High-Speed Transport-Layer Protocols for Fast Long-Distance Networks

Zongsheng Zhang

January 2006

Department of Information Networking
Graduate School of Information Science and Technology
Osaka University

This page is blank.

Preface

Currently, Transmission Control Protocol (TCP) is the most widely used transport-layer protocol in the Internet. TCP is the primary transport protocol in use in most IP networks, and supports the major portion of traffic across the Internet. It is typically employed by applications that require guaranteed delivery.

However, data intensive applications, e.g., Content Distribution Network (CDN) and Storage Area Network (SAN), have appeared. These applications use high speed networks to transfer terabyte/pertabyte-sized files over continents. Recent research has shown that current TCP mechanisms can obstruct efficient use of such fast long-distance networks (LFNs). Addressing the problem of TCP used in LFNs, several high-speed protocols are proposed in recent years. These protocols can be classified into two categories. Both of them modify the algorithm of TCP. The first requires modifications at both end-hosts and the routers in between, e.g., eXplicit Control Protocol (XCP), and Variable-structure congestion Control Protocol (VCP). For using them, the mechanism of routers must be re-constructed, for some information gathered by routers need to be fed back to the end-hosts. The second category only needs the modification of the congestion control mechanism of end-host's TCP, e.g., HighSpeed TCP (HSTCP), Scalable TCP, and FAST TCP. Thus, they are relatively easy to be deployed in the current Internet.

To date, these high-speed protocols are still on the way of development and not widely deployed. Moreover, none of them have given a completely solution, for example, HSTCP, which is a representative of high-speed protocols, may provide higher throughput than TCP Reno, but HSTCP flows starve TCP Reno flows when they share the same network links. A more suitable transport protocol, which can provide higher throughput with better fairness against the competing TCP flows, should be designed at present. Thus we solve the

problem of fairness by proposing an enhanced transport layer protocol – Gentle HighSpeed TCP (gHSTCP). gHSTCP is based on HSTCP, so it can provide high performance and is easy to be deployed. For fairness, gHSTCP uses two modes – HSTCP mode and Reno mode – in the congestion avoidance phase, and switches between modes based on the trend of changing RTT. Simulation results show gHSTCP can significantly improve performance in mixed environments. When gHSTCP and TCP Reno flows share the same bottleneck link, compared with the case when HSTCP is used, gHSTCP may provide better utilization and fairness. When TCP SACK option is turned on, gHSTCP can also provide better performance, though HSTCP may achieve almost the same bottleneck utilization as that done with gHSTCP in some case. For instance, when DropTail is deployed, the bottleneck link bandwidth is 2.5 Gbps, and the propagation delay is 50 ms, gHSTCP rises by 15% on utilization, and by 0.1 on fairness (Jain's fairness index) compared with HSTCP. gHSTCP using SACK option rises by 0.16 on fairness compared with HSTCP using SACK option. Then, the performance of gHSTCP is evaluated when gHSTCP flows co-exist with Web traffic. With the help of gHSTCP, the packet drop rate is depressed, and Web responding time is slightly improved.

However, the performance improvement is limited due to the nature of TailDrop router, and the RED routers can not alleviate the problem completely. Therefore, we present a modified version of adaptive RED (ARED), called gentle adaptive RED (gARED), directed at the problem of simultaneous packet drops by multiple flows in high speed networks. gARED can eliminate weaknesses found in ARED by monitoring the trend in variation of the average queue length of router buffer. Our approach, combining gHSTCP and gARED, is quite effective and fair to competing traffic. With the assistance of gARED mechanism, both of utilization and fairness are boosted.

In the above works, the gHSTCP effectiveness has only been demonstrated by simulations. For its applications, it is necessary to evaluate gHSTCP by experiments. Thus, we construct an experimental environment to check the performance of gHSTCP. Based on the experimental results, a refined gHSTCP algorithm is proposed for application to real networks. The refined gHSTCP algorithm is based on the original algorithm, compares two RTT thresholds and determines which mode is used. Then, the performance of the refined gHSTCP algorithm is assessed experimentally, and compared with TCP Reno/HSTCP and

parallel TCP mechanisms. The experimental results reveal that gHSTCP can provide a better trade-off in terms of utilization and fairness against co-existing traditional TCP Reno connections, whereas HSTCP and parallel TCP suffer from the trade-off problem.

Addressing the performance issue of TCP Reno in LFNs, parallel TCP mechanism has been proposed and employed by some applications. In this thesis, we attend to investigate the problem of TCP Reno in LFNs and attempt to give some suggestions. Therefore, at the end of this thesis, we use mathematical analysis to explore the performance of parallel TCP. Parallel TCP uses many concurrent TCP connections for one task. So far, using parallel TCP is something of black art. We try to answer this question: Is parallel really effective for LFNs? The analytical results reveal that it is difficult to use parallel TCP in practice for the sake of approving throughput. That is, parallel TCP is not really effective in LFNs, because the optimal number of TCP connections cannot be easily obtained. Especially, parallel TCP exactly possesses the properties that induce synchronization. While the router has small buffer size, the performance of parallel TCP in synchronization case deteriorates significantly as the number of TCP connections is increased. In contrast, high-speed protocols have the inherent characteristics which are suitable for LFNs. Based on these results, we recommend using high-speed protocols instead of parallel TCP in LFNs in practice.

List of publications

Journal:

1. Zongsheng Zhang, Go Hasegawa, and Masayuki Murata, "Performance analysis and improvement of HighSpeed TCP with TailDrop/RED routers," *IEICE Transactions on Communications*, Vol. E88-B, No. 6, pp. 2495-2507, June 2005.

2. Zongsheng Zhang, Go Hasegawa, and Masayuki Murata, "Experimental results of implementing high-speed and parallel TCP variants for long-fat networks," *IEICE Transactions on Communications*, Vol. E89-B, No. 3, March 2006.

3. Zongsheng Zhang, Go Hasegawa, and Masayuki Murata, "Analysis evaluation of parallel TCP: Is it really effective for long-fat networks?" submitted to *IEICE Transactions on Communications*, October 2005.

International Conference:

1. Zongsheng Zhang, Go Hasegawa, and Masayuki Murata, "Performance analysis and improvement of HighSpeed TCP with TailDrop/RED routers," in *Proc. the Twelfth IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2004)*, pp. 505–512, October 2004.

2. Zongsheng Zhang, Go Hasegawa, and Masayuki Murata, "Experimental evaluation of Gentle HighSpeed TCP for long-fat networks," in *Proc. the 6th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT 2005)*, pp. 47–52, November 2005.

Domestic Workshop:

1. Zongsheng Zhang, Go Hasegawa, and Masayuki Murata, "Performance Analysis and Improvement of HighSpeed TCP with TailDrop/ARED Routers," *IEICE Technical Report IN2004-1*, Vol. 104, No. 73, pp. 49–54, May 2004.

2. Zongsheng Zhang, Go Hasegawa, and Masayuki Murata, "Implementation Experiments on HighSpeed and Parallel TCP," *IEICE Technical Report IN2005-17*, Vol. 105, No. 113, pp. 17–22, June 2005

3. Zongsheng Zhang, Go Hasegawa, and Masayuki Murata, "Reasons not to Parallelize TCP Connections for Fast Long-Distance Networks," to appear in *IEICE Technical Report*, January 2006.

Acknowledgements

Of course, like any author, I am indebted to those people that do their best to help me on my thesis. Thanks to all of the wonderful people.

First and foremost, I would like to express my gratitude to Prof. Masayuki Murata for his dedication as my advisor, as well as his support and guidance. He was always insightful to my work. This thesis would not be possible without his advice and suggestions.

I am also thankful to my thesis committee members, Prof. Koso Murakami and Prof. Hirotaka Nakano, for their careful reading and constructive comments in completing this thesis.

Associate Prof. Go Hasegawa, as my direct tutor, took me under his wing. He has always remained a constant source of advice, and was very patient to my research. I have profited immensely from his wisdom and high standards.

The members of our team made it possible for me to complete my Ph.D. I am very grateful for their help.

Many people have contributed their ideas and energies. I hesitate to list them because I do not wish to leave anyone out. So let me first apologize to all the wonderful people I have missed. Thank you – all of you!

Finally, my family, as my backup force, is always a tireless advocate. My wife has actively accompanied me in Japan, and my cute son had to live in China with his grandmother. I wish to acknowledge them for their role in my life. If left to me alone, my thesis would never have been done.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we first give the depiction of background. Then the related work is introduced. At last, the contribution and organization of the thesis is presented.

1.1 Research background

Computers, networks and their applications are now a part of daily life. The user machines and server machines are connected by a network, as illustrated in Figure 1.1. The network applications are widespread from entertainment, file processing, to electronic commerce and science computing. When data is exchanged between users, it is fragmented into units called segments/datagrams. Then these segments/datagrams are transferred by different transport-layer protocols.

Currently, Transmission Control Protocol (TCP) [1] is the most widely used transport-layer protocol in the Internet. TCP is one of the keys to the success of the Internet and the major transport protocol in use in most IP networks. Typically, TCP traffic is between 60% and 90% of the total load across the Internet [2]. It is used by applications that require guaranteed delivery. Guaranteed delivery means that an acknowledge (ACK) is returned by the receiver when a packet is correctly delivered from the sender to the receiver. Given this major role for TCP, TCP was designed to provide a reliable end-to-end byte stream over an unreliable IP network, while attempting to maintain high utilization of the network link, avoid overloading the bottleneck and provide fair sharing among competing flows, for

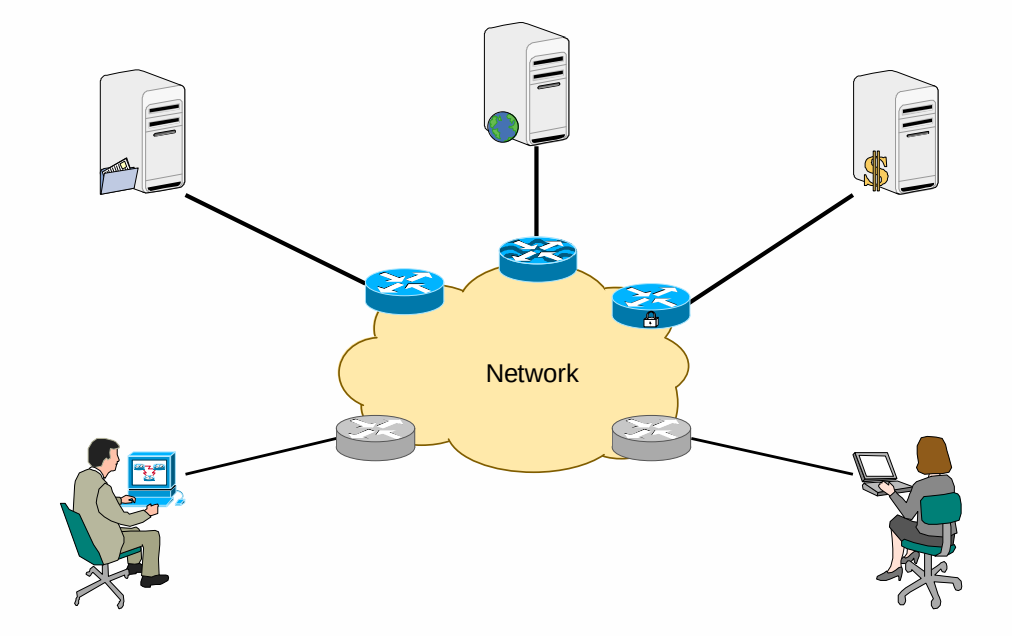


Figure 1.1: Illustration of a network

the network resource is not a private property.

For these functions, TCP has gone through various affairs and enhancements on its journey. The famous one is the first "congestion collapse" happened in October, 1986. During that period, the data throughput from LBL to UC Berkeley dropped from 32 Kbps to 40 bps. In order to avoid "congestion collapse", Van Jacobson proposed TCP flow control in 1988 [3]. This leads to the birth of TCP Tahoe. In addition to the original specification of TCP [1], that is a window-based flow control for the receiver to govern the amount of data sent by the sender, TCP Tahoe includes dynamic adjustment of the flow-control window in response to congestion. These mechanisms operate in the end-hosts to cause TCP connection to "back off" during congestion. Considerable research has been done since then.

In short, there are three major versions of TCP congestion control:

- TCP Tahoe:
  It is implemented in 4.3 BSD Tahoe, Net/1 (around 1988). Slow Start, Congestion Avoidance, and Fast Retransmit algorithms are included in TCP Tahoe. Congestion

window is reduced to one packet on a triple-ACK.

- TCP Reno:
  It is implemented in 4.3 BSD Reno, Net/2 (around 1990). A further optimization, Fast Recovery algorithm, is added to improve performance following Fast Retransmit. Congestion window is halved on a triple-ACK [4].

- TCP Vegas:
  It attempts to predict packet loss before it occurs by monitoring packet round trip times, and uses additive increases and additive decreases to control congestion window [5].

There are other TCP variants, such as NewReno [6] which improves the Fast Retransmit and Fast Recovery algorithms when multiple packets are dropped from a single window of data, TCP Selective Acknowledgment options (SACK) [7] which can also help to overcome the limitation when multiple packets are lost from one window of data. Both of them enhance the performance of TCP Reno. But the mechanism of TCP receiver needs to be modified for TCP SACK options.

So far, TCP Reno is the most popular one among the above-mentioned proposals, and its algorithm is standardized by IETF [8]. There are four algorithms – Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery. TCP Reno is a sliding window congestion control protocol. Its congestion window (CWND) is depicted in Figure 1.2. The follows are the glancing description of its algorithms. The details can be reviewed in RFC 2581.

Beginning transmission into a network, TCP probes the network capacity by Slow Start. The Slow Start algorithm is used by an end host to seek and enter equilibrium state of the network. The algorithm is called Slow Start, but it is not slow at all [3]. Congestion window is increased exponentially by one packet for every non-duplicate ACK until a packet loss is detected or the slow start threshold (*ssthresh*) is reached. If packet loss is detected by three duplicate ACKs, then TCP sender sets its *ssthresh* to half of its current CWND, then switches to Congestion Avoidance phase. Alternatively, a TCP sender also enter Congestion Avoidance phase after its congestion window becomes larger than *ssthresh*.

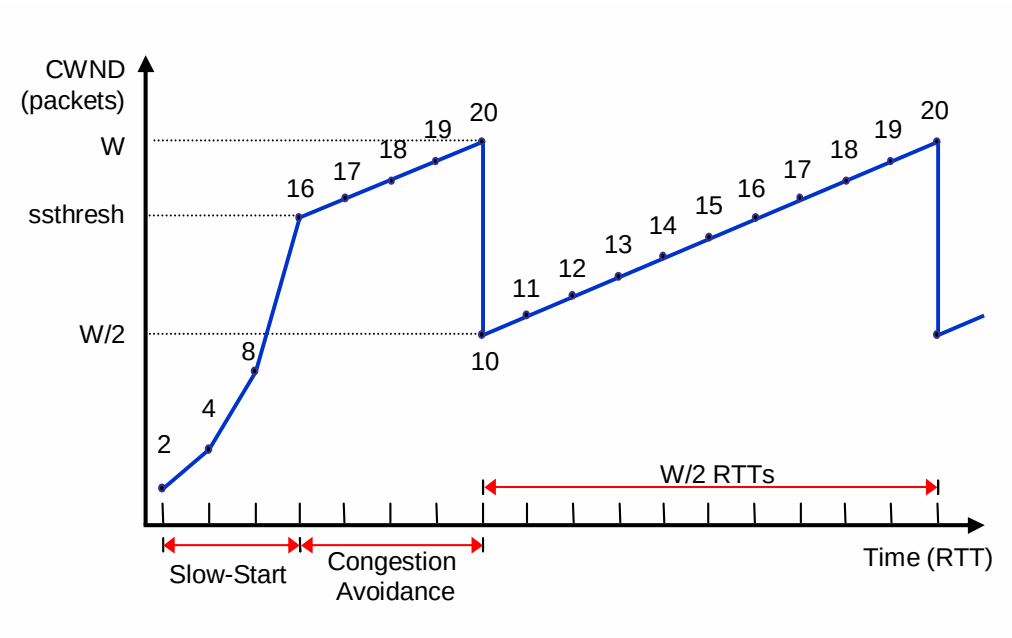


Figure 1.2: Visualization of congestion window (TCP Reno)

In Congestion Avoidance phase, the Additive-Increase and Multiplicative-Decrease (AIMD) algorithm is employed. Congestion window is increased by 1/CWND packet for each ACK that it receives. This means the TCP sender can increase its congestion window by one packet per round trip time (RTT). The window increase is interrupted when a loss is detected by three duplicate ACKs or a timeout. If timeout occurs, the TCP sender infers that a large number of packets were lost. There is severe congestion in the network. In this case, the congestion window is set to one packet, and then Slow Start starts. If packet loss is detected via three duplicate ACKs, the Fast Retransmit and Fast Recovery algorithms are called.

Once *ssthresh* is set to half of its congestion window, and the lost packet is retransmitted by Fast Retransmit, Fast Recovery governs the transmission of new data until a non-duplicate ACK arrives.

In the past years, TCP Reno, as the *de facto* standard transport protocol of TCP, works very well. However, continuous and explosive growth of the Internet has shown that TCP mechanisms can obstruct efficient use of fast long-distance networks (LFNs), which are high-bandwidth and large-delay networks. When TCP was designed in 1960–70s, T1 link

(1.544 Mbps) was a fast network. As time went on, link bandwidth of network has been growing quickly. Now, it is common that link bandwidth is 100 Mbps, 1,000 Mbps, or even higher. In such an environment, it was reported that TCP can not fully utilize the link bandwidth, especially in the case when the delay (distance) between a sender and a receiver hosts is large [9]. This is primarily because of the principle of AIMD [8] in the congestion control mechanism of TCP. In congestion avoidance phase, TCP increases its congestion window linearly by one packet per RTT, and sharply decreases its congestion window by half once it detects packet loss. It then needs a long time to increase its congestion window size for fully utilizing the network link bandwidth. For the example of a single TCP connection, if the link bandwidth is 1.544 Mbps, RTT is 100 ms and packet size is 1,500 bytes, when a packet loss occurs, it takes 0.643 sec to recover its congestion window until it can fully utilize the link bandwidth. However, when the link bandwidth increases to 1,000 Mbps, the recovery time becomes 416 sec, which is almost 650 times of the recovery time in the T1 case. A similar example can be found in [9], where a TCP Reno connection fills a 10 Gbps link and RTT is 100 ms, a congestion window of 83,333 packets is required, and 4,000 sec are needed to recover throughput when packets are lost in the network.

Addressing the problem of TCP used in LFNs, a number of enhancements to TCP have been developed, including TCP window scale option [10], SACK option [7], tuning TCP parameters [11], and using parallel TCP [12, 13]. However, these patches can not solve the problem thoroughly. Some high-speed protocols, e.g., HighSpeed TCP (HSTCP) [9], Scalable TCP [14], FAST TCP [15], and XCP [16], are proposed in recent years. Addressing the problem in AIMD algorithm, these high speed protocols modify the algorithm of TCP.

1.2 Related work

As mentioned above, some efforts have been spent in order to improving the TCP performance. Some early approaches are to patch the shortcomings of TCP Reno.

- **Window scale option:** TCP performance depends not upon the link bandwidth, but rather upon the bandwidth delay product (BDP). The TCP header uses a 16 bit field

to report the receive window size to the sender. Therefore, the largest window that can be used is 2^{16} = 65 Kbytes. This means that TCP can fully utilize a network link that its BDP is not larger than 65 Kbytes. This problem of "window size limit" with the current TCP is mended in [10]. The window scale extension expands the definition of the TCP window to 32 bits.

- **TCP SACK option:** Multiple packet losses from a window of data can have a catastrophic effect on TCP throughput. Address this problem, "TCP Selective Acknowledgment option" (TCP SACK option) is proposed [7]. With TCP SACK option, the data receiver can inform the sender about all segments that have arrived successfully, so the sender need retransmit only the segments that have actually been lost. However, this mechanism needs modification on both of sender and receiver sides.

- **NewReno:** In the absence of SACK option, there is little information available to the TCP sender in making retransmission decisions during Fast Recovery. In [6], the NewReno modification to TCP's Fast Recovery algorithm is proposed, and it needs only modification on the sender.

- **Large initial window:** If the initial window is one segment, a receiver employing delayed ACKs is forced to wait for a timeout before generating an ACK. With an initial window of at least two segments [17], the receiver will generate an ACK after the second data segment arrives. This eliminates the wait on the timeout. This modification also benefits TCP connections transmitting only a small amount of data.

- **Explicit congestion notification:** Since TCP determines the appropriate congestion window to use by gradually increasing the window size until it experiences a dropped packet, this causes the queues at the bottleneck router to build up. If traditional queue management, DropTail, is deployed at routers, the TCP sender get the congestion signal only when buffer overflows and packet is dropped. Another advanced form of router queue management is Active Queue Management (AQM), such as Random Early Detection (RED) [18]. AQM is meant to be a general mechanism using one of several alternatives for congestion indication. It can set a Congestion Experienced (CE) code in the packet header instead of dropping the packet, when such a field is

provided in the IP header and understood by TCP [19]. This is known as Explicit Congestion Notification (ECN) [20]. Upon receipt of a congestion marked packet, the TCP receiver informs the sender about incipient congestion which will in turn trigger the congestion avoidance algorithm at the sender. The congestion window is decreased. Thus, unnecessary packet dropping can be avoided.

The aforementioned patches improve TCP performance for common uses. However, it is out its power in LFNs, even with these enhancements. Parallel TCP is proposed as one possible approach for increasing the performance of TCP in LFNs.

Parallel TCP uses many concurrent TCP connections for one task. However, using parallel TCP is something of black art. This is because the key factor that affects its throughput is the number of concurrent TCP connections, and there is no method to find the optimal value. In addition to the number of TCP connections, another important factor is whether these concurrent TCP connections are synchronized or not, and it depends on the network conditions. Our results show parallel TCP is not an effective approach for LFNs. The details about parallel TCP will be discussed in Chapter 5 of the present thesis.

In recent years, the endeavors for improving TCP performance are to design high-speed protocol, and several such protocols are proposed. These protocols can be classified into two categories, both of them modify the algorithm of TCP. The first category requires modifications at both end-hosts and the routers in between, e.g., XCP [16], and VCP [21]. For using them, the mechanism of routers must be reconstructed, for some information gathered by routers needs to be fed back to the end-hosts. The second category only needs the modification of the congestion control mechanism of end-host's TCP, e.g., HSTCP [9], Scalable TCP [14], and FAST TCP [15]. Thus, they are relatively easy to be deployed in the current Internet. We briefly review two representatives, XCP and HSTCP.

XCP is a generalization of Explicit Congestion Notification (ECN) [20]. Instead of the one-bit congestion indication used by ECN, XCP routers inform the sender about the degree of congestion at the bottleneck. To control utilization, the new protocol adjusts its aggressiveness according to the spare bandwidth in the network and the feedback delay. This prevents oscillations, provides stability for high bandwidth or large delay, and ensures efficient utilization of network resources. Without keeping per-flow information, routers signal back to sources the needed changes in their congestion windows.

XCP also introduces the concept of decoupling utilization control from fairness control. A router has both an efficiency controller and fairness controller. The purpose of efficiency controller is to maximize link utilization while minimizing packet drop rate and persistent queues. The fairness controller uses the AIMD algorithm to converge to fairness. If the aggregate feedback is positive, allocate it so that the increase in throughput of all flows is the same. If the aggregate feedback is negative, allocate it so that the decrease in throughput of a flow is proportional to its current throughput. However, the mechanism of router must be modified. This may block its application.

HSTCP is the delegate of the second category. It aims at improving the loss recovery time of standard TCP by changing standard TCP's AIMD algorithm. This modified algorithm would only take effect with larger congestion windows, i.e., if the congestion window is smaller than a given threshold, it uses the standard AIMD algorithm, else it uses high speed AIMD algorithm.

In order to utilizing high bandwidth links, a large congestion window is required. Once a TCP flow is in congestion avoidance phase, it takes a large number of RTTs for this flow to use extra bandwidth available on the link. This leads to low utilization of high-bandwidth large-delay links. It also takes a large number of RTTs to recover its congestion window from consecutive timeouts. These result in low performance. HSTCP improves the performance of TCP in high bandwidth links where TCP operates with a large congestion window. In HSTCP, the parameters of AIMD algorithm are functions of the congestion window itself instead of being constant values as in standard TCP. This makes it possible to use high bandwidth links. HSTCP requires changes only on the TCP sender, so it is easy to be deployed. But, fairness is a problem when networks are shared by traditional flows. A new protocol should not achieve high performance based on pillaging resource from the competing one.

1.3 Contribution and organization of this thesis

The evolution of TCP is a careful balance between innovation and considered constraint. The evolution of TCP must avoid making radical changes that may not easy to be deployed, and also must avoid a congestion control "arms race" among competing protocols [22]. On

the other hand, the above-mentioned high-speed protocols are still on the way of development and not widely deployed. Any of them is not the consummate method. We think that it is better at present to design a suitable protocol for LFNs.

Therefore, addressing the TCP performance in LFNs and solve the problem of fairness against the traditional TCP Reno, an enhanced transport-layer protocol – Gentle High-Speed TCP (gHSTCP) – is proposed in the present thesis. gHSTCP which is based on HSTCP uses two modes in the congestion avoidance phase according to the changing trend of RTT. Simulation results show gHSTCP can significantly improve performance in mixed environments, in terms of throughput and fairness against the traditional TCP Reno flows. However, the performance improvement is limited due to the nature of TailDrop router, and the RED routers can not alleviate the problem completely. Thus, we present a modified version of adaptive RED (ARED), called gentle adaptive RED (gARED), directed at the problem of simultaneous packet drops by multiple flows in high speed networks. gARED can eliminate weaknesses found in ARED by monitoring the trend in variation of the average queue length of the router buffer. For application to real networks, gHSTCP is then tested in experimental environment, and a refined gHSTCP algorithm is proposed. Both results of simulation and experiment show that gHSTCP can utilize the high speed network while preserving the better fairness. An alternative approach for improving TCP performance in LFNs is parallel TCP mechanism. We also explore its performance by mathematical analysis in this thesis. The analytical results show that it is difficult to use parallel TCP in practice. The remaining chapters of the thesis is organized as follows.

Chapter 2. Gentle HighSpeed TCP (gHSTCP) for fast long-distance networks

HSTCP was proposed as one way to overcome the problems of TCP Reno and provide considerably greater throughput than TCP Reno in such environments. It modifies the increase/decrease algorithms of the congestion window size in the congestion avoidance phase of the TCP mechanism. That is, HSTCP increases its congestion window more quickly, and decreases it more slowly, than does TCP Reno to keep the congestion window size large enough to fill a high speed link.

Although HSTCP appears intuitively to provide greater throughput than TCP Reno, HSTCP performance characteristics have not been fully investigated, such as the fairness

issue when HSTCP and TCP Reno connections share the same link. Fairness issues are very important to TCP and have been actively investigated in past literatures [23–28]. Almost all of these studies have focused on the fairness among connections for a certain TCP version used in different environments and consider such factors as RTT, packet dropping probability, the number of active connections and the size of transmitted documents. Fairness among traditional and new TCP mechanisms, such as HSTCP, is a quite important issue when we consider the migration paths of new TCP variants. It is very likely that HSTCP connections between server hosts, and the many traditional TCP Reno connections for Web access and e-mail transmissions, will share high speed backbone links. It is therefore important to investigate the fairness characteristics between HSTCP and TCP Reno. It has also been mentioned in [9] that the relative fairness between standard TCP and HSTCP worsens as link bandwidth increases. When HSTCP and TCP Reno compete for a bandwidth on a bottleneck link, we do not attempt to provide the same throughput that they are capable of achieving. But in this case, high throughput by HSTCP should not occur at great sacrifice by TCP Reno, i.e., HSTCP should not pillage too many resources at the expense of TCP Reno. To address this problem, we propose an enhanced transport-layer protocol called gHSTCP, which is based on HSTCP. gHSTCP uses two modes in the congestion avoidance phase according to the changing trend of RTT. Simulation results show gHSTCP can significantly improve performance in mixed environments, in terms of throughput and fairness against the traditional TCP Reno flows.

Chapter 3. Improvement of adaptive RED mechanism for gHSTCP

The performance improvement of TCP is limited due to the nature of TailDrop router, which causes bursty packet losses and the large queueing delay. Congestion control to alleviate these problems can be accomplished by end-to-end congestion avoidance together with an active queue management (AQM) mechanism. Traditional TailDrop queue management could not effectively prevent the occurrence of serious congestion. Furthermore, global synchronization [29] could occur during the period of congestion, i.e., a large number of TCP connections could experience packet drops and reduce their transfer rates at the same time, resulting in under-utilization of the network bandwidth and large oscillations in queueing delay. Particularly in high-speed long-delay networks, where routers

may have large buffers, TailDrop can cause long queueing delays. To address these problems, Random Early Detection (RED) [18] has been recommended for wide deployment in the Internet as an active queue management mechanism [30]. However, control parameter settings in RED have been proven highly sensitive to the network scenario, and misconfiguring RED can degrade performance significantly [31–35]. Adaptive RED (ARED) was therefore proposed as a solution to these subsequent problems [36]. ARED can adaptively change the maximum drop probability in accordance with network congestion levels. However, in high speed and less multiplexed networks, our results indicate some remaining problems with ARED, such as synchronized packet drops and instability in queue length, leading us to develop a more robust ARED mechanism. This improved adaptive RED, which we call gARED, monitors average queue length and trends in the variation in order to dynamically adapt the maximum packet drop probability. Our approach, combining gARED and gHSTCP, is quite effective and fair to competing traffic than HSTCP with ARED.

Chapter 4. Experimental evaluation of gHSTCP

Chapter 4 contains the experimental results of gHSTCP. gHSTCP has been proposed in [37]. However, its effectiveness has only been demonstrated in simulation experiments. In this chapter, the performance of gHSTCP is checked in experimental environment, and a refined gHSTCP algorithm is proposed for application to real networks. The performance of the refined gHSTCP algorithm is then assessed experimentally. The refined gHSTCP algorithm is based on the original algorithm and compares two RTT thresholds, then judges which mode will be used. The performance of gHSTCP is compared with TCP Reno/HSTCP and parallel TCP mechanisms. The experimental results demonstrate that gHSTCP can provide a better tradeoff in terms of utilization and fairness against coexisting traditional TCP Reno connections, whereas HSTCP and parallel TCP suffer from the trade-off problem.

Chapter 5. Analysis of parallel TCP

In this chapter, the performance of parallel TCP is analyzed based on a dumbbell topology, i.e., there are N TCP connections competing for a bottleneck link. Packet drop rate

and aggregate goodput are used as two metrics to characterize the performance of parallel TCP. Two cases, namely synchronization and non-synchronization cases, are analyzed in detail. The synchronization case is common in parallel TCP, but the goodput deteriorates seriously. The non-synchronization case may benefit parallel TCP, whereas the extra mechanisms are required, and it is not easy to be implemented in the real world. Analytical results show that the issue of choosing the number of TCP connections is hard to be solved. Despite the mechanism that adjusts the number of TCP connections during data transfer is proposed, some potential problems still remain. All of these results show the difficulty of using parallel TCP in practice.

Chapter 6. Conclusion

In this chapter, we summarize the thesis and offer our conclusion.

Chapter 2

Gentle HighSpeed TCP (gHSTCP) for fast long-distance networks

Hosts (server machines) providing services that encompass data grids and storage area networks (SANs) have gigabit-level network interfaces such as gigabit ethernet. These hosts connect directly to high-speed networks for terabyte/petabyte-sized data exchange to move program data, perform backups, synchronize databases, and so on. Although they require large amounts of network bandwidth and disk storage, such services will grow in the future Internet as their costs are rapidly decreasing. However, the most popular version of TCP used on the current Internet, TCP Reno [8], cannot achieve sufficient throughput for this kind of high-speed data transmission because of the essential nature of the TCP congestion control mechanism.

In this chapter, the brief overview of HSTCP is given, and the investigation upon the throughput and fairness properties of HSTCP is proceeded when HSTCP shares link bandwidth with TCP Reno on a bottleneck link. Addressing the issues of HSTCP, a new high-speed transport-layer protocol – Gentle HighSpeed TCP (gHSTCP) – is proposed, and its performance is evaluated by simulations.

2.1 Introduction

2.1.1 HighSpeed TCP (HSTCP)

To overcome problems with TCP mentioned in Chapter 1, HSTCP was proposed [9]. The HSTCP algorithm uses the principle of Additive Increase Multiplicative Decrease (AIMD) as in standard TCP, but is more aggressive in its increases and more conservative in its decreases. HSTCP addresses this by altering the AIMD algorithm for the congestion window adjustment, making it a function of the congestion window size rather than a constant as in standard TCP.

In response to a single acknowledgment, HSTCP increases the number of segments in its congestion window w as:

$$w \leftarrow w + \frac{a(w)}{w}.$$

In response to a congestion event, HSTCP decreases the number of segments in its congestion window as:

$$w \leftarrow (1 - b(w)) \times w,$$

Here, $a(w)$ and $b(w)$ are given by:

$$a(w) = \frac{2w^2 \cdot b(w) \cdot p(w)}{2 - b(w)} \tag{2.1}$$

$$b(w) = (b_{high} - 0.5)\frac{\log(w) - \log(W_{low})}{\log(W_{high}) - \log(W_{low})} + 0.5 \tag{2.2}$$

$$p(w) = \frac{0.078}{w^{1.2}} \tag{2.3}$$

where b_{high}, W_{high} and W_{low} are parameters of HSTCP.

According to Equations (2.1) and (2.2) and a typical parameter set used in [9] (b_{high}, W_{high} and W_{low} are 0.1, 83,000 and 38, respectively), Figure 2.1 shows how $a(w)$ and $b(w)$ vary with the congestion window. We can see that the "increase" parameter $a(w)$ becomes larger, and the "decrease" parameter $b(w)$ becomes smaller, as the congestion widow size increases. In this manner, HSTCP can sustain a large congestion window and fully utilize the high-speed long-delay network.

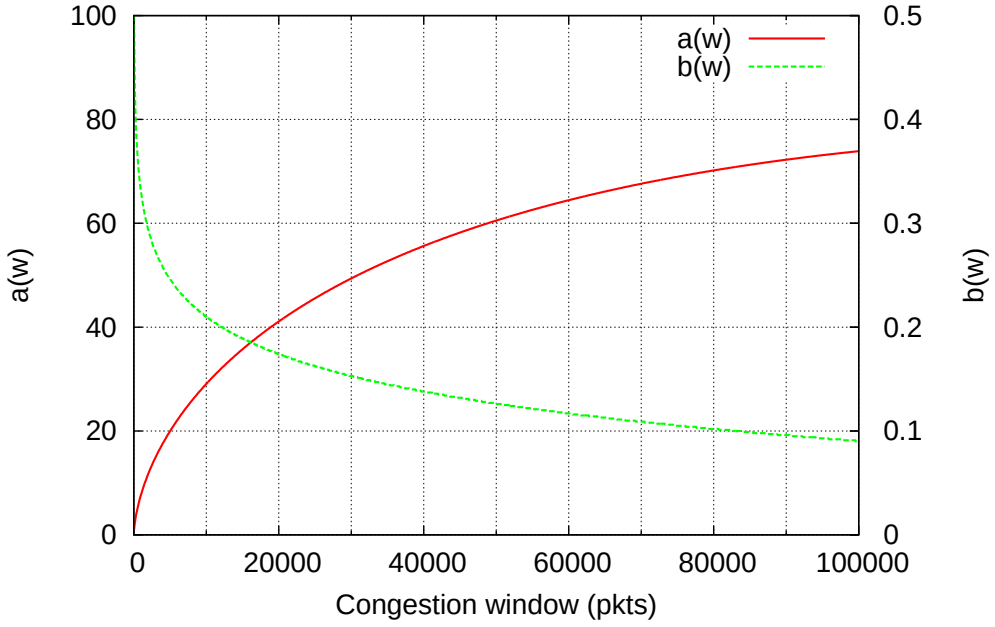


Figure 2.1: AIMD parameters of HSTCP

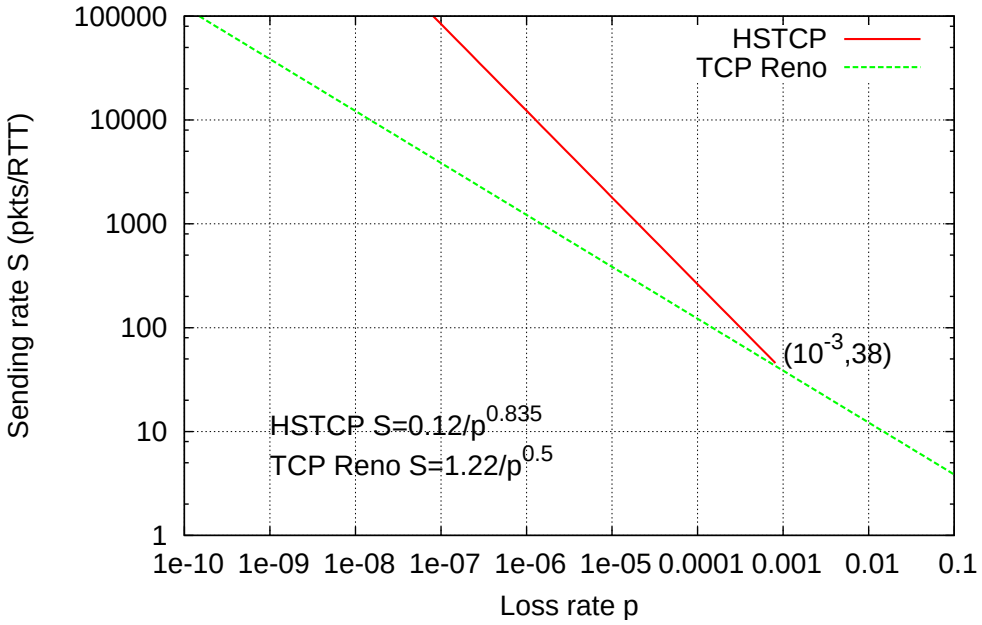


Figure 2.2: Response function of TCP Reno and HSTCP

In steady-state, the performance of TCP can be characterized by response function. Response function of TCP is the average throughput of a TCP connection in terms of the packet loss probability, the packet size, and the round-trip time. It maps the steady-state packet drop rate to the TCP average sending rate in packets per RTT. The HSTCP response function (2.3) is illustrated in Figure 2.2. We can observe from this figure that HSTCP relaxes the constraint between drop probability and the congestion window. For example, when $p = 10^{-7}$ is in steady-state, HSTCP can send at the rate of 100,000 packets/RTT while the sending rate of TCP Reno is around only 4,000 packets/RTT. Consequently, HSTCP can achieve a large congestion window even with a high loss rate.

2.1.2 Related work

There are other solutions for overcoming the limitations of standard TCP in high-speed networks. For example, Scalable TCP [14] is a simple change to the traditional TCP congestion control algorithm. On detection of congestion, it reduces the congestion window in segments by $0.125 \times cwnd$. For each acknowledgment received when congestion has not been detected, it increases the congestion window in segments to $cwnd + 0.01$. This increase is exponential instead of linear. Scalable TCP probing times are proportional only to the RTT to make the scheme scalable to high-speed networks. However, Scalable TCP exhibits unfairness to TCP Reno greater than that of HSTCP [9].

FAST TCP [15] is another example which based on TCP-Vegas [5] to provide a stable protocol for high-speed networks. In addition to packet loss, it uses queuing delay as the main measure of congestion. Although experimental results show Vegas can achieve better throughput and fewer losses than standard TCP Reno, there are few theoretical explanations for it. Any problems with TCP-Vegas exist possibly within FAST TCP, since its congestion control mechanism is based on that of TCP Vegas [38].

The above-mentioned proposals need modifications only on end-hosts. XCP [16], however, is a router-assisted protocol. XCP-enabled routers inform senders concerning the degree of congestion at a bottleneck. XCP introduces a new concept in which utilization control is decoupled from fairness control. It produces excellent fairness and responsiveness as well as a high degree of utilization. However, it requires the deployment of XCP

routers, therefore it cannot be deployed incrementally.

Protocols aiming at high speed environment are still on the way of development and not widely deployed. We think that it is better at present to design a suitable protocol for high speed network. Thus we focus on the performance and solve the problem of fairness by modifying aggressive protocol in the whole network as the case of gHSTCP in this chapter. gHSTCP can utilize the high-speed network while preserving the better fairness against the traditional TCP Reno. In addition, it is simply and easy to deploy.

Both FAST TCP and gHSTCP use RTT as a method to regulate the congestion windows. But gHSTCP is easy to implement. It only changes the increasing speed of congestion window based on the increasing RTT trend. Even with inaccurate estimation, gHSTCP maintains the same increasing speed of congestion window as TCP Reno does. For using XCP, the mechanism of routers must be reconstructed for the end hosts to get information from the routers.

2.2 Gentle HighSpeed TCP (gHSTCP)

In this section we present simulation results to show problems with HSTCP and propose a simple yet effective modification, which we call gHSTCP. We take advantage of HSTCP in terms of its AIMD algorithm for aggressive increase and conservative decrease of the congestion window. To gain better fairness with TCP Reno, we modify the strategy for increasing the congestion window. We then illustrate how gHSTCP outperforms HSTCP through simulations.

2.2.1 Shortcomings of HSTCP

We first present the results of simulation experiments to clarify HSTCP problems with throughput and fairness. *ns-2* network simulator [39] is used for the simulations. The network topology is shown in Figure 2.3, where S_1 and S_2 represent sender groups consisting of sender hosts, and D_1 and D_2 represent sink groups consisting of destination hosts. R_1 and R_2 are routers with buffer size of 10,000 packets. The packet size is 1,500 bytes. The

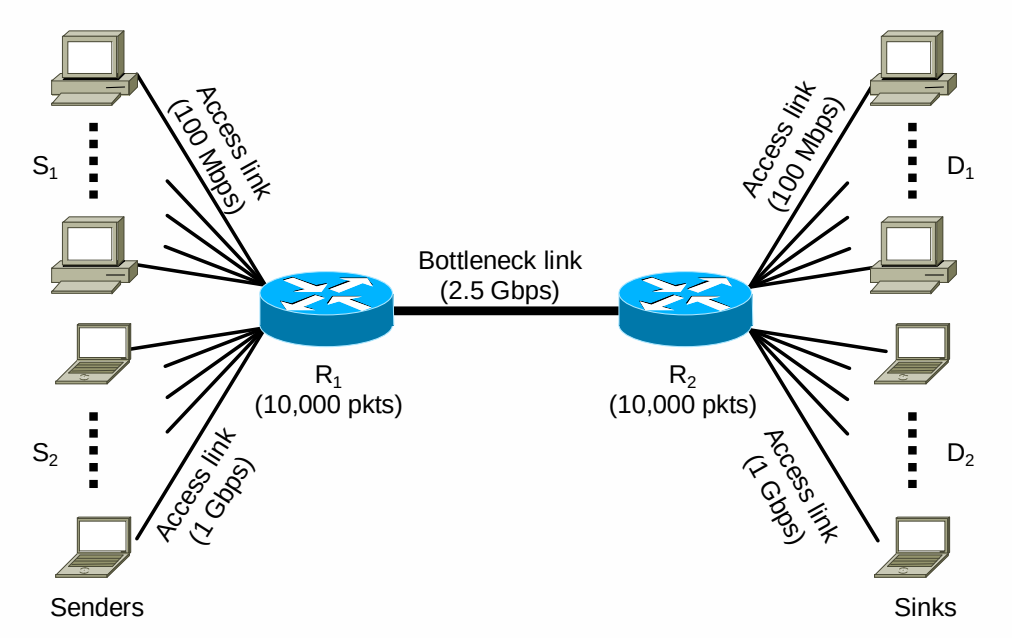


Figure 2.3: Network topology for simulation

bandwidth of the bottleneck link is set to 2.5 Gbps, and the propagation delay of the bottleneck link is set to 25, 50 and 100 ms, respectively. UDP traffic is used as background traffic. There are 10 TCP connections between senders and sinks. S_1 contains five connections with an access link bandwidth of 100 Mbps. S_2 contains five connections with an access link bandwidth of 1 Gbps. For TCP Reno and HSTCP connections, we show the simulation results with and without the Selective ACKnowledgement (SACK) option [7]. We denote HSTCP+SACK (Reno+SACK) and HSTCP (Reno) in the results, respectively. TailDrop is used as the queue management mechanism in this section. We use a greedy FTP source for data transmission.

We consider homogeneous environment in the following simulations although it is necessary to investigate heterogeneous environment, i.e., different delay of each connection. Homogeneous environment represents the worst case, which is worthy of special consideration to evaluate the performance of a new protocol.

Two metrics for the performance evaluation are used: aggregate throughput and fairness (Jain's fairness index). From a viewpoint of protecting a Reno connection as much as

possible, max-min fairness criteria is used in the chapter. Other methods such as proportional fairness need to cooperate with other mechanisms. We thought it is very difficult to attain proportional fairness only by improvement of TCP in end hosts. Jain's fairness index is defined as:

$$FairnessIndex = \frac{(\sum_{i=1}^{n} x_i)^2}{n\sum_{i=1}^{n} x_i^2}.$$

Here, n is the total connection number and x_i is the normalized throughput for flow i defined as $x_i = M_i/C_i$, where M_i is the measured throughput and C_i is the fair throughput found by max-min optimality. The fairness index always lies between 0 and 1. A value of 1 indicates that all connections are receiving the fairest allocation of bandwidth.

We first show the results of four simulations.

- Case 1: TCP Reno is used for S_1 and S_2.
- Case 2: TCP Reno is used for S_1 and HSTCP is used for S_2.
- Case 3: TCP Reno is used for S_1 and HSTCP+SACK is used for S_2.
- Case 4: TCP Reno+SACK is used for S_1 and HSTCP+SACK is used for S_2.

Table 2.1 shows the simulation results of four cases, and it presents the average throughput in the latter half of the simulation time and the fairness index defined by above equation. In Case 1, TCP Reno flows having high-bandwidth access links compete with TCP Reno flows having lower-bandwidth access links. We can see that S_1 group fully utilizes its access link bandwidth, and S_2 group, although it achieves higher throughput, does not utilize the entire available bandwidth. This confirms that TCP Reno cannot fully utilize the high link bandwidth.

In Case 2, HSTCP is used in S_2 group instead of TCP Reno. S_2 group obtains slight benefit from HSTCP, but performance of S_1 group is severely damaged and degradation in total throughput occurs. This is because the congestion window is inflated in S_2 group, resulting in more frequent buffer overflows and increasing packet loss in all of the flows. As we know, TCP Reno lacks a mechanism for recovering from a multiple packet loss event without incurring a timeout. Lost packets cause retransmission timeout (this is a fundamental mechanism of TCP Reno [40]), and timeout places the connection in the slow-start

phase, resulting in serious throughput degradation. Note that HSTCP uses the same algorithm as TCP Reno for packet retransmission. This is the reason why HSTCP connections in Case 2 cannot obtain high throughput compared with the TCP Reno connections in Case 1.

In Case 3, the TCP SACK option is applied with HSTCP for S_2 group. The TCP SACK mechanism [7], combined with a selective retransmission policy, can help overcome limitations in recovering from many packet losses. Table 2.1 shows that S_2 group achieves very high throughput while that of TCP Reno is severely degraded. Although there are still multiple packet drops, S_2 group, using the SACK option, infers the dropped packets and retransmits only the missed ones. Since this function is not available to S_1 group, it receives less link bandwidth compared to Case 2.

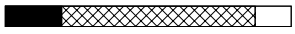
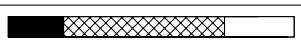
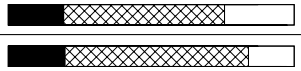
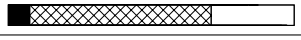
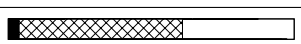
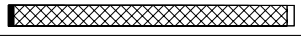
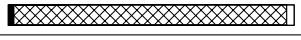
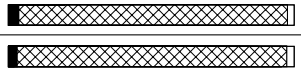
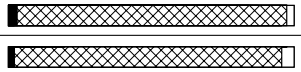
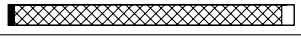
In Case 4, we can observe the aggregate throughput of S_1 group is slightly improved comparing with Case 3. It is because there is not timeout occurred to S_1 group due to the SACK option used. However, the throughput of S_1 group is still low. It means that other mechanisms are necessary to improve the fairness between S_1 and S_2 group.

It is clear in Case 1 that as propagation delay increases S_2 group does not affect S_1 group. This is because both groups employ the same mechanism and S_2 group cannot fully utilize the leftover bandwidth of S_1 group. But in Cases 2–4, the larger the propagation, the smaller the throughput that can be achieved by S_1 group due to the use of different algorithms by the two groups.

2.2.2 gHSTCP description

In view of the simpleness and standardization of HSTCP, we think it is a better way to design a new protocol based on HSTCP. HSTCP increases the congestion window size based solely on the current congestion window size. This may lead to bursty packet losses, because the window size continues to be rapidly increased even when packets begin queued at the router buffer. In addition, differences in speed gains among the different TCP variants result in unfairness. To alleviate the problem of HSTCP, we consider changing the behavior of HSTCP for speed increases to account for full or partial utilization of bottleneck links. We regulate the congestion avoidance phase in two modes, HSTCP mode and Reno mode,

Table 2.1: Performance of HSTCP with DropTail router

Case	S_1 group	S_2 group	Delay (ms)	Average throughput of S_1/S_2 S_1(Mbps) S_2(Mbps)	Confident interval[a]	Utilization (%)	Fairness
1	Reno	Reno	25		113.824	87.361	0.994
			50		109.676	75.634	0.976
			100		199.217	83.989	0.989
2	Reno	HSTCP	25		12.549	70.925	0.887
			50		14.005	64.362	0.821
			100		49.861	60.923	0.747
3	Reno	HSTCP +SACK	25		0.947	97.489	0.582
			50		2.040	97.397	0.581
			100		1.084	95.603	0.556
4	Reno +SACK	HSTCP +SACK	25		3.345	97.618	0.652
			50		1.532	97.431	0.629
			100		0.981	95.668	0.591

[a] It is for the total average throughput, the confidence level is 95%.

and switch between modes based on the trend of changing RTT.

Denote the departure time and RTT value of a transmitted packet i as d_i and t_i, respectively, the correlation between d_i and t_i is tested statistically. From pairs (d_i,t_i) to calculate the correlation coefficient r [41]:

$$r = \frac{\sum_{i=1}^{N}(d_i - \bar{d})(t_i - \bar{t})}{\sqrt{\sum_{i=1}^{N}(d_i - \bar{d})^2 (t_i - \bar{t})^2}},$$

where N is the size of congestion window in packet, $\bar{d}$, $\bar{t}$ are the mean values of d_i and t_i. If d_i and t_i tend to increase together, r is positive. If, on the other hand, one tends to increase as the other tends to decrease, r is negative. The value of correlation coefficient lies between -1 and +1.

Because the pairs (d_i,t_i) are N independent observations, r can be used to estimate the population correlation ρ. To make inference about ρ using r, usually N is a large number,

we require the sampling distribution of r by calculating the statistic Z:

$$Z = \frac{1}{2}\ln\left(\frac{1+r}{1-r}\right)\sqrt{N-3}.$$

If Z is larger than a certain value, there is very strong evidence of statistical significance, i.e., (d_i, t_i) is positive correlation, otherwise it is non-positive correlation. Z of 3.09 is used in the following simulation results. The parameter Z corresponds to the level of significance. The larger value of Z shows there is very strong evidence of correlation. In order to make a right estimation on the increasing RTT trend, we recommend to select a larger value for Z.

If a positive correlation is recognized, that is, an increasing trend in the observed RTT values is present, then bottleneck congestion is occurring for a sender. If more and more packets are buffered in the router queue, then the bottleneck is fully used. The sender should therefore slow down its increasing speed of the sending rate to keep the fairness against TCP Reno connections. The process during this period is referred to as Reno mode, in which the sender increases its congestion window linearly as with standard TCP. This will maintain fairness among TCP Reno and gHSTCP connections. On the other hand, if there is a non-positive correlation between d_i and t_i, it means the network is in an under-utilized state and the sender should increase the congestion window rapidly to utilize the unused bandwidth. The process during this period is called HSTCP mode. The sender increases the window size in the same way as HSTCP. The algorithm is summarized as follows.

When a new acknowledgment is received, gHSTCP increases its congestion window in segments as:

$$w \leftarrow w + \frac{a(w)}{w},$$

where $a(w)$ is given by:

$$a(w) = \begin{cases} \dfrac{2w^2 \cdot b(w) \cdot p(w)}{2 - b(w)} & \text{HSTCP mode} \\ 1 & \text{Reno mode.} \end{cases}$$

Once a retransmission timeout occurs, or duplicated acknowledgments are received, the sender decreases the congestion window in the same way as HSTCP does. When a timeout occurs, the congestion window size is reset to one packet and the phase is changed to slow-start. When a packet loss event is detected and retransmitted by fast retransmit algorithm then sets its congestion window size to $(1-b(w))\times w$, $b(w)$ is given by Equation (2.2) for two modes. If the sender host is in HSTCP mode, it remains in HSTCP mode. If a retransmission happens during Reno mode, the sender switches to HSTCP mode.

2.3 gHSTCP evaluation with simulations

2.3.1 Evaluation with DropTail router

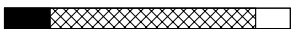
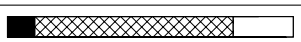
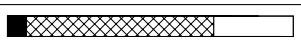
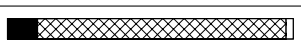
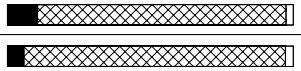
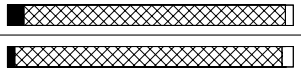
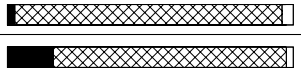
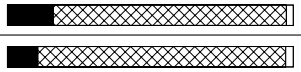
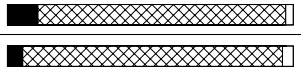
In this subsection we compare the performance of HSTCP and gHSTCP based on simulations. Using TailDrop as the queue management mechanism, the following simulations are performed:

- Case 5: TCP Reno is used for S_1 and gHSTCP is used for S_2.
- Case 6: TCP Reno is used for S_1 and gHSTCP+SACK is used for S_2.
- Case 7: TCP Reno+SACK is used for S_1 and gHSTCP+SACK is used for S_2.

The results are shown in Table 2.2. In Case 5, the throughput is significantly improved for both TCP Reno and gHSTCP comparing with Case 2. The fairness is also improved. In Case 6, the throughput of S_2 group is further increased due to the SACK option used for gHSTCP. However, it results in the fairness decreasing. But it is better than that in Case 3. In Case 7, when the SACK option is used with both groups, although total throughput is almost the same as the case when HSTCP is used, the fairness becomes better among the different flow types with the help of gHSTCP. The throughput of S_1 group is greatly improved comparing with that in Case 4.

Tables 2.1 and 2.2 show that the bottleneck is under-utilized when SACK option isn't present and TailDrop is deployed. They also illustrate degraded fairness among HSTCP (or gHSTCP) and TCP Reno flows as the bottleneck link delay become larger. In this

Table 2.2: Performance of gHSTCP with DropTail router

Case	S_1 group	S_2 group	Delay (ms)	Average throughput of S_1/S_2 S_1(Mbps) S_2(Mbps)	Confident interval	Utilization (%)	Fairness
5	Reno	gHSTCP	25		24.621	87.908	0.997
			50		12.540	78.933	0.925
			100		16.031	72.177	0.849
6	Reno	gHSTCP +SACK	25		1.556	97.405	0.893
			50		1.808	97.230	0.741
			100		1.750	96.034	0.612
7	Reno +SACK	gHSTCP +SACK	25		1.176	97.540	0.986
			50		1.936	97.362	0.897
			100		1.209	96.202	0.724

situation, HSTCP (or gHSTCP) connections are able to obtain larger throughput while the TCP Reno connections suffer degraded throughput. This is caused by the different algorithms used for increasing/decreasing the congestion window size. TCP Reno resizes its congestion window in the same way regardless of the current window size. HSTCP (or gHSTCP) increases its congestion window more rapidly and decreases it more slowly when the window size is larger. Consequently, when the propagation delay of the bottleneck becomes large, that is, when the bandwidth delay product of the bottleneck link becomes large, HSTCP (or gHSTCP) connections increase the size of their congestion windows quickly.

2.3.2 Evaluation with RED router

To improve network performance in terms of link utilization and system fairness, it has been proposed that Active Queue Management (AQM) such as RED be deployed in the Internet [30]. In contrast to TailDrop, which drops incoming packets only when the buffer is fully utilized, the RED algorithm drops arriving packets probabilistically, with the probability calculated based on changes in queue length of the router buffer [18]. Here, we replace TailDrop with RED and investigate the performance of HSTCP and gHSTCP.

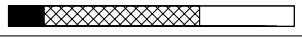
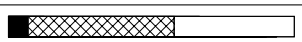
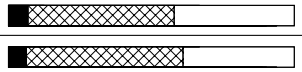
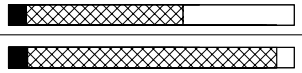
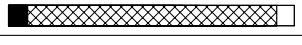
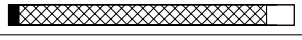
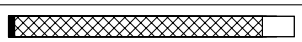
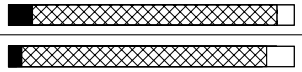
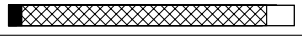
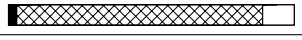
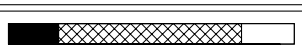
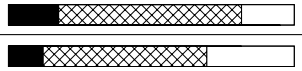
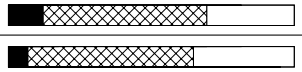
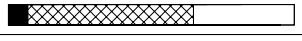
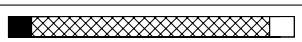
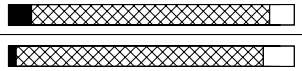
Topology and other conditions are the same as for the previous simulation experiments. According to [42], the latest router (especially backbone router) tends to have a buffer of 250 msec, and based on the simulation that we have conducted, the parameter used for RED is set as follows to maintain a good performance. The queue length minimum threshold, min_{th}, is set to 2,500 packets. The other RED parameters are set to their default values in *ns-2* ($max_{th} = 3 * min_{th}$, $w_q = 0.002$ and $max_p = 0.1$). Some applications cannot admit long queuing delay caused by such a large threshold. It can be solved by adjusting the parameters of RED. This is one of our future subjects that the better performance can be maintained while the queuing delay is shortened. The following simulation experiments are performed:

- Case 8: TCP Reno is used for S_1 and HSTCP is used for S_2 with RED deployed.
- Case 9: TCP Reno is used for S_1 and HSTCP+SACK is used for S_2 with RED deployed.
- Case 10: TCP Reno+SACK is used for S_1 and HSTCP+SACK is used for S_2 with RED deployed.
- Case 11: TCP Reno is used for S_1 and gHSTCP is used for S_2 with RED deployed.
- Case 12: TCP Reno is used for S_1 and gHSTCP+SACK is used for S_2 with RED deployed.
- Case 13: TCP Reno+SACK is used for S_1 and gHSTCP+SACK is used for S_2 with RED deployed.

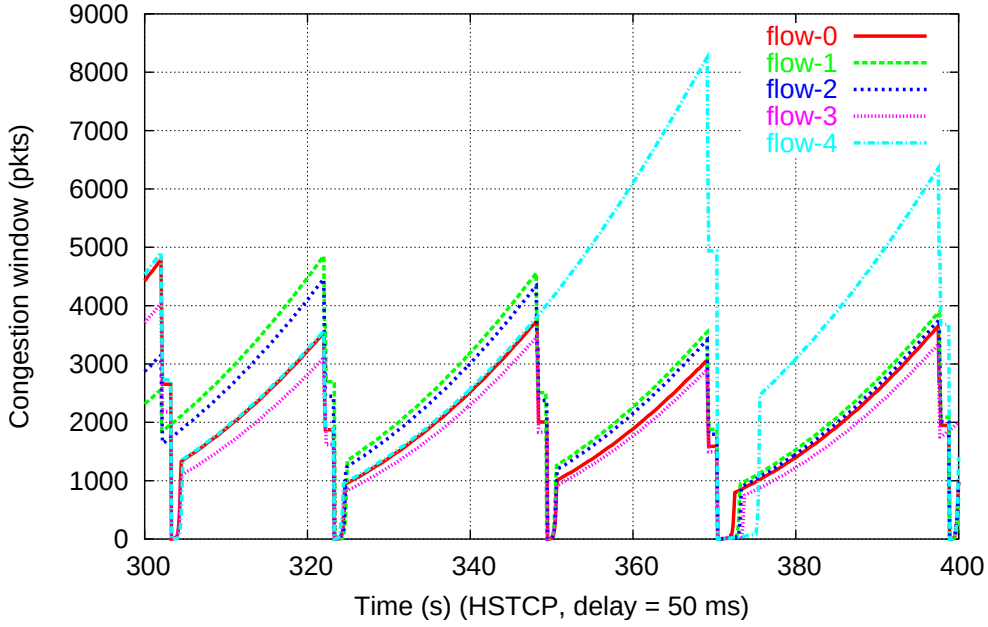
Comparing the results of Table 2.3 for Cases 8–13 with Tables 2.1 and 2.2, we see that fairness is improved, but link under-utilization is present and total throughput is less than that using TailDrop in some cases, especially in the case when the SACK option is not available. We expect the fairness is to be improved while maintaining the high utilization by introducing RED based on its policy of randomly dropping packets. However, the under-utilization problem can't be alleviated.

In this high-speed environment, high-speed flow has a very large congestion window. Once a packet loss event occurs, multiple packets are dropped although the packet drop

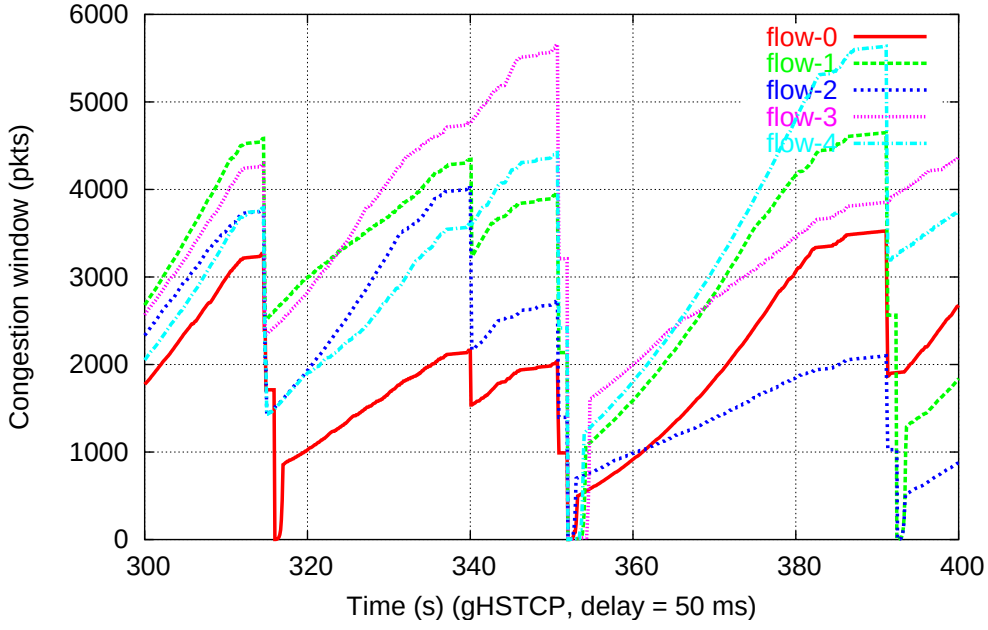
Table 2.3: Performance of HSTCP/gHSTCP with RED router ($max_p = 0.1$)

Case	S_1 group	S_2 group	Delay (ms)	Average throughput of S_1/S_2 S_1(Mbps) S_2(Mbps)	Confident interval	Utilization (%)	Fairness
8	Reno	HSTCP	25		59.000	66.891	0.998
			50		35.620	57.937	0.920
			100		80.048	61.054	0.881
9	Reno	HSTCP +SACK	25		4.097	93.889	0.782
			50		2.191	90.303	0.666
			100		4.882	88.831	0.600
10	Reno +SACK	HSTCP +SACK	25		1.564	93.920	0.841
			50		1.610	90.356	0.710
			100		3.842	88.943	0.635
11	Reno	gHSTCP	25		12.774	81.549	0.998
			50		25.582	69.398	0.993
			100		47.055	64.702	0.885
12	Reno	gHSTCP +SACK	25		2.906	95.065	0.979
			50		7.477	91.388	0.838
			100		4.729	89.124	0.625
13	Reno +SACK	gHSTCP +SACK	25		1.385	95.062	0.993
			50		4.357	91.595	0.906
			100		3.568	89.240	0.691

probability is quite small. This results in timeout if the SACK option is not used for high-speed flow. Figure 2.4 shows the change in the congestion window when HSTCP (or gHSTCP) is used with RED and the bottleneck link propagation delay is 50 ms. Although RED is deployed at the routers, global synchronization also occurs because of the multiple packet losses. This phenomena is present to a smaller extent when gHSTCP is used but can still happen. If the SACK option is used for the HSTCP (or gHSTCP) flows, though the congestion windows will not be reset to 1 packet as shown in Figure 2.5, it still occurs that all flows simultaneously decrease their congestion window due to the improper setting of RED. This may result in an under-utilization of the bottleneck link.

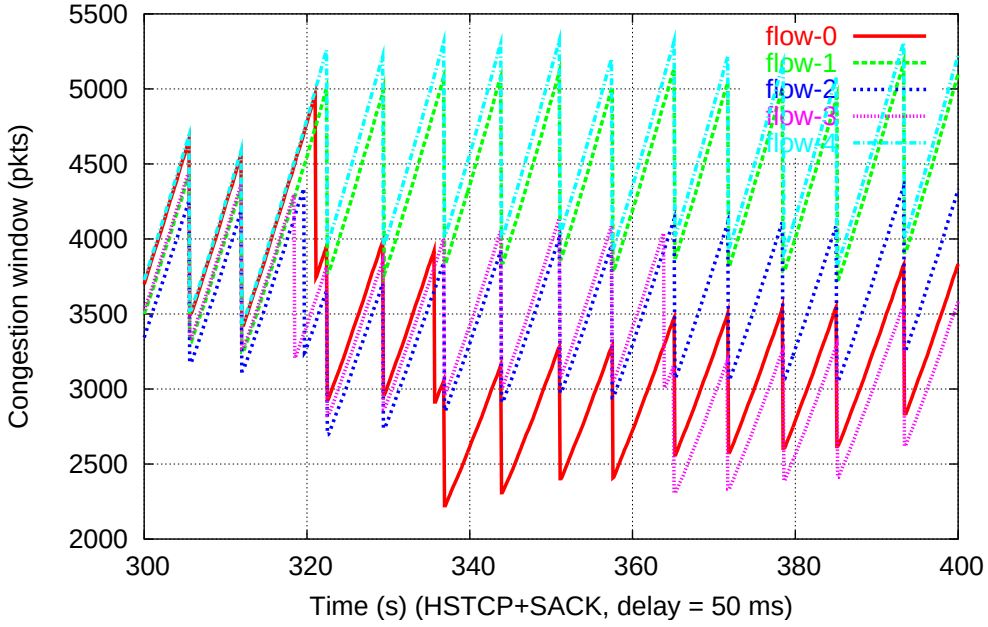


(a) HSTCP

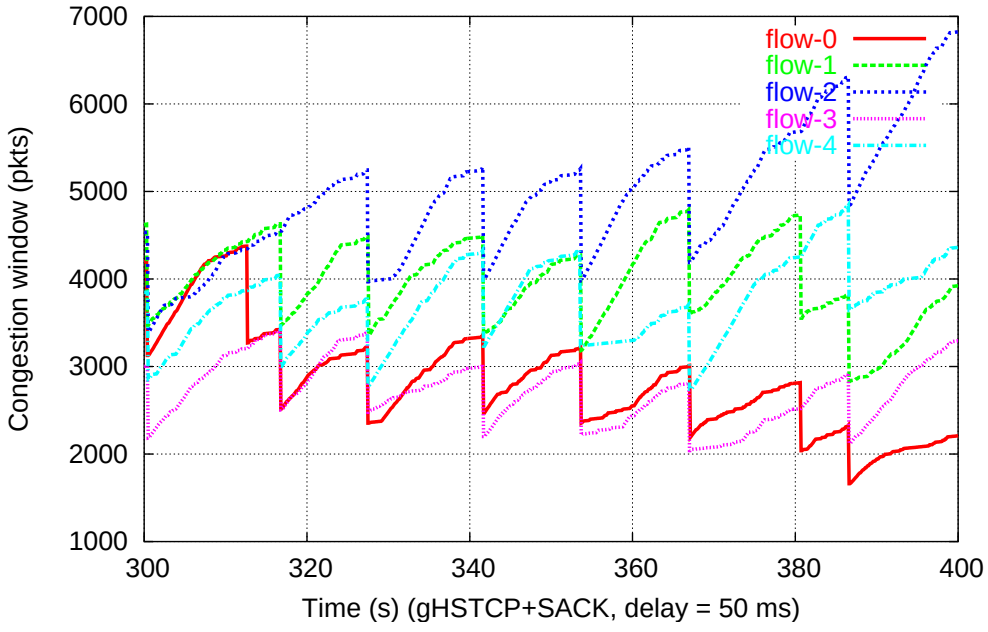


(b) gHSTCP

Figure 2.4: Changes of congestion window (HSTCP/gHSTCP with RED router)



(a) HSTCP+SACK



(b) gHSTCP+SACK

Figure 2.5: Changes of congestion window (HSTCP+SACK/gHSTCP+SACK with RED router)

In addition, it is reported [42] that synchronization tends to exist with certain condition, such as the number of coexisting connections is 100 or less regardless of the variation in RTT. In the environment where HSTCP (or gHSTCP) is used, since it is assumed that the multiplexed degree is not so high, it is necessary to develop a mechanism to reduce synchronization occurring.

It is well-known that system performance is quite sensitive to the RED parameters [31–35]. The following simulation experiments illustrate this problem, with correctly tuned RED parameter max_p set to 0.001:

- Case 14: TCP Reno is used for S_1 and HSTCP is used for S_2 with RED ($max_p = 0.001$).
- Case 15: TCP Reno is used for S_1 and HSTCP+SACK is used for S_2 with RED ($max_p = 0.001$).
- Case 16: TCP Reno+SACK is used for S_1 and HSTCP+SACK is used for S_2 with RED ($max_p = 0.001$).
- Case 17: TCP Reno is used for S_1 and gHSTCP is used for S_2 with RED ($max_p = 0.001$).
- Case 18: TCP Reno is used for S_1 and gHSTCP+SACK is used for S_2 with RED ($max_p = 0.001$).
- Case 19: TCP Reno+SACK is used for S_1 and gHSTCP+SACK is used for S_2 with RED ($max_p = 0.001$).

The results in Table 2.4 show that the system can achieve both higher throughput and better fairness in this situation. It means that in this situation, max_p is an important parameter to improve the performance of the RED algorithm. However, there is no complete parameter set of the RED mechanism to successfully cope with the various network conditions, since the RED parameters are very sensitive to the network factors [31–35]. In the next chapter, an additional mechanism will be introduced to address this problem.

Table 2.4: Performance of HSTCP/gHSTCP with RED router ($max_p = 0.001$)

Case	S_1 group	S_2 group	Delay (ms)	Average throughput of S_1/S_2 (S_1(Mbps) S_2(Mbps))	Confident interval	Utilization (%)	Fairness
14	Reno	HSTCP	25		2.643	97.193	0.985
			50		14.792	94.621	0.950
			100		29.052	87.515	0.947
15	Reno	HSTCP +SACK	25		1.634	97.483	0.981
			50		2.891	96.377	0.934
			100		6.305	93.250	0.829
16	Reno +SACK	HSTCP +SACK	25		0.745	97.481	0.979
			50		1.486	96.463	0.946
			100		6.065	93.356	0.848
17	Reno	gHSTCP	25		0.849	97.435	1.000
			50		2.516	96.615	1.000
			100		24.977	92.151	0.987
18	Reno	gHSTCP +SACK	25		0.767	97.455	1.000
			50		1.738	97.145	0.996
			100		3.384	94.022	0.951
19	Reno +SACK	gHSTCP +SACK	25		1.983	97.403	1.000
			50		1.726	97.062	1.000
			100		7.117	93.896	0.953

2.3.3 Evaluation with Web-traffic

In previous sections, we evaluated the performance of HSTCP (or gHSTCP) in the environments where it competes the system resources with long-lived flows. A recent study [43] shows that short-lived flows such as Web traffic is one of main class applications in the Internet. In this section, we assess the performance of HSTCP (or gHSTCP) when they co-exist with Web traffic.

Topology used in simulation is shown in Figure 2.6. The delay of the bottleneck link is 25 ms. TailDrop is deployed at routers R_1 and R_2. S_i, D_i are HighSpeed flow senders and receivers, respectively. The access link bandwidth of each sender/receiver is 1 Gbps. Access-link of WWW server cluster and WWW client cluster are also 1 Gbps. In WWW

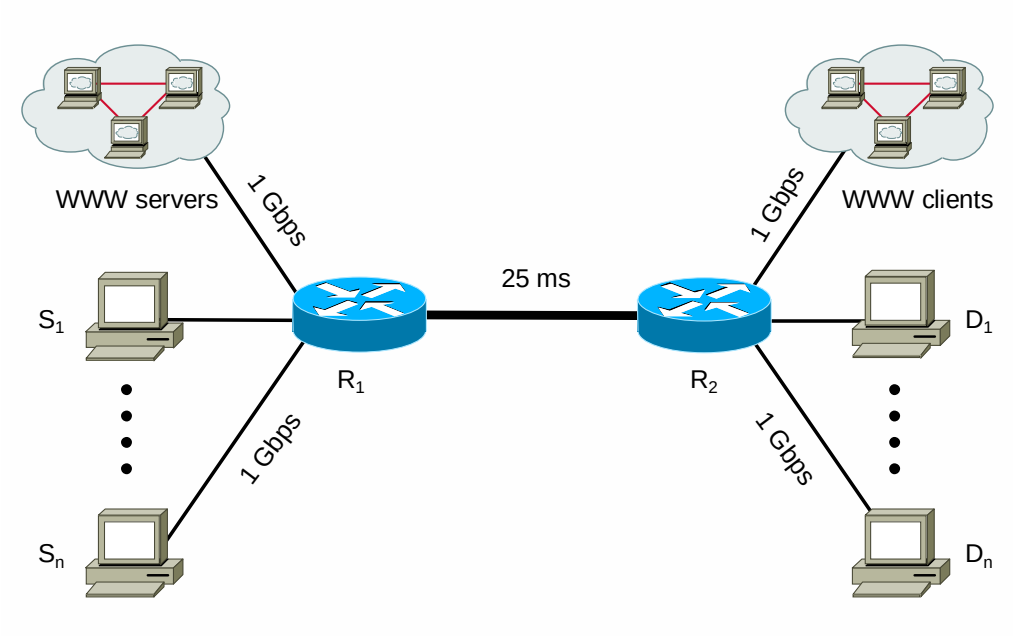


Figure 2.6: Network topology for simulation with Web-traffic

server cluster, there are 200 servers. Each www server access link is 1 Gbps, its link delay is uniform [10, 20] ms. In WWW client cluster, there are 1,000 clients with access link bandwidth of uniformly distributed in [100, 155] Mbps, and the delay is distributed in [20, 50] ms.

We use PagePool/WebTraf, a Web traffic model of *ns-2*, to generate synthetic Web traffic between the Web servers and clients. Probability distributions for user/session attributes are as follows [44]:

- Inter-page time: Pareto, mean = 10 s, sharp = 2
- Objects per page: Pareto, mean = 3, sharp = 1.5
- Inter-Object time: Pareto, mean = 0.5 s, sharp = 1.5
- Object size: Pareto, mean = 12 KB, sharp = 1.2

The packet loss rate at the router R_1 is used as a performance metric. In the simulations, the most Web traffic is alive in 50–800 s of the simulation time. The results are obtained in this period.

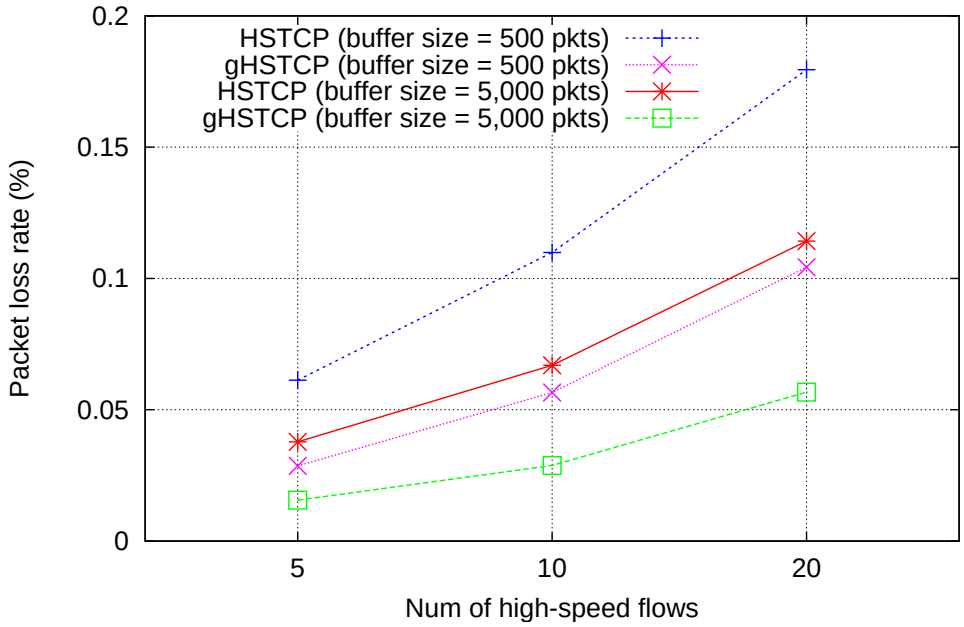


Figure 2.7: Packet loss rate (Bandwidth = 1,000 Mbps)

There are 4 sets of simulation conducted:

- Case 20: The bottleneck link bandwidth is 1,000 Mbps, the router buffer size is 5,000 packets.
- Case 21: The bottleneck link bandwidth is 1,000 Mbps, the router buffer size is 500 packets.
- Case 22: The bottleneck link bandwidth is 500 Mbps, the router buffer size is 5,000 packets.
- Case 23: The bottleneck link bandwidth is 500 Mbps, the router buffer size is 500 packets.

In each case, one of two different protocols – HSTCP and gHSTCP, is used for the high-speed flows and the number of high-speed flows is set to 5, 10 and 20, respectively. The results when the bottleneck link bandwidth is 1,000 Mbps are illustrated in Figure 2.7. Figure 2.8 shows the results when the bottleneck link bandwidth is 500 Mbps.

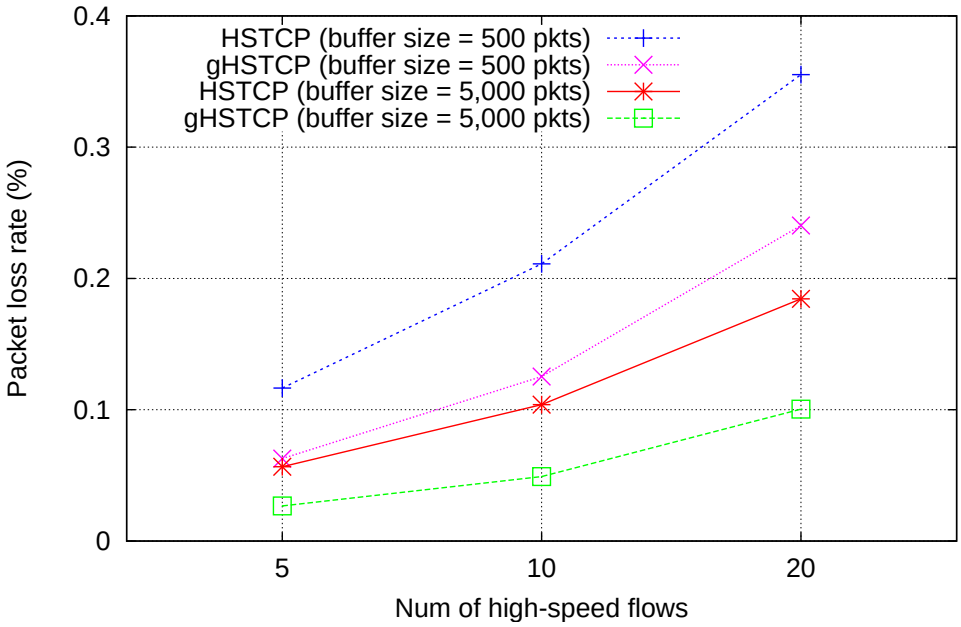


Figure 2.8: Packet loss rate (Bandwidth = 500 Mbps)

From the results we observe that as the router buffer size is decreased, the packet loss rate increases. It is because that there is no room enough to buffer the bursty coming packets, especially when HSTCP is used. If the bottleneck link bandwidth becomes small, the system has no sufficient ability to forward the coming packets, this leads to a higher packet loss rate.

However, we observe that the system has a lower packet loss rate in any case when gHSTCP is used for high-speed flows. These results reveal the merits of gHSTCP again. gHSTCP adjusts the increase speed of the congestion window according to the network conditions, and therefore can avoid the buffer overflow occurring frequently.

Web responding time is also checked. Figure 2.9 shows a CDF (Cumulative Distribution Function) graph of web responding time when the number of high speed flow is 10, the bottleneck is 1,000 Mbps and the router buffer size is 5,000 packets. It is slightly improved under the circumstance when gHSTCP is used. The loss rate is relatively small in these experiments. As discussed in [27], the responding time of web flows is not very sensitive to lower loss rate. Thus, the difference of web responding time isn't so conscious in these experiments. However, associating the previous experiments in which gHSTCP

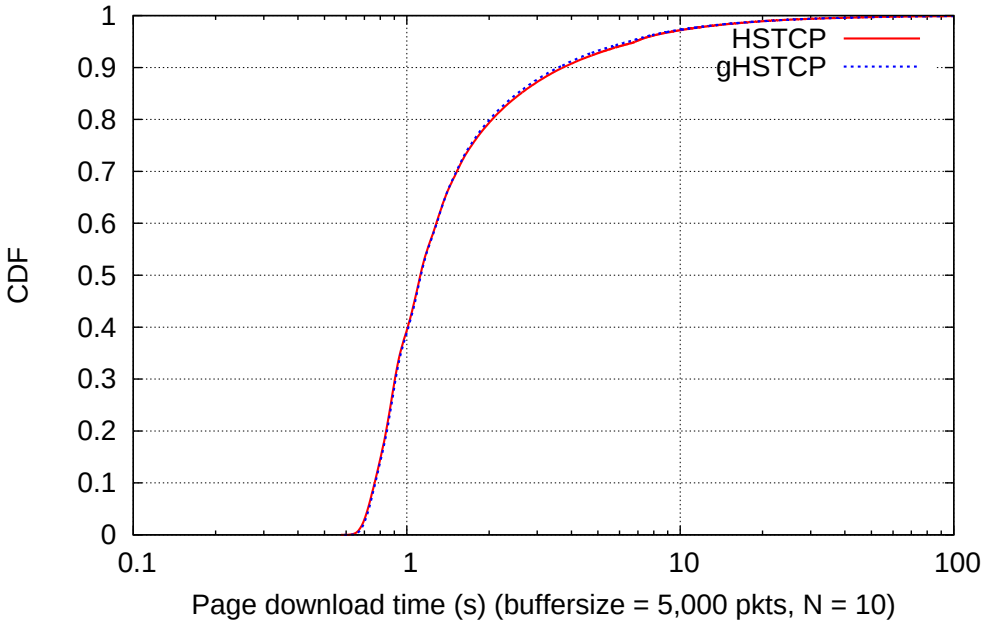


Figure 2.9: Web responding time

can provide better utilization and fairness when competing against TCP flows with the experiments here it does not increase the responding time of short TCP flows yet, it can be concluded that gHSTCP is valid.

2.4 Summary

We have proposed a new approach for improving HSTCP performance in terms of fairness and throughput. Our proposal, gHSTCP, achieves this goal by introducing two modes in the congestion avoidance phase: Reno mode and HSTCP mode. When there is an increasing trend in RTT, gHSTCP uses Reno mode; otherwise, it uses HSTCP mode. The performance of gHSTCP is apprised by simulation experiments. However, the performance improvement is limited due to the nature of TailDrop router, and the RED routers can not alleviate the problem completely. Therefore, we present a modified version of adaptive RED – gentle adaptive RED (gARED) – in the next chapter. It directed at the problem of simultaneous packet drops by multiple flows in high speed networks.

Chapter 3

Improvement of adaptive RED mechanism for gHSTCP

The results in Chapter 2 are primarily the effects of the TailDrop and RED mechanisms at the bottleneck routers. We observed that max_p is an important parameter that significantly affects system performance when RED is deployed. We need a mechanism that can adjust the parameters automatically, especially max_p, in response to the network environment. Adaptive RED (ARED) [36], an improved version of RED, is such a mechanism, and its application is expected to improve system performance. Because ARED enhances the performance of RED, and that RED has been recommended by IETF and implemented by some device manufacturers, we think the modification based ARED is easily carried out when the migration/update is considered. We first conduct simulation experiments with ARED and reveal its shortcomings from the results. We then propose a modification to alleviate these deficiencies, but that still preserves the effectiveness of the ARED mechanism, especially aiming at the absence of the SACK option.

3.1 Introduction

3.1.1 Adaptive RED mechanism

RED monitors impending congestion by maintaining an exponential weighted moving average of the queue length ($\bar{q}$). However, RED parameter settings have proven to be highly sensitive to network conditions, and performance can suffer significantly for a misconfigured RED [31, 32]. The motivation for ARED is to diminish or eliminate the shortcomings of RED, i.e., remove the effect of the RED parameters on average queue length and performance. Following is a brief overview of the differences between RED and ARED, the details of which can be reviewed in [36].

In RED, max_p does not change at runtime. In ARED, max_p is dynamically adapted to keep the average queue size within the target queue boundaries according to network conditions. When the average queue size is larger than the target queue size, max_p is increased. When the average queue size is less than the target queue size, max_p is decreased. One recommended range for max_p is (0.01, 0.5).

The suggestion of setting max_{th} is different. RED recommends setting max_{th} to at least twice min_{th}. In ARED, the rule of thumb is to set max_{th} to three times that of min_{th}. The target queue is determined by max_{th} and min_{th} as $[min_{th} + 0.4 * (max_{th} - min_{th}), min_{th} + 0.6 * (max_{th} - min_{th})]$. The target queue, the objective for ARED adapting the max_p setting, determines the queuing delay expected at the router. The setting for min_{th} is determined by the network manager.

The parameter of w_q is used as a low-pass filter on the instantaneous queue size in order to estimate the long-term queue average. RED sets it to a fixed value. The fixed value is not suitable as the bandwidth link increases. ARED sets it to $1 - exp(-1/C)$, where C is the link capacity in packets/second. The intent here is to maintain the time constant on the order of RTT. Calculating the average queue size is the basis of the RED algorithm.

Of the above three changes, the first is a key factor because it is an adaptation to network conditions. The other settings are determined at system startup.

3.1.2 Simulation with ARED router

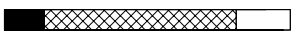
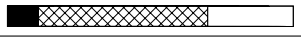
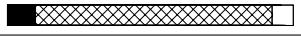
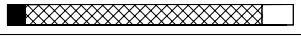
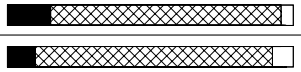
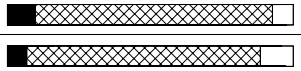
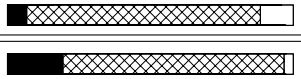
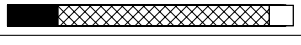
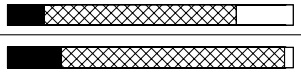
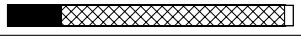
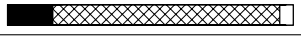
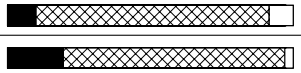
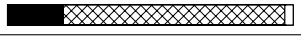
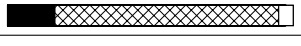
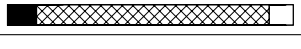
To evaluate the effectiveness of ARED in a high-speed long-delay network some simulations are conducted under the same conditions as in the previous chapter but with ARED deployed at the routers. Setting min_{th} to 2,500 packets, and setting the other ARED parameters as described in the previous subsection:

- Case 1: TCP Reno is used for S_1 and HSTCP is used for S_2 with ARED deployed.
- Case 2: TCP Reno is used for S_1 and HSTCP+SACK is used for S_2 with ARED deployed.
- Case 3: TCP Reno+SACK is used for S_1 and HSTCP+SACK is used for S_2 with ARED deployed.
- Case 4: TCP Reno is used for S_1 and gHSTCP is used for S_2 with ARED deployed.
- Case 5: TCP Reno is used for S_1 and gHSTCP+SACK is used for S_2 with ARED deployed.
- Case 6: TCP Reno+SACK is used for S_1 and gHSTCP+SACK is used for S_2 with ARED deployed.

The results are shown in Table 3.1. System performance is improved in terms of throughput and fairness compared with that of RED (Table 2.3). However, the bottleneck link remains under-utilized. Figure 3.1 shows the change in queue length for a propagation delay of 50 ms, and it is apparent that the router buffers are frequently in the idle state. This is why the bottleneck link bandwidth is not fully utilized due to an improper setting for the ARED packet drop probability. We now describe the shortcomings of ARED in detail.

The graph in Figure 3.2 shows a sketch map of the average queue size as it varies over time when using ARED. The purpose of the changing max_p is to maintain an average queue size within the target queue range. In the figure, the x-axis is time, the y-axis is the average queue length. When the average queue size increases to greater than the target queue size, ARED will increase max_p which in turn causes many of the flows to reduce their sending rates. This results in a decrease of the average queue size. When the average queue size

Table 3.1: Performance comparison of HSTCP/gHSTCP with ARED router

Case	S_1 group	S_2 group	Delay (ms)	Average throughput of S_1/S_2 S_1(Mbps) S_2(Mbps)	Confident interval	Utilization (%)	Fairness
1	Reno	HSTCP	25		10.075	91.465	0.995
			50		24.128	81.059	0.992
			100		20.574	70.070	0.976
2	Reno	HSTCP +SACK	25		4.323	95.813	0.970
			50		7.216	92.558	0.887
			100		7.465	89.026	0.778
3	Reno +SACK	HSTCP +SACK	25		4.080	95.690	0.980
			50		7.515	92.707	0.887
			100		16.732	88.473	0.801
4	Reno	gHSTCP	25		4.665	96.766	1.000
			50		6.549	91.555	1.000
			100		27.368	80.035	0.984
5	Reno	gHSTCP +SACK	25		0.828	96.965	1.000
			50		8.815	95.294	0.988
			100		5.446	91.502	0.897
6	Reno +SACK	gHSTCP +SACK	25		2.842	97.041	1.000
			50		3.232	94.892	0.994
			100		15.010	91.522	0.902

decreases to less than the target queue, max_p is decreased. With a smaller max_p, fewer connections suffer packet losses and the average queue size therefore increases. In this manner, ARED achieves its expected performance.

A problem with ARED is that it does not consider the trend in average queue variation. Given $t = t_1$, $max_p = p_1$, the average queue size ($\bar{q}$) is q_1. As $\bar{q}$ increases, max_p reaches a local maximum value p_2 at $t = t_2$, $\bar{q} = q_m$. This p_2 is large enough to ensure an average queue reduction. At $t = t_3$, $\bar{q}$ decreases and max_p is still increasing. At $t = t_4$, max_p reaches its maximum p_m, $p_m > p_2$. The larger max_p will converge the average queue size to the target queue size at a faster pace, but at the expense of a less stable state.

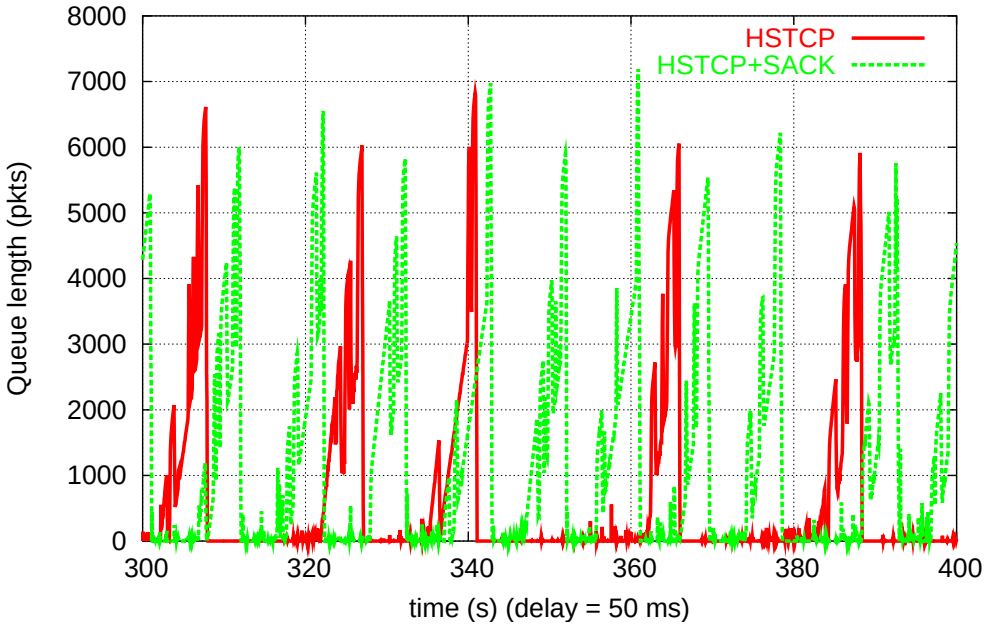


Figure 3.1: Instantaneous queue length (HSTCP/HSTCP+SACK with ARED router)

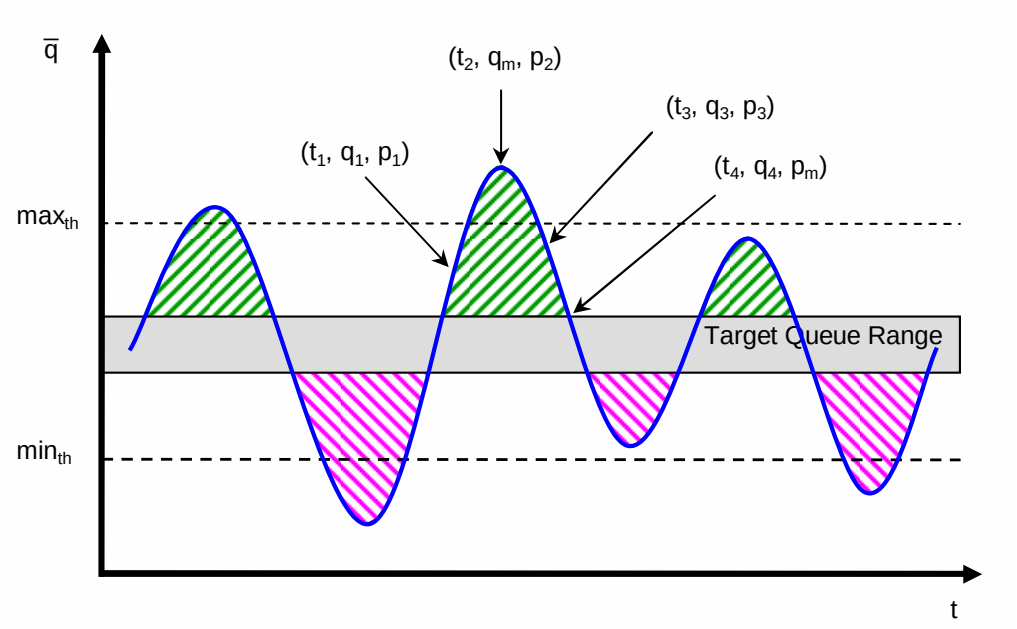


Figure 3.2: Sketch of average queue length (ARED mechanism)

We can view this process as a feedback control system [34] with the TCP senders as the controlled element, the drop module as the controlling element and the drop probability as the feedback signal. The feedback signal, delayed by about one RTT, causes senders to decrease their send rates to less than the ideal rate. Especially, the larger the drop probability, the more the TCP senders rates will be less than ideal. Moreover, as the propagation delay and queue size increase, this phenomenon will become more serious.

Another problem with ARED, the same as with RED, is that the lower bound of parameter max_p is determined to some extent by the network manager to ensure ARED performance.

3.2 Improvement of ARED: Gentle adaptive RED (gARED)

To solve these problems inherent to ARED, we propose a modified version referred to gentle adaptive RED (gARED) as shown in Figure 3.3. When the average queue becomes larger than the target queue and there is an increasing trend, max_p is increased. When the average queue becomes smaller than the target queue, then only if the average queue length is larger than min_{th} and there is a decreasing trend, max_p is decreased. When the average queue size is within target queue or less than min_{th}, there is no change on max_p.

Comparing gARED with ARED, if the average queue size is larger than the target queue size while t is in the interval (t_2, t_4), ARED increases max_p but gARED does not. The small max_p gives the network more stability. On the other hand, if the average queue size is less than the target queue, max_p is larger for gARED than one for ARED, So that the average queue can return to the target queue slowly.

Another difference between gARED and ARED is that there is no limit on the lower bound of max_p in gARED. It is determined automatically based on min_{th}.

The algorithm of gARED is given as:

```
avg: average queue length
old_avg: previous average queue length
top: upper bound of max_p
alpha: increment, min(0.01,max_p/4)
beta: decrease factor, 0.9
```

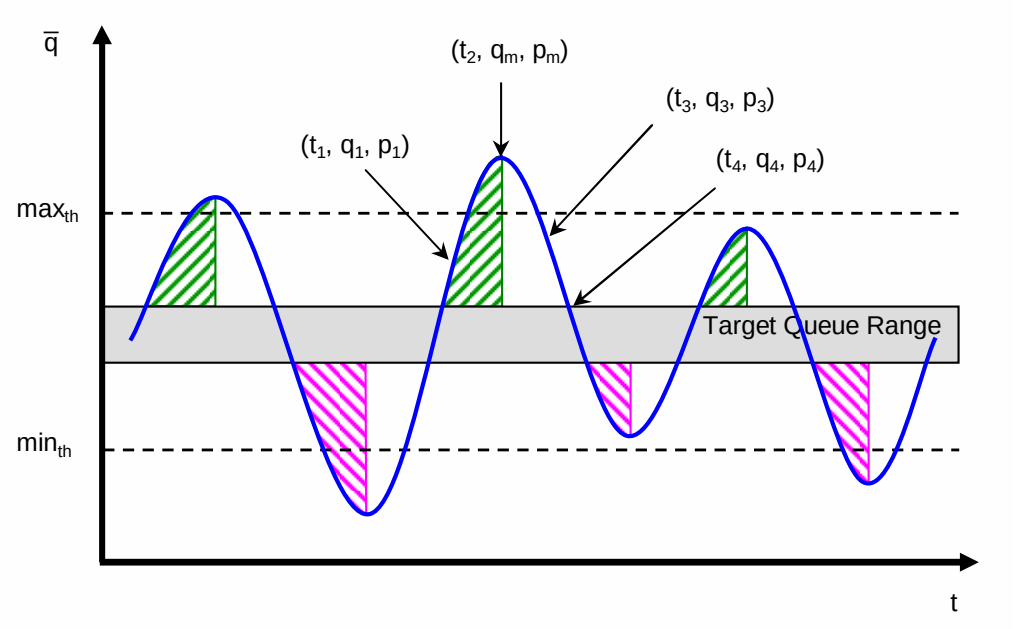


Figure 3.3: Sketch of average queue length (gARED mechanism)

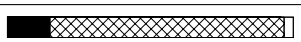
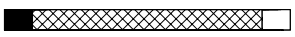
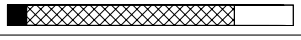
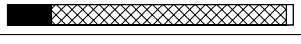
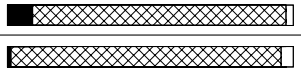
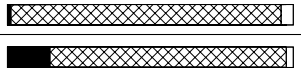
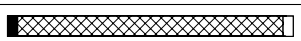
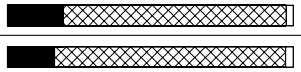
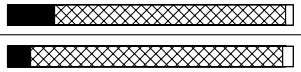
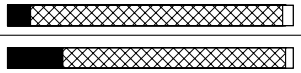
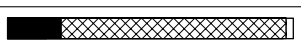
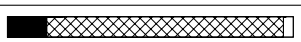
```
Every interval seconds:
if (avg > target and max_p < top and avg > old_avg)
    increase max_p:
    max_p ← max_p + alpha
if (min_th < avg and avg < target and avg < old_avg)
    decrease max_p:
    max_p ← max_p * beta
```

3.3 Evaluation of HSTCP/gHSTCP with gARED router

Table 3.2 shows throughput and fairness of simulation experiments when there are 5 HSTCP (or gHSTCP) flows competing with 5 TCP Reno flows, and gARED is used at the routers:

- Case 7: TCP Reno is used for S_1 and HSTCP is used for S_2 with gARED deployed.
- Case 8: TCP Reno is used for S_1 and HSTCP+SACK is used for S_2 with gARED deployed.

Table 3.2: Performance comparison of HSTCP/gHSTCP with gARED router

Case	S_1 group	S_2 group	Delay (ms)	Average throughput of S_1/S_2 (S_1(Mbps) S_2(Mbps))	Confident interval	Utilization (%)	Fairness
7	Reno	HSTCP	25		10.496	96.761	0.976
			50		22.905	90.103	0.895
			100		185.851	79.353	0.827
8	Reno	HSTCP +SACK	25		1.591	97.543	0.980
			50		1.542	97.406	0.844
			100		2.875	95.702	0.555
9	Reno +SACK	HSTCP +SACK	25		1.741	97.508	0.975
			50		1.860	97.425	0.789
			100		3.914	96.357	0.652
10	Reno	gHSTCP	25		10.335	97.277	1.000
			50		30.243	95.227	0.995
			100		113.194	94.362	0.942
11	Reno	gHSTCP +SACK	25		2.014	97.511	1.000
			50		6.657	97.328	0.990
			100		12.327	96.307	0.824
12	Reno +SACK	gHSTCP +SACK	25		2.023	97.310	1.000
			50		2.773	97.443	1.000
			100		9.237	96.528	0.962

- Case 9: TCP Reno+SACK is used for S_1 and HSTCP+SACK is used for S_2 with gARED deployed.
- Case 10: TCP Reno is used for S_1 and gHSTCP is used for S_2 with gARED deployed.
- Case 11: TCP Reno is used for S_1 and gHSTCP+SACK is used for S_2 with gARED deployed.
- Case 12: TCP Reno+SACK is used for S_1 and gHSTCP+SACK is used for S_2 with gARED deployed.

Table 3.2 shows the utilization is improved under gARED deployed. However, the fairness with HSTCP is not good. Especially, as delay is increased. It is because smaller packet

drop rate is set by gARED for keeping the target queue length. HSTCP increases its congestion window rapidly without having consideration for other competing TCP Reno flows. In contrast, gHSTCP based on RTT detection not only can achieve approving throughput, but also the better fairness can be obtained.

3.4 Summary

In this chapter, addressing problems with ARED in high-speed long-delay networks, we proposed a modified version of ARED, called gARED, that adjusts max_p according to the average queue length and the trend in variation. This technique avoids the problem of determining an appropriate lower bound for max_p. The simulation results reveal that our proposed algorithms outperform the original algorithms. gARED can eliminate weaknesses found in ARED by monitoring the trend in variation of the average queue length of the router buffer. Our approach, combining gARED and gHSTCP, is quite effective and fair to competing traffic than ARED with HSTCP.

Chapter 4

Experimental evaluation of gHSTCP

Based on our previous study [37], we have demonstrated that the fairness is a weakness of HSTCP. That is, HSTCP achieves high throughput, whereas the throughput of the competing TCP Reno is decreased when HSTCP and TCP Reno share the link bandwidth. In order to address this problem, we proposed the gHSTCP mechanism in [37]. gHSTCP, which is based on HSTCP, uses two modes in the congestion avoidance phase according to the increasing RTT trends. The simulation results presented in a previous study [37] indicate that, compared to HSTCP, gHSTCP provides better throughput on LFNs and maintains higher fairness against the traffic that passes through the same network paths.

However, we have investigated the characteristics of gHSTCP only by simulation experiments. Simulation plays a vital role in attempting to characterize a protocol, whereas the simulation condition is relatively ideal compared to the real network. Because the heterogeneity of the real network ranges from individual links and network equipments to the protocols that inter-operate over the links and the "mix" of different applications in the Internet, the protocol behavior in the simulation may be quite different from that in a real network. Therefore, emulating a protocol in a test-bed network is important with respect to its application to real networks, because the emulation network is more similar to a real network. In addition, the repetition of experiments under controlled conditions can be easier than in a real network. Thus, we herein present the evaluation results of gHSTCP in the test-bed network.

The present chapter makes the following three contributions:

- A refined gHSTCP algorithm that improves the behavior of gHSTCP in real networks is proposed.
- The performances of TCP Reno, HSTCP and gHSTCP are evaluated experimentally in an emulating test-bed network.
- The parallel TCP mechanism is evaluated as a possible candidate for the high-speed transport mechanism in LFNs.

The remainder of this chapter is organized as follows. First, we provide a short description of HSTCP and gHSTCP. Then, a refined gHSTCP algorithm is proposed. The experimental results for TCP Reno/HSTCP/gHSTCP and parallel TCP implementation in the test-bed network are presented. At last, we summarize the conclusions of the present chapter.

4.1 Introduction

In this section, we briefly illustrate the algorithms of HSTCP and gHSTCP. For more detailed descriptions, please refer to [9, 37].

In order to overcome the problems associated with using TCP Reno in LFNs, High-Speed TCP (HSTCP) was proposed in [9]. Figure 4.1 shows a rough sketch of the changes in the congestion window sizes of TCP Reno and HSTCP. The HSTCP algorithm uses the AIMD principle of TCP Reno but is more aggressive with respect to increases and more conservative with respect to decreases in the congestion avoidance phase.

HSTCP addresses this behavior by altering the parameters of the AIMD algorithm for the congestion window adjustment, making these parameters functions of the congestion window size, rather than constants, as in the case of TCP Reno. Based on this characteristic, TCP connections using the HSTCP mechanism can maintain large congestion windows in LFNs, so that the network link bandwidth can be better utilized.

HSTCP increases the congestion window size based solely on the current congestion window size. This may lead to bursty packet losses, because the congestion window size continues to be rapidly increased even when packets are queued at the router buffer, that is,

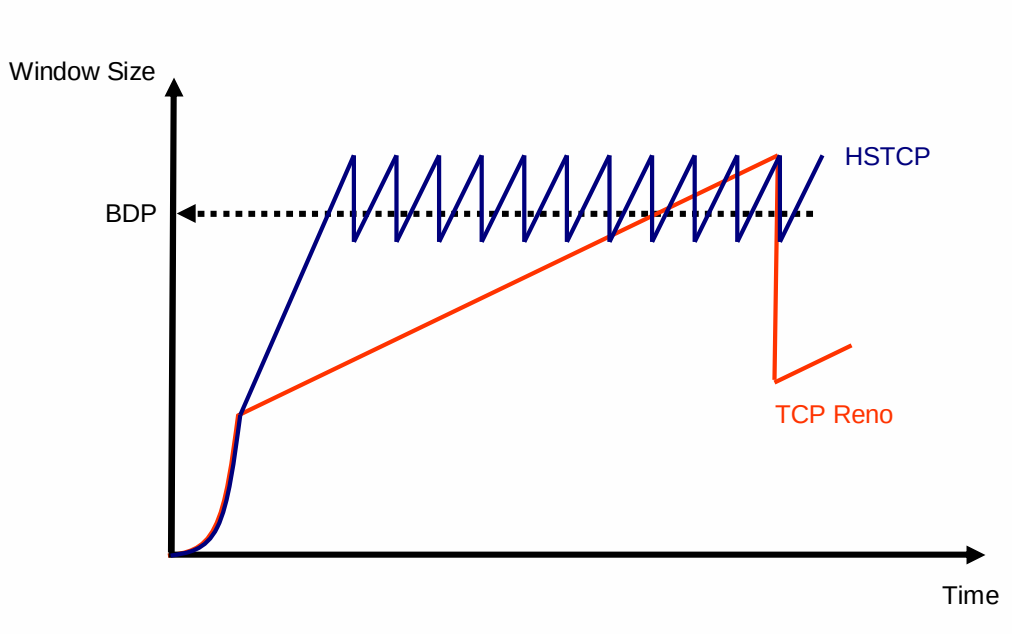


Figure 4.1: Changes of congestion window (TCP Reno and HSTCP)

when the network becomes congested. In addition, differences in speed gains among TCP Reno and HSTCP result in unfairness when these protocols co-exist in the network. In order to alleviate this problem, we considered changing the behavior of HSTCP in [37]. Two modes, the HSTCP mode and the Reno mode, are used in the congestion avoidance phase. Mode switching is based on the trend of changes in RTT values. Figure 4.2 shows the concept of the gHSTCP mechanism. The HSTCP mode is used before the link bandwidth is fully utilized, and the Reno mode is used if the link bandwidth is fully utilized. Therefore, TCP flows using gHSTCP can catch the link bandwidth as quickly as the original HSTCP, while providing better fairness with respect to competing TCP Reno flows.

We have shown that gHSTCP based on this mechanism can provide better performance and fairness by simulations [37].

Because the algorithm of gHSTCP is based on the RTT trend in one sample cycle, the granularity of time should be high enough to depict RTT's change in one sample cycle. The algorithm of gHSTCP shows its effectiveness under this condition and no matter whether Delayed-ACK option is used or not. As a matter of fact, the bottleneck link bandwidth is a crucial factor. For example, if the bottleneck link bandwidth is 10 Gbps, it needs 1.2

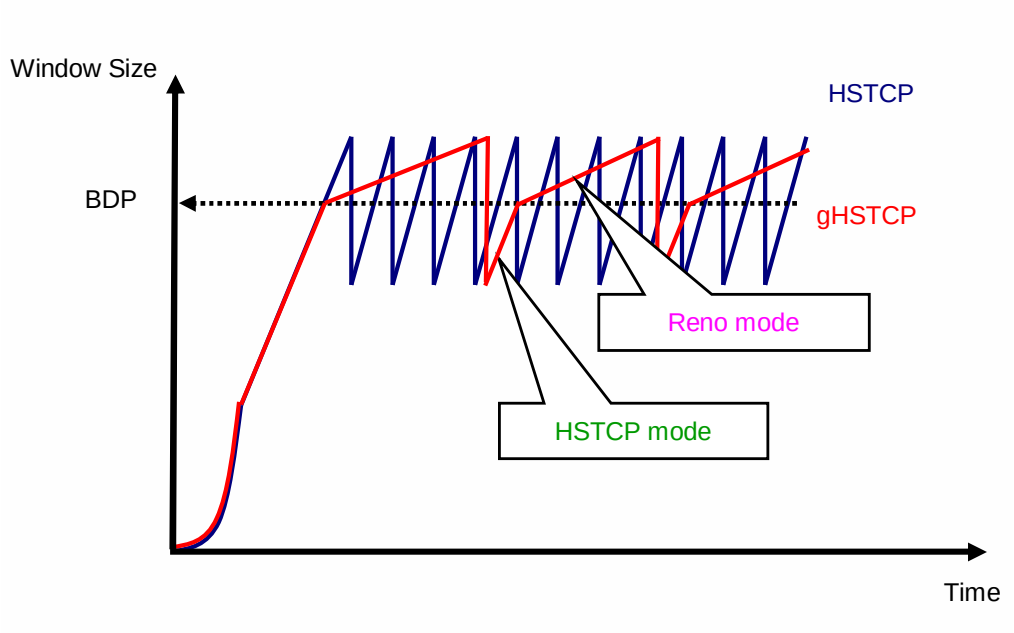


Figure 4.2: Changes of congestion window (HSTCP and gHSTCP)

microsecond to transmit one packet (given packet size is 1,500 bytes). In this case, gHSTCP can work if the granularity of time is microsecond. If the bottleneck link bandwidth is higher than 10 Gbps, higher time granularity is needed, or alternatively, larger sample cycle is used. In following experiments, gHSTCP is implemented in application level[1] and the granularity of time is microsecond, for *gettimeofday* and *RDTSC* functions are used. When an ACK is received by sender, the RTT value is calculated based on ACK's arrival time and timestamp in this ACK. Note that TCP timestamp option is used in the implementation of gHSTCP.

4.2 Validation of gHSTCP algorithm

In this subsection we present experimental results to demonstrate the problems with the original gHSTCP algorithm and then propose a refined algorithm.

[1]If gHSTCP is implemented in kernel, it is possible to use time slot with precision of kernel's timer (usually 1 msec). If timestamp option is turned on, the method of gHSTCP can be applied.

4.2.1 Problem description

We first conduct an experiment to check the behavior of gHSTCP in our test-bed network. The topology of the test-bed network, which is also used in following experiments, is shown in Figure 4.3. Dummynet [45] is used to emulate the bottleneck link between the sender and receiver hosts, which defines the link bandwidth, the delay and the buffer size. The specs of PC used by Dummynet are: OS is FreeBSD v5.2.1, CPU is Intel Xeon 3.2 GHz, Memory is 1 GB, Network adapter is Intel PRO/1000 MT Server Card. The specs of PCs used by end-hosts are: OS is Vine Linux v3.1, CPU is Intel Pentium 4 1.7 GHz, Memory is 256 MB. IPS and IPC use Intel PRO/1000 MT Server Card, IRS and IRC use Intel Fast Ethernet card. In this experiment, there is only one TCP flow from IPS to IPC to transfer unlimited data. The gHSTCP mechanism introduced in the previous section is used by the TCP connection. The run-time of the experiment is 90 s. In this experiment and following experiments, MSS is set to 1,460 bytes, and window scale option and timestamp option are used. SACK option is not used. The setting of Dummynet in this experiment is such that the bandwidth is 200 Mbps and delay is 22 ms. Thus, the bandwidth-delay product (BDP) of the network is 770 packets. A TailDrop mechanism is deployed at the bottleneck link buffer, and the buffer size is equal to 137 packets.

The experimental results of the change of the congestion window size as a function of time are shown in Figure 4.4, where the mode-switching and BDP are also plotted. The mode of gHSTCP does not change as expected. When the congestion window size is less than the BDP of the network path between the sender and receiver hosts, HSTCP mode is used, otherwise Reno mode is used. Its mode-switching oscillates severely in the experimental result. The shortcomings of this oscillation are as follows. First, gHSTCP cannot fill the link bandwidth quickly when the congestion window size is less than the BDP. Second, this oscillating action induces unfairness against the competing TCP Reno flow when the congestion window size is larger than the BDP. Third, the oscillation will lead to bursty packet losses if gHSTCP is in HSTCP mode just before the buffer overflows. Note that bursty packet losses cause retransmission timeout in TCP.

The reason for the mode oscillation is that the metric by which to determine the mode is based only on the increasing trend of the RTT. In a real network, RTT does not increase

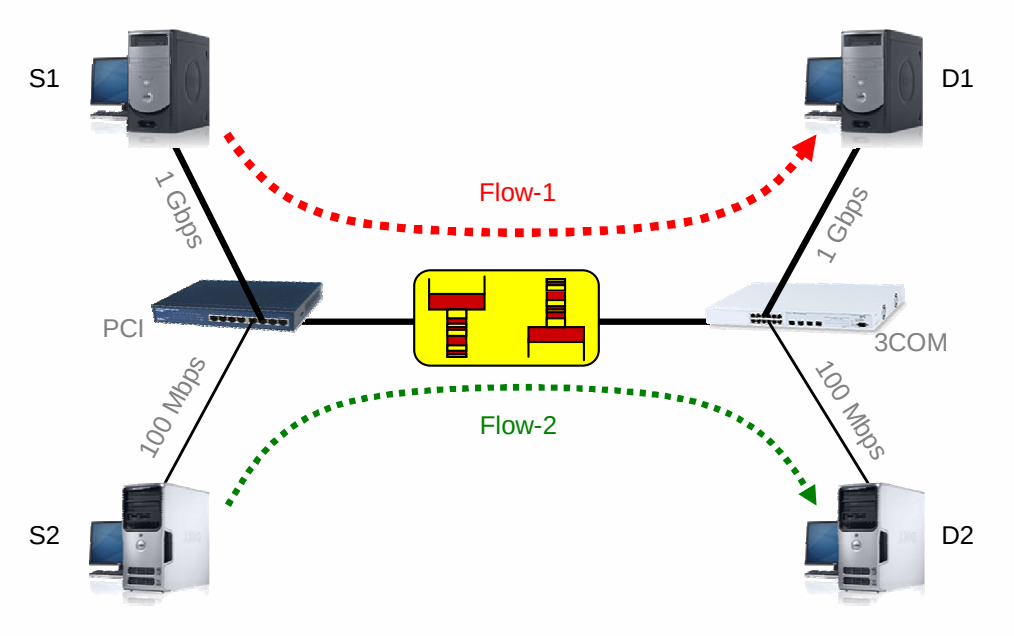


Figure 4.3: Experimental network topology

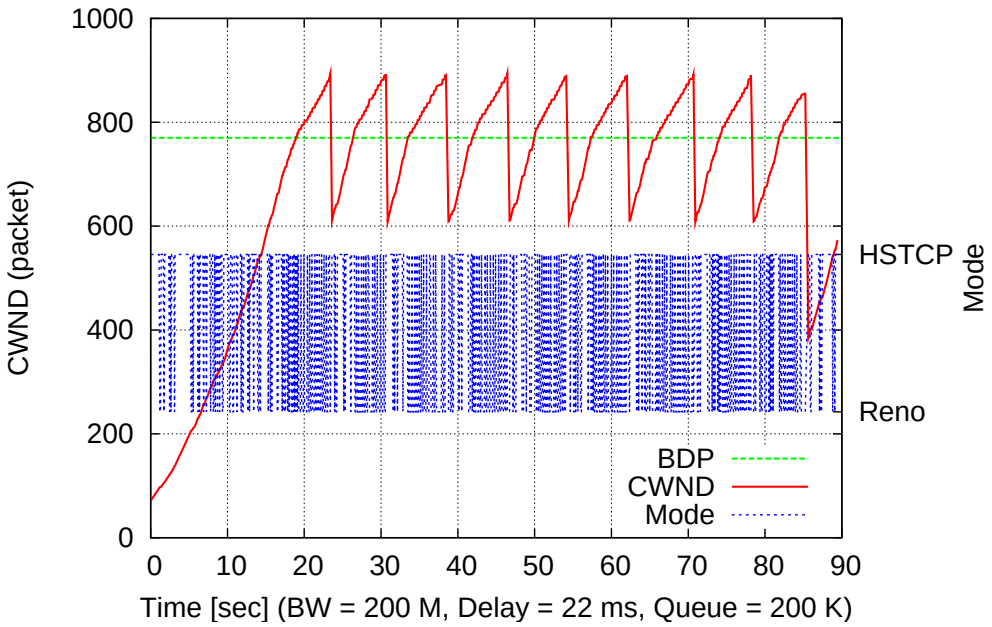


Figure 4.4: Congestion window and mode (Original algorithm)

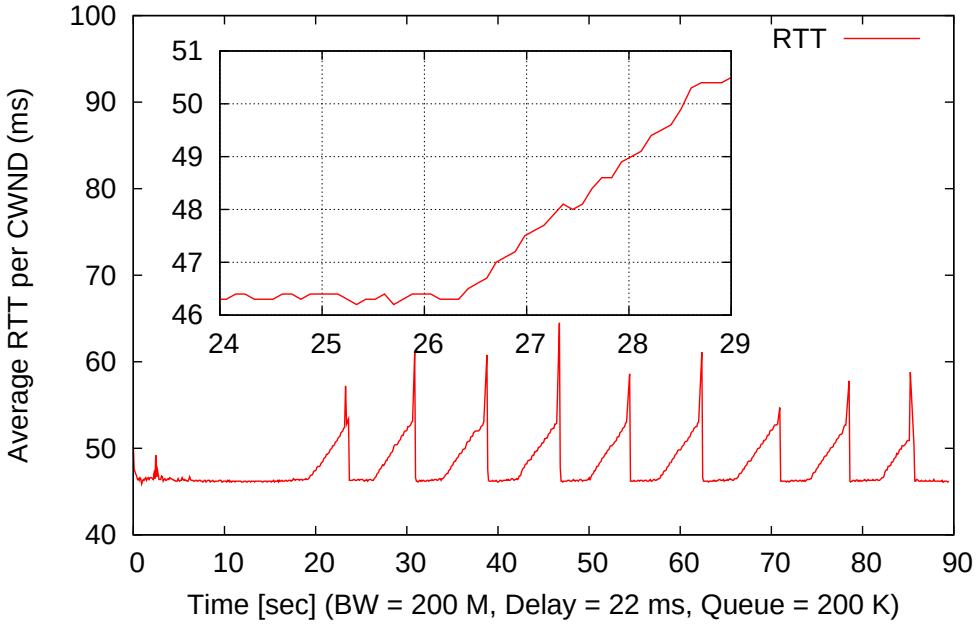


Figure 4.5: Measured RTT for validating gHSTCP

monotonously in a local period, even if the congestion window becomes large. In Figure 4.5, the average RTT values per congestion window are plotted so that the RTT trend behavior can be clarified further. The enlarged sub-figure in Figure 4.5 shows the period of 24–29 s. Together with Figure 4.4, this figure shows that the RTT fluctuates near the minimum RTT before the congestion window size reaches the BDP. When the congestion window size is larger than the BDP, on the whole, the RTT increases as the congestion window increases. However, the RTT does not always remain in the increasing state. The change of the RTT is affected by several factors, such as the performance of end host, the process schedule of the operating system and interaction with other flows.

4.2.2 Refined gHSTCP algorithm

In order to reduce the above-mentioned unnecessary mode-switching behavior, a refined algorithm for gHSTCP is required. The basic idea of the modification is that the RTT is larger than the propagation delay when the link bandwidth is fully utilized. That is, the RTT is larger than a pre-defined threshold, Reno mode should be used even when the fluctuation

of the RTT is large and there is a short-term decreasing trend. In particular, Reno mode is expected to be used at the point before the packet drop occurs, so that large amounts of simultaneously dropped packets can be avoided when the buffer overflow occurs. On the other hand, HSTCP mode is used if the RTT oscillates around the minimum RTT and the RTT is not larger than a pre-defined threshold.

In an ideal case, if RTT is larger than RTT_min, this means the bottleneck link bandwidth is fully utilized. However, the sample of RTT oscillates even when the bottleneck link is not fully utilized. As we know, the standard deviation is a measure of the degree of dispersion of the data from the mean value. It is the "expected" variation around an average. So, we consider that RTT is larger than RTT_min+RTT_std, it can be considered the bottleneck link is fully utilized. However, because of instability of RTT value, we consider that there exits a range of RTT, it is bounded by two thresholds, in which the bottleneck is in a critical state, i.e., the bottleneck may be fully utilized or not if RTT value is in this range. The lower threshold of this range is not less than RTT_min+RTT_std. For the ability of gHSTCP to catch the bottleneck link bandwidth and based on experimental results on testbed, 2*RTT_std is chosen and RTT_min+2*RTT_std is used as lower threshold. On the other hand, if RTT is far from RTT_min, the bottleneck is fully utilized. Based on experimental results on testbed, 4*RTT_std is chosen, then RTT_min+4*RTT_std is used as the upper threshold. About the exact values of the thresholds, there is no especially theoretical reason. They are heuristic values based on experimental results. However, the implicit reason is: 4 is chosen as the method of setting the time-out value of TCP. 2 is set based on the upper threshold of 4. Of course, the appropriate parameters might vary in different network environments. The auto-tuning parameters is one of our future task.

Based on this concept, the algorithm of gHSTCP is refined as follows:

```
RTT_min: minimum of the average RTT in
    one sample cycle between two loss events.
RTT_std: standard deviation of RTT in a
    sample cycle. RTT_std is used as a
    metric for evaluating the dynamic
    property of RTT.
RTT_min+2*RTT_std, RTT_min+4*RTT_std: two
    thresholds that indicate the boundaries
```

```
    in which gHSTCP is in effect.

If RTT < RTT_min + 2*RTT_std
    HSTCP mode is used.
If RTT >= RTT_min + 2*RTT_std and
    RTT < RTT_min + 4*RTT_std
    (this period is considered as an
    incredible interval)
    the mode is decided by the RTT trend.
If RTT >= RTT_min + 4*RTT_std
    Reno mode is used.
```

The following equation is used for calculation of RTT_std:

$$RTT\_std = \sqrt{\frac{n\sum rtt_i^2 - (\sum rtt_i)^2}{n(n-1)}},$$

rtt_i is the RTT value, n is the sample size. RTT_std is calculated when an ACK is received by sender. There is less demand on memory, for only three additional variables are needed. The sample size is determined by the congestion windows size in packets at the beginning of a sample cycle. Because gHSTCP takes effect when the congestion window size is larger than 38 packets, this condition is the same as that used by HSTCP, so that the sample size of RTT is always larger than 2 and standard deviation can be calculated. Here, square root function can be used, for gHSTCP is implemented in application level in this chapter[2].

Next, we check the refined gHSTCP algorithm experimentally. The experimental condition and the environment are identical to those of the previous experiment. The experimental result for the congestion window is illustrated in Figure 4.6. The TCP connection is in HSTCP mode when the congestion window size is less than the BDP. If the congestion window size is larger than the BDP, Reno mode is used. When the congestion window is around the BDP, the mode is changed according to the RTT trend. This mode-switching behavior is as expected based on the refined algorithm. In the following experiments, we use the refined algorithm for gHSTCP. gHSTCP hereafter refers to the refined gHSTCP except specified otherwise.

[2]By adding an integer square root function, standard deviation can be calculated even in kernel.

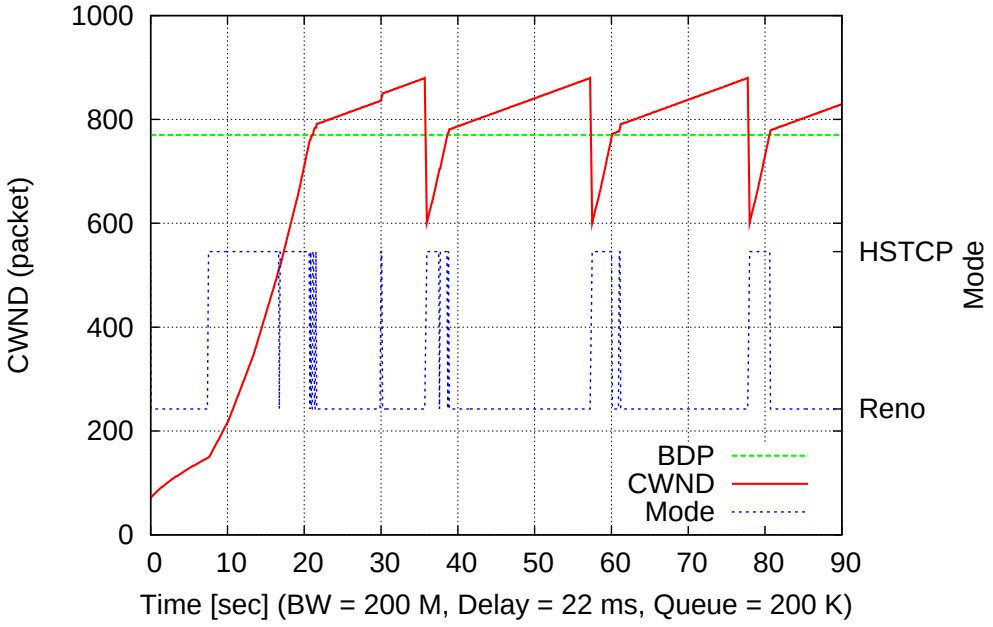


Figure 4.6: Congestion window and mode (Refined algorithm)

4.3 Performance evaluation

4.3.1 Experimental setup

In this section, we use the test-bed network to assess the behavior of high-speed TCP and parallel TCP variants. All of the following experiments use Dummynet as the infrastructure, which is included in FreeBSD 5.2.1. In the following experiments, the dumbbell topology shown in Figure 4.3 is used. In each experiment, there is one TCP flow from IPS to IPC, using TCP Reno, HSTCP, gHSTCP, and parallel TCP, respectively. There are two additional TCP Reno connections between IRS and IRC. For convenience, the TCP flow from IPS to IPC is referred to as Flow-1, and the TCP flow from IRS to IRC is referred to as Flow-2. The access link bandwidth of Flow-1 is 1 Gbps, and the access link bandwidth of Flow-2 is 100 Mbps. The link between two Ethernet switches (labeled PCI and 3Com in Figure 4.3) is referred to as the bottleneck link. The experiment run-time is 300 s.

In order that the socket buffer size does not restrict the throughput of Flow-1, the socket buffer size is set to a large value if TCP Reno/gHSTCP/HSTCP is used. When parallel

TCP is used, the system default value of 64 Kbytes is used because the main factor of parallel TCP is the number of parallel TCP connections. In our experiments, the RTT of each connection is approximately 45 ms. In this situation, the largest throughput that Flow-2 can achieve is approximately 12 Mbps, if its socket buffer size is 64 Kbytes. In this condition, the two connections in Flow-2 using socket buffer size of 64 Kbytes cannot fully utilize its access link. However, the access link can be fully utilized if the socket buffer size of Flow-2 is set to 512 Kbytes. Therefore, we present the experimental results when the socket buffer size for Flow-2 connections are set to 64 Kbytes and 512 Kbytes.

There are two scenarios designed for experiments according to differences in the Dummynet settings:

- Scenario-1: Delay = 23 ms, Bandwidth = 100 Mbps, and Buffer-size = 200 Kbytes.
- Scenario-2: Delay = 23 ms, Bandwidth = 200 Mbps, and Buffer-size = 500 Kbytes.

Each scenario contains two cases, i.e., the socket buffer size of Flow-2 is set to 64 Kbytes and 512 Kbytes. In Scenario-1, the access link bandwidth of Flow-2 is equal to the bottleneck link bandwidth. In Scenario-2, the access link bandwidth of Flow-2 is less than the bottleneck link bandwidth. Thus, the position of the bottleneck link of Flow-2 varies for different experiments. Because of the performance limitation of PCs we use, we can't construct a network with larger BDP by now. In this chapter, although BDP of testbed network is not very large, we have confirmed the fundamental behavior of gHSTCP. We believe, combined with the results that have been checked by simulation in our previous work, gHSTCP can be applied in situations with more large BDP.

4.3.2 Performance metrics

Throughput, link utilization and fairness are used as performance evaluation metrics. The throughput is the average rate of data successfully received by a TCP receiver. The link utilization is defined as the ratio of the aggregate throughput over the bottleneck link bandwidth. The fairness (Jain's fairness index) is defined as follows:

$$FairnessIndex = \frac{(\sum_{i=1}^{n} x_i)^2}{n\sum_{i=1}^{n} x_i^2}.$$

Table 4.1: Fair throughput (C_i) (Mbps)

Socket buffer size of Flow-2		64 KB	512 KB
Scenario-1 (BW = 100 Mbps)	Flow-1	76	33
	Flow-2	12, 12	33, 33
Scenario-2 (BW = 200 Mbps)	Flow-1	176	100
	Flow-2	12, 12	50, 50

Here, n is the total number of connection and x_i is the normalized throughput for flow i, defined as $x_i = M_i/C_i$, where M_i is the measured throughput and C_i is the fair throughput determined by max-min optimality. Table 4.1 shows the fair throughput determined by max-min optimality in our experiments. By this metric, we evaluate the fairness between gHSTCP/HSTCP/parallel TCP variants and TCP Reno. Note that the access links of Flow-1 and Flow-2 are different in the current evaluation model, this is formed based on this imagination: Flow-1 represents the users with high demand of link bandwidth, Flow-2 represents the common users they do not need high access link. They share the same bottleneck link. From the view of fairness of Flow-2, this condition is worse than that when they have the same access link bandwidth, e.g., 1 Gbps. Thus, the current model can show the advantages of gHSTCP.

4.3.3 Experimental results

In Scenario-1, the following four experiments are performed, where the buffer size of Flow-2 is set to 64 Kbytes or 512 Kbytes:

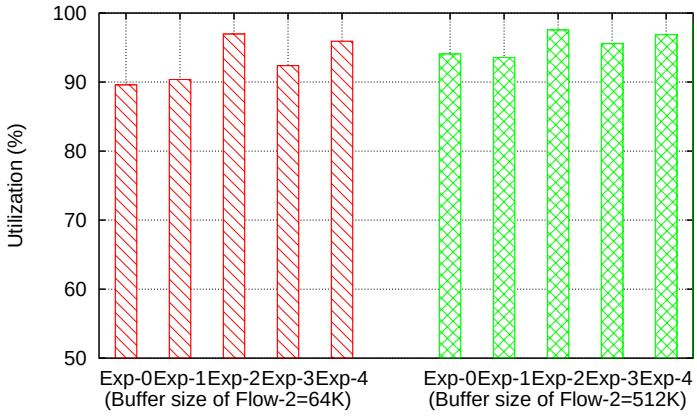
- Exp-0: Flow-1 uses TCP Reno.
- Exp-1: Flow-1 uses HSTCP.
- Exp-2: Flow-1 uses the parallel TCP mechanism.
- Exp-3: Flow-1 uses the original gHSTCP.

- Exp-4: Flow-1 uses the refined gHSTCP.

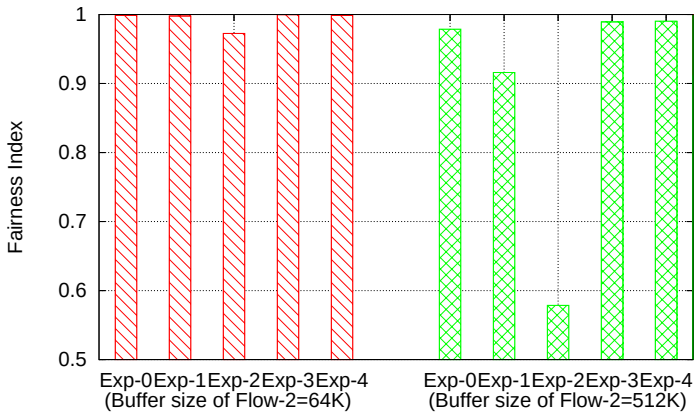
Note that when the parallel TCP mechanism is used, we use eight TCP connections in order to fully utilize the bottleneck link due to the default buffer size of 64 Kbytes. The results of link utilization, fairness index and throughput are illustrated in Figure 4.7. Note that the throughput of Flow-2 represents the total throughput of the two TCP connections in Flow-2.

Figure 4.7(a) shows that the link utilization of gHSTCP is slightly less than the largest link utilization (for parallel TCP). However, the link utilization of gHSTCP is better than that for the case in which TCP Reno or HSTCP is used for Flow-1. The utilization when HSTCP is used by Flow-1 is approximately the same as that when TCP Reno is used by Flow-1, because packet losses occur frequently when HSTCP is used. Figure 4.7(b) shows that the fairness is better in all cases when the buffer size of Flow-2 is set to 64 Kbytes. This is because the main limitation on the throughput of Flow-2 is its socket buffer size. In contrast, when the buffer size of Flow-2 is set to 512 Kbytes, the fairness is determined by the algorithms of TCP and the competing flows. When parallel TCP is used with this condition, the fairness is very poor, although the best utilization can be achieved. The fairness of parallel TCP is determined by the number of parallel TCP connections. This factor also affects its throughput. We will illustrate this problem in the following. Figure 4.7(c) intuitively shows the performance and interaction of competing flows through the throughput of Flow-1 and Flow-2 in each case. The throughput of Flow-2 is clearly influenced by the competing TCP flows when its socket buffer size is set to 512 Kbytes. This means that the fairness must be taken into consideration when a new mechanism is deployed in networks. In addition, it can be observed that the refined gHSTCP outperforms the original gHSTCP in terms of utilization and fairness (see the results of Exp-3 and Exp-4 shown in Figures 4.7(a) and 4.7(b)).

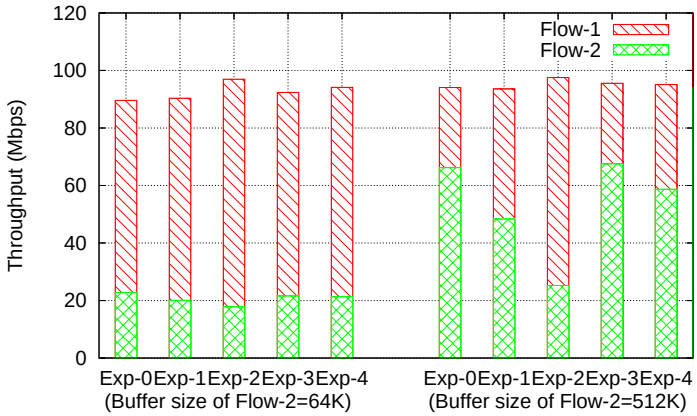
To summarize, gHSTCP offers the best tradeoff in terms of utilization and fairness due to its graceful behavior. Before the link bandwidth of the bottleneck is fully utilized, gHSTCP increases its congestion window size as rapidly as HSTCP. Therefore, it can achieve higher utilization. When the link bandwidth of the bottleneck is fully utilized, gHSTCP increases its congestion window size in the manner of TCP Reno. Therefore, gHSTCP can



(a) Utilization



(b) Fairness



(c) Throughput

Figure 4.7: Results in Scenario-1 (Bandwidth = 100 Mbps)

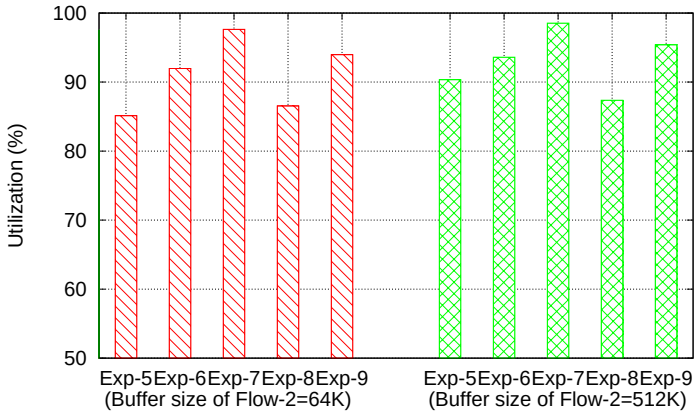
maintain better fairness while sharing the bottleneck bandwidth with the competing TCP Reno.

In Scenario-2, four experiments are conducted, in which similar to Scenario-1, the buffer size of Flow-2 is set to either 64 Kbytes or 512 Kbytes, respectively. The difference between Scenario-1 and Scenario-2 is that the bandwidth of the bottleneck link is set to 200 Mbps and the buffer size of the router is 500 Kbytes.

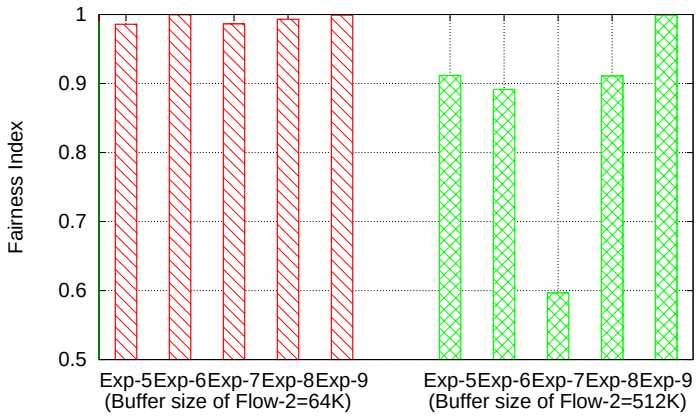
- Exp-5: TCP Reno is used by Flow-1.
- Exp-6: HSTCP is used by Flow-1.
- Exp-7: Parallel TCP mechanism is used by Flow-1.
- Exp-8: The original gHSTCP is used by Flow-1.
- Exp-9: The refined gHSTCP is used by Flow-1.

As discussed in Scenario-1, when the parallel TCP mechanism is used, we use 16 TCP connections in order to fully utilize the bottleneck link due to the default buffer size of 64 Kbytes. The results of utilization, fairness index and throughput are shown in Figure 4.8.

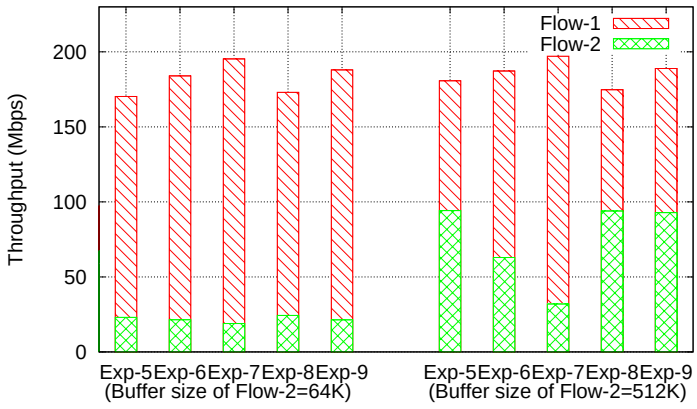
On the whole, the utilization and fairness trends are the same as those demonstrated in Scenario-1. Parallel TCP achieves the best utilization, but the worst fairness. gHSTCP offers higher utilization and better fairness than the other protocols. That is, gHSTCP is the best tradeoff in terms of link utilization and fairness. On the other hand, differences between the two scenarios remain because the link bandwidth of the bottleneck is changed from 100 Mbps to 200 Mbps. First, when TCP Reno is used, the utilization decreases as the link bandwidth increases. This illustrates the well-known problem of TCP Reno in LFNs. TCP Reno cannot fully utilize the network, due to the characteristics of conservative increase and dramatic decrease. Second, the access link of Flow-2 is equal to the bottleneck link bandwidth in Scenario-1 (Figure 4.7). In this case, the access link bandwidth is not the bottleneck for Flow-2. Thus, any increase in cross traffic will affect the throughput of Flow-2 when the buffer size of Flow-2 is set to 512 Kbytes. However, Figure 4.7(c) shows that gHSTCP steals resources from Flow-2, as compared with HSTCP and parallel TCP. In Scenario-2, the bottleneck link bandwidth is larger than the access link bandwidth



(a) Utilization



(b) Fairness



(c) Throughput

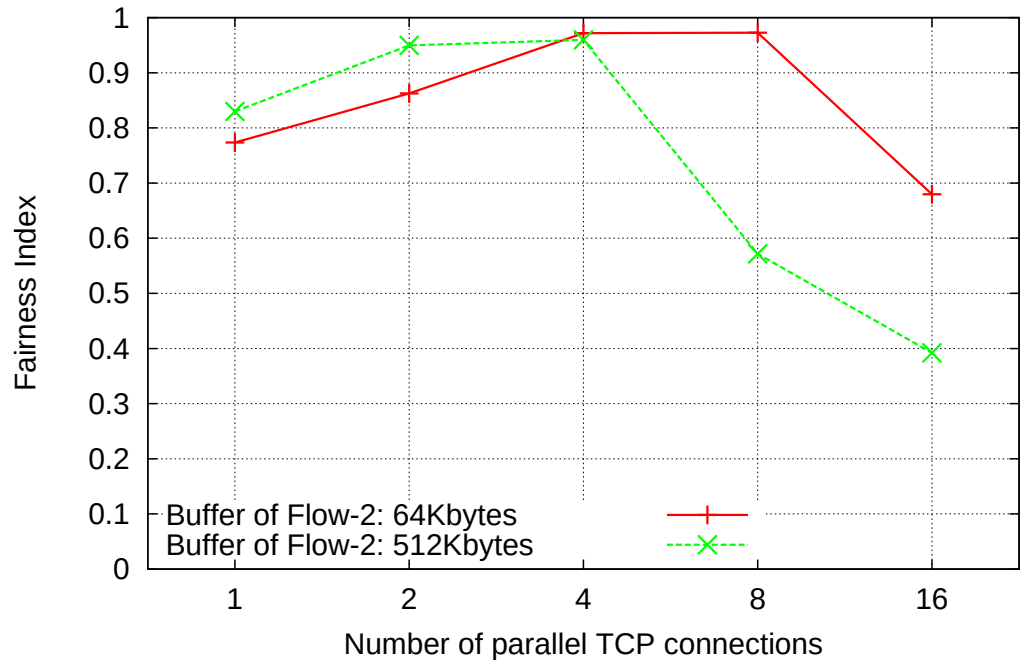
Figure 4.8: Results in Scenario-2 (Bandwidth = 200 Mbps)

of Flow-2. Therefore, redundant link bandwidth exists that can be used by other flows. As illustrated in Figure 4.8(c), gHSTCP can use the redundant link bandwidth very well when the buffer size of Flow-2 is set to 512 Kbytes. In this situation, HSTCP pillages vast resources from TCP Reno because of the aggressive increase of its congestion window size. By the comparison of results of Exp-8 and Exp-9, it is observed that the refined gHSTCP achieves higher throughput than that the original gHSTCP does (see Figure 4.8(a)), while the better fairness is obtained by the refined gHSTCP (see Figure 4.8(b)).

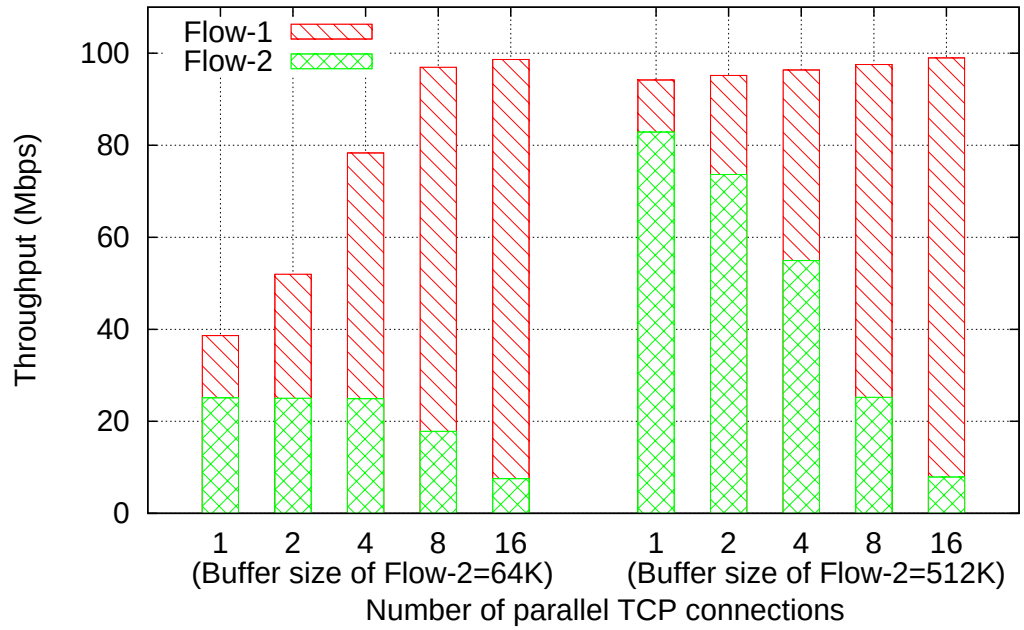
The results of both Scenario-1 and Scenario-2 show that parallel TCP outperforms gHSTCP in terms of link utilization. However, this advantage is at the expense of fairness with respect to Flow-2. There exists an important parameter when parallel TCP is used, i.e., the number of parallel TCP connections, and it is quite difficult to choose a suitable value. That is, the bottleneck link bandwidth cannot be utilized well if the number of parallel TCP connections is small. In contrast, if the number of parallel TCP connections is too large, severe unfairness results with respect to the competing flows. Figures 4.9 and 4.10 show the experimental results when the number of parallel TCP connections varies and the bottleneck bandwidth is set to 100 and 200 Mbps, respectively. These results show that it is hard to find an appropriate number of parallel TCP connections for different environments. For example, four is the best number of parallel TCP connections in terms of utilization and fairness when the bottleneck is set to 100 Mbps and the buffer size of Flow-2 is set to 512 Kbytes. However, this number is not good if the bottleneck link bandwidth is changed to 200 Mbps or the buffer size of Flow-2 is set to 64 Kbytes.

These experimental results show that the throughput of parallel TCP is increased as the number of parallel TCP connections is increased. In the literature, in the following cases, the performance has been shown to decline as the number of parallel TCP connections increases:

- When the load for using parallel TCP overcomes the load that the end-hosts can support [46].
- When the sending host is much faster than the receiving host [47].
- When the load on the data path is different [48].

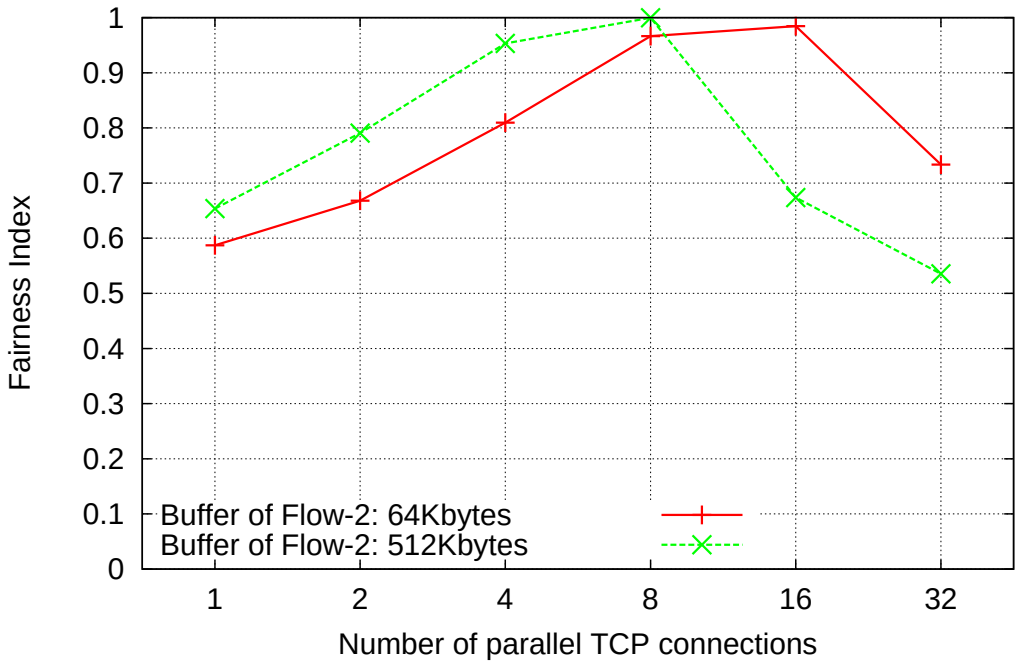


(a) Fairness

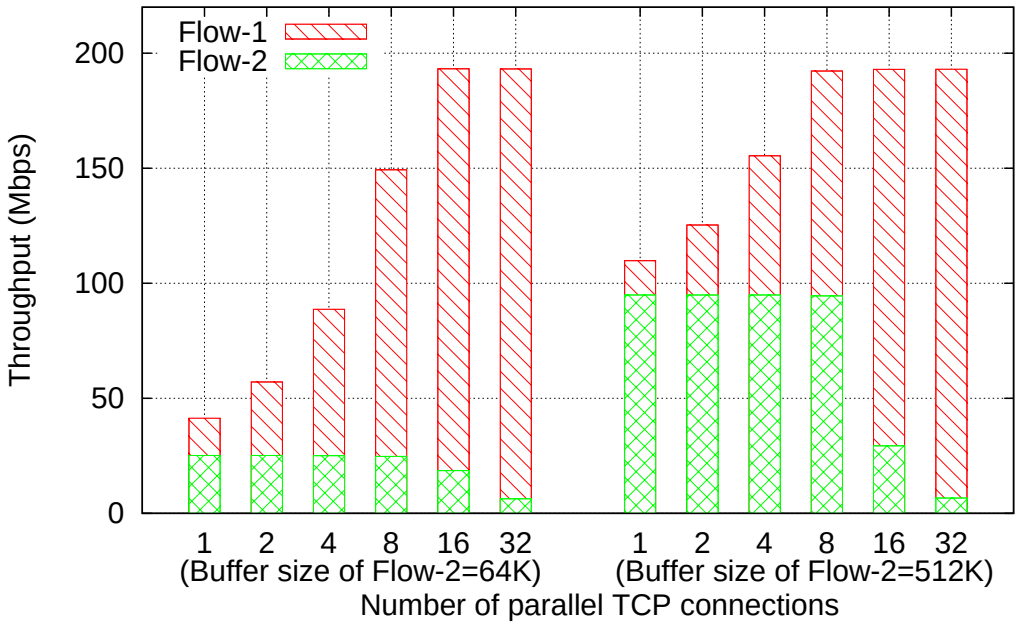


(b) Throughput

Figure 4.9: Performance of parallel TCP (Bandwidth = 100 Mbps)



(a) Fairness



(b) Throughput

Figure 4.10: Performance of parallel TCP (Bandwidth = 200 Mbps)

In other words, parallel TCP should be used in a dedicated network, and the use of parallel TCP in public networks is not trivial.

4.4 Summary

In this chapter, we performed an experimental study to assess the performance of high-speed TCP in terms of utilization, throughput and fairness. Based on these experiments, a refined gHSTCP algorithm was proposed for its application in a real network. The results indicate that gHSTCP can offer a better tradeoff between utilization and fairness on LFNs.

Chapter 5

Analysis of parallel TCP

5.1 Introduction

High-speed protocols have the capability of utilizing LFNs and their performance have been evaluated by simulations and experiments [49–51]. However, prior to these high-speed protocols, parallel TCP is proposed as one method to deal with the problem of TCP in LFNs, and employed by some applications, e.g., BBCP [12] and GridFTP [13]. In parallel TCP, multiple TCP connections are utilized between two endhosts for one data transmission task instead of using one TCP connection. The implementation of parallel TCP is relative simple compared with the TCP modification mentioned above, because it can be realized in the application layer.

Although increasing throughput is the primary purpose of parallel TCP, fairness of parallel TCP should be taken into account when it traverses the public network, i.e., when parallel TCP improves the data transmission throughput, it should preserve resources for competing TCP flows, and some works have been done [52, 53]. The results in our previous work [52] show that parallel TCP can increase the throughput at the expense of fairness against competing flows. H. Hacker et al [53] discuss this issue and propose a solution which uses a long "virtual round trip time" in combination with parallel TCP to prioritize fairness at the expense of effectiveness when the network is fully utilized.

However, this chapter emphasizes throughput while it does not give prominence to fairness. That is, we focus on the issue whether parallel TCP can really achieve high

throughput even when fairness is not taken into account, especially for the purpose of the comparison with high-speed protocols, such as HSTCP. The results in past researches [46, 54–56] show that parallel TCP can improve aggregate TCP throughput, but they either check the performance of parallel TCP by experiments, or address a lossy and un-congested path. For example, Psockets [46] introduces an application-level library that creates many TCP connections to increase data transfer throughput. In [55], the authors suggest some practical guidelines for the use of parallel sockets on a lossy wide-area network. However, there is no analytical results to answer the issue whether parallel TCP can be easily utilized to obtain high throughput. In addition, "global synchronization" issue, which impacts the aggregate throughput of parallel TCP quite considerably [26, 29], is not investigated as well. Synchronization means that TCP connections sharing the same network path reduce their congestion windows at the same time when packet losses occur. Because parallel TCP uses many TCP connections which pass through the same network link and have the same RTT, these TCP connections are easily synchronized. The problem of "global synchronization" must be investigated in evaluating parallel TCP.

In this chapter, the performance of parallel TCP is evaluated by mathematical analysis. In the analysis when DropTail is deployed, not only is the number of TCP connections taken into account, but "global synchronization" is also investigated. When the impact of "global synchronization" is considered, two extreme cases, synchronization case and non-synchronization case, are evaluated. In the synchronization case, all TCP connections are synchronized, and the throughput of this case is regarded as the lower limit. In the non-synchronization case, TCP connections are not synchronized at all, and this case gives the upper limit. The results show the hardness of using parallel TCP in practice. Even in the non-synchronization case which benefits the throughput of parallel TCP, the results show that choosing the number of TCP connections also depends on network conditions. The performance of parallel TCP is also evaluated when a Random Early Detection (RED) gateway [18] is deployed, and the results reveal that the difficulty of parameter setting in RED [31, 32, 35, 57] remains unchanged in parallel TCP case.

The remainder of this chapter is organized as follows. At first, we briefly review the background of parallel TCP. Then, the performance of parallel TCP is analyzed when DropTail is deployed. The performance of parallel TCP in the case of RED deployed is evaluated

in Section 5.4. The supplemental discussion is given in Section 5.5. Finally, the conclusion is given.

5.2 Parallel TCP mechanism

High-speed protocols modify the TCP's congestion control mechanism for use with large congestion windows in LFNs. They get rid of the constraints of the AIMD algorithm in current standard TCP (TCP Reno), that is, they adjust the increasing/decreasing step towards congestion window according to network conditions.

However, parallel TCP addresses the problem of TCP using a different approach from high-speed protocols. It uses many concurrent TCP connections for one transmission task. Its mechanism can be viewed from different points. When parallel TCP mechanism is used for bulk data transfer, the data file is divided into a number of small chunks, and each chunk is transmitted by one TCP connection. Since each of TCP connections uses the AIMD algorithm, the aggregate of congestion window is increased by N (N: the number of TCP connections) packets per RTT when there is no packet loss. So it can be considered that parallel TCP uses a larger additive increasing parameter for congestion window than that used by one normal TCP connection. The aggregate behavior of parallel TCP can also be regarded as one TCP connection with a large "virtual maximum segment size" [56], i.e., parallel TCP increases its congestion window size by one segment of $N * MSS$ Bytes (MSS: maximum segment size) when an ACK is received. From the view of network, the link bandwidth is shared by concurrent TCP connections of parallel TCP. Intuitively, each TCP connection passes a "stripped" network link [46]. "stripped" network link can be considered as a "tight" network link, but it has a smaller BDP value. In other words, compared with the case of only one TCP connection, the bandwidth-delay product (BDP) becomes small for each TCP connection. Thus, each TCP connection needs less time to recover its congestion window for utilizing "stripped" link after packet loss occurs. So that parallel TCP can boost the throughput of TCP in LFNs.

Generally, the throughput of parallel TCP is increased as the number of parallel TCP connections is increased [54]. However, the overhead of end-hosts, e.g., partitioning data file into chunks on sender, reassembling chunks of data file on receiver and handling a lot of

TCP connections, is also increased. Consequently, using twice the number of parallel TCP connections does not necessarily mean doubling the performance. The throughput declines if the number of parallel TCP connections is larger than a certain value [58]. However, the overhead on end-hosts of using parallel TCP is out of the scope of this chapter. So it is assumed that end-hosts have "unlimited" power in this chapter, i.e., the bottleneck is not the end-hosts but link bandwidth.

In this chapter, we intend to appraise the impact on parallel TCP, which comes from network, besides the number of TCP connections. As it is known, packet loss is one impact factor coming from network, and it is a main cause that affects the performance of TCP. There are two kinds of packet losses in the data transmission over the Internet – systemic packet loss and congestion packet loss. Systemic loss, such as bit error, is not related to network congestion. If there is only systemic packet loss in the network and the network link bandwidth is not sufficiently utilized, parallel TCP increases the aggregate throughput over using a single TCP stream and does not steal the bandwidth from competing flows [56]. Nowadays, the systemic packet loss is a relatively rare event because most of long-haul network links are equipped with fiber cables, e.g., the receiver of 10 Gbps shall operate with a bit error rate (BER) of better than 10^{-12} [59]. Therefore, it is assumed that the packet loss is just because of network congestion. To a turn, the network congestion is because there are excessive TCP connections.

In short, the performance of parallel TCP is evaluated in this chapter based on considering the number of TCP connections, the mechanism deployed at routers, and whether TCP connections are synchronized or not. Among these components, whether synchronization occurs or not is related to the other components. "global synchronization" affects the TCP performance severely. In the case of DropTail deployed, synchronization is common when the number of concurrent TCP flows is under 100. It is very rare if the concurrent TCP flows is above 500 [42]. But, even when the number of TCP connections is large, synchronization may appear if the sum of BDP and buffer size of router in packets is larger than $3 \times N$ (N is the number of TCP connections) [26]. To get rid of synchronization, it is needed to add random processing time [26, 29] or change queue management mechanism, e.g., using Random Early Detection (RED) gateway [18].

In the synchronization case, all connections suffer from packet losses when packet

drops occur, therefore the sum of congestion window is decreased by $1/2$. The under-utilization is one obvious problem in this case when the buffer size of router is less than BDP [42]. In the non-synchronization case, each of TCP connections suffers from packet drop at different time. Thus, congestion window sizes of TCP connections are different from each other. The underlying expectation is a single packet drop event most probably causes one connection with the largest congestion window to halve, since this TCP connection has more packets in flight in the network, and is therefore the most likely one to be impacted by a packet drop event. Then this connection uses congestion avoidance algorithm to recover its congestion window after its congestion windows is halved. Thus, if the aggregate congestion window size of parallel TCP is W_{sum} when the router buffer overflows, the congestion window reduction is no longer $W_{sum}/2$ at any time once packet drop occurs, but much smaller. Thus, the throughput in synchronization case is looked at as the lower limit that parallel TCP can obtain, and the throughput in non-synchronization case is considered as the upper limit.

5.3 Analysis with DropTail router

It is impossible to get a uniform expression that can be used to evaluate the performance of parallel TCP in any cases. As mentioned above, its performance is influenced by two main factors, the number of parallel TCP connections and synchronization or not. When the number of parallel TCP connections has been determined, its performance varies according the degree of synchronization, that is, whether all of parallel TCP connections are synchronized or part of them are synchronized. Two extreme cases, synchronization and non-synchronization cases, are analyzed in this section, and the results are regarded as the lower and upper limits of its throughput.

5.3.1 Network topology and metrics

A dumbbell topology, shown in Figure 5.1, is used in the analyzes. R1 and R2 are two routers with buffer size of B packets, DropTail management is deployed (RED is deployed

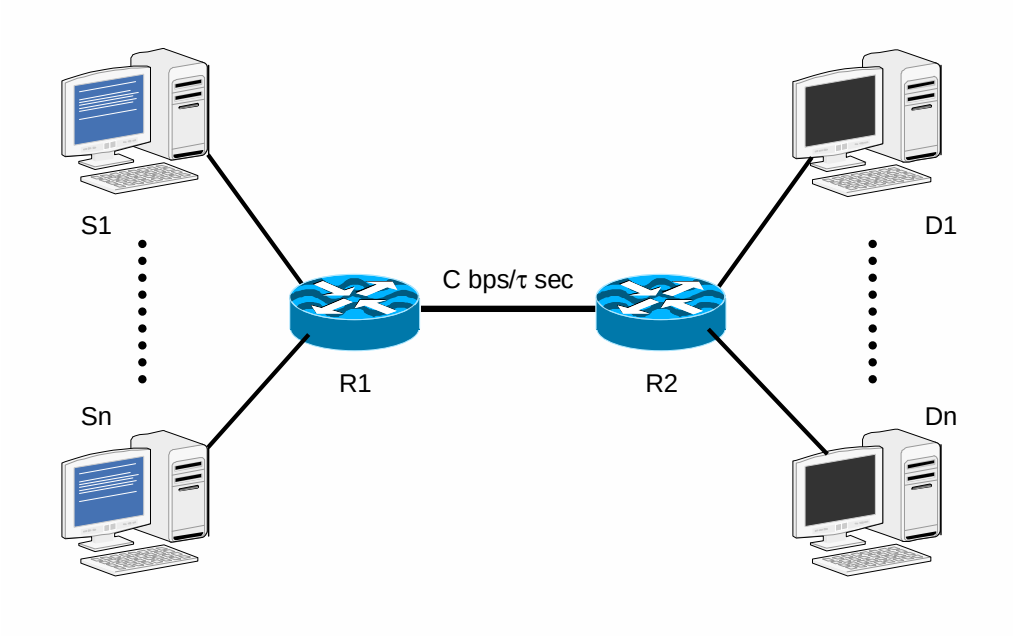


Figure 5.1: Network topology for analysis

for analysis in next section). The link bandwidth between the routers is C bps, the propagation delay is τ sec. The value of BDP is D ($D = RTT_{min} \times C$). There are N TCP connections with the same access link bandwidth and propagation delay competing for a fixed bottleneck link. They use the same AIMD algorithm as TCP Reno. The access link bandwidth of each connection is larger than C bps. The propagation delay of access link is very small, and therefore the minimum RTT (RTT_{min}) approximates 2τ sec.

We focus on the aggregate behavior of parallel TCP. Packet drop rate p and $goodput$ are used as metrics. Here, goodput is the amount of data received by the receiver in unit time. Here, goodput is the amount of data received by the receiver in unit time, and is not the same as useful throughput, for duplicated packets may be received. It is calculated as:

$$goodput = throughput \times (1 - p) \tag{5.1}$$

5.3.2 Synchronization case

Under synchronization, we assume each of the parallel TCP connections fairly shares the bottleneck link, the buffer of the routers, and their behaviors are identical. So the aggregate behavior of parallel TCP with N connections can be considered as follows. In response to a single acknowledgment, parallel TCP increases the number of segments in its congestion window as:

$$cwnd \leftarrow cwnd + \frac{a(cwnd)}{cwnd},$$

$cwnd$ denotes the aggregate congestion window of N TCP connections. In response to a congestion event, it decreases the number of segments in its congestion window as:

$$cwnd \leftarrow (1 - b(cwnd)) \times cwnd.$$

Here, $a(cwnd) = N$, and $b(cwnd) = 1/2$. Figure 5.2 shows the sketch of congestion window. The total packets transmitted in 1-cycle (N_{pkts}) is the sum of congestion window increasing from $(B+D)/2$ to $B+D$:

$$N_{pkts} = \frac{3(B+D)(B+D+2N)}{8N} \tag{5.2}$$

When packet loss occurs, each connection suffers from packet drop, i.e, there are N packets dropped in 1-cycle. So the packet loss rate is:

$$p = N/N_{pkts} = \frac{8N^2}{3(B+D)(B+D+2N)} \tag{5.3}$$

The time of 1-cycle is $t1+t2$, as shown in Figure 5.2. $t1$ and $t2$ are:

$$\begin{aligned} t1 &= (\frac{D-B+2N}{2N})RTT, \\ t2 &= (\frac{2DB+NB+B^2}{2DN})RTT. \end{aligned}$$

Thus, the throughput can be calculated as:

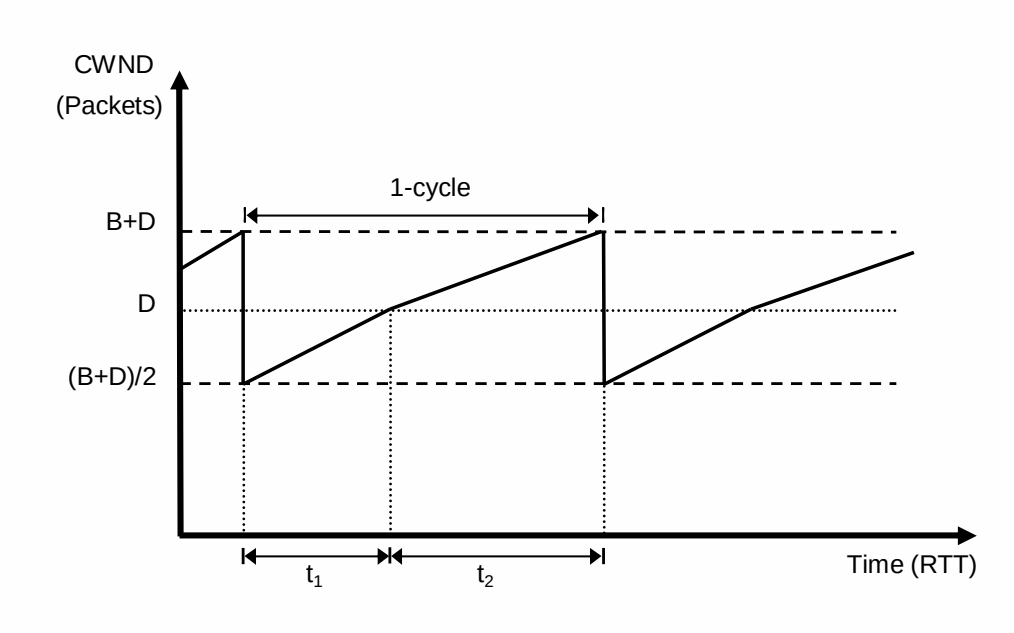


Figure 5.2: Changes of congestion window in steady state

$$\begin{aligned} throughput &= \frac{N_{pkts}}{t1+t2} \\ &= \frac{1}{RTT}\frac{D}{4}\frac{3(B+D)(B+D+2N)}{B^2+(N+D)B+2ND+D^2} \end{aligned} \tag{5.4}$$

So far, we have not considered the effect of time-out, which can occur reasonably. In the above derivation, the packet loss only leads to the current window size being halved. Upon a timeout, the congestion window is set to 1 packet, then the lost packet is retransmitted. Now, we use p_{to} to denote the probability that a packet loss results in time-out. $E(t)$ is the mean time, and $E(n)$ is the mean amount of packet sent in the period of time-out. We can write the throughput as:

$$throughput = \frac{N_{pkts}+N\cdot p_{to}\cdot E(n)}{t1+t2+p_{to}\cdot E(t)} \tag{5.5}$$

The quantities of q, $E(n)$, and $E(t)$ are determined by the follows equations [60]:

$$p_{to} = min\Big(1, \frac{(1-(1-p)^3)(1+(1-p)^3(1-(1-p)^{w-3}))}{1-(1-p)^w}\Big),$$

$$E(n) = \frac{1}{1-p},$$

$$E(t) = T_0 \frac{1+p+2p^2+4p^3+8p^4+16p^5+32p^6}{1-p},$$

w is the congestion window size of each TCP connection when packet loss occurs, here $w = (B+D)/N$. T_0 is the time of time-out. Assume that the congestion avoidance phase plays major role in a TCP connection used for a bulk transfer, we can use Equation (5.3) to calculate p.

By now, the impact of congestion window limitation is not considered. Generally, there is limit on TCP socket buffer size, e.g., the default value of some OS is 64 KBytes. TCP sender uses this buffer to keep a copy of all unacknowledged packets. Likewise, TCP receiver uses the advertised window to limit the amount of data it can receive. We assume that the limitation on the congestion window size, the maximum of congestion window size, is W_{max}. If the number of parallel TCP connections is less than a certain value, there is no congestion on bottleneck link. Therefore, the packet loss rate and throughput are determined by the different equations according to the aggregate value of congestion window:

$$p = \begin{cases} 0 & \text{if } N \times W_{max} \leq B+D \\ \dfrac{8N^2}{3(B+D)(B+D+2N)} & \text{if } N \times W_{max} > B+D \end{cases} \tag{5.6}$$

$$throughput = \begin{cases} N \times \dfrac{W_{max}}{RTT} & \text{if } N \times W_{max} < D \\ BW & \text{if } D \leq N \times W_{max} \leq B+D \\ \dfrac{N_{pkts} + N \cdot p_{to} \cdot E(n)}{t1+t2+p_{to} \cdot E(t)} & \text{if } N \times W_{max} > B+D \end{cases} \tag{5.7}$$

5.3.3 Non-synchronization case

When there are many TCP connections sharing a bottleneck link, each TCP connection obeys the AIMD algorithm. Its throughput can be calculated according to *square root p* formula [60] if the packet drop rate is known. From the view of all TCP connections, the distribution of aggregate window size is a normal distribution [42] based on the central limit theorem if TCP connections are not synchronized. The packet drop rate can be obtained from this distribution.

For each TCP connection, the model proposed in [60] is best-known and widely used. It captures the essence of TCP's congestion avoidance behavior by taking time-out into account. According to this model, TCP throughput can be estimated as:

$$b(p) \approx \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)} \tag{5.8}$$

where RTT is the average round trip time, T_0 is the timeout, b is the number of packets that are acknowledged by a received ACK, p is the packet loss rate.

In steady-state, the aggregate of congestion window size converges to a normal distribution based on the central limit theorem [42]:

$$W(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.9}$$

Here, W is the aggregate of congestion window size. μ is the mean of the aggregate congestion window size, σ is its standard deviation.

$$\mu = N \times b(p) \times RTT \tag{5.10}$$

$$\sigma = \sqrt{N}\frac{1}{3\sqrt{3}} b(p) \times RTT \tag{5.11}$$

From this distribution, we can get the packet drop rate p and RTT:

$$p = 1 - \frac{1}{2}(1 + erf\frac{B+D-\mu}{\sigma\sqrt{2}}) \tag{5.12}$$

$$RTT = RTT_{min} + \frac{\mu - D}{BW} \tag{5.13}$$

Based on Equations (5.8) – (5.13), the performance of parallel TCP can be evaluated by a fix point method.

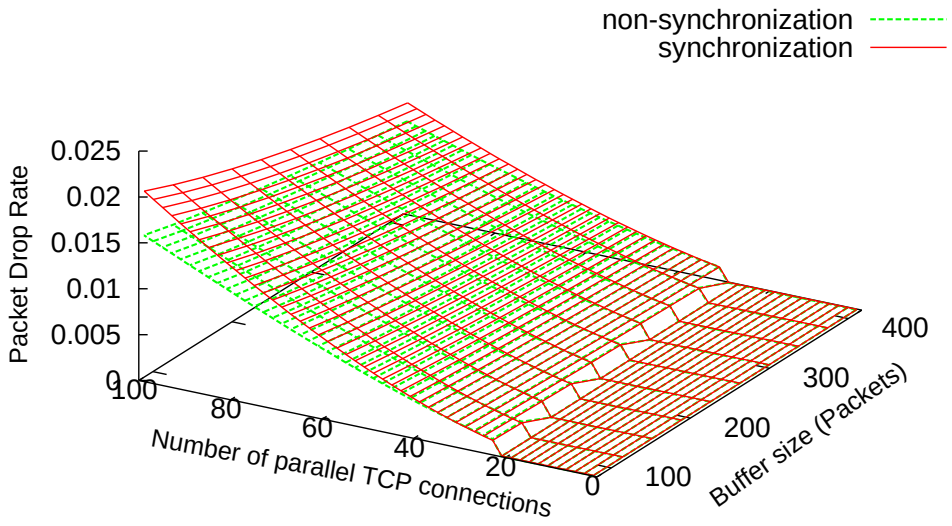
5.3.4 Numerical results and discussion

In this subsection, the performance of parallel TCP is shown visually by numerical results. We consider N connections that compete for a bottleneck link. The topology is illustrated in Figure 5.1. The parameters are set as follows:

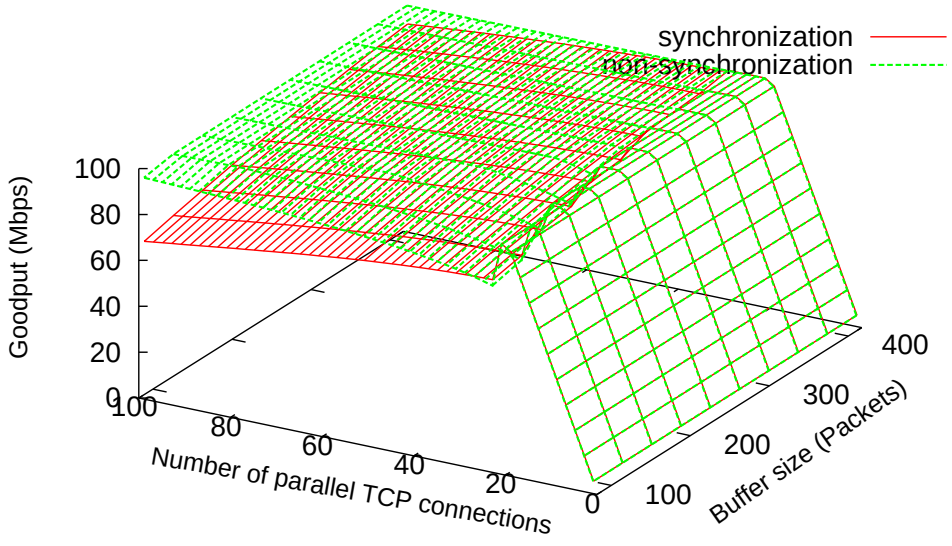
```
Example-1:
  Bandwidth = 100 Mbps/1 Gbps/10 Gbps, RTT = 100 ms
  Packet size = 1,500 Bytes, T0 = 5*RTT,
  Buffer size = (0.1--0.5)BDP, Wmax = 64 KBytes.
```

Note that the maximum value of router buffer size is set to BDP/2, because building a router with a buffer size of BDP is very difficult as the link bandwidth is increased further [42]. The numerical results of packet drop rate and goodput when the bottleneck link bandwidth equals 100 Mbps, 1 Gbps, and 10 Gbps are shown in Figures 5.3–5.5, respectively.

As the throughput equation (Equation (5.7)) suggests there exist three regions based on the aggregate congestion window size W_{sum}. The first region is that W_{sum} is less than the value of BDP. Because of limitation on congestion window, the bottleneck link cannot be fully utilized if the number of TCP connections is less than a certain value. In this region, throughput and goodput are identical, because there is no congestion on the bottleneck link. They increase linearly as the number of TCP connections increases. But, the utilization is very low if there are a small number of TCP connections. The buffer size of the routers has no effect in this region, and there is no difference between synchronization and non-synchronization. In the second region, W_{sum} lies between BDP and BDP+buffersize. This region is the best one, for parallel TCP achieves its maximum throughput, and goodput equals throughput, too. However, it is hard to find the condition that fulfills W_{sum} within this region, for this condition varies sensitively with many parameters, such as W_{max}, the value of BDP and buffer size of the router. This matter is illustrated by Figures 5.3–5.5.

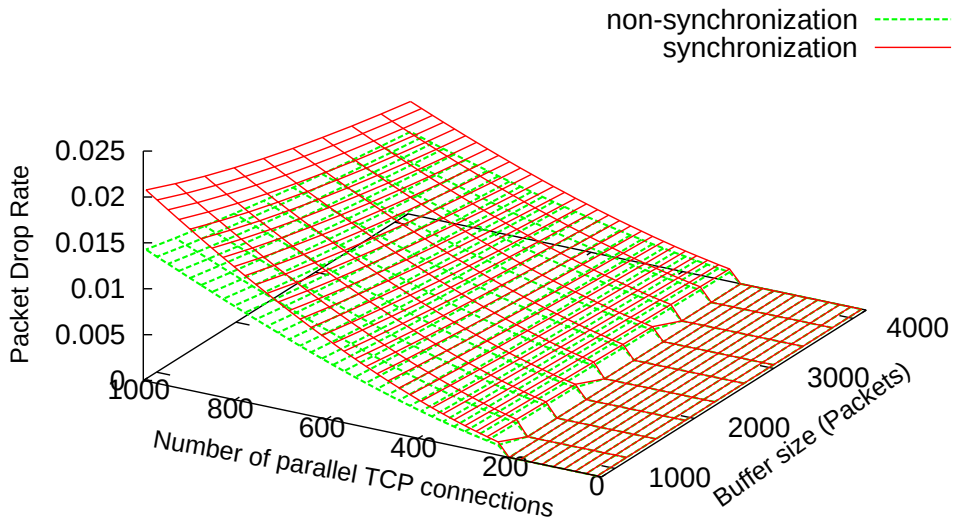


(a) Packet loss rate

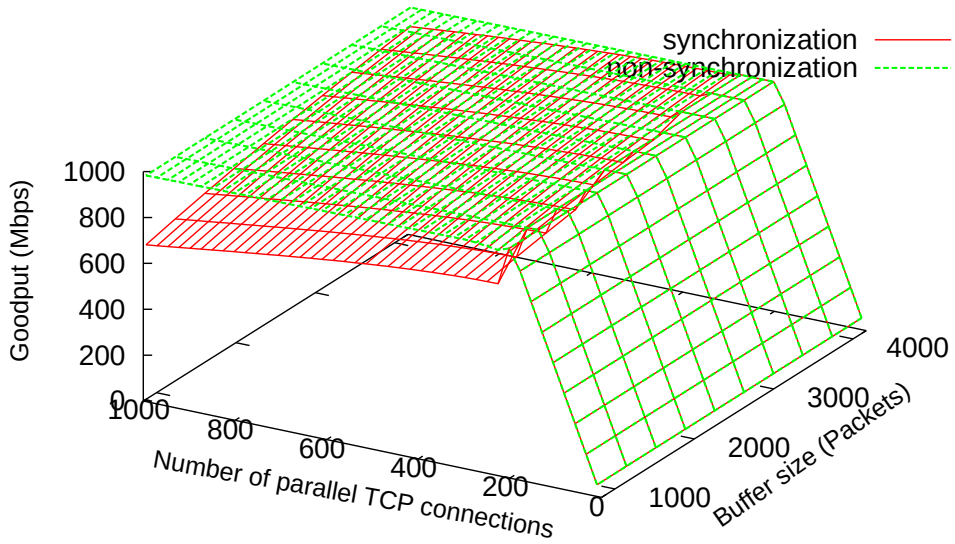


(b) Goodput

Figure 5.3: Numerical results (Bandwidth = 100 Mbps)

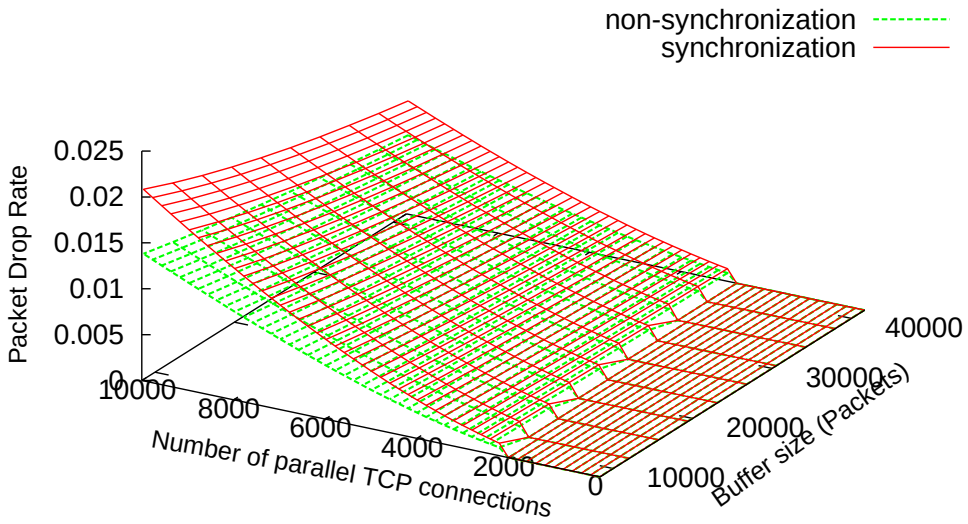


(a) Packet loss rate

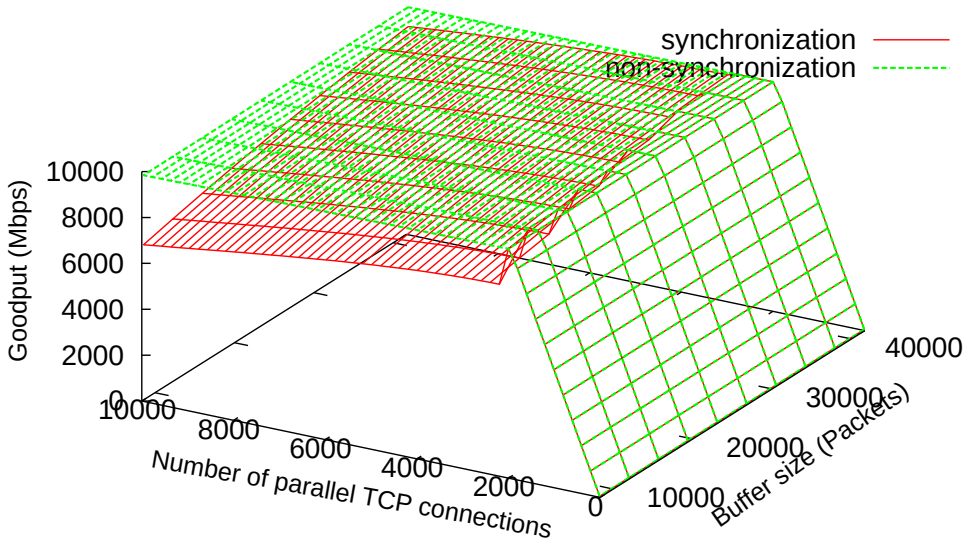


(b) Goodput

Figure 5.4: Numerical results (Bandwidth = 1 Gbps)



(a) Packet loss rate



(b) Goodput

Figure 5.5: Numerical results (Bandwidth = 10 Gbps)

Usually, the value of BDP and buffer size of the routers are unknown to the end-hosts. On the other hand, if the network link is shared by many users, the valid values of these parameters for a pair of end-hosts varies along with time. These causes make it more impossible in practice to find the optimal number of TCP connections. Of course, there are some users they maybe not expect the optimal performance when they employ parallel TCP. Their purpose is not completely consistent with the object of parallel TCP. For these users, we think their purpose of using parallel TCP should be achieving the expected throughput. They also have to face the problem of choosing the number of TCP connections as well because of dynamics of network. When W_{sum} is larger than BDP+buffersize, network congestion appears, and the throughput of parallel locates in the third region. This is because the number of TCP connections is too large. In this region, the packet drop rate becomes larger (Figures 5.3(a), 5.4(a), and 5.5(a)) and the goodput becomes smaller (Figures 5.3(b), 5.4(b), and 5.5(b)) as the number of TCP connections increases further. The difference between synchronization and non-synchronization is visible, and it becomes more noticeable for the larger number of TCP connections. In the synchronization case, the throughput is degraded badly. Even in case of non-synchronization, if the number of TCP connections is too many, the throughput is degraded. On the other hand, if the number of TCP connections is not large enough, the problem of low throughput is the same as that in the synchronization case.

Synchronization is common when DropTail is deployed, and it easily occurs if TCP connections have the same RTT [26, 29]. Parallel TCP exactly possesses the properties that induce synchronization. In the synchronization case while the router has small buffer size, the performance of parallel TCP deteriorates significantly as the number of TCP connections is increased. And it is difficult to build a router with buffer size of BDP as the link bandwidth is increased further because of limitation of the commercial memory devices used by routers [42]. Because of these reasons, much attention should be paid on synchronization.

For distinctly showing the difficulty choosing the number of TCP connections in synchronization case, the contours of the expected throughput are plotted in Figures 5.6 and 5.7. We assume that the expected throughput of parallel TCP is 95% of bottleneck link bandwidth. Note that Y-axis is the relative value of the router's buffer size, which denotes

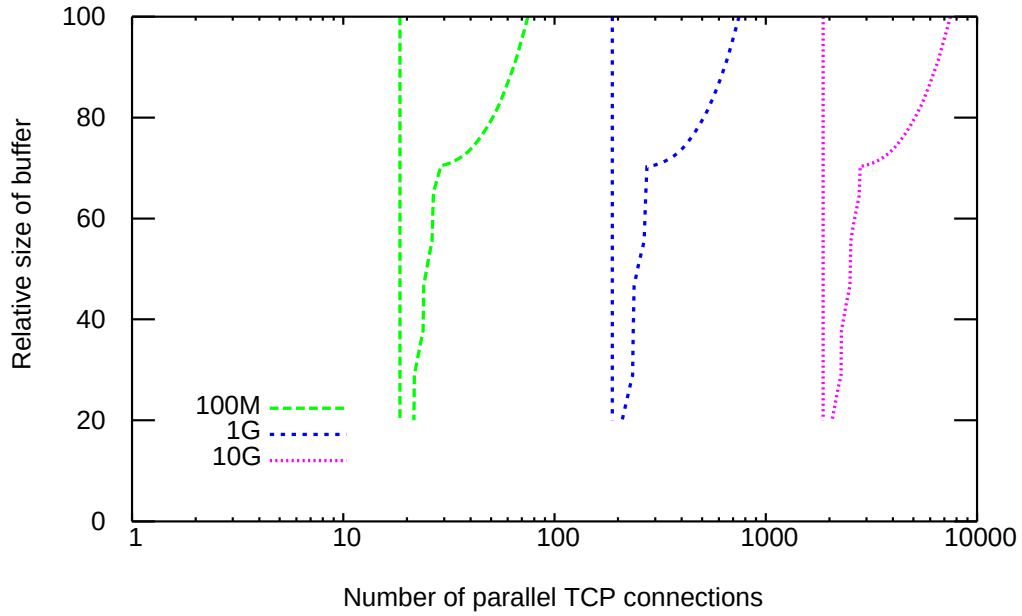


Figure 5.6: Contour of utilization = 95% (RTT = 100ms, BW = 100 M/1 G/10 Gbps)

the percentage of buffer size to BDP/2. Here, BDP/2 is used as a normalization constant because the maximum buffer size is set to BDP/2 in Example-1. Figure 5.6 shows the change of the number of TCP connections for the expected utilization with different bottleneck link bandwidth. The parameter setting is the same as that in Example-1. Figure 5.7 shows the change of the number of TCP connections with different RTTs. The parameters for this graph are: The bottleneck link bandwidth is fixed to 1 Gbps, and RTT is set to 100 ms, 200 ms, or 500 ms, respectively. Other parameters are set as that in Example-1.

In the graphs, the areas bounded with the same type lines are expected in each case. It can be observed that the positions of the areas are different in each case. That is, in order to achieve the expected throughput, the number of TCP connections must be changed with the different network conditions. In particular, the range of the number of TCP connections for the expected throughput is narrow if the buffer size of the routers is small. This makes it more difficult to find the optimal number of TCP connections. This range has significance in practice, for building a router with large buffer size is difficult as mentions above. By now, no method can be used by parallel TCP to find this area.

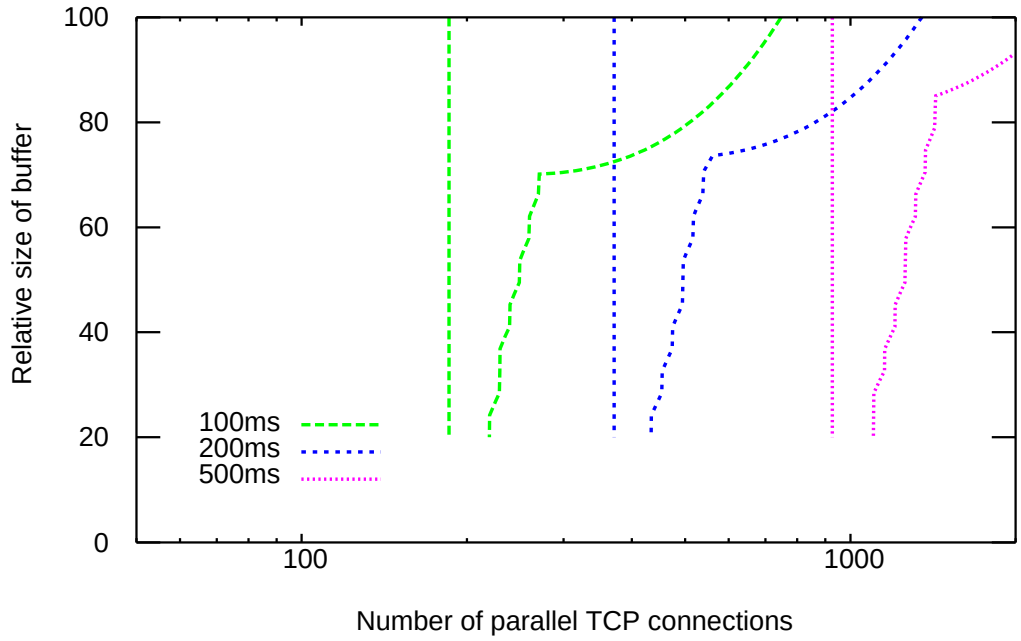


Figure 5.7: Contour of utilization = 95% (RTT = 100/100/500 ms, BW = 1 Gbps)

Although the throughput in non-synchronization case is better than that in the synchronization case, the extra mechanism is necessary. In order to break synchronization, adding random processing time is needed, or Active Queue Management (AQM) is deployed at routers [26, 29]. If the approach of adding random processing time is employed, the extra mechanism on end-hosts is necessary, and moreover, this approach increases RTT. It is contrary to the purpose of parallel TCP of alleviating the problem of TCP used in LFNs. We consider that adding random processing time is not a good method. Therefore, another method of using RED to break synchronization is discussed in the next section.

5.4 Analysis with RED router

In the previous section, we discussed the performance of parallel TCP when DropTail is deployed as queue management at the routers. When network congestion occurs, non-synchronization is expected by parallel TCP users. AQM is an alternative to break synchronization. One representative of AQM is Random Early Detection (RED) [18]. RED

detects congestion before the queue overflows and drops arriving packets probabilistically while DropTail drops packets only when the buffer is full. So that TCP connections do not suffer packet drops at the same time and synchronization is avoided.

5.4.1 Analysis based on "queue law"

In a TCP only network with one congested bottleneck link, there exists a relationship between the average queue size, packet drop probability, capacity of bottleneck link, and the parameters of TCP traffic such as number of TCP flows and the average round time. This relationship is referred to as "queue law" [34]. The network can be viewed as a feedback control system. TCP senders are controlled system. Queue management deployed at router is controlling system. The average queue size (q) is a function of packet drop probability (p) denoted by $q = G(p)$. On the other hand, the queue management of router has a feedback control function denoted by $p = H(q)$.

Suppose the round trip time of bottleneck link is RTT, and the link bandwidth is BW. If the number of TCP flows is N, the average congestion window size of TCP connections is w, the packet drop probability is p, then the average queue size is given by [61]:

$$q = G(p) = N \times w \times (1-p) - RTT \times BW \tag{5.14}$$

The throughput of each TCP connection can be estimated by Equation (5.8). Then, the congestion window size w is calculated by:

$$w = b(p) \times RTT \tag{5.15}$$

Timeout T_0 is typically $5 \times RTT$, then $G(p)$ can be expressed as:

$$\begin{aligned} G(p) \quad \approx \quad & \frac{N \times (1-p)}{\sqrt{\frac{2bp}{3}} + 5 \times min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)} \\ & - RTT \times BW \end{aligned} \tag{5.16}$$

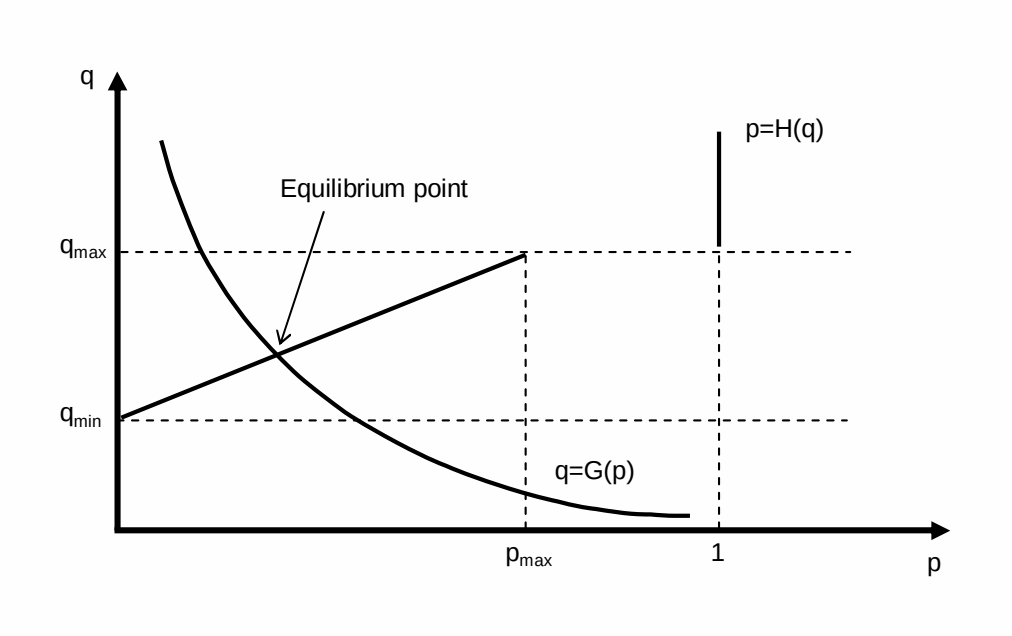


Figure 5.8: Equilibrium point of TCP-RED system

When RED is deployed, the feedback control function $H(q)$ is:

$$H(q) = \begin{cases} 0 & 0 \leq q < q_{min} \\ \dfrac{q - q_{min}}{q_{max} - q_{min}} p_{max} & q_{min} \leq q < q_{max} \\ 1 & q_{max} \leq q \leq B \end{cases} \tag{5.17}$$

Here, B is the buffer size of the router, q_{min}, q_{max} are queue length thresholds. p_{max} is the upper bound of the packet drop probability p.

Figure 5.8 depicts $G(p)$ and $H(q)$ in the case when RED is used. Within the area of $q_{min} < q < q_{max}$ and $0 < p < p_{max}$ (equilibrium area), there exists an equilibrium point and the network can work in steady-state at this point. The equilibrium point varies with the network load. Once out of this area, the network becomes instable, e.g., if queue length is larger than q_{max}, all packets are dropped. This leads to performance decline. It can be observed the equilibrium area is directly related to the setting of RED parameters, i.e., the setting of RED affects the performance of parallel TCP.

5.4.2 Numerical results and discussion

Based on Equations (5.16) and (5.17), we can use a fix point method to find the equilibrium point. The following is a numerical example. Since how to configure RED is not the objective of this chapter, we closely follow the guideline in [57] to configure RED parameters.

```
Example-2:
  Parameters of Network:
    Bandwidth = 100 Mbps/1 Gbps, RTT = 100/200/500 ms,
    Packet size = 1,500 Bytes, Wmax = 64 KBytes.
  Parameters of RED:
    q_min = 5 packets, q_max = 15 packets,
    p_max = 0.1, buffer size = BDP.
```

Because the limitation on congestion window size is considered, there exists a minimum number of TCP connections (N_{min}). N_{min} is determined by limitation of congestion window size, the value of BDP, and the minimum queue threshold. If the number of TCP connections is less than N_{min}, RED does not work, and the characteristic of throughput is the same as that when DropTail is deployed, that is, the throughput is increased linearly as the number of parallel TCP connections increases until the aggregate congestion window is larger than BDP. Then, the throughput equals to the bottleneck link capacity if aggregate of congestion window is less than BPD+q_{min}. As the number of TCP connections increases further, RED starts to work. Since there are always packets settled in buffer of the router within the equilibrium area, the goodput equals bottleneck link capacity.

Figure 5.9 shows the change of packet drop rate with different bottleneck link bandwidth and RTT when RED works. X-axis is the number of TCP connections, and Y-axis is the packet drop rate. The results of 6 cases, which are the different combinations of the bottleneck link bandwidth and RTT, are shown in this graph. On the whole, the packet drop rate becomes large as the number of TCP connections is increased. Although the goodput equals link capacity within the equilibrium area, the effective throughput, which is the effective amount of data received by the receiver and does not include the duplicated packets, may be decreased as the packet drop probability increases. Therefore, the number of TCP connections that leads to the lowest packet drop rate in each case (N_{opt}) is expected. N_{opt}

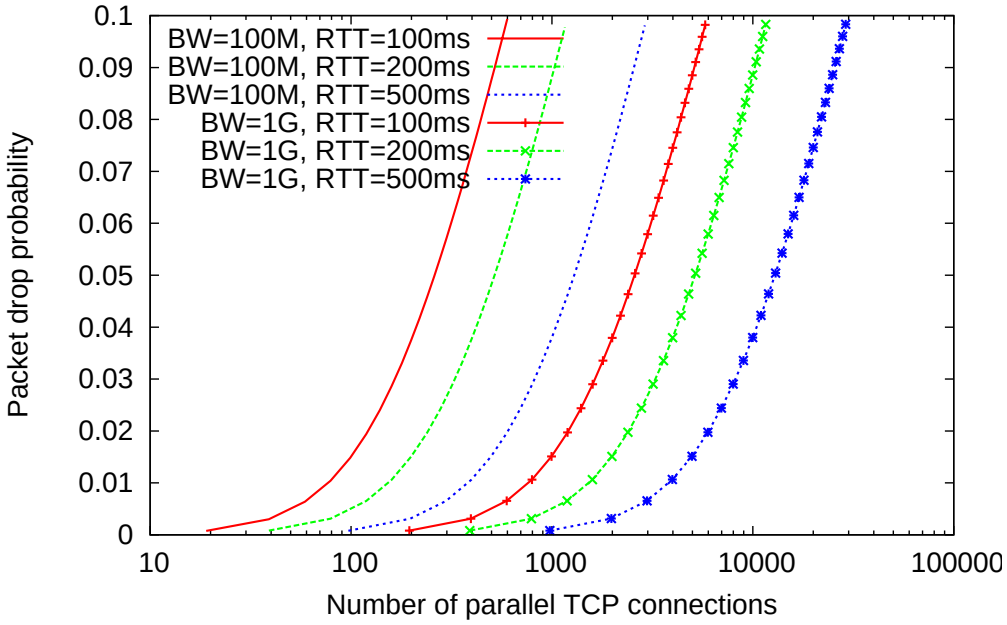


Figure 5.9: Packet loss rate (RED router)

is the optimal number of TCP connections, for not only is the network within equilibrium area, but also the packet drop rate is the smallest. From Figure 5.9, it can be observed that N_{opt} varies with the network conditions. That is, RED can not solve the problem of choosing the optimal number of TCP connections.

Addressing the difficulty in setting the parameters of RED, some improvements are proposed, such as adaptive RED (ARED) [36]. The motivation of ARED is to maintain an expected queuing delay by tuning the maximum packet drop probability. The optimal number of TCP connections is not correlative with the expected queuing delay. ARED can not solve the problem in using parallel TCP.

5.5 Trouble of "dynamic network resources allocation"

To some extent, the analyzes of this chapter show how to choose the optimal number of TCP connections for different network conditions based on the premises that the parameters of network are known by end-hosts. However, the results also show that the optimal

number is sensitive to network parameters. Once these premises do not come into existence, it is difficult to get an optimal value. About the case when RED is deployed, it is assumed that RED is deployed at the router of bottleneck link. In practice, there are many hops between two endhosts. It is usually unknown where is the bottleneck link. This means that all routers along the path must use RED mechanism. It is not an actual requirement.

In addition, the number of TCP connections is unchangeable during data transmission in the above analyzes. When the number of TCP connections (N) is determined, parallel TCP can be looked at as a high-speed protocol with an increase parameter of N packets per RTT. This is not appropriate if the variability of network conditions is taken into account, Because the increase parameter of high-speed protocol varies with network conditions. For example, the increasing parameter of HSTCP becomes larger, and decreasing parameter becomes smaller as congestion window increases. It may bring benefit to parallel TCP if the number of TCP connections is alterable during data transfer. Such mechanism has been proposed, e.g., "dynamic network resources allocation" of GridFTP v2 [62], in which an active peer can open/close one or more additional TCP connections dynamically during data transfer. However, We consider that this mechanism may lead to some problems.

- How to determine the granularity of changing the number of TCP connections. If the granularity is large, there is no effectiveness on tracing the change of link bandwidth. Otherwise, it may lead to overheads on handling TCP connections. However, the overhead concerning granularity is essentially unrelated with high speed protocols, for high-speed protocols use advanced algorithm to update congestion window.

- It is difficult to manage opening/closing TCP connections and control data channels dynamically. In order to increase the number of TCP connections and attain steady-state, a few tens RTT are necessary in each time when a new connection is created due to the effects of 3-way handshake and slow-start phase. In contrast, there is just one time for high-speed protocol during one transfer.

- This mechanism determines the number of TCP connections based on measurement of network conditions. Since parallel TCP uses many TCP connections, and the interaction among these TCP connections may affect the accuracy of measurement.

Therefore, the performance of parallel TCP may be influenced. Despite the mechanism of some high speed protocols is also based on measurement, the interaction issue can be avoided in case of using high-speed protocol because there is only one high speed TCP connection at any time.

- Because the number of TCP connections is changed dynamically, how to setup the chunk size is not easy. In addition, the chunk management is difficult in occasion of decreasing the number of TCP connections, for a TCP connection may be shut off when it is transmitting a chunk. While high-speed protocol uses only one TCP connection to transmit a data file, so the problem of "chunk" does not exist if high-speed protocol is utilized.

In a word, high-speed protocols can offer more flexibility to a dynamic network. Even when there exists such a network scenario in which the performance of parallel TCP is not very sensitive to the number of TCP connections. We believe that high-speed protocols can work finely as well, and are more efficient than parallel TCP in such an environment. Although high speed TCPs are not widely available in production OSs, e.g., Solaris and Windows, this is likely to change shortly.

5.6 Summary

In this chapter, we used mathematical analysis to explore the performance of parallel TCP, which is used as one of the solutions to eliminate the shortcomings of TCP in LFNs. Two queue management mechanisms, DropTail and RED, are examined. When DropTail is deployed, both of the number of TCP connections and "global synchronization" are investigated, and two extreme cases – synchronization and non-synchronization – are analyzed. Synchronization easily occurs because of the property of parallel TCP, and the throughput deteriorates observably. Non-synchronization may benefit the throughput of parallel TCP, but the extra mechanisms are necessary. The throughput in these two cases are analyzed, and considered as the lower and upper limits. Parallel TCP combined with RED, which is one method to break synchronization, is also evaluated, and the difficulty of parameter setting in RED remains unchanged. Although there are mechanisms that may tune the

number of TCP connections during data transfer, some potential problems remain. The analysis results show that it is difficult to use parallel TCP in practice for the sake of approving throughput. That is, parallel TCP is not really effective in LFNs. In contrast, high speed protocols have the inherent characteristics which are suitable for LFNs. Therefore, we recommend to use high-speed protocols instead of parallel TCP in LFNs.

Chapter 6

Conclusion

The story of TCP began in 1960–70s. Since then, improving performance of TCP is an eternal subject, especially as emergence of data-intensive applications and fast long-distance networks (LFNs), i.e., networks operating at 2.5 Gbit/s, or 10 Gbit/s and spanning several countries for terabytes or petabytes data transfer.

Although the network infrastructure is now in place, or will soon be, the transport-layer protocols available to date perform rather poorly over such environments. Current version of TCP (TCP Reno), using the AIMD algorithms, recover very slowly from packet loss when the RTT and the link capacity are large. The open question concerning the use of TCP is how to improve its throughput, while maintaining better fairness among competing flows.

In this thesis, we have presented our design and implementation – Gentle HighSpeed TCP (gHSTCP) for LFNs. As its name implies, gHSTCP can provide high performance, and its behavior looks like a "gentleman" for fairness. This is guaranteed by its valid mechanisms: There are two modes in congestion avoidance phase – HSTCP mode and Reno mode. HSTCP mode is employed when the link bandwidth is under-utilized. gHSTCP increases its congestion window aggressively as HSTCP does. Reno mode is deployed if the link bandwidth is fully utilized, and its congestion window is enlarged by one packet per RTT as TCP Reno does.

On the other hand, high performance can not be obtained only by improving mechanisms of end-hosts. Some mechanisms are needed in routers to complement the endpoint

congestion avoidance mechanisms. So we have proposed a modified version of adaptive RED – gentle adaptive RED (gARED). It cooperates with gHSTCP in achieving better performance in terms of throughput and fairness.

Finally, we have also investigated the performance of parallel TCP mechanism in use in LFNs. The analytical results reveal that, in practice, parallel TCP does not effectively improve throughput. Apparently high-speed protocols have an inborn capability for LFNs. Therefore, we believe that using high-speed protocols is a better choice.

The improvement upon TCP performance is a timeless topic since TCP was designed. The evolution of TCP is a careful balance between innovation and considered constraint. While innovation boosts the performance of TCP, the evolution of TCP must avoid making radical changes that may not easy to be deployed, and also must avoid a congestion control arms race among competing protocols. In this thesis, we have proposed Gentle HighSpeed TCP based on these consideration. It is an enhanced transport-layer protocol with the characteristics of simpleness, friendliness and effectiveness. As long as TCP survives, the story of TCP will not end.

Bibliography

[1] J. Postel, "Transmission control protocol," *RFC 793*, IETF, September 1981.

[2] M. Fomenkov, K. Keys, D. Moore, and k claffy, "Longitudinal study of Internet traffic from 1998-2003," in *Proc. Winter International Symposium on Information and Communication Technologies (WISICT)*, January 2004.

[3] V. Jacobson, "Congestion avoidance and control," in *Proc. SIGCOMM 1988*, pp. 314–329, August 1988.

[4] ——, "Dynamic congestion avoidance/control (long message)," *Available as: http://www-nrg.ee.lbl.gov/nrg-email.html*, February 1988.

[5] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proc. SIGCOMM 1994*, pp. 24–35, August 1994.

[6] M. Floyd, T. Henderson, and A. Gurtov, "The NewReno modification to TCP's fast recovery algorithm," *RFC 3782*, IETF, April 2004.

[7] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgement options," *RFC 2018*, IETF, October 1996.

[8] M. Allman, V. Paxson, and W. Stevens, "Performance evaluation of explicit congestion notification (ECN) in IP networks," *RFC 2581*, IETF, April 2000.

[9] S. Floyd, "HighSpeed TCP for large congestion windows," *RFC 3649*, IETF, December 2003.

[10] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," *RFC 1323*, IETF, May 1992.

[11] J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP buffer tuning," in *Proc. SIGCOMM 1998*, pp. 315–323, August 1998.

[12] A. Hanushevsky, A. Trunov, and L. Cottrell, "Peer-to-Peer computing for secure high performance data copying," in *Proc. CHEP'01*, September 2001.

[13] W. Allcock, "GridFTP: Protocol extensions to FTP for the Grid," *Available as: http://www.ggf.org/documents/GFD.20.pdf*, April 2003.

[14] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *Available as: http://www-lce.eng.cam.ac.uk/~ctk21/scalable/*, February 2003.

[15] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP for high-speed long-distance networks," *Internet Draft: draft-jwl-tcp-fast-01.txt*, June 2003.

[16] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proc. SIGCOMM 2002*, August 2002.

[17] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's initial window," *RFC 3390*, IETF, October 2002.

[18] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

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

[20] S. Floyd, "TCP and explicit congestion notification," *ACM Computer Communication Review*, vol. 24, no. 5, pp. 10–23, October 1994.

[21] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One bit more is enough," in *Proc. SIGCOMM 2005*, Auguest 2005.

[22] S. Floyd, "Congestion control principles," *RFC 2914*, IETF, September 2000.

[23] C. Barakat, E. Altman, and W. Dabbous, "On TCP performance in a heterogenous network: A survey," *IEEE Communications Magazine*, vol. 38, no. 1, pp. 40–46, January 2000.

[24] G. Hasegawa and M. Murata, "Survey on fairness issues in TCP congestion control mechanisms," *IEICE Transactions on Communications*, vol. E84-B, no. 6, pp. 1461–1472, June 2001.

[25] R. Morris, "TCP behavior with many flows," in *Proc. IEEE International Conference on Network Protocols (ICNP)*, pp. 205–211, October 1997.

[26] L. Qiu, Y. Zhang, and S. Keshav, "Understanding the performance of many TCP flows," *Computer Networks*, vol. 37, no. 3–4, pp. 277–306, November 2001.

[27] L. Guo and I. Matta, "The war between mice and elephants," in *Proc. the 9th IEEE International Conference on Network Protocols*, November 2001.

[28] K. Avrachenkov, U. Ayesta, P. Brown, and E. Nyberg, "Differentiation between short and long TCP flows: Predictability of the response time," in *Proc. INFOCOM 2004*, March 2004.

[29] S. Floyd and V. Jacobson, "Traffic phase effects in packet-switched gateways," *Journal of Internetworking: Practice and Experience*, vol. 3, no. 3, pp. 115–156, September 1992.

[30] B. Braden and et al., "Recommendations on queue management and congestion avoidance in the Internet," *RFC 2309*, IETF, April 1998.

[31] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "A self-configuring RED gateway," in *Proc. INFOCOM 1999*, pp. 1320–1328, March 1999.

[32] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons not to deploy RED," in *Proc. 7th. International Workshop on Quality of Service (IWQoS'99)*, pp. 260–262, June 1999.

[33] V. Misra, W. B. Gong, and D. F. Towsley, "A fluid-based analysis of a network of aqm routers supporting TCP flows with an application to RED," in *Proc. SIGCOMM 2000*, pp. 151–160, September 2000.

[34] V. Firoiu and M. Borden, "A study of active queue management for congestion control," in *Proc. INFOCOM 2000*, pp. 1435–1444, March 2000.

[35] M. Christiansen, K. Jaffay, D. Ott, and F. D. Smith, "Tuning RED for Web traffic," in *Proc. SIGCOMM 2000*, pp. 139–150, August 2000.

[36] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An algorithm for increasing the robustness of RED," *Available as: http://www.icir.org/floyd/red.html*, August 2001.

[37] Z. Zhang, G. Hasegawa, and M. Murata, "Performance analysis and improvement of HighSpeed TCP with TailDrop/RED routers," *IEICE Transactions on Communications*, pp. 2495–2507, June 2005.

[38] M. Goutelle and et al., "A survey of transport protocols other than standard TCP," *Available as: http://www.gridforum.org/Meetings/ggf10/GGF10G.pdf*, February 2004.

[39] S. McCanne and S. Floyd, "ns Network Simulator," *Available as: http://www.isi.edu/nsnam/ns/*, 2004.

[40] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," *Computer Communication Review*, vol. 26, no. 3, pp. 5–21, July 1996.

[41] Mathematics group, "Faculty of arts, computing, engineering and sciences scientific statistics," *Available as: http://maths.sci.shu.ac.uk/distance/stats/index.html*, 1998.

[42] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *Proc. SIGCOMM 2004*, pp. 277–288, August 2004.

[43] S. Labs, "Packet trace analysis," *Available as: http://ipmon.sprint-labs.com/packstat/packet.php*, 2004.

[44] A. Feldmann, A. Gilbert, P. Huang, and W. Willinger, "Dynamics of IP traffic: A study of the role of variability and the impact of control," in *Proc. SIGCOMM 1999*, pp. 301–313, September 1999.

[45] L. Rizzo, "Dummynet: A simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, January 1997.

[46] H. Sivakumar, S. Bailey, and R. L. Grossman, "PSockets: The case for application-level network striping for data intensive applications using high speed wide area networks," in *Proc. the IEEE/ACM SC2000*, pp. 38–43, November 2000.

[47] J. Lee, D. Gunter, B. Tierney, B. Allcock, J. Bester, J. Bresnahan, and S. Tuecke, "Applied techniques for high bandwidth data transfers across wide area networks," in *Proc. International Conference on Computing in High Energy and Nuclear Physics*, September 2001.

[48] A. Chierici, T. Ferrari, A. Forte, L. Giado, S. Lusso, and G. Tortone, "Experimental results on GridFTP," *Available as: http://server11.infn.it/netgrid/task/task2/globusftp/ftp-summary.html*, February 2001.

[49] E. de Souza and D. Agarwal, "A HighSpeed TCP study: Characteristics and deployment issues," *LBNL*, Tech. Rep. LBNL–53215, 2003.

[50] K. Kumazoe, K. Kouyama, Y. Hori, M. Tsuru, and Y. Oie, "Transport protocol for fast long-distance networks: Evaluation of their penetration and robustness on JGNII," in *Proc. SIGCOMM 2005*, Feburary 2005.

[51] R. Gupta, S. Ansari, R. L. Cottrell, and R. Hughes-Jones, "Characterization and evaluation of TCP and UDP-based transport on real networks," in *Proc. SIGCOMM 2005*, Feburary 2005.

[52] Z. Zhang, G. Hasegawa, and M. Murata, "Experimental evaluations of Gentle High-Speed TCP for Long-Fat Networks," in *Proc. 6th Asia-Pacific Symposium on Information and Telecommunication Technologies*.

[53] T. Hacker, B. Noble, and B. Athey, "Improving throughput and maintaining fairness using parallel TCP," in *Proc. IEEE INFOCOM 2004*, March 2004.

[54] R. Kalmady and B. Tierney, "A comparison of GSIFTP and RFIO on a WAN," *Available as: http://edg-wp2.web.cern.ch/edg-wp2/docs/GridFTP-rfio-report.pdf*, March 2001.

[55] T. J. Hacker, B. D. Athey, and B. Noble, "The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network," in *Proc. the 16th International Parallel and Distributed Processing Symposium*, April 2002.

[56] T. J. Hacker, B. Noble, and B. D. Athey, "The effects of systemic packet loss on aggregate TCP flows," in *Proc. SC2002: High Performance Networking and Computing*, November 2002.

[57] S. Floyd, "RED: Discussions of setting parameters," *Available as: http://www.aciri.org/floyd/REDparameters.txt*, November 1997.

[58] "Java GridFTP client - programmer guide," *Available as: http://www-unix.globus.org/cog/jftp/guide.html*.

[59] "IEEE Std 802.3ak™ – 2004," *Available as: http://standards.ieee.org/getieee802/download/802.3ak-2004.pdf*, March 2004.

[60] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. SIGCOMM'98 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 303–314, September 1998.

[61] J. Chung and M. Claypool, "Analysis of Active Queue Management," in *Proc. 2nd IEEE International Symposium on Network Computing and Applications (NCA)*, April 2003.

[62] I. Mandrichenko, W. Allcock, and T. Perelmutov, "GridFTP v2 protocol description," *Available as: http://www.ggf.org/documents/GFD.47.pdf*, May 2005.