2013

Study on the Performance of TCP over 10Gbps High Speed Networks

Cheng Cui
Louisiana State University and Agricultural and Mechanical College

STUDY ON THE PERFORMANCE OF TCP OVER 10GBPS HIGH SPEED 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

School of Electrical Engineering and Computer Science

by
Cheng Cui
B.S., Xidian University, 2006
December 2013

Acknowledgments

The fulfilment of this dissertation would not have been possible without the support of many people. First, I want to thank my wife, Shaojie Qin, for her love and her continuous encouragement required to complete this dissertation. I am grateful as well to my mother Zhimin Zhao for her support. I would also like to thank the Community of South Baton Rouge Church of Christ. They give me a big support and always help me better understanding how wonderful it is to live with the Christian faith.

I would also like to thank Dr. Jay (Seung-Jong) Park, my advisor, who has guided me throughout this dissertation. Through his enthusiasm and unlimited support, he helped me to complete this dissertation. He gave me most of the major directions of this work and ensured that I actually turned implementation work into research. I appreciated very much Jay's pragmatic approach to networking for he is also an inexhaustible source of networking references. If I acknowledge all Jay's ideas, discussions, assistance, and guidance, his name will be all over this dissertation.

I would also like to thank my PhD advisory committee members Dr. Jian Zhang, Dr. Konstantin Busch from Devision of Computer Science, for the insightful discussions and guidance. Also, thanks to Dr. Peter Kelle from Department of Information Systems and Decision Sciences, for discussion and kindly offering to read my dissertation. Also my acknowledgement is extended to Dr. Jimmie Lawson from Department of Mathematics for assistance, motivation, and support.

I am grateful to all my coauthors and team members: Lin Xue, Chui-hui Chiu, Praveenkumar Kondikoppa, Suman Kumar and Georgi Stoyanov. I have had long and interesting discussions with them, and some of their writing appears in this document.

Finally, I would like to thank all the persons who supported me and helped me to have some good time here at Louisiana State University and during my dissertation in general, as well as my friends who had to face strange reactions from me when I was too involved in my work and started to be less friendly.

Table of Contents

List of Tables

List of Figures

Abstract

Internet traffic is expected to grow phenomenally over the next five to ten years. To cope with such large traffic volumes, high-speed networks are expected to scale to capacities of terabits-per-second and beyond. Increasing the role of optics for packet forwarding and transmission inside the high-speed networks seems to be the most promising way to accomplish this capacity scaling. Unfortunately, unlike electronic memory, it remains a formidable challenge to build even a few dozen packets of integrated all-optical buffers. On the other hand, many high-speed networks depend on the TCP/IP protocol for reliability which is typically implemented in software and is sensitive to buffer size. For example, TCP requires a buffer size of bandwidth delay product in switches/routers to maintain nearly 100% link utilization. Otherwise, the performance will be much downgraded. But such large buffer will challenge hardware design and power consumption, and will generate queuing delay and jitter which again cause problems. Therefore, improve TCP performance over tiny buffered high-speed networks is a top priority.

This dissertation studies the TCP performance in 10Gbps high-speed networks. First, a 10Gbps reconfigurable optical networking testbed is developed as a research environment. Second, a 10Gbps traffic sniffing tool is developed for measuring and analyzing TCP performance. New expressions for evaluating TCP loss synchronization are presented by carefully examining the congestion events of TCP. Based on observation, two basic reasons that cause performance problems are studied. We find that minimize TCP loss synchronization and reduce flow burstiness impact are critical keys to improve TCP performance in tiny buffered networks. Finally, we present a new TCP protocol called Multi-Channel TCP and a new congestion control algorithm called Desynchronized Multi-Channel TCP (DMCTCP). Our algorithm implementation takes advantage of a potential parallelism from the Multi-Path TCP in Linux. Over an emulated 10Gbps network ruled by routers with only a few dozen packets of buffers, our experimental results confirm that bottleneck link utilization can be much better improved by DMCTCP than by many other TCP variants. Our study is a new step towards the deployment of optical packet switching/routing networks.

Chapter 1
Introduction

Nowadays, it is well known that the classical congestion control algorithms used by Transmission Control Protocol [1] (TCP) protocol are not adapted to very high-speed links or, more generally, to networks buffering a large bandwidth-delay product (BDP). This is mainly due to the Additive-Increase, Multiplicative-Decrease (AIMD) behavior of TCP's congestion control. First, a TCP sender reacts to packet loss by cutting the congestion window *cwnd* by half, so the instantaneous sending rate is roughly divided by two; that is, $cwnd \leftarrow b \times cwnd$, with decrease factor $b = 0.5$. Second, the increase of *cwnd* in the congestion avoidance phase, given by: $cwnd \leftarrow cwnd + a/cwnd$, with increase parameter $a = 1$, results in a growth rate of approximately one segment per round-trip time (RTT). Indeed, if the BDP is large, *cwnd* may attain very large values before packets are lost; hence, after a loss takes place it may take the sender many RTT cycles before *cwnd* reaches again such large values.

In TCP/IP networks with bandwidth of 10Gbps or higher, one big issue is how to use link bandwidth efficiently. For a bottleneck shared by concurrent long-lived TCP flows, congestion can happen and packets can be simultaneously dropped. As a response, the TCP congestion control algorithm reduces sending rate in each flow. When many sending rates are reduced within a short time, it is recognized as a TCP loss synchronization (TCP-LS) event. As shown in Figure 1.1, at a synchronization event, these two flows are losing their shares of bandwidth at the same time so that the aggregated congestion window follows a large sawtooth pattern. Therefore, if all congestions are synchronization events, the bottleneck will be greatly underutilized.

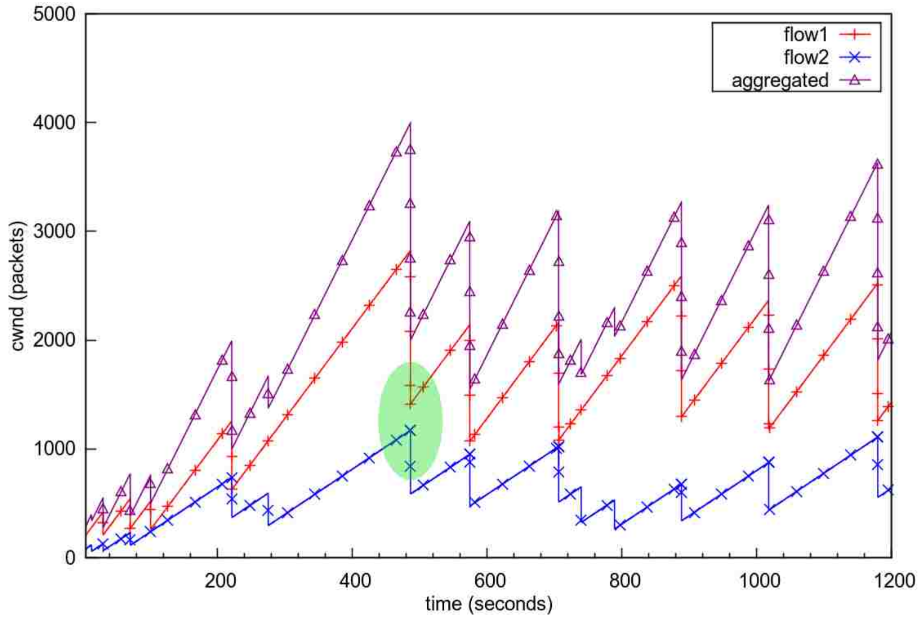


FIGURE 1.1: Demonstration of congestion windows in loss synchronization events where two TCP flows are competing a bottleneck link

TCP-LS is closely related to network parameters, such as router buffer sizes and TCP flow numbers. A recent study [2] summarizes that the buffer sizing problem is critical to link utilization over high-speed networks. Also, study [3] shows synchronization might has different behaviors with their own buffer sizes. Related works have proposed the small-buffer model [4] and the tiny-buffer models [5, 6, 7] for near 100% and 75% link utilization. However, these works rely on the traffic conditions that tens of thousands of TCP flows are neither bursty nor synchronous. These conditions only exists in high-speed core networks, where tens of thousands of TCP flows are statistically multiplexed from branch links. As shown in Figure 1.2, where the 66 packets size buffer 10Gbps bandwidth bottleneck is mixed with 60 ms, 120 ms, 180 ms and 240 ms delay TCP flows, the link utilization is greatly underutilized with a moderate number of flows using versions of TCP congestion control. Even with the most popular version, e.g. CUBIC, the link utilization is as low as around 60%.

Because large buffer size increases latency, complexity and cost, reducing router buffer size is the final goal. In all-optical packet switches/routers, traditional elec-

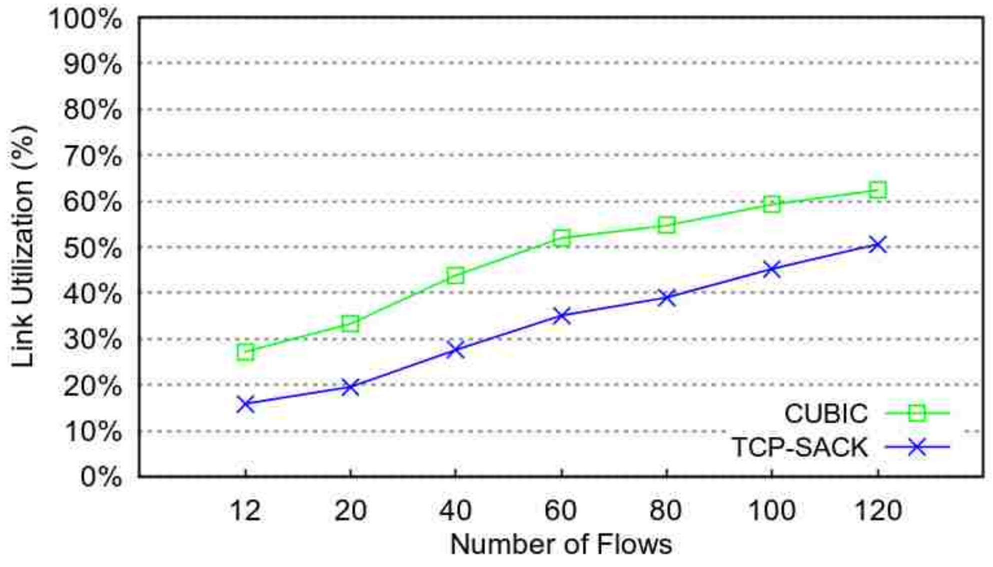


FIGURE 1.2: Link utilization of TCP variants in a 66 packets buffer high-speed network

tronic buffers are replaced with optical buffers, which are usually implemented using the optical-delay-line technique [8], and whose buffer sizes are limited to a dozen of packets [9, 10, 11]. It is shown [12] that the bursty nature of TCP makes flows experience packet drop more frequently when buffer sizes are small, and as a result, the utilization of the shared link is limited to a fairly low level.

We find it is more challenging for TCP performance in access networks with very small buffers because conditions are different in the following aspects: (i) the access links have congestion, and (ii) the network traffic is bursty, and (iii) the number of flows is too small to reach an expected level of asynchronism. We believe most of the problems come from TCP congestion control, as it is an integral module of TCP that directly determines the performance of the protocol. Therefore, to improve link utilization and to increase satisfactory bandwidth allocation for end users, how to reduce the synchronous behavior with a convenient number of flows becomes an urgent and critical issue in high-speed networks.

In a small buffer regime, study of active-queue-management (AQM) [13] points out that, random-early-detection (RED) mostly behaves like a drop-tail buffer because when the buffer is too small, it cannot absorb the large bursts due to the faster window growth inherent in TCP protocols. In another study of small buffers [14], a simple drop-tail buffer serves to enhance stability and appears preferable to the RED scheme. Therefore, counterintuitively, the effect of RED on the synchronization is negligible when the buffer is too small.

Until recently, an emerging TCP protocol called Multipath TCP has drawn great attention by its unique design of transparently transmitting packets through several mediums in the same connection. Of course, using several paths simultaneously can improve end-to-end throughput, but also it shows a good capability to manage peer-flows either through several paths or through the same path. As Multipath TCP supports standard TCP congestion control and many TCP congestion control variants for high-speed networks in Linux operating system, its modular programming interface can be easily adopted for new path management techniques like reducing peer-flows synchronization that may become available.

The goal of this dissertation is that we explore if and how we could achieve an acceptable link utilization in high-speed access networks with very small router buffers. First, it could facilitate the deploy of all-optical routers because they have huge link capacity and lower power requirement but are limited by buffer size [9]. Second, it could reduce the requirement of router complexity, making routers easier to build and easier to scale. Third, it could minimize queuing delay and jitter that were closely related to buffer size in electronic routers.

Some research assumptions are: (i) high-speed edge networks where the number of flows is small and far from tens of thousands; and (ii) very low network components (router, etc.) buffer or bufferless; and (iii) (iv) End user bandwidth requirement is high, e.g. 10 to 100 users share a 10Gbps bandwidth bottleneck as an access link, where each one takes a stake of 1000Mbps to 100Mbps bandwidth.

The adverse impact of TCP loss synchronization and traffic burstiness are: (i) it decreases link utilization. (ii) it decreases TCP fairness. (iii) it decreases bandwidth smoothing for each flow.

The small router buffer and extremely low router buffer are the final destiny for high-speed network, where active-queue-management (AQM) in such router category cannot perform as expected.

Some related solutions have problems: (i) UDP-based data transfer (UDT) [15] protocol is not TCP friendly. (ii) TCP pacing at end point [16] or at intermediate node [17, 18] has bad performance or costs too much.

In order to discover issues in high-speed networks, especially in the category of 10Gbps bandwidth, a high-speed network environment is required for the research. Because most high-speed networks are production networks and these resources are very expensive to share for researchers, my first research objective is to develop a 10Gbps high-speed network emulation testbed, from which I can study and explore the properties of high-speed networks. Second, because my focus is studying the performance of TCP over 10Gbps high-speed networks, I develop an evaluation method on TCP loss synchronization to discover how TCP loss synchronization and flow burstiness decreases the performance of TCP. Third, based on the evaluation method, I focus on the design of a new TCP congestion control algorithm called Desynchronized Multi-Channel TCP (DMCTCP), which pursues goals of minimizing TCP loss synchronization and reducing the impact of traffic burstiness.

1.1 Research Dissertation Outline

The remainder of this dissertation is outlined as follows. Chapter 2 describes TCP congestion control mechanism, design issues, challenges, and the most widely adopted TCP congestion control variants for high-speed networks. Chapter 3 describes the development of CRON, a 10Gbps network emulation testbed for high-speed network research. Chapter 4 describes the experimental study of TCP-LS and flow burstiness over TCP congestion control variants in 10Gbps high-speed optical networks. Chapter 5 describes the new TCP congestion control algorithm called Desynchronized Multi-Channel TCP (DMCTCP). Chapter 6 concludes the dissertation and shows selected directions for future research.

Chapter 2
TCP and TCP Congestion Control

Transmission Control Protocol (TCP) is the dominant protocol in modern communication networks, in which the issues of reliability, flow, and congestion control must be handled efficiently. This chapter describes about TCP, TCP congestion control, design issues and challenges in high-speed packet switching networks.

2.1 Background

Modern communication technologies use the principle of packet switching. In packet-switched networks, packets are multiplexed in network elements and processed by store and forward mechanisms. A network consists of nodes, links, and paths. Nodes can be defined as network components where the input and output links can have different characteristics. A link is a connection between two of these network nodes. A path is defined as a series of links connecting a sequence of nodes.

Protocols define the behavior required by any entity participating in the exchange of information. Communication in packet switched networks can be connection oriented or connectionless. A connection can be defined as a logical relationship between two or more endpoints that exchange data, and it is also known as virtual channel connection. A connection can be either uni-directional or bi-directional, and either point-to-point or point-to-multipoint. A flow denotes a unidirectional sequence of packets, and a session is an abstract temporary association between entities. In general, a session can include several connections, and each bi-directional connection results in at least two flows.

Traditionally, there have been two transport layer protocols in the Internet: The Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). UDP offers connectionless, unreliable transport of datagrams and is basically a multiplexing layer on top of IP.

The Internet can never be fast enough. Responsiveness continues to be one of the most important properties of Internet applications. Most Internet applications use the Transmission Control Protocol (TCP) for reliable, best effort transport. TCP is a pure end-to-end transport protocol and uses TCP congestion control (TCP-CC) in order to adapt the sending rate to the characteristics of a path.

In 1986, the Internet encountered a major crisis, the so-called "congestion collapse" [RFC 896] [19]. Congested links resulted in long delays that caused timeouts and retransmissions, which made the problem worse. By that time, TCP had flow control mechanisms only. In 1988, Jacobson proposed a congestion control scheme to be inserted into TCP [20]. His solution was easily deployable because it required changes in the TCP implementation only. As discussed in Section 2.4, the principles of Jacobsons congestion control are still in use today.

The Transmission Control Protocol (TCP) is a connection-oriented, bidirectional, point-to-point transport protocol with reliable, in-order data delivery. TCP transports a serial byte stream (a "byte-pipe") between applications and manages the recovery from erroneous, lost, or duplicate segments. In the source, the byte stream is fragmented into appropriately sized segments with a Maximum Segment Size (MSS) according to the Maximum Transmission Unit (MTU) of the path. The resulting packets are passed to the IP layer and reassembled at the destination. Figure 2.1 sketches the resulting structure of an application byte stream that is transported in a TCP segment and encapsulated in an IP packet. The TCP header, as well as the IP header, can be extended by header options. TCP options are frequently used, in particular during the initial state synchronization by the three-way handshake. TCP is a window based protocol that realizes both flow control and congestion control. These functions are detailed in Section 2.4.

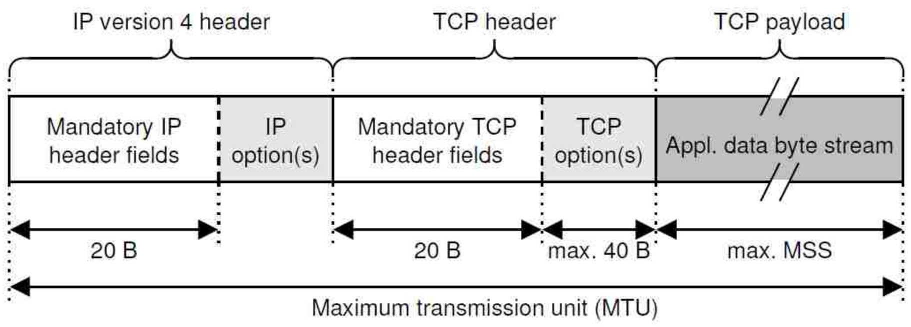


FIGURE 2.1: Structure of a TCP segment encapsulated in an IP packet, including length indications. The shaded fields are optional.

2.2 Congestion

There have been various efforts to precisely define the term “congestion”. A common statement is that “congestion occurs in a computer network when resource demands exceed the capacity” [21]. It is also common to consider a network congested if, due to overload, excessive queueing delays and/or packet losses occur. This definition is also used in this dissertation. Strictly speaking, resources can be bit-congestible or packet-congestible [RFC 6077] [22].

Due to the temporal multiplexing, short term load imbalances are unavoidable in packet networks and have to be corrected by buffering. If a resource gets congested, queueing delays increase, and packets must be dropped if the buffer size is exceeded. Without appropriate countermeasures, a congestion collapse can occur, i. e., resources are either wasted by unnecessary retransmissions or by packets that are dropped before reaching their destination as defined in RFC 2914 [23].

Congestion has to be avoided because it increases delays and wastes resources. The objective of congestion control is to minimize the intensity, spread and duration of congestion. This requires two different functions: First, congestion control should prevent a source from sending data that will get dropped on the path; this aspect is also called congestion avoidance [24]. Second, it must ensure that a network remains operational when congestion occurs and react accordingly. As the name implies, congestion control is a control mechanism. Since the entity that governs the resource usage is not necessarily identical to the resource that gets congested, congestion control is an inherently distributed problem that requires some form of feedback and a closed control loop. Therefore, congestion control can precisely be defined as "the feedback-based adjustment of the rate at which data is sent into the network" [25]. In order to avoid and handle overload situations, congestion control mechanisms must be able to decide on the usage of resources at least to some extent. Thus, congestion control can also be understood as an "algorithm to share network resources among competing traffic sources" [RFC 2914] [23].

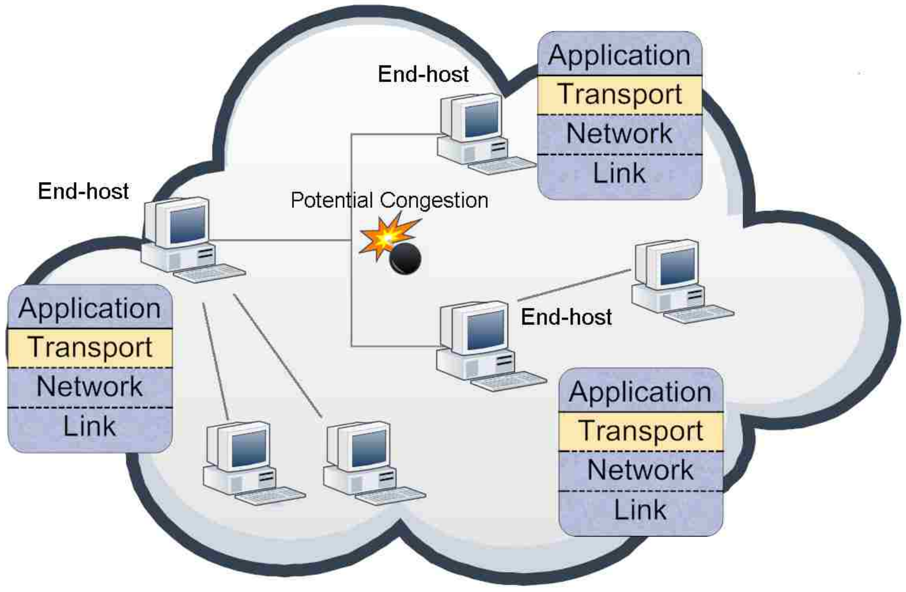


FIGURE 2.2: Congestion in a network caused by competing connections

As shown in Figure 2.2, congestion and its resolution may affect different entities. This inherently results in fairness issues, which are discussed in the next sections. It must also be emphasized that resource management in packet networks is performed by several control loops on different time scales. The reaction time of congestion control depends on RTT of the path and is of the order of milliseconds. This is the main difference compared to other traffic management and traffic engineering mechanisms, such as routing policies or capacity dimensioning, which operate on longer time scales and often also depend on human interaction. Further control loops may also exist inside applications, e. g., by application adaptation functions or by manual user reactions. Such mechanisms typically work on time-scales longer than the path RTT, too.

2.3 Congestion Control Requirements and Design Space

2.3.1 Fundamental Requirements

In general, congestion control algorithms have to satisfy the following major requirements:

- Efficiency: The utilization of the available network resources should be high.
- Responsiveness: The algorithm should respond promptly to changes in the congestion conditions and transient events such as route changes or mobility events. The convergence time to reach the operating point should be small.
- Avoidance of heavy congestion and synchronization: Dropping many packets during congestion events should be avoided. Congestion events may last longer than one RTT.
- Network independence: The protocol should work well regardless of network characteristics, such as router buffer sizes, queue management strategies, or the path MTU. Packet networks encompass a large variety of heterogeneous networks that are realized by a multitude of technologies, which result in a tremendous variety of link and path characteristics: The link capacity can be either scarce in very slow speed radio links (several kbps), or there may be an abundant supply in high-speed optical links (several gigabit per second).

Concerning latency, scenarios range from local interconnects (much less than a millisecond) to certain wireless and satellite links with very large latencies (up to a second). As a consequence, both the available bandwidth and the end-to-end delay may vary over many orders of magnitude, and they can be subject to substantial changes within short time frames. Congestion control mechanisms must also be able to deal with asymmetric routing, i. e., situations in which the forward path and the reverse path are different and potentially both congested.

- Application independence: Congestion control has to deal with quite diverse application sending behaviors. The amount of data that an application may send varies over many orders of magnitude, and the arrival pattern may be arbitrary.

- Robustness and stability: The mechanisms should be robust against noise in the congestion signals. Oscillations should be avoided. Congestion control can be viewed as a classic negative-feedback control problem with delayed feedback signals. Congestion control aims at asymptotic stability, i. e., it should converge to a certain state irrespective of the initial state of the network.

- Scalability: The mechanism must work in a global network that interconnects potentially billions of endsystems. This requires decentralization. With the currently available technology it is impossible to realize a centralized, per-flow resource management on global scale, even if the corresponding business and legal aspects would be solved.

- Simplicity: The implementation complexity and the amount of state in endsystems should be moderate. Per-flow state in core network components should be avoided, since it can hardly be realized with existing technology. Simple solutions are also more likely to become a widely accepted standard.

- Ability to deal with uncooperative entities: Any solution in a multi-domain environment must consider potentially untrusted or malicious sources, sinks, and network entities on the path, as well as outside attackers.

2.3.2 Fairness

A further requirement for resource sharing is fairness. A definition of fairness is non-trivial, since it affects both technical and economic aspects. Numerous fairness concepts have been proposed [26]. The most well-known metric to quantify fairness is Jains fairness index [24]:

$$FI = \frac{(\sum_{i=1}^{n} x_i)^2}{(n \sum_{i=1}^{n} x_i^2)} \in (0,1] \tag{2.1}$$

An allocation with equal values is characterized by $FI = 1$, whereas a totally unequal allocation has a fairness index of $FI = 1/\text{n}$.

Due to the design of the standard TCP algorithms, TCP compatibility inherently implies RTT unfairness, i. e., connections with a shorter RTT will in average obtain a higher share of bottleneck bandwidth. Also, there can be unfairness with respect to packet sizes [22].

The TCP compatibility is subject to ongoing debates in research and standardization communities. It is accepted that in high-speed networks new congestion control algorithms may be moderately more aggressive than standard TCP. But there is no consensus whether RTT fairness is a desirable design goal. There is also disagreement about the right granularity of fairness. The common goal of widely deployed mechanisms is equal bandwidth allocation among flows. However, this flow rate fairness is biased towards users that use many parallel flows. Briscoe [27] argues in favor of cost fairness, which takes into account the amount of congestion caused by a user. Floyd et al. [RFC 5290] [28] disagree and state that some form of rough flow rate fairness is an appropriate goal for simple best-effort traffic.

Fairness is also an economic issue. Most congestion control schemes require the involved parties to behave in a cooperative way. However, for an individual user it is not necessarily the optimal strategy to reduce the sending rate upon detection of congestion. If users act in a selfish manner and try to improve their own position by using more resources, this will result in a tragedy of common problem. In economics, there are three fundamental approaches to deal with congestion externalities: Social norms, rationing, and pricing. The first two reflect the current situation in the Internet, where most endsystems use congestion control and there are some network fairness enforcement mechanisms such as fair queueing and scheduling. The latter approach would require congestion-based charging, which could theoretically be a basis for congestion control. However, due to the unpredictability of such mechanisms there are serious reservations both by customers and by network operators [29].

2.3.3 Classification of End-to-End Congestion Control Algorithms

Without network support, endsystems can detect congestion only by two signals: Packet loss and/or delay. Loss-based congestion control interprets lost packets as a signal for congestion and reacts by reducing the sending rate. This corresponds to a binary feedback model. A fundamental drawback is that packet losses should be rare events and therefore provide a coarse information only. Loss-based congestion control saturates network buffers unless active queue management schemes are used, which are introduced in Section 2.3.5. An alternative mechanism are delay-based congestion control algorithms that use the delay as primary congestion signal. They determine the minimum RTT and interpret increasing delays as a congestion signal. Delay can be measured more frequently and with a finer granularity than loss. Delay-based schemes can detect incipient congestion before buffers overflow. But they have to cope with two problems: The noise in packet delays has to be filtered out, and it is inherently difficult to distinguish between full and empty queues. As a consequence, realistic delay-based algorithms require a loss-based component, too. The most widely used class of congestion control mechanism is sender-oriented, end-to-end, and loss-based. This class controls the sending rate by modification of a Congestion Window (*cwnd*). A control law increases the Congestion Window if there is no sign of congestion, and it decreases it when packet loss is detected. Chiu and Jain [30] developed a fundamental set of algorithms that manipulate a *cwnd* (W) as a reaction to the binary feedback:

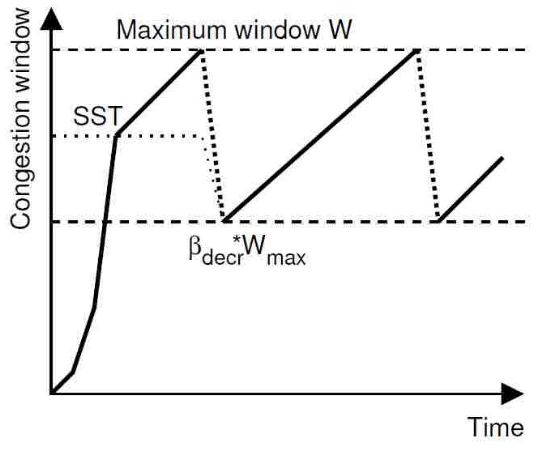
$$\text{Increase per packet in absence of congestion: } W \leftarrow W + \alpha_{incr} W^{i} \tag{2.2}$$

$$\text{Decrease on detection of congestion: } W \leftarrow W - \beta_{decr} W^{j} \tag{2.3}$$

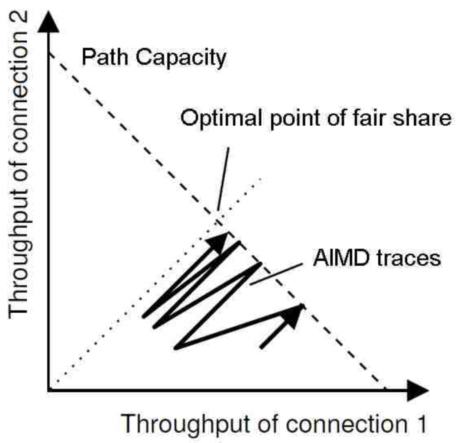
The most important class of algorithm uses $i = 1$ and $j = 1$, which corresponds to Additive-Increase Multiplicative-Decrease (AIMD). The term additive is used since the *cwnd* is increased by the additive term α_{incr} after one RTT. The resulting evolution of the *cwnd* over time is depicted in Figure 2.3(a). A fundamental property of AIMD is that it converges to the optimum point of equal sharing if several competing flows share a bottleneck [30]. This characteristic can be proved by regarding the system transitions as a trajectory through a vector space, which is illustrated in Figure 2.3(b) for the case of two flows. Obviously, there are also several alternatives to the AIMD control law [30], such as Multiplicative-Increase Multiplicative-Decrease with i = 0 and j = 1. These other decision-making functions may not converge to equal sharing in drop-tail networks [30].

2.3.4 Classification of Network-Supported Congestion Control Algorithms

Network components can be involved in congestion control in two ways: First, they can implicitly optimize their functions in order to support the operation of an end-to-end congestion control, e. g., by queue management and scheduling strategies, as introduced later in Section 2.3.5. Second, network components can participate in congestion control via explicit signaling mechanisms. A wide variety of terms are used to



(a) Simplified Congestion Window evolution of an AIMD algorithm



Throughput of connection 2
Path Capacity
Optimal point of fair share
AIMD traces
Throughput of connection 1

(b) AIMDs convergence to fairness

FIGURE 2.3: TCP's AIMD properties

describe congestion control with explicit feedback from network components, including terms such as "router-assisted", "router-supported", "router-aided", "explicit" congestion control, etc. They are not consistently used, and the term "router" is misleading since some schemes do actually require support in all queues along a path, which may also exist in link layer devices. We need to precisely distinguish between the following three terms:

- Network-supported congestion control: Network-supported congestion control schemes use explicit feedback from network components to the source of a flow. The feedback signals state of congestion.

- Network-assisted congestion control: This class of network-supported congestion control mechanism leaves the decision on sending rates to the sources. It can operate even if the sources use another congestion control mechanism.

- Network-controlled congestion control: In this class of network-supported congestion control, the network components control the sending rate of flows with a fine granularity. Sources are merely responsible for executing the control decisions.

These definitions use the term “network component” instead of “router”. All network-supported congestion control schemes require a communication between network components and endsystems. Since interconnection in the Internet is realized at the IP layer, signals can only be transported within the IP layer or in higher protocol layers. Only network components that process IP packets can trigger such notifications. The following sections distinguish clearly between the terms “network component” and “router”; the term “router” is used whenever the processing of IP packets is explicitly required. One fundamental challenge of network-supported congestion control is that typically not all network components along a path are routers [22]. The focus of network-supported congestion control is the improvement of the resource sharing in networks that offer mainly a best effort service. Unlike network QoS mechanisms, network supported congestion control mechanisms are lightweight. They do not provide guarantees, but they also do not require per-flow state in network components.

2.3.5 Interaction with Network Middleboxes

TCPs AIMD strategy is designed to fill the buffer in front of the bottleneck. Therefore, the buffer size in network components is a crucial factor. Buffer sizing is a multi-criteria optimization problem with at least three objectives: First, buffers are required in packet networks in order to absorb short-term traffic bursts. Such transient bursts are an inherent characteristic of window-based protocols. Second, buffers must be large enough to ensure that the link utilization is high, in particular for flows using an AIMD congestion control. But, third, buffers must not be too large, since they can result in persistent queueing delays and implementation costs. The dimensioning of router or switch buffers has long been considered a “black art”. Historically, the size is determined by the bandwidth-delay product as rule-of-thumb [31]. This guideline states that the buffer size B (here, in bit) should be equal to the capacity r of the outgoing link multiplied by the RTT τ of a connection that may be bottlenecked at that link. The rule prevents throughput underflow if one TCP connection with β_{decr} $= 1/2$ traverses this link.

Newer research results suggest that the buffers of network interfaces can be made much smaller if the number of flows is sufficiently large. Appenzeller et al. [4] argue that a buffer size $r \cdot \tau/\sqrt{n}$ is sufficient to saturate a link when n independent, long lived and not synchronized TCP connections share a bottleneck. According to this model, metro and core routers with a large number of flows n need interface buffers much smaller than the worst-case BDP. BDP-sized buffers are only useful if the number of flows $n \ll 100$. Newer research results argue in favor of further reducing buffer sizes and recommend buffers between 20 and 50 packets for core routers[2]. Reference [2] also comprehensively surveys other recently proposed buffer sizing strategies. Still, there is no universal design guideline for buffer sizing so far. In general, larger buffers tend to trade off a potential increase of throughput against larger delays and jitter. An optimal buffer size can, if at all, only be derived for a specific network topology, congestion control algorithm, and application workload, and it will not be optimal in other scenarios.

2.4 State-of-the-Art of TCP Congestion Algorithms

2.4.1 TCP Reno

Jacobson's ground-breaking control algorithms [20] increase the send rate until congestion is detected by packet loss, and then reduce the rate. The objective is to achieve an isarithmic equilibrium [32] in which the number of packets in the path is approximately constant. This conservation of packets principle can easily be realized by a sliding window.

The original TCP standard only specified flow control mechanisms. [RFC 793] [33] mandates the sender to use a sliding window mechanism with a maximum size given by the Receive Window (*rwnd*), i. e., the most recently advertised receive window.

Jacobson introduced a second Congestion Window (*cwnd*), which is an estimation of how much data can be outstanding in the network without packets being lost. A TCP sender can transmit up to the minimum of the *cwnd* and *rwnd*. The control algorithm published in [20] distinguished between two different phases: Slow-Start (SS) and Congestion Avoidance (CA). Later, Jacobson proposed an improved algorithm that became known as TCP Reno and that is standardized in [RFC 2581] [1]. Reno is still based on Additive-Increase Multiplicative-Decrease (AIMD). It distinguishes the following four different phases that are partly illustrated in previous Figure 2.3:

- Slow-Start: At the beginning of a transmission into a network with unknown conditions, the Slow-Start algorithm is used to probe the network and to determine the available bandwidth. After the connection setup, the size of Congestion Window W is set to the initial window w. In Slow-Start, the sender may increment W by at most MSS bytes for each received ACK that acknowledges new data. The Slow-Start ends when W reaches or exceeds the Slow-Start Threshold (SST), or when congestion is observed.

- Congestion Avoidance: When W is equal or larger than the SST, W is incremented by one full-sized segment per RTT. This phase continues until congestion is detected.

- Fast Retransmit and Fast Recovery: The sender can guess that a packet has been lost when there are duplicate acknowledgments. By default, the arrival of three duplicate ACKs triggers a Fast Retransmit. Then, the SST is set to approximately half of the flightsize, i. e., the amount of outstanding data. W is set to the same value plus three MSS. After the Fast Retransmit follows the Fast Recovery phase until the loss recovery ends.

- Retransmission Timeout (RTO): If the RTO expires, the SST is also set to approximately half of the flightsize. W is set to one segment, and the sender continues in Slow-Start.

These algorithms continuously probe the available bandwidth and correspond to an AIMD congestion control with $\alpha_{incr} = 1$ and $\beta_{decr} = 1/2$. According to Jacobson, the design rationale of $\beta_{decr} = 1/2$ is that the sender falls back to a window that worked previously [20].

The Slow-Start heuristic is of particular importance for this dissertation. The original idea can be attributed to Jain [34], who suggested a linear window increase. Jacobson chose an exponential increase, since this function opens the window "quickly enough to have a negligible effect on performance, even on links with a large bandwidth-delay product" [20]. The Slow-Start has two important roles: On the one hand, it has to find an appropriate sending rate for a network path that is unknown, for instance, when the connection is set up. The algorithm probes the available bandwidth of the path, and it guarantees that the source sends data at a rate that is at most twice as large as the maximum possible rate on the path. On the other hand, it must also start the self-clocking mechanism. In a window-based protocol, the transmission of new packets is controlled by the stream of received ACKs. When there are no packets in the network, this process needs bootstrapping in order to limit the burstiness of the sent traffic.

If a sender has been idle for a relatively long period of time, new segments cannot be clocked out by arriving ACKs. If the Congestion Window remained unchanged, a source could potentially send a burst of the size of the *cwnd* with full line rate. In order to prevent such bursts, [RFC 2581] [1] recommends to reset the Congestion Window to the restart window if TCP has not sent data in an interval exceeding the retransmission timeout, i. e., it starts the transmission again in the Slow-Start mode.

The restart window is equal to the initial window. An experimental extension [RFC 2861] [35] describes an alternative Congestion Window Validation and suggests to decay *cwnd* roughly by factor two once per duration of the RTO, while using the SST to save information about the previous value of *cwnd*. [RFC 2861] [35] also provides recommendations for application-limited periods, i. e., when an application sends less data than the *cwnd* allows. Originally, the initial window was one MSS. Today, [RFC 6928] [36] permits an initial window of

$$min(10L; max(2L; 14600B)) \tag{2.4}$$

This new value comes from results of several large-scale experiments showing that the higher initial window improves the overall performance of many web services without resulting in a congestion collapse. It depends on the MSS L and corresponds to w_{max} $=$ 10 segments for MTU $=$ 1500 Bytes, which is the default MTU value in Ethernet and supported by most Internet paths.

This section can only give a brief overview of the TCP Reno algorithms, and it does not cover all subtle aspects. Even the specification [RFC 2581] [1] leaves open several details. For instance, it does not specify an initial value for SST, which may be arbitrarily high. Congestion control issues are also discussed in many other IETF documents.

2.4.2 Survey of New Algorithms

The standard TCP Reno algorithms do not scale well to networks that have a very high path capacity $r \gg 10$ Mbit/s and/or an RTT $\tau \gg 50$ ms. In order to fully utilize such paths, the *cwnd* must exceed the BDP $r \cdot \tau$, which may require window sizes of the order of thousands of segments, or even larger. In order to sustain high steady state throughputs, TCP Reno requires very low packet loss rates $p \ll 10^{-6}$, which is an unrealistically low value in IP networks [RFC 3649] [37]. The reason is that the additive increase algorithm in Renos congestion avoidance is rather slow when the

BDP is large. Furthermore, a flow may not be able to ramp up fast after a transient increase of the available bandwidth. In both cases, the path may not be fully utilized.

Many alternatives to the Reno TCP congestion control have been proposed. The high-speed TCP variants modify the algorithms that calculate the *cwnd*, in particular when it is large. The algorithms only require sender-side modifications and are thus incrementally deployable. In the following, the most important variants ("flavors") are briefly introduced. The discussion is limited to algorithms that are generally applicable and that have a known and validated implementation in a widely used network stack. Other comprehensive surveys can be found in literature [38]. Further domain-specific TCP enhancements have been proposed for wireless networks [39].

The majority of proposals belongs to the class of loss-based congestion control algorithms like Reno, but they use a window growth function other than AIMD. Most flavors only affect the Congestion Avoidance; the Slow-start remains unaltered. In general, window growth functions can be divided into three classes according to their shapes when being plotted over time: (a) concave, (b) convex, and (c) concave-convex. The list of algorithms in Table 2.1 contains representatives of all three cases. In literature there is disagreement concerning the optimal shape. In principle, a convex growth function is needed to ramp up the congestion window to very large values. But a convex function results in a very large window increment around the point of saturation and can cause a large burst of packet losses. As a remedy, the CUBIC congestion control [40] uses a concave-convex scheme.

Many high-speed congestion control algorithms behave like TCP Reno when being used in lowspeed and/or short-distance networks. Several proposals listed in Table 2.1 also use a window growth functions that depends on the elapsed time t since the last loss event. This common design pattern significantly reduces the RTT unfairness [42]. Furthermore, almost all proposals set β_{decr} to a value smaller than $1/2$, either depen-

TABLE 2.1: Classification and comparison of important TCP congestion control variants

Algorithm	Detect.	Probing/backoff	Parameters
Reno [RFC 2581]	Loss	AI/MD	$\alpha_{incr} = 1$, $\beta_{decr} = 1/2$
HS-TCP [RFC 3649]	Loss	Convex AI/MD	$\alpha_{incr} = f(W)$, $\beta_{decr} = f(W)$
Scalable TCP [41]	Loss	Multiplicative incr./MD	$\beta_{decr} = 1/8$
H-TCP [42]	Loss	Convex AI/MD	$\alpha_{incr} = f(t;\ \tau)$, $\beta_{decr} = f(G; d_{max};\ t)$
CUBIC [40]	Loss	Concave-convex AI/MD	$\alpha_{incr} = f(t)$, $\beta_{decr} = 0.2$
Westwood+ [43]	Loss	AI/bandwidth estimation	$\alpha_{incr} = 1$
Vegas [44]	Delay	Function of RTT	Update law: Incr./decr. by 1 MSS per RTT
FAST [45]	Delay	Function of RTT	Update law: $W \leftarrow f(d;\ \tau;\ W)$
Hybla [46]	Hybrid	AI/MD	$\alpha_{incr} = f(d)$, $\beta_{decr} = 1/2$, Slow-Start modif.
Compound [47]	Hybrid	AI+delay component/ MD	Reno emulation with $\alpha_{incr} = 1$, $\beta_{decr} = 1/2$
Illinois [48]	Hybrid	Concave AI/MD	$\alpha_{incr} = f(d;\ \tau)$, $\beta_{decr} = f(d;\ \tau)$

Legend: AI: Additive Increase MD: Multiplicative Decrease

dent on the Congestion Window size W, the throughput G, the minimum RTT t, or the maximum delay d_{max}.

Another class of high-speed TCP approaches uses delay-based congestion control. They permanently measure performance metrics such as the instantaneous RTT d, which includes potential queueing delays, and they try to anticipate congestion before buffer overflows occurs. The control algorithm increases the *cwnd* size if the delay d is not much larger than the minimum RTT τ (base RTT), and they decrease the window if the delay increases. The advantage of delay-based algorithms is that delay can be measured much more frequently than packet loss, which provides a rather coarse information if the BDP is large. Furthermore, delay-based algorithms do not completely fill bottleneck buffers. These advantages motivated the development of delay-based high-speed congestion control algorithms [45]. However, delay-based algorithms suffer from some inherent weaknesses. Delay is not a reliable congestion signal, in particular if there is delay jitter due to other effects such as Media Access Control (MAC) or reverse path congestion. Delay-based schemes also do not interoperate well with TCP Reno: Since delay-based algorithms back off much earlier, they only get a small share of the bottleneck capacity when competing with other flows using Reno. Recently, several hybrid congestion control schemes have been developed, e.g., Compound TCP [47]. These hybrid schemes combine delay-based and loss-based mechanisms (Reno emulation).

Another class of TCP congestion control algorithms, which is out of the scope of this work, addresses low-priority background transport. The purpose of such a less-than best effort congestion control is to realize the Low-Priority Data class [RFC 4594] [49] or “scavenger service” without any network support. If an application uses such a low extra-delay background transport, it should be able to utilize excess bandwidth on a path without significantly perturbing other TCP connections. Most known solutions use delay-based congestion control and back off much more aggressively than Reno when detecting packet loss [50, 51]. A further option is to use inline measurements of the available bandwidth [52].

2.4.3 Selected Loss Based Algorithm: CUBIC

From a practical point of view, the most important high-speed congestion control variant is CUBIC [40], since its is enabled by default in the network stacks of Linux and is the most widely used variant in the Internet [53]. The CUBIC congestion control has been developed by Rhee et al. [40]. It is the default congestion control in Linux since kernel version 2.6.18 and has been further developed since then. CUBIC increases the *cwnd* using a third-order polynomial function of the elapsed time from the last congestion event. This results in a concave window curve until a reference point is reached, which is the old maximum window size. If the reference point is exceeded, it continues with a convex window curve. This cubic function can be observed in the upper part of Figure 2.4, which shows the *cwnd* evolution of a single TCP connection in a emulated scenario with a single bottleneck (cf. Section 4.4). After a window reduction by $\beta_{decr} = 0.2$ due to packet loss, CUBIC stores the maximum window. In the following Congestion Avoidance phase, the cubic function is then set to have its plateau at this maximum window. The motivation for this concave-convex style of window adjustment is that the sender sends for some time approximately with the previously available bandwidth and is not very aggressive at this operational point, i. e., it achieves a high link utilization without risking burst packet losses.

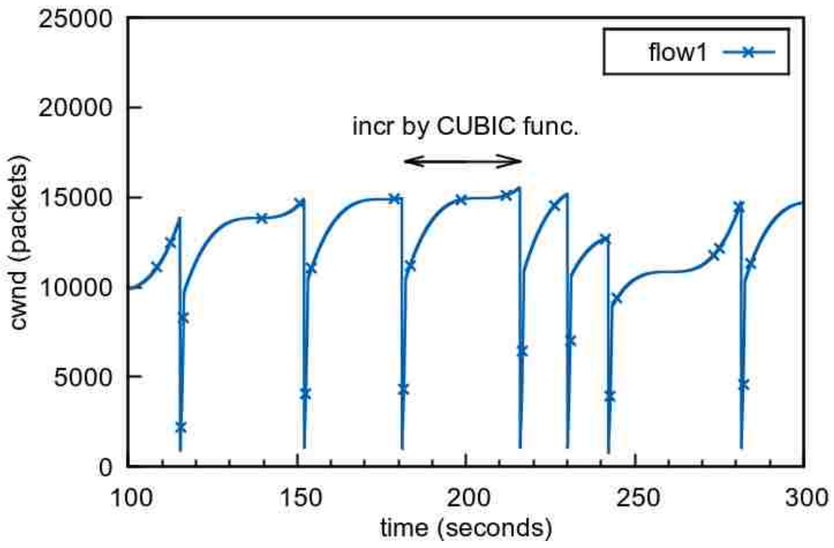


FIGURE 2.4: Trace of CUBIC (emulation, $r = 10$Gbps, $\tau = 60$ ms, buffer = 1875 pkt)

CUBIC can efficiently utilize high-speed WAN paths with RTTs of 200 ms and more. Since the window growth function is independent of the RTT, CUBIC has good RTT fairness characteristics, and it behaves similar like Reno if the BDP or RTT is small. However, there are concerns about its fairness, since it has been observed that the convergence speed can be slow [54]. This is a side effect of the small multiplicative window decrease factor $\beta_{decr} = 0.2$. Other measurement results suggest that the convergence speed of CUBIC is reasonable in environments with sufficient statistical multiplexing.

2.5 Impact of Network stack Implementations

Each operating system implements the TCP/IP stack in a different way, and the stacks are evolving over time [55]. As the specifications leave open many details, the TCP implementations of different operating systems differ significantly, and may also change from version to version. In general, the stacks in modern operating systems support more features and are better tuned.

There is a set of TCP enhancements that is supported by most endsystems in the current Internet. Measurements [55] show that most stacks use an error recovery with Selective Acknowledgments (SACKs). Also, the sizes of advertised receive windows have significantly increased: While a few years ago maximum windows of 16 KB or 64 KB were common, modern stacks support receive window scaling [56, 57]. Other proposed enhancements such as ECN get deployed only very slowly. Table 2.2 lists important TCP extensions that are supported by state-of-the-art stacks. It also illustrates some cases of different design choices. This list is not comprehensive and may change concerning newer releases of the operating systems.

TABLE 2.2: Known characteristics of popular TCP stacks

TCP mechanism	Linux kernel (versions 2.6.18 or newer)	Microsoft Windows (Windows Vista or newer)
Default congestion control	CUBIC	Reno (Windows Vista)
	Reno (in some distributions)	Compound (Windows 2008 Server)
Window scaling (RFC 1323)	Enabled, up to 4 MB	Enabled, up to 16 MB
Default scaling factor	7 (depends on memory)	8 (but only 2 for HTTP)
Automatic buffer tuning	Enabled	Enabled
Delayed ACKs (RFC 2581)	Not during Slow-Start	Enabled
Initial SST value	2,147,483,647	65,535
Limited Slow-Start (RFC 3742)	Supported, disabled	Undocumented
Appr. byte counting (RFC 3465)	Supported, disabled	Enabled
Cong. Window Valid. (RFC 2861)	Enabled	Undocumented
Conn. state caching (RFC 2140)	Enabled	Undocumented
SACK (RFCs 2018, 2883, 3517)	Enabled	Enabled
Timestamps (RFC 1323)	Enabled	Supported, disabled
RTO calculation (RFC 2988)	Not compliant (min. 200 ms)	Compliant
ECN (RFC 3168)	Supported, disabled	Supported, disabled

My work in this dissertation uses the Linux networking stack, which is a powerful and highly optimized stack. Due to the availability of the source code, the Linux stack is widely used in networking research. A comprehensive, yet partly out-dated survey of the specifics of the Linux TCP implementation has been compiled by Sarolahti et al. [56]. A general introduction can also be found in the book of Wehrle et al. [58].

The Linux kernel uses the concept of congestion control modules with a common interface [59]. Since it is simple to design new congestion control modules, more than ten different congestion control algorithms are implemented in newer Linux kernels. The system configuration determines which module is used, and an application can overwrite this choice by a socket option. CUBIC is the default algorithm unless the configuration is changed. Another feature of the Linux stack is a sophisticated SACK processing engine that may even recover if retransmitted segments get lost. As shown in Table 2.2, there are also several Linux-specific mechanisms. Such an example are “QuickAcks”: A Linux receiver acknowledges every segment if it assumes that the sender is in the Slow-Start phase. This disabling of the delayed acknowledgments, which does not violate [RFC 2581] [1], can speed up the data transport in Slow-Start. A heuristic decides when to start to delay acknowledgments. The maximum number of QuickAcks is half of the advertised receive window counted in segments with an upper bound of 16. Linux also implements Congestion Window Validation [RFC 2861] [35].

2.6 Conclusion

In this chapter, we have set the foundations for this dissertation. We have investigated TCP congestion control mechanism, design issues, challenges, and the most widely adopted TCP congestion control variants for high-speed networks. It guides our adventure to discover new problems of TCP and to find a solution for TCP in high-speed networks.

Chapter 3
Development of 10Gbps High Speed Network Emulation Testbed: CRON

CRON is a cyberinfrastructure of reconfigurable 10Gbps optical networking environment that provides multiple virtual networking topologies consisting of routers, delay links, and high-end workstations. To the best of our knowledge, CRON is the first networking testbed that emulates characteristics of high-speed networks and high performance computing in a realistic 10Gbps environment. Resources in CRON are reconfigurable according to user's requests. CRON allocates resources for multiple experiments to run concurrently and continuously in their own slice of the testbed. Both application developers and networking researchers can use those virtual high-speed networking and computing environments without technical knowledge of underlaying hardware and software. This chapter describes the design of CRON and the key features which make CRON running at a bandwidth of 10Gbps. To assist CRON users, diverse research studies at different layers of computing and networking have been demonstrated.

3.1 Background

The success of scientific applications and collaborations relies increasingly on high-speed optical networks because of geographical dispersion of supercomputing facilities and storage resources across the globe. The development and deployment of national and international optical networks (such as NLR [60], Internet2 [61], and LONI [62], etc.) make it possible for many scientists, research and educational institutes to connect and collaborate at previously unachievable levels.

Over those high-speed optical networks, research has been done to combine high performance computing and high-speed networking together, and bring the integrated environment to multi-disciplinary laboratories located at major research institutes for large scale scientific research. However, there is a critical gap between research projects and deployed production networks, caused by the following problems.

- Scarce physical resources: the expensive resources in high-speed optical networks have limitations on the number of simultaneous users who have dedicated paths. At the same time, many research and educational institutes cannot access high-speed optical networks due to the last mile problem. These shortages in networking resources constrain the development of multi-institutional or multi-disciplinary projects.

- Unmodifiable environment: because the currently deployed networks are used for production as well as research, individual users cannot modify the network environment parameters, such as network protocol stacks, operating systems, middleware, and applications.

- Limited environment: applications or protocols developed for one specific high-speed network should work correctly over all kinds of different networks, irrespective of different network features. Each deployed physical network, however, has one physical characteristic, such as bandwidth and delay. Therefore, it is difficult for developers to evaluate the performance of their application over diverse networking environment.

CRON [63] is a reconfigurable 10Gbps optical networking environment that provides multiple virtual networking topologies consisting of routers, delay links, and high-end workstations. The goal of CRON project is to bridge the gap between physical networks, simulation, and high-speed networking environment by developing a 10Gbps networking and computing cyberinfrastructure, which provides integrated and automated access to diverse networking components.

CRON resolves the critical issues by achieving four key objectives.

- Scalable virtual networking resources: CRON enables researchers to explore new network technologies and rapidly assess their impact on applications irrespective of physical limitation and allow educators to introduce a state-of-art networking environment to students who cannot access these physical networks.

- Reconfigurability: CRON provides separate virtual environments. Different users and researchers can launch their own experiments independently, and they can alter characteristics of the environments without interfering each other.

- Versatile environment: CRON provides a large variety of emulated networks, ranging from regional networks to global transoceanic 10Gbps networks, with impairments like bandwidth throttling, delay, jitter, bit corruption, and loss.

- High end resources: CRON provides high-end resources that can be used for high performance computing. For example, widely used Hadoop [64] is provided for distributed computing.

The rest of the chapter is structured as follows. Section 3.2 gives an overview of related experimental facilities. Section 3.3 describes the design of CRON. Section 3.4 illustrates the various studies using CRON by identifying specific experiment types and outlining the support for these experiments. Section 3.5 shows extending work of CRON as part of federation national wide. Section 3.6 concludes our work and mentions future work.

3.2 Related Work

CRON is based on Emulab [65] testbed, from which CRON inherits its advantages such as ease of use, control, and realism. Emulab is a testbed where users can reserve a certain number of machines and get exclusive access to bare hardware. Because of the bandwidth limitation of data plane inside Emulab and capacity limitation of the Dummynet [66] software link shaper, Emulab only provides virtual networking topologies consisting of emulated links with 10/100Mbps network bandwidth and software-based routers less than 1Gbps processing speed. Since most nodes and switches only support 100Mbps bandwidth, Emulab is not adequate for networking researchers to develop future high-speed network architectures or evaluate performance of high-speed network protocols. On the other hand, CRON has extended Emulab to bandwidth of 10Gbps with various hardware and software for high-speed optical networks.

PlanetLab [67] mainly focuses on system and service research over Internet. Although it provides flexible virtual machines, networks connecting virtual machines are not reconfigurable nor reproducible for networking experiments. Furthermore, most federated PlanetLab clusters are connected over regular Internet where bandwidth is varying and limited.

PRObE [68] has the idea of reusing retiring high performance clusters. Reused computing resources are shared by system researchers at high-end computing system community, storage systems community, and data-intensive computing community at large scale. Because of shortage in high-speed routers and emulators as networking components, however, PRObE targets only system and service research.

FutureGrid [69] and Open Cirrus [70] support either system or service research based on federation among different clusters and data centers connected through Internet or dedicated high-speed production networks. Although they can provide flexible virtual computing environments, their networks are limited or restricted from the perspective of bandwidth and network topology.

RENCI's optical BEN [71] is a Triangle Universities (UNC-CH, Duke and NCSU) owned testbed. Equipments are interconnected through fiber at metro scale. The testbed provides automatic switching at optical speed. However, the dedicated testbed resource is time shared and used only by those three organizations. Not all the links are able to be changed. What is more, user management is constrained inside universities and resource governance is restricted by security concerns.

3.3 Infrastructure

Compared with related architectures, CRON has the following unique features from computing and networking perspectives.

- **Networking Layer** provides reconfigurable 10Gbps high-speed networking environment consisting of 10Gbps Ethernet switch, 10Gbps emulated links impaired by modified Dummynet [66] software emulator or Anue hardware emulator [72], 10Gbps Linux software routers.

- **Cloud Management Layer** supports Eucalyptus [73] as cloud infrastructure management software. After allocating and isolating physical resources with multiple virtual LAN (VLAN) blocks, distributed virtual clouds are automatically emulated.

- **Distributed Data-intensive Computing and Application Layer** gives high level services, such as Hadoop over distributed virtual clouds. A set of benchmark applications are provided for the purpose of performance evaluation.

Figure 3.1 shows the general architecture of CRON, which consists of two main components: (i) hardware component, including 10Gbps core switch, optical fibers, network emulators, and workstations that are required to compose 10Gbps paths or function at the ends of these paths; and (ii) software component, including an automatic configuration server as to integrate all the hardware through a second 1Gbps control plane to create virtual network environments.

To allow data movement from outside networks such as NLR, LONI and Internet2 [61], the 10Gbps core switch has two external 10 Gbps connections. These two

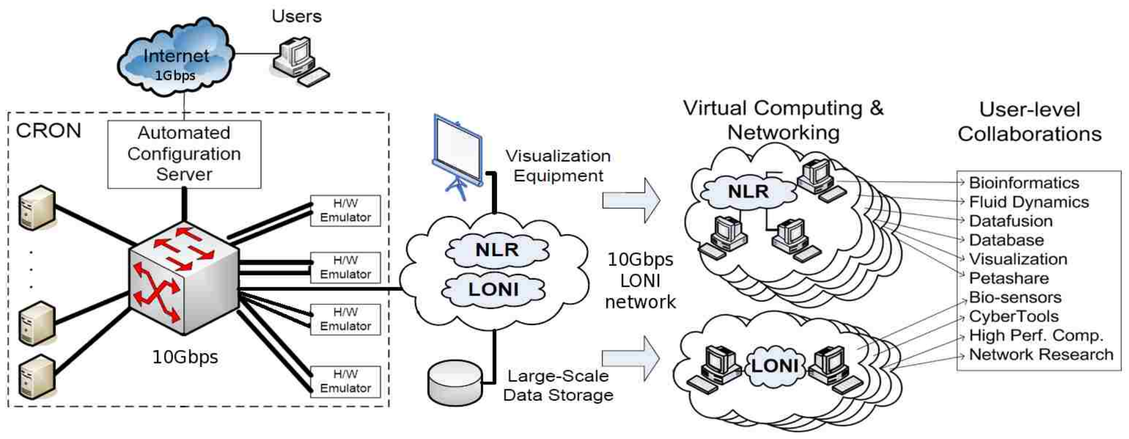


FIGURE 3.1: CRON system architecture

external connections extend and integrate CRON with other cooperative projects, as shown in Figure 3.1 on the right.

3.3.1 Hardware Component

This subsection describes these integrated contemporary commodity hardware such as high-end workstations and hardware network emulators. Figure 3.2 shows the physical hardware component and their connections. All hardware components have 10Gbps capacity, such as 10Gbps hardware emulators, 10Gbps data center switch, 10Gbps network interface cards, etc.

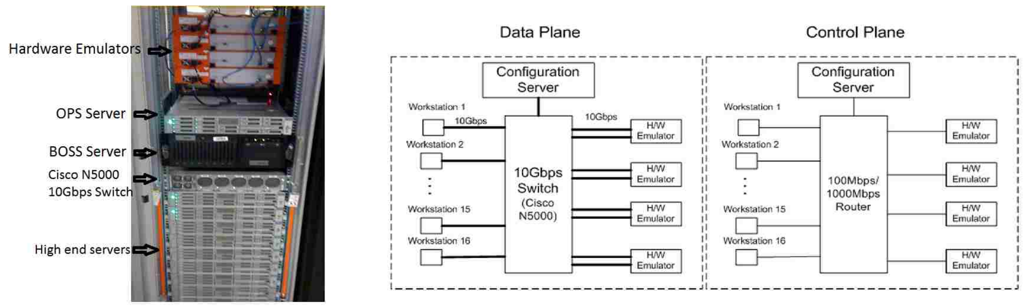


FIGURE 3.2: CRON system hardware component

With the introduction of 10-GigE, network I/O re-entered the “fast network, slow host” scenario that occurred with both the transitions to Fast Ethernet and Gigabit Ethernet. Specifically, three major system bottlenecks limit the efficiency of high performance network adapters [74]: the PCI bus efficiency, the CPU efficiency and the memory efficiency. In the last years, a rapid development of technology addressed these challenges. The PCI-X bus, which has a peak bandwidth of 8.5Gbps operating at 133 MHz, has been replaced by the PCI-Express (PCIe) bus, which has a peak bandwidth of 20Gbps using 8 lanes. CPUs have entered the multi-core era and the memory data rate has increased from 3.2GBps (single channel DDR-400 memory) to 10.7GBps (dual channel DDR2-667 memory). Superior memory management and bus contention handling mechanism from AMD platform substantially reduce the round-trip data processing latency. By comparing symmetric-multiprocessing (SMP) architectures between Intel and AMD, we chose AMD SMP platform as the hardware for our low-cost 10Gbps network nodes. Table 3.1 shows hardware features of CRON high-end workstations.

TABLE 3.1: Hardware features of high-end workstations in CRON

Technology	Description
AMD64	support 32/64-bit application multitasking
DirectConnect	reduce latency between Memory, I/O and cores
HyperTransport	high throughput between cores
IntegratedMemoryController	on-chip controller for low-latency memory

The 10Gbps Anue Ethernet multi-profile hardware network emulator [72] is a precision instrument for emulation testing. It efficiently generates network impairments like link delay, frame drop, jitter, and bit errors as well as it controls link bandwidth ranging from 1Kbps to 10Gbps. It supports 4 network profiles. Each profile is a set of parameters that can be applied to 1 or more streams of network traffic. The hardware emulator can be configured to support either uni-directional or bi-directional transmission. Impairments can be introduced to OSI layers 1 through 7 and it also supports proprietary protocols.

Creating virtual topology to use hardware emulator is similar to creating an experiment, except that a user specifies a delay value starting with an special ‘H’ character.

The CRON resource allocator interprets the character as a parameter to assign the hardware emulator. In Tcl programming format, the piece of script to request a hardware emulator is given below.

```
set link [$ns duplex-link $n1 $n2 10000Mb H50ms DropTail]
```

Upon receiving a user's request, CRON assigns available hardware emulators to the user's experiment. The user can configure link impairments either by the web based management portal or by a Tcl script. CRON maintains a hardware emulator database to track the usage of each hardware emulator.

3.3.2 Software Component

This subsection describes those software enhancements to achieve 10Gbps performance with contemporary commodity hardware.

Our nodes use both Linux and FreeBSD open source operating systems. We applied a handful of techniques to improve the performance of operating systems [75], [3]. Those tunings include driver optimization for 10Gbps network adapter, TCP/IP network buffer tuning, kernel clock tuning in FreeBSD and kernel efficiency tuning in Linux.

At data link layer, IEEE 802.3x Ethernet-flow-control temporarily stops the transmission of data on Ethernet ports when flows are over link capacity. However, it causes a blocking effect known as "head of line blocking" [76] which shortly pauses transmission at each network adapter. Therefore, flow control is disabled in the 10Gbps core switch. On the other hand, no packet drops are detected in the 10Gbps core switch when network links are emulated. This indicates the 10Gbps core switch is transparent to those emulated network links and guarantees each emulated network link with physical 10Gbps speed.

At layers of TCP/IP, large-receive-offload (LRO) [77] is widely used to improve packet processing efficiency. However, because LRO combines multiple packets into a big block to reduce computing overhead, it causes additional latency for IP forwarding. Instead, LRO is enabled in end nodes, and is disabled in nodes as software routers. We also set the default MTU size to 9000 Bytes as jumbo frame. For software routers, we implemented and patched various Linux queuing discipline controllers for traffic control. In addition, we set up network adapter to use standard *skbuf* to hold packets for IP forwarding. Inside Linux kernel, we enlarged the TCP buffer size to reach 10Gbps bandwidth under various high-speed TCP congestion control algorithms.

At application layer, we also implemented the zerocopy Iperf network traffic generator to avoid the overhead of data copy from user-space to kernel space.

The core switch inside CRON is a Cisco N5020 data center switch. Cisco N5020 runs Cisco Nexus operating system, and supports line-rate, low-latency, lossless 10 Gigabit Ethernet and Fibre Channel over Ethernet (FCoE). Originally, the control software from Emulab uses simple-network-management-protocol (SNMP) API to do automatic VLAN operations, for example add-and-delete VLAN, add-and-delete ports from VLAN. SNMP depends on management-information-base (MIB) API to manage the switch configurations. Emulab uses *cisco_vtp* MIB. However, *cisco_vtp* MIB are not supported by Cisco N5020. Also, the Cisco N5020 switch only supports *read* function in SNMP rather than *read_write* which means SNMP is only used for monitoring. Therefore, we need to develope new control software for VLAN management of Cisco N5020 switch.

The Nexus operating system offers several management interfaces. One of the interfaces is command-line-interface (CLI), which supports both show and configuration. Because the core switch requires reconfiguration automatically inside CRON system, CLI is not an idea choice. The second interface is SNMP, but it is only used for monitoring. The third one is extensible-markup-language (XML) interface. The Cisco N5020 switch supports XML by using NETCONF [78] protocol, which is a management protocol developed by the IETF working committee. NETCONF is loosely structured in which XML wraps CLI commands.

We use a PERL programming module called *Expect* for the XML wrapping. We rewrite functions like *createVlan*, *removeVlan*, *setPortVlan*, *removePortsFromVlan*, *listVlans*, *listPorts*. And create a *do_expect* function as an XML interface for *Expect* module.

Dummynet [66] in FreeBSD is a user-configurable software network emulator. In Dummynet, users can apply network shaping such as limit link bandwidth, add delay, increase packet loss rate, and change queue size. Because the hardware network emulator is very expensive, we use Dummynet as an alternative to fulfill the network dynamics in CRON.

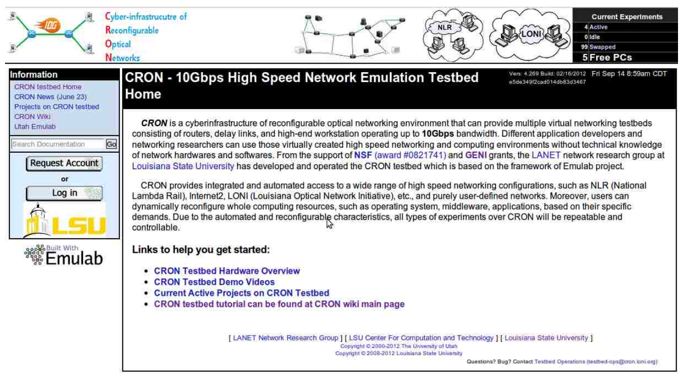
In order to control link bandwidth, users need to specify the bandwidth value every time they use Dummynet. However, all the variables of bandwidth in Dummynet source code are defined as 32bit *int*, which ranges from -2,147,483,648 to 2,147,483,647. Therefore, the 32bit integer width is not enough for a 10G value. This explains why the original Dummynet can only provide around 2Gbps of bandwidth. Therefore, we changed the type of all the bandwidth variables to *int64_t*, both in user space and kernel space, so that the bandwidth variables support 10Gbps of bandwidth. Tuning Dummynet performance over a 10Gbps link is another challenge. We have customized Dummynet source code to support 10Gbps link bandwidth; however, its performance jitters frequently. The reason is a memory zone fragmentation problem [79]: the *Mbuf* memory for holding packets always runs in short in FreeBSD at 10Gbps network speed. To solve this issue, we have to optimize memory allocation by creating a continuous memory zone to avoid memory fragmentation. This changes both OS kernel and network adapter driver. Also, some operating system parameters are tuned for Dummynet, such as *net.inet.ip.dummynet.pipe_byte_limit* is changed to

1,000,000,000 for an enlarged queue size, and *hw.intr_storm_threshold* is increased to 10,000 for a large interrupt storm threshold. Then, the tuned Dummynet is able to shape network traffic at line speed.

3.4 Experiment and Evaluation

In this section, we evaluate CRON by conducting various experimental studies over 10Gbps high-speed networks.

After launching the web portal service, CRON is accessible from the Internet. As shown in Figure 3.3(a), every user can request an account at the CRON website. A list of performed experiments are shown in Figure 3.3(b).



Cyber-infrastrucutre of
Reconfigurable
Optical
Networks

Current Experiments
4 Active
0 Idle
99 Swapped
5 Free PCs

Information
CRON testbed Home
CRON News (June 23)
Projects on CRON testbed
CRON Wiki
Utah Emulab

Search Documentation Go

Request Account
or
Log in

Built With Emulab

CRON - 10Gbps High Speed Network Emulation Testbed Home

Vers: 4.269 Build: 02/16/2012 Fri Sep 14 8:59am CDT
e5de349f2cad014db83d3467

CRON is a cyberinfrastructure of reconfigurable optical networking environment that can provide multiple virtual networking testbeds consisting of routers, delay links, and high-end workstation operating up to **10Gbps** bandwidth. Different application developers and networking researchers can use those virtually created high speed networking and computing environments without technical knowledge of network hardwares and softwares. From the support of **NSF** (award #0821741) and **GENI** grants, the LANET network research group at Louisiana State University has developed and operated the CRON testbed which is based on the framework of Emulab project.

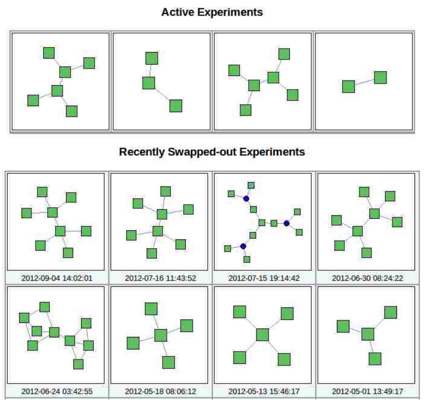
CRON provides integrated and automated access to a wide range of high speed networking configurations, such as NLR (National Lambda Rail), Internet2, LONI (Louisiana Optical Network Initiative), etc., and purely user-defined networks. Moreover, users can dynamically reconfigure whole computing resources, such as operating system, middleware, applications, based on their specific demands. Due to the automated and reconfigurable characteristics, all types of experiments over CRON will be repeatable and controllable.

Links to help you get started:

- **CRON Testbed Hardware Overview**
- **CRON Testbed Demo Videos**
- **Current Active Projects on CRON Testbed**
- **CRON testbed tutorial can be found at CRON wiki main page**

[LANET Network Research Group] [LSU Center For Computation and Technology] [Louisiana State University]
Copyright © 2000-2012 The University of Utah
Copyright © 2008-2012 Louisiana State University
Questions? Bug? Contact Testbed Operations (testbed-ops@cron.loni.org)

(a) CRON testbed website



(b) Some experiments in CRON

FIGURE 3.3: CRON web portal

3.4.1 Research at Network Layer 3 and Layer 4: Evaluation of high-speed TCP Variants and Active Queue Management

In CRON, studies [75, 3] have been done on the evaluation of high-speed TCP variants and active queue management schemes over 10Gbps high-speed networks. We create a Dumbbell topology as shown in Figure 3.4. Three senders run a modified version of Linux 2.6.34 kernel. Those senders are used to initiate TCP flows. The two routers run a modified version of Linux-2.6.39 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.

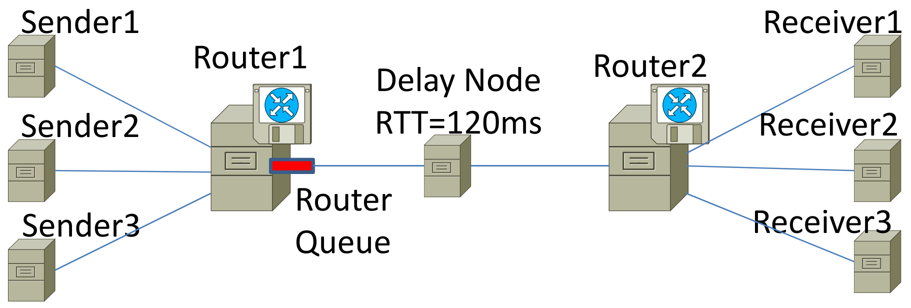


FIGURE 3.4: Topology of networking experiment

In our experiments, three senders send three different TCP congestion control variants, namely, TCP-SACK, CUBIC, and HSTCP. At output queue of Router1, we evaluate three kinds of Linux queuing disciplines, namely, Drop-tail, RED, CHOKe and AFD. For parameters of RED and CHOKe, we set minimum threshold to 20% of buffer size, maximum threshold to 90% of buffer size, and drop probability to 0.02. We vary the queue size at Router1's output queue from 1% to 100% bandwidth-delay-product (BDP). On delay node, default RTT is set as 120ms to emulate the long delay in high-speed networks. Long-lived TCP flows are generated by zero-copy Iperf. 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. The interconnection times are generated from exponential distribution with a mean of 1 second. File sizes are generated from Pareto distribution with alpha=1.2 and shape=1500. The chosen distributions and parameters are based on Internet traffic characteristics. Fairness is calculated among heterogeneous TCP flows in terms of the long term throughput received by each flow as Jains fairness index. All the presented results are averaged over five experiments and duration of each run is 20 minutes.

We have 3 different kinds of heterogeneous TCP flows in the bottleneck link. Figure 3.5(a) shows the bottleneck link utilization for a single long-lived TCP flow sce-

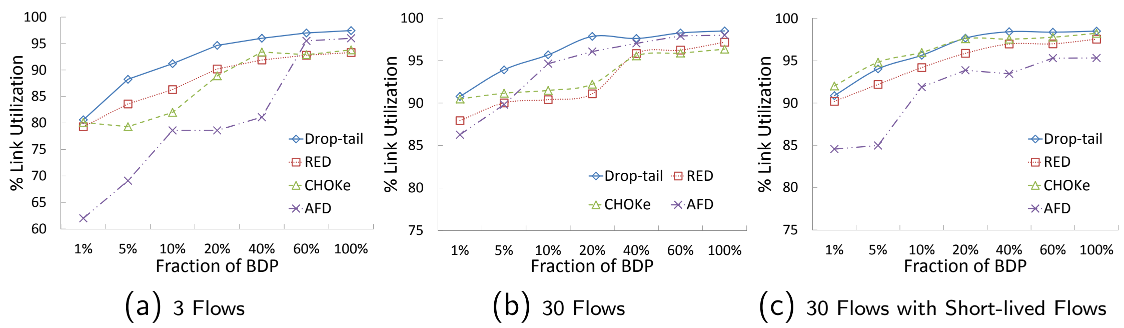


FIGURE 3.5: Link utilization for heterogeneous TCP flows (120ms RTT)

nario. 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 all queue management schemes. Figure 3.5(b) shows link utilization for many long-lived TCP flows case. In 1% BDP, the link utilization is almost up to 85%. And Drop-tail still gets more throughput than other AQM schemes. Figure 3.5(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.

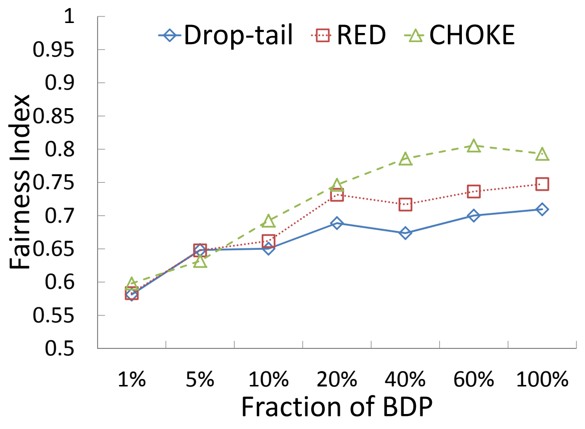


FIGURE 3.6: Fairness for 1 TCP-SACK, 1 CUBIC, and 1 HSTCP flow (120ms RTT)

Next, 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 3.6 shows fairness index as a function of router buffer size for three different types of routers. The network behaves unfairly as compared to previous studies focused on homogeneous TCP flows. To show the degree of decrease of fairness while transitioning from homogeneous network to heterogeneous one, fairness index is presented in Table 3.2 for these two cases.

TABLE 3.2: TCP Fairness: Homogeneous versus Heterogeneous (20% BDP buffer, 120ms RTT)

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

When we use RED and CHOKe on the Router1 with 1% and 5% BDP queue sizes, all three routers have almost the same fairness as shown in Figure 3.6. When the queue size increases to 10-20% BDP, RED and CHOKe start to show improvement in fairness. RED provides 0.05 more fairness index than Drop-tail. We also observe fairness index improvement by 0.03-0.08 in the case of CHOKe as compared to RED.

3.5 Federation CRON Testbed with Other GENI Sites

The Global Environment for Network Innovations (GENI) [80] is a project providing collaborative and exploratory environments for academia, industry and the public to catalyze groundbreaking discoveries and innovation in emerging global networks. There are branches of prototype implementation and deployment of GENI. For example, ProtoGENI [81] and PlanetLab [67]. This section demonstrates the federation of CRON with ProtoGENI sites and Planetlab sites through GENI framework.

Federation between CRON and BBN GPOLab within ProtoGENI has been demonstrated at 9th GENI engineering conference (GEC 9). Federation between CRON and PlanetLab MAX has been demonstrated at GEC12.

3.5.1 Federating CRON with ProtoGENI at BBN GPO Lab

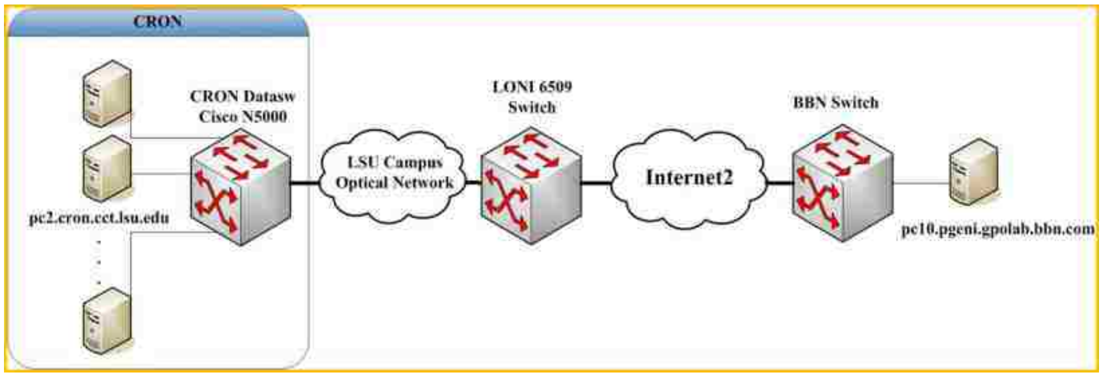


FIGURE 3.7: Federation CRON with ProtoGENI at BBN GPO Lab

Federating CRON with ProtoGENI is to connect one server at CRON testbed and one server from GPOLab at BBN as shown in Figure 3.7. Using the ProtoGENI package from Emulab, we are able to reserve external resources from other ProtoGENI sites. The data interfaces at both sides are connected into Internet2 through ION layer 2 service. A GENI slice holds a collection of computing and communication resources capable of running an experiment or a wide area service. RSpec [82] is used for advertising, requesting, and describing the resources. At first, CRON component manager receives user credentials. Then the user registers a slice at the clearinghouse at Emulab, which will do resource reservation between component managers of CRON and GPOLab. After that, experiment can be created, and RSpecs are exchanged between two sides. Internet2 ION service creates a VLAN, and sets up the connection between CRON and GPOLab.

After federating CRON into ProtoGENI, CRON resources can be shown in the ProtoGENI portal as in Figure 3.8.

3.5.2 Federating CRON with PlanetLab at MAX

At first, the clearinghouse at PlanetLab authenticates users and issues their credentials to obtain GENI resources for experimentation. Then both CRON and MAX provide resources to users with GENI credentials. The GENI aggregate-manager API

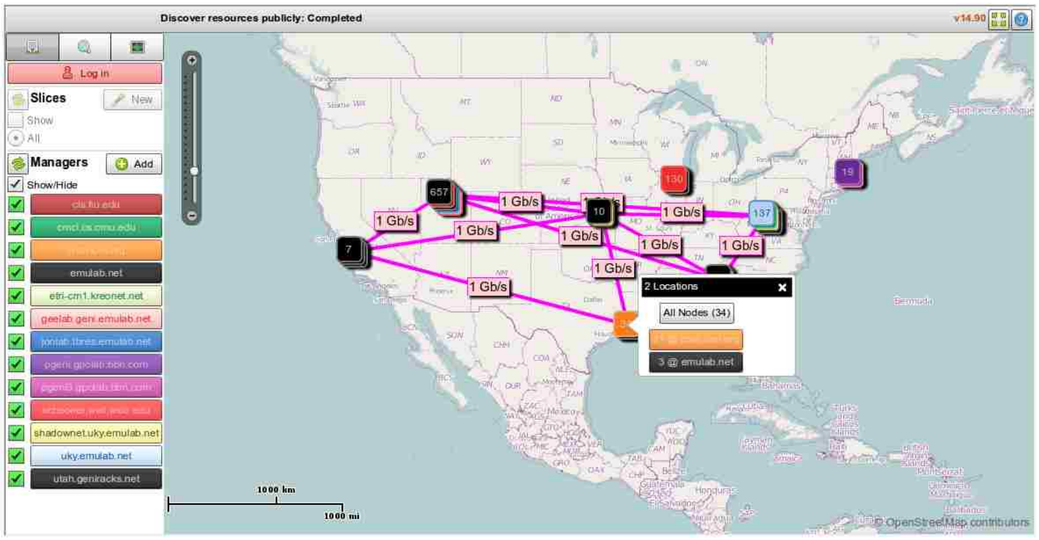


FIGURE 3.8: CRON resources in ProtoGENI portal

provides a common interface to other aggregate-managers, including PlanetLab, ProtoGENI, and OpenFlow. We use the GENI aggregate-manager API, which includes Flack and Omni tools.

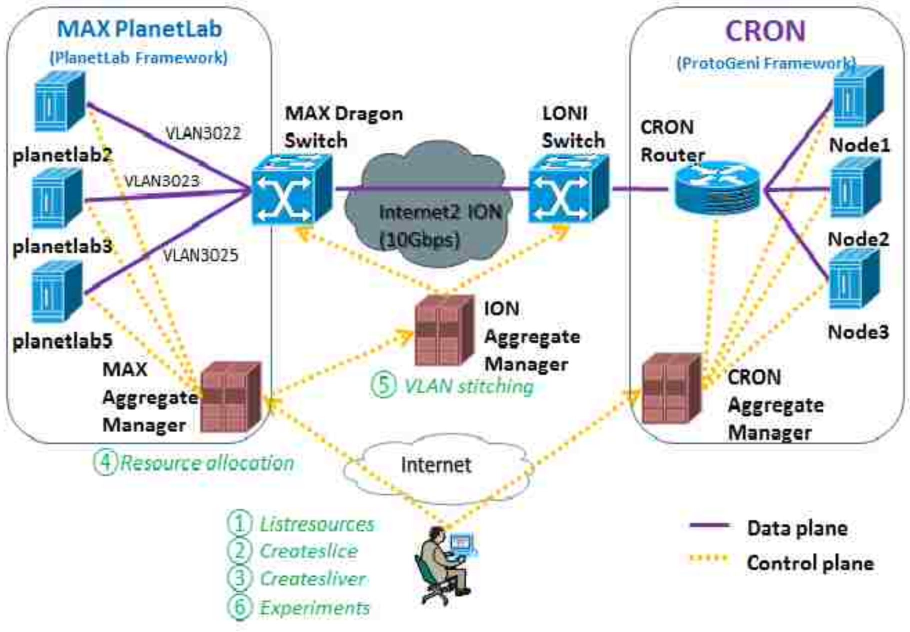


FIGURE 3.9: Federation CRON with PlanetLab at MAX

The Internet2 ION aggregate-manager does VLAN stitching to connect CRON and MAX as a coherent network. Our experiment topology and procedure are shown in Figure 3.9. GENI network stitching operation is to construct a topology of substrates as represented by their aggregate-managers. Each aggregate-manager has a unique Rspec which defines its substrate resources.

3.6 Conclusion

CRON provides integrated resources for emulating a wide range of high-speed networking and high performance computing experiments. CRON gives users access and control over high-end nodes and up to 10Gbps high speed links over their own slice of the testbed. Users can focus on their research without technical knowledge of background environment. For example, users can leverage experiments based on their specific demands among layers from networking to application. Because of reconfigurable and reproducible features, many types of experiments, including cloud computing and distributed computing, can be emulated.

Chapter 4
TCP Loss Synchronization in High Speed Network

TCP loss synchronization (TCP-LS) is a phenomenon in which many long-lived TCP flows drop their sending rates simultaneously in response to a common congested bottleneck. This is a byproduct of flows competing aggressively for more bandwidth. The TCP congestion control (TCP-CC) algorithm inside each flow determines the degree of competition. In high-speed networks, newly designed TCP-CC algorithms are more aggressive in achieving available bandwidth than a traditional additive-increase multiplicative-decrease (AIMD) mechanism. As a result, they are replacing AIMD worldwide.

In this chapter we study the effectiveness of three TCP-CC algorithms in link utilization. We carefully emulate TCP-LS cases using Linux systems in a 10Gbps network testbed. We propose new expressions to evaluate degrees of TCP-LS in order to evaluate the effectiveness of different TCP-CC algorithms for link utilization. From popular variants of TCP-CC in a small buffer environment, we find high-speed TCP (HSTCP) to be most effective for link utilization, though it is only the second most widely used TCP-CC in the Internet.

4.1 Background

In TCP/IP networks with bandwidth of 10Gbps or higher, one big issue is how to use link bandwidth efficiently. For a bottleneck shared by concurrent long-lived flows, congestion can happen and packets can be simultaneously dropped. The congestion is caused by overflow at the bottleneck buffer. As a response, the TCP-CC algorithm reduces sending rate in each flow. When many sending rates are reduced within a short time, it is recognized as a TCP-LS event. As shown in Figure 4.1 on the left, at a TCP-LS event, all flows are losing their shares of bandwidth. Therefore, if all congestions are TCP-LS events, the bottleneck will cause great underutilization. In our analysis, the worst case of unused bandwidth is a loss of 2.5Gbps for a 10Gbps network.

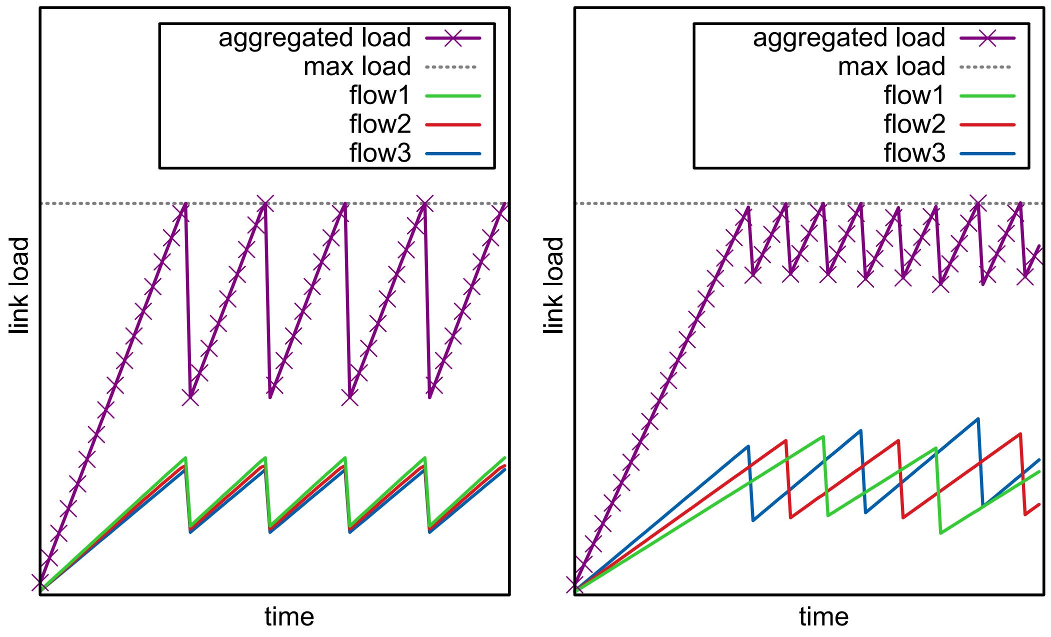


FIGURE 4.1: Synthetic demonstration of complete TCP-LS (left) and complete loss desynchronization (right).

TCP-LS has widely been studied as existing studies propose different expressions [83, 84, 85] to evaluate degrees of TCP-LS. One way to estimate TCP-LS rate of a flow is to divide the number of congestions in a flow by the total number of congestions in the bottleneck. However, this has limits because a congestion may not be a TCP-LS event. When we explore long-lived flow patterns controlled by AIMD and high speed TCP variants, we find common cases where not all flows are reducing sending rates or an extreme case where only one flow is reducing sending rate. We define the common cases as partial TCP-LS and the extreme case as complete loss desynchronization. As exemplified in Figure 4.1 on the right, link utilization is higher in the case of complete loss desynchronization. To the best of our knowledge, these cases have not been considered or evaluated.

A recent study [2] summarizes that enhancing link utilization is a critical challenge between buffer sizing problems and TCP-CC in high-speed networks. Because large buffer size increases latency, complexity and cost, reducing buffer size is the final goal. A small-buffer model [4] and tiny-buffer models [5, 6, 7] are proposed for near 100% and 80% to 90% link utilization. However, these works rely on the asynchronous behavior of tens of thousands of TCP flows. Further, accurate buffer size control is not easily available in commercial routers. Therefore, little experimental study has been done with a few dozens of synchronous flows.

By controlling buffer size precisely in a Linux router, this chapter presents an experimental study of TCP-LS among long-lived flows over 10Gbps high-speed optical networks. Popular TCP variants are studied such as RENO [86] TCP, high-speed TCP (HSTCP) [87], and CUBIC [40] TCP. Major contributions are: (i) we demonstrate various TCP-LS cases among long-lived TCP flows with a method to capture the congestion event precisely at 10Gbps; and (ii) we propose two new TCP-LS expressions, which verify TCP-LS cases and have been tested and proven in our testbed; and (iii) we explore degrees of loss synchronization on popular TCP variants. We find that HSTCP is most effective for link utilization in a small buffer environment, though it is only the second most widely used TCP-CC in the Internet [88].

The remainder of this chapter is organized as follows. Section 4.2 gives a congestion event capturing method implemented in the Linux kernel. Section 4.3 shows two new expressions to evaluate TCP-LS rate. Section 4.4 and section 4.5 show our testbed design and experimental results in homogeneous network traffic (intra-TCPs) and in heterogeneous network traffic (inter-TCPs).

4.2 Emulation Method

We avoid TCP-LS measurement from the Internet [84], because of non-deterministic background traffic, for example short-lived web flows and bursty UDP streams. Those cause non-reproducibility and obscure behaviors among long-lived flows. We also avoid TCP-LS measurement from software simulation [85], because event driven simulation suffers from timing ambiguity and inadequate computing scalability. We use a "bare metal" testbed to emulate state of the art.

In Linux network stack, TCP-CC cuts off congestion window (*cwnd*) immediately upon a detected loss, and it updates flow status (*tcp_ca_state*) to congestion-recovery. Consequently at granularity of packet level, a status switch indicates a congestion event. The following algorithm 4.2.1 is our packet-sniffing method for multiple flows, where *loss_flag* indicates a loss event. For each TCP flow, we record flow ID, time stamp, loss flag, congestion window, slow start threshold, smoothed round trip time (RTT), congestion avoidance state, and total retransmission.

Algorithm 4.2.1 optimized TCP multi-flow probe

```
while each TCP packet of a specific flow do
  if ($current\_cwnd \neq previous\_cwnd$) then
    record $flow\_id, timestamp, cwnd, ssthresh, srtt,$
          $total\_retrans, icsk\_ca\_state$
    if ($current\_tcp\_ca\_state == TCP\_CA\_Recovery$) && ($previous\_tcp\_ca\_state ==$
    $TCP\_CA\_Open \mid\mid TCP\_CA\_Disorder$) then
      $loss\_flag \leftarrow 1$
    else
      $loss\_flag \leftarrow 0$
    end if
  end if
  $previous\_cwnd \leftarrow current\_cwnd$
  $previous\_tcp\_ca\_state \leftarrow current\_tcp\_ca\_state$
end while
```

Monitoring blocks of digital messages at 10Gbps usually requires a faster execution speed than those of modern hardware platforms [89]. Popular packet-sniffing tools such as TCPdump and Ethereal [90] run in user space and are not able to overcome listen-and-write overhead. Instead, we implement our packet-sniffing algorithm in *tcpprobe*, a module that runs in kernel space. In this module, each *loss_flag* is updated upon a status switch from "TCP_CA_Open" or "TCP_CA_Disorder" to

"TCP_CA_Recovery". Then, the module captures TCP headers and copies them from kernel space into user space. Meanwhile, the module is optimized to record data only when *cwnd* changes. As a result, the size of storage is reduced significantly. The performance of our implementation is guaranteed at 10Gbps, no matter how many flows are generated at the same sender.

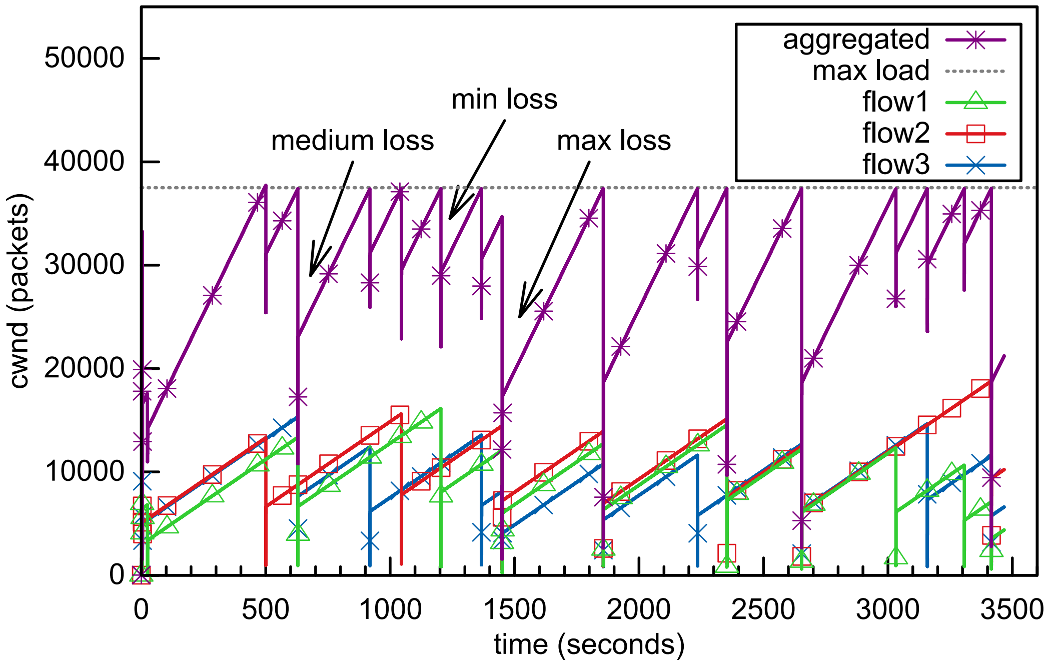


FIGURE 4.2: TCP-LS cases of 3 competing RENO flows: complete synchronization causes maximum bandwidth loss, partial synchronization causes medium loss, and complete desynchronization causes minimum loss.

Figure 4.2 shows congestion window growth during 3500 seconds for 3 concurrent long-lived RENO flows from three 10Gbps senders. Those flows compete a 10Gbps bottleneck. Packet size of each flow is 2000 bytes, resulting a maximum load of 37500 packets in this 60 milliseconds delay pipe. Reasons for choosing the packet size and detail of testbed design are described in section 4.4. Here, we see complete TCP-LS near 1500s, 1900s, and 2600s where the aggregated congestion window drops significantly. We also see partial TCP-LS of 2 flows near 600s, and 2400s. Other losses involve complete desynchronization, which causes minimum loss to the aggregated throughput.

4.3 New Loss Synchronization Expressions

A congestion event happens when there are one or more packets dropped simultaneously at a bottleneck within a "short" interval Δt. These dropped packets are either from a single flow or from multiple flows. In time domain, Δt is roughly one round-trip-time (rtt). There are two separated congestion events if their Δts are not overlapped. For a total of n flows through a bottleneck over a "long" measuring period τ, we let T denote total number of congestion events, where $l_{i,k}$ represents a loss event at the k-th congestion for flow i such that $l_{i,k} = 1$ when flow i loses, and $l_{i,k} = 0$ otherwise. Therefore, the total number of loss events N_i for flow i is: $\sum_{k=1}^{T} l_{i,k} = N_i \in [1, \dots, T]$.

In real world, not all flows are loss synchronized in a certain congestion event, nor do all congestion events belong to the same case of synchronization. As shown in Figure 4.3, there are 9 congestion events and 3 TCP-LS cases, $N_i = 4$, $N_j = 6$, $N_k = 4$, and $T = 9$. At the 1st, 6th and 8th congestion event, 2 out of 3 flows are loss synchronized, causing medium loss. At the 3rd congestion event, 3 flows are completely loss synchronized, causing maximum loss. The other congestion events are complete loss desynchronization. Therefore, a corresponding weight to the degree of a congestion event helps to evaluate TCP-LS. Particularly, if there are 3 long-lived flows, the weight is one of 1/3, 2/3, and 1, which represent the complete loss desynchronization (1/3), the partial TCP-LS (2/3), and the complete TCP-LS (1), respectively. Since TCP-LS degrades link utilization at different level, we propose the following Expression (4.1) to indicate per-flow's contribution.

$$SR_i = \frac{N_i^w}{T} = \frac{1}{T}\sum_{k=1}^{T}(l_{i,k} \times weight_k) \tag{4.1}$$

In Expression (4.1), a ratio between weighted number of loss events N_i^w and total number of congestion events T indicates per-flow TCP-LS rate for flow i. As degree of loss events differs, the weight verifies each event that involves a certain set of flows.

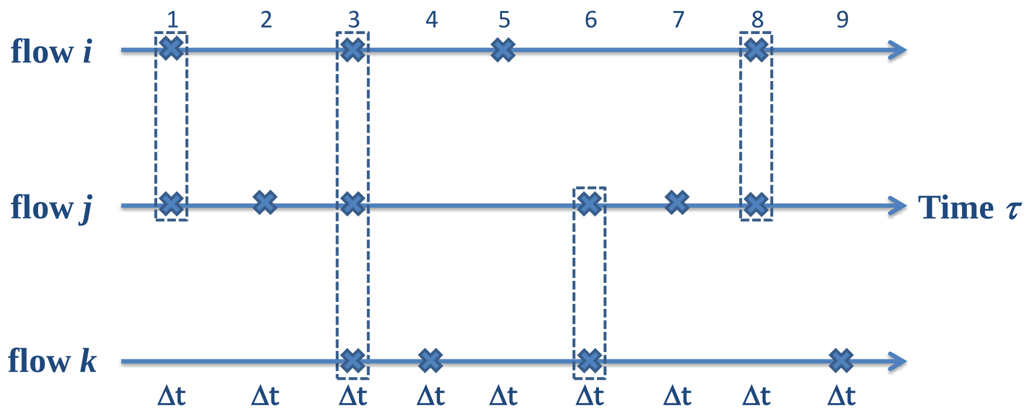


FIGURE 4.3: Real world TCP-LS cases over 3 flows (each x is a loss event)

One might expect the overall impact of TCP-LS for the bottleneck of n flows. A global TCP-LS rate is expressed as a harmonic mean of all per-flow TCP-LS rates in Expression (4.2).

$$\overline{SR} = \frac{n}{\sum_{i=1}^{n} \frac{1}{SR_i}} \tag{4.2}$$

4.4 Testbed Design

We create a dumbbell testbed as shown in Figure 4.4 from CRON [63], which is a cyber-infrastructure of reconfigurable optical networking environment that provides multiple emulation testbeds operating up to 10Gbps bandwidth. Most nodes are symmetric-multiprocessing (SMP) servers containing 10Gbps network interfaces. The delay node in the middle is an Anue XGEM 10Gbps hardware emulator. It provides bi-directional communication delay fixed at 60ms.

We use Ubuntu Server Linux with a re-compiled 2.6.39 kernel. The kernel runs a 1 kHz timer to record time stamp and *srtt* at 1 ms resolution. All nodes are kernel clock synchronized with a local time server (NTP server) at the same resolution. We also adopt CPU affinity over 10Gbps interfaces [**?**] to improve processor cache hit rate. In order to keep router processor load below 85%, we set packet size to 2000 bytes

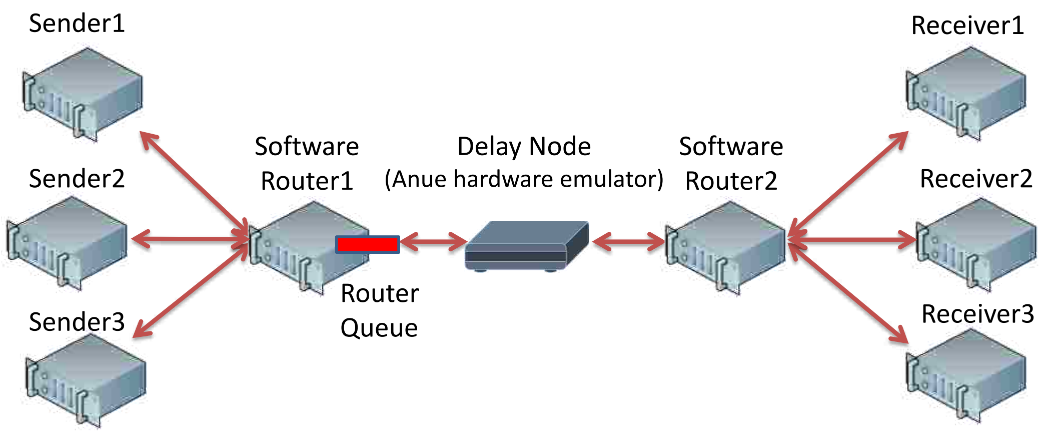


FIGURE 4.4: Experiment topology

instead of standard 1500 bytes. Most system parameters are tuned for performance[1]. In summary, all links are tested initially to confirm operation at line speed.

For buffer sizing at Router1's output port, since arguments for small buffers have been put forth, we use a fixed size of 5% bandwidth-delay-product (BDP) buffer, which is a 3.6 megabytes drop-tail sized "appropriately" between small-buffer model [4] (10.3 megabytes) and tiny-buffer model [6] (51.1 kilobytes). This 5% BDP buffer comprises 1875 packets, and it has two parts inside Linux kernel: 1024 descriptors of *tx_ring* mapped by network interface driver and 851 descriptors of *qdisc* inside IP stack.

We have done 138 hours of tests. These tests are cases of 2, 3, 6, 12, 24 and 48 flows, where each sender sends 1, 2, 4, 8, and 16 flows, except only two senders are used in the first case. In each test, flows are sent by the traffic generator Iperf from $Sender_i$ to $Receiver_i$, respectively. One trial for each test lasts 3500 seconds, and repeats 6 times.

After warm-up of 200 seconds, we calculate TCP-LS rates for 3000 seconds. The TCP-LS rates are the arithmetic means of these 6 repeats with an observed standard error, which are relatively small in the range of ±5%. The 5% BDP buffer reduces

[1]Description about system tuning can be found at CRON wiki at `https://wiki.cct.lsu.edu/cronwiki/`

queuing latency and jitter efficiently as *srtt* read from TCP stack shows delay between 60ms and 63ms.

4.5 Experiment Results

4.5.1 Homogeneous TCP Experiment

Table 4.1 shows average distribution of synchronization phenomena on 2, 3 and 6 flows tests. On the 2 flows test, 40% of all loss events involve only 1 flow, a large part of complete desynchronization. On the 3 flows test, one third of all loss events are partially synchronized and more than one third are completely desynchronized. On the 6 flows test, more than 50% of loss events involve only 2 or 3 flows; at the same time, 29% of loss events are completely desynchronized.

TABLE 4.1: Probability distribution of synchronization for 2, 3 and 6 AIMD flows

	$sync1$	$sync2$	$sync3$	$sync4$	$sync5$	$sync6$
2flows	40%	60%	–	–	–	–
3flows	36%	33%	31%	–	–	–
6flows	29%	29%	24%	10%	4%	4%

Figure 4.5 shows 12, 24 and 48 RENO flows. In each distribution, bell curves resemble Gaussian distribution. Especially for the 48 flows test, 95% of loss events are approximating a Gaussian distribution with mean of 7 flows and a standard deviation of 3 flows. This means most loss synchronization events involve a random subset of flows. Since the probabilities for those subsets are less than 10%, statistical multiplexing effect plays a significant role. Figure 4.5 also implies the global loss synchronization will be smaller when more flows are added.

Becase in each test all flows have the same TCP-CC algorithm, per-flow TCP-LS rates and the corresponding global TCP-LS rates are the same. Figure 4.6 summaries the global TCP-LS rates and the corresponding link utilization rates for RENO flows. Obviously, link utilization increases when synchronization rate decreases.

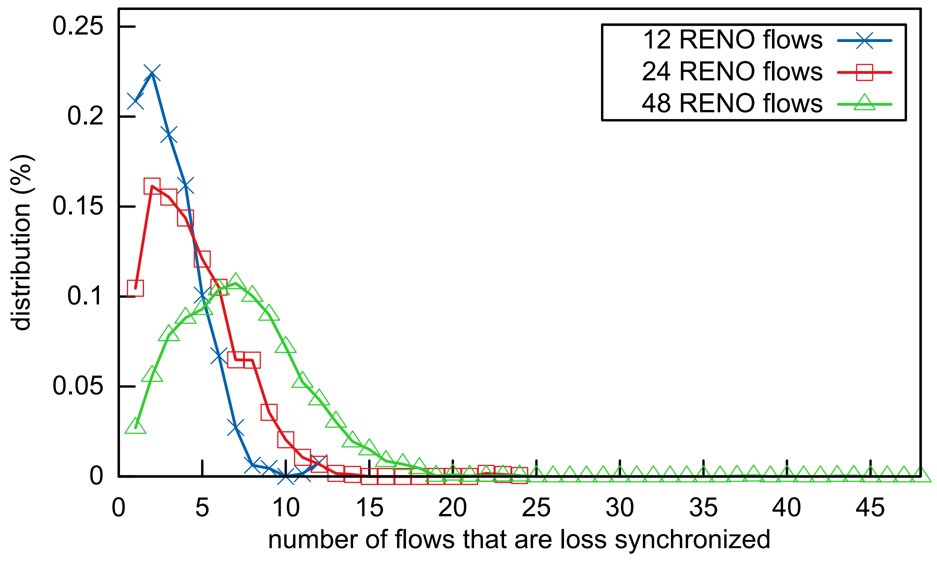


FIGURE 4.5: Distribution of synchronization for 12, 24 and 48 RENO flows

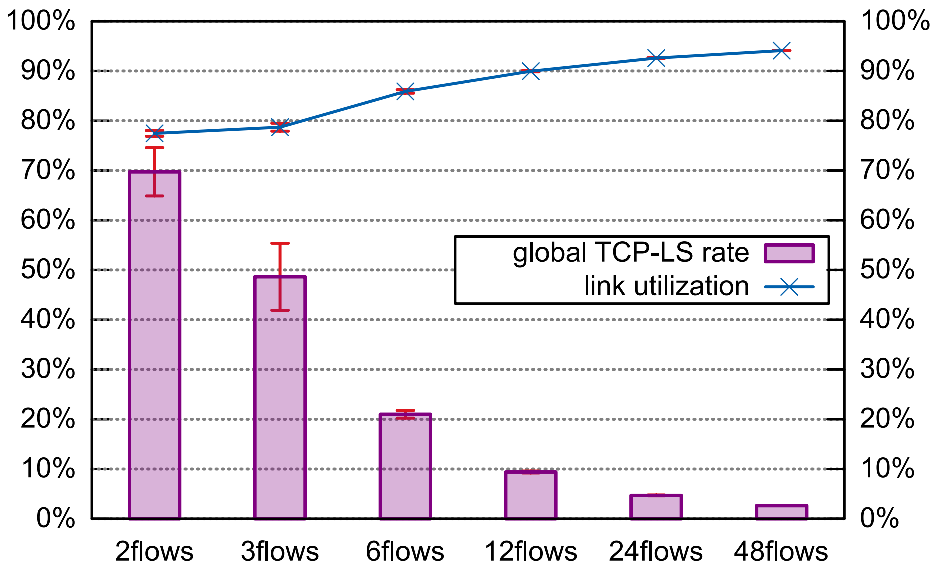


FIGURE 4.6: RENO's global TCP-LS rates and link utilization rates

We know that for AIMD TCP-CC mechanism, the *cwnd* size of a flow at time t is:

$$cwnd(t) = cwnd_{max}(1-\beta) + \alpha\frac{t}{RTT} \tag{4.3}$$

where $cwnd_{max}$ is the *cwnd* size just before the last window reduction, α and β are the increase parameter and the decrease parameter. Because we use RENO TCP to represent AIMD mechanism, $\alpha = 1$ and $\beta = 0.5$, by which a TCP flow increases *cwnd* roughly by one segment per *rtt* and cutting it by half at a congestion. As a result, these two policies form a sawtooth behavior.

Figure 4.7 shows *cwnd* behaviors of three RENO flows. Because of the per-flow sawtooth behavior, the aggregated load forms sawtooth too. However, the aggregated load has different degrees of loss based on various TCP-LS cases,.

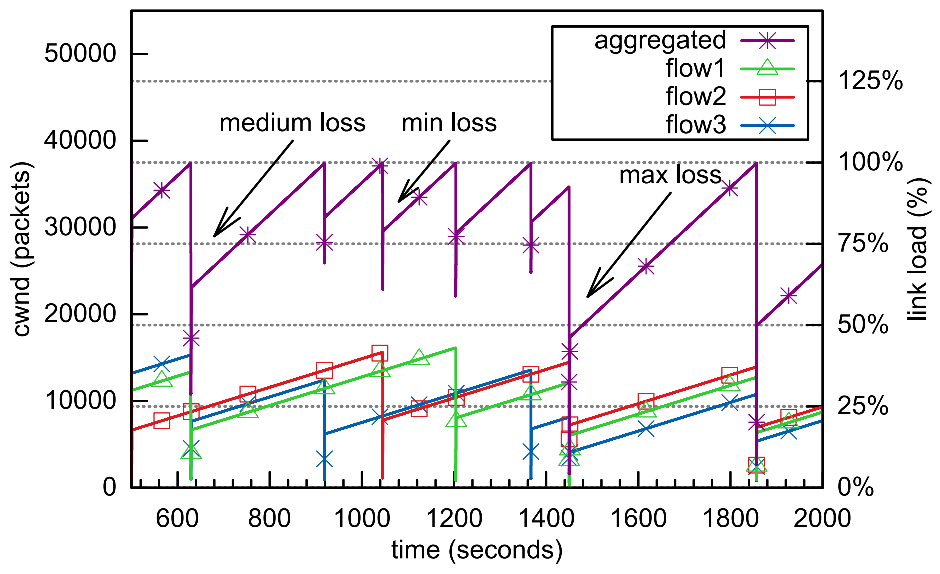


FIGURE 4.7: Congestion window behaviors of three RENO flows

Based on observation, bottleneck link utilization has lower and upper bounds. Mathematically, the link utilization U of n AIMD flows is calculated in Expression (4.4).

$$U = 1 - \frac{1}{2}(\frac{\beta}{n} \times i) \tag{4.4}$$

In Expression (4.4), i is the average number of loss synchronized flows, $i \in [1, n]$. The expression means the lowest and the highest link utilization (L and H) depend on the highest and the lowest TCP-LS rates, respectively. For L, it comes true when all the AIMD flows are completely loss synchronized, which means the aggregated load behaves like a single flow sawtooth. For H, it comes true when all the flows are completely loss desynchronized. Table 4.2 shows some numerical examples of link utilization for RENO flows.

In real world, flows' behaviors are more like chaotic dynamics [91]. The link utilization actually falls inside $[L, H]$. As shown in Figure 4.6, because the global TCP-LS

TABLE 4.2: Numerical examples of link utilization for RENO flows

Flows	Lowest Utilization (L)	Highest Utilization (H)
$n = 2$	75%	87.5%
$n = 3$	75%	91.7%
$n = 6$	75%	95.8%
$n = 12$	75%	97.9%
$n = 24$	75%	98.9%

rate is near 70% in the two flows test, the link utilization is 78%. As more flows are added, TCP-LS rate goes down. For example, the test of 12 RENO flows reaches 90% link utilization, when its global TCP-LS rate is 10%.

For 2, 3 and 6 flows tests, HSTCP has similar synchronization phenomena to RENO. For example, partial and complete desynchronization phenomena have large shares. When number of flows increases, the share of complete synchronization decreases significantly.

Unlike RENO, HSTCP have smaller sets of loss synchronized flows. As shown in Figure 4.8 for average distribution of synchronization phenomena of 12, 24 and 48 flows, the curves peak at 2 and decrease significantly. It means HSTCP flows are less liable to loss synchronization than RENO flows.

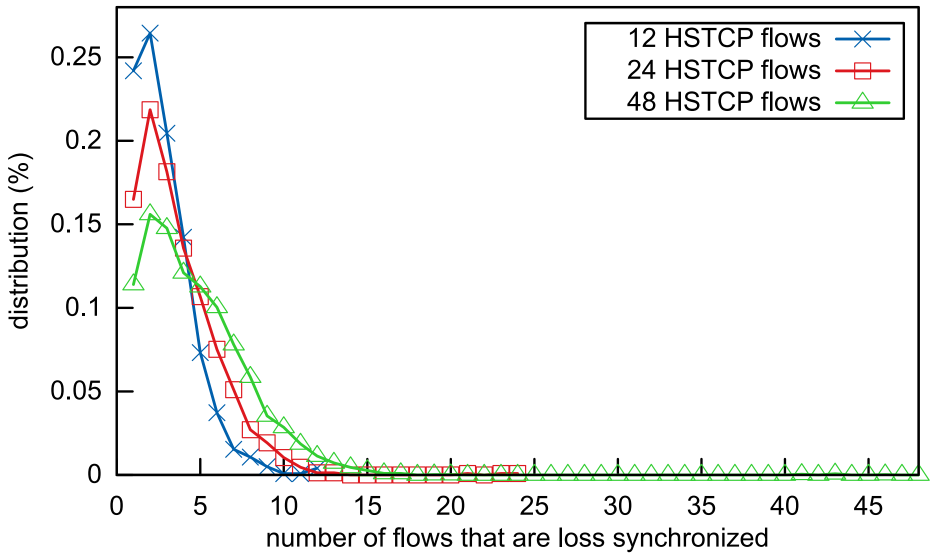


FIGURE 4.8: Distribution of synchronization for 12, 24 and 48 HSTCP flows

Figure 4.9 shows the global TCP-LS rates of HSTCP. All the link utilization rates are higher than RENO, not only because all the TCP-LS rates are smaller, but also because the decrease parameter β is smaller and increase parameter α grows faster (nonlinear). In Figure 4.10, the *cwnd* behaviors of three HSTCP flows are close to AIMD. Therefore, when β is around 0.2, Expression (4.4) still roughly works to interpret the lowest link utilization as 90% for three HSTCP flows.

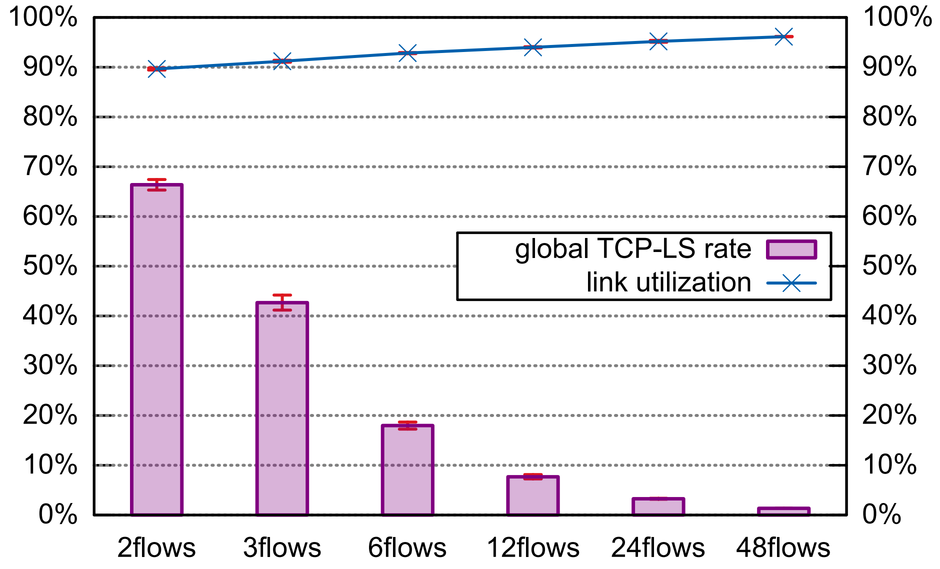


FIGURE 4.9: HSTCP's global TCP-LS rates and link utilization rates

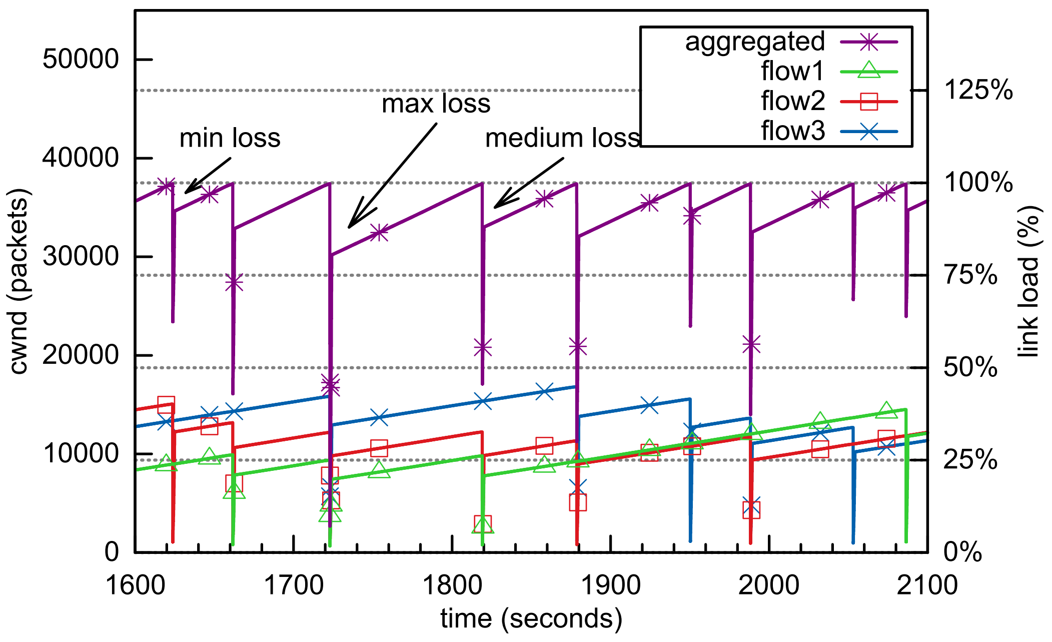


FIGURE 4.10: Congestion window behaviors of three HSTCP flows

Unlike RENO and HSTCP, CUBIC have more loss synchronized flows. From Table 4.3 and Figure 4.11, shares of complete synchronization and partial synchronization are highest among the three TCP variants. As shown in Figure 4.11, peaks of those bell curves locate on many flows. Therefore, more loss synchronized flows cause higher synchronization rates and lower link utilization.

TABLE 4.3: Distribution of synchronization for 2, 3 and 6 CUBIC flows

	$sync1$	$sync2$	$sync3$	$sync4$	$sync5$	$sync6$
2flows	27%	73%	–	–	–	–
3flows	18%	28%	54%	–	–	–
6flows	7%	12%	16%	24%	24%	17%

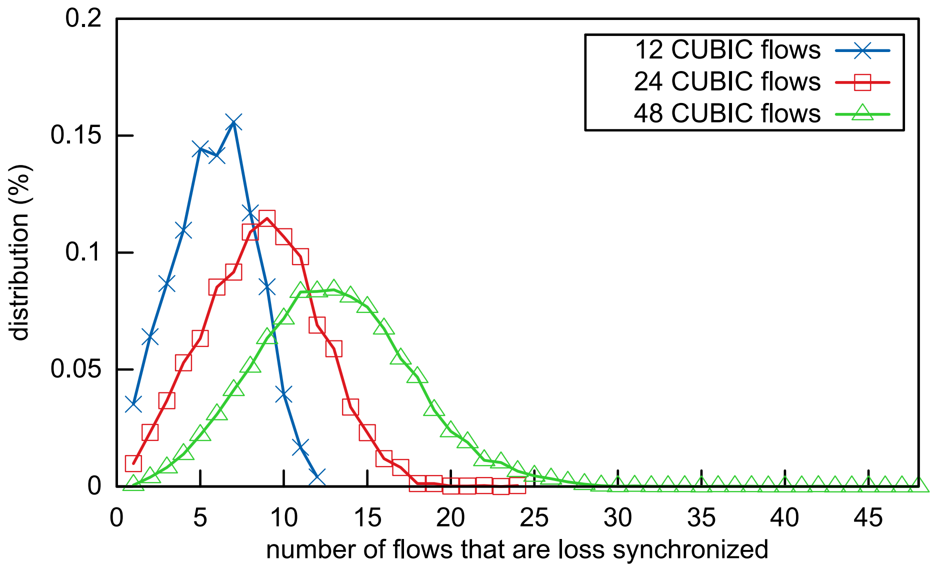


FIGURE 4.11: Distribution of synchronization for 12, 24, and 48 CUBIC flows

As shown in Figure 4.12, CUBIC has the highest TCP-LS rates of all. Link utilization rates are lower than these of HSTCP. When more flows are added, CUBIC shows worse link utilization than RENO. Despite CUBIC is the most widely used TCP-CC algorithm, its TCP-LS character shows unexpected performance.

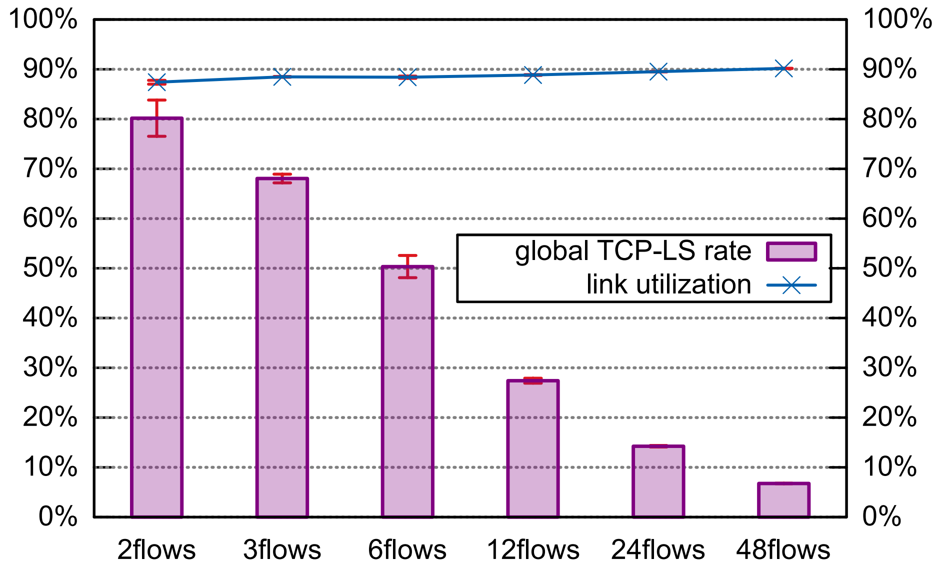


FIGURE 4.12: CUBIC's global TCP-LS rates and link utilization rates

Figure 4.13 shows *cwnd* behaviors of three CUBIC flows. Because CUBIC does not adopt linear increase, it is still our future work to interpret the link utilization boundaries.

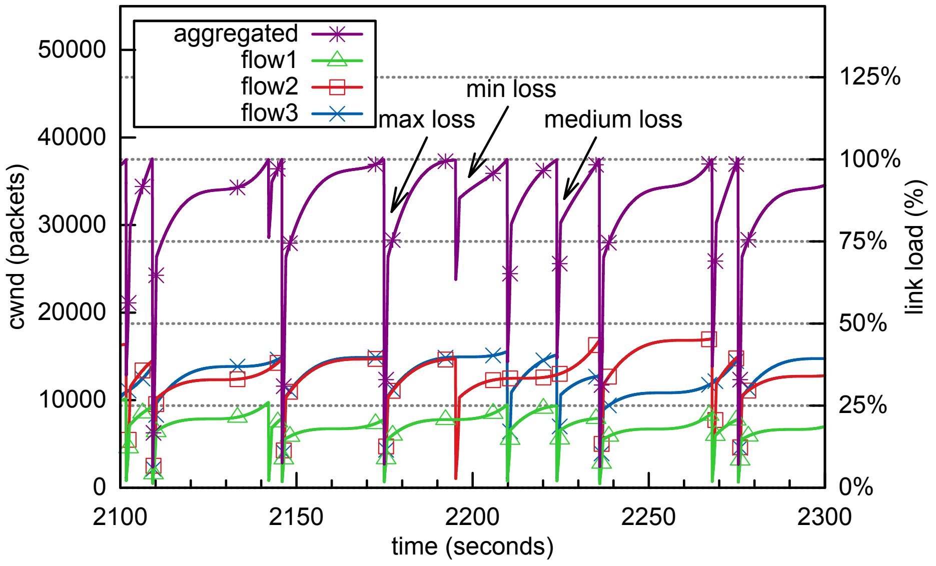


FIGURE 4.13: Congestion window behaviors of three CUBIC flows

4.5.2 Heterogeneous Experiment

As shown in Figure 4.14, global TCP-LS rates are similar to these of RENO and HSTCP. However, link utilizations are similar to these of CUBIC.

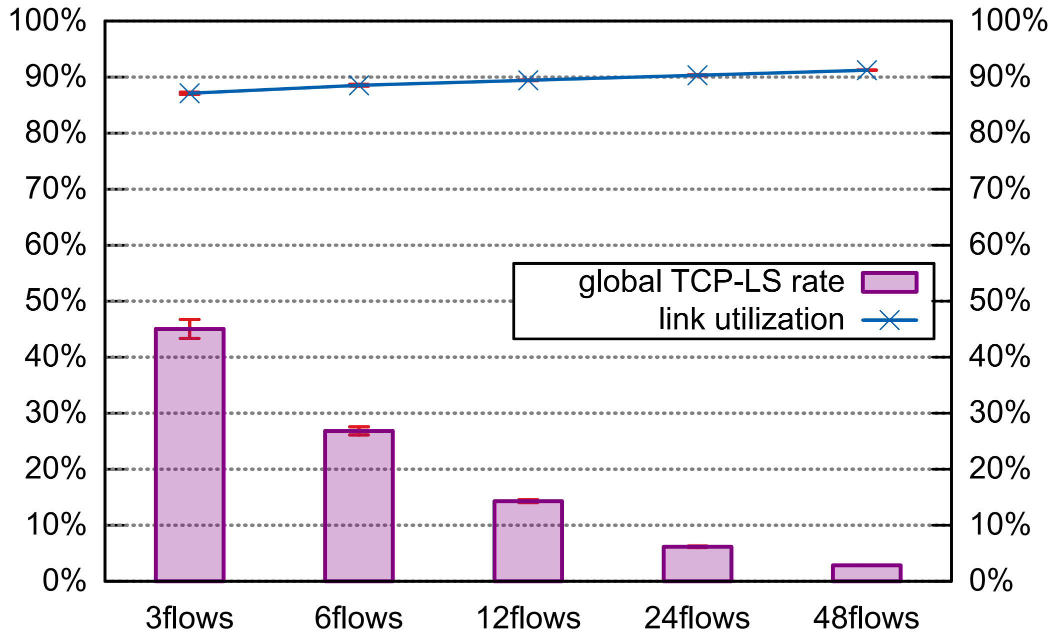


FIGURE 4.14: Global TCP-LS rates and link utilization rates of heterogeneous traffic

As we analyze the per-flow TCP-LS rates, we find they vary according to their TCP-CC algorithms. As shown in Table 4.4, CUBIC flows have higher per-flow rates than RENO flows and HSTCP flows. This means shares of partial TCP-LS or complete desynchronization are similar to these of RENO or HSTCP, and it explains why heterogeneous traffic has TCP-LS rates similar to RENO and HSTCP.

TABLE 4.4: Per-flow rates of CUBIC, HSTCP, RENO and the global rates in three and six flows test

	CUBIC	HSTCP	RENO	global rate
3 flows	62%	47%	34%	45%
6 flows	40%	26%	21%	27%

Figure 4.15 shows *cwnd* behaviors of three heterogeneous flows. Because of the unfairness caused by CUBIC, the aggregated load behaves like a CUBIC flow. This explains why heterogeneous traffic has link utilization rates similar to CUBIC, and it is obvious a big weakness for networks shared by different TCP flows.

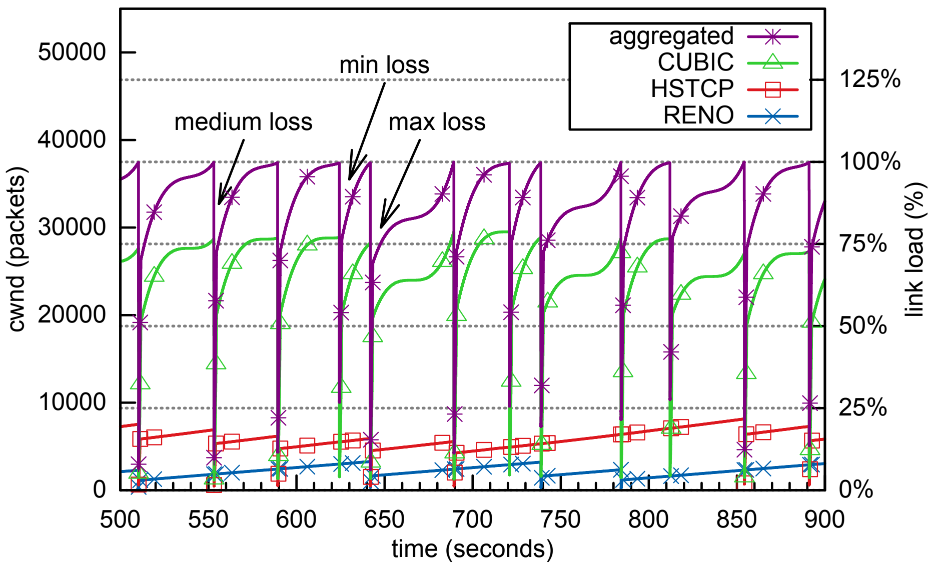


FIGURE 4.15: Congestion window behaviors of heterogeneous traffic. CUBIC flow has the highest *cwnd* and RENO flow has the lowest *cwnd*.

4.6 Conclusion

In this chapter, we analyzed TCP loss synchronization among long-lived flows and proposed two new synchronization expressions by carefully examining the loss events. Through a 10Gbps testbed, we obtained the measurements based on an "appropriate" 5% BDP drop-tail buffer with common *rtt*, and demonstrated the relationship between synchronization and link utilization. A small buffer may not be sufficient for AQMs like RED, but it maintains good link utilization when high-speed TCP variants are applied. We also showed degrees of synchronization of three popular TCP variants within homogeneous and heterogeneous traffic. Although fairness issue exists in heterogeneous traffic, HSTCP is more effective for link utilization than RENO and CUBIC in homogeneous traffic.

Chapter 5
Desynchronized Multi-Channel TCP for High Speed Networks with Tiny Buffers

Regarding the issues of router buffer sizing, an acceptable link utilization of the tiny-buffered core routers over backbone networks requires (i) the backbone links are over-provisioned, and (ii) the network traffic is not bursty, and (iii) tens of thousands of asynchronous TCP flows. However, in high-speed access networks, traffic is bursty and the number of TCP connections is below hundreds with less asynchronous granularity. It is almost impossible to reach an acceptable link utilization for such networks with tiny buffers.

In this chapter, we focus on bottleneck link utilization with goals to minimize TCP loss synchronization and to reduce traffic burstiness impact. We present a new congestion control algorithm called Desynchronized Multi-Channel TCP (DMCTCP). Our algorithm implementation takes advantage of a potential parallelism from the Multi-Path TCP in Linux. Over an emulated 10 Gb/s large delay network ruled by routers with only a few dozen packets of buffers, our experimental results confirm that bottleneck link utilization can be improved to reach more than 80% with just 100 connections.

5.1 Background

Deciding an appropriate buffer size at a high-speed router of Internet has been debated for years among network researchers. The traditional guidance on maintaining a fully utilized link while TCP ramps up its congestion window (*cwnd*) suggests a Bandwidth-Delay-Product (BDP). This *rule-of-thumb* decides the amount of buffering by B = RTT × C [31], where C is the capacity of a bottleneck link and RTT is the round-trip-time of a TCP connection. For a large RTT of 250 ms, a router with a

C = 40 Gb/s link capacity requires 10 Gb of buffering, which is costly and challenging to design and deploy over high-speed networks [4].

Recently, several arguments on buffer sizing at Internet core routers have been put forth. Based on queuing analysis, studies from [92, 93] proposed a *tiny-buffer model* to significantly reduce buffer to a size of $O(logW)$, where W is congestion window size. They recommended that a few dozen packets of buffering can suffice an acceptable link load for TCP traffic, e.g. 75% utilization of a 40 Gb/s link. This model has been examined with promising results from several 1 Gb/s network experiments [94]. However, it relies on the traffic conditions that tens of thousands of TCP flows are neither bursty nor synchronous. These conditions only exists in Internet backbone, where tens of thousands of TCP flows are spread out through over-provisioned core links.

Therefore, we explore if and how we could achieve an acceptable link utilization in high-speed access networks with very small router buffer. First, it could facilitate the deploy of all-optical routers because they have huge link capacity and lower power requirement but are limited by buffer size [9]. Second, it could reduce the requirement of router complexity, making routers easier to build and easier to scale. Third, it could minimize queuing delay and jitter that were closely related to buffer size in electronic routers.

However, we found it is more challenging for performance in access networks with very small buffers because conditions are different in the following aspects: (i) the access links have congestion, and (ii) the network traffic is bursty, and (iii) the number of flows is too small to reach an expected level of asynchronism. We believe most of the problems come from TCP congestion control, as it is an integral module of TCP that directly determines the performance of the protocol. Therefore, we focus on the design of a new TCP congestion control algorithm called Desynchronized Multi-Channel

TCP (DMCTCP), which pursues goals of minimizing TCP loss synchronization and reducing the impact of traffic burstiness.

The key ideas behind DMCTCP are to prevent simultaneous transmission rate cuts among multiple channels of a TCP connection and to prevent transmission rate cuts from burstiness congestion. The algorithm was inspired by parallel TCP and Multi-Path TCP [95] (MPTCP), but with the important distinctions that various congestion events are detected and distinguished with different corresponding actions, and that no modifications to other network layers (such as the application layer) of the end-to-end system need to be made.

5.2 The Multipath TCP

Today, most smartphones support at least 3G and 802.11, and so do tablet PCs like Apples iPad. This has increased interest in using several access mediums in the same connection, so that it becomes possible to transparently change from one medium to another in case of failure. Further, using several paths simultaneously can improve end-to-end throughput with load balance.

The transport layer is the best place to implement multipath functionality because of the high amount of information it collects about each of the paths (delay/bandwidth estimation), and its knowledge of the application byte stream. The network may know path properties, but simply scattering packets of a single transport connection over multiple physical paths will typically reorder many packets, confusing the transport protocol and leading to very poor throughput. The application could implement multipath, but such changes are not easy to get right. If just simply switched from TCP to multipath TCP while maintaining the reliable byte stream semantics, unmodified application could benefit immediately.

Multipath TCP, as proposed by the IETF working group [95], allows a single data stream to be split across multiple paths. This has obvious benefits for reliability that the connection can persist when a path fails. It can also have benefits for load balancing at multi-homed servers and data centers, and for mobility, as shown below.

In the traditional Internet architecture, network devices operate at the network layer and lower layers, with the layers above the network layer instantiated only at the end hosts. As shown in Figure 5.1, while this architecture was initially largely adhered to, this layering architecture no longer reflects the "ground truth" in the Internet with the proliferation of middleboxes [RFC 3234] [96]. Middleboxes routinely interpose on the transport layer; sometimes even completely terminating transport connections, thus leaving the application layer as the first real end-to-end layer, as shown in Figure 5.2.

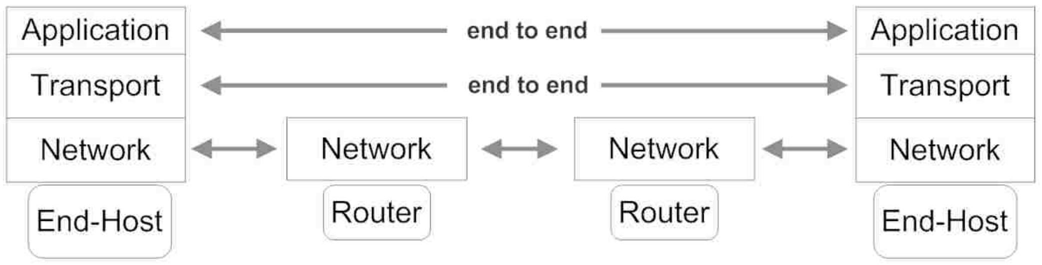


FIGURE 5.1: Traditional Internet Architecture

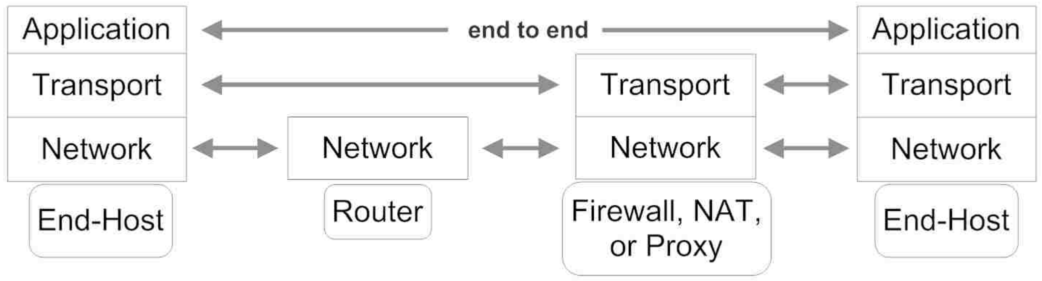


FIGURE 5.2: Internet Reality

Middleboxes that interpose on the transport layer result in loss of "fate-sharing" [RFC 2018] [97], that is, they often hold "hard" state that, when packets are lost or corrupted, the end-to-end transport connection will be lost or corrupted.

The network compatibility goal requires that the multipath extension to TCP retain compatibility with the Internet as it exists today, including making reasonable efforts to be able to traverse predominant middleboxes such as firewalls, NATs, and performance-enhancing proxies [RFC 3234] [96]. This requirement comes from recognizing middleboxes as a significant deployment bottleneck for any transport that is not TCP or UDP, and constrains Multipath TCP to appear as TCP does on the wire and to use established TCP extensions where necessary. To ensure compatibility of the transport layer, Multipath TCP preserves fate-sharing without making any assumptions about middlebox behavior.

The modifications to support multiple paths remain at the transport layer, although some knowledge of the underlying network layer is required. Multipath TCP also works with IPv4 and IPv6 interchangeably, i.e., one connection may operate over both IPv4 and IPv6 networks.

MPTCP makes use of standard TCP sessions, termed “subflows”, to provide the underlying transport per path. MPTCP-specific information is carried in a TCP-compatible manner, although this mechanism is separate from the actual information being transferred so could evolve in future revisions. Figure 5.3 illustrates the layered architecture of MPTCP.

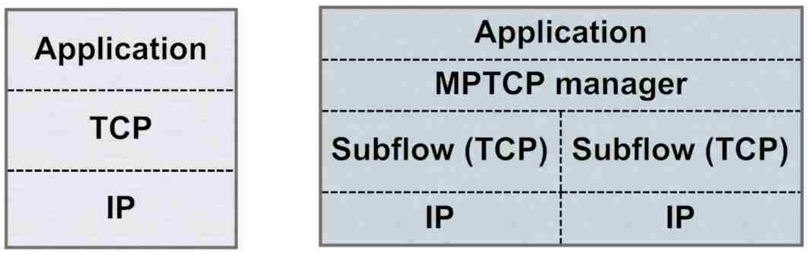


FIGURE 5.3: Comparison of Standard TCP and MPTCP Protocol Stacks

In situation below the application layer, the MPTCP extension in turn manages multiple TCP subflows below it. In order to do this, it implements the following functions:

- Path Management: This is the function to detect and use multiple paths between two hosts. MPTCP uses the presence of multiple IP addresses at one or both of the hosts as an indicator of this. The path management features of the MPTCP protocol are the mechanisms to signal alternative addresses to hosts, and mechanisms to set up new subflows joined to an existing MPTCP connection.

- Packet Scheduling: This function breaks the byte stream received from the application into segments to be transmitted on one of the available subflows. The MPTCP design makes use of a data sequence mapping, associating segments sent on different subflows to a connection-level sequence numbering, thus allowing segments sent on different subflows to be correctly re-ordered at the receiver. The packet scheduler is dependent upon information about the availability of paths exposed by the path management component, and then makes use of the subflows to transmit queued segments. This function is also responsible for connection-level re-ordering on receipt of packets from the TCP subflows, according to the attached data sequence mappings.

- Subflow (single-path TCP) Interface: A subflow component takes segments from the packet-scheduling component and transmits them over the specified path, ensuring detectable delivery to the host. MPTCP uses TCP underneath for network compatibility. Because TCP ensures in-order, reliable delivery, each TCP flow adds its own sequence numbers to the segments; these are used to detect and retransmit lost packets at the subflow layer. For MPTCP, it uses a dual sequence number space, where each subflow has its own sequence space that identifies bytes within a subflow as if it were running alone. There is also a data (or connection level) sequence space, which allows reordering at the aggregate connection level. Each segment carries both subflow and data sequence numbers.

- Congestion Control: This function coordinates congestion control across the subflows. As specified, the congestion control algorithm ensures that an MPTCP connection does not unfairly take more bandwidth than a single path TCP flow would take at a shared bottleneck.

5.3 Multipath TCP implementation in Linux

The architecture of MPTCP implementation is depicted in figure 5.4. All legacy TCP applications directly benefit from the added multipath capability. When a new TCP flow is started, multipath TCP adds the multipath capable option to the SYN packet. If the end-point replies with a SYN/ACK containing the multipath capable option, the connection is then multipath enabled.

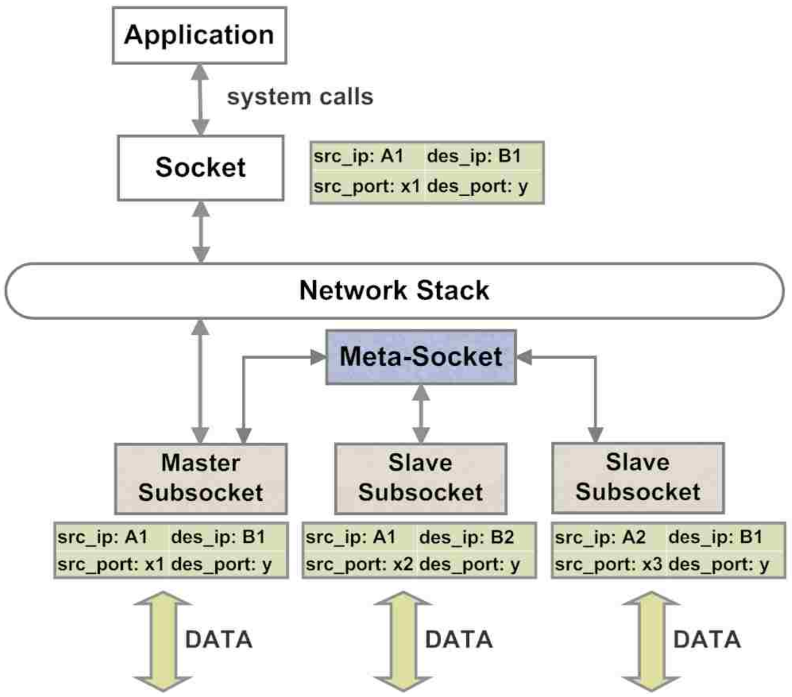


FIGURE 5.4: MPTCP architecture

Connection-specific information is held in a new structure at the connection-level, called meta-socket. This structure keeps multipath identifiers for the connection, the list of subflows associated to this connection, and connection-level reordering queues. Initially there is a single TCP socket opened (the master socket), corresponding to the first subflow in the connection. When additional subflows are opened, new socket structures are created and associated to the meta-socket. The master socket is a special socket as it is the only connection to the application. Application writes to this socket are redirected to the meta-socket which segments the byte stream and decides which subflow should send each segment. Application reads from this socket are serviced from the meta-socket's receive buffer.

Data arriving on the subflows is serviced by the master and slave sockets (checking for in-order, in window sequence numbers, etc.), and passed to the meta-socket once it is in order at subflow level. Here the data is reordered according to the connection sequence number, which is carried in each TCP segment as an option. Retransmissions are driven only by the subflow sequence number. Therefore, MPTCP avoids problems due to connection level reordering of packets.

Additional subflows are only opened after the initial handshake succeeds. The stack checks to see if it has multiple addresses that have routes to the destination; if so it will try to open subflows using currently unused addresses (in the picture this could be address A2). To get around NATs, addresses are also signalled explicitly to the remote end using TCP options.

Subflows are created with the usual three way handshake with SYN packets carrying a "Join" option and a connection identifier. SYN de-multiplexing is done using this connection identifier, and not the destination port as in regular TCP.

The implementation allows opening subflows between different address pairs, or between the same address pairs but different ports. The latter can be used to leverage existing in-network multipath solutions such as Equal Cost Multipath (ECMP), allowing them to load balance at subflow granularity. Finally, this implementation is modular and it is easy to add support for new path management techniques that may become available.

In summary, MPTCP is a proposed TCP extension to use multiple linked paths for a single TCP connection. It manages data streams called subflows of a single connection among multiple paths. In Linux kernel, it currently has two implemented congestion control algorithms named MPTCP-Coupled [98] and MPTCP-OLIA [99], which ensure that an MPTCP connection does not unfairly take more bandwidth than a single path TCP flow would take at a shared bottleneck. MPTCP is likely to work correctly in the Internet through different middleboxes [100]. Study in [101] shows robust performance of MPTCP with a few subflows per-connection in a multiple Fat-Tree-like data center environment. But the related buffer sizing issues are not discussed.

5.4 Motivation: Problems of high-speed Networks with Tiny Buffers

5.4.1 The TCP Loss Synchronization Analysis

Based on the standard Additive-Increase/Multiplicative-Decrease (AIMD) TCP congestion control mechanism, a highly simplified macroscopic model for the steady-state behavior [102] is expressed as follows:

$$\text{average throughput of a connection} = \frac{0.75 \cdot W}{RTT} \tag{5.1}$$

Consider a particular RTT and segment size, the average throughput of a flow will be roughly 75% of its largest congestion window W. If we do not include buffer effect, W will be cut in half when the transmission rate reaches the bottleneck link capacity C and then increases by one segment every RTT until it again reaches W. Therefore, the average throughput of a flow in a bufferless bottleneck link is simplified as:

$$\text{average throughput of a connection} = 0.75 \cdot C \tag{5.2}$$

Interestingly, our experimental study of TCP loss synchronization shows similar pattern when many flows are through a bottleneck link with small buffer. We define a complete TCP loss synchronization event happens when all the flows are experiencing packet drops in a congestion event. As a result, all the flows cut their rates in half at the same time and the bottleneck link is underutilized. This congestion event includes at least n packet drops where n is the number of flows. When n is small, it is highly probable to have many complete loss synchronization events. Figure 5.5 geometrically shows two generalized scenarios of complete loss synchronization. As can be seen from the figure, when buffer effect is not included, the aggregated congestion window of three flows is cut in half on each complete loss synchronization event and follows the same sawtooth pattern as a single flow in (5.1), no matter what RTT each flow has. Therefore, the bottleneck link utilization becomes 75%.

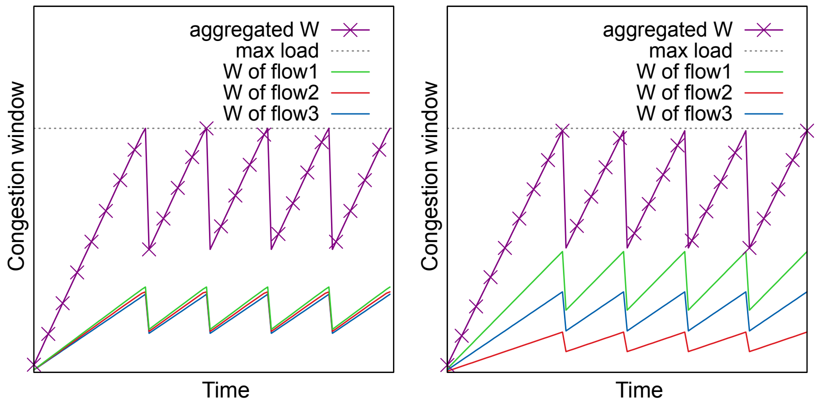


FIGURE 5.5: Two extreme scenarios of the complete loss synchronization: flows with similar RTTs (left) and flows with different RTTs (right)

The analysis above triggers our goal to completely desynchronize TCP flows. In order to desynchronize, ideally a congestion event should include at most one packet drop such that only one flow experiences rate cut once upon a time. By enumeration, Figure 5.6 shows an idealized TCP desynchronization that improves bottleneck link utilization. However, it is almost impossible to only drop a single packet when the buffer is full, especially for tiny buffers that can only hold a few dozen packets. This means Active Queue Management (AQM) mechanisms such as random early detection (RED) will not work as expected. They will mostly behave like a drop-tail buffer because when the buffer is too small, they cannot absorb the large bursts due to the faster window growth inherent in TCP protocols [13]. Therefore, TCP congestion control becomes the target to find a desynchronization oriented solution.

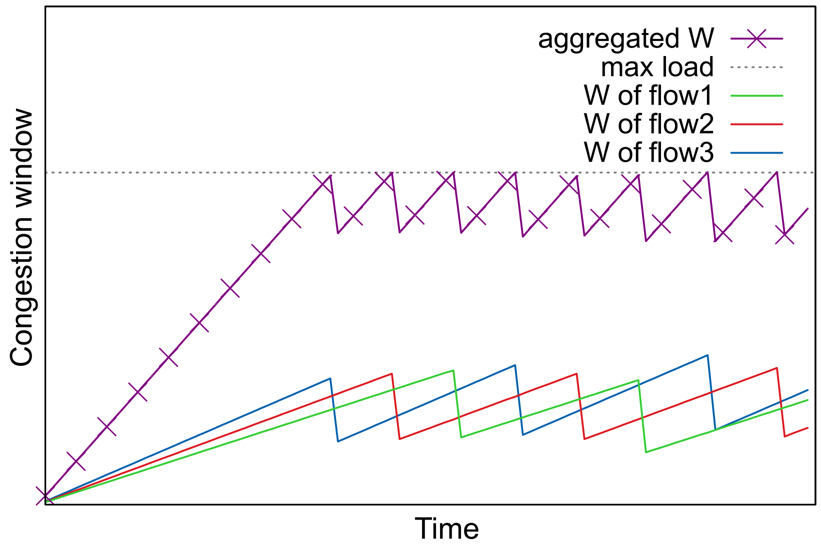


FIGURE 5.6: The completely desynchronized TCP flows

Also as shown in Figure 5.6, it is clear that drop the biggest flow which has the largest congestion window will balance fairness among multiple flows. However, this requires coordination that flows must communicate with each other that smaller flows should not cut transmission rate in the congestion even when packet losses are detected. Because of the network socket legacy, this communication requirement leaves a big challenge on TCP variants that manage single-flow per-connection.

5.4.2 The Burstiness Analysis

In high-speed networks with tiny buffers, the link utilization could be worse. Not only because of TCP loss synchronization, but also because of the inherent burstiness of TCP flows. However, by applying CPU overhead saving techniques such as Interrupt Coalescing and TCP Segmentation Offloading, traffic burstiness is exaggerated. As a result, the normal TCP ACK-clocking is disrupted and packets are burst out of the NIC at line rate. Because these techniques have become standards for high-speed networks beyond 10 Gb/s, burstiness is not avoidable and it induces complex and expensive workarounds [94, 103].

We define two types of congestion that causes packet losses: (a) bandwidth congestion that is caused by the high utilization of bottleneck link among competing flows and (b) burstiness congestion from random burst contention that occurs even when bandwidth utilization is low. Clearly the second congestion type should be avoided if such congestion can be detected and distinguished. This brings another challenge because most loss-based TCP variants use packet losses (duplicated ACKs) as a signal of bandwidth congestion and verify burstiness congestion is a hard work [104].

5.5 The DMCTCP Congestion Control

Inspired by parallel TCP and Multi-Path TCP [95], DMCTCP pursues a goal of minimizing TCP loss synchronism in a high-speed tiny-buffered network environment where bandwidth congestion and burstiness congestion coexist. A DMCTCP connection has multiple channels. These channels carry split data through a single path of a connection from the application layer at a sender and reassemble the data at the corresponding receiver. Each channel has its own congestion window so that a basic communication among channels can be used to detect and distinguish the two congestion events. Therefore, DMCTCP establishes a channeled parallelism in a path where TCP loss synchronism and impact of burstiness can be minimized.

5.5.1 Detailed design of DMCTCP

Let m ($m \geq 2$) be the number of channels of a TCP connection through a single path. Obviously, at least two channels are required to communicate with each other when congestion happens. We denote by w_i the congestion window of channel i ($i \in m$), by w_{max}, w_{min} and w_{total} the largest, the smallest and the aggregated congestion window at time t in m respectively, and by the time stamp $time_i$ of a detected loss in channel i and by the time stamp $time_c$ of the last rate cuts. We also assume all channels have the same round-trip-time rtt because they are through the same path with a negligible queuing variance. Our algorithm is as follows:

- For each loss on channel $i \in m$, decrease w_i by:
    - $w_i/2$, if $w_i = w_{max}$ and $(time_i - time_c) > rtt$, then $time_c \leftarrow time_i$,
    - 0, otherwise.
- For each ACK on channel $i \in m$, increase w_i by:
    - $1/w_i$, if $w_i = w_{min}$,
    - $1/(w_{total} - w_{min})$, otherwise.

The decrease rule decides only the largest channel can be cut in half in a congestion event and this rule de-synchronizes consecutive cuts by guaranteeing these cuts are not within the same congestion event (one RTT). The increase rule simulates a parallel TCP with roughly $\alpha = 2$ so that the aggregated window w_{total} of the connection is capped at $2/w_i$. The difference is that the smallest channel can grow a little faster so that the aggregated window will have a higher leveraged sawtooth pattern.

5.5.2 Implementation

We implemented DMCTCP by taking advantage of a potential parallelism in the MPTCP release supported in the Linux kernel 3.5.0 [105]. Each subflow of a connection in MPTCP is a channel in DMCTCP, with a distinction that multiple channels are through a single path. The fast retransmit and fast recovery algorithms, as well as the Selective Acknowledgment (SACK) option are the same as in TCP. To compute a smoothed estimate of rtt, we use the term $srtt_i$ as implemented for each MPTCP subsocket. Because a master subflow is always initiated before slave subflows can be created in MPTCP, so is the master channel goes ahead of slave channels. However, it may create a large burst when all channels begin their slow-start phase in an overlapped time period. Therefore, we we set all slave channels to share the initial slow-start threshold after the master channel gets cut at the first time.

The time complexity of our implementation depends on the linearly searching for w_{max}, w_{min} and w_{total}. This can be done easily in a one level loop. Because the maximum subflows that can be created in current MPTCP release is 32, hence our number of channels is restricted to 32. As a result, the time complexity is almost constant.

5.5.3 Illustrative Example of DMCTCP

There is a traffic sniffing tool called TCP-Probe [106], which is a kernel module to record the state of a TCP connection in response to incoming ACKs. We extended TCP-Probe to record multiple channels of DMCTCP connections. We also mark the timestamp as loss point when each channel detects congestion and start to fast retransmit. Figure 5.7 shows each channel's congestion window (m=2) as a function of time for a DMCTCP connection. As can be seen from the figure, these loss points show when a congestion happens and how the congestion control reacts to a congestion type: (a) the smaller channel detects burstiness congestion and just fast retransmit the lost packets without cuts its sending rate, and (b) the largest channel detects bandwidth congestion, reduces its sending rate to half, and de-synchronizes multiple channels' cuts within the same RTT.

Our algorithm is still AIMD and loss-oriented. Therefore, it should have the same property of intra-protocol fairness and RTT fairness as in TCP-SACK. However, because TCP loss synchronization is minimized, the convergence time of two competing connections is also reduced. Figure 5.8 illustrates this feature by showing the convergence times between two consecutively started (30 seconds interval) connections using the same congestion control algorithm through a 100 Mb/s bandwidth 100 ms delay bottleneck link with 128 pakcets of buffer. The convergence time is defined to be the

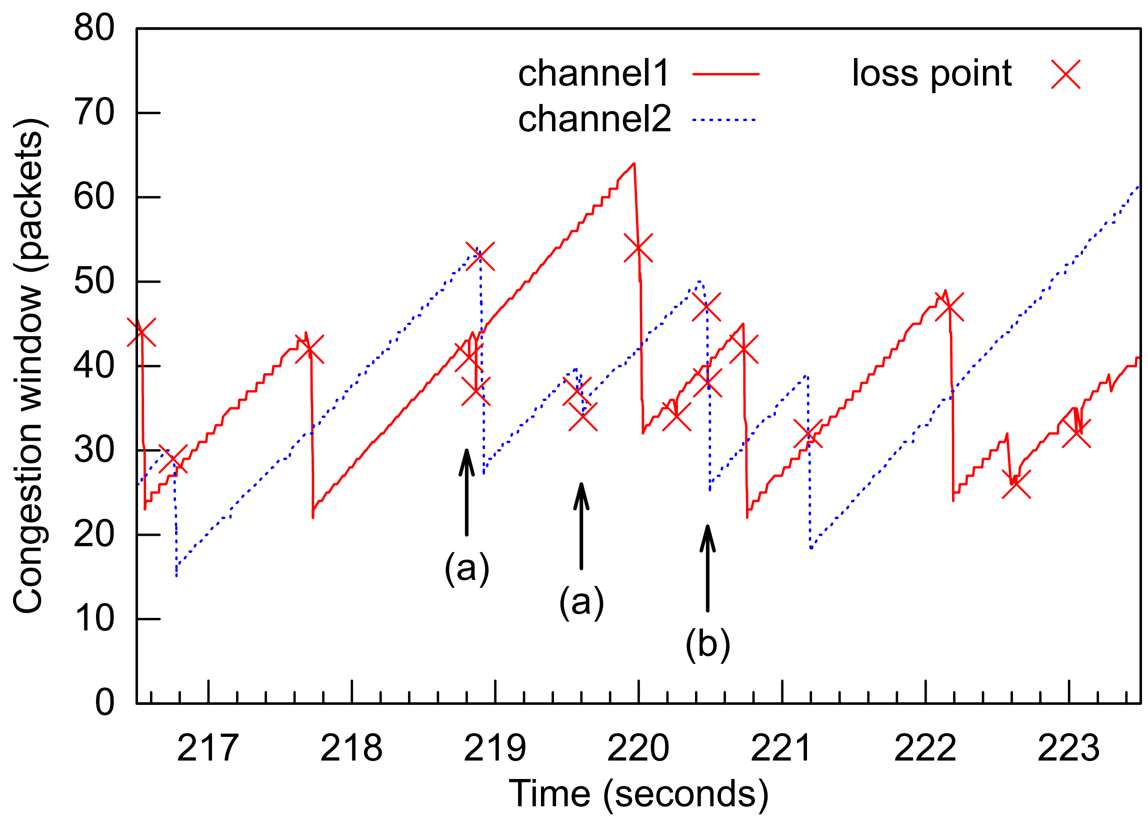


FIGURE 5.7: Steady state behavior of a DMCTCP connection, m=2

elapsed time when the congection window of the second flow reaches roughly 80% of the first flow. As shown in Figure 5.8(a), the convergence time for two TCP-SACK connections is almost 330 seconds. As shown in Figure 5.8(b), 5.8(c) and 5.8(d), the convergence time as compared with aggregated congestion windows for two DMCTCP connections is greatly reduced when the channel number increases.

5.5.4 Determine a good number of channels

However, it is still unknown that how much of synchronization can be canceled and how much of performance can be improved. On the other hand, a large number of channels render additional overhead such as computing states of channels and reassembling data among channels. To leverage performance and overhead, it is desirable to determine how many channels is good enough.

Ideally, the aggregated window of a DMCTCP TCP connection with completely desynchronization is expected to have regular variation between peaks and valleys.

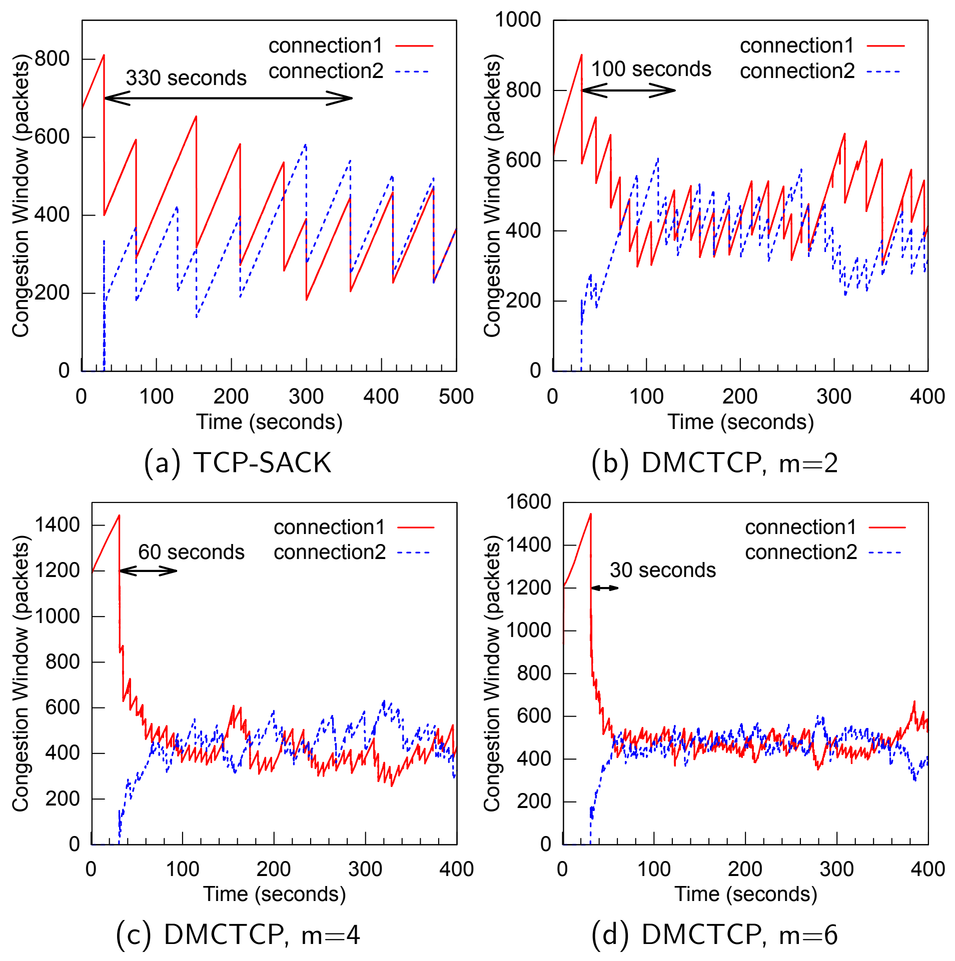


FIGURE 5.8: Convergence time of two connections in a 100 Mb/s bandwidth, 100 ms delay bottleneck link

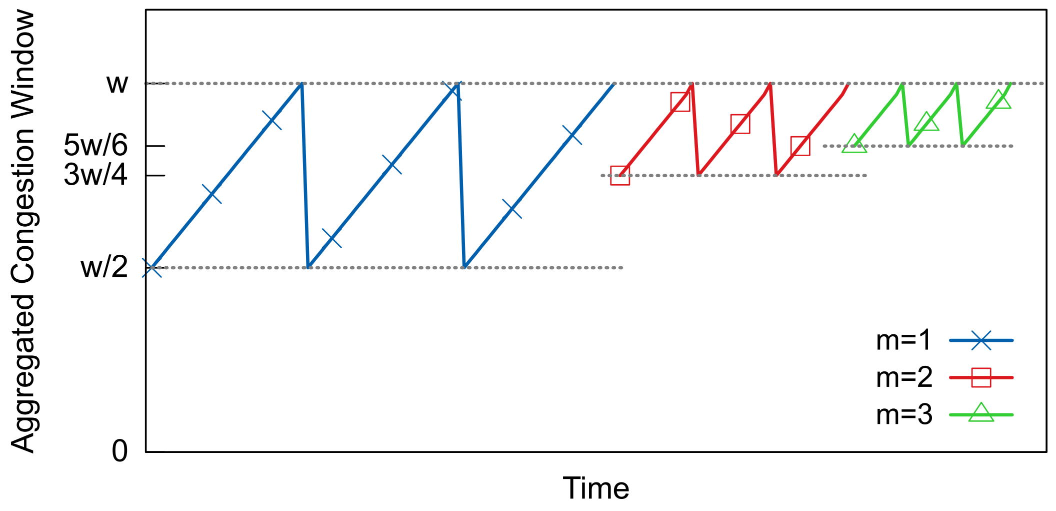


FIGURE 5.9: The idealized aggregated window of various channels per-connection

This is geometrically illustrated in Figure 5.9, which shows the idealized aggregated window of two and three channels per-connection, except that it falls back to normal TCP when channel number equals one. Based on the geometric analysis of regular distributed sawtooth window, we found a simplified macroscopic model for steady-state behavior of a Multi-Channel TCP connection:

$$\text{average throughput} = (1 - \frac{\beta}{2 \cdot m}) \cdot \frac{W}{RTT} \tag{5.3}$$

where β is the decrease parameter in each cut. Applied in a bottleneck link with $\beta = 1/2$, we can get the average link utilization in Table 5.1 when m increases.

TABLE 5.1: Numerical examples of ideal link utilization by DMCTCP

Number of Channels	Link Utilization
$m = 1$	75%
$m = 2$	87.5%
$m = 3$	91.7%
$m = 4$	93.8%
$m = 5$	95%
$m = 6$	95.8%
$m = 7$	96.4%
$m = 8$	96.9%
...	
$m = 25$	99%

Clearly, five channels are good enough to significantly achieve 95% of link utilization. Beyond five channels, the increment will be smaller and smaller since (5.3) is a rational function with a horizontal asymptote line at 1 when $m \geq 2$. Because the overhead of data reassembling and reordering grows up as the channel number increases, we decided the channel number to be 5.

5.6 Testbed Design

To evaluate the performance of DMCTCP, we created a 10 Gb/s dumbbell testbed shown in Figure 5.10 as an example of access network from CRON [63] platform.

In this testbed, we evaluate the performance of DMCTCP with a small number of long-lived TCP flows and some background traffic.

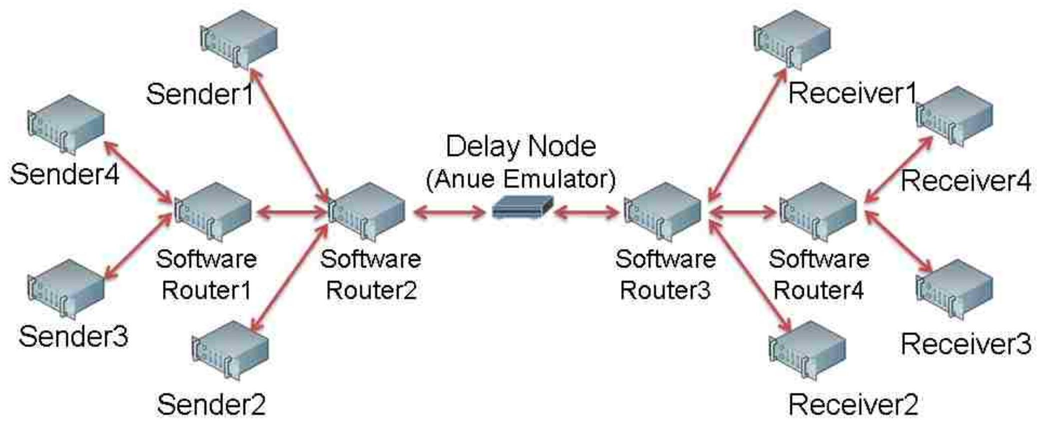


FIGURE 5.10: 10Gbps access network testbed

Sender and receiver nodes are HP DL160G6 servers with two six-core 2.7 GHz processors and 16 GB RAM. Router nodes are identical SUN Fire x4240 servers with two quad-core 2.7 GHz processors and 8 GB RAM. All nodes have Myricom 10 Gb/s NICs so that the bottleneck link rate confirms to be 10 Gb/s. Delay node in the middle is an Anue XGEM optical hardware emulator, which is a precision test instrument for 10 GE emulation. We use this hardware emulator for providing bi-directional communication delay at 60 ms, 120 ms, 180 ms and 240 ms between $Sender_i$ and $Receiver_i$ respectively.

All the nodes use Ubuntu Server 12.04. Each end-host has a MPTCP enabled Linux 3.5.0 kernel to test TCP-SACK, CUBIC, MPTCP-Coupled, MPTCP-OLIA and with our implementation to test DMCTCP. These TCP end-hosts enabled Interrupt Coalescing and TCP Segmentation Offload as a default setting to save CPU overhead, and are configured to have a very large buffer so that the transmission rates of high-speed flows are only limited by the congestion control algorithm. The routers use a normal Linux 3.5.0 kernel with the latest NIC driver. We enabled packet forwarding and set up the FIFO drop-tail queuing policy in each router.

Because we have used an emulation-based network testbed, it is important to make our testbed environment similar to the real network environment which has tiny buffers. We empirically found out that more system tunings at data link layer and network layer are necessary. To implement a tiny buffer environment, we need to reduce both the *tx_ring*\rx_ring at layer 2 in NIC and *netdev_backlog*\\qdisc at layer 3 in kernel. Unfortunately, the size of *tx_ring* and *rx_ring* are 1024 and 512 descriptors. They are inside an appropriate firmware of Myricom NIC and can't be changed. Instead, we hacked the NIC driver to only use a portion of the *tx_ring*. As a result, we can reduce the *tx_ring* to 128 descriptors. Then, we set packet size (MTU) to 8000 bytes because each descriptor in *tx_ring\rx_ring* holds up to 4096 bytes (page size) of data. This means 2 descriptors can buffer a packet. We also set *netdev_backlog*\\qdisc at 2. Because the bottleneck buffer is Router2's output port, our configuration reaches the real network environment with 66 packets of tiny buffers.

We use Iperf traffic generator to generate long-lived TCP connections in even numbers from $Sender_i$ to $Receiver_i$. These connections start within [0,10] seconds. One trial for each test lasts 900 seconds, and repeats 3 to 6 times to get arithmetic means of the throughput. Most standard deviations fall in the range of $\pm3\%$ such that they can be omitted in our results.

We also evaluate our experiments performed in the context of bursty background traffic. We consider two types of flows as the background traffic: short-lived TCP flows and real-time (UDP) flows from Router1 to Router4 with a fixed delay of 120 ms. We used Harpoon traffic generator to infinitely transmit short-lived TCP flows. The inter-arrival times between two successive connections are generated from exponential distribution with mean 1 second. The file sizes are generated from Pareto distribution with the shape parameter alpha = 1.5 (long-tail). The average file size is 1 MB from total 200,000 randomly sorted files. These values are realistic, based on comparisons with actual packet traces [94]. The aggregated throughput of short-lived TCP traffic is averaged at 165 Mb/s. We also added an additional 300 Mb/s UDP flow as an aggregate of many individual real-time streams. As a result, the average background traffic is nearly 4.6% of the bottleneck link capacity.

5.7 Performance Evaluation

5.7.1 Link Utilization

We first verify the performance improvement by increasing the number of channels. Figure 5.11 shows the average link utilization without background traffic from the aggregated throughput of all connections in each test. It shows that utilization gets higher when the number of channels increases. With five or six channels per-connection, it is the first time to reach more than 80% of link utilization with just 100 connections. It also shows that the increment of utilization becomes smaller as the channel number increases. Therefore, the performance result matches our analysis in previous section. And it confirms again that five channels per-connection is good enough.

Secondly, we compare the performance of DMCTCP ($m = 5$) with other TCP variants without background traffic. As shown in Figure 5.12(a), link utilization is much higher in DMCTCP than any other TCP variants. Compared with TCP-SACK on different amount of connections, the utilization is 60% to 200% higher. Compared with the most popular high-speed variant TCP-CUBIC, the utilization is 30% to 80% higher. We also observed that the two MPTCP congestion control algorithms do not have much performance improvement. MPTCP-Coupled has nearly 5% higher and MPTCP-OLIA has nearly 5% lower performance than TCP-SACK, although we set five subflows for each connection. This verifies TCP-SACK, MPTCP-Coupled and MPTCP-OLIA are still conservative under high-speed and long delay networks with tiny buffers.

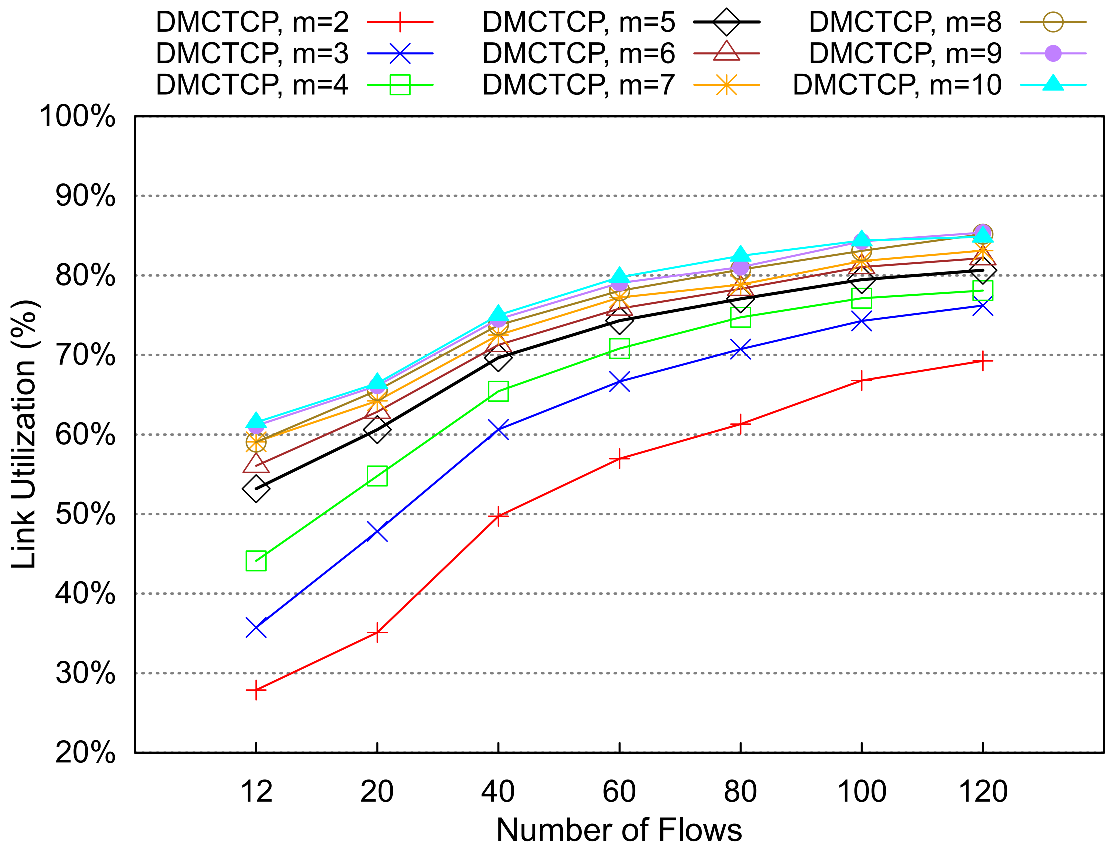
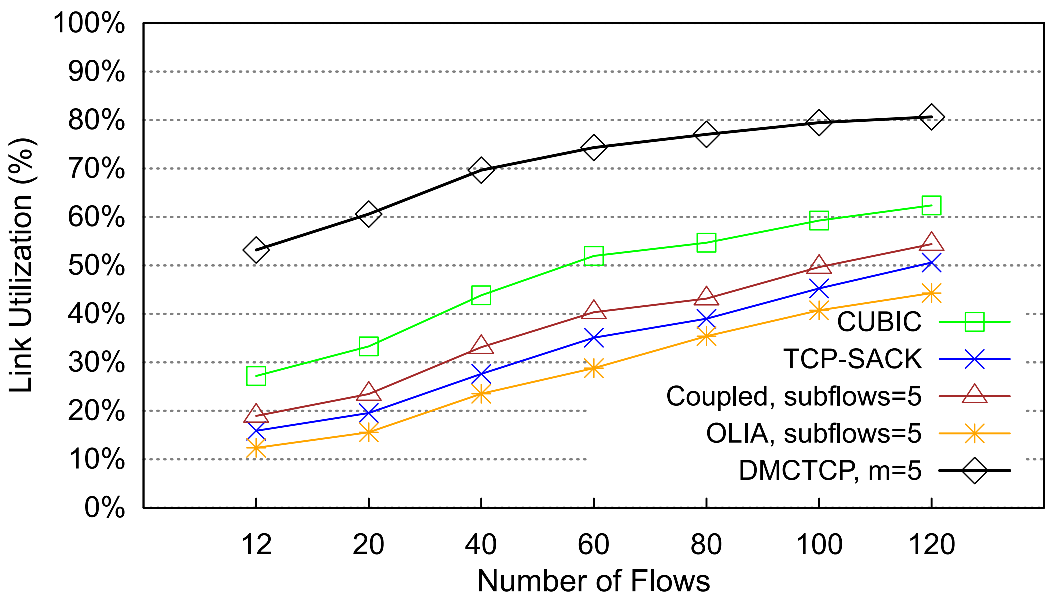


FIGURE 5.11: Link utilization without background traffic, compared with number of channels

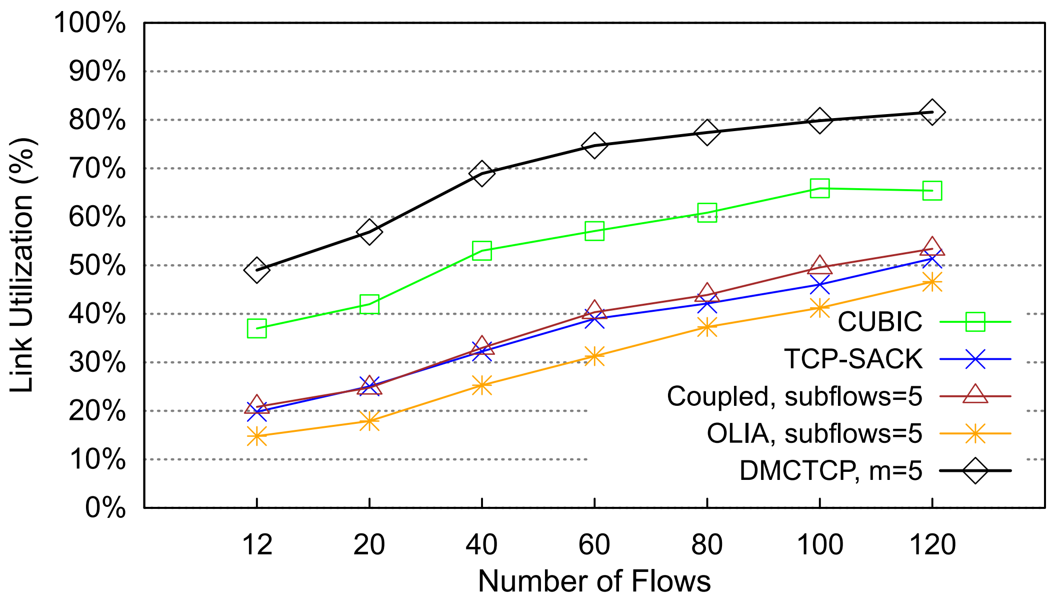
Thirdly, we evaluate the link utilization under bursty background traffic. As shown in Figure 5.12(b), adding the 4.6% background traffic, the link utilizations are 3% to 8% higher on all TCP variants than these without background traffic. This is expected because the background traffic introduces some dynamics which can help improve asynchronism in packet losses. However, we can see the dynamics of background traffic caused very small improvement. Meanwhile, we found that MPTCP-Coupled has almost the same performance as TCP-SACK, which shows MPTCP-Coupled is closely compatible to TCP-SACK under background traffic. Again, we observed DMCTCP ($m = 5$) has much better performance than the other TCP variants.

5.7.2 Intra-protocol Fairness

We measure the intra-protocol fairness by performing experiments with two connections of a TCP variant with the same RTT. These two connections's throughput is



(a) Without background traffic

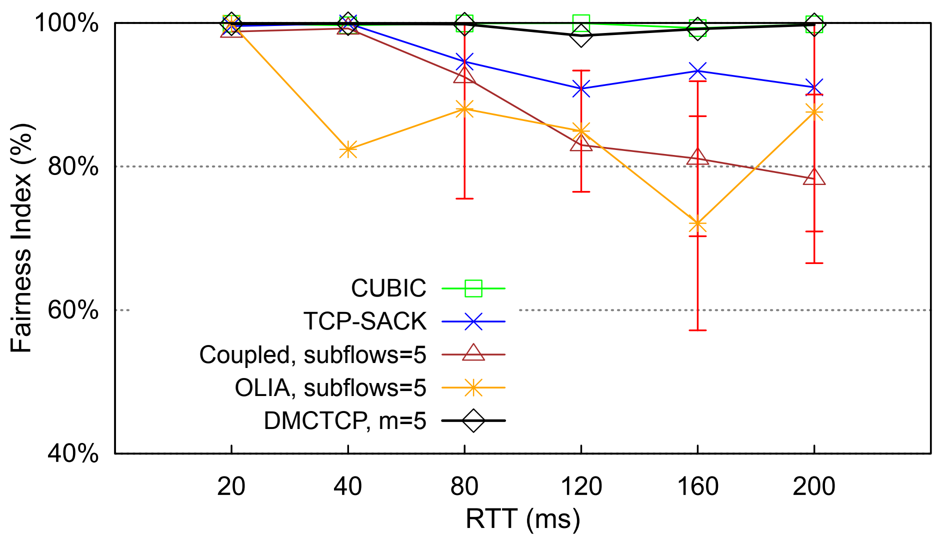


(b) With background traffic

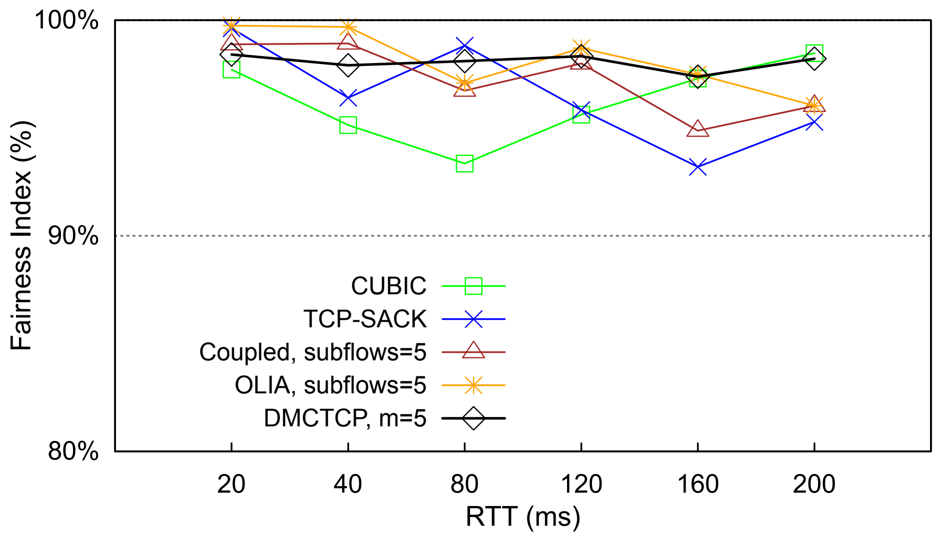
FIGURE 5.12: Link utilization compared with other TCP variants

used as input to compute Jain's fairness index [24]. These tests are conducted when RTTs are varied from 20 ms to 200 ms. Figure 5.13(a) and 5.13(b) show intra-protocol fairness of TCP variants without and with background traffic respectively.

Without background traffic, DMCTCP, TCP-CUBIC and TCP-SACK show very good fairness. All of these variants achieve more than 90% fairness index in all tests.



(a) Without background traffic



(b) With background traffic

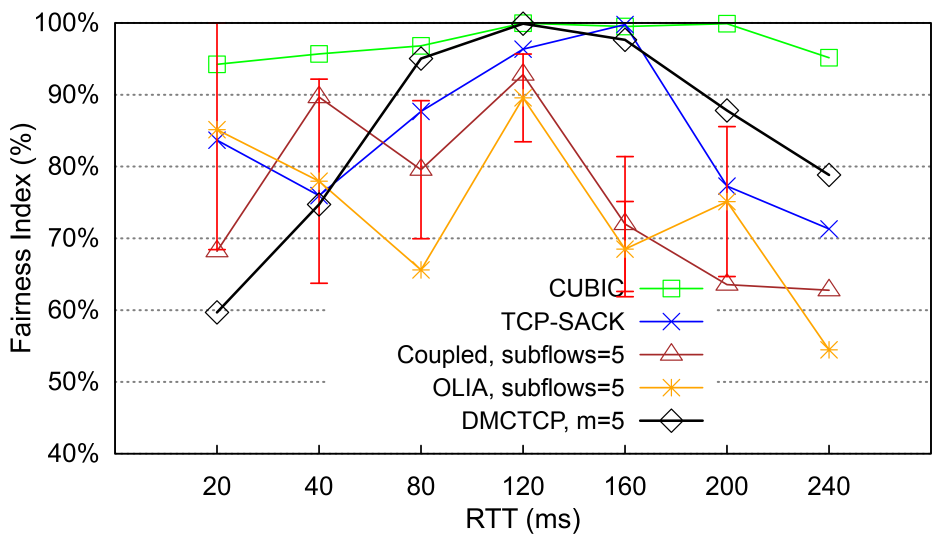
FIGURE 5.13: Intra-Protocol Fairness

In comparison, MPTCP-Coupled and MPTCP-OLIA show lower fairness when RTT increases. Meanwhile, we find that the throughput of these two MPTCP-Coupled and MPTCP-OLIA connections become more unstable as RTT increases. Even repeated for 9 times in each test of these two variants, the standard deviations are still very large (more than 5%) beyond 80 ms. We have to display the related standard deviations of these two in Figure 5.13(a). However, as we add background traffic, we find all the TCP variants achieve greatly improved fairness (more than 90%) and much smaller standard deviations (less than 4%). Again, we find DMCTCP shows very high fairness and very small fairness variance as RTT increases.

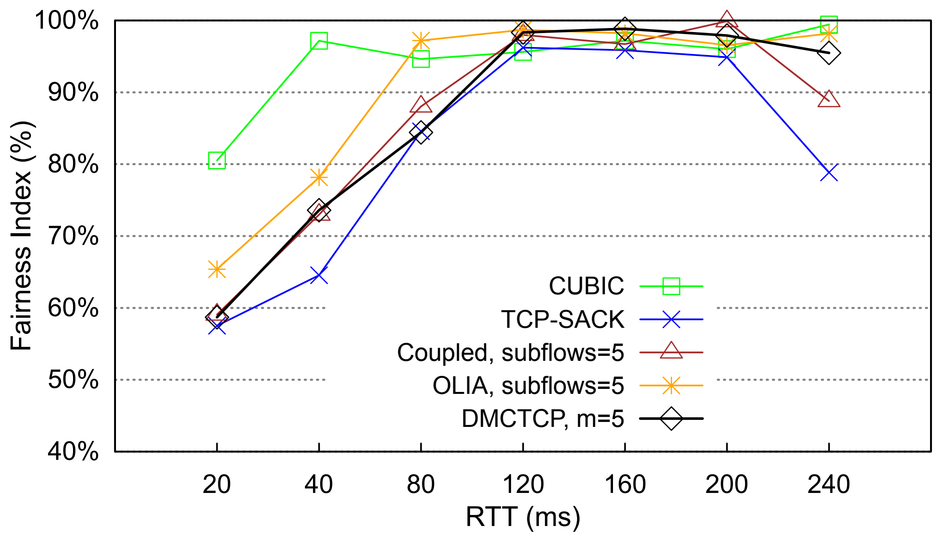
5.7.3 RTT Fairness

We measure the fairness in sharing the bottleneck bandwidth between two competing connections that have different RTTs. One connection's RTT is fixed to 120 ms, and the other's varies from 20 ms to 240 ms. Again, these two connections's throughput is used to compute Jain's fairness index. As mentioned in [107], RTT fairness can be divided into two categories. One category is to reach the equal sharing of the bottleneck bandwidth even when the two competing connections have different RTTs. The other category is to achieve bandwidth shares inversely proportional to the RTT ratios. Although there are no commonly accepted principles to decide which category is better than the other, the later category may be more desirable because long RTT connections used to take more network resources than short RTT connections as they travel through more intermediary devices over a longer path. Any TCP variant belonging to a category should be stable without crossing over the boundaries.

Figure 5.14(a) and 5.14(b) shows the RTT fairness of TCP variants without and with background traffic. TCP-CUBIC has the best fairness index between the two connections regardless of their RTTs. DMCTCP and TCP-SACK have very close fairness index to each other and both of them fall in the second inverse bandwidth share category because these two increase their fairness index when the second connection's RTT is close to 120 ms and decrease their fairness index when the second connection's RTT is far away from 120 ms. This is expected because DMCTCP uses the same ACK-clock mechanism as TCP-SACK does. However, without background traffic, the RTT fairness of both MPTCP-Coupled and MPTCP-OLIA fluctuates because these two have unstable performance that is mentioned in the tests of intra-protocol fairness. With background traffic, MPTCP-Coupled behaves similarly to DMCTCP



(a) Without background traffic



(b) With background traffic

FIGURE 5.14: RTT Fairness

and TCP-SACK, but MPTCP-OLIA does not decrease the fairness index after its second connection has higher RTT than 120 ms.

5.8 Conclusion

In this chapter, we have shown that most current TCP variants including MPTCP have performance problems in high-speed networks with tiny buffers. Based on the problem analysis, we have demonstrated a working DMCTCP congestion control algorithm which is designed for such networks. Our experimental results confirm that

DMCTCP can achieve much higher performance and have very good properties in terms of convergence, intra-protocol fairness and RTT fairness.

DMCTCP brings immediate practical benefits as it matches the condition requirements to deploy all-optical routers in access networks. Because queuing delay is gone, it also significantly reduces the memory required for maintaining the congestion windows in end-hosts and is beneficial to latency sensitive applications.

Chapter 6
Conclusion

In this dissertation, we investigated TCP congestion control mechanism, design issues, challenges, and the most widely adopted TCP congestion control variants for high-speed networks. In order to study the performance of TCP over 10Gbps high-speed networks, we developed CRON, a 10Gbps network emulation testbed for various network research. Using the testbed, we can study the performance issues in TCP and in most TCP congestion control variants. Especially, we explored the performance issues of TCP under various interaction with network components such as routers' queuing management strategies and buffer sizing categories. We analyzed two important performance issues of TCP in extremely small buffered high-speed networks. One is TCP loss synchronization and the other one is flow burstiness. Based on the analysis, we defined two types of congestion as bandwidth congestion and burstiness congestion. Both of these types can cause bottleneck link bandwidth to be underutilized. We demonstrated that most TCP variants suffer from both congestion types because TCP's false-congestion-detection problem exists in high-speed networks with tiny buffers.

6.1 Contributions

The development of CRON system bridges the gap between physical networks, simulation, and high-speed networking environment by providing integrated and automated access to a wide range of high speed networking configurations, such as NLR (National Lambda Rail), Internet2, LONI (Louisiana Optical Network Initiative), etc., and purely user-defined networks. CRON enables researchers to explore new network technologies and rapidly assess their impact on applications irrespective of resource limitation. It also allows educators to introduce a state-of-art networking environment to students who cannot access high-speed production networks.

Using CRON's 10Gbps capability, the performance of TCP and its variants were carefully studied. We proposed two new synchronization expressions by carefully examining the congestion events. And we obtained the measurements based on categories of small sized and tiny sized router buffers. We illustrated the different relationships among TCP loss synchronization, flow burstiness and link utilization. And discovered new problems and challenges of TCP in high-speed networks with tiny buffers.

In regard to the new challenges of TCP, we designed a new TCP congestion control algorithm called Desynchronized Multi-Channel TCP (DMCTCP), which pursues goals of minimizing TCP loss synchronization and reducing the impact of flow burstiness. We implemented this algorithm based on the framework of MPTCP and showed outstanding performance and good properties of DMCTCP compared with other TCP variants. The important features of DMCTCP are that various congestion events are detected and distinguished with different corresponding actions, and that no modifications to other network layers (such as the application layer) of the end-to-end system need to be made.

Therefore, we could achieve an acceptable link utilization in high-speed access networks with very small router buffer. (a) It could facilitate the deploy of all-optical routers because they have huge link capacity and lower power requirement but are limited by buffer size. (b) It could reduce the requirement of router complexity, making routers easier to build and easier to scale. (c) It could minimize queuing delay and jitter that were closely related to buffer size in electronic routers. (d) Its socket API is compatible with common applications without change or modification to existing applications. (e) It is transparent to layers below it that it is likely to work correctly in the Internet through different middleboxes. (f) It significantly reduces the memory required for maintaining the congestion windows in end-hosts because queuing delay is gone. (g) It is beneficial to latency sensitive applications.

6.2 Future Research

Multiple directions could be explored as future work. A first one would be to extend DMCTCP over mutipath links for both ultra-low latency and high performance in data center networks. Another direction would be to explore the performance of DM-CTCP over wireless high-speed networks based on investigation of congestion types in such networks.

References

[1] M. Allman, V. Paxson, and W. Stevens, “TCP Congestion Control,” RFC 2581 (Proposed Standard), Internet Engineering Task Force, Apr. 1999, obsoleted by RFC 5681, updated by RFC 3390. [Online]. Available: http://www.ietf.org/rfc/rfc2581.txt

[2] A. Vishwanath, V. Sivaraman, and M. Thottan, “Perspectives on router buffer sizing: recent results and open problems,” *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 2, pp. 34–39, 2009.

[3] L. Xue, S. Kumar, C. Cui, and S.-J. 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)*, Clearwater, Florida, USA, 2012, pp. 348–351.

[4] G. Appenzeller, I. Keslassy, and N. McKeown, “Sizing router buffers,” in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4. ACM, 2004, pp. 281–292.

[5] D. Wischik and N. McKeown, “Part i: Buffer sizes for core routers,” *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 3, pp. 75–78, 2005.

[6] G. Raina, D. Towsley, and D. Wischik, “Part ii: Control theory for buffer sizing,” *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 3, p. 82, 2005.

[7] N. Beheshti, Y. Ganjali, R. Rajaduray, D. Blumenthal, and N. McKeown, “Buffer sizing in all-optical packet switches,” in *Optical Fiber Communication Conference*. Optical Society of America, 2006.

[8] R. Tucker, “The role of optics and electronics in high-capacity routers,” *Journal of Lightwave Technology*, vol. 24, no. 12, pp. 4655–4673, 2006.

[9] H. Park, E. Burmeister, S. Bjorlin, and J. Bowers, “40-gb/s optical buffer design and simulations,” *Numerical Simulation of Optoelectronic Devices (NUSOD)*, 2004.

[10] V. Lal, J. Summers, M. Masanovic, L. Coldren, and D. Blumenthal, “Novel compact inp-based monolithic widely tunable differential mach-zehnder interferometer wavelength converter for 40 gbps operation,” in *Indium Phosphide and Related Materials, 2005. International Conference on*. IEEE, 2005, pp. 25–27.

[11] M. Masanovic, V. Lal, J. Summers, J. Barton, E. Skogen, L. Rau, L. Coldren, and D. Blumenthal, “Widely tunable monolithically integrated all-optical wavelength converters in inp,” *Lightwave Technology, Journal of*, vol. 23, no. 3, pp. 1350–1362, 2005.

[12] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, "Routers with very small buffers," in *Proc. IEEE Infocom*, vol. 6. Citeseer, 2006.

[13] 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.* IEEE, 2009, pp. 569–576.

[14] G. Patil, S. McClean, and G. Raina, "Drop tail and red queue management with small buffers: stability and hopf bifurcation," *ICTACT Journal on Communication Technology*, vol. 2, no. 2, pp. 339–344, 2011.

[15] Y. Gu and R. Grossman, "Udt: Udp-based data transfer for high-speed wide area networks," *Computer Networks*, vol. 51, no. 7, pp. 1777–1799, 2007.

[16] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of tcp pacing," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2000, pp. 1157–1165.

[17] V. Sivaraman, H. Elgindy, D. Moreland, and D. Ostry, "Packet pacing in small buffer optical packet switched networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 4, pp. 1066–1079, 2009.

[18] Y. Cai, B. Jiang, T. Wolf, and W. Gong, "A practical on-line pacing scheme at edges of small buffer networks," in *INFOCOM, 2010 Proceedings IEEE.* IEEE, 2010, pp. 1–9.

[19] J. Nagle, "Congestion Control in IP/TCP Internetworks," RFC 896, Internet Engineering Task Force, Jan. 1984. [Online]. Available: http://www.ietf.org/rfc/rfc896.txt

[20] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4. ACM, 1988, pp. 314–329.

[21] R. Jain and K. Ramakrishnan, "Congestion avoidance in computer networks with a connectionless network layer: Concepts, goals and methodology," in *Computer Networking Symposium, 1988., Proceedings of the.* IEEE, 1988, pp. 134–143.

[22] D. Papadimitriou, M. Welzl, M. Scharf, and B. Briscoe, "Open Research Issues in Internet Congestion Control," RFC 6077 (Informational), Internet Engineering Task Force, Feb. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6077.txt

[23] S. Floyd, "Congestion Control Principles," RFC 2914 (Best Current Practice), Internet Engineering Task Force, Sep. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2914.txt

[24] R. Jain, D. Chiu, and W. Hawe, *A quantitative measure of fairness and discrimination for resource allocation in shared computer system.* Eastern Research Laboratory, Digital Equipment Corp., 1984.

[25] M. Welzl and W. Eddy, "Congestion Control in the RFC Series," RFC 5783 (Informational), Internet Engineering Task Force, Feb. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5783.txt

[26] M. Welzl, *Network Congestion Control: Managing Internet Traffic (Wiley Series on Communications Networking & Distributed Systems).* John Wiley & Sons, 2005.

[27] B. Briscoe, "Flow rate fairness: Dismantling a religion," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 63–74, 2007.

[28] S. Floyd and M. Allman, "Comments on the Usefulness of Simple Best-Effort Traffic," RFC 5290 (Informational), Internet Engineering Task Force, Jul. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5290.txt

[29] J. Roberts, "Internet traffic, qos, and pricing," *Proceedings of the IEEE*, vol. 92, no. 9, pp. 1389–1399, 2004.

[30] D. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN systems*, vol. 17, no. 1, pp. 1–14, 1989.

[31] C. Villamizar and C. Song, "High performance tcp in ansnet," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 5, pp. 45–60, 1994.

[32] D. Davies, "The control of congestion in packet-switching networks," *Communications, IEEE Transactions on*, vol. 20, no. 3, pp. 546–550, 1972.

[33] J. Postel, "Transmission Control Protocol," RFC 793 (Standard), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168, 6093, 6528. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[34] R. Jain, "A timeout-based congestion control scheme for window flow-controlled networks," *Selected Areas in Communications, IEEE Journal on*, vol. 4, no. 7, pp. 1162–1167, 1986.

[35] M. Handley, J. Padhye, and S. Floyd, "TCP Congestion Window Validation," RFC 2861 (Experimental), Internet Engineering Task Force, Jun. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2861.txt

[36] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis, "Increasing TCP's Initial Window," RFC 6928 (Experimental), Internet Engineering Task Force, Apr. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6928.txt

[37] S. Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649 (Experimental), Internet Engineering Task Force, Dec. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3649.txt

[38] Y. Li, D. Leith, and R. Shorten, "Experimental evaluation of tcp protocols for high-speed networks," *Networking, IEEE/ACM Transactions on*, vol. 15, no. 5, pp. 1109–1122, 2007.

[39] K. Leung and V. Li, "Transmission control protocol (tcp) in wireless networks: issues, approaches, and challenges," *Communications Surveys & Tutorials, IEEE*, vol. 8, no. 4, pp. 64–79, 2006.

[40] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[41] T. Kelly, "Scalable tcp: Improving performance in highspeed wide area networks," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 2, pp. 83–91, 2003.

[42] D. Leith and R. Shorten, "H-tcp: Tcp for high-speed and long-distance networks," in *Proceedings of PFLDnet*, vol. 2004, 2004.

[43] L. Grieco and S. Mascolo, "Performance evaluation and comparison of westwood+, new reno, and vegas tcp congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 25–38, 2004.

[44] L. Brakmo and L. Peterson, "Tcp vegas: End to end congestion avoidance on a global internet," *Selected Areas in Communications, IEEE Journal on*, vol. 13, no. 8, pp. 1465–1480, 1995.

[45] C. Jin, D. Wei, and S. Low, "Fast tcp: motivation, architecture, algorithms, performance," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4. IEEE, 2004, pp. 2490–2501.

[46] C. Caini and R. Firrincieli, "Tcp hybla: a tcp enhancement for heterogeneous networks," *International Journal of Satellite Communications and Networking*, vol. 22, no. 5, pp. 547–566, 2004.

[47] K. Tan and J. Song, "A compound tcp approach for high-speed and long distance networks," in *In Proc. IEEE INFOCOM.* Citeseer, 2006.

[48] S. Liu, T. Başar, and R. Srikant, "Tcp-illinois: A loss-and delay-based congestion control algorithm for high-speed networks," *Performance Evaluation*, vol. 65, no. 6, pp. 417–440, 2008.

[49] J. Babiarz, K. Chan, and F. Baker, "Configuration Guidelines for DiffServ Service Classes," RFC 4594 (Informational), Internet Engineering Task Force, Aug. 2006, updated by RFC 5865. [Online]. Available: http://www.ietf.org/rfc/rfc4594.txt

[50] A. Venkataramani, R. Kokku, and M. Dahlin, "Tcp nice: A mechanism for background transfers," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 329–343, 2002.

[51] A. Kuzmanovic and E. Knightly, "Tcp-lp: low-priority service via end-point congestion control," *Networking, IEEE/ACM Transactions on*, vol. 14, no. 4, pp. 739–752, 2006.

[52] C. Man, G. Hasegawa, and M. Murata, "Imtcp: Tcp with an inline measurement mechanism for available bandwidth," *Computer communications*, vol. 29, no. 10, pp. 1614–1626, 2006.

[53] P. Yang, W. Luo, L. Xu, J. Deogun, and Y. Lu, "Tcp congestion avoidance algorithm identification," in *Proceedings of the 2011 31st International Conference on Distributed Computing Systems.* IEEE Computer Society, 2011, pp. 310–321.

[54] D. Leith, R. Shorten, and G. McCullagh, "Experimental evaluation of cubic-tcp," in *Proceedings of PFLDnet*, 2008.

[55] A. Medina, M. Allman, and S. Floyd, "Measuring the evolution of transport protocols in the internet," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 2, pp. 37–52, 2005.

[56] P. Sarolahti and A. Kuznetsov, "Congestion control in linux tcp," in *Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference.* USENIX Association, 2002, pp. 49–62.

[57] D. Thaler and M. Sridhara, "D. bansal," implementation report on experiences with various tcp rfcs", presentation to the ietf transport area," 2007.

[58] K. Wehrle, F. Pählke, H. Ritter, D. Müller, and M. Bechler, "The linux networking architecture," *Design and Implementation of Network Protocols in the Linux Kernel*, 2005.

[59] D. Wei and P. Cao, "Ns-2 tcp-linux: an ns-2 tcp implementation with congestion control algorithms from linux," in *Proceeding from the 2006 workshop on ns-2: the IP network simulator.* ACM, 2006, p. 9.

[60] NLR, "National Lambda Rail: The Network for Advanced Research and Innovation," July 2011, http://www.nlr.net/.

[61] Internet2, "The new Internet2 network," in *6th GLIF meeting*, 2006.

[62] LONI, "The Louisiana Optical Network Initiative," 2011, "http://www.loni.org/".

[63] "Cyberinfrastructure of reconfigurable optical networking environment," 2011, "website:http://www.cron.loni.org/".

[64] A. Bialecki, M. Cafarella, D. Cutting, and O. OMALLEY, "Hadoop: a framework for running applications on large clusters built of commodity hardware," *Wiki at http://lucene. apache. org/hadoop*, vol. 11, 2005.

[65] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of the Fifth Symposium on Operating Systems Design and Implementation.* Boston, MA: USENIX Association, Dec. 2002, pp. 255–270.

[66] M. Carbone and L. Rizzo, "Dummynet revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, 2010.

[67] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[68] "PRObE: Parallel Reconfigurable Observation Environment," 2012, "website: http://newmexicoconsortium.org/probe".

[69] G. von Laszewski, G. Fox, F. Wang, A. Younge, A. Kulshrestha, G. Pike, W. Smith, J. Vockler, R. Figueiredo, J. Fortes *et al.*, "Design of the futuregrid experiment management framework," in *Gateway Computing Environments Workshop (GCE), 2010.* IEEE, 2010, pp. 1–10.

[70] A. Avetisyan, R. Campbell, I. Gupta, M. Heath, S. Ko, G. Ganger, M. Kozuch, D. O'Hallaron, M. Kunze, T. Kwan *et al.*, "Open cirrus: A global cloud computing testbed," *Computer*, vol. 43, no. 4, pp. 35–43, 2010.

[71] I. Baldine, J. Chase, G. Rouskas, and R. Dutta, "At-scale experimentation with resource virtualization in a metro optical testbed," *Proceedings of ICVCI*, 2008.

[72] "Anue Systems Inc." 2012, "http://www.anuesystems.com/".

[73] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on.* IEEE, 2009, pp. 124–131.

[74] M. Bencivenni, D. Bortolotti, A. Carbone, A. Cavalli, A. Chierici, S. Dal Pra, D. De Girolamo, M. Donatelli, A. Fella, D. Galli *et al.*, "Performance of 10 Gigabit Ethernet Using Commodity Hardware," *Nuclear Science, IEEE Transactions on*, vol. 57, no. 2, pp. 630–641, 2010.

[75] L. Xue, C. Cui, S. Kumar, and S. 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.* IEEE, 2012, pp. 315–319.

[76] O. Feuser and A. Wenzel, "On the effects of the ieee 802.3 x flow control in full-duplex ethernet lans," in *Local Computer Networks, 1999. LCN'99. Conference on.* IEEE, 1999, pp. 160–161.

[77] L. Grossman, "Large receive offload implementation in neterion 10gbe ethernet driver," in *Linux Symposium*, 2005, p. 195.

[78] "NETCONF," http://en.wikipedia.org/wiki/NETCONF, 2011.

[79] B. Milekic, "Network buffer allocation in the freebsd operating system," *Proceedings of BSDCan*, 2004.

[80] "Integrating a CRON (Cyberinfrastructure of Reconfigurable Optical Network) Testbed into GENI," 2009, "website:http://groups.geni.net/geni/wiki/CRON/ ".

[81] "Protogeni framework inside geni project," http://www.protogeni.net/trac/protogeni/, 2010.

[82] "Resource Specification for GENI," 2011, "http://www.protogeni.net/trac/protogeni/wiki/RSpec".

[83] S. Floyd. (2008) Tools for the evaluation of simulation and testbed scenarios. [Online]. Available: http://tools.ietf.org/html/draft-irtf-tmrg-tools-05

[84] Q. Fu and P. Jay, "A step towards understanding loss synchronisation between concurrent tcp flows," in *INFOCOM Workshops 2008, IEEE.* IEEE, 2008, pp. 1–6.

[85] S. Hassayoun and D. Ros, "Loss synchronization and router buffer sizing with high-speed versions of tcp," in *INFOCOM Workshops 2008, IEEE.* IEEE, 2008, pp. 1–6.

[86] K. Fall and S. Floyd, "Comparisons of tahoe, reno, and sack tcp," *Computer Communications Review*, vol. 26, no. 3, pp. 5–21, 1996.

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

[88] P. Yang, W. Luo, L. Xu, J. Deogun, and Y. Lu, "Tcp congestion avoidance algorithm identification," *IEEE International conference on distributed computing systems*, 2011.

[89] F. Schneider, J. Wallerich, and A. Feldmann, "Packet capture in 10-gigabit ethernet environments using contemporary commodity hardware," *Passive and Active Network Measurement*, pp. 207–217, 2007.

[90] F. Fuentes and D. Kar, "Ethereal vs. tcpdump: a comparative study on packet sniffing tools for educational purpose," *Journal of Computing Sciences in Colleges*, vol. 20, no. 4, pp. 169–176, 2005.

[91] G. Vattay, A. Fekete, J. Stéger, and M. Maródi, "Modeling competition, fairness and chaos in computer networks," *Network*, vol. 1, no. 243, p. 276, 2002.

[92] G. Raina and D. Wischik, "Buffer sizes for large multiplexers: Tcp queueing theory and instability analysis," in *Next Generation Internet Networks, 2005.* IEEE, 2005, pp. 173–180.

[93] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, "Part iii: Routers with very small buffers," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 3, pp. 83–90, 2005.

[94] N. Beheshti, Y. Ganjali, M. Ghobadi, N. McKeown, and G. Salmon, "Experimental study of router buffer sizing," in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement.* ACM, 2008, pp. 197–210.

[95] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines for multipath tcp development," *RFC6182 (March 2011), www. ietf. ort/rfc/6182*, 2011.

[96] B. Carpenter and S. Brim, "Middleboxes: Taxonomy and Issues," RFC 3234 (Informational), Internet Engineering Task Force, Feb. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3234.txt

[97] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018 (Proposed Standard), Internet Engineering Task Force, Oct. 1996. [Online]. Available: http://www.ietf.org/rfc/rfc2018.txt

[98] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath tcp," in *Proceedings of the 8th USENIX conference on Networked systems design and implementation.* USENIX Association, 2011, pp. 8–8.

[99] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec, "Mptcp is not pareto-optimal: performance issues and a possible solution," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies.* ACM, 2012, pp. 1–12.

[100] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend tcp?" in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference.* ACM, 2011, pp. 181–194.

[101] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 266–277.

[102] J. Mahdavi, "Tcp-friendly unicast rate-based flow control," Jan 1997, unpublished note. [Online]. Available: http://www.psc.edu/index.php/component/remository/Networking/Networking-Papers/TCP-Friendly-Unicast-Rate-Based-Flow-Control/

[103] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proceedings of USENIX NSDI conference*, 2012.

[104] B. Shihada and P.-H. Ho, "Transport control protocol in optical burst switched networks: Issues, solutions, and challenges," *Communications Surveys & Tutorials, IEEE*, vol. 10, no. 2, pp. 70–86, 2008.

[105] http://mptcp.info.ucl.ac.be/.

[106] http://www.linuxfoundation.org/collaborate/workgroups/networking/tcpprobe/.

[107] S. Ha, Y. Kim, L. Le, I. Rhee, and L. Xu, "A step toward realistic performance evaluation of high-speed tcp variants," in *Fourth International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet06)*, 2006.

Vita

Cheng Cui received his bachelor's degree in School of Software Engineering from Xidian University, Xi'an, China in 2006. After graduation, he started his career in IBM China as an IT specialist from 2006 to 2007. He started doctoral program in Department of Computer Science at the Louisiana State University in 2008, and joined the Center for Computation and Technology as a research assistant in the same year.

Cheng Cui is currently a PhD candidate and will be awarded in December, 2013 from the Computer Science and Engineering Division in School of Electrical Engineering and Computer Science, at The Louisiana State University. He is passionate about building high performance distributed systems, with a particular interest in high-speed networking systems and cloud computing systems. He also enjoys working on Linux kernel development and computer system performance tuning.