Experimental Evaluations of Gentle HighSpeed TCP for Long-Fat Networks

Zongsheng ZHANG, Go HASEGAWA, and Masayuki MURATA
Graduate School of Information Science and Technology, Osaka University
1–32 Machikaneyama, Toyonaka, Osaka 560–0043, Japan
{zhang, hasegawa, murata}@ist.osaka-u.ac.jp

Abstract

It is well-known that TCP Reno cannot provide satisfactory performance in high-speed long-delay networks. As a means addressing this problem, gentle HighSpeed TCP (gHSTCP) has been proposed in [1]. However, its effectiveness has only been demonstrated in simulation experiments. In the present paper, 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, which uses two modes (Reno mode and HSTCP mode) in the congestion avoidance phase and switches modes based on RTT increasing trends. The refined gHSTCP algorithm compares two RTT thresholds and judges which mode will be used. The experimental results demonstrate that gHSTCP can provide a better tradeoff in terms of utilization and fairness against co-existing traditional TCP Reno connections.

1 Introduction

Transmission Control Protocol (TCP) [2] has been widely used as a transport-layer protocol in the current Internet from its inception. TCP has played a great role in the advancement of the Internet. However, the infrastructures of networks have been changing both with respect to end-hosts and network links. End-hosts are becoming faster, and network link bandwidths are becoming wider at an amazing rate. Moreover, an increasing number of new applications, such as data grids and storage area networks (SANs), have begun to appear. These applications are placing new demands on networks, especially in terms of transmission speed. At present, networking infrastructure has the capability to transmit data quickly, and the problem is how TCP uses it. Current TCP implementations, which are primarily based on TCP Reno, cannot fully utilize Long Fat Networks (LFNs) [3], which are high-speed long-delay networks. Essentially, the characteristics of TCP Reno, i.e. the congestion window size is halved when packet loss occurs and is increased by one packet per Round Trip Time (RTT) when no packet is dropped, limit its performance. In order to address this problem, a number of improvements have been proposed [1, 3–8].

One traditional method by which to improve TCP performance on LFNs is to tune certain TCP parameters, e.g. using the Selective ACKnowledgement (SACK) option [4] and tuning the TCP socket buffer size [5]. However, TCP cannot achieve satisfactory throughput in LFNs because the TCP algorithm itself is a limitation. Another solution is the use of parallel TCP mechanism, which utilizes multiple TCP connections concurrently to transmit a large amount of data. Parallel TCP has been widely used to increase TCP performance, primarily because of its easy implementation. For example, GridFTP [6] supports parallel TCP connections to transfer data. Fundamental to the use of parallel TCP is the selection of the number of TCP connections. This number affects both the aggregate throughput of parallel TCP and the impact on other competing traffic that shares the same links. Selecting the optimal number of parallel TCP connections in order to maximize the performance without affecting the fairness is not an easy task.

In recent years, efforts to improve the TCP performance in LFNs have focused on modifying the congestion control mechanism of TCP itself. These efforts include HSTCP [3], FAST TCP [7], XCP [8] and gHSTCP [1]. In particular, HSTCP is a simple, representative example that uses the Additive Increase and Multiplicative Decrease (AIMD) principle of TCP Reno and so is easily deployed in the current Internet. In addition, HSTCP is currently the only protocol that is recommended by IETF as an Experimental RFC in LFNs [3].

However, the fairness between these new TCP variants and the traditional TCP Reno is quite an 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 share the same high-speed backbone links. Note that this does not mean that HSTCP connections and TCP Reno connections receive the same throughput; however, the HSTCP connections should not achieve high performance by sacrificing the performance of the TCP Reno connections.

Based on our previous study [1], 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 [1]. 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 [1] 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

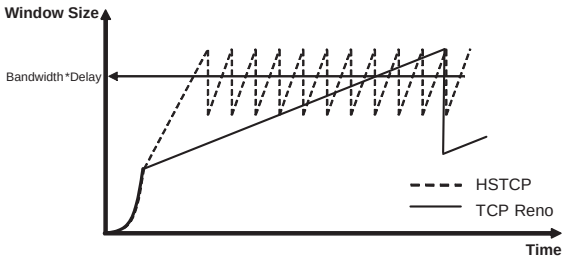


Figure 1: Congestion Window (TCP Reno and HSTCP)

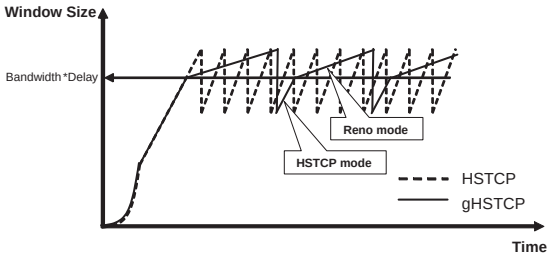


Figure 2: Congestion Window (HSTCP and gHSTCP)

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 paper 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 paper is organized as follows. In Section 2, we provide a short description of HSTCP and gHSTCP. In Section 3, 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 in Section 4. Section 5 summarizes the conclusions of the present study and discusses future areas for investigation.

2 HSTCP and gHSTCP

In this section, we briefly describe the algorithms of HSTCP and gHSTCP. (For more detailed descriptions, please refer to [1, 3].)

2.1 HSTCP

In order to overcome the problems associated with using TCP Reno in LFNs, HSTCP was proposed in [3]. Figure 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. 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, $a(w)$ is a monotonically increasing function of w, whereas $b(w)$ is a monotonically decreasing function of w. Based on this characteristic, TCP connections using the HSTCP mechanism can maintain large congestion windows in LFNs, as shown in Figure 1 so that the network link bandwidth can be better utilized.

2.2 gHSTCP

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, 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 [1]. 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 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.

For this purpose, the following algorithm is employed. Denote the departure time and the RTT value of the i-th transmitted packet as d_i and t_i, respectively, the correlation between d_i and t_i is tested statistically. If a positive correlation is recognized, that is, if an increasing trend in the observed RTT values is present, then the sender determines that congestion is occurring. The sender should therefore slow down the increase in the sending rate in order to maintain fairness against TCP Reno connections. The process during this period is referred to as *Reno mode*, in which the sender increases its congestion window in a manner identical to that in the standard TCP Reno. This will maintain fairness between TCP Reno and gHSTCP connections. On the other hand, if there is a

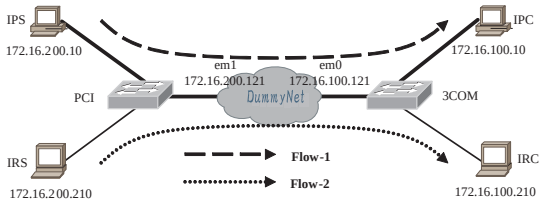


Figure 3: Topology

non-positive correlation between d_i and t_i, the network is in an under-utilized state and the sender should increase the congestion window rapidly in order to utilize the unused bandwidth. The process during this period is called the *HSTCP mode*. The sender increases the congestion window size in the same manner as in 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:

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

We have shown that gHSTCP based on this mechanism can provide better performance and fairness by simulations [1].

3 The Refined gHSTCP Algorithm

3.1 Problem Description

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

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 3. 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. The packet size is 1460 bytes. Dummynet [9] 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 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, 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

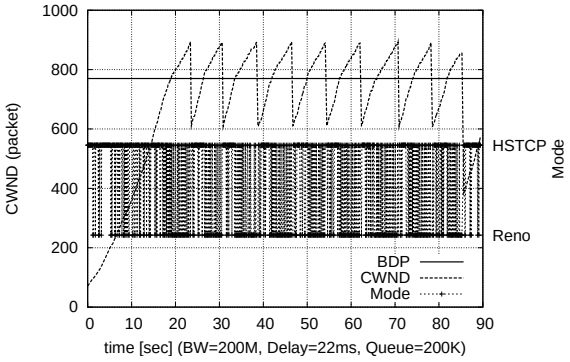


Figure 4: Congestion Window and Mode (Original Algorithm)

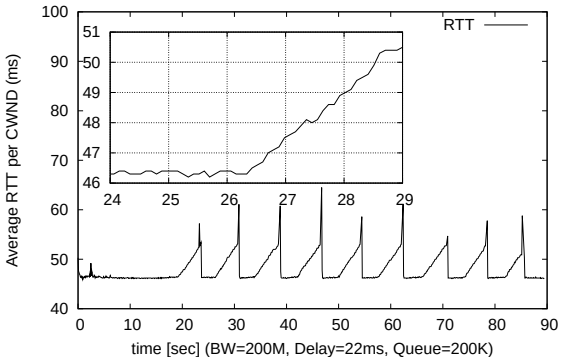


Figure 5: RTT Change

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 monotonously in a local period, even if the congestion window becomes large. In Figure 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 5 shows the period of 24 – 29 s. Together with Figure 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.

3.2 Refined 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

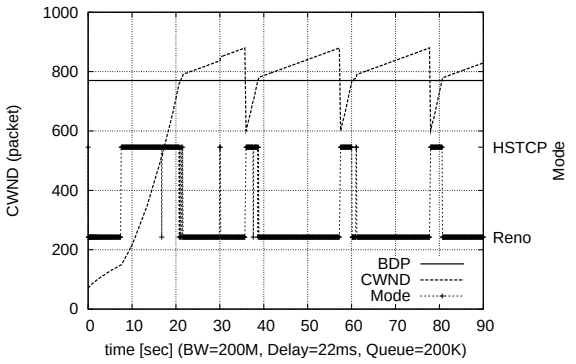


Figure 6: Congestion Window and Mode (Refined Algorithm)

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

```
Notation:
RTT_min: minimum of the average RTT in
  a 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
   the mode is decided by the RTT trend.
If RTT >= RTT_min + 4*RTT_std
   Reno mode is used.
```

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 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.

4 Performance Comparison

4.1 Test-bed Network 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 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 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.

4.2 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:

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

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 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.

Table 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

4.3 Experiments of Scenario-1

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

- Exp-1: Flow-1 uses TCP Reno.
- Exp-2: Flow-1 uses HSTCP.
- Exp-3: Flow-1 uses the parallel TCP mechanism.
- Exp-4: Flow-1 uses gHSTCP.

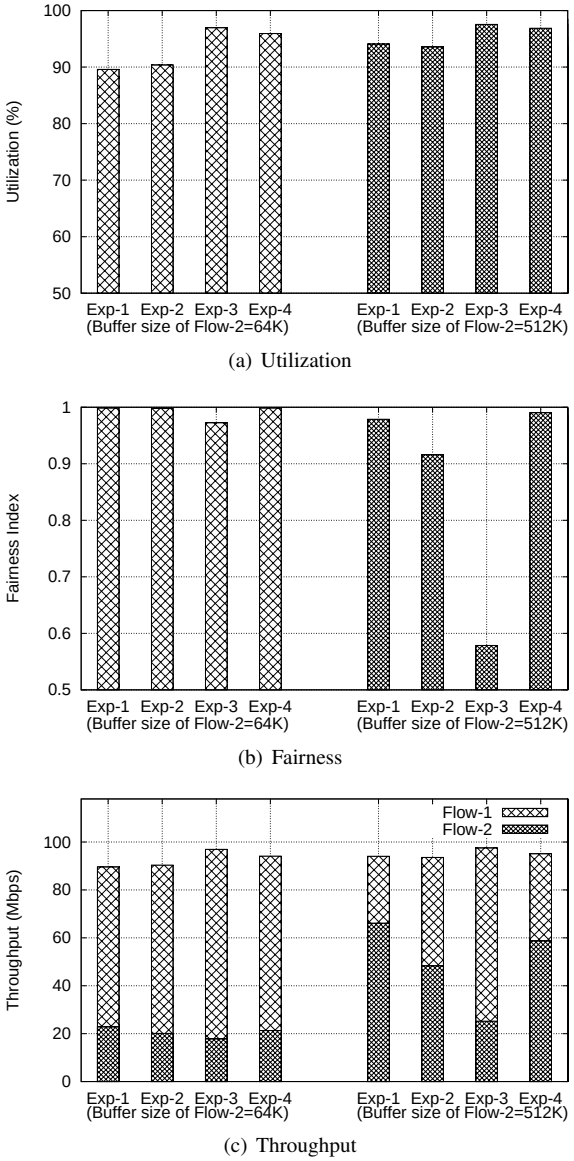
Note that when the parallel TCP mechanism is used, we use eight TCP connections in order to fully utilize the bottleneck link due to the default buffer size of 64 Kbytes. The results of link utilization, fairness index and throughput are illustrated in Figure 7. Note that the throughput of Flow-2 represents the total throughput of the two TCP connections in Flow-2.

Figure 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 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. Figure 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.

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

4.4 Experiments of Scenario-2

In Scenario-2, four experiments are conducted, in which similar to Scenario-1, the buffer size of Flow-2 is



(a) Utilization

(b) Fairness

(c) Throughput

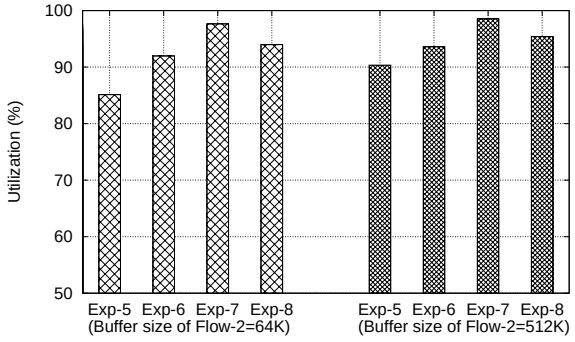
Figure 7: Scenario-1 (Bandwidth=100 Mbps)

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.

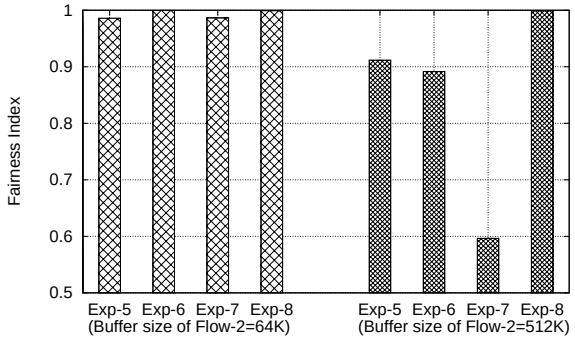
- Exp-5: TCP Reno is used by Flow-1.
- Exp-6: HSTCP is used by Flow-1.
- Exp-7: Parallel TCP mechanism is used by Flow-1.
- Exp-8: gHSTCP is used by Flow-1.

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 8.

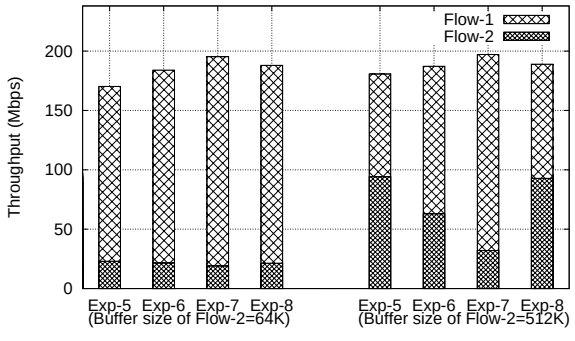
On the whole, the utilization and fairness trends are the same as those demonstrated in Scenario-1. Parallel TCP achieves the best utilization, but the worst fairness. gHSTCP offers higher utilization and better fairness than the other protocols. That is, gHSTCP is the best tradeoff in terms of link utilization and fairness. On the other hand, differences between the two scenarios remain because the link bandwidth of the bottleneck is changed



(a) Utilization



(b) Fairness



(c) Throughput

Figure 8: Scenario-2 (Bandwidth=200 Mbps)

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 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 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 of Flow-2. Therefore, redundant link bandwidth exists that can be used by other flows. As illustrated in Figure 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.

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. Due to limited space, the results of varying number of parallel TCP connections are not presented here.

5 Conclusion

In this paper, we performed an experimental study to assess the performance of high-speed TCP and parallel TCP variants 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.

In the present paper, the performance of gHSTCP is evaluated only when the TailDrop mechanism is deployed at routers. Active Queue Management (AQM), such as Random Early Detection (RED) [10], is an important queue management mechanism. Furthermore, it is necessary to evaluate gHSTCP with AQM. In addition, gHSTCP must also be evaluated in both a higher speed network (e.g. the link bandwidth of the bottleneck is 1 Gbps) and the Internet.

References

[1] Z. Zhang, G. Hasegawa, and M. Murata, "Performance analysis and improvement of HighSpeed TCP with TailDrop/RED routers," in *Proc. 12th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, October 2004.

[2] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," *IETF RFC 2581*, April 1999.

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

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

[5] J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP buffer tuning," in *Proc. ACM SIGCOMM*, August 1998.

[6] W. Allcock, "GridFTP: Protocol extensions to FTP for the grid," April 2003, available as: http://www.globus.org/research/papers/GFD-R.0201.pdf.

[7] 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.

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

[9] L. Rizzo, "IP dummynet," available as: http://info.iet.unipi.it/~luigi/ip_dummynet/.

[10] 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.