

PAPER

# Performance Analysis and Improvement of HighSpeed TCP with TailDrop/RED Routers

Zongsheng ZHANG<sup>†a)</sup>, Go HASEGAWA<sup>†b)</sup>, and Masayuki MURATA<sup>†c)</sup>, *Members*

**SUMMARY** Continuous and explosive growth of the Internet has shown that current TCP mechanisms can obstruct efficient use of high-speed, long-delay networks. To address this problem we propose an enhanced transport-layer protocol called gHSTCP, based on HighSpeed TCP proposed by Sally Floyd. It uses two modes in the congestion avoidance phase based on 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/ARED routers can not alleviate the problem completely. Therefore, we present a modified version of Adaptive RED, called gARED, directed at the problem of simultaneous packet drops by multiple flows in high speed networks. gARED can eliminate weaknesses found in Adaptive RED 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 Adaptive RED with HighSpeed TCP.

**key words:** *TCP Reno, HighSpeed TCP, fairness, TailDrop, RED*

## 1. Introduction

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 [1], cannot achieve sufficient throughput for this kind of high-speed data transmission because of the essential nature of the TCP congestion control mechanism.

According to [2], in order for a TCP Reno connection, with a packet size of 1,500 bytes and RTT (Round Trip Time) of 100 ms, to fill a 10 Gbps link, a congestion window of 83,333 packets is required. This means a packet loss rate of less than  $2 \times 10^{-10}$ , well below what is possible with present optical fiber and router

technology. Furthermore, when packets are lost in the network, 40,000 RTTs (about 4,000 sec) are needed to recover throughput. As a result, standard TCP cannot possibly obtain such a large throughput, primarily because TCP Reno drastically decreases its congestion window size when packet loss is taking place, increases it only very slightly when experiencing no packet loss.

HighSpeed TCP (HSTCP) [2] was recently proposed as one way to overcome the problems discussed above 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 [3]. 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 intuitively HSTCP appears 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 literature [4]–[9]. 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 [2] 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 our knowledge, there has been limited research

<sup>†</sup>The author is with Graduate School of Information Science and Technology, Osaka University, Japan

a) E-mail: zhang@ist.osaka-u.ac.jp

b) E-mail: hasegawa@ist.osaka-u.ac.jp

c) E-mail: murata@ist.osaka-u.ac.jp

on this issue [10]–[12]. In [10], [11], only simulations or results from experimental implementations are assessed. In [12], the author addresses “a serious RTT unfairness problem.” In this paper we evaluate throughput and fairness properties when HSTCP and TCP Reno connections share a network bandwidth. From the results we observe that HSTCP can achieve high throughput, but it is accompanied by a large degradation in TCP Reno throughput. To resolve this problem, we propose a modification to HSTCP called “gentle HighSpeed TCP” (gHSTCP) that implements two modes, HSTCP mode and Reno mode, in the congestion avoidance phase to improve fairness yet allow both gHSTCP and traditional TCP to achieve satisfactory performance. The simulation results show that gHSTCP can achieve both higher throughput and better fairness than HSTCP.

However, the performance improvement 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 [13] 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) [14] has been recommended for wide deployment in the Internet as an active queue management mechanism [15]. However, control parameter settings in RED have been proven highly sensitive to the network scenario, and misconfiguring RED can degrade performance significantly [16]–[20]. Adaptive RED (ARED) was therefore proposed as a solution to these subsequent problems [21]. 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.

The remainder of this paper is organized as follows. In Section 2 we give a brief overview of HSTCP and review some related works on TCP variants for high speed networks. In Section 3, we investigate, through simulations, the throughput and fairness properties of HSTCP

when sharing bandwidth with TCP Reno on a bottleneck link. We then propose a modification to HSTCP. In Section 4, we analyze and evaluate ARED, show its weaknesses, propose an improved version of ARED and then conduct simulation experiments to evaluate the proposed mechanisms. Section 5 assesses the packet loss rate when high-speed flows compete the resource with web traffic. Finally, Section 6 presents our conclusions of this paper.

## 2. Background

### 2.1 HSTCP (HighSpeed TCP)

To overcome problems with TCP mentioned in Section 1, HSTCP was proposed in [2]. 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:

$$a(w) = \frac{2w^2 \cdot b(w) \cdot p(w)}{2 - b(w)} \quad (1)$$

$$b(w) = (b_{high} - 0.5) \frac{\log(w) - \log(W_{low})}{\log(W_{high}) - \log(W_{low})} + 0.5 \quad (2)$$

$$p(w) = \frac{0.078}{w^{1.2}} \quad (3)$$

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

According to Equations (1) and (2) and a typical parameter set used in [2] ( $b_{high}$ ,  $W_{high}$  and  $W_{low}$  are 0.1, 83,000 and 38, respectively), Fig. 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 window size increases. In this manner, HSTCP can sustain a large congestion window and fully utilize the high-speed long-delay network.

The HSTCP response function<sup>†</sup> (3) is illustrated

<sup>†</sup>The TCP response function maps the steady-state packet drop rate to the TCP average sending rate in packets per RTT.

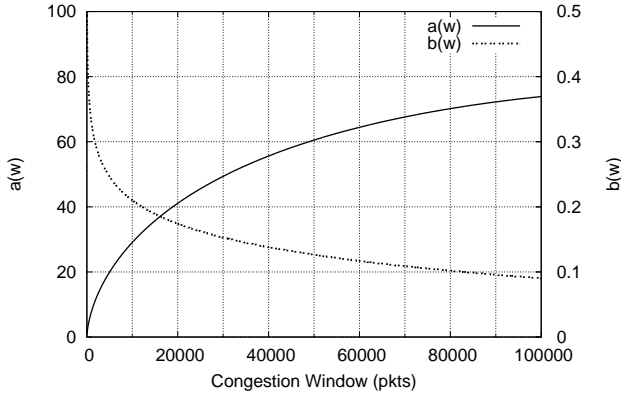


Fig. 1 AIMD Parameters in HSTCP

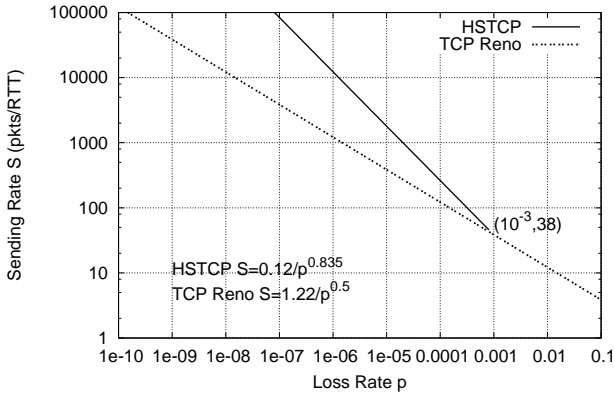


Fig. 2 Response Function of TCP Reno and HSTCP

in Fig. 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.

## 2.2 Related Work

There are other solutions for overcoming the limitations of standard TCP in high-speed networks.

- Scalable TCP [22]. This 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 [2].

- FAST TCP [23]. This protocol is based on TCP-Vegas [24] 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 [25].
- XCP [26]. This 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 is 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 paper. 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. In our proposal, gHSTCP can achieve good performance even without gARED. The performance of gHSTCP will become better if gARED can be deployed at routers.

## 3. gHSTCP: Gentle HighSpeed TCP

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.

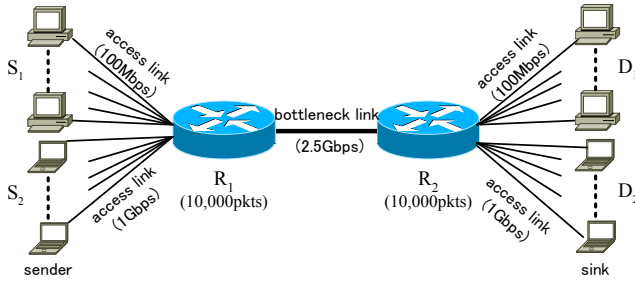


Fig. 3 Simulation Topology

### 3.1 Simulation with HSTCP

We first present the results of simulation experiments to clarify HSTCP problems with throughput and fairness. *ns-2* network simulator [27] is used for the simulations. The network topology is shown in Fig. 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 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 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 [28]. 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. Future work will be conducted to find out how much difference exists between homogeneous and heterogeneous environments.

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 paper. 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 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, as mentioned in Section 1.

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 timeouts (this is a fundamental mechanism of TCP Reno [29]), and timeouts place 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 [28], combined with a selective retransmission policy, can help overcome limitations in recovering from many packet losses. Table 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.

### 3.2 gHSTCP Description

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 this problem, 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, 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  $r$  [30]:

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

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  $\rho$ . To make inference about  $\rho$  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:

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

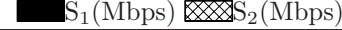
Once a retransmission timeout occurs, or duplicated acknowledgments are received, the sender decreases the congestion window in the same way as HSTCP does. When a timeout occurs, the congestion window size is reset to one packet and the phase is changed to slow-start. When a packet loss event is detected 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) 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.

### 3.3 gHSTCP Evaluation with Simulations

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  $S_2$ .

**Table 1** Performance of HSTCP with DropTail

Case	S <sub>1</sub> group	S <sub>2</sub> group	Delay (ms)	Average throughput of S <sub>1</sub> /S <sub>2</sub> 	Confident Interval <sup>a</sup>	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

<sup>a</sup> It is for the total average throughput, the confidence level is 95%.

- Case 7: TCP Reno+SACK is used for S<sub>1</sub> and gHSTCP+SACK is used for S<sub>2</sub>.

The results are shown in Table 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<sub>2</sub> 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<sub>1</sub> group is greatly improved comparing with that in Case 4.

Tables 1 and 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/gHSTCP and TCP Reno flows as the bottleneck link delay become larger. In this situation, HSTCP/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/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/gHSTCP connections increase the size of their congestion windows quickly.

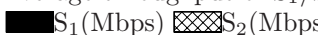

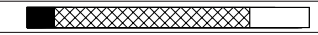
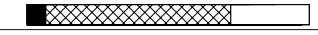






To improve network performance in terms of link utilization and system fairness, it has been proposed that Active Queue Management (AQM) such as RED be deployed in the Internet [15]. In contrast to

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 [14]. 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 [31], 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  $max_p = 0.1$ ). The following simulation experiments are performed:

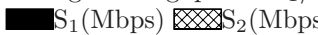


















- Case 8: TCP Reno is used for S<sub>1</sub> and HSTCP is used for S<sub>2</sub> with RED deployed.
- Case 9: TCP Reno is used for S<sub>1</sub> and HSTCP+SACK is used for S<sub>2</sub> with RED deployed.
- Case 10: TCP Reno+SACK is used for S<sub>1</sub> and HSTCP+SACK is used for S<sub>2</sub> with RED deployed.
- Case 11: TCP Reno is used for S<sub>1</sub> and gHSTCP is used for S<sub>2</sub> with RED deployed.
- Case 12: TCP Reno is used for S<sub>1</sub> and gHSTCP+SACK is used for S<sub>2</sub> with RED deployed.
- Case 13: TCP Reno+SACK is used for S<sub>1</sub> and gHSTCP+SACK is used for S<sub>2</sub> with RED deployed.

Comparing the results of Table 3 for Cases 8–13 with Tables 1 and 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

**Table 2** Performance of gHSTCP with DropTail

Case	S <sub>1</sub> group	S <sub>2</sub> group	Delay (ms)	Average throughput of S <sub>1</sub> /S <sub>2</sub> 	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

**Table 3** Performance of HSTCP/gHSTCP with RED ( $max_p = 0.1$ )

Case	S <sub>1</sub> group	S <sub>2</sub> group	Delay (ms)	Average throughput of S <sub>1</sub> /S <sub>2</sub> 	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

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 probability is quite small. This results in timeouts if the SACK option is not used for high-speed flow. Fig. 4 shows the change in the congestion window when HSTCP/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/gHSTCP flows, though the congestion windows will not be reset to 1 packet as shown in Fig. 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.

In addition, it is reported [31] 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/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 [16]–[20]. 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<sub>1</sub> and HSTCP is used for S<sub>2</sub> with RED ( $max_p = 0.001$ ).
- Case 15: TCP Reno is used for S<sub>1</sub> and

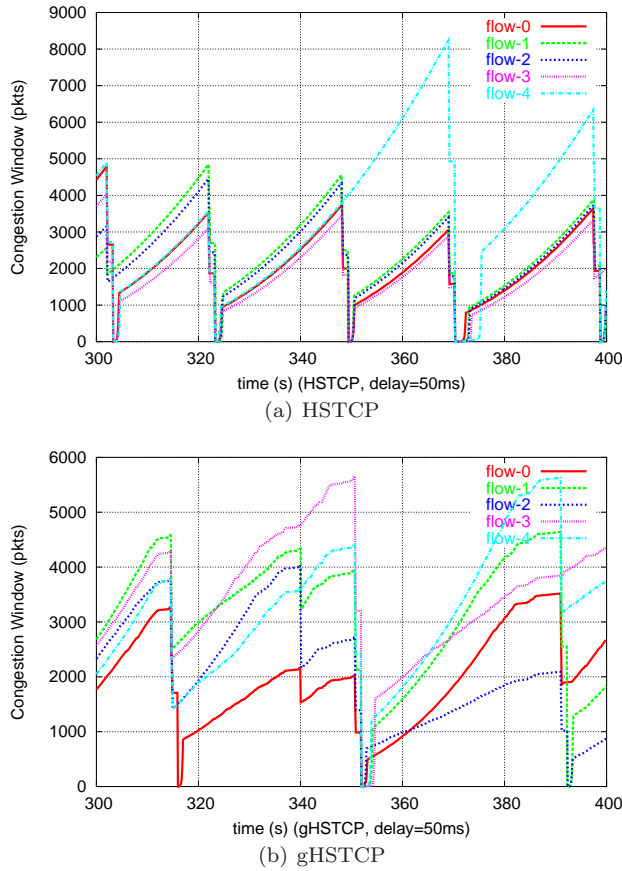


Fig. 4 Congestion Window (HSTCP/gHSTCP with RED)

- 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 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 [16]–[20]. In the next section, an additional mechanism will be introduced to address this problem.

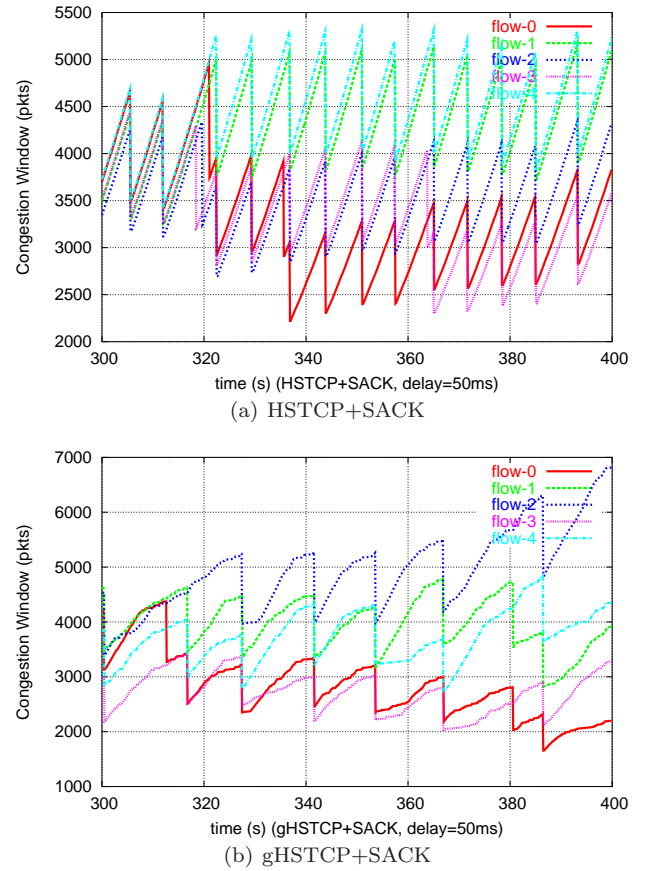


Fig. 5 Congestion Window (HSTCP+SACK/gHSTCP+SACK with RED)

#### 4. gARED: Gentle Adaptive RED

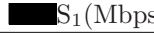
The results in Section 3 are primarily the effects of the TailDrop and RED mechanisms at the bottleneck routers. We observed that  $max_p$  is an important parameter that significantly affects system performance when RED is deployed. We need a mechanism that can adjust the parameters automatically, especially  $max_p$ , in response to the network environment. Adaptive RED (ARED) [21], an improved version of RED, is such a mechanism, and its application is expected to improve system performance. We first conduct simulation experiments with ARED and reveal its shortcomings from the results. We then propose a modification to alleviate these deficiencies, through a process of automatic parameter setting, but that still preserves the effectiveness of the ARED mechanism, especially aiming at the absence of the SACK option.

##### 4.1 ARED Mechanism

RED monitors impending congestion by maintaining an exponential weighted moving average of the queue length ( $\bar{q}$ ). However, RED parameter settings have



**Table 4** Performance of HSTCP/gHSTCP with RED ( $max_p = 0.001$ )

Case	S <sub>1</sub> group	S <sub>2</sub> group	Delay (ms)	Average throughput of S <sub>1</sub> /S <sub>2</sub> 	Confident Interval	Utilization (%)	Fairness
8	Reno	HSTCP	25		2.643	97.193	0.985
			50		14.792	94.621	0.950
			100		29.052	87.515	0.947
9	Reno	HSTCP+SACK	25		1.634	97.483	0.981
			50		2.891	96.377	0.934
			100		6.305	93.250	0.829
10	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
11	Reno	gHSTCP	25		0.849	97.435	1.000
			50		2.516	96.615	1.000
			100		24.977	92.151	0.987
12	Reno	gHSTCP+SACK	25		0.767	97.455	1.000
			50		1.738	97.145	0.996
			100		3.384	94.022	0.951
13	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

proven to be highly sensitive to network conditions, and performance can suffer significantly for a misconfigured RED [16], [17]. 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 [21].

- $max_p$ : In RED, this value 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).
- $max_{th}$ : 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.
- $w_q$ : This parameter 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

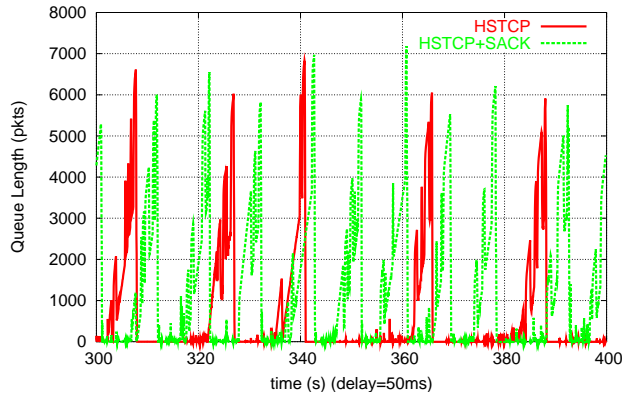
to  $1 - exp(-1/C)$ , where  $C$  is the link capacity in packets/second. The intent here is to maintain the time constant on the order of RTT. Calculating the average queue size is the basis of the RED algorithm.

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.

## 4.2 Simulation with ARED

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 section 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 20: TCP Reno is used for S<sub>1</sub> and HSTCP is used for S<sub>2</sub> with ARED deployed.
- Case 21: TCP Reno is used for S<sub>1</sub> and HSTCP+SACK is used for S<sub>2</sub> with ARED deployed.
- Case 22: TCP Reno+SACK is used for S<sub>1</sub> and HSTCP+SACK is used for S<sub>2</sub> with ARED deployed.
- Case 23: TCP Reno is used for S<sub>1</sub> and gHSTCP is used for S<sub>2</sub> with ARED deployed.
- Case 24: TCP Reno is used for S<sub>1</sub> and gHSTCP+SACK is used for S<sub>2</sub> with ARED deployed.
- Case 25: TCP Reno+SACK is used for S<sub>1</sub> and gHSTCP+SACK is used for S<sub>2</sub> with ARED de-



**Fig. 6** Instantaneous Queue Length (HSTCP/HSTCP+SACK with ARED)

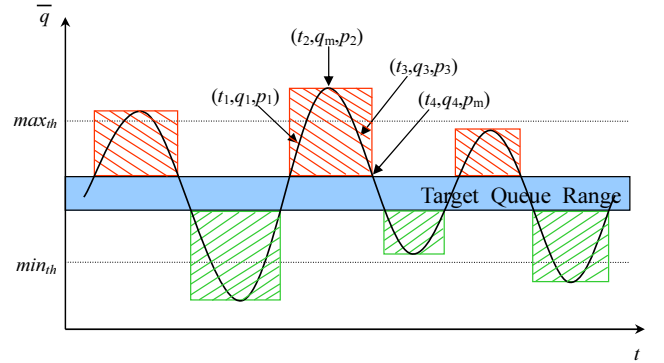
played.

The results are shown in Table 5. System performance is improved in terms of throughput and fairness compared with that of RED (Table 3). However, the bottleneck link remains under-utilized. Fig. 6 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.

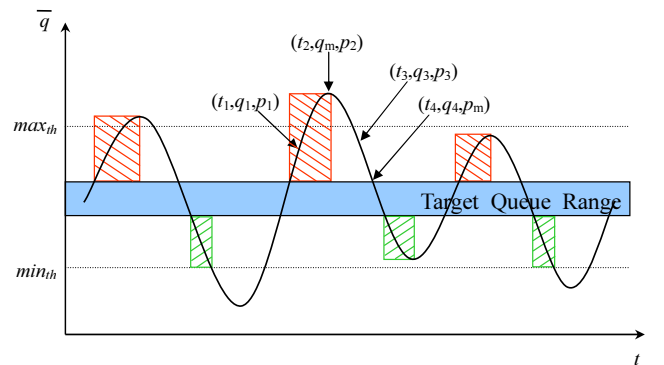
The graph in Fig. 7 shows a sketch map of the average queue size as it varies over time when using ARED. The purpose of the changing  $max_p$  is to maintain an average queue size within the target queue range. In the figure, the x-axis is time, the y-axis is the average queue length. When the average 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 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.

We can view this process as a feedback control system [19] with the TCP senders as the controlled element, the drop module as the controlling element and the drop probability as the feedback signal. The feed-



**Fig. 7** Sketch of Average Queue Length (ARED)



**Fig. 8** Sketch of Average Queue Length (gARED)

back 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.

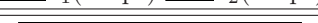
Another problem with ARED, the same as with RED, is that the lower bound of parameter  $max_p$  is determined to some extent by the network manager to ensure ARED performance.

### 4.3 An Improvement of ARED

To solve these problems inherent to ARED, we propose a modified version referred to gARED as shown in Fig. 8. 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

**Table 5** Performance Comparison of HSTCP/gHSTCP with ARED

Case	S <sub>1</sub> group	S <sub>2</sub> group	Delay (ms)	Average throughput of S <sub>1</sub> /S <sub>2</sub> 	Confident Interval	Utilization (%)	Fairness
20	Reno	HSTCP	25		10.075	91.465	0.995
			50		24.128	81.059	0.992
			100		20.574	70.070	0.976
21	Reno	HSTCP+SACK	25		4.323	95.813	0.970
			50		7.216	92.558	0.887
			100		7.465	89.026	0.778
22	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
23	Reno	gHSTCP	25		4.665	96.766	1.000
			50		6.549	91.555	1.000
			100		27.368	80.035	0.984
24	Reno	gHSTCP+SACK	25		0.828	96.965	1.000
			50		8.815	95.294	0.988
			100		5.446	91.502	0.897
25	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

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:

```

Every interval seconds:
  if (avg > target and avg > old_avg and
      max_p < top)
    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
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
    
```

#### 4.4 Evaluation of HSTCP/gHSTCP with gARED

Table 6 shows throughput and fairness of simulation experiments when there are 5 HSTCP/gHSTCP flows competing with 5 TCP Reno flows, and gARED is used at the routers:

- Case 26: TCP Reno is used for S<sub>1</sub> and HSTCP is used for S<sub>2</sub> with gARED deployed.

- Case 27: TCP Reno is used for S<sub>1</sub> and HSTCP+SACK is used for S<sub>2</sub> with gARED deployed.
- Case 28: TCP Reno+SACK is used for S<sub>1</sub> and HSTCP+SACK is used for S<sub>2</sub> with gARED deployed.
- Case 29: TCP Reno is used for S<sub>1</sub> and gHSTCP is used for S<sub>2</sub> with gARED deployed.
- Case 30: TCP Reno is used for S<sub>1</sub> and gHSTCP+SACK is used for S<sub>2</sub> with gARED deployed.
- Case 31: TCP Reno+SACK is used for S<sub>1</sub> and gHSTCP+SACK is used for S<sub>2</sub> with gARED deployed.

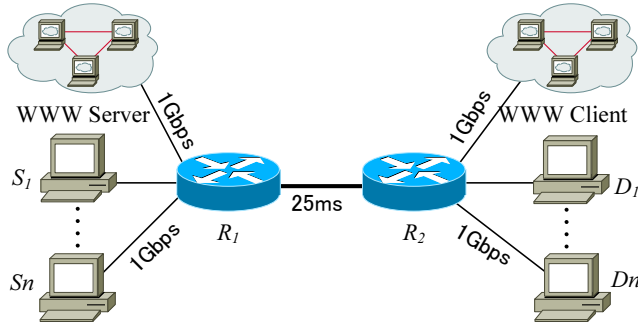
Table 6 shows the utilization is improved under gARED deployed. However, the fairness with HSTCP is not good. Especially, as delay is increased. It is because smaller packet drop rate is set by gARED for keeping the target queue length. HSTCP increases its congestion window rapidly without having consideration 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.

#### 5. Evaluation with Web-traffic

In previous sections, we evaluated the performance of HSTCP/gHSTCP in the environments where it competes the system sources with long-lived flows. A recent study [32] shows that short-lived flows such as Web traffic is one of main class applications in the In-

**Table 6** Performance Comparison of HSTCP/gHSTCP with gARED

Case	S <sub>1</sub> group	S <sub>2</sub> group	Delay (ms)	Average throughput of S <sub>1</sub> /S <sub>2</sub> ■ S <sub>1</sub> (Mbps) ▨ S <sub>2</sub> (Mbps)	Confident Interval	Utilization (%)	Fairness
26	Reno	HSTCP	25		10.496	96.761	0.976
			50		22.905	90.103	0.895
			100		185.851	79.353	0.827
27	Reno	HSTCP +SACK	25		1.591	97.543	0.980
			50		1.542	97.406	0.844
			100		2.875	95.702	0.555
28	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
29	Reno	gHSTCP	25		10.335	97.277	1.000
			50		30.243	95.227	0.995
			100		113.194	94.362	0.942
30	Reno	gHSTCP +SACK	25		2.014	97.511	1.000
			50		6.657	97.328	0.990
			100		12.327	96.307	0.824
31	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

**Fig. 9** Topology with Web-traffic

ternet. In this section, we assess the performance of HSTCP/gHSTCP when they co-exists with Web traffic.

Topology used in simulation is shown in Fig. 9. 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 is also 1 Gbps. In WWW 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 1000 client 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 [33]:

- 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.

There are 4 sets of simulation conducted:

- Case 32: The bottleneck link bandwidth is 1000 Mbps, the router buffer size is 5000 packets.
- Case 33: The bottleneck link bandwidth is 1000 Mbps, the router buffer size is 500 packets.
- Case 34: The bottleneck link bandwidth is 500 Mbps, the router buffer size is 5000 packets.
- Case 35: 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 5, 10 and 20, respectively. The results when the bottleneck link bandwidth is 1000 Mbps are illustrated in Fig. 10. Fig. 11 shows the results when the bottleneck link bandwidth is 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.

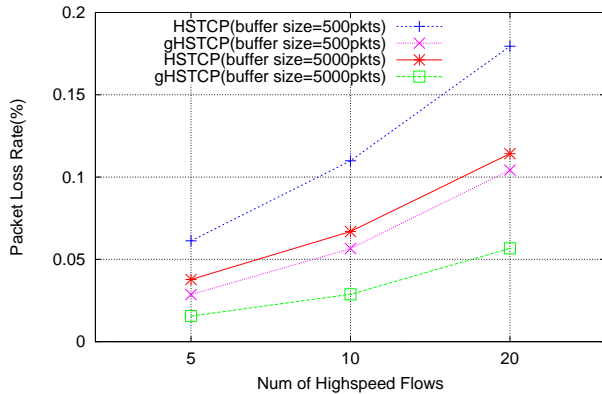


Fig. 10 Packet Loss Rate (Bandwidth=1000 Mbps)

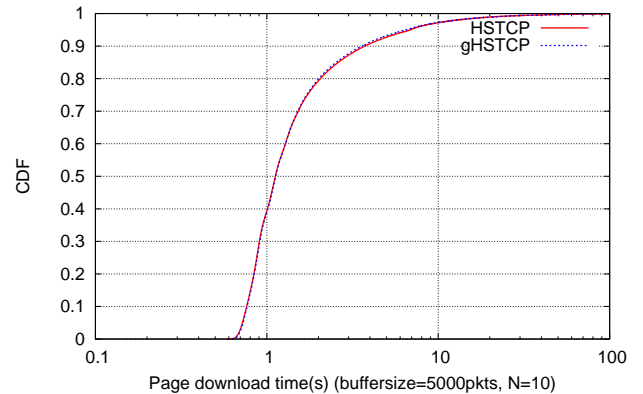


Fig. 12 Web Responding Time

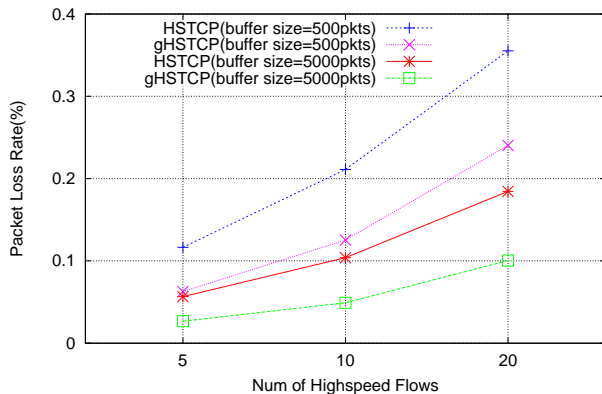


Fig. 11 Packet Loss Rate (Bandwidth=500 Mbps)

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. Fig. 12 shows a CDF (Cumulative Distribution Function) graph of web responding time when the number of high speed flow is 10, the bottleneck is 1000 Mbps and the router buffer size is 5000 packets. It is slightly improved under the circumstance when gHSTCP is used.

## 6. Conclusion

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. In addition, to address problems with ARED in high-speed long-delay networks, we also 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  $max_p$ . We showed through simulations that our proposed algorithms outperform the original algorithms. Future work will include further investigation of gHSTCP, e.g., how to recover effectively from simultaneous packet losses, refinement of the technique for making estimations based on trends.

## References

- [1] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," RFC 2581, April 1999.
- [2] S. Floyd, "HighSpeed TCP for large congestion windows," RFC 3649, December 2003.
- [3] W.R. Stevens, TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994.
- [4] 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.
- [5] 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.
- [6] R. Morris, "TCP behavior with many flows," Proceedings of IEEE International Conference on Network Protocols (ICNP), pp.205–211, October 1997.
- [7] 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.
- [8] L. Guo and I. Matta, "The war between mice and elephants," Proceedings of the 9th IEEE International Conference on Network Protocols, November 2001.
- [9] K. Avrachenkov, U. Ayesta, P. Brown, and E. Nyberg, "Differentiation between short and long TCP flows: Predictability of the response time," Proceedings of INFOCOM 2004, March 2004.
- [10] E. de Souza and D. Agarwal, "A HighSpeed TCP study: Characteristics and deployment issues," Tech. Rep. LBNL–53215, LBNL, 2003.
- [11] H. Bulot and L. Cottrell, "TCP stacks testbed," 2003. Available as: <http://www-iepm.slac.stanford.edu/bw/tcp-eval/>.
- [12] L. Xu, K. Harfoush, and I. Rhe, "Binary increase congestion control (BIC) for fast long-distance networks," Proceedings

- of INFOCOM 2004, March 2004.
- [13] 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.
  - [14] 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.
  - [15] B. Braden and et al., "Recommendations on queue management and congestion avoidance in the Internet," RFC 2309, April 1998.
  - [16] W. Feng, D.D. Kandlur, D. Saha, and K.G. Shin, "A self-configuring RED gateway," *Proceedings of INFOCOM 1999*, pp.1320–1328, March 1999.
  - [17] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons not to deploy RED," *Proceedings of 7th. International Workshop on Quality of Service (IWQoS'99)*, London, pp.260–262, June 1999.
  - [18] 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," *Proceedings of SIGCOMM 2000*, pp.151–160, September 2000.
  - [19] V. Firoiu and M. Borden, "A study of active queue management for congestion control," *Proceedings of INFOCOM 2000*, pp.1435–1444, March 2000.
  - [20] M. Christiansen, K. Jaffay, D. Ott, and F.D. Smith, "Tuning RED for web traffic," *Proceedings of SIGCOMM 2000*, pp.139–150, August 2000.
  - [21] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An algorithm for increasing the robustness of RED," 2001. Available as: <http://www.icir.org/floyd/papers/>.
  - [22] T. Kelly, "Scalable TCP: Improving performance in high-speed wide area networks," February 2003. Available as: <http://www-lce.eng.cam.ac.uk/~ctk21/scalable/>.
  - [23] 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.
  - [24] L.S. Brakmo, S.W. O'Malley, and L.L. Peterson, "TCP vegas: New techniques for congestion detection and avoidance," *SIGCOMM*, pp.24–35, August 1994.
  - [25] M. Goutelle and et al., "A survey of transport protocols other than standard TCP," February 2004. Available as: <http://www.gridforum.org/Meetings/ggf10/GGF10%20Documents/Survey%20DT-RG.pdf>.
  - [26] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," *Proceedings of SIGCOMM 2002*, August 2002.
  - [27] S. McCanne and S. Floyd, "ns Network Simulator," 2004. Available as: <http://www.isi.edu/nsnam/ns/>.
  - [28] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," RFC 2018, October 1996.
  - [29] 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.
  - [30] Mathematics group, "Faculty of Arts, Computing, Engineering and Sciences Scientific Statistics," 1998. Available as: <http://maths.sci.shu.ac.uk/distance/stats/index.html>.
  - [31] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," *Proceedings of SIGCOMM 2004*, pp.277–288, August 2004.
  - [32] S. Labs, "Packet trace analysis," 2004. Available as: <http://ipmon.sprint-labs.com/packstat/packet.php>.
  - [33] 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," *Proceedings of SIGCOMM 1999*, pp.301–313, September 1999.



**Zongsheng ZHANG** received the M.S. degree from Jilin University, China, in 1993. He is now a doctoral student at Graduate School of Information Science and Technology, Osaka University, Japan.



**Go HASEGAWA** received the M.E. and D.E. degrees in Information and Computer Sciences from Osaka University, Osaka, Japan, in 1997 and 2000, respectively. From July 1997 to June 2000, he was a Research Assistant of Graduate School of Economics, Osaka University. He is now an Associate Professor of Cybermedia Center, Osaka University. His research work is in the area of transport architecture for future high-speed networks. He is a member of the IEEE and IEICE.



**Masayuki MURATA** received the M.E. and D.E. degrees in Information and Computer Sciences from Osaka University, Japan, in 1984 and 1988, respectively. In April 1984, he joined Tokyo Research Laboratory, IBM Japan, as a Researcher. From September 1987 to January 1989, he was an Assistant Professor with Computation Center, Osaka University. In February 1989, he moved to the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University. From 1992 to 1999, he was an Associate Professor in the Graduate School of Engineering Science, Osaka University, and from April 1999, he has been a Professor of Osaka University. He moved to Advanced Networked Environment Division, Cybermedia Center, Osaka University in 2000, and moved to Graduate School of Information Science and Technology, Osaka University in April 2004. He has more than two hundred papers of international and domestic journals and conferences. His research interests include computer communication networks, performance modeling and evaluation. He is a member of IEEE, ACM, The Internet Society, IEICE and IPSJ.