社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

TailDrop/ARED ルータ下での HighSpeed TCP の性能改善手法

張　　宗升† 　長谷川　剛† 　村田　正幸†

† 大阪大学大学院情報科学研究科
E-mail: †{zhang,hasegawa,murata}@ist.osaka-u.ac.jp

あらまし　現在実現しつつある 1〜10Gbps 級のリンクを持つ超高速ネットワークにおいて、TCP を用いてデータ転送を行うと、高速リンク帯域を使い切れないことが指摘されている。この問題に対して、HighSpeed TCP が提案されているが、その詳細な性能評価は行われていない。特に、既存の TCP Reno との公平性に関しては考慮されていない。本稿では、HighSpeed TCP の持つ問題点を指摘し、その不公平性を改善する gHSTCP 方式を提案を行う。また、超高速ネットワークにおける RED の問題点を指摘し、それを解消する方式として、ARED を改善した gARED 方式の提案を行う。その結果、gHSTCP と gARED を組み合わせることによって、リンク帯域を有効に利用し、かつ TCP Reno との公平性を大幅に改善できることをシミュレーション結果を用いて明らかにする。
キーワード　TCP Reno, HighSpeed TCP, 公平性, TailDrop, RED

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

Zongsheng ZHANG†, Go HASEGAWA† and Masayuki MURATA†

† Graduate School of Information Science and Technology, Osaka University
E-mail: †{zhang,hasegawa,murata}@ist.osaka-u.ac.jp

Abstract　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 the HighSpeed TCP protocol 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 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. This new Active Queue Management (AQM) mechanism 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 more 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 2×10^{-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.

HighSpeed TCP (HSTCP) [2] was recently proposed by Sally Floyd 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. 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.

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.

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 underutilization 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. 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 mechanisms to implement the proposal. Finally, Section 5 presents our conclusions for this study.

2 Background

To overcome problems with TCP mentioned in Section 1, HSTCP was proposed [2]. The HSTCP algorithm uses the principle of Additive Increase Multiplicative Decrease (AIMD) as does 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.

Equations (1),(2) show that increasing parameter $a(w)$ becomes larger as congestion widow size increases, while decreasing parameter $b(w)$ becomes smaller. In this manner, HSTCP can sustain a large congestion window fully utilizing high-speed long-delay network.

HSTCP response function[1] (3) illustrates 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.

HSTCP is simple 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 and effective modification, the result of 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 simulation experiments.

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 [22] is used for the simulations. The network topology is shown in Fig 1, 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 or 100 ms. 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 in the results, respectively. TailDrop is used as the queue management mechanism in this section. We use a greedy FTP source for data transmission. The aggregate throughput is used as an evaluating metric. 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.

Fig 2 shows throughput of the three cases. In Case 1, TCP Reno flows having high-bandwidth access links compete with similar flows having lower bandwidth access links. In

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

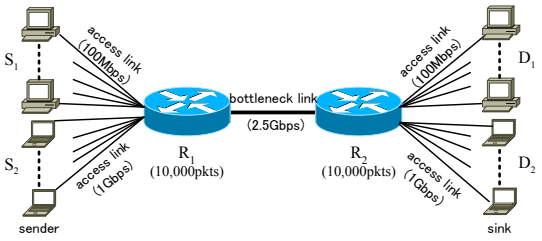

Fig 1: Topology

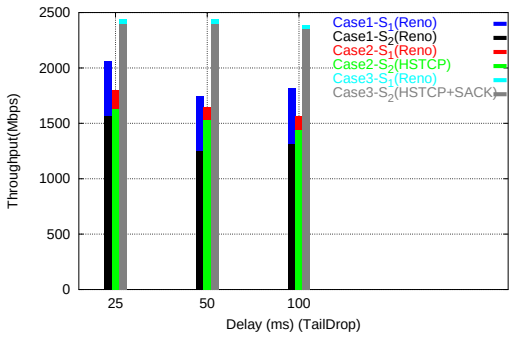

Fig 2: Throughput Comparison (Reno/HSTCP, TailDrop)

the figure, "Case1-S_1 (Reno)" represents aggregate throughput of the low-bandwidth access link flows (S_1 group), and "Case1-S_2 (Reno)" represents the aggregate throughput of the high-bandwidth access link flows (S_2 group), for Case 1. We can see that the S_1 group fully utilizes its access link bandwidth, and the 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 the S_2 group instead of TCP Reno. The S_2 group obtains a slight benefit from HSTCP, but performance of the S_1 group is severely damaged and a degradation in total throughput occurs. This is because the congestion window is inflated in the 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 [23]), 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.

The TCP SACK mechanism [24], combined with a selective retransmission policy, can help overcome limitations in recovering from many packet losses. In Case 3, the TCP SACK option is applied with HSTCP for group S_2. Fig 2 shows that the S_2 group achieves very high throughput while that of TCP Reno is severely degraded. Although there are still multiple packet drops, the S_2 group, using the SACK option, infers the dropped packets and retransmits only the missed ones. This function is not available to the 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 the S_2 group does not affect the 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

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 queueing at the router buffer. In addition, differences in speed gains among the different connection types 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.

Defining the departure time and RTT value of a transmitted packet as d_i and t_i, respectively, then a correlation between d_i and t_i is tested statistically. 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 nonpositive correlation between d_i and t_i, it means the network is in an underutilized 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. Once a retransmission timeout occurs, or if duplicated acknowledgments are received, the sender decreases the congestion window in the same way as original HSTCP. For example, when a timeout occurs, the congestion window size is reset to one packet and the phase is changed to slow-start. If three duplicated acknowledgments are received, the window size is set to $(1 - b(w)) \times w$. 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:

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

3.3 gHSTCP Evaluation with Simulations

In this subsection we compare the performance of HSTCP and gHSTCP based on simulations using *ns-2*. 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.

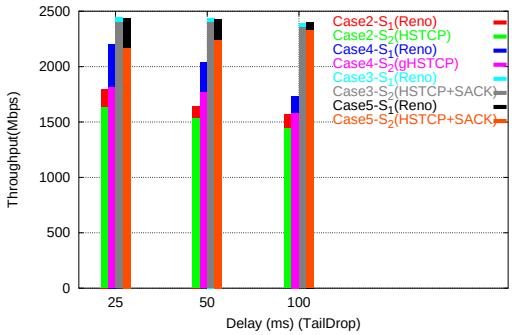

Fig 3: Performance Comparison (TailDrop)

From Fig 3, where TCP Reno is used with gHSTCP (Case 4), we can see that throughput is significantly improved for both TCP Reno and gHSTCP and fairness is also improved compared with the HSTCP case. Fig 3 also shows that, 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. We can also see that fairness is better among the different flow types with the help of gHSTCP. This 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.

4 gARED: Gentle Adaptive RED

The results in Section 3 are primarily the effects of the TailDrop mechanisms at the bottleneck routers. To address the problems with TailDrop as pointed in Section 1, RED is recommended to use. However, control parameter settings in RED have proven highly sensitive to the network scenario, and misconfiguring RED can degrade performance significantly [16–20]. 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 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 the key point 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,

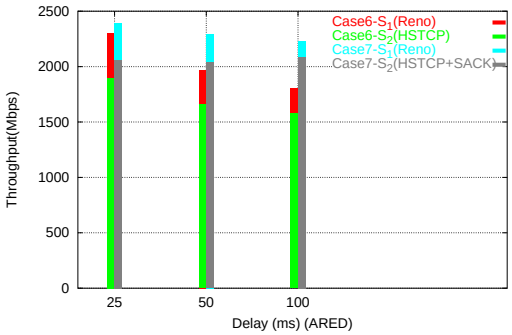

Fig 4: Performance Comparison (ARED)

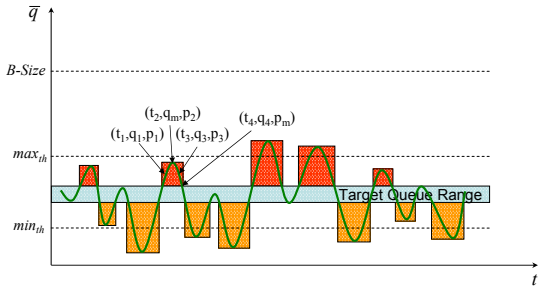

Fig 5: Sketch of Average Queue (ARED)

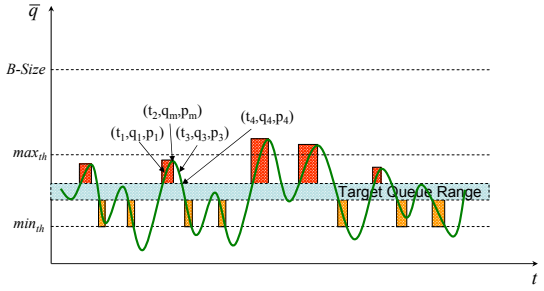

Fig 6: Sketch of Average Queue (gARED)

0.5). The objective of ARED is to achieve an expected target queue $[min_{th}+0.4*(max_{th}-min_{th}), min_{th}+0.6*(max_{th}-min_{th})]$ by adapting max_p.

4.2 Simulation with ARED

To evaluate the effectiveness of ARED in a high-speed long-delay network we perform two simulations, 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 [21], the following simulations are performed:

- Case 6: TCP Reno is used for S_1 and HSTCP is used for S_2 with ARED deployed.
- Case 7: TCP Reno is used for S_1 and HSTCP+SACK is used for S_2 with ARED deployed.

The results are shown in Fig 4. When TCP Reno is used for HSTCP, the system performance is significantly improved in terms of throughput and fairness compared with that of TailDrop (Fig 3). If SACK option is used with HSTCP, the fairness is better than that with TailDrop used. The bottleneck link remains underutilized, however. This is due to an improper setting for the ARED packet drop probability. We will now describe the shortcomings of ARED in detail.

The graph in Fig 5 shows a sketch map of the average queue size as it varies with 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, and the y-axis is the average queue length. When the average queue size increases to greater than the target queue size, as shown in the orange areas, 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, as shown in the yellow areas in the figure, 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 feedback signal, delayed by about one RTT, causes senders to decrease their send rates to less than the ideal rate. Especially, the larger the drop probability, the more the TCP senders rates will be less than ideal. Moreover, as the propagation delay and queue size increase, this phenomenon will become more serious.

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

4.3 An Improvement of ARED

To solve these problems inherent to ARED we propose a modified version we refer to as gARED. The modifications shown in Fig 6. When the average queue becomes larger than the target queue and there is an increasing trend (orange areas), 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 (yellow areas), max_p is decreased. When the average queue size is less than min_{th}, no packets are being dropped, therefore, even for a decreasing trend max_p is not decreased.

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 for ARED.

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}.

4.4 Evaluation of gHSTCP with gARED

Finally, Fig 7 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 8: TCP Reno is used for S_1 and gHSTCP is used for S_2 with gARED deployed.
- Case 9: 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 HSTCP

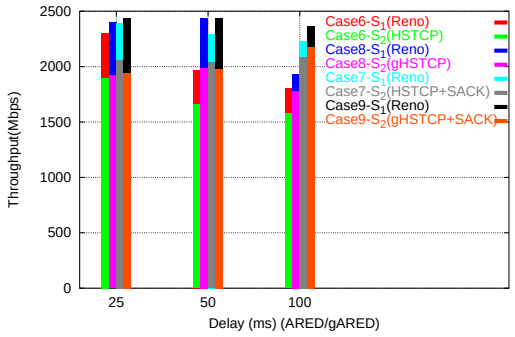

Fig 7: Performance Comparison (HSTCP+ARED/gHSTCP+gARED)

and gHSTCP (Case 6 and 8) 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, gHSTCP throughput and total throughput are greater than for HSTCP, while TCP Reno throughput is sightly less than for HSTCP. These are acceptable results compared to that when ARED is deployed. In this case, however, there is a tradeoff between total throughput and fairness. Fig 7 also shows the case for HSTCP and gHSTCP using the SACK option (Case 7 and 9). Individual group throughputs for connections with gARED deployed outperform those when HSTCP and ARED are deployed. Fairness is also improved between gHSTCP and TCP Reno flows. Even if the propagation delay of the bottleneck link link becomes large, our proposal can minimize the loss in throughput.

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, 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 extensive investigation of gHSTCP, e.g., how to recover effectively from simultaneous packet losses, refinement of the technique for making estimations based on trends and analysis of the impact that gHSTCP flows may have on Web traffic.

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] Y. Z. L. Qiu 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] "TCP stacks testbed." [Online]. Available: 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] 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," 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 and R. Gummadi and S. Shenker, "Adaptive RED: An algorithm for increasing the robustness of RED." [Online]. Available: http://www.icir.org/floyd/papers/

[22] "The network simulator - ns-2." [Online]. Available: http://www.isi.edu/nsnam/ns/

[23] 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.

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