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

Zongsheng ZHANG, Go HASEGAWA, and Masayuki MURATA
Graduate School of Information Science and Technology, Osaka University
{zhang,hasegawa,murata}@ist.osaka-u.ac.jp

Abstract

Continuous and explosive growth of the Internet has shown that current TCP mechanisms cannot achieve efficient utilization of networks with large bandwidth-delay products. To address this problem, we propose an enhanced transport-layer protocol called gHSTCP, which is based on HighSpeed TCP proposed by S. Floyd. By monitoring the changing trend of RTT, gHSTCP adapts to the traffic load by switching between two control modes, which is shown to provide significant performance improvement against traditional TCP Reno in terms of throughput and fairness. Furthermore, it is observed that the performance of gHSTCP is limited by both TailDrop and RED/ARED routers, thus we develop a modified adaptive RED called gARED to address the problem of simultaneous packet drops among multiple flows. By adapting to the trend in variation of the average queue length, gARED performs active queue management more effectively than ARED. Simulations show that combining gHSTCP together with gARED leads to effect utilization of network bandwidth and good fairness.

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 due to application demands as well as technological advances. 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 2×10^{-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, and, even when experiencing no packet loss, increases it only very slightly.

Focused on the above problem, HighSpeed TCP (HSTCP) [2] was proposed to achieve high throughput in networks with large bandwidth-delay products. The basic idea is to employ a different window size control mechanism in the congestion avoidance phase of the TCP mechanism [3]. Compared to TCP Reno, HSTCP increases the windows more quickly and decease it more slowly, which leads to a windows size large enough to fully utilize the link bandwidth.

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 literatures [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 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 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. In particular, when TailDrop is chosen as the queue management mechanism, gHSTCP can achieve both higher throughput and better fairness than the original HSTCP.

Furthermore, the performance improvement is limited due to the nature of TailDrop router, which causes bursty packet losses and 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 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 presents the performance evaluation. Finally, Section 5 presents our conclusions for this study.

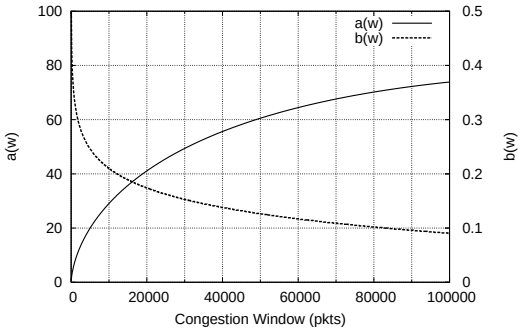

Figure 1. AIMD Parameters in HSTCP

2. Background

2.1. HSTCP (HighSpeed TCP)

To overcome problems with TCP mentioned in Section 1, HSTCP was proposed in [2]. The HSTCP algorithm employs 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 as used in [2] (b_{high}, W_{high} and W_{low} are 0.1, 83,000 and 38, respectively), Figure 1 shows how $a(w)$ and $b(w)$ vary with the congestion window. We can see that the "increase" parameter $a(w)$ becomes larger, and the "decrease" parameter $b(w)$ becomes smaller, as the congestion widow size increases. In this manner, HSTCP can sustain a large congestion window and fully utilize the high-speed long-delay network.

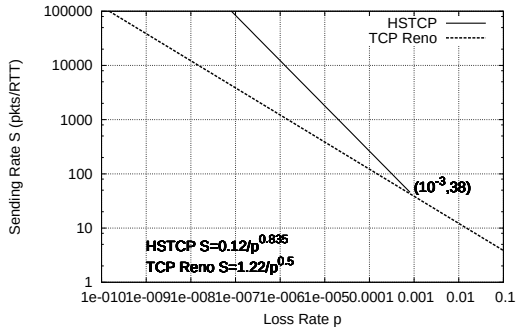

Figure 2. Response Function of TCP Reno and HSTCP

The HSTCP response function[1] (3) is illustrated in Figure 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 pkts/RTT while the sending rate of TCP Reno is around only 4,000 pkts/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 IP 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, which it modifies 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 [24].
- XCP [25]. 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.

Compared with the above proposals, HSTCP has less complexity and uses the AIMD algorithm similarly to TCP Reno. It does not require additional feedback from routers and TCP receivers and is therefore easy to deploy gradually in current networks. In this paper we propose gHSTCP. Based on HSTCP, gHSTCP can achieve better fairness with competing traditional TCP flows while extending the HSTCP strongpoint of achieving high throughput.

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 the original 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.

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 [26] is used for the simulations. The network topology is shown in Figure 3, where S_1/S_2 represents sender groups consisting of sender hosts, and D_1/D_2 represents sink groups consisting of destination hosts. R_1 and R_2 are routers with a 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, its maximum rate is about 125 Mbps. 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 HSTCP connections, we show the simulation results with and without the Selective ACKnowledgement (SACK) option. We denote HSTCP+SACK and HSTCP, respectively. TailDrop is used as the queue management mechanism in this section. We use a greedy FTP source for data transmission.

Two metrics are used: aggregate throughput and fairness (Jain's fairness index). 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

1 The TCP response function maps the steady-state packet drop rate to the TCP average sending rate in packets per RTT.

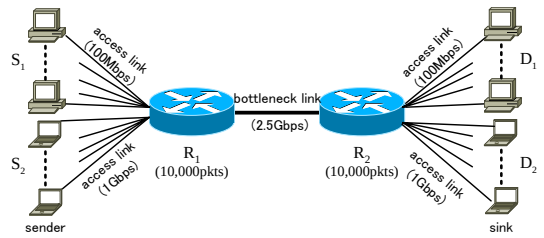

Figure 3. Simulation Topology

lies between 0 and 1. A value of 1 indicates that all connections are receiving the fairest allocation of bandwidth.

Three simulation sets are conducted:

- 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.

Table 1 shows results of the three cases. In Case 1, TCP Reno flows having high-bandwidth access links compete with similar 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 a slight benefit from HSTCP, but the performance of S_1 group is severely damaged and a 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 [27]), 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 group S_2.[2] The TCP SACK mechanism [29], 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. This function is not available to S_1 group, and that group therefore receives less link bandwidth compared to Case 2.

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 group S_2 cannot fully utilize the leftover bandwidth of group S_1. But in Case 2 and Case 3, the larger the propagation, the smaller the throughput that can be achieved by group S_1 due to the use of different algorithms by the two groups.

3.2. gHSTCP Description

The original 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:

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

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

Because the pairs (d_i,t_i) are N independent observations, r can be used to estimate the population correlation ρ. To make inference about ρ using r, usually N is a large number, we require the sampling distribution of r by calculating the statistic Z:

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

If Z is larger than a certain value (3.09 in the following simulation results), there is very strong evidence of statistical significance, i.e. (d_i, t_i) is positive correlation, otherwise it is non-positive correlation.

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

2 In this paper, we don't apply SACK option to group S_1 though it is pointed that half of TCP connections are using TCP/Sack [28]. Group S_1 represents the ordinary users in simulations, they maybe not explicitly enable SACK option on their systems.

Case	S_1 group	S_2 group	Router Mechanism	Delay (ms)	Throughput of S_1 (Mbps)	Throughput of S_2 (Mbps)	Fairness
1	Reno	Reno	TailDrop	25	493.83	1565.58	0.99
				50	493.38	1251.27	0.95
				100	500.00	1315.89	0.95
2	Reno	HSTCP	TailDrop	25	168.22	1630.03	0.85
				50	112.40	1533.62	0.76
				100	129.23	1439.23	0.78
3	Reno	HSTCP with SACK option	TailDrop	25	45.42	2396.71	0.57
				50	42.55	2396.17	0.57
				100	36.95	2353.52	0.55

Table 1. Aggregate Throughput and Fairness Comparison with Reno/HSTCP+TailDrop

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, and adaptively changes mode as needed. The algorithm is summarized as follows.

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

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

where $a(w)$ is given by:

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

Once a retransmission timeout occurs, or duplicated acknowledgments are received, the sender decreases the congestion window in the same way as original HSTCP. 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 in the model of Figure 3. Using TailDrop as the queue management mechanism, the following simulations are performed:

- Case 4: TCP Reno is used for S_1 and gHSTCP is used for S_2.
- Case 5: TCP Reno is used for S_1 and gHSTCP+SACK is used for S_2.

From the results (Table 2), where TCP Reno is used with gHSTCP (Case 4), we can see that throughput is significantly improved for both S_1 and S_2 groups. Fairness also improves compared with the HSTCP case (Case 2). Although total throughput for the high-speed flows of Cases 3 and 5 is the same when the SACK option is used, throughput is greatly improved for the TCP Reno connections in Case 5. Is is because gHSTCP can adaptively change its increase mode according to the network congestion level and avoid the "starving" condition that occurs in traditional TCP, better fairness among the different flow types is obtained.

Table 1 and Table 2 also illustrate degraded fairness among HSTCP/gHSTCP and TCP Reno flows as the bottleneck link delay becomes 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 when the window size is larger and decreases it more slowly. 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. This shows the disadvantage of TCP Reno in a long-delay network as discussed in Section 1. When TCP Reno is used for high-speed flows, fairness is better than that if SACK option is used but too much link bandwidth goes unused.

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 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. The queue length minimum

Case	S_1 group	S_2 group	Router Mechanism	Delay (ms)	Throughput of S_1 (Mbps)	Throughput of S_2 (Mbps)	Fairness
4	Reno	gHSTCP	TailDrop	25	392.79	1810.27	0.99
				50	269.47	1768.84	0.93
				100	147.11	1580.86	0.80
5	Reno	gHSTCP with SACK option	TailDrop	25	266.38	2168.85	0.89
				50	189.09	2241.03	0.80
				100	79.24	2324.06	0.63

Table 2. Aggregate Throughput and Fairness Comparison with gHSTCP+TailDrop

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 6: TCP Reno is used for S_1 and HSTCP is used for S_2 with RED deployed.
- Case 7: TCP Reno is used for S_1 and HSTCP+SACK is used for S_2 with RED deployed.
- Case 8: TCP Reno is used for S_1 and gHSTCP is used for S_2 with RED deployed.
- Case 9: TCP Reno is used for S_1 and gHSTCP+SACK is used for S_2 with RED deployed.

From the results,[3] we find that fairness is improved, but link under-utilization is still present and total throughput is less than that using TailDrop in some cases. In this high-speed environment, every 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. 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, a simultaneous decrease in the congestion windows of multiple flows can still occur, but the congestion windows will not be reset to 1 packet.

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 initially to 0.001:

- Case 10: TCP Reno is used for S_1 and HSTCP is used for S_2 with RED ($max_p = 0.001$).
- Case 11: TCP Reno is used for S_1 and HSTCP+SACK is used for S_2 with RED ($max_p = 0.001$).
- Case 12: TCP Reno is used for S_1 and gHSTCP is used for S_2 with RED ($max_p = 0.001$).
- Case 13: TCP Reno is used for S_1 and gHSTCP+SACK is used for S_2 with RED ($max_p = 0.001$).

3 Due to limited space, the results are not presented here.

The results of these simulations show that the system can achieve both high throughput and better fairness in this situation. 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.

In the next section, an additional mechanism will be introduced to address this problem.

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 deduce 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.

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 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

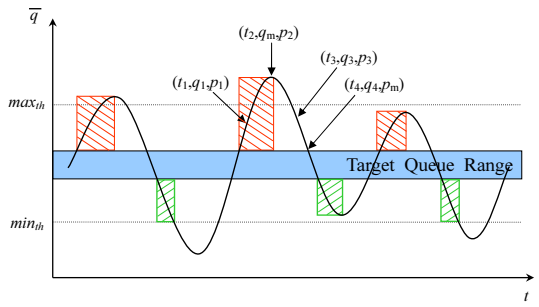

Figure 4. Sketch of Average Queue (gARED)

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. In ARED, it is set 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.

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.

However, based on analysis and simulation, we find some problems with ARED. One is that it does not consider the trend in average queue variation when it changes the upper bound of max_p. Another one is that the lower bound of parameter max_p is determined to some extent by the network manager to ensure ARED performance.

4.2. An Improvement of ARED

To solve these problems inherent to ARED we propose a modified version referred to gARED as shown in Figure 4. 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 the original ARED, if the average queue size is larger than the target queue size while t is in the interval (t_2, t_4), ARED increases max_p but gARED does not. The small max_p gives the network more stability. On the other hand, if the average queue size is less than the target queue, max_p is larger for gARED than one for ARED, So that the average queue can return to the target queue slowly.

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

The algorithm of gARED is given as:

```
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
```

4.3. Evaluation of gHSTCP with gARED

Finally, Table 3 shows throughput and fairness when there are five gHSTCP flows competing with 5 TCP Reno flows, and gARED is used at the routers. Two simulation experiments are conducted:

- Case 14: TCP Reno is used for S_1 and gHSTCP is used for S_2 with gARED deployed.
- Case 15: TCP Reno is used for S_1 and gHSTCP+SACK is used for S_2 with gARED deployed.

We can see that when TCP Reno is used with gHSTCP with a propagation delay of 25 ms or 50 ms, the virtues of gHSTCP and gARED in combination are exhibited in terms of throughput and fairness. Even without the SACK option, gHSTCP achieves the same performance as when the SACK option is used. At a propagation delay of 100 ms, these are acceptable results compared to that when ARED is deployed.

5. 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, which adjusts max_p according to the average queue length and the trend in variation. This technique also avoids the problem of determining an appropriate lower bound for max_p. We showed through simulations

Case	S_1 group	S_2 group	Router Mechanism	Delay (ms)	Throughput of S_1 (Mbps)	Throughput of S_2 (Mbps)	Fairness
14	Reno	gHSTCP	gARED	25	477.55	1921.03	1.00
				50	454.17	1986.32	0.99
				100	148.25	1778.03	0.79
15	Reno	gHSTCP with SACK option	gARED	25	493.17	1943.71	1.00
				50	454.15	1979.05	0.99
				100	188.28	2177.32	0.77

Table 3. Aggregate Throughput Comparison with gHSTCP+gARED

that the 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 the trends of changing RTT.

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," in *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, October 1997, pp. 205–211.

[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," in *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," in *Proceedings of INFOCOM 2004*, March 2004.

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

[11] H. Bullot 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," in *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] ——, "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," in *Proceedings of INFOCOM 1999*, March 1999, pp. 1320–1328.

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

[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," in *Proceedings of SIGCOMM 2000*, September 2000, pp. 151–160.

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

[20] M. Christiansen, K. Jaffay, D. Ott, and F. D. Smith, "Tuning RED for web traffic," in *Proceedings of SIGCOMM 2000*, August 2000, pp. 139–150.

[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] 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.

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

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

[27] 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.

[28] S. Floyd, "Thoughts of the evolution of TCP in the Internet," February 2004, available as: http://dsd.lbl.gov/DIDC/PFLDnet2004/talks/Floyd-slides.pdf.

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