Master's Thesis

Title

On Congestion Control Mechanism of High–Speed TCP

Supervisor

Prof. Masayuki Murata

Author

Koichi Tokuda

February 12th, 2003

Department of Informatics and Mathematical Science

Graduate School of Engineering Science

Osaka University

Master's Thesis

On Congestion Control Mechanism of High–Speed TCP

Koichi Tokuda

Abstract

Among Internet services that have recently been initiated, including data GRID network and storage area network (SAN), the server machines have gigabit-level network interfaces such as Gigabit Ethernet, and directly connect to high-speed network to deliver gigabyte/terabyte data to other hosts. Although these services require a large amount of network bandwidth and disk storage, such services will emerge increasingly in the future Internet since their costs are rapidly decreasing.

When the TCP Reno version, which is the most popular version of TCP included in the current OSs, is used for such high-speed data transmissions, it cannot achieve enough throughput because of the essential nature of TCP's congestion control mechanism. HighSpeed TCP has recently been proposed as one of the possible ways to improve the throughput by modifying the congestion control mechanism. However, the performance of HighSpeed TCP has not been fully investigated. Especially, fairness issues between HighSpeed TCP and TCP Reno have not been considered.

In this thesis, we first investigate the throughput and fairness properties of HighSpeed TCP through mathematical analysis and simulation studies. We show that HighSpeed TCP can provide larger throughput than TCP Reno, but it cannot fully utilize the link bandwidth because of bursty packet losses at the router buffer. We also demonstrate that HighSpeed TCP degrades the throughput of TCP Reno when they share the bottleneck link. We then propose Gentle HighSpeed TCP, which has new congestion control mechanisms of TCP that improve the throughput of HighSpeed

TCP. One of the major features in Gentle HighSpeed TCP is that it has two modes in the congestion avoidance phase, each of which uses different algorithms in changing window size. It also has a new mechanism to avoid throughput degradation in the initial slow start phase.

We confirm the effectiveness of Gentle HighSpeed TCP by simulation experiments. We observe from the simulation results that the throughput of Gentle HighSpeed TCP becomes larger than that of the original HighSpeed TCP by up to 30%. Furthermore, Gentle HighSpeed TCP can greatly improve the data transmission performance without degrading the throughput of competing TCP Reno connections. That is, Gentle HighSpeed TCP outperform the original HighSpeed TCP in both terms of the throughput and the fairness with TCP Reno.

Keywords

TCP (Transmission Control Protocol)

TCP Reno

HighSpeed TCP

Fairness

Analysis

Contents

List of Figures

1 Introduction

Although the amount of Internet traffic is explosively increasing with the rapid growth of Internet users, the Internet works well without serious network collapses. This is of course due to the increase of network bandwidth brought by technology improvement such as WDM (Wavelength Division Multiplexing) [1, 2]. However, the key reason is that TCP (Transmission Control Protocol) [3, 4], used by most of the current Internet applications, detects network congestion, avoids/resolves it, and regulates its packet transmission rate adaptively and consistently. However, as recently-emerging Internet applications have more service diversity, various kinds of demands arise, which TCP cannot satisfy successfully.

For example, in the services including data GRID network [5] and storage area network (SAN) [6], the hosts (server machines) have gigabit-level network interfaces such as Gigabit Ethernet, and directly connect to high-speed network to deliver gigabyte/terabyte size of data to other hosts for moving program data, backup, database synchronization, and so on. Although these services require a large amount of network bandwidth and disk storage, such services will emerge increasingly in the future Internet since their costs are rapidly decreasing [7]. When TCP Reno version, which is the most popular version of TCP in the current OSs, is used for such data transmission, it cannot achieve enough throughput because of the essential nature of TCP's congestion control mechanism. According to [8] on HighSpeed TCP, for a TCP Reno connection with 1500 Byte packet size and 100 msec RTT (Round Trip Time) to fill 10 Gbps link, a congestion window of 83,333 packets is required, which means that packet loss rate becomes less than $2 * 10^{-10}$. Furthermore, when packets are lost in the network, it takes over 40,000 RTT (about 4,000 sec.) to recover the throughput. It means that it is impossible to obtain such a large throughput with TCP Reno. The main reason is that TCP Reno decreases its congestion window size dramatically when packet losses take place but increases it very slightly when it experiences no packet loss.

HighSpeed TCP was recently proposed in [8] as one possible way to overcome the above prob-

lem and provide considerably larger throughput than TCP Reno in such a situation. It modifies the increasing/decreasing algorithm of a congestion window size in the congestion avoidance phase. That is, HighSpeed TCP quickly increases and slowly decreases its congestion window than TCP Reno, to keep the congestion window size enough large to fill a high-speed link. Although it intuitively becomes possible to obtain larger throughput than TCP Reno, the performance and characteristics of HighSpeed TCP have not been fully investigated except [9]. One example is a fairness issue of HighSpeed TCP. The fairness issues of TCP are important and have been actively investigated in the past literature [10-18]. Almost all of them focus on the fairness among connections of a certain TCP version having differences in environment such as RTT, packet dropping probability, the number of active connections, the size of transmitted documents. The fairness between the traditional TCP and new TCP mechanisms like HighSpeed TCP, is also quite important problem when we consider a migration path of the new TCPs. However, the HighSpeed TCP mechanism does not consider the situation where HighSpeed TCP and TCP Reno connections share the network bandwidth. It is very likely that HighSpeed TCP connections between server hosts and traditional TCP Reno connections for Web access and e-mail transmission share a high-speed backbone link, which means that it is important to investigate the fairness characteristics between HighSpeed TCP and TCP Reno.

In this paper, we first investigate the throughput and fairness properties of HighSpeed TCP through mathematical analysis study. In the analysis, we model a cyclic change in the occupation level of the router buffer and the congestion window size of HighSpeed TCP and TCP Reno connections triggered by packet loss events, and calculate their average throughput values. From the analysis and simulation results, we demonstrate that HighSpeed TCP can provide larger throughput than TCP Reno, but it cannot fully utilize the link bandwidth because of bursty packet losses at the bottleneck router buffer. It is also shown that HighSpeed TCP degrades the throughput of TCP Reno when they share the bottleneck link since it opens the congestion window quickly even when the transmitted packets are queued at the router buffer.

We then propose a new congestion control mechanism, called Gentle HighSpeed TCP, to solve the above problems of HighSpeed TCP. Gentle HighSpeed TCP can improve the throughput of both HighSpeed TCP and TCP Reno connections sharing the network bandwidth. One of the major features of Gentle HighSpeed TCP is that it has two modes in its congestion avoidance phase, each of which uses different algorithms in changing window size. That is, when the utilization of the bottleneck link is under 100, Gentle HighSpeed TCP uses the same algorithm as the original HighSpeed TCP (*HighSpeed mode*). When the bottleneck link is fully utilized, on the other hand, Gentle HighSpeed TCP behaves equally to TCP Reno (*Reno mode*). To estimate the bottleneck link utilization, Gentle HighSpeed TCP uses RTT values of transmitted packets. When the measured RTTs indicate an increasing trend, it determines that the link bandwidth is fully utilized, and vice versa. Furthermore, it also has a new mechanism in the slow start phase to avoid throughput degradation in the initial slow start phase.

We confirm the effectiveness of Gentle HighSpeed TCP by extensive simulation experiments. The simulation results show that Gentle High Speed TCP can avoid the bursty packet losses occurred in the original HighSpeed TCP, which results in that it can successfully alleviate retransmission timeout. Consequently, we can observe that the throughput of Gentle HighSpeed TCP becomes larger than that of the original HighSpeed TCP by up to 30%. Furthermore, Gentle HighSpeed TCP can greatly improve the fairness with competing TCP Reno connections, which means that it can enhance the data transmission performance without degrading the throughput of TCP Reno connections. That is, Gentle HighSpeed TCP outperform the original HighSpeed TCP in both terms of the throughput and the fairness with TCP Reno.

The rest of this thesis is organized as follows. In Section 2 we summarize the congestion control mechanisms of TCP Reno and HighSpeed TCP. In Section 3 we explain our motivation in this work by showing the problems of HighSpeed TCP through some simulation results. In Section 4 we propose a mathematical analysis to derive the average throughput of HighSpeed TCP and TCP Reno connections when they share a bottleneck link, and discuss the fairness property of

HighSpeed TCP by using the analysis results. In Section 5 we propose a new congestion control mechanism of TCP, called Gentle HighSpeed TCP, and show its effectiveness through various simulation experiments. We finally present concluding remarks in Section 6.

2 Congestion Control Mechanisms of TCP

TCP changes its congestion window size dynamically to regulate its packet transmission rate according to the network congestion level. In this section, we describe the congestion control mechanisms of TCP Reno and HighSpeed TCP. In particular, we concentrate on explaining their algorithms of changing their congestion window sizes. For more detailed algorithms, refer to [3] and [8].

2.1 TCP Reno

TCP Reno changes its congestion window periodically triggered by a packet loss event. Here, we define the interval from the $(i-1)$-th packet loss event to the i-th packet loss event as the i-th *cycle*. We further divide the i-th cycle into RTTs and consider a congestion window size in each RTT. The congestion window size of TCP Reno at the j-th RTT of the i-th cycle is defined as $W_{Reno}(i,j)$.

The congestion control mechanism of TCP Reno consists of two phases: the slow start phase and the congestion avoidance phase. in each of which TCP Reno uses a different algorithm in increasing/decreasing congestion window size. In the slow start phase, TCP Reno increases its window size by one packet on receiving an ACK packet. On the other hand, in the congestion avoidance phase, TCP Reno increases its window size $W_{Reno}(i,j)$ by $1/W_{Reno}(i,j)$ packets when it receives an ACK packet. Focusing on the changes in congestion window in a RTT, $W_{Reno}(i,j)$ is derived as follows:

$$W_{Reno}(i,j) = \begin{cases} \text{(Slow Start Phase:)} & \\ 2 \cdot W_{Reno}(i,j-1) & \text{if } W_{Reno}(i,j-1) < S_{Reno}(i) \\ \text{(Congestion Avoidance Phase:)} & \\ W_{Reno}(i,j-1) + 1 & \text{if } W_{Reno}(i,j-1) \geq S_{Reno}(i) \end{cases} \quad (1)$$

where $S_{Reno}(i)$ is a $ssthresh$ value in the i-th cycle at which TCP Reno changes its phase from

the slow start phase to the congestion avoidance phase. According to Eq. (1), TCP Reno continues to increase its congestion window until it detects packet losses. When packet losses occur in the network, TCP detects them either by waiting a retransmission timeout or by receiving duplicate ACK packets (three or more ACK packets with the same sequence number), and retransmits them. If the packet losses are detected by the retransmission timeout, TCP Reno decreases the congestion window size to one packet. On the other hand, when the packet losses are detected by duplicate ACK packets, TCP Reno sets its congestion window to a half of that just before the packet loss. In both cases, TCP Reno halves $S_{Reno}(i)$.

2.2 HighSpeed TCP

According to [8], when we use TCP Reno for data transmission via a high-speed link of larger than 1 Gbps bandwidth, TCP Reno requires quite a low packet loss rate to fully utilize the link and it takes considerable time to achieve sufficient throughput. Therefore, it is almost impossible for TCP Reno to provide such a large throughput in the actual network. It is mainly because TCP Reno decreases its congestion window size dramatically when packet losses occur but increases it slightly when it experiences no packet loss, which results in its requiring many RTTs to get a large enough congestion window.

To overcome this problem, a new congestion control mechanism, called *HighSpeed TCP*, was proposed in [8]. Its algorithm in increasing/decreasing the congestion window is essentially based on the AIMD (Additive Increase Multiplicative Decrease) discipline [19], which is equal to that of TCP Reno. The difference between them is that the degree of increasing/decreasing congestion window size in one RTT. In what follows, we explain the algorithm of HighSpeed TCP. Similarly to TCP Reno, we define the congestion window size of HighSpeed TCP at the j-th RTT of the i-th cycle as $W_{HS}(i, j)$.

When the current congestion window size is smaller than W_{low}, HighSpeed TCP changes its congestion window size according to the same algorithm as that of TCP Reno. On the other hand,

when the current congestion window is larger than W_{low}, it increases its congestion window more quickly, and decreases it more slowly than TCP Reno. Therefore, it is expected to obtain larger throughput than TCP Reno by keeping the congestion window size to a larger value. The degree of increasing/decreasing congestion window size depends on its current value. That is, when the congestion window size is larger, HighSpeed TCP increases it more quickly, and decreases it more slowly. We define the increasing degree of congestion window in each RTT as $a(w)$, and the decreasing degree of congestion window as $b(w)$, when its current congestion window is w. That is, HighSpeed TCP increases its window size w by $a(w)$ packets in one RTT without packet losses and decreases it to $w \cdot (1 - b(w))$ when it detects packet loss by duplicate ACK packets. Focusing on the change in congestion window in a RTT, $W_{HS}(i, j)$ is derived as follows:

$$
W_{HS}(i,j) = \begin{cases}
\text{(Slow Start Phase:)} & \\
2 \cdot W_{HS}(i, j-1) & \text{if } W_{HS}(i,j-1) < S_{HS}(i) \\
\text{(Congestion Avoidance Phase:)} & \\
W_{HS}(i, j-1) + 1 & \text{if } W_{HS}(i,j-1) \geq S_{HS}(i) \\
 & \text{and } W_{HS}(i,j-1) \leq W_{low} \\
W_{HS}(i, j-1) + a(W_{HS}(i,j-1)) & \text{if } W_{HS}(i,j-1) \geq S_{HS}(i) \\
 & \text{and } W_{HS}(i,j-1) > W_{low}
\end{cases} \tag{2}
$$

where $S_{HS}(i)$ is a $ssthresh$ value in the i-th cycle. According to [8], $a(w)$ and $b(w)$ are given by:

$$
\begin{aligned}
a(w) &= \frac{2w^2 \cdot b(w) \cdot p(w)}{2 - b(w)} \\
b(w) &= \frac{\log(w) - \log(W_{low})}{\log(W_{high}) - \log(W_{low})}(b_{high} - 0.5) + 0.5 \\
p(w) &= \exp\left[\frac{\log(w) - \log(W_{low})}{\log(W_{high}) - \log(W_{low})}\{\log(P_{high}) - \log(P_{low})\} + \log(P_{low})\right]
\end{aligned} \tag{3}
$$

In Eq. (3), P_{low} is defined by Eq. (4). It represents the packet loss ratio when the average size of

the congestion window in TCP Reno equals to W_{low} [20]:

$$P_{low} = \frac{1.5}{W_{low}^2} \tag{4}$$

P_{high}, W_{high}, b_{high}, and W_{low} in Eq. (2) are the parameters of HighSpeed TCP and have the following meanings: HighSpeed TCP determines $a(w)$ and $b(w)$ so that it achieves the congestion window size of W_{high} packets with the packet loss probability of P_{high}. Furthermore, when the congestion window is W_{high}, it decreases its congestion window size to $(1 - b_{high}) \cdot W_{high}$ in receiving three duplicate ACK packets. We can easily determine these parameters when a RTT value between sender and receiver hosts, a packet loss probability, a packet size, and a target throughput are given. For example, [8] shows the parameter setting $W_{low} = 31$, $P_{high} = 10^{-7}$, and $b_{high} = 0.1$ to achieve 10 Gbps throughput when the propagation delay for the round-trip path between sender and receiver hosts is 100 msec, and the packet size is 1500 Byte.

3 Problems of HighSpeed TCP

Although HighSpeed TCP is expected to obtain a larger throughput than TCP Reno in a high-speed network, the detailed performance has not been fully investigated. Especially, since HighSpeed TCP does not consider the fairness against TCP Reno connections in the region where it opens the congestion window very large, the fairness evaluation has not been done at all. In this section, we show the results of simple simulation experiments to exhibit the problems of HighSpeed TCP in both terms of throughput and fairness. We use ns [21] for the simulation experiment.

The network model used in the simulation is depicted in Fig. 1. In this model, we assume the network where two local networks are interconnected via a high-speed link. N_{HS} endhosts (hstcp$_1$, ..., hstcp$_{N_{HS}}$) are connected directly to the link and use HighSpeed TCP for data transmission. At the same time, N_{Reno} endhosts (reno$_1$, ..., reno$_{N_{Reno}}$) using the traditional TCP Reno share the same link through 100 Mbps link. That is, there are N_{HS} HighSpeed TCP connections and N_{Reno} TCP Reno connections on the bottleneck link. The bandwidth and the propagation delay of the bottleneck link between router R_A and router R_B are set to 1 Gbps and 25 msec, and those between R_B and R_C are set to 1 Gbps and 5 msec. The propagation delays of the links between routers and endhosts are all set to 5 msec. We use a Taildrop discipline at the buffer of R_A. The buffer size of R_A is set to 4167 packets, which is equal to the bandwidth-delay product of the bottleneck link between R_A and R_B.

Figure 2 shows the total throughput values of HighSpeed TCP and TCP Reno connections when N_{Reno} = 10 and N_{HS} is changed from 0 to 3. For comparison purpose, we also plot the results when N_{HS} endhosts use TCP Reno for data transmission (labelled "broad tcp reno" in Fig. 2). From this figure, it is observed that the bottleneck link is not fully utilized when HighSpeed TCP connection does not exist. In Fig. 2, about 200 Mbps bandwidth remains unused. This is because packet losses take place at the buffer of the router R_A, which degrades the throughput of TCP Reno connections. The throughput of TCP Reno connections is degraded by introducing

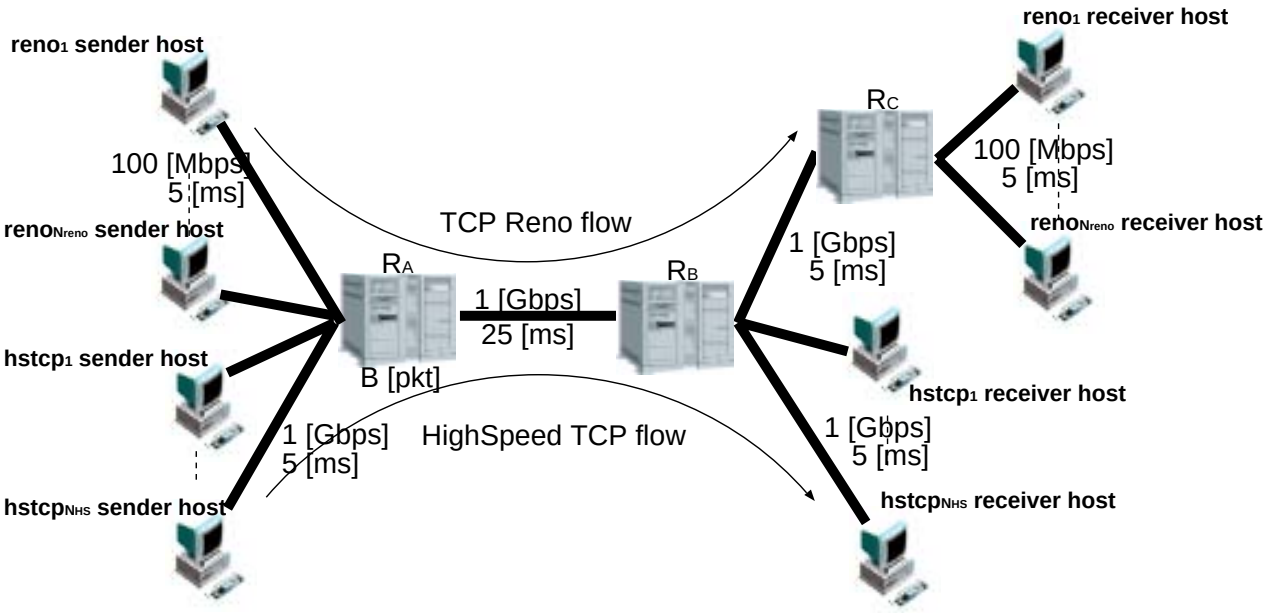


Figure 1: Simulation Model

HighSpeed TCP instead of TCP Reno for the data transmission between hstcp hosts. Especially, the degradation degree becomes large as the number of HighSpeed TCP connection increases. It is because HighSpeed TCP does not consider the fairness with existing TCP Reno connections, and the throughput of TCP Reno connection degrades unexpectedly. This unfairness is caused by the difference of increasing/decreasing speed of the congestion window size, as explained above.

It can be also observed from Fig. 2 that the bottleneck link utilization does not reach 100% even when we use HighSpeed TCP for data transmission. The reason is that bursty packet losses occur at the router buffer. Since HighSpeed TCP greatly increases its congestion window size, more than one packet is lost at the router buffer. Such lost packets cause a retransmission timeout, since more than three packet losses can not be recovered by a fast retransmit algorithm, which is the essential nature of TCP [22, 23]. Since the timeout sets the congestion window size to one packet, it causes the serious throughput degradation. This phenomena can be seen in Fig. 3(a), which shows the change in the congestion window size as a function of time. It can be observed

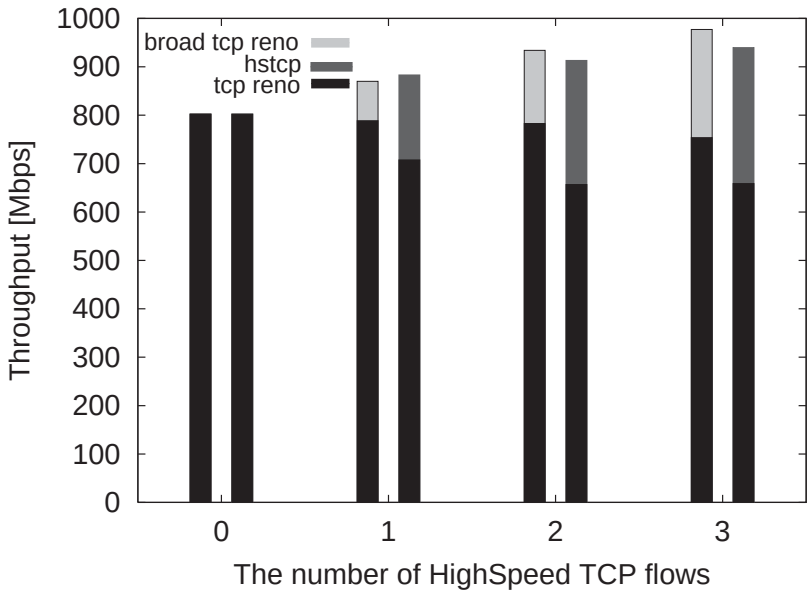


Figure 2: Fairness between HighSpeed TCP and TCP Reno connections

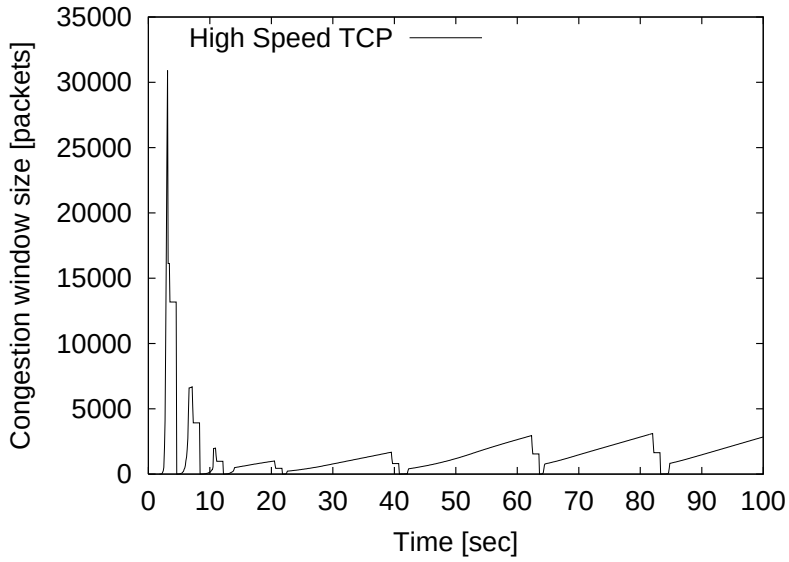
that the HighSpeed TCP connection waits a retransmission timeout and the congestion window size is set to one packet every time packet losses occur at the router buffer.

We also note that in the initial slow start phase from 0 sec to 5 sec in Fig. 3(a), the window size increases exponentially and suddenly drops to one packet. This means that many packet losses occur at the router buffer and the connection experiences a retransmission timeout. This is because of the slow start algorithm of the traditional TCP which doubles the congestion window size in each RTT as described in Eq. (1). The problem becomes worse in HighSpeed TCP since HighSpeed TCP is likely to be used in the high-speed network. To overcome this problem, the authors in [8] recommended using *Limited Slow Start* algorithm in [24]. It limits the speed of increasing congestion window size in the slow start phase. However, it is not an essential solution since it requires manually setting the threshold value at which it starts limiting.

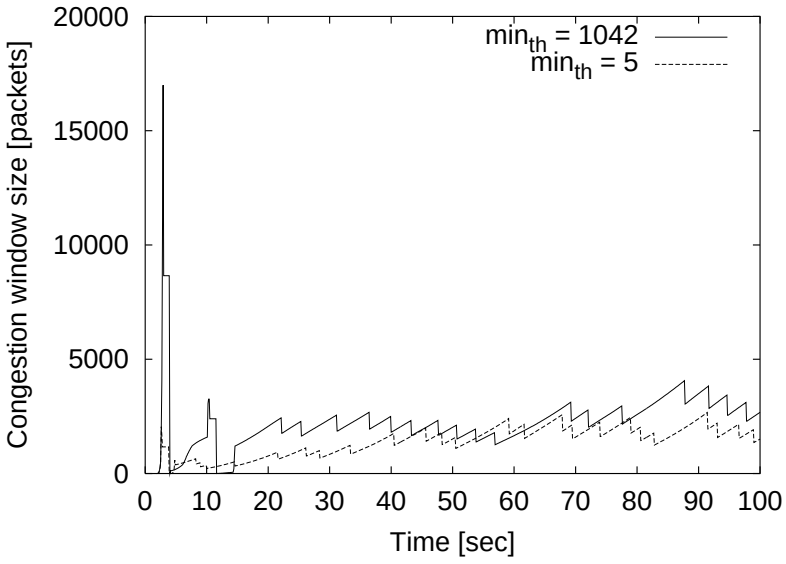
This problem also occurs even if we use RED (Random Early Detection) [25], which avoids bursty packet losses at the router buffer. Figure 3(b) shows the simulation results when we use RED at the buffer of the router R_A. In this figure, we show the change in the congestion window

size as a function of time, in the cases where min_{th} to 1042 packets (one fourth of the router buffer size B) and 5 packets. We set $w_q = 0.002$, $max_p = 0.1$, and $max_{th} = 3125$ packets in both cases. From this figure, we can see that when we set min_{th} to a larger value, many packets are dropped at the initial slow start phase as in the case of the Taildrop router. This is because the RED determines its packet discarding probability based on an average queue length at the router buffer, which results in inability to follow up the drastic increase of the congestion window size. When we set min_{th} to small value, on the other hand, we can avoid burst packet losses at the initial slow start phase. However, it takes much time for the congestion window size to become sufficiently large, as shown in Fig. 3(b).

From the above results, we conclude that HighSpeed TCP can provide larger throughput than TCP Reno, but it cannot fully utilize the link bandwidth because of bursty packet losses at the router buffer. It is also shown that HighSpeed TCP degrades the throughput of TCP Reno when they share the bottleneck link. Furthermore, the problem in the initial slow start phase is more serious than TCP Reno since HighSpeed TCP is assumed to be used in the high-speed network.



(a) Taildrop



(b) RED

Figure 3: Change of Congestion Window Size

4 Fairness Analysis

Through the simulation experiments in the previous section, we have shown that HighSpeed TCP degrades the throughput of the traditional TCP Reno connection. In this section, we develop a mathematical analysis to derive the average throughputs of TCP Reno and HighSpeed TCP when they share the bottleneck link. By using it, we confirm the unfairness of HighSpeed TCP against TCP Reno analytically.

4.1 Analysis Model

Figure 4 depicts the network model for our analysis. The network model consists of sender and receiver hosts of TCP Reno connection, sender and receiver hosts of HighSpeed TCP connection, two routers, and links interconnecting hosts and routers. The bandwidth of the link between the router $\mathrm{R_A}$ and router $\mathrm{R_B}$ is μ packet/sec, the buffer size at the router $\mathrm{R_A}$ is B packets, the propagation delay between the sender and receiver hosts is τ sec, the bandwidths of the links between the HighSpeed TCP hosts and the routers are μ_{HS} packet/sec, and those between the TCP Reno hosts and the routers are μ_{Reno} packet/sec. For simplification, there is only one TCP Reno connection and one HighSpeed TCP connection in the network. In our analysis, the HighSpeed TCP host assumed to be connected to the high-speed link directly. That is, we set $\mu = \mu_{HS}$. On the other hand, we assume $\mu_{Reno} < \mu$, which means that the access link bandwidth for the TCP Reno host is smaller than the bottleneck link bandwidth. We assume the sender hosts always have data to send and continue transmitting as much data as allowed by their congestion window sizes.

4.2 Analysis

In our analysis, we model cyclic changes in the occupation level of the router buffer and the congestion window size of HighSpeed TCP and TCP Reno connections triggered by packet loss events, and calculate their average throughput values. As in Section 2, we define the interval from

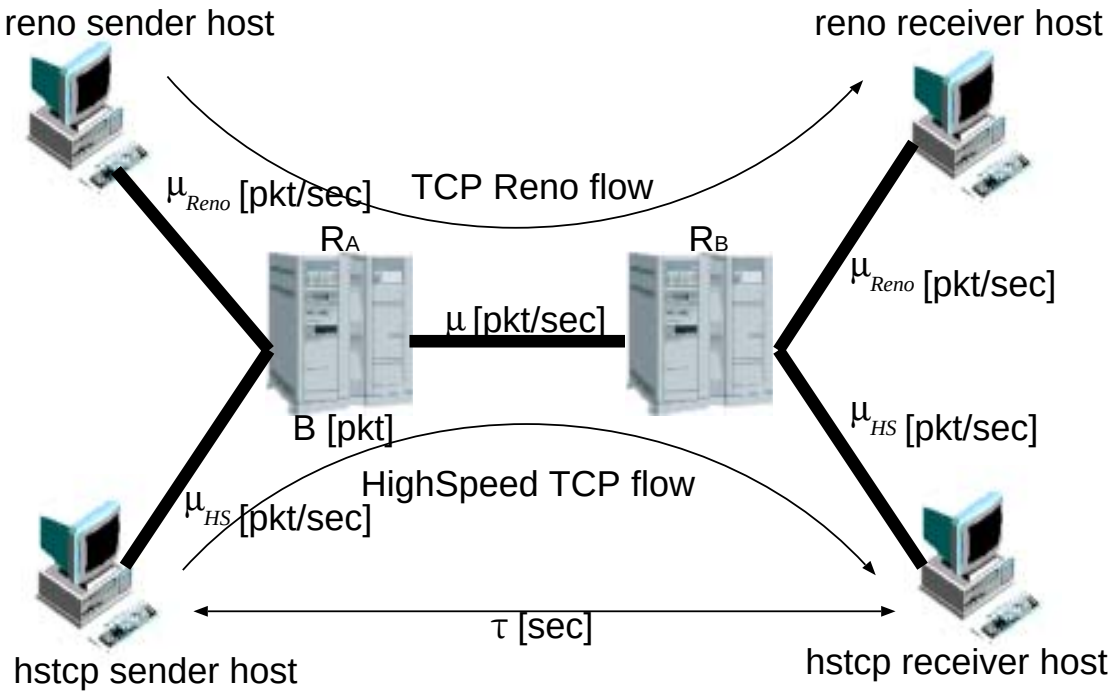


Figure 4: Analysis Model

the $(i-1)$-th packet loss event to the i-th packet loss event as the i-th cycle. We depict a typical evolution of congestion window sizes of HighSpeed TCP and TCP Reno at the i-th cycle in Fig. 5. We denote the congestion window sizes of the TCP Reno and HighSpeed TCP connections at the j-th RTT of the i-th cycle by $W_{Reno}(i,j)$ and $W_{HS}(i,j)$, respectively. We also define $S_{Reno}(i)$ and $S_{HS}(i)$ as the $ssthresh$ values of TCP Reno and HighSpeed TCP connections in the i-th cycle. Furthermore, the window size of both connections in the end of the i-th cycle is defined as $W'_{Reno}(i)$ and $W'_{HS}(i)$.

Since the congestion window size at the beginning of the i-th cycle is equal to that at the end of the $(i-1)$-th cycle, the following equations are satisfied:

$$W_{Reno}(i,1) = W'_{Reno}(i-1)$$

$$W_{HS}(i,1) = W'_{HS}(i-1)$$

From Eq. (2) in Subsection 2.2, the congestion window size of HighSpeed TCP in the slow start

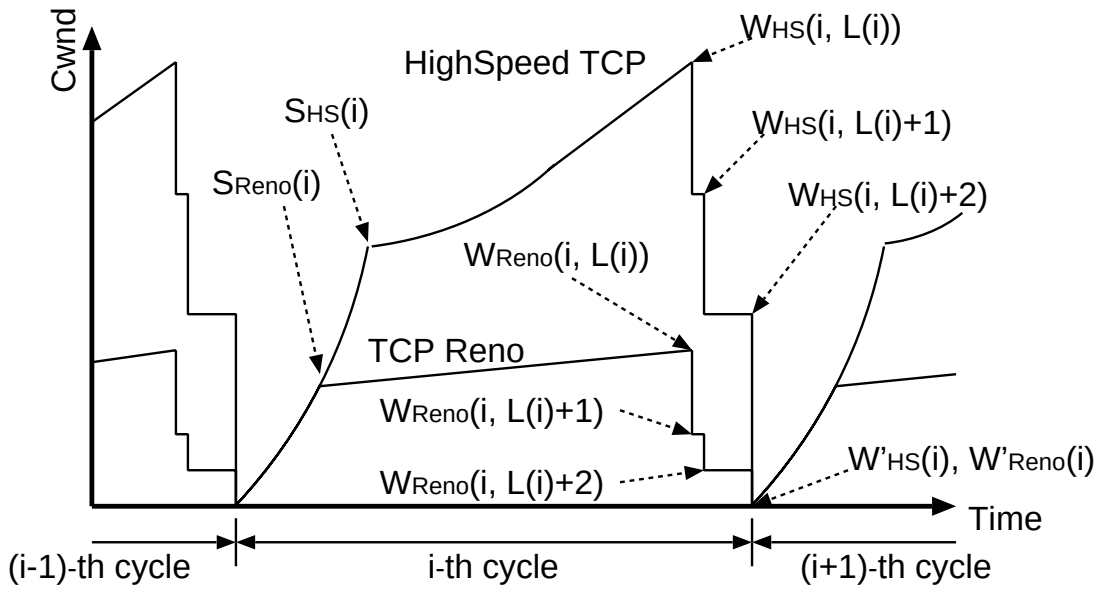


Figure 5: A Typical Evolution of Window Sizes of TCP Reno and HighSpeed TCP Connections

phase and congestion avoidance phase is given by:

$$W_{HS}(i,j) \quad = \quad \begin{cases} 2 \cdot W_{HS}(i,j-1) & \text{if } W_{HS}(i,j-1) < S_{HS}(i) \\ W_{HS}(i,j-1) + a(W_{HS}(i,j-1)) & \text{if } W_{HS}(i,j-1) \geq S_{HS}(i) \end{cases}$$

We next derive $W_{Reno}(i,j)$, the change in the congestion window size of the TCP Reno connection. Since we assume $\mu_{Reno} < \mu$, $W_{Reno}(i,j)$ cannot be determined only by Eq. (1). To derive $W_{Reno}(i,j)$, we introduce W_{lim}, the maximum number of packets that can be transmitted in one RTT by the TCP Reno connections. Since the TCP Reno connection traverses the link of μ_{Reno} bandwidth, W_{lim} can be described as follows:

$$W_{lim} \quad = \quad 2\tau\mu_{Reno}$$

Therefore, TCP Reno transmits $\min(W_{lim}, W_{Reno}(i,j))$ packets to the network in each RTT. It

means that it receives $\min(W_{lim}, W_{Reno}(i,j))$ ACK packets. Therefore, $W_{Reno}(i,j)$ is given by:

$$W_{Reno}(i,j) = \begin{cases} W_{Reno}(i,j-1) + \min(W_{lim}, W_{Reno}(i,j-1)) \\ \qquad\qquad\qquad\qquad (\text{if } W_{Reno}(i,j-1) < S_{Reno}(i)) \\ W_{Reno}(i,j-1) + \frac{\min(W_{lim}, W_{Reno}(i,j-1))}{W_{Reno}(i,j-1)} \\ \qquad\qquad\qquad\qquad (\text{if } W_{Reno}(i,j-1) \geq S_{Reno}(i)) \end{cases}$$

When the buffer of the router R_A overflows and a packet loss event occurs, the bottleneck link and the router buffer are filled by packets from both TCP connections. Considering that the packet loss event occurs at $L(i)$-th RTT in i-th cycle, the following equation is satisfied:

$$\min(W_{Reno}(i, L(i)), W_{lim}) + W_{HS}(i, L(i)) > 2\tau\mu + B$$

We define $D(i)$ as the number of dropped packets at the buffer overflow. Then $D(i)$ is given by:

$$D(i) = W_{Reno}(i, L(i)) + W_{HS}(i, L(i)) - (2\mu\tau + B)$$

We further define the number of dropped packets of the TCP Reno connection as $D_{Reno}(i)$ and that of the HighSpeed TCP connection as $D_{HS}(i)$. By assuming that the ratio of $D_{Reno}(i)$ and $D_{HS}(i)$ is equal to the ratio of their congestion window sizes, the following equations are satisfied:

$$\begin{aligned} D_{Reno}(i) &= \frac{W_{Reno}(i, L(i))}{W_{Reno}(i, L(i)) + W_{HS}(i, L(i))} \cdot D(i) \\ D_{HS}(i) &= \frac{W_{HS}(i, L(i))}{W_{HS}(i, L(i)) + W_{HS}(i, L(i))} \cdot D(i) \end{aligned}$$

We next consider the congestion window size of each connection just after the packet losses. Since we assume Taildrop discipline at the router buffer in our analysis, the packets are dropped in bursty fashion. That is, we assume $D(i) > 1$. When three or more packets are dropped in a window, TCP Reno retransmits the first two packets by the fast retransmit algorithm, and then the timeout occurs for the remaining lost packets [22, 23]. HighSpeed TCP has the same algorithm in recovering lost packets. Since the fast retransmit algorithm halves the congestion window when invoked, the congestion window sizes of TCP Reno and HighSpeed connections after the first retransmission

are derived by:

$$\begin{aligned} W_{Reno}(i, L(i)+1) &= \frac{W_{Reno}(i, L(i))}{2} \\ W_{HS}(i, L(i)+1) &= W_{HS}(i, L(i)) \cdot (1 - b(W_{HS}(i, L(i)))) \end{aligned}$$

Similarly, the congestion window sizes after the second retransmission are given by:

$$\begin{aligned} W_{Reno}(i, L(i)+2) &= \frac{W_{Reno}(i, L(i)+1)}{2} \\ W_{HS}(i, L(i)+2) &= W_{HS}(i, L(i)+1) \cdot (1 - b(W_{HS}(i, L(i)+1))) \end{aligned}$$

The retransmission timeout occurs after the second retransmission, then the congestion window sizes are set to one packet, and the values of $ssthresh$ are updated as follows:

$$\begin{aligned} S_{Reno}(i+1) &= \frac{W_{Reno}(i, L(i)+2)}{2} \\ S_{HS}(i+1) &= \frac{W_{HS}(i, L(i)+2)}{2} \end{aligned}$$

Consequently, we can derive the congestion window size after the packet losses:

$$W'_{Reno}(i) = \begin{cases} W_{Reno}(i, L(i)+1) & \text{if } D_{Reno}(i) = 1 \\ W_{Reno}(i, L(i)+2) & \text{if } D_{Reno}(i) = 2 \\ 1 & \text{if } D_{Reno}(i) \geq 3 \end{cases}$$

$$W'_{HS}(i) = \begin{cases} W_{HS}(i, L(i)+1) & \text{if } D_{HS}(i) = 1 \\ W_{HS}(i, L(i)+2) & \text{if } D_{HS}(i) = 2 \\ 1 & \text{if } D_{HS}(i) \geq 3 \end{cases}$$

We finally derive the average throughputs of the HighSpeed TCP and the TCP Reno connections by using their congestion window sizes derived by the above analysis. The buffer occupancy of the bottleneck router $\mathrm{R_A}$ at the j-th RTT of the i-th cycle is derived by $\max(W_{Reno}(i,j) + W_{HS}(i,j) - 2\mu\tau, 0)$. Therefore, we can derive the queuing delay at the router buffer, $Q(i,j)$, at the j-th RTT of the i-th cycle as follows:

$$Q(i,j) = \frac{\max(W_{Reno}(i,j) + W_{HS}(i,j) - 2\mu\tau, 0)}{\mu}$$

Therefore, the average throughputs of TCP Reno ρ_{Reno} and HighSpeed TCP ρ_{HS} are given by:

$$\rho_{Reno} = \frac{\sum_{i=1}^{\infty}\sum_{j=1}^{L(i)} W_{Reno}(i,j)}{\sum_{i=1}^{\infty}\sum_{j=1}^{L(i)}(Q(i,j)+2\tau)} \quad (5)$$

$$\rho_{HS} = \frac{\sum_{i=1}^{\infty}\sum_{j=1}^{L(i)} W_{HS}(i,j)}{\sum_{i=1}^{\infty}\sum_{j=1}^{L(i)}(Q(i,j)+2\tau)} \quad (6)$$

The average throughput can be obtained by calculating Eqs. (5) and (6) until ρ_{Reno} and ρ_{HS} converge to a certain value.

4.3 Numerical Results and Discussions

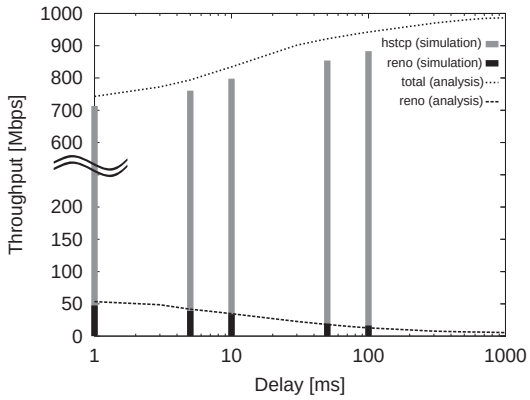
In this subsection, we verify the accuracy of the analysis by comparing the analysis results with simulation results. In the simulation, we use the network model depicted in Fig. 4 where $\mu = \mu_{HS} = 1$ Gbps, the propagation delays of the links between sender/receiver hosts and the routers are all set to 5 msec, the propagation delay of the link between the routers is 25 msec, the packet size is 1500 Byte, and the buffer size of the bottleneck link is 4167 packets, corresponding to the bandwidth-delay product of the bottleneck link. We use the parameter set for the HighSpeed TCP connection as described in Subsection 2.2. Figure 6 shows the analysis results of the average throughput of each connection as a function of the propagation delay of the bottleneck link. We change the bottleneck link bandwidth to 100 Mbps (Fig. 6(a)), 12 Mbps (Fig. 6(b)), and 1.5 Mbps (Fig. 6(c)). For comparison purposes, we also plot the simulation results in each figure. It can be observed that our analysis gives highly accurate estimate of the throughputs of HighSpeed TCP and TCP Reno connections regardless of the network environment.

It is also observed that when the propagation delay of the bottleneck link becomes larger, the HighSpeed TCP connection obtains larger throughput. On the other hand, the throughput of the TCP Reno connection degrades seriously. This is caused by the difference of the increasing/decreasing algorithm of the congestion window size: TCP Reno changes its congestion window size independent of its current window size, while HighSpeed TCP increases it faster when the congestion window size is larger. Consequently, when the propagation delay of the bottleneck

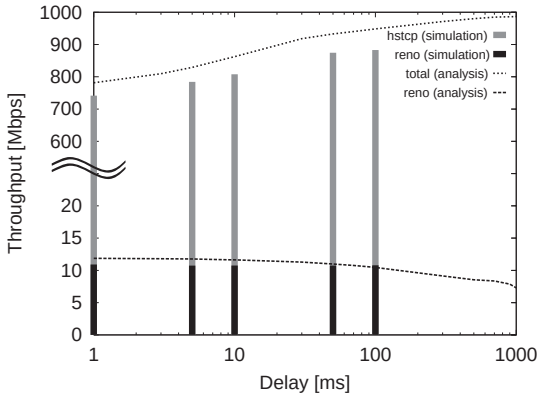
becomes large, that is, when the bandwidth-delay product of the bottleneck link becomes large, HighSpeed TCP occupies the increased bandwidth-delay product quickly, whereas the increasing speed of TCP Reno's congestion window size remains constant. This brings the unfairness between HighSpeed TCP and TCP Reno connections.

Especially when μ_{Reno} is larger, the throughput degradation of TCP Reno connection is larger. This is because the rapid increase of the congestion window size of the HighSpeed TCP connection steals the bottleneck link bandwidth unfairly. HighSpeed TCP takes care of the traditional TCP connections, by setting the threshold value of the congestion widow size, W_{low}, at which it behaves identically to TCP Reno. This mechanism mainly aims to keep HighSpeed TCP's fairness with TCP Reno in the low-speed network. However, the fairness is not considered at all in the high-speed network, where HighSpeed TCP can inflate its congestion window size larger than TCP Reno. This makes the unfairness between HighSpeed TCP and TCP Reno connections in Fig. 6.

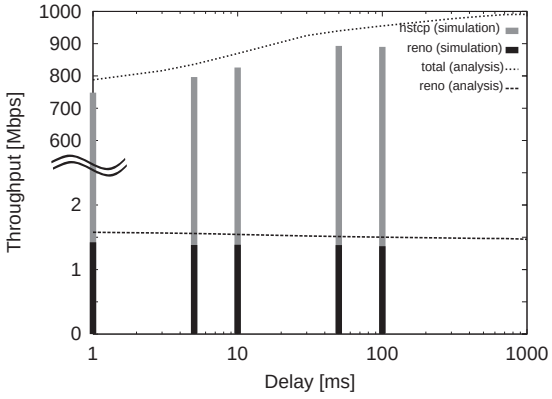
As described in Section 1, however, it is the reasonable assumption that there exists both of the high-speed data transmission between servers and the traditional traffic on a high-speed link. That is, it is important to improve the fairness characteristics between HighSpeed TCP and TCP Reno. In the next section, we propose a new congestion control mechanism of TCP that alleviates the problems of HighSpeed TCP.



(a) μ_{Reno}=100 [Mbps]



(b) μ_{Reno}=12 [Mbps]



(c) μ_{Reno}=1.5 [Mbps]

Figure 6: Analysis Results

5 Gentle HighSpeed TCP

In this section, we propose a new congestion control mechanism, called *Gentle HighSpeed TCP*, to overcome the problems of HighSpeed TCP described in the previous sections. Gentle HighSpeed TCP can improve throughput of both TCP Reno connections and HighSpeed TCP connections sharing the network bandwidth. The effectiveness of Gentle HighSpeed TCP is then confirmed by simulation experiments.

5.1 Algorithm

Figure 7 shows the state transition diagram of TCP Reno. As explained in Subsection 2.1, TCP Reno has two phases of slow start phase and congestion avoidance phase, and shifts its phase based on packet loss events and the change of the congestion window. Similarly, Gentle HighSpeed TCP has two phases. The major differences are the initial slow start phase and the congestion avoidance phase. In the initial slow start phase, to avoid the degradation of throughput after bursty packet losses, Gentle HighSpeed TCP moves its state to congestion avoidance phase just after receiving duplicate ACK packets without waiting a retransmission timeout expiration. In the congestion avoidance phase, it consists of two *modes*. Each phases/modes has a different algorithm of increasing/decreasing congestion window size. Figure 8 shows the state transition diagram of Gentle HighSpeed TCP. Gentle HighSpeed TCP estimates the network congestion level based on packet loss and changes in RTTs of transmitted packets, and shifts its state based on the estimation. In what follows, we explain detailed algorithm of Gentle HighSpeed TCP in all phases and modes.

5.1.1 Slow Start Phase

Many studies have been done on avoiding the burst packet losses in the initial slow start phase shown in Section 3 and various methods have been proposed [24, 26]. But, most of those methods have essential problems in setting parameters to work effectively. The difficulty is that the appro-

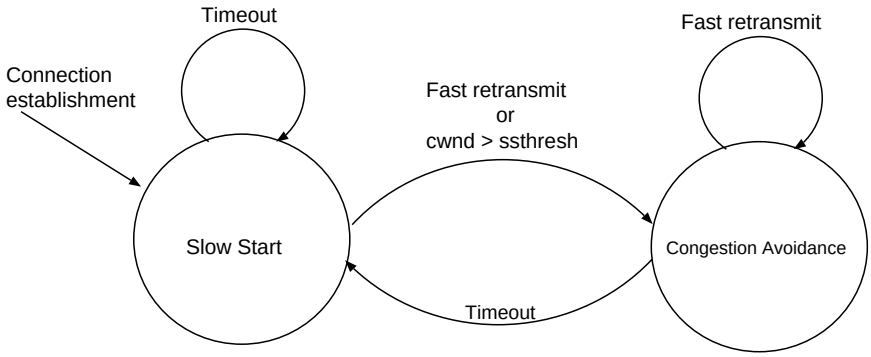


Figure 7: State Transition Diagram of TCP Reno

priate parameter set significantly depends on network environment such as RTT, link bandwidth, buffer size at the router buffer, number of competing connections, and so on. Needless to say, the objective of increasing the congestion window size exponentially at the initial slow start phase is to find the available bandwidth of the traversing path quickly. In general, however, it is difficult to achieve the two opposite goals at the initial slow start phase, namely, to find the available bandwidth quickly and to avoid burst packet losses at the bottleneck router, since the sender host has little information about the network at the beginning of the data transmission.

Instead of trying to avoid burst packet losses at the initial slow start phase, therefore, we focus on avoiding waiting retransmission timeout after burst packet losses and resuming the data transmission quickly. When a sender host receives duplicate ACK packets at the initial slow start phase, it changes its $ssthresh$ value and congestion window size to one half and one fourth of the current congestion window size, respectively, and it restarts its packet transmission without waiting the retransmission timer expiration, from the packet requested to be retransmitted by the duplicate ACK packets. The reasons of one half for $ssthresh$ value and one forth for the congestion window size, instead of one half for $ssthresh$ and one packet for the congestion window size in the traditional TCPs, are based on the following discussion.

We assume that a TCP receiver host sends an ACK packet in receiving a data packet from a

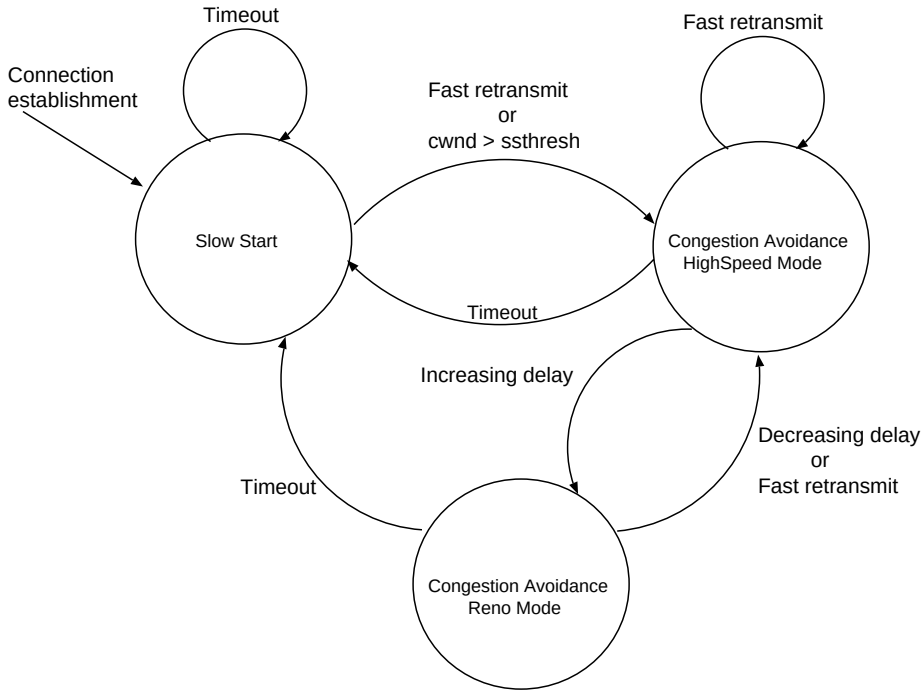


Figure 8: State Transition Diagram of Gentle HighSpeed TCP

TCP sender host, and that the available bandwidth of the network remains stable. Then, the arrival rate of ACK packets at the TCP sender host becomes almost equal to the packet processing speed of the bottleneck router of the forward path of the TCP connection. Let us consider the case where the TCP sender host receives duplicate ACK packets when the congestion window size is W in the slow start phase. Since the sender TCP sends two packets on receiving an ACK packet in the slow start phase as explained in Section 2, it is considered that $W/2$ out of W packets are queued at bottleneck router buffer. Consequently, the size of the router buffer is smaller than $W/2$ since the sender host experiences packet losses in this cycle. Furthermore, looking at the previous cycle, we can guess that $W/2$ packets are successfully transmitted since the window size has increased to W. Then, by applying the same discussion as above, we can estimate that the router buffer size is larger than $W/4$. Consequently, by setting the $ssthresh$ value to $W/2$ and the congestion window size to $W/4$, we can obtain larger throughput without packet losses.

Gentle HighSpeed TCP shifts its phase from slow start phase to congestion avoidance phase when the congestion window size becomes larger than the $ssthresh$ value as TCP Reno does. In addition to this, it quits the slow start phase when it receives an ACK packet for a packet at the beginning of a window before completing transmission of the packets in the window.

5.1.2 Congestion Avoidance Phase

The original HighSpeed TCP increases its congestion window size dependently only on the current size of the congestion window. It causes bursty packet losses because it continues increasing the congestion window size even when the packets in the window begin to be queued at the router buffer. Furthermore, the difference of the increasing speed of the congestion window makes unfairness between HighSpeed TCP and TCP Reno. Therefore, we consider that the ideal mechanism is to change the behavior of TCP according to whether the bottleneck link is fully utilized of not. Our proposed mechanisms, called *Gentle HighSpeed TCP* realizes such behavior. That is, when the utilization of the bottleneck link is under 100%, Gentle HighSpeed TCP uses the same algorithm as the original HighSpeed TCP. When the bottleneck link is fully utilized, on the other hand, Gentle HighSpeed TCP behaves equally to TCP Reno. In what follows, we explain the detailed algorithm of the two modes of Gentle HighSpeed TCP, which is HighSpeed mode and Reno mode, and the algorithm how to change its mode.

HighSpeed mode

When the bottleneck link utilization is under 100%, the sender host uses the HighSpeed mode, where the algorithm of changing the congestion window size is the same as that of the original HighSpeed TCP. Therefore, the window size increases rapidly, which results in that it can utilize the network bandwidth quickly and effectively. Furthermore, it judges whether the network bandwidth is fully utilized, and determines that it shifts its mode to Reno mode or it remains in HighSpeed mode. To estimate the bottleneck link utilization, Gentle HighSpeed TCP uses RTT values of transmitted packets. When the measured RTTs indicate an increasing trend, it determines

whether the link bandwidth is fully utilized, and vice versa. The algorithm is quite simple: Define RTT values and departure times of n transmitted packets as $d_1, \cdots, d_n$ and $t_1, \cdots, t_n$, respectively, and check a correlation between $d_1, \cdots, d_n$ and $t_1, \cdots, t_n$ by statistical test. If a positive correlation is recognized, we determine that there is an increasing trend in the observed RTT values and move to the Reno mode.

When a packet dropping event is detected and retransmitted by fast retransmit algorithm, the sender host remains in HighSpeed mode. When a retransmission timeout occurs, it halves the $ssthresh$ value, reset the congestion window size to one packet, and change its phase to slow start phase described in Subsection 5.1.1.

Reno mode

When Gentle HighSpeed TCP operates in the Reno mode, it is considered that the network bandwidth is fully utilized and packets are going to be queued at the bottleneck router. One possible way to keep the link utilization is to stop increasing or to decrease the congestion window size like TCP Vegas [27]. However, it was reported that TCP Vegas can not keep fairness with competing TCP Reno connections since it is too conservative compared with TCP Reno, which continues increasing the congestion window size [17, 28, 29]. Consequently, our solution is to make Gentle HighSpeed TCP behave identically with TCP Reno. It is a very straightforward idea to keep fairness with TCP Reno, but gives quite good results as shown in the next subsection.

The behavior in facing packet losses is identical to that in the HighSpeed mode. In Reno mode the sender host also measures RTT values of transmitted packets, and checks their changing trend by the same method as in HighSpeed mode. When a decreasing trend is recognized by statistical test, the sender host changes its mode to HighSpeed mode since it is considered that the link utilization becomes lower than 100%.

5.2 Performance Evaluation

In this subsection, we evaluate the performance of Gentle HighSpeed TCP through simulation experiments. The simulation environments here are the same as in Section 3 except that hstcp$_1$, . . . , and hstcp$_{N_{HS}}$ hosts in Fig. 1 use Gentle HighSpeed TCP. Figure 9 shows the total throughput of Gentle HighSpeed TCP and TCP Reno connections when N_{Reno} = 10 and the number of Gentle HighSpeed TCP N_{HS} is changed from 0 to 3. From this figure, Gentle HighSpeed TCP can utilize almost 100% bandwidth of the link. It is because Gentle HighSpeed TCP behaves identically to TCP Reno when the queue length at the bottleneck router buffer begins to increase. As a result of that, the number of lost packets in a buffer overflow is significantly decreased since TCP Reno increases its congestion window size by 1 packet every RTT, which successfully avoids retransmission timeouts. It can be confirmed from Fig. 10, which shows the change in packet loss rate of the TCP connections. We can clearly see that the packet loss rate dramatically decreases by introducing our Gentle HighSpeed TCP.

Moreover, as compared with the simulation results of HighSpeed TCP shown in Fig. 2, Gentle HighSpeed TCP does not degrade the throughput of existing TCP Reno connections. This is because Gentle HighSpeed TCP connections compete equally with the TCP Reno connections when the bottleneck link is fully utilized. This characteristics of Gentle HighSpeed TCP can be also confirmed from Fig. 11, where we show the simulation results under the same network model as that in Fig. 6 for Gentle HighSpeed TCP. It can be clearly observed that our Gentle HighSpeed TCP does not steal the throughput of a TCP Reno connection while keeping 100% utilization of the bottleneck link, regardless of the propagation delay of the bottleneck link and the access link bandwidth of the TCP Reno connection.

From the above simulation results, we conclude that Gentle HighSpeed TCP proposed in this thesis can fully utilize the bottleneck link and keep good fairness with TCP Reno connections.

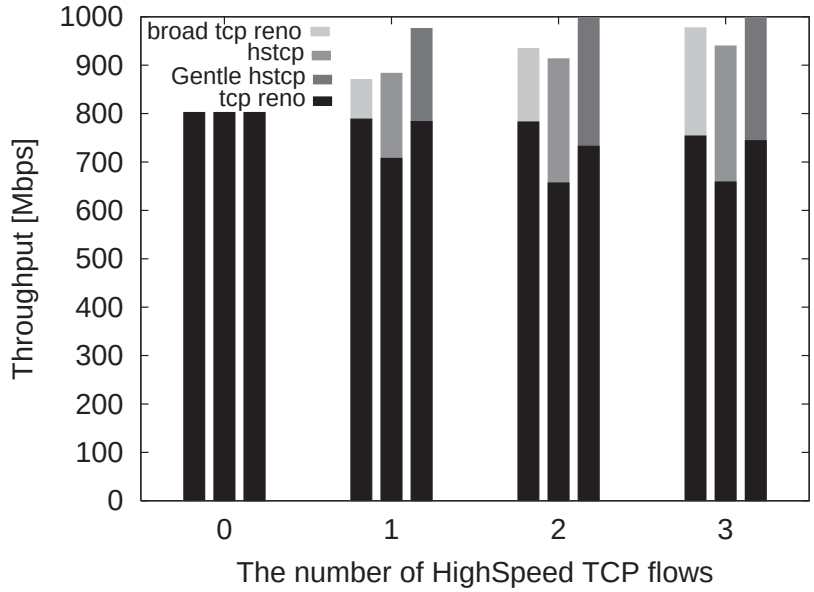


Figure 9: Effect of Number of Gentle HighSpeed TCP Connections

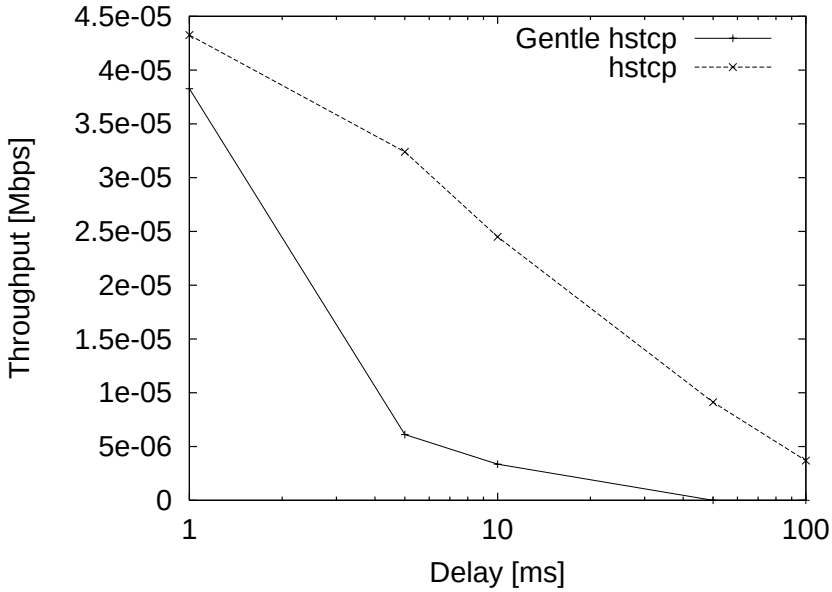
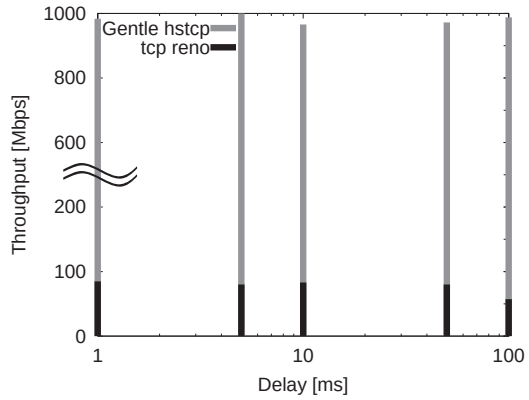
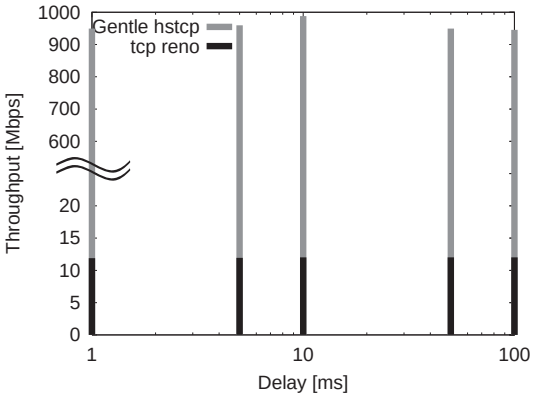


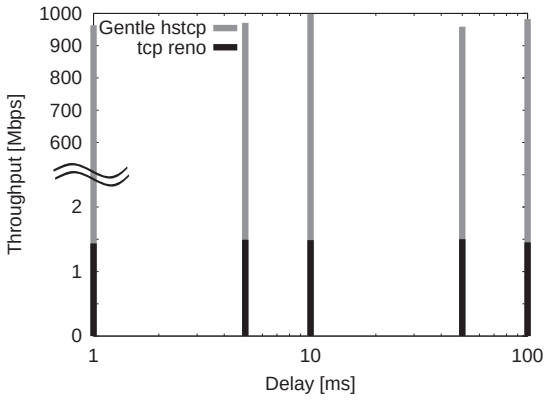
Figure 10: Comparison of Packet Loss Probability



(a) μ_{Reno}=100 [Mbps]



(b) μ_{Reno}=12 [Mbps]



(c) μ_{Reno}=1.5 [Mbps]

Figure 11: Effect of Propagation Delay

6 Conclusion

In this thesis, we have first evaluated the characteristics of HighSpeed TCP in terms of both throughput and fairness with TCP Reno, through simulation and mathematical analysis studies. The results are that HighSpeed TCP can provide larger throughput than TCP Reno, but it cannot fully utilize the link bandwidth because of bursty packet losses at the router buffer. It has been also shown that HighSpeed TCP degrades the throughput of TCP Reno when they share the bottleneck link. Moreover, the problem of bursty packet losses at the initial slow start phase has been pointed out. To overcome these problems, we have proposed a new congestion control mechanism, called Gentle HighSpeed TCP, which has the enhanced mechanisms in both of slow start and congestion avoidance phase. We have evaluated the performance of Gentle HighSpeed TCP through simulation experiments, and confirmed that Gentle HighSpeed TCP can fully utilize the network bandwidth and keep better fairness against existing TCP Reno connections.

Acknowledgements

I would like to express my sincere appreciation and deep gratitude to my advisor, Professor Masayuki Murata of Osaka University, who introduced me to the area of computer networks including the subjects in the thesis and advice throughout my studies and during preparation of this manuscript.

This thesis would not been possible without the support of Associate Professor Go Hasegawa of Osaka University. He has been constant sources of encouragement and advice throughout my studies and in the preparation of this thesis. It gives me great pleasure to acknowledge his support.

Thanks are also due to Professor Hideo Miyahara and Professor Shinji Shimojo of Osaka University who gave me a great deal of valuable advice.

I am also indebted to Associate Professor Masashi Sugano of Osaka Prefecture College of Health Sciences, Associate Professor Ken-ichi Baba of Osaka University, Associate Professor Naoki Wakamiya of Osaka University, Associate Professor Hiroyuki Ohsaki of Osaka University and Research Assistant Shinichi Arakawa of Osaka University for their many helpful comments and feedbacks.

I also wish to express my heartfelt thanks to many friends and colleagues in the Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University for their generous, enlightening and valuable suggestions and advice.

Finally, I am deeply grateful to my parents. They have always stood behind me and supported me.

References

[1] M. Murata, "Challenges for the next-generation Internet and the role of IP over photonic networks," *IEEE Transactions on Communications*, vol. E83-B, pp. 2153–2165, Oct. 2000.

[2] J. L. Wei, C.-D. LIU, S.-Y. PARK, K. H. LIU, R. S. Ramamurthy, H. KIM, and M. W. MAEDA, "Network control and management for the next generation Internet," *IEEE Transactions on Communications*, vol. E83-B, pp. 2191–2209, Oct. 2000.

[3] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Reading, Massachusetts: Addison-Wesley, 1994.

[4] V. Jacobson and M. J. Karel, "Congestion avoidance and control," in *Proceedings of ACM SIGCOMM '88*, pp. 314–329, Nov. 1988.

[5] "Global Grid forum." available at `http://www.gridforum.org/`.

[6] B. phillips, "Have storage area networks come of age?," *IEEE Computer*, vol. 31, pp. 10–12, July 1998.

[7] J. J. Bunn, J. C. Doyle, S. H. Low, H. B. Newman, and S. M. Yip, "Ultrascale network protocols for computing and science in the 21st century," *White Paper of United States of America*, Sept. 2002.

[8] S. Floyd, "HighSpeed TCP for large congestion windows," *Internet Draft draft-floyd-tcp-highspeed-01.txt*, Aug. 2002.

[9] E. de Souza, "Simulation study of proposed HighSpeed TCP for large congestion windows." available at `http://www-itg.lbl.gov/~evandro/hstcp/`.

[10] G. Hasegawa, M. Murata, and H. Miyahara, "Fairness and stability of congestion control mechanisms of TCP," *Telecommunication Systems Journal*, vol. 15, pp. 167–184, Nov. 2000.

[11] L. Guo and I. Matta, “The war between mice and elephants,” *Technical Report BU-CS-2001-005*, May 2001.

[12] K. Tokuda, G. Hasegawa, and M. Murata, “Analysis and improvement of fairness between long-lived and short-lived TCP connections,” in *Proceedings of IEEE PfHSN 2002*, pp. 151–158, Apr. 2002.

[13] K. Tokuda, G. Hasegawa, and M. Murata, “TCP throughput analysis with variable packet loss probability for improving fairness among long/short-lived TCP connections,” in *Proceedings of IEEE CQR 2002*, pp. 52–56, May 2002.

[14] P. Hurley, J.-Y. L. Boudec, and P. Thiran, “A note on the fairness of additive increase and multiplicative decrease,” in *Proceedings of 16th International Teletraffic Congress*, pp. 336–350, June 1999.

[15] J. Martin, A. Nilsson, and I. Rhee, “The incremental deployability of RTT-based congestion avoidance for high speed TCP Internet connections,” in *Proceedings of ACM SIGMETRICS ’2000*, pp. 134–144, June 2000.

[16] A. Veres and M. Boda, “The chaotic nature of TCP congestion control,” in *Proceedings of IEEE INFOCOM ’2000*, Mar. 2000.

[17] O. Ait-Hellal and E. Altman, “Analysis of TCP Vegas and Reno,” *Journal of Telecommunication Systems*, vol. 15, no. 3,4, pp. 381–404, 2000.

[18] Robert Morris, “TCP behavior with many flows,” in *Proceedings of IEEE International Conference on Network Protocols*, October 1997.

[19] D.-M. Chiu and R. Jain, “Analysis of the increase and decrease algorithms for congestion avoidance in computer networks,” *Journal of Computer Networks and ISDN Systems*, pp. 1–14, June 1989.

[20] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 27, pp. 67–82, July 1997.

[21] The VINT Project, "UCB/LBNL/VINT Network Simulator - ns (Version 2)." available at `http://www.isi.edu/nsnam/ns/`.

[22] J. C. Hoe, "Start-up dynamics of TCP's congestion control and avoidance schemes," *Master Thesis, Massachusetts Institute of Technology*, June 1995.

[23] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," *Computer Communication Review*, vol. 26, pp. 5–21, July 1996.

[24] S. Floyd, "Limited slow-start for TCP with large congestion windows," *Internet Draft draft-floyd-tcp-slowstart-01.txt*, Aug. 2002.

[25] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, Aug. 1993.

[26] J. C. Hoe, "Improving the start-up behavior of a congestion control scheme of TCP," *ACM SIGCOMM Computer Communication Review*, vol. 26, pp. 270–280, Oct. 1996.

[27] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proceedings of IEEE SIGCOMM '94*, pp. 24–35, 1994.

[28] G. Hasegawa, K. Kurata, and M. Murata, "Analysis and improvement of fairness between TCP Reno and Vegas for deployment of TCP Vegas to the Internet," in *Proceedings of IEEE ICNP 2000*, pp. 177–186, Nov. 2000.

[29] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCP Reno and Vegas," in *Proceedings of IEEE INFOCOM '99*, pp. 1556–1563, Mar. 1999.