Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

An algorithm for reducing loss rate of high-speed TCP

SU Fan-jun[†1,2], PAN Xue-zeng[1], WANG Jie-bing[1], WAN Zheng[1]

([1]*School of Computer Science, Zhejiang University, Hangzhou 310027, China*)

([2]*College of Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China*)

[†]E-mail: sufanjun@163.com

Received Dec. 21, 2005; revision accepted Feb. 26, 2006

Abstract: Some high-speed protocols such as HSTCP have been proposed to improve the ability of bandwidth utilization in high-speed networks. However, the increased scalability of high-speed TCP leads to many dropped packets in a single loss event in drop tail environment. In addition, there exists burstiness on short time scales that may cause lots of packets loss. In this paper, we analyze the problem of packet loss, and then propose ACWAP (Adaptive Congestion Window Adjustment plus Pacing) algorithm to reduce the loss rate of high-speed TCP. Along with pacing algorithm for avoiding burstiness on short time scales, ACWAP uses delay information to estimate the network state and adaptively changes the increase parameter to 1 before congestion to reduce the number of dropped packets. Many simulation results show our proposed algorithm can reduce the number of dropped packets in a single loss event, alleviate synchronized loss phenomena and improve the RTT unfairness while keeping the advantages of high-speed TCP.

Key words: High-speed, Loss rate, HSTCP, Congestion control
doi:10.1631/jzus.2006.AS0245 **Document code:** A **CLC number:** TP393

INTRODUCTION

Over the past few decades, though the traffic on the Internet has increased by several orders of magnitude, the Internet still works well, which attributes primarily to the congestion control algorithms of TCP (Jacobson, 1988).

However, the same congestion control algorithm performs badly in high-speed networks with bandwidth larger than 1 Gbps, or even 10 Gbps (Floyd, 2003). In order to enhance the performance of TCP in high-speed networks, HSTCP (Floyd, 2003), STCP (Kelly, 2003), BIC (Xu *et al*., 2004), LTCP (Bhandarkar *et al.*, 2004) are proposed. HSTCP and STCP change the congestion window adjustment algorithm by making $a(w)$ and $b(w)$ become the function of congestion window size, where $a(w)$ and $b(w)$ are increase parameter and decrease parameter respectively. BIC and LTCP also increase the scalability of standard TCP. Though the increased scalability of high-speed protocols is the requirement for full bandwidth utilization, some adverse effects arise together.

HSTCP has been adopted by IETF, so many researchers focus on the performance of HSTCP. To improve the convergence performance of HSTCP, a new algorithm (Nabeshima and Yata, 2004) was proposed recently. Pan *et al*.(2006) proposed CW-HSTCP to reduce the RTT unfairness of HSTCP. Barman *et al*.(2004) researched the effect of router buffer size on the performance of HSTCP. Souza and Agarwa (2003) concluded that HSTCP has a better performance with RED (Floyd and Jacobson, 1993) queue management through simulation evaluation.

However, DT (Drop Tail) queue management is widely used due to its simplicity. Floyd (2003) pointed out that when DT queue management is adopted, there are at most $a(w)$ dropped packets per loss event in a DT environment as the scalability of HSTCP, and burstiness on short time scales is severe because of the large congestion window achieved by HSTCP, but he had not deeply analyzed the problem.

Huge packet loss requires retransmission, which increases the burden of the sender and wastes network

resource. Consequently, we present an algorithm to reduce the loss rate of HSTCP. Our experiments showed that the number of dropped packets in one loss event is related to $a(w)$ and that packets are not dropped successively to result in severe synchronized loss. On the basis of a deep analysis, we propose ACWAP (Adaptive Congestion Window Adjustment plus Pacing). The basic idea is that the sender adaptively changes $a(w)$ to 1 before congestion according to the variation of RTT, which is a reflection of network state (Brakmo and Peterson, 1995; Kuzmanovic and Knightly, 2003). To avoid burstiness, we also adopt pacing algorithm. ACWAP-HSTCP is the combination of HSTCP and ACWAP algorithm. We did some simulations using ns2 to evaluate the performance of ACWAP-HSTCP. Simulation results showed that ACWAP can reduce the number of dropped packets, alleviate synchronized loss and RTT unfairness. Particularly, ACWAP-HSTCP can coexist with HSTCP fairly and can be deployed gradually in Internet. Briefly speaking, our algorithm is scalable. In addition to HSTCP, ACWAP algorithm can be combined with some other algorithms, such as STCP, LTCP.

SLIDING WINDOW MECHANISM AND CONGESTION CONTROL MECHANISM OF TCP AND HSTCP

Sliding window mechanism

TCP uses sliding window mechanism and end-to-end acknowledgments to provide reliable data transfer across a network (Stevens, 1994). Fig.1 shows the TCP window management mechanism. The window size (W) is determined as the minimum of receiver's advertised buffer space and the congestion window size of sender. The sender allows up to W outstanding or unacknowledged packets at a time.

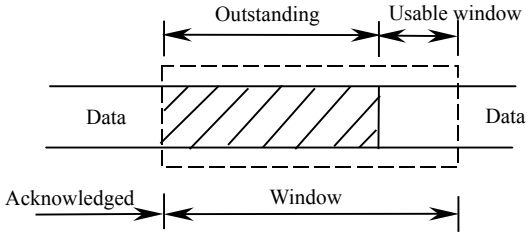

Fig.1 TCP window management

This results in a "usable window" size equal to W minus the number of outstanding packets. After receiving a new ACK packet, the window will slide to right.

Congestion control mechanism

Congestion control mechanism of TCP and HSTCP consists of slow start, congestion avoidance, fast recovery and so on. In congestion avoidance phase, TCP and HSTCP use the following algorithm to adjust congestion window:

$$\text{A new ACK: } w \leftarrow w+a(w)/w, \quad (1)$$

$$\text{Congestion: } w \leftarrow w-b(w)\times w, \quad (2)$$

where w denotes congestion window size (cwnd), $a(w)$ and $b(w)$ are additive increase and multiplicative decrease parameters respectively. For standard TCP, $a(w)=1$, $b(w)=0.5$. For HSTCP, $a(w)$ and $b(w)$ become the function of w. As w increases, $a(w)$ will increases, while $b(w)$ decreases. For example, when $w=118$, $a(w)=2$, and $b(w)=0.44$. When $w=347$, $a(w)=4$, and $b(w)=0.38$. Floyd (2003) gives a detailed introduction of HSTCP.

PACKET LOSS ANALYSIS

In this section, we analyze the packet loss problem of HSTCP with drop tail queue management using simulation. We adopted ns2 simulator (version 2.26). Simulation topology and configuration are the same as those in (Pan *et al*., 2006).

Simulation results

Let the bandwidth of the bottleneck link be 1 Gbps and run three HSTCP flows ($H1$, $H2$, $H3$) with different RTT. Fig.2 shows the congestion window evolution and results of a detailed analysis are listed in Table 1, through which we conclude as follows:

(1) The number of dropped packets has relation to $a(w)$.

(2) There is severe synchronized loss between HSTCP flows, which means that multiple competing flows simultaneously encounter loss events. Xu *et al*.(2004) pointed out that synchronized loss and the character of HSTCP were the causative reasons for the severe RTT unfairness. Compared with HSTCP

Table 1 Simulation results of HSTCP during 50~400 s

Flow	*RTT* (ms)	Bandwidth utilization (%)	Congestion numbers	Dropped packets number in one congestion event	Cwnd size when congestion	$a(w)$
*H*1	60	78.34	83	31.61	6653~7219	23~25
*H*2	120	7.96	62	11.59	1007~1946	8~12
*H*3	180	0.93	56	3.78	119~293	2~4
*R*1	120	1.71	20	2.25	112~443	1

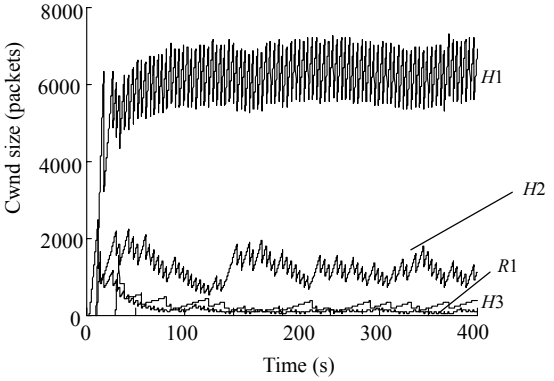

Fig.2 The congestion window evolution of HSTCP under DT router

flows, standard TCP flow (denoted as *R*1) enconters less congestion.

Analysis of packet loss problem

A simple simulation is carried out for deep analysis to gain understanding of packet loss details. One HSTCP connection is established without background traffics, and the bottleneck bandwidth was set to 500 Mbps. Some simulation results are presented in Fig.3, where W_c denotes the congestion window size when congestion occurs. From Fig.3a, we can find that W_c=8400. Our simulation results showed in one congestion event 26 packets equaling the value of $a(W_c)$ dropped. Moreover, we find that packets are not dropped successively with the sequence gap of dropped packets being about 323. For example, given that the sequence number of one dropped packet is *i*, the sequence number of the next dropped packet is about (*i*+323).

If w_i and RTT_i denote the congestion window size and RTT of flow *i* respectively, throughput of flow *i* equals w_i/RTT_i. When throughput reaches 500 Mbps (Fig.3b), the queue length of the router will increase too (Fig.3c), resulting in the increase of RTT as shown in Fig.3d, where narrow solid line denotes instantaneous RTT, and bold solid line denotes smoothed RTT (the saw-toothed shape is caused by coarse timer granularity). When the router queue buffer overflows and if the sender increases its throughput further, the router will drop some packets. Once one packet is dropped, the sender can detects packet loss after one RTT through 3 duplicated ACK packets (Stevens, 1994). In interval of one RTT, the congestion window will increase by $a(w)$, which will all be dropped, so the number of dropped packets in one congestion event is $a(w)$.

Next, we explain why the packets are not dropped successively. The sending process deduced from Fig.3 is shown in Fig.4. We mentioned in Section 2 that HSTCP is based on sliding window mechanism (as shown in Fig.1). If receiver's buffer has no limit, the sender will send all the packets allowed back-to-back, and the usable window will become 0. Once a new ACK packet is received, the sender acknowledges one data packet and updates the congestion window according to Eq.(1), together with the window sliding rightward one packet. The usable window will become 1 again, leading to one packet being sent. Therefore, the interval of two data packets (denoted as *t* in Fig.4) is affected by the bottleneck link bandwidth, which is called ACK-clock mechanism (Jacobson, 1988). As shown in Fig.4, once a new ACK packet is received and the calculated increases of congestion window reach 1, the usable window will become 2 that allow 2 packets such as packet *a* and packet *b* to be sent back-to-back. If the router overflows, packet *b* will be dropped since it is a bursty packet. According to Eq.(1), the congestion window increment is $a(w)/w$ for a new ACK packet, so $w/a(w)$ ACK packets are required to let the increment reach 1 (as shown in Fig.4), which means such burstiness occurs in every $w/a(w)$ data packets. In our simulation, when congestion occurs, W_c=8400, and $a(W_c)$=26, so the interval of dropped packets' sequence number is about 323.

The un-successive packets loss leads to severe

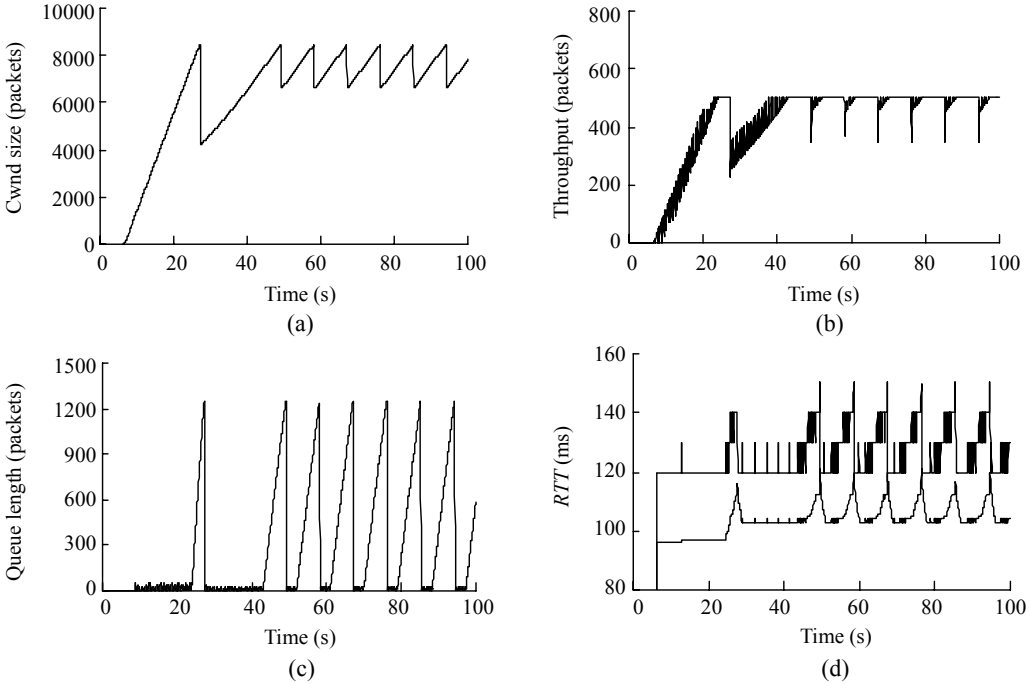

Fig.3 Simulation results of single HSTCP flow. (a) Congestion window evolution versus time; (b) Throughput versus time; (c) Queue length in DT router versus time; (d) *RTT* versus time

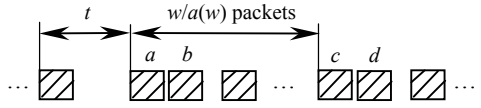

Fig.4 The microcosmic sending process

synchronized loss. An interval of RTT is needed for the sender to detect a congestion event. For standard TCP, after detecting congestion, a short RTT flow decreases its throughput, so it is possible for a long RTT flow to avoid packet loss. For HSTCP flows, there are $a(w)$ [$a(w)\geq 1$] dropped packets per RTT and packets are not dropped successively, which increases the possibility of synchronized packet loss. For example, in Table 1, for flow $H2$, RTT=120 ms. When congestion occurs, there are about 10 dropped packets, which means the interval of two dropped packets is at most 12 ms that is smaller than the RTT of flow $H1$, so severe synchronized loss occurs.

In an actual network, the situation is more complex. Congestion, reordering on the reverse path, or idling of the connection are common occurrences. On the other hand, in high-speed networks, a very large congestion window, such as a congestion window of 83000, can be achieved by HSTCP flows. Therefore, an ACK packet may acknowledge hundreds or thousands of packets, which leads to a large burst on short time scales. This is the reason why the number of loss packets of $H1$ in Fig.2 is larger than $a(w)$.

In short, the increased scalability of high-speed TCP and burstiness on short time scales are the causative reason of the large number of dropped packets.

ACWAP ALGORITHM AND ACWAP-HSTCP

In this section, based on the above analysis, we propose ACWAP algorithm and ACWAP-HSTCP which is the application of ACWAP to HSTCP.

The main idea of ACWAP is that the sender adaptively changes the value of $a(w)$ to 1 before congestion. The variation of RTT reflects the network state (Brakmo and Peterson, 1995; Kuzmanovic and Knightly, 2003), which can be used to predict congestion. In our algorithm, we use an exponentially smoothed high accuracy RTT estimate that is supported by standard TCP (Stevens, 1994).

Let R_C denote current RTT value, RTT_{min} denote the minimum of RTT and RTT_{max} denote the maximum of RTT, then RTT_{min} is determined by propagation delay, so (R_C−RTT_{min}) is caused by router queue

increase and ($RTT_{max}-RTT_{min}$) is the biggest increased delay. We use the following equation to predict congestion:

$$R_C \geq RTT_{min}+\beta\times(RTT_{max}-RTT_{min}), \quad (3)$$

where β is congestion factor. If Eq.(3) is satisfied, $a(w)$ is set to 1.

To avoid large burstiness on short time scales, we also adopt pacing algorithm as a supplement. Let *Interval* denote the sending interval of packets, then $Interval=R_C/w$.

A challenge in implementation is the need for a fine-grained timer. This problem was studied by Aron and Druschel (2000), whose work results showed that the soft timer technique allows the system timer to achieve 10 μs granularity without significant system overhead.

When a connection is established, RTT_{min} and RTT_{max} are unknown to the sender, so a detection phase is needed. After being established, a connection will experience slow start phase and congestion avoidance phase (Stevens, 1994). Through slow start phase, a connection can get RTT_{min}, but RTT_{max} may not be correct, because during slow start phase, the congestion window increases exponentially. Therefore, a congestion epoch in congestion avoidance phase is needed. We use a variable, *numofcon* (as shown in Fig.5) to let the change of $a(w)$ be carried out after two congestion events have occurred since a connection is established.

ACWAP-HSTCP is the combination of HSTCP and ACWAP. Namely, ACWAP-HSTCP serves as HSTCP at first, but once Eq.(3) is satisfied, ACWAP algorithm will come into effect by changing $a(w)$ to 1. If β=1, ACWAP-HSTCP will serve as HSTCP completely. Fig.6 shows an example of the congestion window evolution of ACWAP-HSTCP. The details of ACWAP-HSTCP are shown in Fig.5.

SIMULATION EVALUATION

In this section, we evaluate ACWAP algorithm through ACWAP-HSTCP using simulation. Simulation topology and configuration are the same as those in (Pan *et al.*, 2006).

To evaluate the fairness between high-speed TCP flows, fair index (Chiu and Jain, 1989) is used as follows:

$$FI(x)=\left(\sum_{i=1}^{n}x_i\right)^2 \Big/ \left(n\sum_{i=1}^{n}x_i^2\right),$$

where x_i ($x_i \geq 0$) is the link utilization of flow i.

Case of single high-speed flow

We set β=0.8, and run a single ACWAP-HSTCP flow with 500 Mbps bottleneck bandwidth. The congestion window evolution of our algorithm is shown in Fig.6 which can be compared with Fig.3a. It is easy to find that during 40~100 s ACWAP-HSTCP can reduce the number of congestion events from 6 to

```
Initial values:
   RTT_min=10000;  // the minimum of RTT
   RTT_max=0;  // the maximum of RTT
   numofcon=0; // the number of congestion
   β=0.8;
On receiving a new ACK in congestion avoidance phase:
   interval=RTT/cwnd;
   // cwnd is the congestion window size
   if(RTT<RTT_min)
         RTT_min=RTT;
   if(RTT>RTT_max)
         RTT_max=RTT;
   diff=RTT_max–RTT_min;
   if((RTT>=RTT_min+β*diff)&&numofcon>=2)
         a(w)=1;
   else
         a(w)=increment( );
      //increment( ) is used to calculate a(w)
On congestion occurring:
   cwnd=cwnd*(1–b(w));
   numofcon+=1;
   RTT=RTT_min;
On sending data:
   output(interval);
   // send data after interval
```

Fig.5 Pseudo-codes of ACWAP-HSTCP

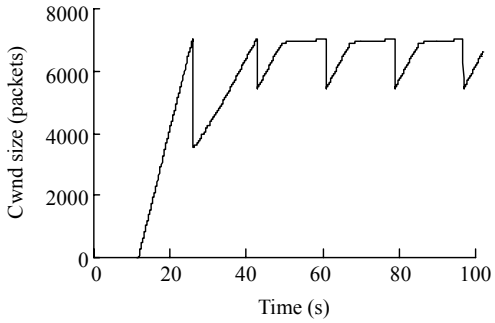

Fig.6 The congestion window evolution of ACWAP-HSTCP

4. Our simulation results also show that only one packet is dropped in one congestion event during congestion avoidance phase.

The case of several high-speed flows with different RTT

We run three ACWAP-HSTCP flows with different RTT for 400 s with β being 0.8. The bottleneck bandwidth is set to be 1 Gbps. We list the simulation results in Table 2 for comparison with those in Table 1.

Table 2 Simulation results of ACWAP-HSTCP during 50~400 s

Flow	*RTT* (ms)	Bandwidth utilization (%)	Congestion event number	Dropped packets number in one congestion
f_1	60	56.32	35	2.56
f_2	120	21.23	22	1.60
f_3	180	12.21	13	1.20

We can find that the number of dropped packets and congestion events both decrease greatly. Moreover, our algorithm can alleviate RTT unfairness, because ACWAP can avoid severe synchronized packet loss. Compared with the case of single high-speed flow, the average number of dropped packets in one congestion event is more than 1, because of packets overlap of different flows. However, the number of dropped packets is still much less than that of HSTCP, which is easily found by comparing of Table 1 and Table 2.

Compatibility and the choice of β

Up to now, there are several proposed high-speed TCP protocols, so compatibility of a new protocol must be considered. Here we study how ACWAP-HSTCP can coexist with HSTCP.

We run a HSTCP flow and an ACWAP-HSTCP flow for 400 s. The simulation results are shown in Figs.7, 8 and 9.

The compatibility of ACWAP-HSTCP depends on the choice of β. We can find that ACWAP-HSTCP flow with a small β such as 0.2 has low loss rate (as shown in Fig.7) but poor fairness (as shown in Fig.8 and Fig.9). On the other hand, for a too large β, such as 0.9, though good fairness is achieved (Fig.9), the loss rate is high (Fig.7).

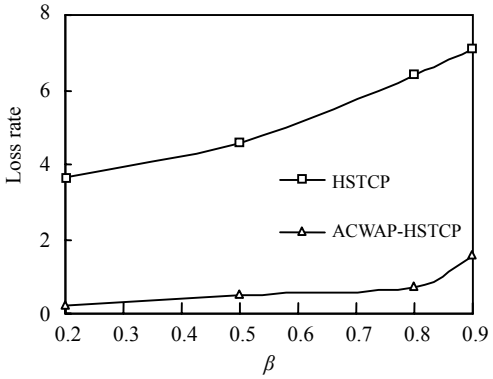

Fig.7 Loss rate of two flows versus β

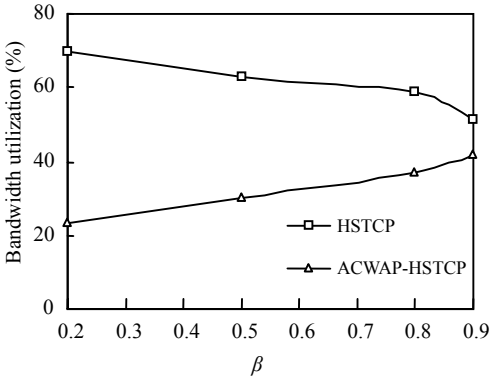

Fig.8 Bandwidth utilization of two flows

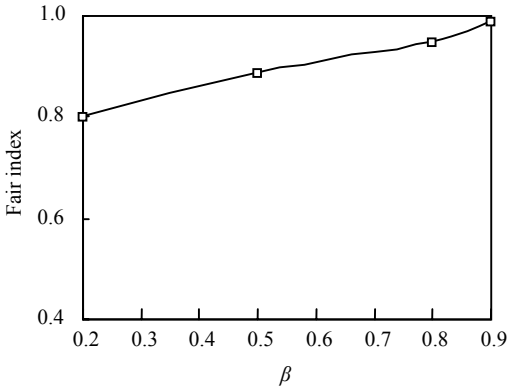

Fig.9 Fairness index versus β

It is easy to understand that the smaller β is, the earlier ACWAP algorithm works. That is to say, with a small value of β, ACWAP-HSTCP can change $a(w)$ to 1 early that will reduce the number of congestion events and avoid synchronized loss. However, HSTCP flow increases the congestion window faster than ACWAP-HSTCP flow. Therefore, with a small β, ACWAP-HSTCP has lower less rate but poor fairness than HSTCP. Compared with a big β, good fairness is

achieved but the loss rate is high. Especially when β=0.9, it is possible for the sender not to change $a(w)$ to 1 before congestion as the effect of ACWAP depends on the RTT estimate accuracy, which may be affected by some factors such as congestion on the reverse path.

Therefore, considering fairness and loss rate, β=0.8 is optimal.

CONCLUSION

In this paper, we propose ACWAP to reduce the loss rate of high-speed protocols. ACWAP adaptively changes $a(w)$ to 1 according to the network state. What is more, pacing is used to avoid burstiness. ACWAP-HSTCP, a combination of HSTCP and ACWAP, is an application of ACWAP algorithm. Simulation results showed ACWAP can reduce the congestion events number, cut down loss rate, and alleviate RTT unfairness of HSTCP. Moreover, ACWAP-HSTCP has good compatibility with HSTCP.

References

Aron, M., Druschel, P., 2000. Soft timers: efficient microsecond software timer support for network processing. *ACM Trans. on Computer Systems*, **18**(3):197-228. [doi:10.1145/354871.354872]

Barman, D., Smaragdakis, G., Matta, I., 2004. The Effect of Router Buffer Size on HighSpeed TCP Performance. Proceedings of IEEE Globecom, Dallas, p.1617-1621.

Bhandarkar, S., Jain, S., Reddy, A.N., 2004. LTCP: A Layering Technique for Improving the Performance of TCP in HighSpeed Networks. INTERNET DRAFT: draft-bhandarkar-ltcp-01.txt.

Brakmo, L., Peterson, L., 1995. TCP Vegas: End to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications*, **13**(8):1465-1480. [doi:10.1109/49.464716]

Chiu, D., Jain, R., 1989. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, **17**(1):1-14. [doi:10.1016/0169-7552(89)90019-6]

Floyd, S., 2003. HighSpeed TCP for Large Congestion Windows. RFC 3649.

Floyd, S., Jacobson, V., 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, **1**(4):397-413. [doi:10.1109/90.251892]

Jacobson, V., 1988. Congestion avoidance and control. *ACM SIGCOMM Computer Communication Review*, **18**(4): 314-329. [doi:10.1145/52325.52356]

Kelly, T., 2003. Scalable TCP: Improving performance in high-speed wide area networks. *ACM SIGCOMM Computer Communication Review*, **33**(2):83-91. [doi:10.1145/956981.956989]

Kuzmanovic, A., Knightly, E., 2003. TCP-LP: A Distributed Algorithm for Low priority Data Transfer. Proceedings of IEEE INFOCOM, San Francisco, p.1691-1701.

Nabeshima, M., Yata, K., 2004. Improving the Convergence Time of HighSpeed TCP. IEEE International Conference on Networks, p.19-23.

Souza, E., Agarwa, D., 2003. A HighSpeed TCP Study: Characteristics and Deployment Issues. LBNL Technical Report LBNL-53215. Http://www.icir.org/floyd/hstcp.html.

Stevens, W.R., 1994. TCP/IP Illustrated, Volume 1: the Protocols. Addison-Wesley.

Pan, X.Z., Su, F.J., Lü, Y., Ping, L.D., 2006. CW-HSTCP: Fair TCP in high-speed networks. *Journal of Zhejiang University SCIENCE A*, **7**(2):172-178. [doi:10.1631/jzus.2006.A0172]

Xu, L., Harfoush, K., Rhee, I., 2004. Binary Increase Congestion Control (BIC) for Fast Long-distance Networks. Proceedings of INFOCOM, Hong Kong, p.2514-2524.