Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT

Computer Communications 26 (2003) 1614–1621

computer communications

www.elsevier.com/locate/comcom

TCP with sender-based delay control

H.T. Kung[a], Koan-Sin Tan[b,*], Pai-Hsiang Hsiao[a]

[a]*Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 01238, USA*
[b]*Institute of Information Management, National Chiao-Tung University, 1001 Ta Hsueh Road, Hsinchu 30050, Taiwan, ROC*

Received 2 May 2002; revised 12 December 2002; accepted 20 January 2003

Abstract

This paper describes a congestion control method for TCP that adjusts the transmission rate of a TCP connection by changing not only the congestion window size as in normal TCP, but also by delaying the transmission of packets at the sender. We refer to this method as TCP with sender-based delay control, or simply SDC. SDC can keep the window size of a TCP connection above a certain threshold even when its fair share of bandwidth is arbitrarily small. Since TCP fast retransmit and recovery is likely to work when the window size of the connection is sufficiently large, the new method can result in reduced frequency of TCP timeouts for the connection. In particular, SDC allows many TCP flows to share a link without experiencing many timeouts. In addition, SDC can reduce a well-known TCP bias against connections with large RTTs. This paper presents the principle behind SDC, and simulation results demonstrating its properties and advantages.

Keywords: TCP; Congestion control; Retransmission timeout; Explicit congestion notification

1. Introduction

TCP is a widely used protocol in Internet applications. There has been a great deal of research on TCP in the literature. However, a problem related to the many-flow case [1] remains. That is, when the number of TCP connections sharing the link is sufficiently large, some of these connections will become 'fragile' in the sense that they will be subject to frequent TCP timeouts [2]. For applications which require long-lived TCP connections, delay introduced by these timeouts may significantly degrade the network performance as perceived by end users. These include audio streaming applications, such as RealAudio and Shoutcast, and interactive applications that use single persistent TCP connections to transfer many files, such as certain on-line games [3].

We elaborate on the objective and challenge of devising solutions for this many-flow TCP problem. When n TCP flows compete on the same bottleneck link, we want each of them to get $1/n$ of the link bandwidth over small averaging intervals such as a few seconds. The small averaging intervals are important because interactive and real-time applications typically demand low-latency performance of the network. This means that these TCP flows should not experience TCP timeouts, as timeouts typically last seconds or longer [4]. Ideally, after having passed the TCP slow start phase, these flows must be kept in the TCP congestion avoidance phase, without experiencing timed out, until the end of the connection.

We note that, during the congestion avoidance phase, the rate of a TCP flow is determined by W/RTT, where W is the congestion window size and RTT is the round-trip time. Thus, when the number n of competing TCP flows increases, each flow must either decrease its W or increase its RTT.

Recall, however, that W cannot be smaller than one packet. In fact, to avoid TCP timeouts, W needs to be larger than four or five packets to allow TCP fast retransmit and fast recovery to work [4]. To be resilient to TCP timeouts, W actually needs to be at least a few packets larger than four packets [5,6].

Since it is undesirable to reduce W below a certain limit as stated above, increasing RTT becomes necessary when the number n of competing flows is sufficiently large. Usually, when n increases, so does buffer occupancy in routers due to congestion. This means increased buffer size

* Corresponding author.
E-mail addresses: freedom@acm.org (K.-S. Tan), kung@harvard.edu (H.T. Kung), shawn@eecs.harvard.edu (P.-H. Hsiao).

0140-3664/03/$ - see front matter © 2003 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0140-3664(03)00110-5

to accommodate increased queueing delay and thus increased RTT. However, introducing any significant queueing delay in a network is generally regarded as undesirable, because such delay would slow down every flow sharing the network. It is therefore a common practice to keep the buffer occupancy low [7]. Furthermore, this approach is not scalable because the size of the buffer needs to be proportional to the number of flows traveling through the router, which can be very large.

In this paper we describe a congestion control method, called sender-based delay control (SDC), that can increase RTT by delaying packet transmission at the sender. By delaying packets and thus extending the RTT, this method can keep the window size of a TCP connection above a certain threshold even when its fair share of bandwidth is arbitrarily small. Since TCP fast retransmit and recovery is likely to work when the window size of the connection is sufficiently large, our method can reduce the frequency of TCP timeouts for the connection.

We demonstrate by simulations that SDC allows many TCP flows to share a link without experiencing many timeouts (see e.g. Figs. 4 and 12). In addition, SDC reduces a well-known TCP bias against connections with relatively large RTTs (see e.g. Figs. 9 and 10.)

It is natural to compare SDC with a version of TCP that is extended with explicit congestion notification (ECN) [8]. Under both ECN and SDC, the TCP sender uses the same congestion notification messages, that is, ACKs with the Congestion Experienced (CE) bits on or off. Because ECN and SDC assume the same network infrastructure, namely ECN-capable routers that can set the CE bits for packets, and because ECN is generally believed to perform better over other versions of TCP, we compare the performance of SDC with that of ECN in this paper. When we refer to traditional TCP in performance comparison, we mean TCP with ECN.

The rest of the paper has the following organization. We present the SDC mechanism in Section 2. In Section 3, we evaluate the performance of SDC with simulation. Related work is reviewed in Section 4. Finally, we conclude in Section 5.

2. Sender-based delay control

The objective of SDC is to keep the window size of a TCP connection above a certain threshold even when the bandwidth fair share of the connection is very small. To achieve this, we extend the RTT by delaying the transmission of packets when necessary.

Under traditional TCP, when network congestion is detected the sender of a connection slows down its transmission rate by reducing its window size so that $W_{\text{reduced}}/\text{RTT}$ is sufficiently small. In contrast, under SDC the sender of a connection may slow down its transmission rate by *adding* delay to packets, thereby extending RTT to $\text{RTT}_{\text{increased}}$, and thus achieving sufficiently small $W/\text{RTT}_{\text{increased}}$. If $W/\text{RTT}_{\text{increased}}$ is equal to $W_{\text{reduced}}/\text{RTT}$, then SDC can slow down a connection to the same transmission rate as that of traditional TCP, without having to reduce W. Therefore, by adding delay to packets when necessary, SDC can slow down a TCP connection while keeping its W above a threshold, say eight packets, so that fast retransmit and recovery can work.

SDC retains the same additive-increase and multiplicative-decrease (AIMD) [9,10] behavior of traditional TCP. The AIMD behavior is important as it assures that TCP connections can reach equilibrium when they are in the congestion avoidance phase [9,11]. Under SDC, AIMD is achieved by making sure that appropriate amounts of delays are added to packets. Consider first the additive-increase case, where traditional TCP increases its window size W by one packet per RTT. That is, when a new ACK is received, the new transmission rate is increased to $(W + 1/W)/\text{RTT}$ from W/RTT. SDC approximates this behavior by properly decreasing the delay. Specifically, RTT is reduced to RTT' so that

$$W/\text{RTT}' = \left(W + \frac{1}{W}\right)/\text{RTT} \quad \text{or } \text{RTT}' = W^2/(W^2+1) \times \text{RTT} \tag{1}$$

Eq. (1) above is used by equation in the SDC algorithm described below in Fig. 2.

On the other hand, for multiplicative-decrease case the delay is doubled to reduce the transmission rate of the TCP connection by half.

2.1. Two-phase control for SDC

SDC uses a two-phase control on the delay of packet transmission at the TCP sender. TCP connection may be in one of the two phases depending on whether or not its window size has reached a predetermined threshold. In this paper, we set the threshold to be eight packets. This particular threshold value reflects the fact that TCP connections with window size greater than eight packets are resilient to timeouts [5]. Specifically, these two phases are as follows.

Small window phase. A TCP connection is in this phase when its window has not reached the window-size threshold of eight packets. Upon receiving the congestion notification, i.e. an ACK with the CE bit on, the sender increases the current delay to be added to packets, rather than decreasing the window size. Upon receiving an ACK with the CE bit off, the sender increases the window size as in traditional TCP. Thus during this small window phase, the window size never decreases, unless there is a timeout. The algorithm enters the large window phase when the window size reaches the threshold of eight packets.

Large window phase. A TCP connection is in this phase when its window size has reached the window-size

threshold of eight packets. Suppose that at the time when the window reaches eight or more packets, the delay to be added to RTT is positive. Then the sender will grow the delay when receiving an ACK with the CE bit on, and shrink the delay when receiving an ACK with the CE bit off, until the delay becomes zero. When growing the delay, the objective is to double the smoothed RTT, i.e. SRTT, and thus reduce the transmission rate by half. When shrinking the delay, the objective is to reduce the delay so that the transmission rate will increase from W/RTT to $(W + 1/W)$/RTT. When the delay to be added to RTT reaches zero, the sender will grow and shrink the window size as in traditional TCP, without adding any delay to packets, until the connection terminates or the window size falls below eight packets. In the later case, the algorithm enters the small window phase.

Fig. 1 shows the two-phase control of SDC. The small window phase consists of two time intervals (a, f) and (i, l). The window size grows during sub-intervals (a, b), (c, d), (e, f), (i, j) and (k, l). During these sub-intervals, delay added to packet is either zero or decreasing. On the other hand, the delay added to packets increases during sub-intervals (b, c), (d, e) and (j, k) when the window size is kept constant. The large window phase consists of two intervals (f, i) and (l, ∞). During these intervals, the window grows or shrinks according to the congestion control algorithms of traditional TCP.

2.2. SDC algorithm

Fig. 2 presents the detailed SDC algorithm that governs the sender behavior after receiving an ACK.

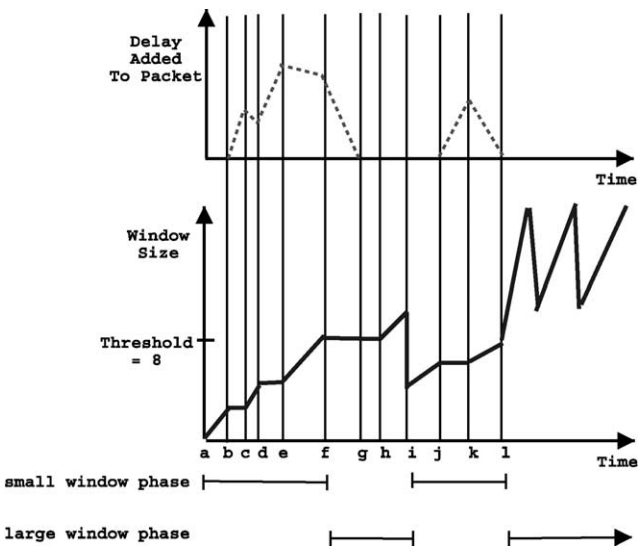

Fig. 1. Two-phase control for TCP with SDC.

Notation :

- $W \leftarrow \min(cwnd, awnd)$, where $cwnd$ and $awnd$ are the size of congestion window and that of advertised windows, respectively.
- D: Amount of delay before transmitting the packet
- $SRTT$: Smoothed RTT, which is updated by

$$SRTT_{new} \leftarrow (1/8) * RTT_{new} + (7/8) * SRTT_{old}$$

- Congestion Experienced: Either the sender receives an ACK packet with ECN CE-bit or three duplicate ACKs.

SDC Algorithm :

```
if W < 8 then
  {Small Window Phase}
  if Congestion Experienced then
    D ← max(2 * SRTT − RTT, 0)
  else
    cwnd ← cwnd + 1/cwnd
    D_new ← 0.9 * (RTT_new + D_old) − RTT_old
  end if
else
  {Large Window Phase}
  if Congestion Experienced then
    if D > 0 then
      D ← max(2 * SRTT − RTT, 0)
    else
      cwnd ← cwnd/2 {also adjust W so that W = min(cwnd, awnd)}
    end if
  else
    if D > 0 then
        D_new ← (W^2/(W^2 + 1)) × (RTT_new + D_old) − RTT_new      (2)
    else
      grow W as in traditional TCP
    end if
  end if
end if
```

Fig. 2. The SDC algorithm.

3. Simulation results

In this section, we compare the performance of SDC to TCP with ECN [8,12] using simulations. We compare SDC to ECN for three reasons. First, both SDC and ECN make use of the CE bit in the packet. Second, they require the same support from routers that can set the CE bit of packets to signal congestion. Third, ECN is generally regarded to have superior performance over other versions of traditional TCP [13]. In particular, under ECN, the TCP sender will reduce its window size upon receiving an ACK packet with the CE bit on. This means that ECN can adapt to congestion quickly before any packet loss. For the rest of this paper, by traditional TCP we mean TCP with ECN, and in all diagrams TCP means TCP with ECN.

We use simulations in ns-2 [14] to conduct the performance comparison. We use three metrics: the number of timeouts, the number of packet drops and packet delivery latency. These metrics are important for several reasons. First, when there are retransmission timeouts, the instantaneous transmission rate of a connection can drop a lot. Reducing the number of timeouts helps the connection to sustain a stable throughput. Second, the throughput of a TCP connection in the congestion avoidance phase is dominated by the packet loss rate. The fewer packets lost the higher throughput the connection can achieve. Finally, packet

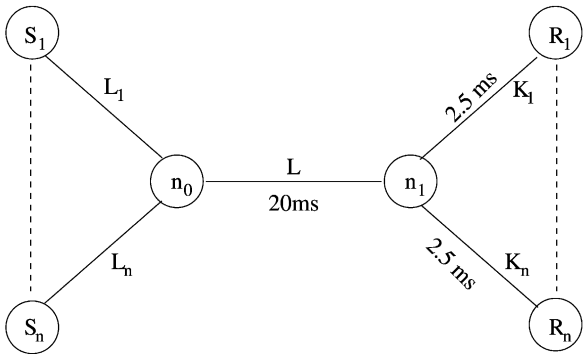

Fig. 3. Simulation configuration. There are n flows, each being a flow from source S_i to destination R_i over link L_i, node n_0, link L, node n_1, and link K_i, for some i in $\{1, 2, \ldots n\}$. The link L is the bottleneck link. Depending on experiments, these n flows can be all SDC flows, all traditional TCP flows, or a mixture of the two types of flows.

delivery latency is related to delay and delay jitter for the particular connection, and is especially important for interactive and real-time applications.

The simulation topology is depicted in Fig. 3, with various parameters related to the simulation summarized in Table 1.

The simulations conducted here intend to study the performance of both methods under the situation where the network on average can only hold a few packets for each flow. Note that in this configuration, the round-trip time of a flow can be as large as 50 ms $((2.5 + 20 + 2.5) \times 2)$. The network path in this case can hold about 108 packets $((10 \times 10^6) \times (50 \times 10^{-3})/(576 \times 8))$, excluding the buffer at the bottleneck router. Thus the maximum number of packets that the network can hold, including the 100 packets at the router, is about 208 packets. In addition, the senders and routers are configured as follows:

- The senders run TCP NewReno [15] to avoid multiple fast retransmits during a single window of data.

Table 1
Simulation parameters

Parameter	Value
Packet size (bytes)	576
Bandwidth of bottleneck link L (Mbps)	10
Propagation delay between n_0 and n_1 (ms)	20
Bandwidth of link L_i (Mbps)	$10 \times (10/n)$
Propagation delay from S_i to n_0	Random value from 1 to 2.5 ms
Propagation delay from n_1 to R_n	Random value from 1 to 2.5 ms
Router buffer	100 packets
RED thresh	Five packets
RED maxthresh	50 packets
RED w_q	0.002
RED $\max_p$	0.1
RED gentle_	True
Simulated length (s)	100

- The senders enable ECN and Limited Transmit [16].
- Routers enable ECN and use RED [17] in ECN marking. Later we present further performance improvement of SDC by using other Active Queue Management schemes (AQM) [7].

3.1. Number of timeouts

Under ECN, timeouts occur under the following two scenarios:

- Received an ACK with the CE bit on when the sender's window size equals one packet.
- In the event of a packet loss, there are not enough unacknowledged packets for the sender to receive three duplicate ACKs.

We note that TCP flows with small windows are more likely to encounter these two scenarios.

As depicted in Fig. 4, when the number of flows increases, the average number of timeouts for a connection can be kept small under SDC. This is because under SDC the window size can be kept above the eight-packet threshold. In the figure, when there are less than 100 flows, one measurement is plotted for each increment of 10 flows; otherwise, one measurement is plotted for each increment of 100 flows. (The same holds for Figs. 6, 9, 10 and 12..)

In contrast, under ECN the average number of timeouts increases rapidly when the number of flows is larger than 30. Note that in these cases, the window size for each connection can hardly exceed four packets and as a result fast retransmit and recovery cannot work. This results in frequent timeouts. To make the situation worse, due to its exponential ramp-up of transmission rate, the slow start phase following each timeout often causes additional timeouts. Moreover, as the number of connections increases beyond a certain value (around 100 in this case), more and more connections enter the exponential backoff phase due to packet loss and timeout. This explains why timeouts are

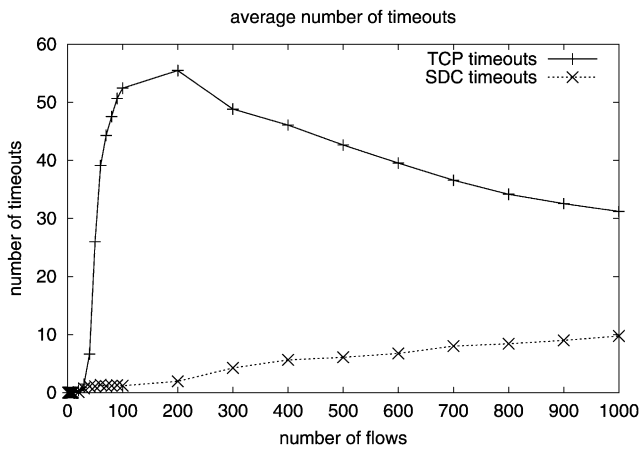

Fig. 4. Average number of timeouts as a function of number of flows.

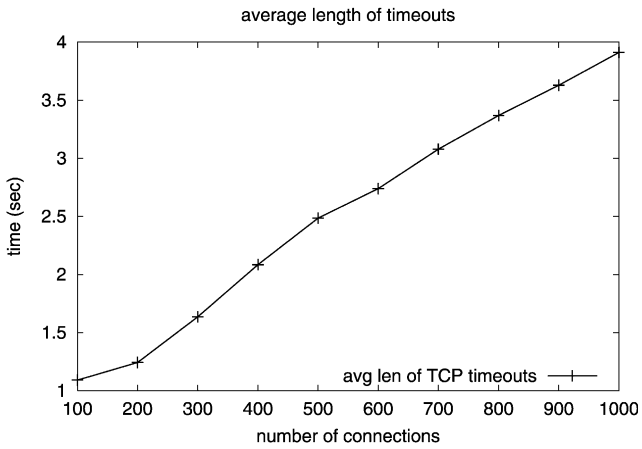

Fig. 5. Average length of timeouts as a function of number of flows for traditional TCP.

reduced when the number of connections is larger than 200 in Fig. 4. In this case, a connection spends a large amount of time in the backoff state and not sending any packet, thus the total number of timeouts decreases. Fig. 5 shows the average time a TCP sender remains in the backoff phase increases as the number of flows increases.

3.2. Number of packet drops

The average number of packet drops for TCP and SDC is presented in Fig. 6. SDC has significantly fewer packet losses than traditional TCP. There are two reasons. First when connections experience fewer timeouts, there are also fewer slow-start bursts, which are major sources of packet drops. Second, by adding delay to packets, SDC allows flows to slow down gracefully to avoid packet losses.

3.3. Packet delivery latency

SDC helps reduce packet delivery latency and its variation. Packet delivery latency is the time between

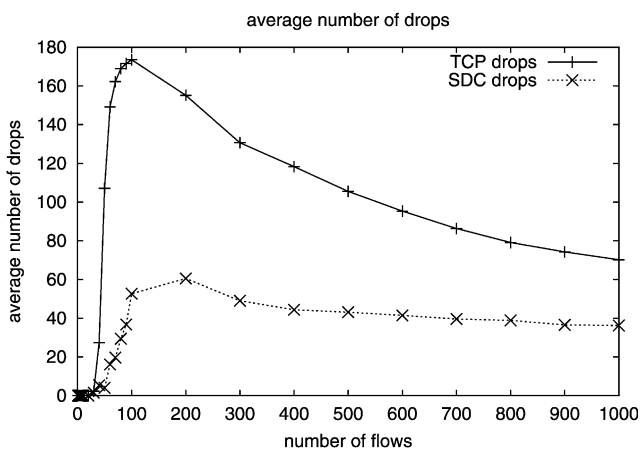

Fig. 6. Average number of packet drops as a function of number of flows.

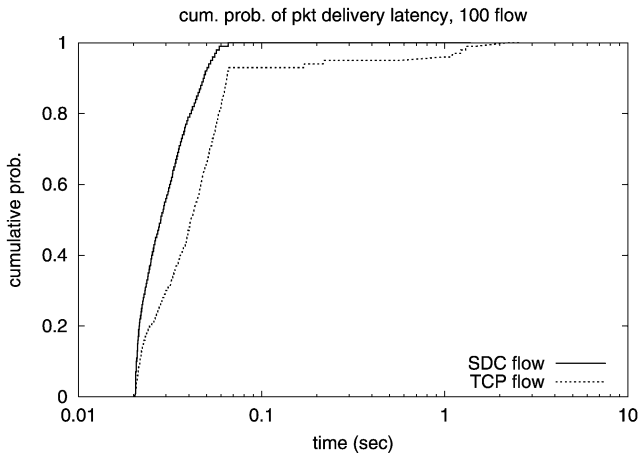

Fig. 7. Cumulative probability distribution of packet delivery within various latency budgets (100 competing flows).

when a packet is first sent and when the packet arrives at the receiver, regardless of how many times the packet is retransmitted. We record all the delivery latencies of a single flow, and report the cumulative probability of packet delivered within the budget. Figs. 7 and 8 report the cumulative probability distribution for 100 and 500 flows, respectively. These figures demonstrate that SDC outperforms TCP in the worst-case and average packet delivery latency. This is mainly due to reduced timeouts in SDC.

As shown in both figures, for SDC flows the cumulative probability of packet delivered within a latency budget increases rapidly to one as the latency budget increases, while that for TCP flows exhibits slower increase and wider variations. The average latencies for SDC and TCP are 0.03 and 0.10 s for the 100 flows simulation; and 0.09 and 0.30 s for the 500 flows simulation.

3.4. Reducing bias against long RTT

SDC reduce a well-known bias of TCP against connections with long RTTs [18,19]. This is because SDC can

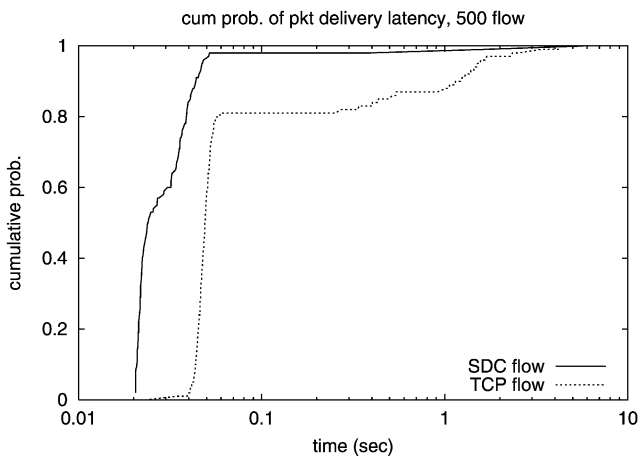

Fig. 8. Cumulative probability distribution of packet delivery within various latency budgets (500 competing flows).

Table 2
Configuration for RTT bias simulations

Parameters	Value
Prop. delay between n_0 and n_1 (ms)	1
Prop. delay between S_i, n_0 for group 1 (ms)	1
Prop. delay between S_i, n_0 for group 2	$(3 \times x - 2)$ ms, with x from 1 to 100
Prop. delay between n_1, R_i (ms)	1

extend RTTs, and thereby reduce the relative differences among their original RTTs.

To demonstrate this, we modify the configuration of Fig. 3 with modified parameters summarized in Table 2. The competing flows belong to two groups: group 1 with propagation delay equal to 1 ms, and group 2 with propagation delay equal to $(3 \times x - 2)$ ms, for x larger than one. That is, RTT1 for group 1 is 6 ms $((1 + 1 + 1) \times 2)$, and RTT2 for group 2 is $6x$ ms $((1 + (3 \times x - 2) + 1) \times 2)$. Thus, RTT2/RTT1 $= x$.

We consider two scenarios: (1) there are a total of 10 flows with five in each group; and (2) there are a total of 100 flows with 50 in each group. For the 10-flow configuration the buffer at the router is 10 packets, with RED's minimum and maximum thresholds set to 2 and 5 packets, respectively. For the 100-flow configuration, the buffer size is 100 packets, with RED's minimum and maximum thresholds set to 5 and 50 packets, respectively.

Figs. 9 and 10 show the ratio of total bandwidth achieved by group 1 to group 2, for the 10- and 100-flow configurations, as a function of the ratio x. We notice that for both configurations, the performance disparity is much smaller when SDC is used.

3.5. Bandwidth competition with traditional TCP

Suppose both SDC and TCP connections compete in the same network. SDC connections in general are less aggressive in bandwidth usage than TCP connections,

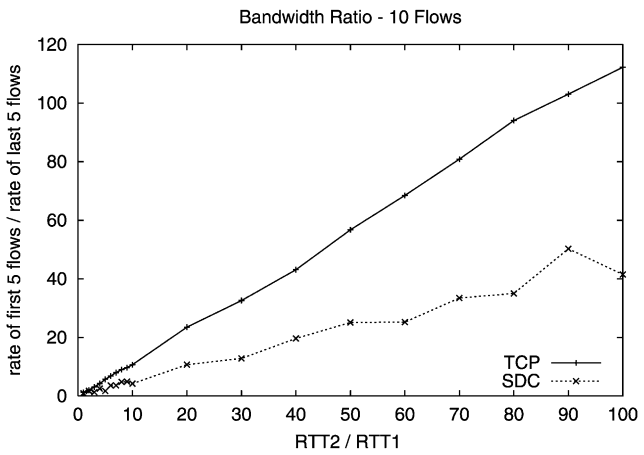

Fig. 9. Ratio of achieved bandwidths as a function RTT ratio (a total of 10 flows with five in each group).

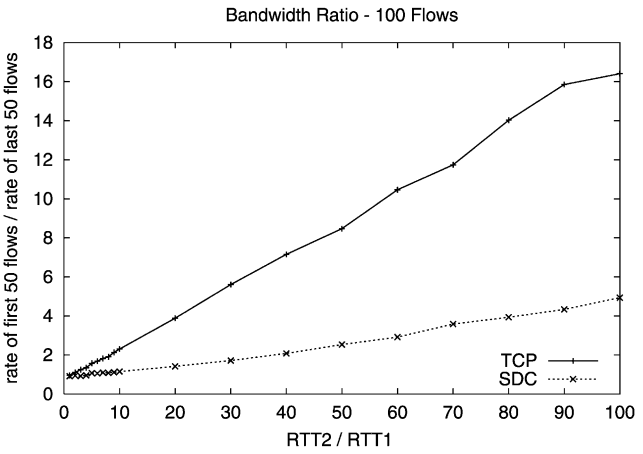

Fig. 10. Ratio of achieved bandwidths as a function RTT ratio (a total of 100 flows with 50 in each group).

since the former will add delay to RTT and this causes a slower ramp-up speed. Thus, SDC connections can be considered to be 'TCP friendly' [20] by design.

SDC can be made more aggressive by tuning how fast the TCP sender reduces the delay D to be added to RTT when receiving an ACK with the CE bit off. Consider the case when 50 SDC flows compete with 50 TCP flows on the configuration of Fig. 3. Fig. 11 shows that if D is reduced to $0.9 \times D$ then SDC connections get about 30% of the bandwidth of ECN connections. On the other hand, if D is reduced to $0.1 \times D$, then SDC connections get about 90% of the bandwidth of ECN connections.

When there is no congestion or no bandwidth competition, SDC achieves the same bandwidth throughput as traditional TCP.

3.6. Enhancement via the use of AVQ

We can further improve the performance of SDC in reducing timeouts by using adaptive AQM schemes, such as

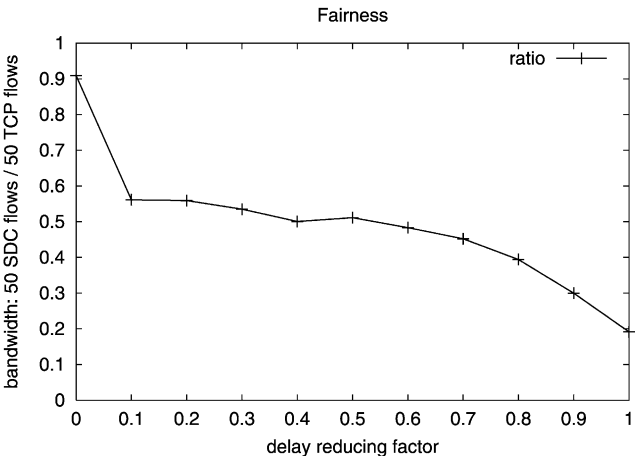

Fig. 11. When competing with TCP connections, SDC connections can be made increasingly competitive by reducing more aggressively the delay to be added to RTT when an ACK with the CE bit off is received.

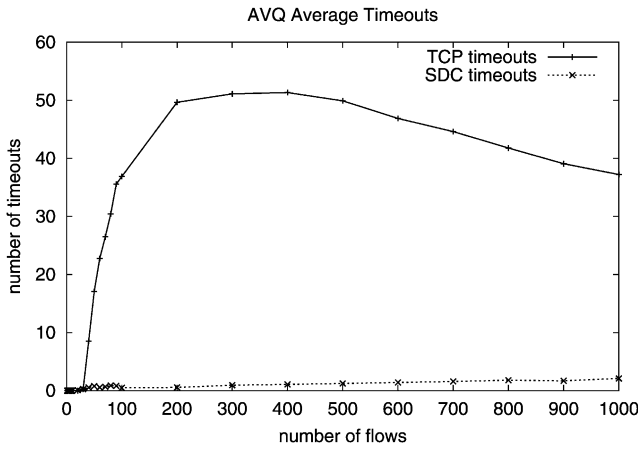

Fig. 12. Average number of timeouts as a function of number of flows under AVQ. This result represents an improvement over that of Fig. 4.

REM [21], adaptive RED [22,23], adaptive virtual queue (AVQ) [24], FRED [2], and SRED [25]. For illustration, we consider AVQ. AVQ limits the arrival rate to a fraction, e.g. 98%, of the output link capacity C. It uses virtual capacity $\tilde{C}$ to decide whether a packet should be marked with the CE bit. The virtual capacity is updated using

$$\frac{\mathrm{d}}{\mathrm{d}t}\tilde{C} = \alpha(\gamma C - \lambda)$$

where λ is the arrival rate, γ is the desired utilization at the link and α is a damping factor used to control the adaptive rate of AVQ. In this simulation, α and γ are set to be 0.12 and 0.98, respectively, as suggested in Ref. [24].

Fig. 12 shows the average number of timeouts as a function of number of flows under AVQ. We see the number of timeouts under SDC is further reduced, compared to the result of Fig. 4. The improvement is attributed to AVQ's ability in capturing TCP's congestion avoidance behavior.

4. Related work

Two methods were suggested in Ref. [1] to cure the timeout problem in the many-flow situation. The first method proposes that the size of the buffer deployed at the router should be proportional to the total number of active flows, instead of one round-trip time as suggested in Ref. [26]. Both FRED [2] and FPQ [6] take this approach. The second method makes TCP less aggressive and more adaptive when its congestion window is small. SUBTCP [22] is one such method. However, SUBTCP uses a multiplicative increase/multiplicative decrease algorithm and consequently will not converge to a fair point [9]. Additional simulation study of TCP with many flows can be found in Ref. [27].

There are extensive literature on enhancing TCP's loss recovery capability. Examples are TCP SACK [28] and TCP NewReno [15]. These methods can improve the loss recovery capability of TCP when the window size is sufficiently large. However, they are unable to offer much help when the window size is small. Some researchers have recognized this problem and proposed solutions [5,16,29]. For example, with Limited Transmit [16] the TCP sender can transmit a new segment after receiving two, instead of three, duplicate ACKs. This modification makes TCP less tolerant to packet reordering in network due to reasons such as the use of multiple links.

While we do not propose any new buffer management method in this paper, as shown in Section 3.6, SDC can take advantage of advanced AQM [7] methods to avoid global synchronization and to allow early congestion notification.

Recently, there has been work on TCP-friendly equation-based congestion control, such as TFRC [30]. We did some preliminary tests using TFRC. It seems that TFRC connections with the delay-bandwidth product smaller than two packets per connection cannot survive well when competing with other TCP connections. The equation [31] used by TFRC is perhaps too 'friendly' when the average bandwidth per connection is small.

SDC described in this paper can be viewed as a method of implementing Active Delay Control [32], an extension to TCP where TCP endpoints impose delays on the transmission of packets in order to improve performance. Basic concepts and motivations behind Active Delay Control and some of our previous results can be found in Refs. [32,33].

5. Conclusion

We have shown in this paper the feasibility of performing TCP congestion control by adjusting not only the congestion window size, but also the delay of packet sending. Using this additional delay-based control, SDC can keep the window size of a TCP connection above a certain threshold while decreasing its transmission rate. This reduces the frequency of TCP timeouts for the connection as well as the fluctuation in its bandwidth usage. For these reasons, SDC allows many TCP flows to share a link without excessive timeouts. In addition, because of the added delay, SDC reduces a well-known TCP bias against connections with relatively large RTTs.

Acknowledgements

This research was supported in part by NSF grant ANI-9710567, Air Force Office of Scientific Research Multidisciplinary University Research Initiative Grant F49620-97-1-0382, and grants from Microsoft Research, Nortel Networks, and Sun Microsystems.

References

[1] R.T. Morris, TCP behavior with many flows, IEEE International Conference on Network Protocols, Atlanta, Georgia (1997).

[2] D. Lin, R. Morris, Dynamics of random early detection, SIGCOMM'97, Cannes, France (1997) 127–137.

[3] S. McCreay, kc claffy, Trends in wide area IP traffic patterns, May 2000, http://www.caida.org/outreach/papers/2000/AIX0005/

[4] W.R. Stevens, TCP/IP Illustrated, The Protocols, vol. 1, Addison-Wesley, Reading, MA, 1993.

[5] D. Lin, H.T. Kung, TCP fast recovery strategies: analysis and improvements, Proceedings of the Conference on Computer Communications (IEEE Infocom), San Francisco, California (1998) http://www.ieee-infocom.org/1998/papers/02d_4.pdf.

[6] R.T. Morris, Scalable TCP congestion control, PhD thesis, Harvard University, 1999

[7] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, Recommendations on queue management and congestion avoidance in the Internet, RFC 2309 April (1998) ftp://ftp.isi.edu/in-notes/rfc2309.txt.

[8] K. Ramakrishnan, S. Floyd, A proposal to add explicit congestion notification (ECN) to IP, RFC 2481 January (1998) ftp://ftp.isi.edu/in-notes/rfc2481.txt.

[9] D.-M. Chiu, R. Jain, Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, Computer Networks and ISDN Systems 17 (1989) 14.

[10] V. Jacobson, Congestion avoidance and control, ACM Computer Communication Review, Proceedings of the SIGCOMM'88 Symposium in Stanford, CA 18 (4) (1988) 314–329. ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z.

[11] F.P. Kelly, Stochastic modes of computer communication systems, Journal of the Royal Statistical Society Series B 47 (3) (1985) 379–395.

[12] S. Floyd, TCP and explicit congestion notification, ACM Computer Communication Review 24 (5) (1994) 10–23.

[13] Y. Zhang, L. Qiu, Understanding the end-to-end performance impact of RED in a heterogeneous environment, Technical Report 2000-1802, Cornell, CS, July 2000

[14] UCB/LBNL/VINT, Vint network simulator—ns, 1997

[15] S. Floyd, T. Henderson, The NewReno modification to TCP's fast recovery algorithm, RFC 2582 April (1999) ftp://ftp.isi.edu/in-notes/rfc2582.txt.

[16] M. Allman, H. Balakrishnan, S. Floyd, Enhancing TCP's loss recovery using limited transmit, RFC 3042 January (2001) ftp://ftp.isi.edu/in-notes/rfc3042.txt.

[17] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE ACM Transactions on Networking 1 (4) (1993) 397–413.

[18] S. Floyd, Connections with multiple congested gateways in packet-switched networks part1: one-way traffic, ACM Computer Communication Review 21 (5) (1991) 30–47.

[19] S. Floyd, V. Jacobson, Traffic phase effects in packet-switched gateways, Internetworking: Practice and Experience 3 (3) (1992) 115–156.

[20] S. Floyd, K.R. Fall, Promoting the use of end-to-end congestion control in the internet, IEEE/ACM Transactions on Networking 7 (4) (1999) 458–472.

[21] S. Athuraliya, V.H. Li, S.H. Low, Q. Yin, REM: active queue management, IEEE Network 15 (3) (2001) 48–53.

[22] W. Feng, D.D. Kandlur, D. Saha, K.S. Shin, Techniques for eliminating packet loss in congested TCP/IP networks, Technical Report CSE-TR-349-97, U. Michigan, 4, 1997

[23] W. Feng, D.D. Kandlur, D. Saha, K.G. Shin, A self-configuring RED gateway, Proceedings of INFOCOM 99 3 (1999) 1320–1328.

[24] S. Kunniyur, R. Srikant, Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management, Proceedings of SIGCOMM 2001, San Diego, California, USA (2001).

[25] T.J. Ott, T.V. Lakshman, L.H. Wong, SRED: stabilized RED, Proceedings of INFOCOM 3 (1999) 1346–1355.

[26] C. Villamizar, C. Song, High performance TCP in ansnet, ACM Computer Communication Review 24 (5) (1994) 45–60.

[27] L. Qiu, Y. Zhang, S. Keshav, On individual and aggregate TCP performance, Proceedings of ICNP (1999).

[28] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP selective acknowledgment options, RFC 2018 October (1996) ftp://ftp.isi.edu/in-notes/rfc2018.txt.

[29] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, M. Stemm, R.H. Katz, TCP behavior of a busy internet server: analysis and improvements, Proceedings of the Conference on Computer Communications (IEEE Infocom) 1 (1998) 252–262.

[30] S. Floyd, M. Handley, J. Padhye, J. Widmer, Equation-based congestion control for unicast applications, Proceedings of SIGCOMM 2000, Stockholm, Sweden (2000) 43–56.

[31] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, Modeling TCP throughput: a simple model and its empirical validation, ACM Computer Communication Review 28 (4) (1998) 303–314.

[32] P.-H. Hsiao, H.T. Kung, K.-S. Tan, Active delay control for TCP, Proceedings of the IEEE Conference on Global Communications (GLOBECOM), San Antonio, TX (2001).

[33] P.-H. Hsiao, H.T. Kung, K.-S. Tan, Video over tcp with explicit delay notification, Proceedings of NOSSDAV 2001, Port Jefferson, New York, USA (2001).