Proceedings of *IEEE Globecom 2001*, November 2001

Active Delay Control for TCP

Pai-Hsiang Hsiao and H. T. Kung
Division of Engineering and Applied Sciences
Harvard University, USA

Koan-Sin Tan
Institute of Information Management
National Chiao-Tung University, Taiwan

***Abstract* -- Active Delay Control is a novel extension for TCP, where TCP endpoints impose delays on the transmission of packets to improve performance. The amount of delay can be calculated by routers from the level of congestions, or by endpoints from the received congestion signals. In particular, when there are many TCP flows competing for the bandwidth of a link, they can reduce their transmission rates to arbitrary degrees by increasing delays, without experiencing TCP time-outs. Active Delay Control is therefore useful for those long-lived TCP-based applications that can not tolerate time-outs. Examples of such applications are video streaming and storage networks. It is also useful for short-lived flows that require short transfer time. Examples of such applications are HTTP transactions. We present the concept and motivation behind Active Delay Control, and evaluate them by simulations.**

I. Introduction

TCP is widely used in Internet applications. There have been a great deal of research results on TCP in the literature [7]. However, a problem related to the many-flow case remains [12]. When the number of TCP flows sharing the link is sufficiently large, some of these flows will become "fragile" in the sense that they will be subject to frequent TCP time-outs. These time-outs will significantly degrade the performance perceived by end users.

We elaborate the challenge of devising solutions for this many-flow TCP problem. Our objective is that when *N* TCP flows compete on the same bottleneck link, each will get *1/N* of the link bandwidth over averaging intervals. It is important to use small averaging intervals, which need to be as small as few seconds, to reflect the responsiveness required by the interactive applications and to provide the steady throughput required by streaming applications. This means that these TCP flows must not experience TCP time-outs, as they could last seconds or longer. That is, after having passed the slow start phase, these flows will need to be kept in their congestion avoidance phase until the end of the connections [9].

During the congestion avoidance phase the rate of a TCP flow is determined by *CWND/RTT*, where *CWND* 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 decreases its *CWND* or increases its *RTT* to reduce the rate. Recall that *CWND* cannot be smaller than one packet. In fact, to avoid TCP time-outs, *CWND* needs to be larger than five or six packets to allow TCP fast retransmit and fast recovery to work [16]. To be truly "non-fragile", i.e., resilient to TCP time-outs, *CWND* needs to be a few packets more than five or six packets [10]. Several modifications have been proposed to reduce the required congestion window size, such as ECN [15] and Limited Transmit [1].

We suggest a new solution, Active Delay Control, to this problem. The routers notify endpoints when packets need to be delayed at endpoints during congestion, and endpoints delay either the transmission of data packets at the sender, or the generation of ACK packets at the receiver. This added delay increases the *RTT* of the flow, thereby allowing larger *CWND* to reduce time-outs. Since it is undesirable to reduce *CWND* below a certain limit as stated above, increasing *RTT* becomes necessary when *N* is sufficiently large.

The remainder of the paper is organized as follows. In Section II, we introduce the concepts and mechanisms of Active Delay Control. Then Section III evaluates the effectiveness of delay control with simulations using a sender-based delay control mechanism. We review related work in Section IV and conclude in Section V.

II. Active Delay Control

When it is expected that many flows will share a congested link, we propose Active Delay Control to avoid frequent time-outs. In this section, we introduce the concepts of Active Delay Control and some mechanisms for its implementation.

A. Concepts

A solution to cut down frequent time-outs is to keep *CWND* large enough, e.g., more than eight packets. It is possible to keep *CWND* large by increasing the buffer size of the router. However, this approach is undesirable and not scalable. With a larger buffer size, the buffer occupancy can grow higher during congestion, resulting in a larger queueing delay. This is undesirable because packets from all flows going through the shared buffer would suffer from the queueing delay. This approach is also not scalable because the size of the buffer must be determined by the number of active flows traveled through the router, which can be very large.

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.

Instead of using large buffer in the router, under Active Delay Control the transmission of packets are delayed in the endpoints to achieve the same effect.

Suppose there is a router with infinite buffer space, so the queueing delay can be arbitrarily large such that every flow passing through it can have a large enough *CWND*. The goal of Active Delay Control is to delay the packet at an endpoint for the same amount of time as the queueing delay the packet would experience when traveling through such a router. Thus, without relying on queueing delay in the router, the *RTT* can be increased to allow large *CWND*.

B. Mechanisms

There are two ways to implement delay control in endpoints. We briefly describe some mechanisms we have investigated and their issues.

The delay can be imposed at the receiver. That is, after receiving a data packet, the receiver will forward the received data immediately to the application, but defer the generation of the ACK packet until the required amount of delay has elapsed. The delay can also be imposed at the sender. In this case, after receiving the ACK packet, the sender does not process it until the amount of delay has elapsed. Out-of-order packet delivery and duplicate ACKs are handled as in traditional TCP.

We have investigated two receiver-based delay control mechanisms. The first one is called "Exact Receiver-based Delay Control (RDC)", and the second one is called "1-bit RDC". Both mechanisms are covered in [8] with detailed descriptions, so we only summarize them briefly in this section. Under Exact RDC, routers are responsible for computing necessary delays based on the congestion level. A rate-based queue management scheme is used to compute the exact amount of delay that the packet would experience when traveling through a traditional FIFO router. The amount of delay then be appended to the packet as delay notification. The receiver delays the ACKing by the amount of delay specified in the delay notification. With this mechanism, the receiver will delay the generation of the ACK by the same amount as that the packet would experience in a router with a large buffer. However, this mechanism requires new features in the router such as delay calculation.

In contrast, under 1-bit RDC, the receiver approximates the delay based on congestion signal generated by routers. A router uses active queue management algorithms (AQM) [2], such as RED [5], to generate congestion signal, and sends it to the receiver via ECN's Congestion Experience (CE) bit.

The receiver maintains a value as the amount of time to delay the ACKing of each packet. This value will increase when congestion signals are received from the network, and will decrease when no congestion signal is received. The value is increased multiplicatively and decreased additively so the resulting rate follows the principle of "Additive Increase and Multiplicative Decrease" (AIMD) [3] of TCP's congestion control algorithm [9]. This mechanism does not require additional features from routers beyond those required by ECN. A major drawback of 1-bit RDC is that for the precise update of the delay, it requires dynamic estimation of *CWND* and *RTT*.

We have also devised a sender-based delay control (SDC) mechanism. Similar to 1-bit RDC, SDC uses ECN CE bit to approximate the delay value; however, this is now done at the sender rather than the receiver. Since the sender maintains *CWND* and measures *RTT*, it can use these values to update the delay so that the rate change in SDC approximates that of traditional TCP. To allow large congestion window the sender will not halve its size when receiving CE; instead, the sender will increase the delay. When the congestion window is sufficiently large, e.g., above eight packets, the sender will disable delay control and resume window-based congestion control as in normal TCP. For the simulations in next section, we use this SDC mechanism for the delay control.

III. Evaluation of Underlying Concept

In the section, we evaluate the concept of delay control using simulations performed in *ns-2* [17]. We compare three approaches of increasing the congestion window size in order to reduce the time-out frequency. We show that SDC can reduce time-outs without introducing network delays to applications.

Theoretically, we could increase the congestion window size by extending the RTT of TCP flows with one of the following three methods: increasing the propagation delay of links, queue size of routers, or delay of packets at endpoints. Accordingly, in our simulation, we first show that increasing propagation delay of links helps reduce time-outs. Then, we show similar results by increasing queue size of the router. However, neither increasing propagation delay nor enlarging queue size of routers is desirable. We show, finally, that SDC can reduce time-outs even when queue size of the router is very small. In particular, we demonstrate that SDC is superior to traditional TCP and a couple of TCP's recent modifications.

A. Simulation Environment

All of the simulations in the paper were performed on a simple configuration depicted in Figure 1. In this configuration, there are *N* TCP flows merging at the router $\mathbf{G_0}$, and router $\mathbf{G_0}$ connects to router $\mathbf{G_1}$ via the bottleneck link **L**. In general, the simulations are intended to study the situation where the network path only allows few in-flight packets for each flow and thus, frequent time-outs is expected [12]. The default parameters of the simulations are summarized in Table 1.

In addition, we make the following assumptions about the protocols and the active queue management algorithms used in routers, unless otherwise is stated explicitly:

- Senders run the TCP NewReno algorithm that only halves the congestion window at most once per RTT [6]. More-

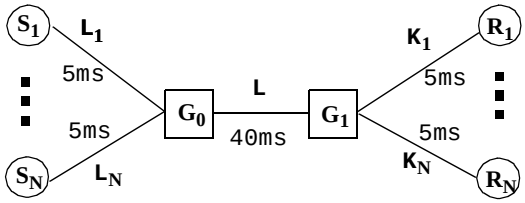

Figure 1: Simulation configuration. Originating from S_i, the *i*-th TCP flow travels over link L_i to router G_0, link L to router G_1, and link K_i to reach R_i. The link L between G_0 and G_1 is a bottleneck link. All links are symmetric and full-duplex.

TABLE 1 SUMMARY OF SIMULATION PARAMETERS

Parameter	Value
packet size	1000 bytes
bandwidth of link L	1.5 Mb/s
propagation delay between G_0 and G_1	40 ms
bandwidth of link L_i	10 Mbps
propagation delay between S_i and G_0	5 ms
propagation delay between G_1 and R_i	5 ms
queue size of router G_0	*b* packets
RED min thresh	*b* / 6
RED max thresh	*b* / 2
RED w_q	0.002
RED max_p	0.1
RED gentle_	true
simulation time	100 seconds

over, senders start with an initial CWND of two packets.

- Senders enable Limited Transmit. That is, if allowed by the congestion window and advertised window, a sender will transmit a new segment after receiving only two duplicate ACKs.
- All routers and TCP endpoints are ECN-Capable. Also, routers employ RED to perform the ECN marking.

B. Number of Time-outs as a Function of Propagation Delay

In this set of simulations, we vary RTTs of flows by changing propagation delays. The propagation delays between links from $\mathbf{S_i}$ to $\mathbf{G_0}$ and between $\mathbf{G_1}$ and $\mathbf{R_i}$ are set so that the RTT of each flow varies in the range between 100 ms and 500 ms in increments of 100 ms. We use 50 bulky TCP flows, which are all randomly started in the first second and the queue size of router $\mathbf{G_0}$ is set to be 100 packets (*b* equals to 100).

When RTT is 100 ms, the network allows a total of 118 in-flight packets. Thus, on average, each flow only has less than three in-flight packets. When the RTT is 500 ms, the network allows a total of 193 in-flight packets, so on average, each flow has about four in-flight packets.

As depicted in Figure 2, the average congestion window size increases as RTT grows. With a larger RTT, more in-flight

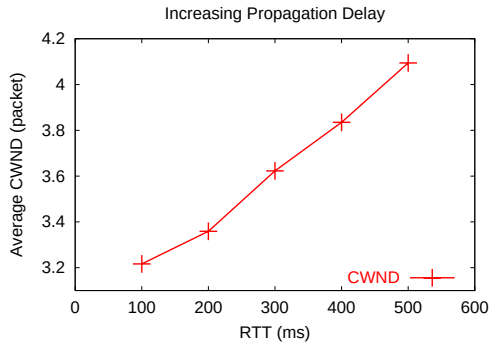

Figure 2: For TCP, average congestion window size grows as RTT increases. This is due to a larger RTT allows more in-flight packets for each flow.

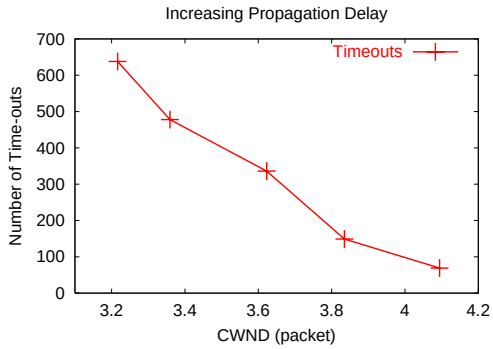

Figure 3: For TCP, the total number of time-outs decreases as the average congestion window size increases. When *CWND* is less than four packets, fast retransmit and fast recovery can not work and time-outs are frequent.

packets are allowed for each flow so the average congestion window size is increased. Figure 3 shows the total number of time-outs decreases as the average congestion window size increases.

We note here that a congestion window size of four packets, rather than five or six packets or more, is sufficient to make flows non-fragile. This is due to our use of ECN and Limited Transmit. When the congestion window is able to grow to four or more packets, most time-outs happen as the flows first started.

C. Number of Time-outs as a Function of Router Queue Size

In this set of simulations, we change the queue size of the router to increase RTT and thus CWND. This works because a larger queue size leads to a larger queueing delay and thus a larger RTT. We vary the queue size *b* of router $\mathbf{G_0}$ in the range from 100 to 1500 packets with steps of 100 packets.

There are 40 long-lived bulky TCP flows randomly started in the first second and 10 short-lived TCP flows started after 30 seconds. Each of the short-lived flows transfers 20 packets, terminates, then restarts. We measure the transfer time of short-lived flows as an indication of application delay caused by protocols.

Corresponding to Figure 2, Figure 4 shows that the average congestion window size for the 40 long-lived flows grows with

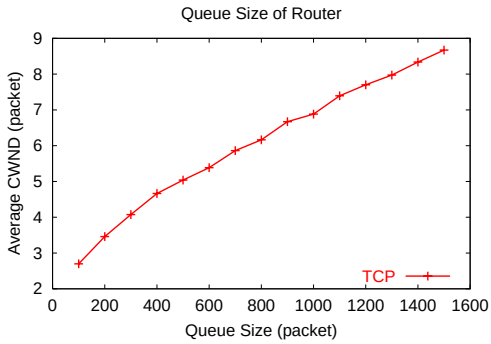

Figure 4: For TCP, the average CWND grows as queue size increases.

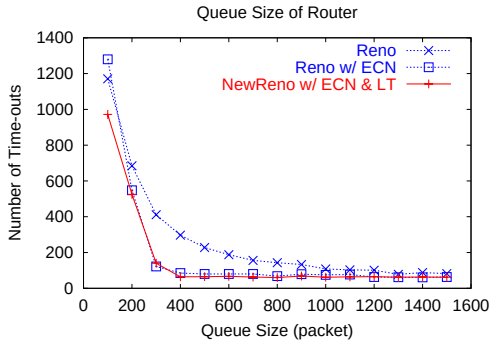

Figure 5: For three versions of TCP, the total number of time-outs decreases as queue size increases.

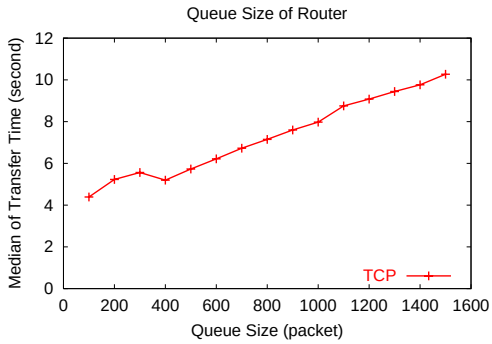

Figure 6: For TCP, the median transfer time increases as queue size increases. If the queue size is less than 400 packets, the median transfer time for short-lived TCP is larger due to frequent time-outs.

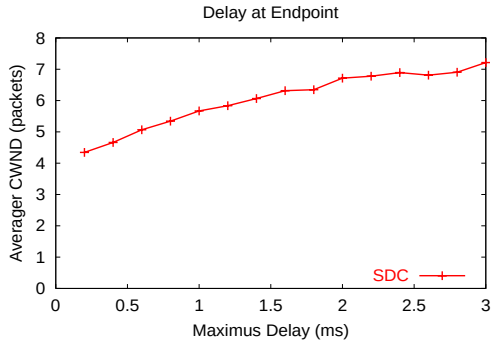

Figure 7: For SDC, the average congestion window size grows as maximum delay increases. With a maximum delay of 0.2 seconds, it is enough to increase the average congestion window from less than 3 (in Figure 4) to above 4.

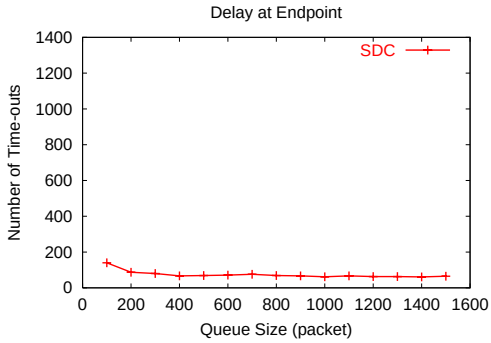

Figure 8: For SDC, only a smaller queue size is needed to avoid frequent time-outs, compared to the TCP case of Figure 5.

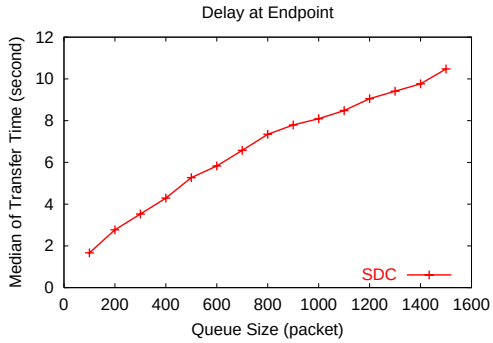

Figure 9: For SDC, short-lived flows on average have shorter transfer times for small queue sizes, compared with TCP flows of Figure 6.

the queue size. Note that when the queue size is larger than 400 packets, the average congestion window size grows to exceed four packets.

Figure 5 shows that increasing the queue size of router will decrease the total number of time-outs. Three versions of TCP are considered: TCP Reno, TCP Reno with ECN, as well as TCP NewReno with ECN and Limited Transmit.

For TCP Reno, the number of time-outs does not go below 200 until the queue size reaches 800 packets, when the path on average can hold more than five packets for each flow. After the queue size reaches 1200 packets, when the path can hold about seven packets for each flow, the performance is close to the other two versions of TCP. This means that the congestion window size of TCP Reno needs to be a few packets more than four to avoid frequent time-outs. We note that due to the RED algorithm used in the router, the queue is not fully occupied, and as a result, the average occupancy is much lower than queue size.

For TCP Reno with ECN, the total number of time-outs is significantly reduced only when the router has a moderate queue size, that is, 300 or more packets. The performance is not much better than TCP Reno for smaller queue sizes.

For TCP NewReno with ECN and Limited Transmit, it has fewer time-outs than TCP Reno with ECN for queue sizes smaller than 300 packets, and has similar numbers of time-outs for larger queue sizes. This is because for small queue sizes, Limited Transmit helps lower the total number of time-outs. For both versions of TCP, when the queue size is larger than 400 packets, the total number of time-outs stays flat. This is due to the fact that, in this case, the average CWND is already larger than four packets.

To measure the delay the protocols may cause toward the applications, we compute the transfer time of the short-lived 20-packet flows. Figure 6 shows that as the queue size increases the median transfer time for the 10 concurrent short-lived flows will increase. However, the median transfer time is larger when queue size is smaller than 400 packets, because in this case, the CWND is not large enough to avoid frequent time-outs. Note that frequent time-outs slows down the transfer more than the speed-up gained by shortened queueing delays.

On the other hand, for the case when the queue size is larger than 400 packets, the median transfer time increases linearly with the queue size. In this case, the queueing delay dominates the transfer time.

D. Performance Improvement of Delay Control at Endpoints

We use SDC to demonstrate the advantages of delaying packets at endpoints. In particular, we show that SDC has superior performance to TCP.

Under SDC, the sender can increase RTT by imposing delays to packet transmission. As depicted in Figure 7, the average congestion window size increases with the maximum delay at the sender, using the simulation configuration of Section III.C with queue size of router being 100 packets. Observe that when the maximum delay equals 0.2 seconds, SDC can increase the average congestion window size from less than three (in Figure 4) to more than four packets. When the maximum delay equals 2 seconds, the average congestion window size grows to seven packets.

Using the simulation configuration of the previous section, Figure 8 presents the total number of time-outs of SDC with the maximum delay of 2 seconds, as a function of queue size. The figure shows that while achieving the same level of time-out frequency as TCP, delaying packets at senders allows smaller queue size at router. This implies that SDC will introduce smaller queueing delay to applications.

In Figure 9, we show the median transfer times of SDC flows as a function of queue size. Note that SDC has shorter median transfer time than TCP, when the queue size is small. This is because delaying packets at the sender allows CWND to grow larger to avoid frequent time-outs, and consequently shorten transfer time. When the queue size is larger than 400 packets, SDC and TCP have similar numbers of time-outs, and thus have similar transfer time.

IV. Related Work

TCP congestion control has been extensively studied in the literature. An important issue in this area is TCP performance in terms of the time-out frequency when there are many competing flows [12][14]. However, not much work has been devoted to improve the performance for the many-flow case, where each flow does not have large enough congestion window to make fast retransmit and fast recovery work.

In [12], two approaches are proposed to address the frequent time-outs problem of TCP in the many-flow situation. The first one is based on provision of buffer space at router that is proportional to the total number of active flows. Both FRED [11] and FPQ [13] use this approach. In FRED, the router requires to have at least two packets for each active flow, so that flows run the Net Reno TCP [10] algorithm can avoid frequent time-outs. In FPQ, the router requires to have at least five packets for each active flow to avoid the same problem, without changing the congestion control algorithm. Both FRED and FPQ recognize the need of limiting queueing delays, by limiting queue occupancy of every active flow. But the total occupancy can still be large.

The second approach is to make TCP adapt better, when the number of flows is large and congestion window size is small, by modifying TCP's congestion control algorithm. An example of this approach is SubTCP [4]. However, it uses a multiplicative increase and multiplicative decrease algorithm, a scheme that is well known not able to converge [3].

V. Concluding Remarks

We have described the motivation and approach of Active Delay Control for TCP, as well as evaluated its concept using simulations. Simulation results show that Active Delay Control is effective in addressing the many-flow scenario of TCP. That is, it can reduce the frequency of time-outs when a large number of TCP flows are sharing a congestion link with small buffer space. Simulations also demonstrate that both long-lived and short-lived TCP flows benefit from the use of Active Delay Control. Other approaches, such as increasing the queue size of router or per-flow buffer provisioning, will introduce delays to applications and are not scalable.

The concept of Active Delay Control is to delay the transmission of packets or the ACKing of packets at the endpoints. For congestion control purposes, this endpoint delay has the same effect as the router queueing delay, but without introducing delays to applications.

The endpoint delay can be implemented at the TCP sender or receiver, as we have described in the paper. The method we used for the simulation is a sender-based delay control (SDC) mechanism, where the sender approximates the delay using congestion signal, the CE bit, sent by routers. This mechanism does not require additional features from routers beyond those required by ECN.

REFERENCES

[1] Allman, M., Balakrishnan, H., and Floyd, S. "*Enhancing TCP's loss recovery using Limited Transmit*," RFC 3042, January 2001.

[2] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., Zhang, L. "*Recommendations on queue management and congestion avoidance in the Internet*," RFC 2309, April 1998.

[3] Chiu, D., and Jain, R. "*Analysis of the increase and decrease algorithm for congestion avoidance in computer networks*," Journal of Computer Networks and ISDN, 17(1), June 1989.

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

[5] Floyd, S., and Jacobson, V. "*Random Early Detection gateways for congestion avoidance*," IEEE/ACM Transactions on Networking, 1 (4), August 1993.

[6] Floyd, S. and Henderson, T. "*The NewReno modification to TCP's fast recovery algorithm*," RFC 2582, April 1999.

[7] Floyd, S. "*A report on some recent developments in TCP congestion control*," IEEE Communications Magazine, April 2001.

[8] Hsiao, P. H., Kung, H. T. and Tan, K. S., "*Video over TCP with receiver-based delay control,*" in Proceedings of the NOSSDAV 2001, June 2001.

[9] Jacobson, V. "*Congestion avoidance and control*," Computer Communication Review, 18(4), 1988.

[10] Lin, D., and Kung, H. T. "*TCP fast recovery strategies: Analysis and improvements*," in Proceedings of the IEEE INFOCOM'98, April 1998.

[11] Lin, D., and Morris, R. "*Dynamics of Random Early Detection*," in Proceedings of the ACM SIGCOMM, September 1997.

[12] Morris, R. "*TCP behavior with many flows*," in Proceedings of the International Conference on Network Protocols (ICNP), October 1997.

[13] Morris, R. "*Scalable TCP congestion contro*l," Ph. D. thesis, Harvard University, 1999.

[14] Qiu, L., Zhang, Y., and Keshav, S. "*On individual and aggregate TCP performance*," in Proceedings of the International Conference on Network Protocols (ICNP), November 1999.

[15] Ramakrishnan, K. K. and Floyd, S. "*A proposal to add Explicit Congestion Notification (ECN) to IP*," RFC 2481, 1999.

[16] Stevens, W. R. "*TCP/IP illustrated, volume 1: The protocols*," Addison-Wesley, 1994.

[17] UCB/LBL/VINT. "*Network simulator — ns*," http://www.isi.edu/nsnam/ns/.