heterogeneous TCP flows over a 10Gbps high-speed optical network testbed.

2.2.2 Related Works

In [64, 66], authors explored that bandwidth allocation among heterogeneous flows is coupled with router parameters, such as router queue management schemes, router buffer size, etc. Their solution for fairness is source-based, which means every source should change its algorithm. The authors conducted pairwise comparison only between FastTCP and TCP-Reno through regular network simulations. In [52], authors

proposed a queue management scheme called Approximate Fair Dropping (AFD) to achieve reasonable fair bandwidth allocation. Then, AFD was evaluated by mixing heterogeneous TCP flows in the bottleneck. In their work, they assumed that router's buffer is sufficient. Although they considered different shadow buffer size b, they overlooked the impact of router buffer sizing on fairness behavior. Effect of buffer sizing on fairness has been investigated in [71]. Authors studied the interrelationship among fairness, small buffer size, and desynchronization of long-lived TCP flows. Their analysis is based on the pair of TCP-Reno and Drop-tail, but not on the heterogeneous instinct of current high-speed optical networks. In [32], authors added RED into consideration, and evaluated the impact of loss synchronization and buffer sizing on fairness behavior. However, they only evaluated intra-protocol fairness for homogeneous TCP flows. Also, in [72], authors raised TCP performance issue of intra-protocol fairness for 10Gbps high-speed optical networks, but they did not consider the influence of router parameters. In [70], the authors investigated the bandwidth sharing issue when heterogeneous traffic multiplex at an optical router with very small buffers. The authors explored buffer allocation strategies to share bandwidth among heterogeneous flows in optical packet switched networks. Their study only focused on the interplay between traditional TCP and UDP under different buffer sizes. Authors in [74] presented experimental results of intra-protocol fairness with different router parameters, and considered evaluation of inter-protocol fairness for heterogeneous TCP flows as future work. As a continuous work, authors in [76] evaluated fairness among heterogeneous high-speed TCP variants, but the results of AFD queue and RTT fairness for heterogeneous TCP variants were not presented.

In this study, we evaluate the fairness behavior of heterogeneous TCP flows mixed in the bottleneck link. Different queue management schemes, as well as different router

buffer sizes, have been considered in our experiments. Moreover, this study has been done for the first time in a 10Gbps high-speed optical networking testbed.

2.2.3 Experimental Design

1. Testbed Setup

As shown in Figure 2.8, we create a dumbbell topology in CRON. Multiple TCP sessions share a bottleneck link between two routers. Three separate senders, which run a modified version of Linux 2.6.34 kernel, are used to initiate TCP flows. The two routers run a modified version of Linux-2.6.39.3 kernel, which supports various Linux queuing disciplines. The delay node runs a modified version of FreeBSD 8.1, which supports 10Gbps version of Dummynet as a software network emulator. The bottleneck link is the link between Router1 and Router2. So the bottleneck queue is Router1's output queue.

All the presented results are averaged over five experiments and duration of each run is 20 minutes.

All links in the testbed support 10Gbps data transfer. We perform system tuning for a 10Gbps network environment which includes kernel optimizations for Linux and FreeBSD, TCP parameters tuning, NIC driver optimizations, jumbo frame, patches for 10Gbps version of RED and CHOKe queuing disciplines both in Linux user-space and kernel-space, and patches for Dummynet with enlarged queue size and bandwidth

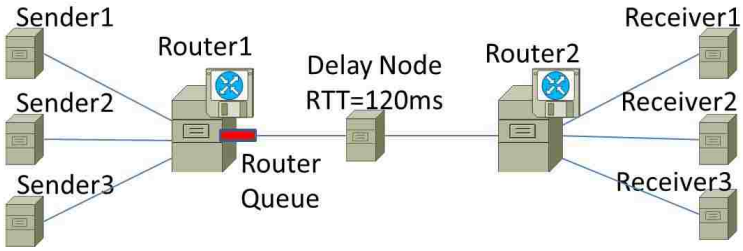


FIGURE 2.8. Experimental Topology

both in FreeBSD user-space and kernel-space. We implement AFD queuing discipline in Linux kernel as well as *tc* traffic control support for AFD.

2. Network Parameters

In our experiments, three senders send three different TCP variants, namely, TCP-SACK, CUBIC, and HSTCP. At output queue of Router1, we evaluate four kinds of Linux queuing disciplines, namely, Drop-tail, RED, CHOKe, and AFD. For parameters of RED and CHOKe, we set minimum queue size threshold to 20% of buffer size, maximum queue size threshold to 90% of buffer size, and drop probability to 0.02. For AFD, we set a shadow buffer to be 2000 slots and a sample interval of 500 packets. We vary the queue size at Router1's output queue from 1% to 100% of BDP. On delay node, default RTT is set as 120ms to emulate the long propagation delay in high-speed optical networks. For RTT fairness as shown in Section 2.2.4, we fix the RTT of two links as 120ms, and vary the RTT of the other link from 30ms to 240ms.

3. Traffic Generation

In our experiments, long-lived TCP flows are generated by zero-copy Iperf [78], which avoids the overhead of data copy from user-space to kernel space. In case of experiments with short-lived TCP flows, we add a pair of sender and receiver, and generate short-lived TCP flows using Harpoon traffic generator [62]. Harpoon generates continuous short-lived TCP requests from the client to the server. The TCP client and server have inter-connection times generated from exponential distribution with mean 1 second, and have file sizes generated from Pareto distribution with $alpha$=1.2 and $shape$=1500. The chosen distributions and parameters are taken from Internet traffic characteristics [29].

4. Fairness Index

Fairness is calculated among heterogeneous TCP flows in terms of the long term throughput received by each flow as Jain's fairness index [38]:

$$f(x_1, x_2, x_3, ..., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} \tag{2.4}$$

where f is Jain's fairness index, x is the long term throughput of the flow, and n is the flow number.

2.2.4 Evaluation Results and Discussion

In this section, we start from a simple scenario of single long-lived TCP flow. Next, we evaluate more practical scenarios of multiple long-lived TCP flows, and multiple long-lived TCP flows with short-lived TCP flows. Then we present our findings in RTT fairness. Link utilization results are shown at the end.

1. Fairness: A Case for Single Long-lived TCP Flow

We simultaneously send one TCP-SACK flow from Sender1 to Receiver1, one HSTCP flow from Sender2 to Receiver2, and one CUBIC flow from Sender3 to Receiver3. Figure 2.9 shows fairness index as a function of router buffer size for four different types of routers. The network behaves very unfairly as compared to previous studies focused on homogeneous TCP flows [29], [74]. To show the degree of decrease of fairness while transitioning from homogeneous network to heterogeneous one, fairness index is presented in Table. 2.2 for these two cases.

TABLE 2.2. Fairness: Homogeneous TCP vs Heterogeneous TCP (buffer size = 20% of BDP, RTT = 120ms, heterogeneous TCP variants include CUBIC, HSTCP, and TCP-SACK)

	Drop-tail	RED	CHOKe	AFD
CUBIC	0.988	0.994	0.991	0.995
HSTCP	0.978	0.987	0.990	0.993
TCP-SACK	0.936	0.977	0.970	0.985
Heterogeneous TCP	0.681	0.732	0.747	0.884

As shown in Figure 2.9, when the queue size on Router1 is larger than 10% BDP, AFD, CHOKe and RED all show more fairness behavior than Drop-tail. AFD shows

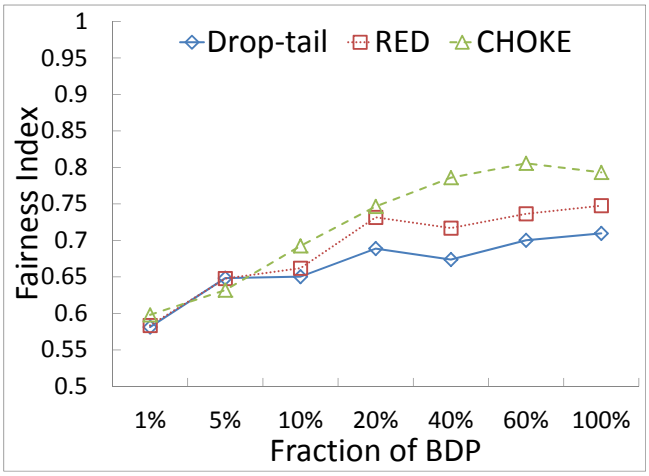


FIGURE 2.9. Fairness for 1 TCP-SACK, 1 CUBIC, and 1 HSTCP flow (RTT = 120ms)

much higher fairness index than other queue management schemes. CHOKe is the second, whereas RED is the third. Interestingly, when the queue size is less than 10% BDP, fairness behavior of AQM schemes degrades. CHOKe, RED and Drop-tail all show almost the same fairness, whereas AFD also has a performance degradation on fairness.

(1). Fairness problem for heterogeneous TCP flows

Based on our observation in Figure 2.9, firstly, we need to know why there is a fairness problem for heterogeneous TCP flows. We know that for loss based congestion control mechanism, the congestion window size $W(t)$ of a flow at time t is:

$$W(t) = W_{max}\beta + \alpha\frac{t}{RTT} \tag{2.5}$$

where W_{max} is the congestion window size just before the last window reduction, α and β are increase parameter and decrease parameter respectively.

Different TCP variants have different α and β. In our experiment, TCP-SACK has α to be 1 and β to be 0.5. HSTCP uses table driven increase and decrease factors, that is α is 1 to 72 and β is 0.5 to 0.9. CUBIC increases its congestion window based on a cubic function, which has a steadier growth. Among these three, CUBIC is the fastest in growth, HSTCP is the second, and TCP-SACK is the slowest. In

packet loss event, CUBIC and HSTCP get less congestion window degradation, while TCP-SACK gets more. Due to the large-bandwidth character of high-speed optical networks, the fairness problem for heterogeneous TCP variants becomes severer.

(2). Heterogeneous TCP flows over different queue management schemes

The second question is why different queue management schemes perform differently for fairness. The sending rate of a source not only depends on TCP variants in the sender but also depends on queue parameters in intermediate router. Router parameters may include loss probabilities, queue size, explicitly feedbacks, etc. Different queue management schemes create different congestion feedbacks to the senders.

Drop-tail drops packets for every flow when the queue is full. That causes a severe fairness problem as slow TCP variants (e.g. TCP-SACK) grow up slowly, and suffer from large loss penalty; while fast TCP variants (e.g. CUBIC and HSTCP) can always consume most of the bandwidth. RED alleviates the fairness problem by monitoring the average queue size and using randomization to choose flows to notify of congestion. The probability of dropping or marking a packet from a particular flow is roughly proportional to that flow's share of the bandwidth through the router. In case of CHOKe, besides the randomization method, it has a simple mechanism to approximately identify the flow rate. CHOKe checks a incoming packet with a randomly selected packet in the queue. If the two packets belong to a same flow, CHOKe identifies an estimated fast flow and drops the packet. Thus, fast flows get penalized, while slow flows get protected. AFD uses a shadow buffer and flow table to approximately estimate the sending rate of a flow. As the number of fast flows is small, AFD could be considered to be a stateless queuing scheme[52]. AFD also estimates a fair share rate of the network. If the estimated rate of a flow is faster than the fair share rate, the flow is penalized by dropping packets. Therefore, AFD performs much better than other queue management schemes in terms of fairness.

(3). Heterogeneous TCP flows over different queue sizes

The third question is why different queue sizes make different fairness behaviors over queue management schemes. To answer this question, we need to explore the relationship among queue size, loss synchronization and fairness.

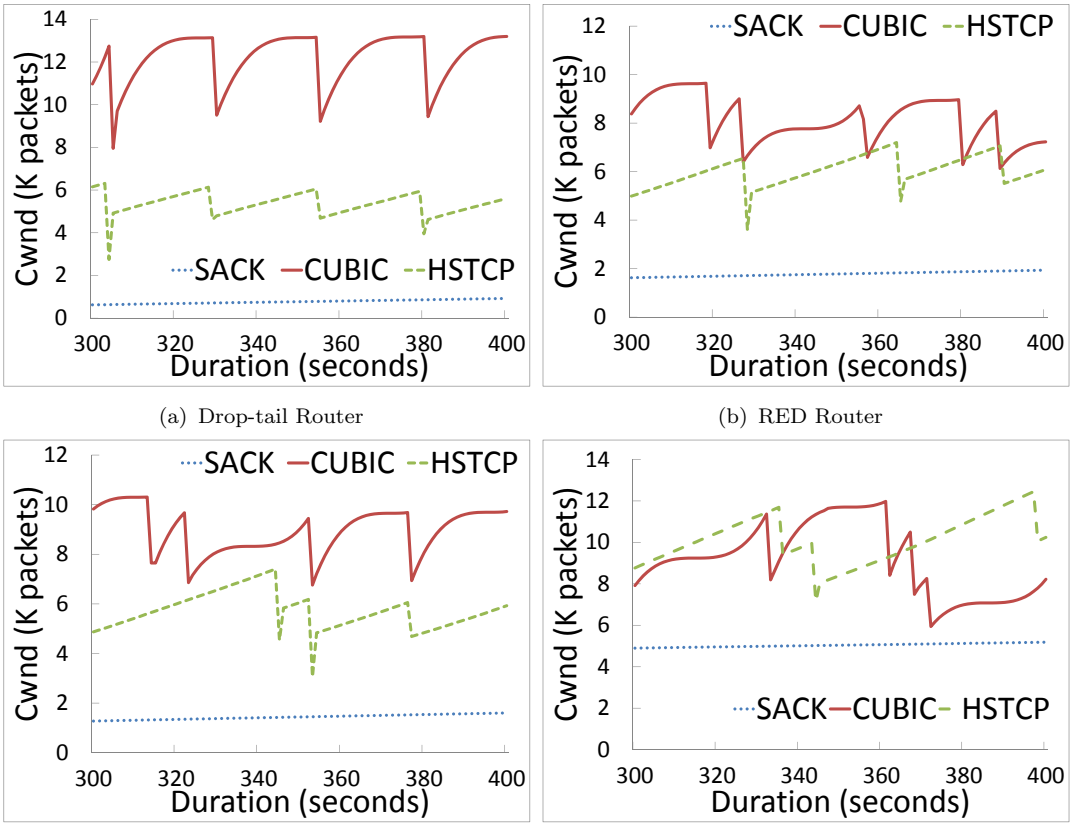
Different queue sizes create different behaviors of flow loss synchronization. A small queue size may induce more loss synchronization among flows as the queue is filled up very quickly and all flows have loss events at the same time. On the other hand, a large queue size is less likely to create loss synchronization among flows. AQM schemes are able to perform their own mechanisms before the queue is filled up, which may avoid the loss synchronization.

Different behaviors of flow loss synchronization create different fairness behaviors among heterogeneous TCP flows. Loss synchronization is the instinct of TCP, which happens when two or more TCP flows experience packet losses at same short time interval. Because of the synchronized loss events, every TCP flow simultaneously decreases its congestion window (Cwnd) without any differentiation. Therefore, fast TCP variants with large Cwnd can always grow faster than slow TCP variants. Thus, loss synchronization induces unfairness.

In the following, we zoom into Cwnd dynamics of these TCP variants to take a close look at the relationship among queue size, loss synchronization and fairness.

As Figure 2.9 shows, 20% of BDP queue size can distinguish AQM schemes from Drop-tail. We first set the router queue size to 20% BDP. In Figure 2.10, we show Cwnd dynamics between 300 to 400 second on time axis for these three TCP variants. Our goal is to analyze the difference in fairness for these queue management schemes. Figure 2.10(a) shows that Drop-tail makes CUBIC flow and HSTCP flow highly synchronized. In the RED router, Figure 2.10(b) shows CUBIC and HSTCP flows are desynchronized. At different times, CUBIC flow and HSTCP flow get Cwnd drops.

Figure 2.10(c) is for CHOKe router, flows are also desynchronized. CHOKe tries to drop fast flows, and to protect slow flows. Although CHOKe drops one time at HSTCP flow at the 345th second, this is because of the instinct of mis-detection (detecting a non-high-bandwidth flow) of AQM schemes. Results for AFD router are shown in Figure 2.10(d). AFD makes more fairness than other queue management schemes. CUBIC flow is penalized because of its fast sending rate. In the figure, the line of HSTCP flow Cwnd overlaps with the line of CUBIC flow Cwnd. The Cwnd of TCP-SACK flow is also increased. In all four figures, there is no Cwnd drop on TCP-SACK flow, but on a bigger time scale, we observe that RED, CHOKe, and AFD get fewer drops at TCP-SACK flow than Drop-tail. Given these figures, we confirm that AQM schemes tend to avoid loss synchronization among TCP flows. That's why these three



(a) Drop-tail Router

(b) RED Router

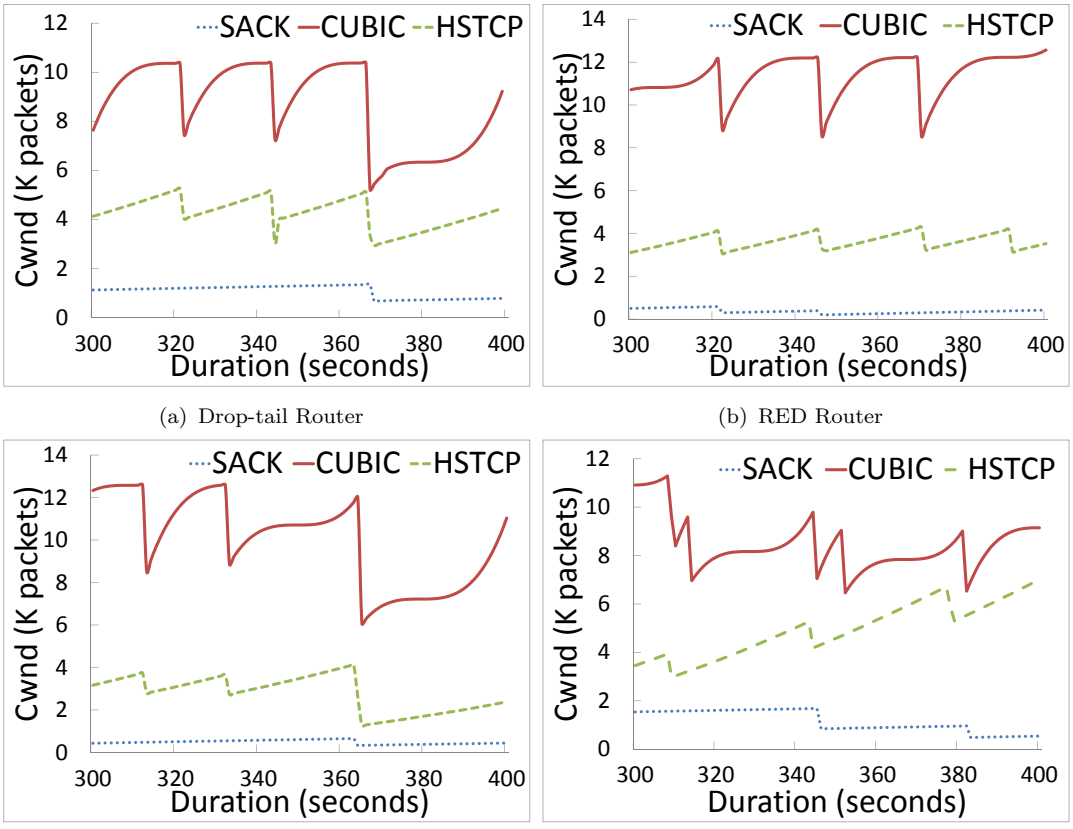
(c) CHOKe Router

(d) AFD Router

FIGURE 2.10. Instant Cwnd for 3 TCP Flows in 300-400 Seconds (Buffer Size = 20% BDP, RTT = 120ms)

AQM schemes perform better fairness than Drop-tail when the queue size is large enough.

As Figure 2.9 shows in 1% of BDP queue size all queue management schemes have similar poor fairness behavior, we set router queue size to 1% of BDP. We get the Cwnd plot again between 300 to 400 second in Figure 2.11. This time, Figure 2.11(a), Figure 2.11(b), Figure 2.11(c), and Figure 2.11(d) show three TCP flows get loss synchronization in all four routers. The reason why AQM schemes behave like Drop-tail in small queue size is that with small queue size heterogeneous high-speed TCP flows flush the queue quickly in a high-bursty manner. Within such a short amount of time, AQM schemes consider early drops the same as tail drops. Small queue size



(a) Drop-tail Router

(b) RED Router

(c) CHOKe Router

(d) AFD Router

FIGURE 2.11. Instant Cwnd for 3 TCP Flows in 300-400 Seconds (Buffer Size = 1% BDP, RTT = 120ms)

makes loss synchronization no matter which queue management scheme the router uses and therefore, all four queue management schemes perform similar poor fairness.

Our experimental results confirm that, in case of large queue sizes, AQM schemes achieve desynchronization among heterogeneous high-speed TCP flows, which penalizes the fast flows, protects the slow flows and improves fairness. With small queue sizes, heterogeneous high-speed TCP flows become synchronized regardless of what kind of router these TCP flows go through, which in consequence makes no differentiation on the flows and makes all routers behave similar in fairness.

3. Fairness: A Case for Multiple Long-lived TCP Flows

To emulate a more practical scenario, we start more flows into the bottleneck. We simultaneously send 10 flows for each of TCP-SACK, CUBIC, and HSTCP to the corresponding receivers. Fig 2.12 shows fairness for a total of 30 heterogeneous TCP flows competing for 10Gbps bottleneck bandwidth. The fairness index varies from 0.65-0.95, which is more than the scenario of single flow case. Multiplexing of large number of heterogeneous high-speed TCP flows on a single bottleneck link creates desynchronization of the TCP flows. As a result, fairness is improved.

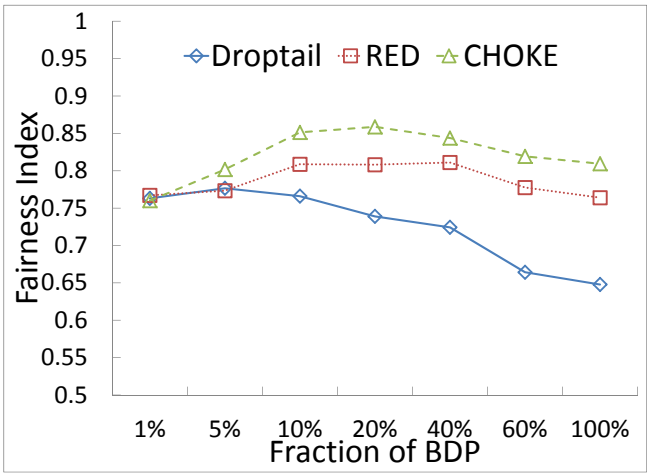


FIGURE 2.12. Fairness for Multiple Long-lived Flows: 10 TCP-SACK, 10 CUBIC, and 10 HSTCP flows (RTT = 120ms)

AFD still shows the best fairness performance among all queue management schemes. Heterogeneous TCP flows get 0.8 to 0.96 fairness index in all queue sizes, but still the fairness performance decreases with the queue size being small. The reason is that a larger router buffer is more likely to hold the incoming packets for all TCP flows, and therefore AFD is able to perform its mechanism to approximately estimate the sending rate of flows and to tame the unfairness.

For RED and CHOKe, if the queue size is more than 10% BDP, RED gets more fairness index than Drop-tail by 0.05 to 0.15, and the larger the buffer is the more fairness RED gets than Drop-tail does. And CHOKe further improves fairness index to RED by around 0.05. Again, when queue size is 1% BDP, RED and CHOKe perform the same as Drop-tail. Compared to the case of single TCP flow with 1% BDP in Figure 2.9, fairness index improves from less than 0.6 to more than 0.75. Although with 1% BDP buffer size, multiplexing of 30 heterogeneous TCP flows makes flow desynchronized so that fairness gets much improved. However, RED and CHOKe still do not have advantage over Drop-tail in terms of fairness. We observe the Cwnd for all 30 heterogeneous flows in 1% BDP. Unlike the case for single long-lived TCP flow, the Cwnd of 30 long-lived TCP flows this time shows desynchronized because of multiplexing. Also, RED, CHOKe and Drop-tail get similar desynchronization, which is because with small size buffer and bursty traffic, RED and CHOKe do not have adequate time to drop packets early and gently for many high-speed TCP flows.

We also see that with buffer sizes larger than 10% BDP Drop-tail starts to get fairness degradation in case of many long-lived TCP flows. The Cwnd plot for Drop-tail router shows that with buffer size set to some extent which is large enough to accommodate the incoming packets, some of the fast flows have high chance to be dropped at tail, while other fast flows can still grow without tail dropping. Thus, slow flows get lesser throughput and fairness becomes poor.

4. Fairness: A Case for Multiple Long-lived TCP Flows with Short-lived TCP Flows

Now we want to do some more realistic experiments as in high-speed optical networks. The traffic statistics of high-speed optical networks [37] show the distribution of network traffic consists of long-lived TCP flows and short-lived TCP flows. Although most of the flows are short-lived TCP flows (web browsing, small file transfers, etc.), they consume very little bandwidth. A large fraction of the traffic is sent by long-lived TCP flows, but the number of them is small. In this section, we add short-lived TCP flows along with 30 heterogeneous TCP flows into the bottleneck.

Figure 2.13 shows our results. When queue size is less than 10% BDP, we see fairness improvement than the case without short-lived TCP flows (around 0 - 0.05 fairness index). We plotted out the Cwnd graph for 30 long-lived TCP flows when short-lived TCP flows are injected with 10% BDP buffer size, and found that short-lived TCP flows induce more randomized losses for long-lived TCP flows. Thus, long-lived TCP flows get more desynchronized than the case without short-lived TCP flows. Because of randomized losses caused by short-lived TCP flows, fast flows have higher probability to be dropped. Accordingly, fairness is better than the case without short-lived TCP flows.

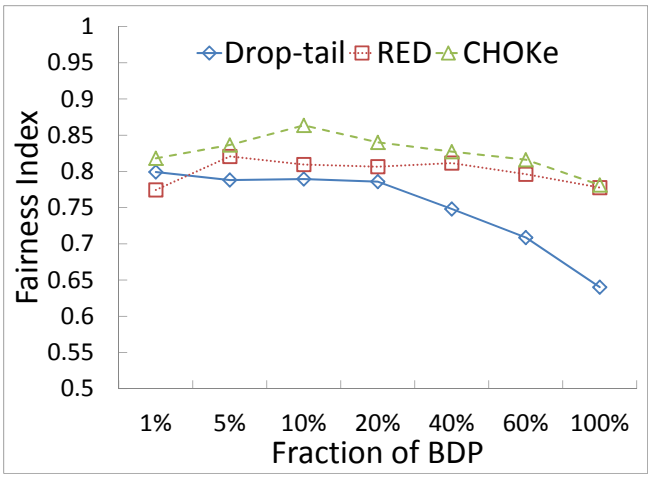


FIGURE 2.13. Fairness for Multiple Long-lived flows with Short-lived flows: 10 TCP-SACK, 10 CUBIC, 10 HSTCP flows, and Short-lived flows (RTT = 120ms)

When queue size is more than 10% BDP, we do not see much fairness improvement as compared to the case without short-lived TCP flows. That is because long-lived TCP flows get less chance to be impacted by the short-lived TCP flows as the queue size is large enough to hold all the packets.

Our results show that with short-lived flows, heterogeneous high-speed flows generally get more fairness. Those short-lived TCP flows make the long-lived TCP flows more desynchronized, especially when queue size is less than 10% BDP, and therefore fairness performance is improved.

5. RTT Fairness

RTT fairness is an important metric in TCP study. It measures fairness of bandwidth sharing among competing TCP flows that have different RTTs. As we know that bandwidth of a TCP flow is inversely proportional to RTT of the TCP flow. TCP flows with long RTT tend to get less bandwidth than TCP flows with short RTT flows. For end users of high-speed optical networks, it is desirable to understand the RTT fairness behavior to avoid starvation for long RTT TCP flows.

We first measure the RTT fairness for 2 homogeneous TCP flows to get a clear view of each TCP's RTT fairness in 10Gbps high-speed optical networks. We have 2 senders, both of which send 1 same TCP flow. One flow's RTT is fixed to 120ms, while the other's is changed from 30ms to 240ms. We fix queue size to 20% BDP in all experiments of RTT fairness.

Figure 2.14 shows that CUBIC maintains very high RTT fairness up to 0.98. The reason is that Cwnd size of CUBIC W_{CUBIC} is according to a cubic function:

$$W_{CUBIC} \leftarrow C(t - \sqrt[3]{W_{max}\beta/C})^3 + W_{max} \tag{2.6}$$

where C is a scaling factor, t is the elapsed time since last window reduction, W_{max} is the window size just before the last window reduction, and β is the decrease

parameter. Compared to Equation 2.5 of other loss based TCP variants, Cwnd growth function of CUBIC is independent of RTT and therefore, CUBIC flows with different RTTs have the same Cwnd growth rate. In order to maintain an ideally flat line of RTT fairness index, like CUBIC does, could be considered as RTT fair for TCP variants.

RTT fairness indexes of TCP-SACK and HSTCP are similar between 0.75 - 0.95, while HSTCP's is little higher than TCP-SACK's. The peak in Figure 2.14 indicates that two TCP flows both have 120ms RTT, and they perform fairly.

Next, we measure the RTT fairness for heterogeneous TCP flows. We initiate simultaneously 10 TCP-SACK, 10 HSTCP and 10 CUBIC long-lived TCP flows in the bottleneck link. We fix two kinds of the TCP flows' RTT to 120ms, and change the other kind of TCP flows' RTT from 30ms to 240ms. For example, TCP-SACK's RTT fairness in heterogeneous TCP flows is calculated by fixing the RTTs of CUBIC and HSTCP flows to 120ms, and varying the RTT of TCP-SACK flows from 30ms to 240ms. As short-lived TCP flows can improve fairness, we measure RTT fairness for

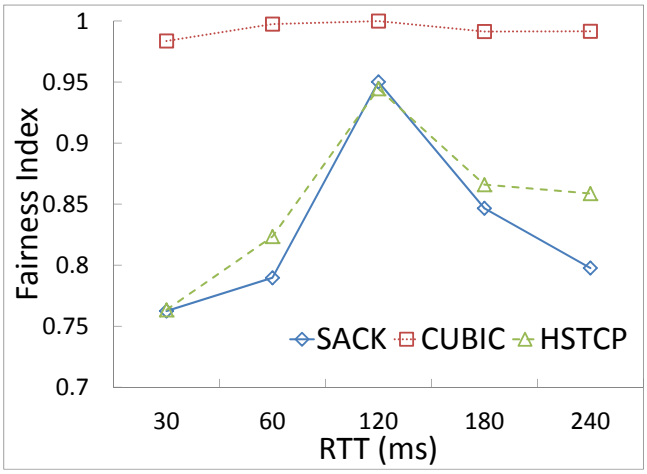
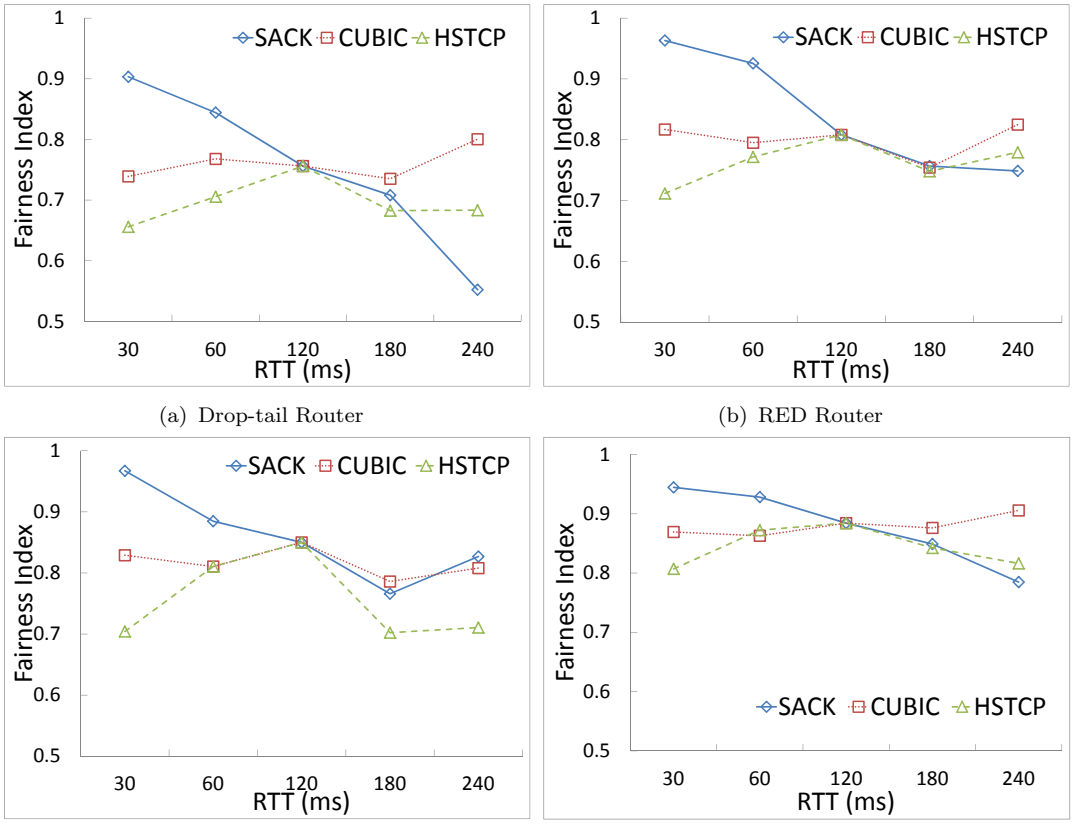


FIGURE 2.14. RTT Fairness for 2 Homogeneous TCP flows (Buffer Size = 20% BDP, RTT of one flow is fixed to 120ms, RTT of the other flow is changed from 30ms to 240ms shown on x-axis)

both with and without short-lived TCP flows scenarios to see the impact of short-lived TCP flows on RTT fairness.

Figure 2.15 shows the case without short-lived TCP flows. Figure 2.15(a) shows results for Drop-tail. If we increase the RTT of TCP-SACK flows, TCP-SACK's RTT fairness decreases linearly. It is because that TCP-SACK is a slow TCP variant. With longer RTT, TCP-SACK flows get lesser throughput. On the other hand, with shorter RTT, TCP-SACK flows get back throughput from other fast TCP variants, and improve fairness. CUBIC keeps the RTT fairness index stable to be around 0.75. Although CUBIC is RTT independent, it still experiences poor RTT fairness behavior among heterogeneous TCP flows. If we vary the RTT of HSTCP flows, the fairness



(a) Drop-tail Router

(b) RED Router

(c) CHOKe Router

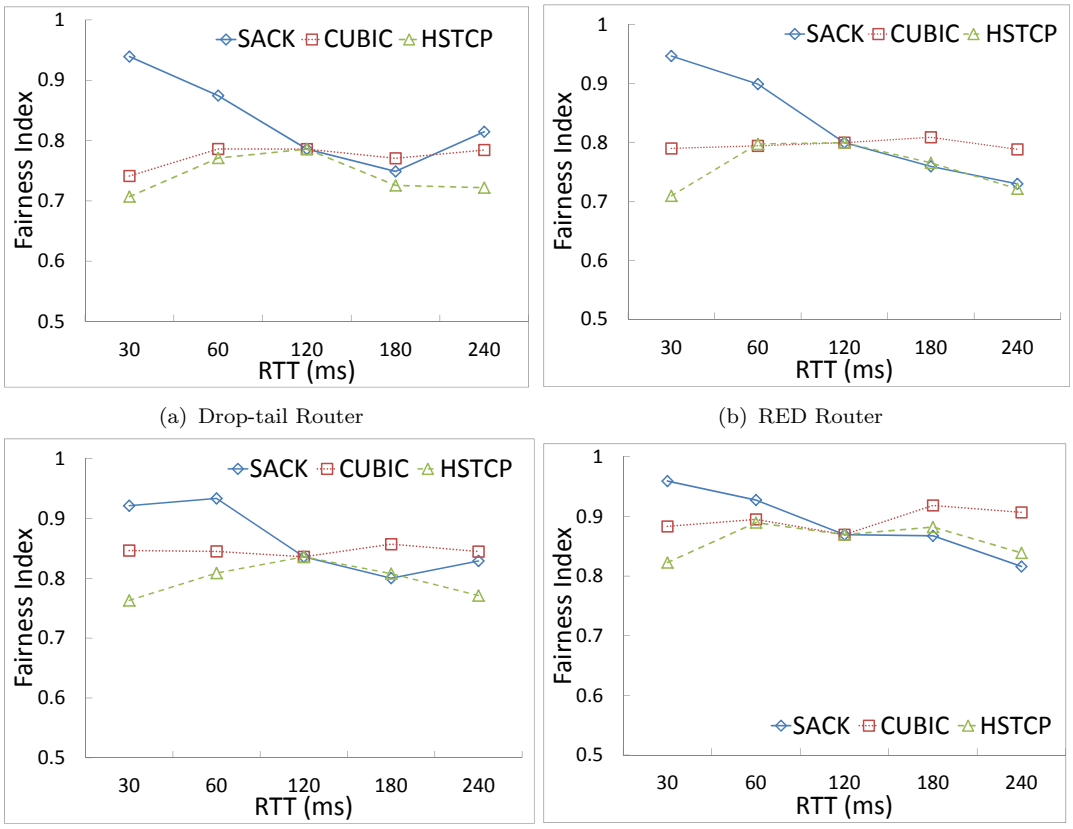
(d) AFD Router

FIGURE 2.15. RTT Fairness for Heterogeneous TCP Flows without Short-lived flows (Two kinds of TCP's RTTs are fixed to 120ms, the other TCP's RTT is changed from 30ms - 240 ms shown on x-axis. Buffer Size = 20% BDP)

index varies in a range of 0.65-0.75. HSTCP's RTT fairness is low when it has short RTTs like 60ms and 30ms. The reason is that with short RTT HSTCP flows, fast TCP variants such as HSTCP and CUBIC almost consume all of the bandwidth. Figure 2.15(b) shows RED controls the RTT fairness in a higher level than Drop-tail. RTT fairness index of TCP-SACK gets improvement up to 0.1, while RTT fairness index of CUBIC and HSTCP are both improved around 0.05. Figure 2.15(c) is for CHOKe, and its results are similar to RED. CHOKe also gets more RTT fairness than Drop-tail. Figure 2.15(d) is for AFD. AFD creates even more RTT fairness than other queue management schemes. AQM schemes again tame unfairness. Compared to RTT fairness for homogeneous TCP flows in Figure 2.14, RTT fairness for heterogeneous TCP flows becomes poor.

Figure 2.16 shows the case with short-lived TCP flows. In Figure 2.16(a) for Drop-tail, CUBIC and HSTCP get relatively stable RTT fairness index ranging from 0.7 to 0.8. TCP-SACK's RTT fairness index is from 0.75 to 0.95, which is an improvement as compare to the case without short-lived TCP flows. Figure 2.16(b) shows RED gets similar RTT fairness as Drop-tail. Figure 2.16(c) shows CHOKe gets higher RTT fairness than the other two, and it always has 0.05 more RTT fairness index for CUBIC and HSTCP than the case without short-lived TCP flows. Again, AFD in Figure 2.16(d) shows the best RTT fairness for heterogeneous TCP flows in the presence of short-lived TCP flows. In general, short-lived TCP flows improve RTT fairness.

In summary, in 10Gbps high-speed optical networks, we firstly observe that heterogeneous TCP flows induce more RTT unfairness than homogeneous TCP flows. Secondly, CUBIC maintains its RTT unawareness, while TCP-SACK performs the worst RTT fairness. Thirdly, AQM schemes tame the RTT unfairness to some extent.



(a) Drop-tail Router

(b) RED Router

(c) CHOKe Router

(d) AFD Router

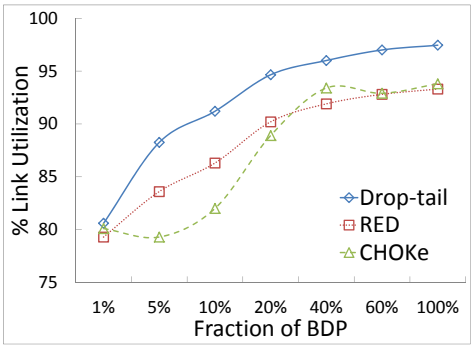
FIGURE 2.16. RTT Fairness for Heterogeneous TCP Flows with Short-lived flows (Two kinds of TCP's RTTs are fixed to 120ms, the other TCP's RTT is changed from 30ms - 240 ms shown on x-axis. Buffer Size = 20% BDP)

And finally, we confirm that short-lived TCP flows improve RTT fairness to some extent.

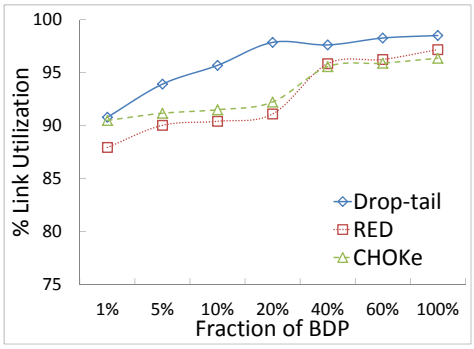
6. Link Utilization

In this section, we present link utilization results. We calculate link utilization as the percentage of total bottleneck capacity which has been utilized.

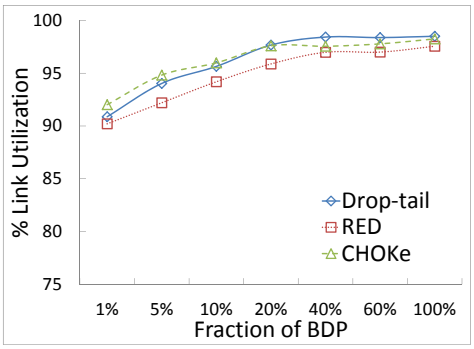
We have 3 different kinds of heterogeneous TCP flows in the bottleneck link, namely TCP-SACK, CUBIC, and HSTCP. Figure 2.17(a) shows link utilization for a single long-lived TCP flow scenario. Link utilization is improved when buffer size is increased. AFD performs the worst in terms of link utilization because AFD does a lot packet drops to ensure fairness. Drop-tail performs the best in link utilization among



(a) 3 TCP Flows



(b) 30 TCP Flows



(c) 30 TCP Flows with Short-lived TCP Flows

FIGURE 2.17. Link Utilization for Heterogeneous TCP Flows

all queue management schemes. Figure 2.17(b) shows link utilization for many long-lived TCP flows case. In 1% BDP, the link utilization is almost up to 85%. While Drop-tail still gets more throughput than other AQM schemes. Figure2.17(c) is for the case of many long-lived TCP flows with short-lived TCP flows, link utilization gets further improved as compare to the case without short-lived flow except for AFD. And Drop-tail still almost always gets the highest link utilization. There is an inevitable trade-off between fairness and link utilization for queue management schemes. AQM schemes get more fairness, while Drop-tail performs the best in link utilization.

7. Fairness: A Case for Large Size High-speed Optical Networks

In this section, we evaluate the fairness behavior for a large size high-speed optical network. According to the observation of networking flow data in next generation high-speed networks [37] (Internet2), [48] (LONI), the number of long-lived TCP flows is usually not large. Therefore, we setup a considerably large networking topology

as shown in Figure 2.18 in the high-speed optical networking environment CRON. Sender1 to Sender3 each sends 8 CUBIC flows to Receiver1, creating total of 24 CUBIC flows. Sender4 to Sender 6 each sends 8 HSTCP flows to Receiver2, creating total of 24 HSTCP flows. Sender7 to Sender9 each sends 8 TCP-SACK flows to Receiver3, creating total of 24 TCP-SACK flows. Sender1 also generates short-lived TCP flows to Receiver1.

The performance of loss-based TCP flows depends on the loss point, which is the most congested point along the sending path of TCP flows. Since Router1 to Router5 all have different levels of congestion, we have to identify the most congested link to be the bottleneck link. In the topology, since all senders send TCP flows to receivers connecting to Router4, we identify the most congested link is the link between Router5 and Router4. We set corresponding queue management schemes for our experiments at the output queue of Router5.

Fairness behavior of heterogeneous TCP flows is shown in Figure 2.19. We see that AFD still outperforms other queue management schemes in all BDP conditions. CHOKE performs the second best fairness with RED following next. With more multiplexing TCP flows, fairness behavior is generally improved in all BDP conditions

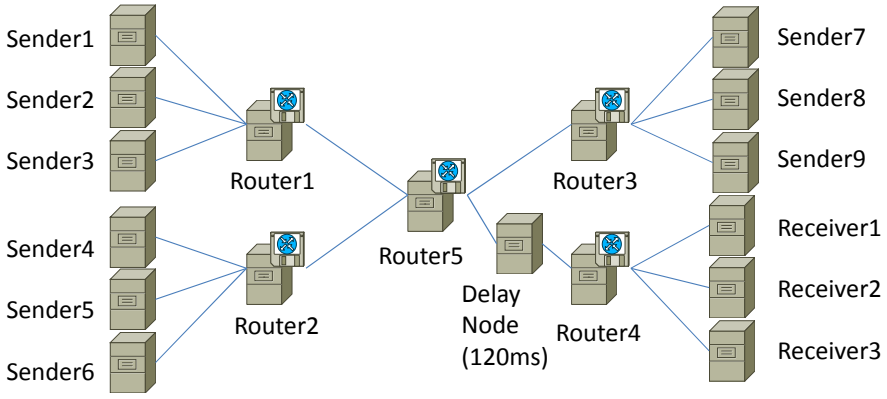


FIGURE 2.18. A Topology for a Large Size High-speed Optical Network

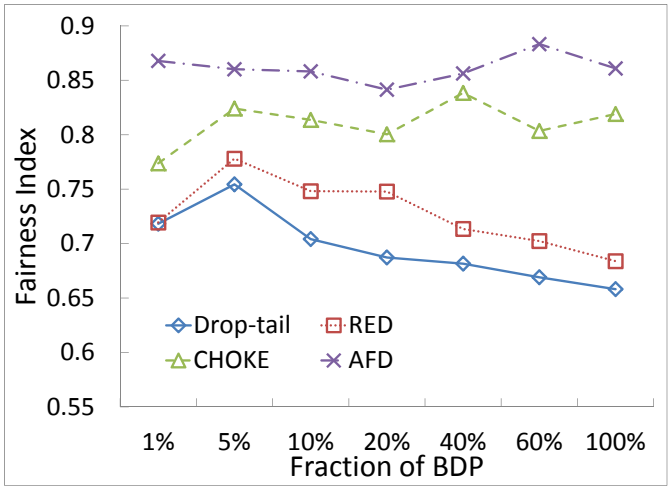


FIGURE 2.19. Fairness for a Large Size High-speed Optical Network: 24 CUBIC, 24 HSTCP, 24 TCP-SACK flows, and Short-lived flows (RTT = 120ms)

to previous experiments. The advantage of AQM schemes in terms of fairness still remains in large scale networks.

8. Discussion: Fairness for Heterogeneous TCP Variants in 40Gbps, 100Gbps and beyond High-speed Optical Networks

Assuming C is the link capacity and W is the congestion window size of a TCP flow. We know that to reach the full capacity of the link, time T will be taken and given by:

$$T = \frac{C}{W} \tag{2.7}$$

According to our explanation in Section 2.2.4, W is determined by Equation 2.5 where different TCP variants have their own parameter of α and β. From Equation 2.5 and Equation 2.7, when the system capacity grows larger, the time taken to reach the maximum system capacity becomes longer. Therefore, more aggressive TCP variants (with bigger α and smaller β) consume more and more bandwidth than less aggressive TCP variants (with smaller α and bigger β). Eventually, the system becomes more unfairness in presence of heterogeneous TCP flows. Previous study [56] presented challenges in 40Gbps high-speed optical networks to deploy all-optical routers with

very small buffer size. One of the challenges is fairness issue which remains open. As the system capacity grows larger, fairness among heterogeneous TCP variants must be addressed. This study suggests that AQM schemes somehow improve fairness issues among heterogeneous TCP flows to some extend, but fairness issues still depend on several networking conditions, such as buffer size, flow number, short-lived flows, system capacity, etc. We hope this study provides experimental data to further a step toward the deployment of all-optical routers in high-speed optical networks.

2.2.5 Summary

In this section, we present a comprehensive study of fairness among heterogeneous TCP variants over 10Gbps high-speed optical networks. We address the fairness behavior of heterogeneous TCP flows by investigating different queue management schemes with various queue sizes. In summary, fairness problem becomes severer in case of heterogeneous TCP flows. AFD performs the best in terms of fairness with CHOKe being the second, RED being the third, and Drop-tail being the worst. If buffer size in the router is small, say less than 10% of BDP, we observe similar fairness performance irrespective of queue management schemes. Also, short-lived flows improve fairness among long-lived high-speed optical TCP flows. RTT fairness also becomes poor with heterogeneous TCP traffic. The tradeoff between link utilization and fairness still exists in 10Gbps high-speed optical networks. Our experimental results confirm that fairness among heterogeneous TCP variants depends on router parameters, such as queue management schemes and buffer sizes. We believe this study sheds a light to experimental study of fairness issues in future high-speed optical networks. Through this study, we further motivate to gain more understanding about behavior of high-speed optical networks with heterogeneous TCP flows.

CHAPTER 3
AFCD: AN APPROXIMATED-FAIR AND CONTROLLED-DELAY QUEUING SCHEME FOR HIGH SPEED NETWORKS

In this section, we present the design, algorithm, and evaluation of an approximated-fair and controlled-delay queuing for high speed networks: AFCD.

3.1 Overview

We are highly motivated to address fairness and queuing delay issues in high speed networks. We propose an AQM scheme: Approximated-Fair and Controlled-Delay (AFCD) queuing that satisfies the requirement of high speed networks by providing approximated fairness and controllable low queuing delay. AFCD is a synergy of fair AQM design and controlled delay AQM design. The key idea of our novel AFCD queuing is to form an alliance between fairness based queuing and controlled delay based queuing. Proposed AQM uses a relatively small amount of state information to provide approximated fairness while ensuring very low queuing delay for high speed networks. AFCD achieves comparable throughput as other popular AQM schemes as well. Extensive evaluation of AFCD shows that AFCD performs well in a 10Gbps high speed networking testbed CRON [16].

3.2 Related Works

Drop-tail (DT) is the most used QM in commercial router nowadays. Packets are served in the order of first in first out. When DT queue is full, packets are simply dropped at the tail. However several studies have shown the limitations that imposed by DT. AQM scheme is suggested to eliminate those limitations. The authors in [34] compare AQM with DT and shows that AQM have a minor impact on the

aggregate performance metrics and AQM is sensitive to traffic characteristics that may compromise their operational deployment.

Random early detection (RED) [24] is an AQM scheme. Packets are dropped early and randomly before the queue is full. RED has two thresholds: *min* threshold and *max* threshold. When the exponentially average queue size is smaller than the *min* threshold, no packet is dropped. When the exponentially average queue size is bigger than the *max* threshold, all packets are dropped. When the exponentially average queue size is between *min* threshold and *max* threshold, packets are marked/dropped based on a probability which is increased linearly with the queue size. RED has its own limitations. Level of congestion and the parameter settings affect the average queue size and the throughput is also sensitive to the traffic load and to RED parameters. Various modifications have been suggested to improvise RED. DRED with multiple packet drop precedence to allow differentiating traffic based on priority [5], Gentle RED [58] with smooth dropping functions and many more.

Besides, totally new AQM schemes are also proposed in different studies such as BLUE [13] which uses packet loss and link idle events, instead of queue length, to manage congestion. Stochastic fair blue (SFB) [21] is an AQM for enforcing fairness among a large number of flows. SFB uses a bloom-filter to identify the non-responsive flows. The bloom-filter hashes the incoming packets to a hash value which stands for the flows. SFB maintains a mark/drop probability *pm* for each of the flows and a *qlen* which is the number of queued packets belonging to the flow. SFB has two thresholds: *max* is the maximum length of *qlen*, *target* is the desired length of *qlen*. If a flow's *qlen* is larger than *max*, packets are dropped. If a flow's *qlen* is smaller than *max*, packets are marked/dropped randomly with probability *pm*, meanwhile *pm* is adjusted to keep *qlen* between 0 and *target*. If *pm* of a flow reaches 1, the flow

is identified as non-responsive flow, and therefore SFB enters rate-limit function to rate-limit the non-responsive flows.

In next few paragraphs, we describe the two notable approaches namely, AFQ [52] and CoDel [50] which form the basis and background for our approach.

Approximate fair dropping queue (AFQ) [52] provides approximate fair bandwidth allocation among flows. AFQ makes the drop decision based on the sending rate of the flow. To estimate the sending rate of the flow, AFQ uses a shadow buffer and a flow table. The shadow buffer is used to sample the incoming packets. The flow table contains the packet count of the flow. AFQ estimates the flow's rate r_i and the fair share rate r_{fair}. r_i is estimated by m_i which is the amount of traffic from flow i during an interval. r_{fair} is estimated dynamically by m_{fair}, which is determined by:

$$m_{fair} \leftarrow m_{fair} + \alpha(Q_{old} - Q_{target}) - \beta(Q - Q_{target}) \tag{3.1}$$

where Q is the instantaneous queue length in current interval, Q_{old} is the queue length in previous interval, Q_{target} is the target queue length, α and β are the averaging parameters. The drop probability of a packet from flow i is denoted as:

$$D_i = (1 - r_{fair}/r_i)_+ \tag{3.2}$$

If $r_i < r_{fair}$, no packet is dropped. If $r_i > r_{fair}$, D_i is increased and packets are dropped based on D_i.

Recently proposed CoDel queue [50] provides extremely low queuing delay and aims to solve the Bufferbloat problem in the Internet. Unlike other AQM's complicated setting of parameters, CoDel is parameterless, and works efficiently for a wide range of scenarios. CoDel calculates the packet-sojourn time at the dequeue function and keeps a single-state variable of how long the minimum packet queuing delay has been above the *target* value. If the packet queuing delay is larger than *target* for at least

interval, a packet is dropped and CoDel's control law is set for the next drop time. The control law sets the next drop time in inverse proportion to the square root of the number of drops happened since CoDel has entered *dropping* state. If the packet queuing delay is smaller than $target$, CoDel's controller stops dropping.

In summary, all of these QM schemes aim to concentrate on some specific aspects of the network performance. However, none of these QM schemes provides fairness, very low queuing delay, and a considerable throughput at the same time for high speed networks.

3.3 Design of AFCD

In this section, we present the idea and mechanism behind Approximated-Fair and Controlled-Delay queuing.

3.3.1 Design Goals

AFCD has the following key design goals:

(1). Fairness: AFCD needs to provide approximately fair bandwidth allocation for the flows sharing a high speed bottleneck link. Here, the approximate fairness means the max-min fairness among long-lived flows.

(2). Minimal Queuing Delay: In addition to providing approximate fairness, AFCD needs to provide a very low queuing delay for high speed networks. In terms of queuing delay, AFCD needs to perform as well as CoDel does.

(3). Acceptable Link Utilization: Like other AQM schemes, AFCD drops packets early and gently, which makes some sacrifices in terms of throughput. Here, by throughput we mean the long term throughput achieved by the flows in the bottleneck link. We do not claim that AFCD can provide the same throughput performance as DT, but AFCD needs to have similar throughput performance to other AQM schemes.

(4). Simple Implementation: Our design of AFCD aims to facilitate fair share among flows while keeping queuing delay very low in high speed networking environment. According to the statistics of high speed networks [37], most of the network resources are consumed by long-lived bulk data transfer and the number of the long-lived flows is small. Thus, it is feasible for us to take an approach similar to AFQ's to estimate the flow's sending rate by using shadow buffer and flow table, which only require a very small amount of state information. When we make drop decision at dequeue function, the single-state variable of delay information is also extremely light-weighted to implement.

3.3.2 Architecture

Architecture of AFCD is presented in Figure 3.1. When a packet comes to the enqueue function of AFCD, we sample the packet based on a sample interval. As presented in AFQ [52], packet sampling is good enough for the queue to obtain enough state information to estimate the rate for long-lived flows. A shadow buffer and a flow table is used for estimating the flow's sending rate information. The shadow buffer is used to keep the sampled packets. The flow table contains the flow's packet count. The shadow buffer and the flow table do not need to be big, because the number of long-lived flows in high speed networks is small [37]. Thus, AFCD only maintains a small amount of flow state information. If the packet is sampled, the packet is hashed based on its flow information tuples. We update the shadow buffer and the flow table by the hash value. The packet is mapped into the shadow buffer, and the flow table is also updated by increasing or decreasing the flow's packet count. Therefore, AFCD gets enough state information to estimate the sending rate of the long-lived flows.

After we get the flow's approximate rate information from the enqueue function, we make the drop decision at the dequeue function based on each packet's queuing delay. We only require single-state variables for a minimum delay within an interval

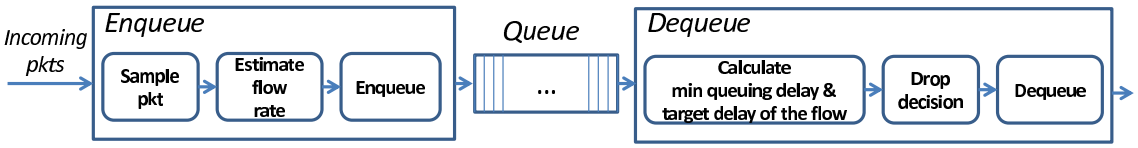


FIGURE 3.1. Functional block diagram of AFCD queuing

and a target delay for individual flow. A target delay for a flow is a threshold. Within an interval, if the minimum delay has been above the target delay of this flow, we start to make drop decision on this flow. We create formulas to calculate the target delay for individual flow based on the bandwidth share of this flow. Because AFCD makes the drop decision at dequeue function with the information of packets queuing time, AFCD controls queuing delay very accurately.

3.3.3 Design

In this section, we elaborate the design of AFCD with detailed formulas which form the dynamics between bandwidth share and latency.

With the flow table, we know flow i's bandwidth share m_i is proportional to flow i's packet count in flow table:

$$m_i \leftarrow flow\_table[i] \tag{3.3}$$

The fair bandwidth share m_f can be estimated by the shadow buffer size, flow count and an averaging value of m_f:

$$m_f \leftarrow \frac{SHADOW\_BUFFER\_SIZE}{flow\_count} \tag{3.4}$$

$$m_f(t2) \leftarrow m_f(t1) + \alpha(Q(t1) - Q_{target}) - \beta(Q(t2) - Q_{target}) \tag{3.5}$$

where Equation 3.5 is the same as in AFQ [52]. t1 is the previous time, t2 is the current time, Q is the instantaneous queue length, and α and β are the averaging parameters.

In AFCD's dequeue function, there is a $target\_delay$ that needs to be set by network operators the same as in CoDel[50]. Within an $interval$, if the minimum queuing de-

lay is higher than this *target_delay*, packets are dropped. Based on this *target_delay*, AFCD sets different target delay for different flows to approximately enforce fair bandwidth allocation among the flows. Flow i's target delay $target\_delay_i$ is a calculated value based on *target_delay* for individual flow. When the minimum packet queuing delay of flow i has been above $target\_delay_i$ for the *interval*, AFCD starts to drop on flow i. $target\_delay_i$ is calculated as follows.

The Cwnd (congestion window) W_i of a TCP flow i at time t can be modeled as [33]:

$$W_i(t) = \frac{1}{R(t)} - W_i(t)\frac{W_i(t - R(t))}{2R(t - R(t))}P(t - R(t)) \tag{3.6}$$

where R is the round trip time and P is the drop probability in the router.

Assuming Q_i is the queuing delay created by flow i in the bottleneck queue. Q_i can be calculated by:

$$Q_i(t) \approx \frac{W_i(t)}{R(t)C} \tag{3.7}$$

where C is link capacity.

The penalization (drop probability) of flow i P_i is set to be proportional to the queuing delay flow i created:

$$P_i(t) \leftarrow Q_i(t) \tag{3.8}$$

From Equation (3.6) - (3.8), the maximum sending rate of flow i W_{i_max} is bounded by:

$$W_{i\_max} \leftarrow \frac{W_i(t)^3}{R(t)^2C} \tag{3.9}$$

From Equation (3.9), the target delay of flow i $target\_delay_i$ is bounded by target Cwnd W_{i_target}:

$$target\_delay_i \leftarrow \frac{W_{i\_target}(t)^3}{R(t)^2C} \tag{3.10}$$

$target\_delay$ is determined by W_{i_max}, therefore it is bounded by:

$$target\_delay \leftarrow \frac{W_{i\_max}(t)^3}{R(t)^2 C} \tag{3.11}$$

The max sending rate of flow i is bounded by the estimated sending rate in enqueue function:

$$W_{i\_max}(t) \leftarrow m_i(t) \tag{3.12}$$

The goal of AFCD is to make fair bandwidth share, therefore the target sending rate of flow i is the estimated fair bandwidth share in enqueue function:

$$W_{i\_target}(t) \approx m_f(t) \tag{3.13}$$

From Equation (3.10) - Equation (3.13), $target\_delay_i$ is calculated as:

$$target\_delay_i \leftarrow target\_delay \times (\frac{m_i}{m_f})^{-3} \tag{3.14}$$

With this cubic kind function, we form a relationship between bandwidth share and latency. If flow's bandwidth share is equal to the fair bandwidth share, flow's target delay is the target delay. If flow's bandwidth share becomes lower than the fair bandwidth share, flow's target delay becomes much higher than the target delay, and therefore packets from the slow flows are not dropped. If flow's bandwidth share becomes higher than the fair bandwidth share, flow's target delay becomes much lower than the target delay, and therefore the fast flows are penalized.

A packet from flow i is dropped because the queuing delay has exceeded $target\_delay_i$ for at least $interval$. Then AFCD controller dequeues next packet from the queue, and sets the next drop time which is decreased in inverse proportion to the square root of the number of drops since the last dropping state. Until the queuing delay of a packet from flow i is lower than $target\_delay_i$, AFCD stops dropping.

3.3.4 Algorithm

A pseudo code of AFCD algorithm is shown in Algorithm 1. The Linux kernel code implementation can be found at [1].

In the algorithm, AFCD maintains a shadow buffer $shadow\_buffer$ and a flow table $flow\_table$.

In enqueue function, AFCD samples packet by using $sample\_packet()$. Then AFCD calculates hash value $afcd\_hash$ for sampled packets and updates $shadow\_buffer$ and $flow\_table$ correspondingly by the hash value $afcd\_hash$.

In dequeue function, AFCD calculates the target delay of flow i $target\_delay_i$ and the minimum queuing delay $minimum\_queuing\_delay$ within interval $interval$. Then AFCD makes its drop decision.

Algorithm 1 Algorithm of AFCD

```
shadow_buffer[SHADOW_BUFFER_SIZE]
flow_table[FLOW_TABLE_SIZE]

function AFCD_QDISC_ENQUEUE
    if sample_packet() then
        Calculate hash for packet, afcd_hash
        Use afcd_hash to update shadow_buffer
        Use afcd_hash to update flow_table
    end if
    do_enqueue()
end function

function AFCD_DEQUEUE
    Calculate hash for packet, afcd_hash
    Use afcd_hash and flow_table to calculate target delay of the flow i, target_delay_i
    while within interval, minimum_queuing_delay > target_delay_i do
        qdisc_drop()
        Dequeue next packet
        Schedule next drop time
    end while
    Stop drop
end function
```

[1] $https://github.com/AFCD-Linux/afcd-source$

3.4 Experimental Evaluation

In this section, we evaluate AFCD carefully, and compare AFCD to other related QM schemes such as DT, RED, SFB, CoDel, and AFQ. We conduct various experiments to emulate various scenarios in high speed networks.

We setup a dumbbell networking topology in a 10Gbps high speed networking testbed CRON [16] as shown in Figure 3.2. All servers and links in the topology have 10Gbps capacity. The bottleneck queue is the output queue of Router1, where we set the queue management schemes. Three pairs of Senders and Receivers run a modified Linux 2.6.34 kernel, and are used to transfer TCP/UDP flows by using Zero-copy Iperf [78]. Two routers run a modified Linux 3.6.6 kernel, which supports all related Linux queue disciplines including newly developed CoDel. The 10Gbps hardware emulator is used to accurately emulate propagation delay. We initially set the propagation delay in the hardware emulator to be 120ms. Queue size in the bottleneck queue is set to be 100% of bandwidth-delay product (BDP).

In CRON, we did system tunning for 10Gbps high speed networking environment as described in [76, 74]. In addition, we implement AFQ and AFCD queue disciplines in Linux kernel as well as user-space *tc* command. For AFCD, we have initial setting of parameters as CoDel [50]. $target\_delay$ is set to 5ms and $interval$ is set to 100ms. $target\_delay_i$ will be calculated by AFCD algorithm.

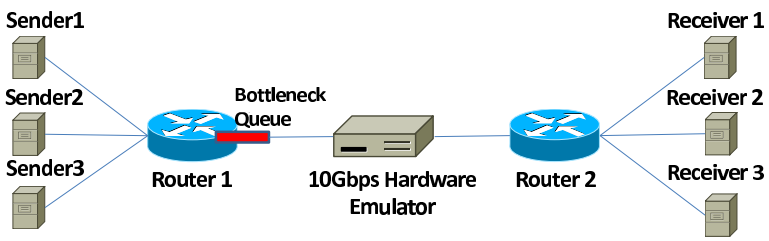


FIGURE 3.2. Dumbbell experimental topology with 10Gbps environment

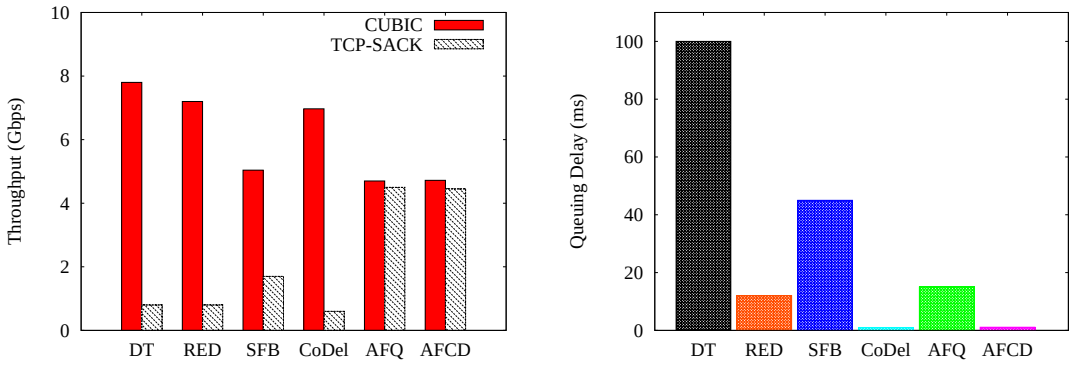
We set the parameters of queue management schemes as shown in Table 3.1. Settings of RED and SFB in 10Gbps environment are chosen according to previous studies [76, 74]. We set typical parameters for CoDel [50] and AFQ [52], with which parameters of AFCD are set accordingly.

3.4.1 1 CUBIC Flow and 1 TCP-SACK Flow

We first send 1 CUBIC flow from Sender1 to Receiver1 and 1 TCP-SACK flow from Sender2 to Receiver2. The experiment ran for 300 seconds. Figure 3.3 shows the results of throughput and delay for all QM schemes. Figure 3.3(a) shows DT, RED, and CoDel have serious unfairness when CUBIC and TCP-SACK are competing each other in a 10Gbps high speed bottleneck. SFB alleviates unfairness a little, but TCP-SACK still gets much less throughput. AFQ and AFCD show a good fairness performance, where CUBCI and TCP-SACK have almost the same throughput. Figure 3.3(b) shows the maximum queuing delay created by all the QM schemes. DT creates extremely

TABLE 3.1. Parameter setup for 6 queue management schemes

Queue	Parameter Setup
DT	queue size 100% BDP
RED	queue size 100% BDP qth_min: 0.2; qth_max: 0.8 $avpkt$: 9000; max_P: 0.02
SFB	queue size 100% BDP dropping probability $increment/decrement$: 0.00050/0.00005 Bloom filter: two 8 x 16 bins per-flow $target$: 1.5/N of total buffer (N: number of flows) maximum packets queued max: $1.2 \times target$
CoDel	queue size 100% BDP $target\_delay$: 5ms, $interval$: 100ms
AFQ	queue size 100% BDP $SHADOW\_BUFFER\_SIZE$: 2000 $SAMPLE\_INTERVAL$:500
AFCD	queue size 100% BDP $target\_delay$: 5ms, $interval$: 100ms $SHADOW\_BUFFER\_SIZE$: 2000 $SAMPLE\_INTERVAL$:500



(a) Throughput of TCP flows

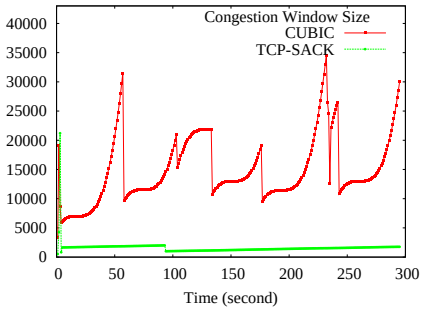
(b) Maximum queuing delay

FIGURE 3.3. 1 CUBIC and 1 TCP-SACK: performance of QM schemes when 1 CUBIC flow and 1 TCP-SACK flow are competing at the 10Gbps bottleneck, Propagation Delay = 120ms, Queue Size = 100% BDP

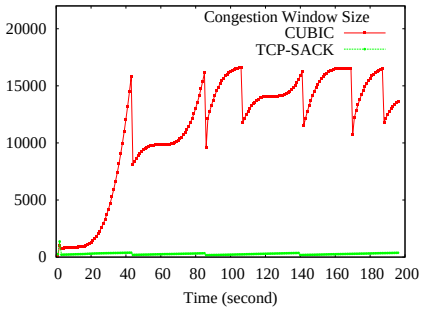
high queuing delay. RED, SFB, and AFQ all create a queuing delay larger than 15ms. CoDel and AFCD show an extremely low queuing delay compare to other QM schemes, which is less than 1ms.

We see that AFCD performs the same as AFQ in terms of fairness, while AFCD performs the same as CoDel in terms of delay. To get a clear view of why AFCD has such kind of performance, we plot out the instantaneous congestion window (Cwnd) and instantaneous RTT of DT, CoDel, AFQ, and AFCD in Figure 3.4. Figure 3.4(a) and 3.4(e) show the performance of a DT QM scheme. DT makes unfairness between CUBIC and TCP-SACK, and a very high queuing delay. Figure 3.4(b) and 3.4(f) show that CoDel makes unfairness between CUBIC and TCP-SACK, but CoDel keeps an extremely low queuing delay. 3.4(c) and 3.4(g) are for AFQ. AFQ, on the other hand, treats CUBIC and TCP-SACK fairly, but AFQ makes some queuing delay.

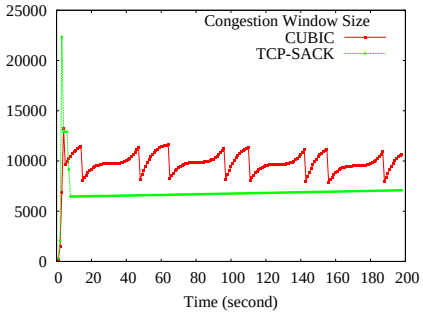
As shown in 3.4(d) and 3.4(h), AFCD makes approximated fair between two different TCP flows as well as low queuing delay. AFCD estimates the sending rate of the flows and sets different target delay for different flows based on the sending rate of the flows, and therefore both fairness and very low queuing delay are achieved. In case of AFQ and AFCD, there is a peak of bursty traffic at the beginning of the traffic,



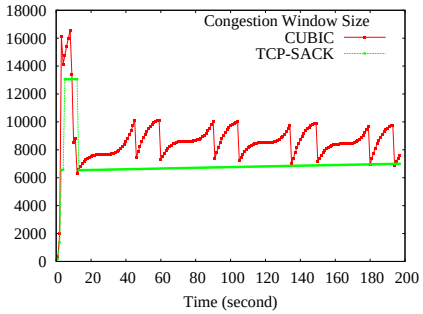
(a) DT: congestion window



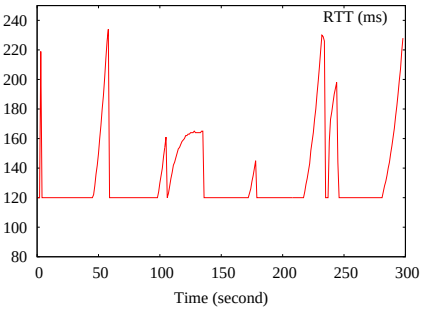
(b) CoDel: congestion window



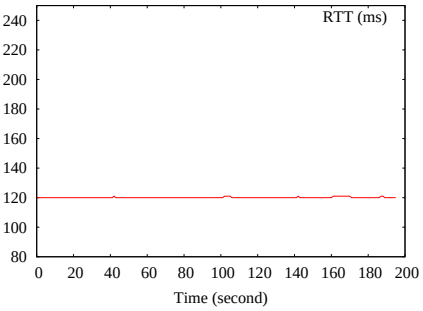
(c) AFQ: congestion window



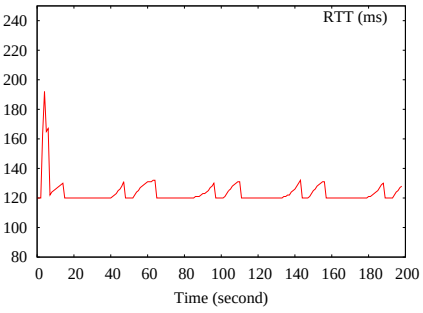
(d) AFCD: congestion window



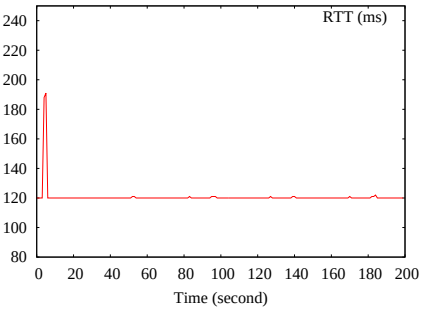
(e) DT: delay



(f) CoDel: delay



(g) AFQ: delay



(h) AFCD: delay

FIGURE 3.4. 1 CUBIC and 1 TCP-SACK: instant congestion window size and delay for different QM schemes 1 CUBIC flow and 1 TCP-SACK flow are competing at the 10Gbps bottleneck, Propagation Delay = 120ms, Queue Size = 100% BDP

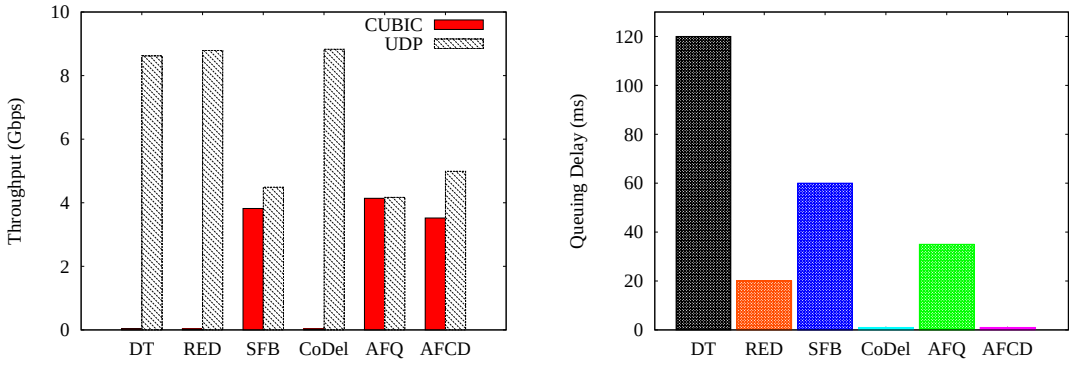
which is because that AFQ and AFCD need some time to gain the state information of the flows.

3.4.2 1 CUBIC Flow and 1 UDP Flow

Next we start 1 CUBIC flow from Sender1 to Receiver1 and 1 UDP flow from Sender2 to Receiver2 simultaneously. The experiment also runs for 300 seconds. The UDP flow is a non-responsive flow sending at a speed of 10Gbps. Figure 3.5 shows throughput and delay performance of the QM schemes. In Figure 3.5(a), DT, RED and CoDel all have serious unfairness. CUBIC flow barely gets any bandwidth, but UDP flow consumes almost all of the bandwidth. SFB, AFQ, and AFCD make fairness between CUBIC and UDP. SFB has an improvement in fairness performance compared to Section 3.4.1 because of its instinct to detect non-responsive flows. Figure 3.5(b) shows that DT, RED, SFB, and AFQ all make different degrees of large queuing delay, but CoDel and AFCD keep the queuing delay less than 1ms. This test shows that AFCD works well in the presence of non-responsive flow.

3.4.3 3 Flows Case: 1 CUBIC, 1 HSTCP, and 1 TCP-SACK Flow

In this section, we mix 3 of the most popular TCP variants [77] in the bottleneck. 1 CUBIC flow, 1 HSTCP flow, and 1 TCP-SACK flow are transfered between the 3



(a) Throughput of TCP/UDP flows

(b) Maximum queuing delay

FIGURE 3.5. 1 CUBIC and 1 UDP: performance of QM schemes when 1 CUBIC flow and 1 UDP flow are competing at the 10Gbps bottleneck, Propagation Delay = 120ms, Queue Size = 100% BDP

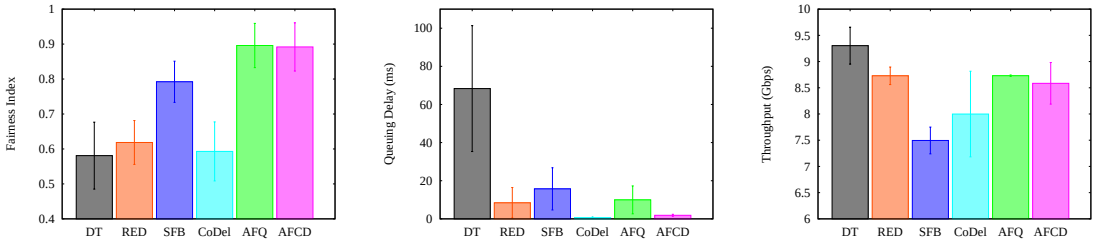
pairs of senders and receivers respectively. We run the experiments 5 to 7 times, and get the results in Figure 3.6. Figure 3.6(a) shows AFQ and AFCD get the highest Jainś fairness index, while DT, RED, and CoDel perform bad in terms of fairness. In terms of queuing delay as shown in Figure 3.6(b), CoDel and AFCD keep the delay very low, while DT gets the highest delay. Figure 3.6(c) is the result of throughput. AFCD performs almost the same as RED and AFQ.

3.4.4 Many Multiplexed Heterogeneous TCP Flows

We increase the multiplexing of long-lived TCP flows in this test. 10 CUBIC, 10 HSTCP, 10 TCP-SACK flows are mixed in the 10Gbps bottleneck link. Figure 3.7 shows the performance of the QM schemes in case of increased multiplexing. Figure 3.7(a) shows fairness performance is improved in all QM schemes because of the multiplexing. AFQ and AFCD still get the highest fairness among all QM schemes. Figure 3.7(b) is the delay results. CoDel and AFCD still create the least queuing delay. Figure 3.7(c) shows AFCD has similar throughput performance to SFB and CoDel.

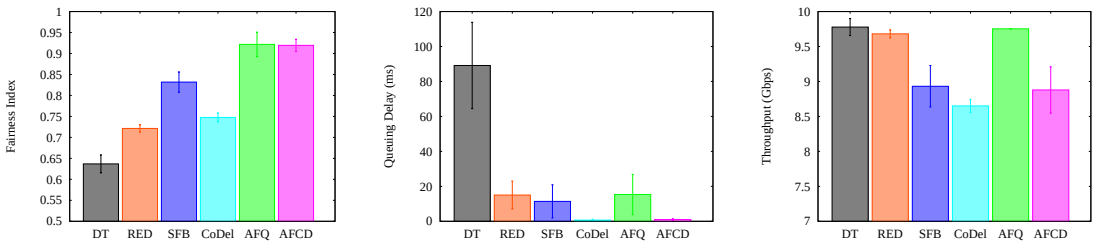
3.4.5 Reduced RTT

We reduce the delay in the hardware emulator to be 60ms in this test to see the performance of AFCD with smaller propagation delay. We get the result in Figure 3.8. Because of the reduced RTT, all performances are improved compared to that



(a) Jain's fairness index (b) Average Queuing Delay (c) Throughput

FIGURE 3.6. 3 flows case: performance of queue management schemes in 10Gbps high speed networks 1 CUBIC, 1 HSTCP, and 1 TCP-SACK flow, Propagation Delay = 120ms, Queue Size = 100% BDP



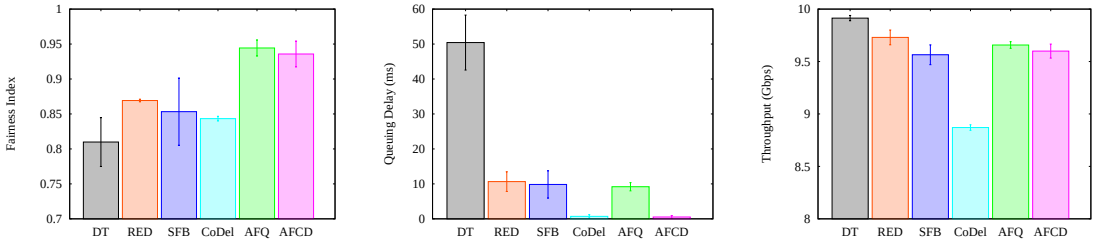
(a) Jain's fairness index (b) Average Queuing Delay (c) Throughput

FIGURE 3.7. Many multiplexed heterogeneous TCP flows: performance of queue management schemes in 10Gbps high speed networks 10 CUBIC, 10 HSTCP, and 10 TCP-SACK flow, Propagation Delay = 120ms, Queue Size = 100% BDP

of Section 3.4.4. Figure 3.8(a) shows AFQ and AFCD make more fairness than other QM schemes. Figure 3.8(b) shows CoDel and AFCD have the least queuing delay among all QM schemes. Figure 3.8(c) shows AFCD has throughput performance as good as other AQM schemes.

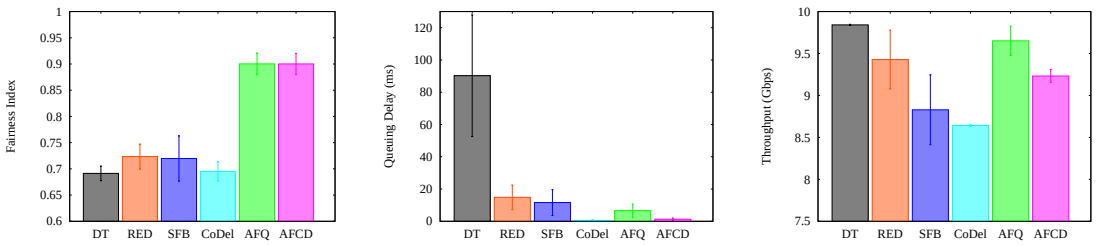
3.4.6 With Short-lived TCP Flows

To test the performance of AFCD in a more realistic high speed networking environment, we evaluate the QM schemes with both long-lived and short-lived TCP flows going through the queue. We add a pair of sender and receiver, and use Harpoon traffic generator [62] to create two-way short-lived TCP flows in the bottleneck link. The inter-connection times from Harpoon TCP client to Harpoon TCP server follow exponential distribution with 1 second of mean. The request file sizes follow Pareto distribution with alpha=1.2 and shape=1500. Figure 3.9(a) shows AFQ and AFCD



(a) Jain's fairness index (b) Average Queuing delay (c) Throughput

FIGURE 3.8. Reduced RTT: performance of queue management schemes 10 CUBIC, 10 HSTCP, and 10 TCP-SACK flow, RTT = 60ms, Queue Size = 100% BDP, Bottleneck = 10Gbps



(a) Jain's fairness index (b) Average Queuing Delay (c) Throughput

FIGURE 3.9. With short-lived TCP flows: performance of queue management schemes in 10Gbps high speed networks 10 CUBIC, 10 HSTCP, 10 TCP-SACK flow, and short-lived TCP flows, Propagation Delay = 120ms, Queue Size = 100% BDP

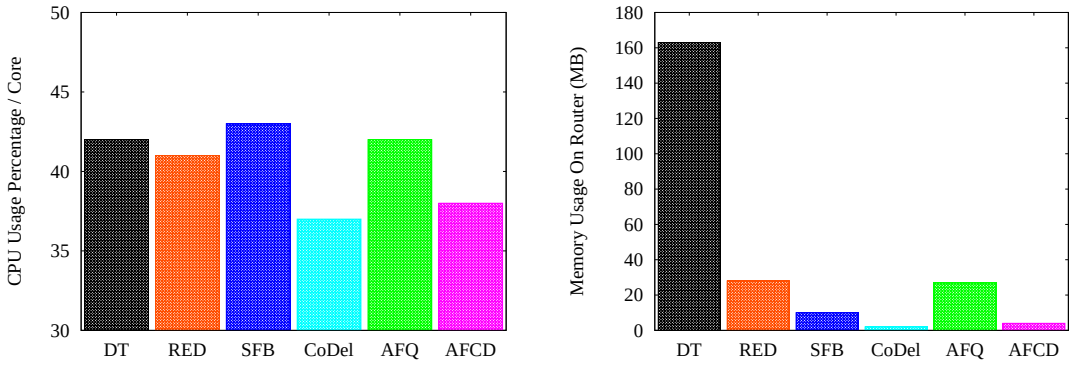
still perform better in terms of fairness. Figure 3.9(b) shows CoDel and AFCD still maintain very low queuing delay. Figure 3.9(c) shows AFCD has similar throughput performance to other QM schemes in the presence of short-lived TCP flows.

3.4.7 CPU and Memory Usage

To see the complexity and scalability of AFCD, we show the CPU and memory usage of AFCD on the router. The workstation running Linux software router is Sun Fire X4240 Server with 2x AMD Opteron Model 2384 2.7GHz quad-core processor and 8 GB (4 x2GB) DDR2-667 memory. We ran *htop* on the router and calculated the average values of CPU usage per core and memory usage within 300 seconds of experimental run. The results are based on the scenario of with short-lived TCP flows in Section 3.4.6. As shown in Figure 3.10(a), AFCD has similar CPU usage as other queue management schemes. Memory usage on the router is shown in Figure 3.10(b). AQM schemes all get much smaller memory usage than Drop-tail queue. AFCD shows very small memory usage.

3.5 Conclusion of AFCD over High Speed Networks

AFCD uses very small amount of state information of flows to approximately estimate the flows' sending rate when packets are enqueued. When packets are dequeued, AFCD uses a single-state variable to calculate a target delay of the flow and makes drop decisions for different flows based on the target delay of the flow. We evaluate



(a) CPU usage percentage per core

(b) Memory usage (MB)

FIGURE 3.10. CPU and memory usage of queue management schemes: 10 CUBIC, 10 HSTCP, 10 TCP-SACK flow, and short-lived TCP flows, Propagation Delay = 120ms, Queue Size = 100% BDP

the performance of AFCD in a 10Gbps high speed networking environment. Overall performance of AFCD is superior among its peers in various scenarios. In terms of fairness, AFCD performs as good as AFQ. Heterogeneous flows get approximated fair bandwidth share over AFCD. In terms of queuing delay, AFCD controls queuing delay to be very low, which is as good as CoDel. Also, AFCD gets comparable throughput performance to other AQM schemes.

CHAPTER 4
FALL: A FAIR AND LOW LATENCY QUEUING SCHEME FOR DATA CENTER NETWORKS

In this section, we present the design, algorithm, implementation, and evaluation of a fair and low latency queuing for data center networks: FaLL.

4.1 Overview

We propose FaLL, a novel high performance queuing scheme for data center networks that has following features that are inline with the stringent requirements on the performance of data center networks:

(1). Fairness: FaLL provides fair bandwidth sharing among competing flows by addressing issues such as TCP outcast, multi-tenant issue, etc.

(2). Low Latency: FaLL provides a very low queuing delay and addresses latency related issues in data center networks, such as TCP incast, queue buildup, buffer pressure, etc.

(3). High Link Utilization: Compare to other solutions in data center networks, FaLL also provides the same or better link utilization to its peers.

(4). Ease of deployment: FaLL is a queuing scheme for data center networks therefore, no server side change is required and it makes it easy to deploy and manage for data center operators. Through experiments we show that FaLL provides high network efficiency even with regular TCP at end servers in data center networks.

FaLL utilizes an efficiency module to ensure low latency and a fairness module to enforce approximate fairness among competing flows while achieving high throughput. FaLL drops packets based on a single state variable of target delay of individual

flows and in doing so, FaLL requires very small amount of state information keeping the memory overhead very low. Through rigorous experiments on a real high speed networking testbed, we show that FaLL outperforms its peers significantly in variety of network conditions.

4.2 Related Works

In this section, we aim to describe some most promising approaches organized in categories based on the target network locations of those solutions. We first introduce the related works and then list the pros and cons of these works.

4.2.1 End Severs and Switches

DCTCP (Data Center TCP) [2] is a congestion control mechanism for data center networks. DCTCP uses ECN from the switch to adjust the sending rate of servers. DCTCP provides a very high link utilization the same or better than traditional TCP. DCTCP maintains a very small queue size in switches, and therefore controls the latency at minimum. DCTCP requires that servers run DCTCP congestion control and switches support ECN mechanism. Widely used operating systems have not supported DCTCP yet.

HULL (High-bandwidth Ultra-Low Latency) [3] is a further step of DCTCP to trade a litter bandwidth for an extremely low latency. HULL employs DCTCP in servers and Phantom queues in switches. HULL also uses packet pacing to overcome the high burstiness. HULL requires changes of hardware or software both in servers and switches.

4.2.2 End Severs

Seawall [59] runs on a virtualization layer on servers and addresses fairness issues by a network bandwidth allocation scheme. However, the authors also stated that

Seawall alone may not maintain a low queuing delay unless corporate with DCTCP when the switch supports ECN.

4.2.3 Switches

Although most solutions described here are not specific to data center networks, for fair comparison and completeness of the proposed work, we choose to mention them

RED (Random Early Detection) [24] is an AQM (active queue management) scheme which drops or marks packets early and randomly before the queue is full. The probability of dropping or marking is proportional to queue size. RED avoids congestion in switches and maintains a smaller queue size than drop-tail queue. Some studies suggest RED also alleviates fairness issues to some extend [76], but there is still way to go to improve on latency and fairness issues.

AF-QCN (Approximate Fairness with Quantized Congestion Notification) QCN [55] is a congestion control mechanism running inside switches. AF-QCN [42] makes an extension to QCN for fair bandwidth share among multi-tenants in data center networks. AF-QCN requires no modification on QCN sources and a little modification on QCN switches. To enforce fairness, AF-QCN employs AFD (Approximate Fair Dropping) algorithm [52]. However, the latency issues have been overlooked in AF-QCN.

Codel[50] is a recently proposed AQM scheme that controls queuing delay at an extremely low level in the Internet. CoDel is parameterless and easy to deploy in switches. Codel monitors the packet sojourn time in the queue and keeps a single-state variable of how long the minimum packet queuing delay has been above a target value. If the queuing delay of a packet is longer than the target value for at least an interval, the packet is dropped. Although Codel works well in controlling delay, Codel does not aim to solve fairness issues.

PIE (Proportional Integral controller Enhanced) [54] makes random drops on congestion to control bufferbloat in the Internet. Like in Codel, PIE uses queuing delay to determine the congestion level and maintains low latency. PIE is also deployable to data center networks. However, PIE has not been extended to ensure fairness issues.

There are also several fair queuing [61] based solutions FQ_Codel [19] and FQ_PIE [54] trying to bring fairness into corresponding AQM schemes by classifying different flows into different queues. However, this requires huge amount of state information of flows, and therefore it is not scalable in large scale networks.

AFCD (Approximated-Fair and Controlled-Delay) [75] is a recently proposed AQM scheme to provide approximated fairness and controlled low queuing delay for high speed networks. The design of AFCD combines approximated fair queuing (AFD) and controlled delay queuing (Codel). AFCD uses small amount of state information to estimate the sending rate of flows. AFCD then calculates the target delay value for individual flow by estimated sending rate of the flow. If the packet queuing delay is beyond this individual target delay value, the packet is dropped. However, the design of AFCD targets at high speed networks which carry much less network traffic and dynamics than data center networks. Thus, it is critical to design a novel AQM scheme based on the traffic nature of data center networks.

4.3 Design Overview of FaLL

In this section, we elaborate the design of FaLL: a Fair and Low Latency queuing scheme for data center networks.

Figure 4.1 shows the general architecture of FaLL.

The enqueue function estimates the sending rate of a flow. When a packet comes into the enqueue function, FaLL samples the packet based on a sample interval. FaLL maintains a shadow buffer to keep the sampled packets and a flow table to

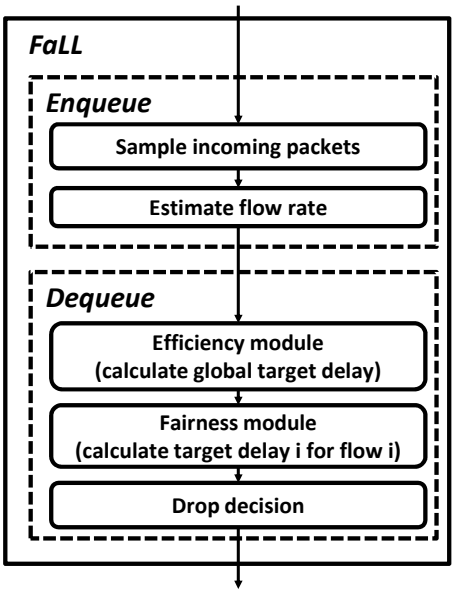


FIGURE 4.1. Functional Block Diagram of FaLL

count the flows. Recent proposals [52, 42, 75] have acknowledged that this sampling and counting method works well in various scenarios in large scale networks. Recent studies on traffic pattern in data center networks [43, 7] also presented that long-lived flows won't last for long in data center networks (e.g. more than 80% of the flows last less than ten seconds). Thus, the shadow buffer and the flow table in FaLL do not need to be big. FaLL only maintains a small amount of flow state information.

In Dequeue function, FaLL runs two detached modules: efficiency module and fairness module.

The efficiency module is used to trade minimum latency for optimally maximum throughput. Unlike the Internet or high speed networks, traffic in data center networks are heavily mixed and exhibit dynamic changes [43, 7]. With this observation, FaLL utilizes a $global\_target\_delay$ parameter which dynamically adjusts itself according to the network conditions, such as flow number, queuing delay etc. We measure the packet queuing delay accurately by recording packet enqueue time and packet dequeue time. Whenever the minimum packet queuing delay within an *interval* has been above

the *global_target_delay*, FaLL proceeds to possible packet drops (see Codel [50] and AFCD [75] papers where this approach for estimation of queuing delay works pretty well).

It is to be noted that the key difference between FaLL and Codel/AFCD is that FaLL dynamically changes the target value according to the network dynamics. This mechanism is inline with the observation that a link with N flows requires buffer sizing no more than $(C * RTT)/\sqrt{N}$, where C is link capacity and RTT is round trip time [4]. Using this minimum buffer size, FaLL estimates *global_target_delay* to achieve high throughput.

The fairness module is designed to enforce bandwidth fair share among all competing long-lived flows. After getting *global_target_delay*, FaLL computes individual target delay $target\_delay_i$ for individual flow i. Within an *interval*, if the minimum packet queuing delay of flow i has been above the $target\_delay_i$, flow i will be penalized by a dropping probability. We have formulas to calculate $target\_delay_i$ based on the estimated sending rate of flow i and *global_target_delay* (detail in Section 4.4). The formulas allow faster flows to have less $target\_delay_i$ and slower flows to have more $target\_delay_i$, and therefore FaLL enforces fair bandwidth share among flows.

4.4 Algorithm

In this subsection, we present the detailed algorithm and explain the mathematical basis behind our proposal.

4.4.1 Enqueue Function

The enqueue function of FaLL is similar to other AFD-based solutions [52, 42, 75]. Considering flow i exists in the FaLL queue, the bandwidth share of flow i is m_i. m_i is proportional to the packet count of flow i in flow table:

$$m_i \leftarrow flow\_table[i] \tag{4.1}$$

With above equation, The fair bandwidth share m_f can be estimated by the shadow buffer size, flow count and an averaging value of m_f:

$$m_f \leftarrow SHADOW\_BUFFER\_SIZE/flow\_count \tag{4.2}$$

where SHADOW_BUFFER_SIZE is the shadow buffer size, and flow_count is flow count in the flow table. Apply m_f to an averaging value [52]:

$$m_f(t2) \leftarrow m_f(t1) + \alpha(Q(t1) - Q_{target}) - \beta(Q(t2) - Q_{target}) \tag{4.3}$$

where t1 and t2 are the previous time and the current time respectively, Q is the queue length at current time, and α and β are the averaging parameters.

4.4.2 Efficiency Module

Since the network traffic in data center networks show dynamic changes [43, 7], it is impractical to set a fixed *target_delay* like in Codel and AFCD to control the delay.

Previous study on sizing router buffers [4] found when N TCP flows multiplexed together on a single bottleneck link, the buffer B needed for these long-lived or short-lived TCP flows is no more than:

$$B = (C * RTT)/\sqrt{N} \tag{4.4}$$

where C is data rate of the link and RTT is round trip time. With this minimum buffer size needed and the link rate C, we know that the minimum queuing delay D_{min} created by buffer B is:

$$D_{min} = RTT/\sqrt{N} \tag{4.5}$$

We set *global_target_delay* as D_{min}, meaning we trade this minimum queuing delay for an optimally maximum throughput. Only if the minimum queuing delay within an *interval* has been above this *global_target_delay*, FaLL proceeds to possible drops.

$$global\_target\_delay = D_{min} \tag{4.6}$$

global_target_delay is set dynamically according to the network dynamics such as number of flows, delay, etc.

4.4.3 Fairness Module

After getting *global_target_delay*, we calculate $target\_delay_i$ which is the target delay of individual flow i. Whenever the minimum packet queuing delay of flow i has been above the $target\_delay_i$ for an *interval*, FaLL may start to penalize flow i. Next we present the guidelines to set $target\_delay_i$.

Previous study [33] has summarized a behavior model of a TCP sender:

$$W_i(t) = \frac{1}{RTT(t)} - W_i(t)\frac{W_i(t - RTT(t))}{2RTT(t - RTT(t))}P(t - RTT(t)) \tag{4.7}$$

where W_i is the Cwnd (congestion window) of flow i, t is current time, and P is the drop probability.

With flow i in the bottleneck queue, we know that the queue is occupied by flow i by Q_i and Q_i can be calculated by:

$$Q_i(t) = \frac{W_i(t)}{RTT(t)} - C \tag{4.8}$$

Queuing delay created by flow i D_i is:

$$D_i(t) \approx \frac{W_i(t)}{RTT(t)C} \tag{4.9}$$

We set the penalization of flow i according to the queuing delay flow i created:

$$P_i(t) \leftarrow D_i(t) \approx \frac{W_i(t)}{RTT(t)C} \tag{4.10}$$

where $P_i(t)$ is the drop probability of flow i. From Equation (4.7) and Equation (4.10), the maximum sending rate of flow i W_{imax} is bounded by:

$$W_{imax} \leftarrow \frac{W_i(t)^3}{RTT(t)^2C} \tag{4.11}$$

Using Equation (4.11), we know the target delay of flow i which is used to get target Cwnd $W_{itarget}$ for flow i is bounded by:

$$target\_delay_i \leftarrow \frac{W_{itarget}(t)^3}{RTT(t)^2 C} \tag{4.12}$$

We calculate $global\_target\_delay$ in Section 4.4.2. $global\_target\_delay$ is determined by W_{imax}, therefore it is bounded by:

$$global\_target\_delay \leftarrow \frac{W_{imax}(t)^3}{RTT(t)^2 C} \tag{4.13}$$

We know that according to the estimation of flow sending rate in Section 4.4.1, the max sending rate of flow i is bounded by the estimated sending rate:

$$W_{imax}(t) \leftarrow m_i(t) \tag{4.14}$$

Our goal is to make fair bandwidth share, therefore the target sending rate of flow i needs to be the estimated fair bandwidth share:

$$W_{itarget}(t) \leftarrow m_f(t) \tag{4.15}$$

Using Equation (4.12) - Equation (4.15), we calculate $target\_delay_i$ as following:

$$target\_delay_i \leftarrow global\_target\_delay(\frac{m_f(t)}{m_i(t)})^3 \tag{4.16}$$

With Equation (4.16), we know that if the sending rate of a flow is equal to the estimated fair sending rate, $target\_delay_i$ is equal to $global\_target\_delay$. So this flow will not be penalized. If the sending rate of a flow is slower than the estimated fair sending rate, $target\_delay_i$ is much higher than $global\_target\_delay$, thus this slow flow will not be penalized. If the sending rate of a flow is faster than the estimated fair sending rate, $target\_delay_i$ is much lower than $global\_target\_delay$, thus this fast flow will be penalized.

A pseudo code of FaLL algorithm is shown in Algorithm 2. In Enqueue function, FaLL runs sampling method to update $shadow\_buffer$ and $flow\_table$. In Dequeue function, FaLL first estimates sending rate m_i and fair rate m_f. Then FaLL utilizes an efficiency module and a fairness module to calculate $global\_target\_delay$ and $target\_delay_i$. Finally, FaLL makes its drop decision.

Algorithm 2 Pseudo code of algorithm of FaLL

```
function FALL_QDISC_ENQUEUE
    if sample_packet() then
        Calculate hash for packet, fall_hash
        Use fall_hash to update shadow_buffer and flow_table
    end if
    do_enqueue()
end function

function FALL_QDISC_DEQUEUE
    Calculate hash for packet, fall_hash
    Use fall_hash and flow_table to estimate m_i and m_f
    function EFFICIENCY_MODULE
        Calculate global_target_delay
    end function
    function FAIRNESS_MODULE
        Use global_target_delay to calculate target_delay_i
    end function
    function DROP_DECISION
        while in interval, min_qdelay > target_delay_i do
            qdisc_drop()
            Dequeue next packet
            Schedule next drop time
        end while
        Stop drop
    end function
end function
```

4.5 Experimental Evaluation and Discussion

In this section, we present the experimental evaluation of FaLL in a data center networking environment. We compare FaLL to other congestion control mechanisms such as DCTCP. Also, we compare FaLL to other AQM schemes, such as RED, Codel,

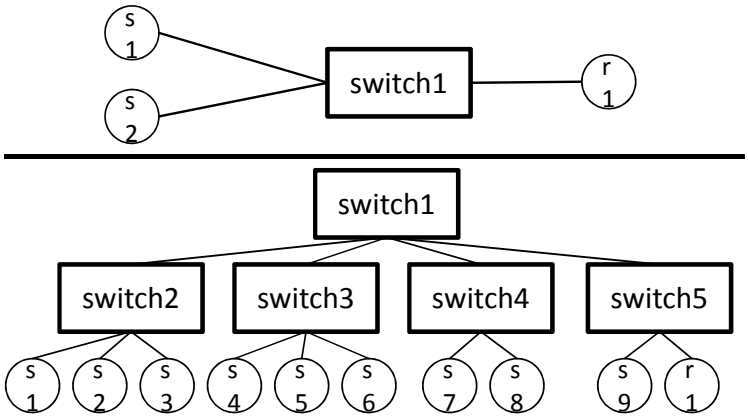


FIGURE 4.2. Experimental topologies: Y-shape (top) and Tree topology (botton). All nodes, switches, and links have 10Gbps capacity

and AFCD. We show that FaLL outperforms its peer AQM schemes in terms of both fairness and latency. FaLL also has same or better link utilization among its peers.

4.5.1 Experimental Setup

Our experimental evaluation consists of two networking topologies: a Y-shape topology and a tree topology as shown in Figure 4.2. Both networking topologies are set up in a 10Gbps high speed networking testbed CRON [16].

All servers, switches, and links in both Y-shape topology and tree topology have 10Gbps capacity [74].

The servers and switches are high-end Sun Fire X4240 machines. The servers run a Linux 2.6.38.3 kernel with a patch of DCTCP. We use Zero-copy Iperf [78] to transfer long-lived TCP flows and Harpoon traffic generator [62] to generate short-lived TCP flows. The default TCP is CUBIC except experiments of DCTCP.

The switches run a Linux 3.6.6 kernel which supports newly developed Codel AQM scheme. We implemented AFCD and FaLL modules in Linux 3.6.6 kernel as well as user-space *tc* command. The buffer size in the switches is 1MB.

Each experiment runs 5 to 7 times. We show the average results. The detailed experimental setup for all solutions is shown in Table 4.1.

TABLE 4.1. Experimental setup for all solutions

Solutions	Server Side Setup	Switch Side Setup
DCTCP	Linux 2.6.38.3 kernel with DCTCP patch; min_rto = 1ms [69]	Patched RED: both the low and high thresholds set to K (K = 20), mark packets based on instant queue length, instead of average queue length. Enabled ECN
RED	CUBIC TCP min_rto = 1ms	min = 0.2; max = 0.8; $probability$ = 0.02
Codel	CUBIC TCP min_rto = 1ms	Linux 3.6.6 kernel ; $target\_delay$ = 1ms; $interval$ = 5ms
AFCD	CUBIC TCP min_rto = 1ms	Linux 3.6.6 kernel with AFCD module; $target\_delay$ = 1ms, $target\_delay_i$ dynamic calculation
FaLL	CUBIC TCP min_rto = 1ms	Linux 3.6.6 kernel with FaLL module $global\_target\_delay$ and $target\_delay_i$ both dynamic calculation

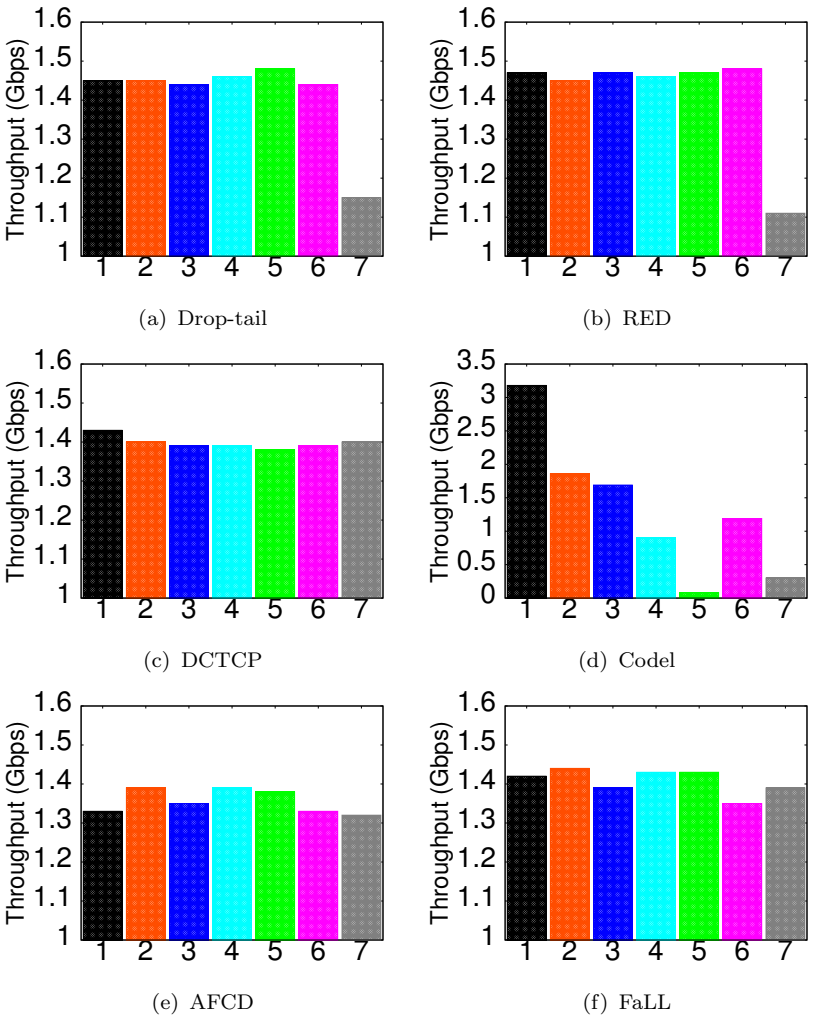
4.5.2 Results of Y-shape Topology

We first use Y-shape topology to simultaneously send 6 TCP flows from s1 to r1 and 1 TCP flows from s2 to r1. The experiment runs 10 seconds.

(1). Fairness: TCP outcast

In this scenario, a large set of flows (flow 1 to flow 6) compete with a small set of flows (flow 7). Also, we have very small TCP min_rto in senders. Therefore, TCP outcast may induce unfairness among competing flows.

Figure 4.3 shows the throughput of 7 flows. We see that Drop-tail (Figure 4.3(a)) and RED (Figure 4.3(b)) create TCP outcast problem, where small set of flows (flow 7 in this case) have throughput starvation. DCTCP (Figure 4.3(c)) solves TCP outcast problem because DCTCP maintains a very small queue size and therefore there is no port blackout [57]. Codel (Figure 4.3(d)) shows unfairness among competing flows. AFCD (Figure 4.3(e)) and FaLL (Figure 4.3(f)) show a fair behavior among competing flows, thus solve the TCP outcast problem.



(a) Drop-tail (b) RED

(c) DCTCP (d) Codel

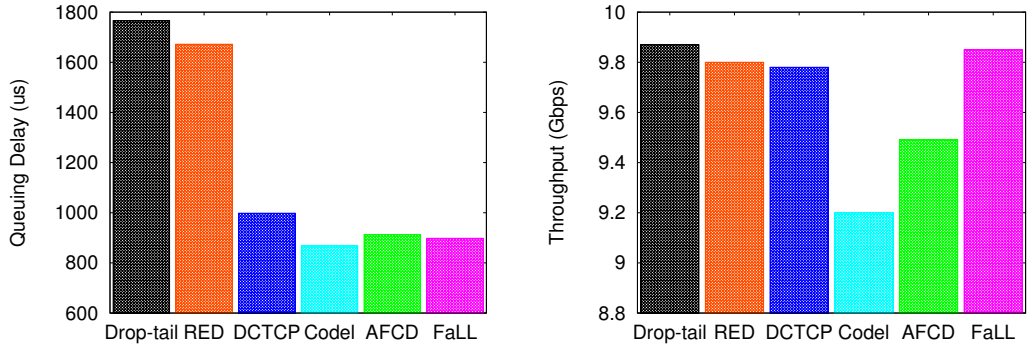
(e) AFCD (f) FaLL

FIGURE 4.3. TCP outcast performance of 7 flows (Y-shape topologies, Flow1 to Flow6 are from s1 and Flow7 is from s2)

(2). Queuing delay and throughput

Figure 4.4(a) shows results of averaging queuing delay in switch1. Drop-tail and RED create huge queuing delay (1600 - 1800 μs), whereas DCTCP, Codel, AFCD and FaLL control delay very well and have low queuing delay as 800 - 1000 μs.

Figure 4.4(b) shows total throughput of 7 flows. DCTCP and FaLL both have very high throughput (up to 9.8Gbps) similar to Drop-tail and RED. Codel and AFCD have relatively low throughput because of inappropriate packet dropping in a data center networking environment.



(a) Queuing delay in the switch

(b) Total throughput of 7 flows

FIGURE 4.4. Queuing delay and throughput (Y-shape topologies, 6 long-lived TCP flows from s1 and 1 long-lived TCP flow from s2)

4.5.3 Results of Tree Topology

In this subsection, we evaluate the performance of FaLL in tree topology where s1 to s9 send TCP flows to r1. We evaluate various scenarios including with short-lived flows, without short-lived flows, multiplexed flows, and different flow start time. We choose Jain's fairness index as fairness metric for results of fairness. Latency results is shown as average queuing delay of the switch. We also show link utilization of the bottleneck link. By default, all experiments run for 10 seconds.

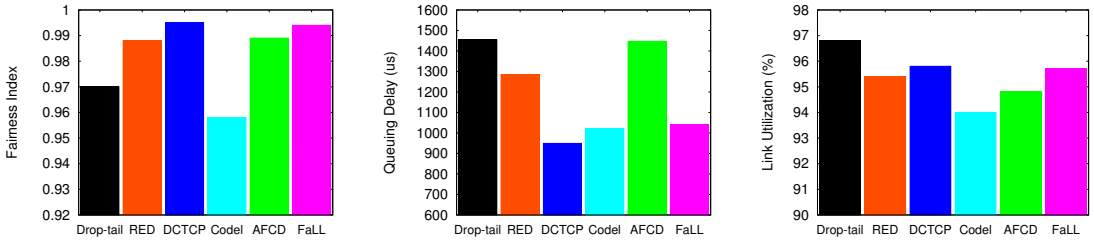
(1). Without short-lived flows

We first send 1 long-lived TCP flow each from s1 - s9 to r1 without any short-lived TCP flows. Results of fairness, latency, and link utilization is as follows:

Figure 4.5(a) shows DCTCP and FaLL get the highest fairness among 9 competing long-lived TCP flows. Figure 4.5(b) shows DCTCP, Codel, and FaLL all get very low queuing delay. Figure 4.5(c) shows DCTCP and FaLL both get high link utilization better than other AQM schemes and similar to Drop-tail.

(2). With short-lived flows

Now we send many short-lived TCP flows from s8 to r1. Other senders still each sends 1 long-lived TCP flows to r1. The size of short-lived TCP flows is Pareto



(a) Jain's fairness index (b) Average Queuing Delay (c) Link utilization

FIGURE 4.5. Without short-lived flows: Tree topology, 9 long-lived flows competing for 10Gbps bandwidth (s1 - s9 each sends 1 flow to r1)

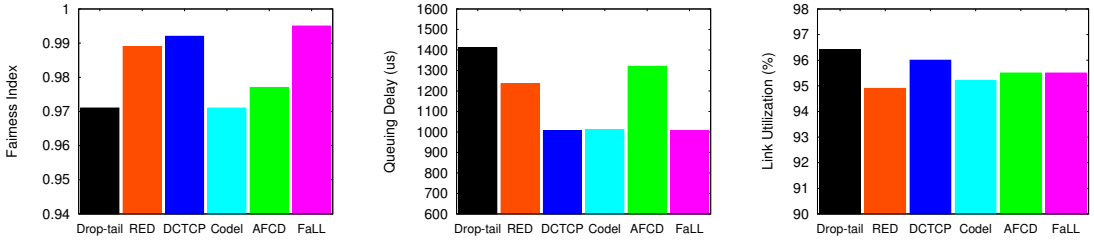
distributed with a mean of 10KB and a shape parameter of 1.1. This kind of short-lived TCP flows represent the common dynamic flows in data center networks [42].

Figure 4.6(a) shows DCTCP and FaLL again get the highest fairness. Figure 4.6(b) shows DCTCP, Codel, and FaLL still get very low queuing delay. Figure 4.6(c) shows FaLL gets similar link utilization to other peer solutions.

(3). Multiplexed flows

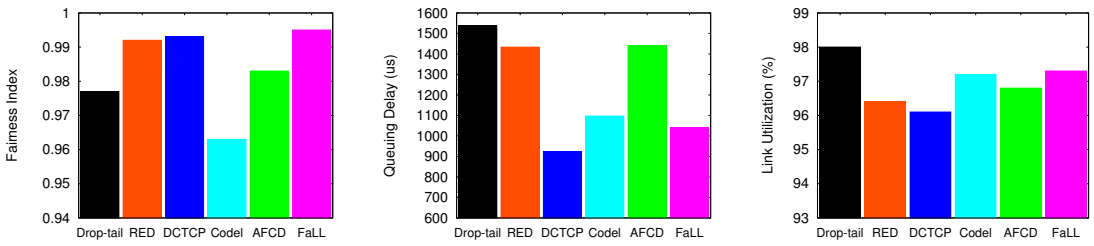
We now increase the multiplexing of long-lived TCP flows. s8 still sends many short-lived TCP flows to r1. Other 8 senders each sends 5 long-lived TCP flows to r1. So total of 40 long-lived TCP flows compete for the 10Gbps bandwidth.

Figure 4.7(a) shows DCTCP and FaLL again get the highest fairness. Figure 4.7(b) shows DCTCP and FaLL both get very low queuing delay. Figure 4.7(c) shows FaLL gets a high link utilization similar to Drop-tail.



(a) Jain's fairness index (b) Average Queuing Delay (c) Link utilization

FIGURE 4.6. With short-lived flows: Tree topology with short-lived flows, 8 long-lived flows competing for 10Gbps bandwidth (s8 sends many short-lived flows to r1, other 8 senders each sends 1 long-lived flow to r1)



(a) Jain's fairness index (b) Average Queuing Delay (c) Link utilization

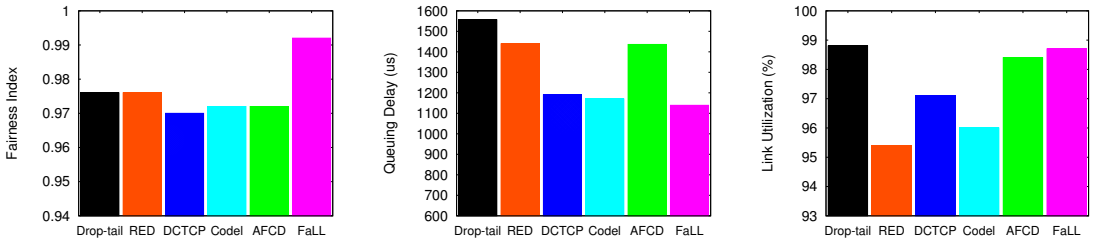
FIGURE 4.7. Multiplexed traffic: Tree topology with short-lived flows, 40 long-lived flows competing for 10Gbps bandwidth (s8 sends many short-lived flows to r1, other 8 senders each sends 5 long-lived flow to r1)

(4). Different flow start time

To increase the the dynamics like in data centers, we start long-lived TCP flows at different times. We first start s1- s4 each sending 5 flows to r1. After 10 seconds, s5, s6, s7 and s9 each sends 5 flows to r1. s8 still sends many short-lived TCP flows to r1. The experiment runs for 20 seconds.

Figure 4.8(a) shows FaLL achieves the highest fairness. Figure 4.8(b) shows FaLL gets the lowest queuing delay. Figure 4.8(c) shows FaLL gets very high link utilization similar to Drop-tail.

Because FaLL adjusts its parameters according to the dynamics of networking environment, Fall shows better performance in case of a dynamically changing networking environment. Moreover, all the single-state variables make sure FaLL remains efficient.



(a) Jain's fairness index (b) Average Queuing Delay (c) Link utilization

FIGURE 4.8. Different start time: Tree topology with short-lived flows, 40 long-lived flows competing for 10Gbps bandwidth. 20 long-lived flows start 10 seconds earlier than other 20 long-lived flows (s1 - s4 first each sends 5 flows to r1. After 10 seconds, s5, s6, s7 and s9 each sends 5 flows to r1. s8 sends many short-lived flows to r1)

4.6 Conclusion of FaLL over Data Center Networks

We present the design of FaLL, an AQM scheme to address the performance requirements of data center networks. FaLL runs inside the data center switches and requires no changes in the servers. This approach makes FaLL easy to deploy and manage. FaLL utilizes very small amount of state information to estimate flow sending rate. Through an efficiency module and a fairness module, FaLL achieves performance requirements on data center networks imposed by contemporary evolution of data center networks: low latency, fairness, and high throughput. Through experimental evaluation, we show that FaLL achieves superior performance among its peers in various traffic conditions and scenarios. In the future, we plan to implement FaLL in 10Gbps NetFPGA to improve the performance of FaLL.

CHAPTER 5
CONCLUSION AND FUTURE WORKS

In this study, we first present two experimental evaluation of the performance issues over 10Gbps high speed networks: an experimental study of the impact of queue management schemes and TCP variants on 10Gbps high speed networks and a study of fairness among heterogeneous TCP variants over 10Gbps high speed networks. Two critical performance issues are found: inter-protocol unfairness and large queuing latency.

We then propose AFCD queuing scheme which provides approximated fairness and controlled queuing delay over high speed networks. AFCD approximately estimate the sending rate of flows by maintaining very small amount of state information. When packets are dequeued, AFCD uses a single-state variable to calculate an individual target delay of the flow and makes drop decisions based on the individual target delay. Performance evaluation of AFCD over high speed networks shows AFCD is superior among its peers in various scenarios. AFCD enforces approximated fair bandwidth share with a very low queuing delay.

We also present the design of FaLL, a Fair and Low Latency queuing scheme to address fairness and latency issues of data center networks. FaLL runs only inside the data center switches and requires no changes in the servers, which makes FaLL easy to deploy and manage. By using an efficiency module, a fairness module, and a target delay based dropping mechanism, FaLL achieves following goals: low latency, fairness, and high throughput. Through experimental evaluation in a real testbed, we show that FaLL achieves superior performance among its peers in various traffic conditions and scenarios over data center networks.

In the future, we plan to optimize the algorithm of AFCD and FaLL to get better performance in high speed networks and data center networks. We also plan to implement the AFCD and FaLL algorithm in 10Gbps NetFPGA board.

REFERENCES

[1] CRON demonstration, 2011. https://www.cron.loni.org/crondemo.php/.

[2] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). *ACM SIGCOMM Computer Communication Review*, 40(4):63–74, 2010.

[3] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *Proc. of NSDI*, 2012.

[4] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 281–292. ACM, 2004.

[5] Aweya, Michel Ouellette, Abel Dasylva, and Delfin Y. Montuno. Dred-mp: queue management with multiple levels of drop precedence. *International Journal of Network Management*, 14(6), 2004.

[6] Luiz André Barroso and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 4(1):1–108, 2009.

[7] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review*, 40(1):92–99, 2010.

[8] A. Bitorika, M. Robin, and M. Huggard. An evaluation framework for active queue management schemes. In *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on*, pages 200–206. IEEE, 2003.

[9] A. Bitorika, M. Robin, M. Huggard, and C. Mc Goldrick. A comparative study of active queue management schemes. In *Proceedings of IEEE ICC 2004*, volume 201, page 6. Citeseer, 2004.

[10] L.S. Brakmo, S.W. O'malley, and L.L. Peterson. *TCP Vegas: New techniques for congestion detection and avoidance*, volume 24. ACM, 1994.

[11] M. Carbone and L. Rizzo. Dummynet revisited. *ACM SIGCOMM Computer Communication Review*, 40(2):12–20, 2010.

[12] Maurizio Casoni, Walter Cerroni, and Matteo Fiorani. Tcp performance in multi-epon access networks under different optical core switching paradigms. *Optical Switching and Networking*, 13:17–33, 2014.

[13] Wu chang Feng, Kang G. Shin, Dilip D. Kandlur, and Debanjan Saha. The blue active queue management algorithms. *IEEE/ACM Transactions on Networking (TON)*, 10(4), 2002.

[14] L. Chrost and A. Chydzinski. On the evaluation of the active queue management mechanisms. In *Evolving Internet, 2009. INTERNET'09. First International Conference on*, pages 113–118. IEEE, 2009.

[15] Cisco. Buffers, Queues, and Thresholds on the Catalyst 6500 Ethernet Modules, 2007. http://www.cisco.com/en/US/prod/collateral/switches/ps5718 /ps708/prod_white_paper09186a0080131086.pdf.

[16] CRON. CRON Project: Cyberinfrastructure for Reconfigurable Optical Networking Environment, 2011. http://www.cron.loni.org/.

[17] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, February 2013.

[18] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 205–220, 2007.

[19] Eric Duzamet. CoDel fair queuing, 2012. https://dev.openwrt.org/browser/ trunk/target/linux/generic/patches-3.3/042-fq_codel-Fair-Queue-Codel-AQM.patch.

[20] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden. Routers with very small buffers. In *Proc. IEEE Infocom*, volume 6. Citeseer, 2006.

[21] W. Feng, D.D. Kandlur, D. Saha, and K.G. Shin. Stochastic fair blue: A queue management algorithm for enforcing fairness. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1520–1529. IEEE, 2001.

[22] S. Floyd. Highspeed tcp for large congestion windows. *RFC 3649*, 2003.

[23] S. Floyd. Metrics for the evaluation of congestion control mechanisms. 2005.

[24] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *Networking, IEEE/ACM Transactions on*, 1(4):397–413, 1993.

[25] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking (TON)*, 9(4):392–403, 2001.

[26] D.A. Freedman, T. Marian, J.H. Lee, K. Birman, H. Weatherspoon, and C. Xu. Exact temporal characterization of 10 gbps optical wide-area network. In *Proceedings of the 10th annual conference on Internet measurement*, pages 342–355. ACM, 2010.

[27] GENI. Network Stitching, 2012. http://groups.geni.net/geni/wiki/GeniNetworkStitching/.

[28] J. Gettys and K. Nichols. Bufferbloat: dark buffers in the internet. *Communications of the ACM*, 55(1):57–65, 2012.

[29] S. Ha, Y. Kim, L. Le, I. Rhee, and L. Xu. A step toward realistic evaluation of high-speed tcp protocols. In *Proc. International Workshop on Protocols for Fast Long-Distance Networks (PFLD-net2006)*, 2006.

[30] S. Ha, L. Le, I. Rhee, and L. Xu. Impact of background traffic on performance of high-speed tcp variant protocols. *Computer Networks*, 51(7):1748–1762, 2007.

[31] S. Ha, I. Rhee, and L. Xu. Cubic: A new tcp-friendly high-speed tcp variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[32] S. Hassayoun and D. Ros. Loss synchronization, router buffer sizing and high-speed tcp versions: Adding red to the mix. In *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*, pages 569–576. IEEE, 2009.

[33] CV Hollot, Vishal Misra, Don Towsley, and Wei-Bo 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. IEEE, 2001.

[34] Gianluca Iannaccone, Martin May, and Christophe Diot. Aggregate traffic performance with active queue management and drop from tail. In *ACM SIGCOMM*, 2001.

[35] Internet2. Global Concert Series over Internet2, 2009. https://k20.internet2.edu/projects/100.

[36] Internet2. Internet2, 2012. http://www.internet2.edu/.

[37] Internet2. Internet2 Netflow Data, 2012. http://www.internet2.edu/research-solutions/research-support/observatory/.

[38] R. Jain, D.M. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. *DEC Research Report TR-301*, 1984.

[39] R. Jain, A. Durresi, and G. Babic. Throughput fairness index: an explanation. In *ATM Forum Contribution 99*, volume 45, 1999.

[40] S. Jain and G. Raina. An experimental evaluation of cubic tcp in a small buffer regime. In *Communications (NCC), 2011 National Conference on*, pages 1–5. IEEE, 2011.

[41] C. Jin, D.X. Wei, and S.H. Low. Fast tcp: motivation, architecture, algorithms, performance. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2490–2501. IEEE, 2004.

[42] Abdul Kabbani, Mohammad Alizadeh, Masato Yasuda, Rong Pan, and Balaji Prabhakar. Af-qcn: Approximate fairness with quantized congestion notification for multi-tenanted data centers. In *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*, pages 58–65. IEEE, 2010.

[43] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 202–208. ACM, 2009.

[44] T. Kelly. Scalable tcp: Improving performance in highspeed wide area networks. *ACM SIGCOMM Computer Communication Review*, 33(2):83–91, 2003.

[45] S. Kumar, M. Azad, and S.J. Park. A fluid-based simulation study: The effect of loss synchronization on sizing buffers over 10gbps high speed networks. In *PFLDNeT*, 2010.

[46] S. Kumar, S.J. Park, and S. Sitharama Iyengar. A loss-event driven scalable fluid simulation method for high-speed networks. *Computer Networks*, 54(1):112–132, 2010.

[47] Suman. Kumar, Lin. Xue, and Seung-Jong Park. Impact of loss synchronization on reliable high speed networks: A model based simulation. *Journal of Computer Networks and Communications*, 2014, 2014.

[48] LONI. Louisiana Optical Network Initiative, 2012. http://www.loni.org/.

[49] P. Mrozowski and A. Chydzinski. On the deployment of aqm algorithms in the internet. In *Proceedings of the 11th WSEAS international conference on Mathematical methods and computational techniques in electrical engineering*, pages 276–281. World Scientific and Engineering Academy and Society (WSEAS), 2009.

[50] K. Nichols and V. Jacobson. Controlling queue delay. *Communications of the ACM*, 55(7):42–50, 2012.

[51] NLR. National LambdaRail, 2012. http://nlr.net/.

[52] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate fairness through differential dropping. *ACM SIGCOMM Computer Communication Review*, 33(2):23–39, 2003.

[53] R. Pan, B. Prabhakar, and K. Psounis. Choke-a stateless active queue management scheme for approximating fair bandwidth allocation. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 942–951. IEEE, 2000.

[54] Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Suryanarayana Prabhu, Vijay Subramanian, Fred Baker, and Bill VerSteeg. Pie: A lightweight control scheme to address the bufferbloat problem. In *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*, pages 148–155. IEEE, 2013.

[55] Rong Pan, Balaji Prabhakar, and Ashvin Laxmikantha. Qcn: Quantized congestion notification. *IEEE802*, 1, 2007.

[56] Hyundai Park, Emily F Burmeister, S Bjorlin, and John E Bowers. 40-gb/s optical buffer design and simulations. *Numerical Simulation of Optoelectronic Devices (NUSOD)*, 2004.

[57] Pawan Prakash, Advait Dixit, Y Charlie Hu, and Ramana Kompella. The tcp outcast problem: Exposing unfairness in data center networks. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 30–30. USENIX Association, 2012.

[58] Vincent Rosolen, Olivier Bonaventure, and Guy Leduc. A red discard strategy for atm networks and its performance evaluation with tcp/ip traffic. *SIGCOMM Comput. Commun. Rev.*, 29(3):23–43, July 1999.

[59] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. Sharing the data center network. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, pages 23–23. USENIX Association, 2011.

[60] Basem Shihada, Sami El-Ferik, and Pin-Han Ho. Fast tcp over optical burst switched networks: Modeling and stability analysis. *Optical Switching and Networking*, 10(2):107–118, 2013.

[61] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round-robin. *Networking, IEEE/ACM Transactions on*, 4(3):375–385, 1996.

[62] J. Sommers, H. Kim, and P. Barford. Harpoon: a flow-level traffic generator for router and network tests. In *ACM SIGMETRICS Performance Evaluation Review*, volume 32, pages 392–392. ACM, 2004.

[63] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: a scalable architecture to approximate fair bandwidth allocations in high-speed networks. *IEEE/ACM Transactions on Networking (TON)*, 11(1):33–46, 2003.

[64] A. Tang, J. Wang, S.H. Low, and M. Chiang. Equilibrium of heterogeneous congestion control: Existence and uniqueness. *Networking, IEEE/ACM Transactions on*, 15(4):824–837, 2007.

[65] A. Tang, D. Wei, S.H. Low, and M. Chiang. Heterogeneous congestion control: Efficiency, fairness and design. In *Network Protocols, 2006. ICNP'06. Proceedings of the 2006 14th IEEE International Conference on*, pages 127–136. IEEE, 2006.

[66] A. Tang, X. Wei, S.H. Low, and M. Chiang. Equilibrium of heterogeneous congestion control: Optimality and stability. *Networking, IEEE/ACM Transactions on*, 18(3):844–857, 2010.

[67] New York Times. Traders profit with computers set at high speed, 2009. http://dealbook.nytimes.com/2009/07/24/traders-profit-with-computers-set-at-high-speed/.

[68] Balajee Vamanan, Jahangir Hasan, and T. N. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). In *Proceedings of the ACM SIGCOMM*, 2012.

[69] Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David G Andersen, Gregory R Ganger, Garth A Gibson, and Brian Mueller. Safe and effective fine-grained tcp retransmissions for datacenter communication. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 303–314. ACM, 2009.

[70] Arun Vishwanath and Vijay Sivaraman. Sharing small optical buffers between real-time and tcp traffic. *Optical Switching and Networking*, 6(4):289–296, 2009.

[71] M. Wang and Y. Ganjali. The effects of fairness in buffer sizing. *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, pages 867–878, 2007.

[72] Y. Wu, S. Kumar, and S.J. Park. Measurement and performance issues of transport protocols over 10 gbps high-speed optical networks. *Computer Networks*, 54(3):475–488, 2010.

[73] 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.

[74] Lin Xue, Cheng Cui, Suman Kumar, and Seung-Jong Park. Experimental evaluation of the effect of queue management schemes on the performance of high speed tcps in 10gbps network environment. In *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pages 315–319. IEEE, 2012.

[75] Lin Xue, Suman Kumar, Cheng Cui, Praveenkumar Kondikoppa, Chui-Hui Chiu, and Seung-Jong Park. Afcd: An approximated-fair and controlled-delay queuing for high speed networks. In *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, pages 1–7. IEEE, 2013.

[76] Lin Xue, Suman Kumar, Cheng Cui, and Seung-Jong Park. An evaluation of fairness among heterogeneous TCP variants over 10gbps high-speed networks. In *37th Annual IEEE Conference on Local Computer Networks (LCN 2012)*, pages 348–351, 2012.

[77] P. Yang, W. Luo, L. Xu, J. Deogun, and Y. Lu. Tcp congestion avoidance algorithm identification. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 310–321. IEEE, 2011.

[78] T. Yoshino, Y. Sugawara, K. Inagami, J. Tamatsukuri, M. Inaba, and K. Hiraki. Performance optimization of TCP/IP over 10 gigabit ethernet by precise instrumentation. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 11. IEEE Press, 2008.

VITA

Lin Xue was born on December 1983, in Shanghai, China. He finished his undergraduate studies in computer science at China Agricultural University, Beijing, China, in June 2005. He earned a master degree in computer science from Beijing University of Posts and Telecommunications, Beijing, China, in April 2008. He then worked as a member of technical staff 1 at Alcatel-Lucent, Beijing, China for a year and a half. In January 2010 he came to Louisiana State University, Baton Rouge, LA to pursue graduate studies in computer science. He spent summer 2012 and summer 2013 at Google, Mountain View, CA as a network engineering intern for Google Fiber and Google Platforms Networking respectively. He is currently a candidate for the degree of Doctor of Philosophy in computer science for spring 2014.