Protecting TFRC from a Selfish Receiver

Manfred Georg Sergey Gorinsky

Applied Research Laboratory
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO 63130-4899, USA

{*mgeorg, gorinsky*}*@arl.wustl.edu*

Abstract

This paper examines operation of TFRC (TCP-Friendly Rate Control) in scenarios where the receiver is untrustworthy and can misbehave to receive data at an unfairly high rate at the expense of competing traffic. Several attacks are considered for a selfish receiver to take advantage of TFRC. After confirming experimentally that the identified receiver attacks are effective, we design Robust TCP-Friendly Rate Control (RTFRC), a TFRC variant resilient to the attacks. We also show that additional attacks targeted directly at RTFRC are unable to compromise the protocol.

1. Introduction

As multimedia applications gain importance in the Internet, providing them with appropriate congestion control becomes vital to the overall stability of Internet communications. Unfortunately, traditional TCP (Transmission Control Protocol) congestion control [1, 11] exhibits two features that are detrimental to multimedia applications. First, the use of retransmissions to provide in-order reliability introduces extra delay, which is undesirable for multimedia. Second, the high variability of TCP transmission rates over short timescales undermines human perception of audio and video. TFRC (TCP-Friendly Rate Control) [4, 10] is a promising alternative that addresses these concerns. TFRC offers no support for reliable delivery and transmits data at smooth rates that remain fair to TCP over long timescales. In addition to purely unicast communications, TFRC is successfully used as a component of overlay systems for multicast data dissemination, such as Bullet [12].

Whereas multimedia servers have an interest in fair distribution of offered content to all their clients, an individual receiver has incentives to maximize its own share of the bottleneck link bitrate. Hence, receivers may misbehave to obtain an unfair share of the network capacity at the expense of competing traffic. Furthermore, the Internet architecture contains no safeguards against attacks by a selfish receiver. In particular, since TFRC has an open-source application-level implementation [10], it is extremely easy for a selfish receiver to deviate from the specifications in order to acquire data at an unfairly high rate.

Receiver misbehavior is a problem that is not unique to TFRC. Savage et al. show how incorrect feedback enables a misbehaving TCP receiver to increase substantially its throughput at the expense of cross traffic [3, 15]. Protection of TCP from receiver misbehavior relies on the elegant idea of a cumulative nonce: the TCP receiver must prove in-order delivery of data segments by providing the sender with XOR values of random numbers (nonces) that the sender has attached to the data segments.

Due to fundamental differences in the designs of TCP and TFRC, protecting TFRC against receiver misbehavior poses new challenges. Since TFRC separates reliability from congestion control and does not retransmit lost data segments, the TCP solution of a cumulative nonce is not directly applicable to TFRC. Also, unlike in TCP where the receiver sends acknowledgments upon delivery of data segments, feedback in TFRC is asynchronous from delivery and includes aggregate information which is difficult to verify.

In this paper, we first identify and experimentally validate vulnerabilities of TFRC to selfish receiver misbehavior. Then, we propose Robust TCP-Friendly Rate Control (RTFRC), a TFRC variant that is resilient to the identified receiver attacks. Our evaluation of RTFRC confirms that the new design renders the attacks ineffective. The source code of our application-level RTFRC implementation is made freely available [7].

The rest of the paper is organized as follows. Section 2 describes TFRC. Section 3 presents our threat model and experimentally demonstrates vulnerabilities of TFRC

Proceedings of the Joint International Conference on Autonomic and Autonomous Systems
and International Conference on Networking and Services (ICAS/ICNS 2005)
0-7695-2450-8/05 $20.00 © 2005 IEEE

to selfish receiver misbehavior in a real network. Section 4 derives RTFRC, our robust version of TFRC. Section 5 analyzes the protection offered by RTFRC. Finally, Section 6 provides a summary of the paper.

2. TFRC

TCP cuts its transmission rate at least in half in response to even a single packet loss. TFRC offers smoother transmission that suits multimedia applications better [4]. To share the network capacity with TCP cross traffic fairly despite the differences in congestion response, a TFRC connection determines its transmission rate based on the TCP throughput equation [14]:

$$X = \frac{s}{R\sqrt{\frac{2bp}{3}} + 3 \cdot p \cdot t_{RTO}\sqrt{\frac{3bp}{8}}(1+32p^2)} \qquad (1)$$

where X denotes the fair transmission rate, s is the average packet size, R represents RTT (round-trip time), t_{RTO} denotes the retransmission timeout, b is the number of data packets acknowledged by a single feedback packet, and p is the loss event rate. A loss event is defined as one or more packet losses within a single RTT. To avoid undesirable delay added by TCP in-order reliability, TFRC offers no support for reliable delivery.

The role of the receiver in TFRC is more prominent than in TCP. To enable the sender to compute the transmission rate, the receiver measures the loss event rate and reports it in feedback packets. Feedback also echoes the timestamps of data packets, thereby relieving the sender from storing these values. To understand why TFRC offloads as much work as possible to the receiver, one should recall that TFRC emerged in conjunction with TCP-Friendly Multicast Congestion Control (TFMCC) [17]. Due to the asymmetry of multicast communications, minimizing the sender involvement is a rational design choice in TFMCC. Furthermore, trustworthy environments offer no reasons for picking a different split of responsibilities between the sender and receiver in the related TFRC.

TFRC feedback comprises four fields: (1) timestamp of a data packet, (2) time passed since the data packet was delivered, (3) loss event rate, and (4) receiver rate, i.e. the rate of data packet delivery. The receiver rate is reported to avoid excessive transmission into a congested network: TFRC limits its transmission to twice the receiver rate.

3. Evaluation of TFRC vulnerabilities

While the original TFRC design assumes trustworthy participants, this assumption of universal trust is no longer tenable in the Internet. In this section, we relax this assumption and demonstrate that an untrustworthy receiver can exploit TFRC to acquire data at an unfairly high rate.

3.1. Threat model

Although TFRC trusts the receiver, RFC 3448 admits that TFRC "may potentially be manipulated by a greedy receiver that wishes to receive more than its fair share of network bandwidth. A receiver might do this by claiming to have received packets that were lost due to congestion" [10]. We explore the possibility of such receiver misbehavior in more detail. We assume that the only goal of the untrustworthy receiver is to acquire its data at an unfairly high rate at the expense of competing traffic [8, 9]. Our threat model does not include purely malicious attacks. In particular, we do not consider denial-of-service attacks where a receiver congests the network by transmitting spurious packets or terminates other connections by spoofing their control packets.

3.2. Experimental methodology

To evaluate vulnerabilities of TFRC to receiver misbehavior, we conduct experiments in ONL (Open Network Laboratory) network testbed built around extensible two-gigabit routers [2, 13]. ONL enables an experimenter to configure network parameters such as topology, link capacities, buffer sizes, and queuing disciplines. We experiment with a simple two-link topology where a link with capacity 15 Mbps is followed by a bottleneck link with capacity 7.54 Mbps. All link buffers are FIFO (First-In First-Out) and Droptail. Link buffer sizes are configured to accommodate 88.4 KB of IP data, which is equivalent to about sixty 1500-byte IP datagrams. Each experiment involves seven parallel connections between a pair of hosts. Of the seven connections, five are TCP NewReno [6], another is well-behaving TFRC, and the last one is TFRC with a misbehaving receiver. Instead of using the standard TFRC implementation [5], we implement TFRC to be consistent with our code for RTFRC [7]. This choice eliminates implementation-specific differences between TFRC and RTFRC in our experimental results.

We deviate from common simulation setups by using natural propagation delays, which are very small in ONL. However, our buffer sizes are selected so that the queuing delay at the bottleneck link is at least 48 ms after initial convergence; this is roughly equivalent to having a configuration with 48 ms link propagation delay and a bitrate-delay-product buffer. In future extensions of this work, we plan to emulate larger propagation delays explicitly at network nodes.

Proceedings of the Joint International Conference on Autonomic and Autonomous Systems
and International Conference on Networking and Services (ICAS/ICNS 2005)
0-7695-2450-8/05 $20.00 © 2005 IEEE

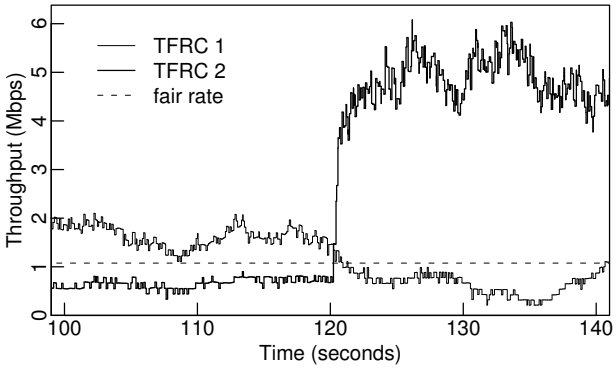


Figure 1. Exploit of the loss event rate

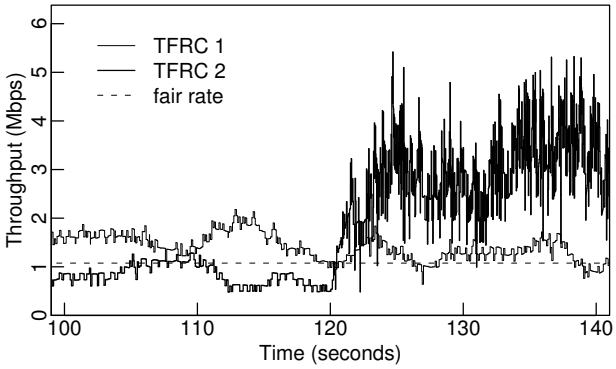


Figure 2. Exploit of RTT calculations

3.3. Assessment of specific attacks

Based on the TFRC feedback format, we identify and evaluate three types of attacks where the receiver manipulates the loss event rate, RTT calculations, and reported receiver rate respectively.

Manipulating the loss event rate is the most direct and brutal assault on TFRC. By underreporting the loss event rate, a misbehaving receiver can easily deceive the sender into transmitting at an unfairly high rate. The misbehavior can be implemented by changing a single line of the TFRC code. Figure 1 shows an instance of the attack. Both displayed connections adhere to TFRC until 120 seconds into the experiment and transmit at close-to-fair rates. After 120 seconds, the receiver of one TFRC connection misrepresents the loss event rate by reporting a value that is 32 times smaller than the actual rate. The graphs confirm that the misbehaving receiver succeeds in boosting the transmission rate of its connection, mostly at the expense of the five parallel TCP connections. After the misbehavior starts, the packet loss rate changes from 1.13% to 1.5% and from 3.23% to 5.97% for the well-behaving and misbehaving TFRC connections respectively.

Manipulating RTT calculations is another potent way to deceive the sender. The sender computes RTT based on the echoed timestamp and t_{delay}, which is the delay between arrival of a data packet to the receiver and departure of the feedback packet. The receiver can abuse both fields to trick the sender into underestimating RTT and consequently transmitting at an unfairly high rate. A side effect of decreasing the RTT estimate is a lower sender timeout value. A misbehaving receiver can easily avoid undesirable timeouts by sending its feedback more frequently in accordance to the lowered RTT estimate. Since the TFRC sender explicitly tells its RTT estimate to the receiver, the receiver can precisely control calculations at the sender. However, the receiver should exercise care in not deflating the RTT estimate too much: behavior of the sender under a negative RTT estimate is not specified by TFRC and depends on the implementation. Figure 2 shows an attack where the receiver distorts RTT calculations by overstating t_{delay}. The misbehavior starts 120 seconds into the experiment and results in an RTT estimate that is one fourth of the actual value. Once again, the receiver increases the transmission rate of its TFRC connection at the expense of the five parallel TCP connections. After the misbehavior starts, the packet loss rate changes from 1.06% to 1.99% and from 3.92% to 5.64% for the well-behaving and misbehaving TFRC connections respectively.

Manipulating the receiver rate allows the receiver to circumvent the limit imposed by the sender on the transmission rate. The receiver can use this attack to increase the transmission rate acceleration during slow start, under massive losses, and after quiescent periods. In our experiments, we were unable to translate potential transient benefits from manipulating the receiver rate into any noticeable long-term advantage for a misbehaving receiver.

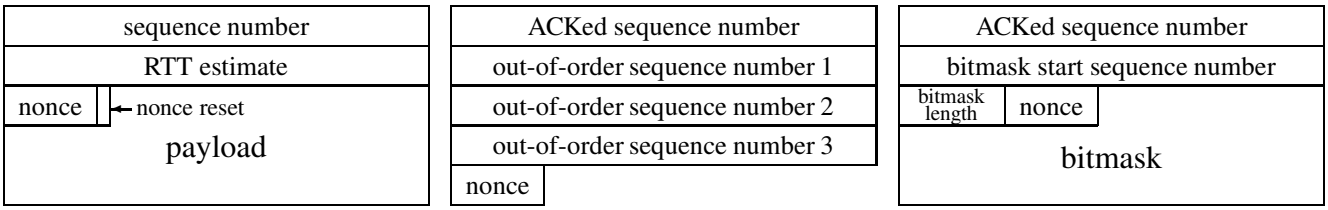
4. Robust TFRC

Section 3 demonstrated vulnerabilities of TFRC to receiver misbehavior. We now enhance the protocol to make it resilient to the identified attacks. Our Robust TCP-Friendly Rate Control (RTFRC) combines two ideas to provide this protection: computations are shifted from the receiver to the sender, and feedback is verified at the sender. Below, we discuss in detail how RTFRC applies these ideas.

4.1. Protecting the loss event rate

An intuitive way to protect TFRC from manipulations of the loss event rate is to verify correctness of receiver reports at the sender. However, verifying the loss event rate is difficult both due to the complex definition and aggregate nature of the loss event rate. Instead, we move the computation of the loss event rate from the receiver to the sender, which is an option mentioned in RFC 3448 [10]. Under our assumption of an untrustworthy receiver, this new split of

Proceedings of the Joint International Conference on Autonomic and Autonomous Systems
and International Conference on Networking and Services (ICAS/ICNS 2005)
0-7695-2450-8/05 $20.00 © 2005 IEEE



(a) Data packet (b) Normal ACK (c) Bitmask ACK

Figure 3. Packet header formats in RTFRC

responsibilities has substantial advantages in terms of design simplicity.

With the sender computing the loss event rate, the challenge shifts to enabling the sender to verify whether the receiver has received a particular data packet. To achieve this goal, we design a cumulative nonce mechanism similar to those used for robust TCP [3, 15, 16]. Specifically, since TFRC does not support reliable in-order delivery, we adopt a scheme that works despite loss of nonces [16]. Hence, we allow a cumulative nonce to confirm a data range that is not necessarily contiguous or aligned with the beginning of the message.

Adding nonces leads to different packet formats in RTFRC. Figure 3a shows the format of RTFRC data packets. The header is simple and includes only a sequence number, RTT estimate, packet nonce, and nonce reset flag. RTFRC uses two types of acknowledgment (ACK) packets. As Figure 3b shows, normal ACKs contain sequence numbers for up to three data segments received out of order. These fields enable accurate computation of RTT despite potential packet reordering in the network. A normal ACK also reports a cumulative nonce ensuring that the receiver cannot conceal loss of a data packet. When recovering from loss events, RTFRC uses bitmask ACKs. To acknowledge a noncontiguous range of data, a bitmask ACK specifies the beginning and length of the range as well as a bit vector identifying the data packets that the receiver has obtained within the range. As Figure 3c shows, the bitmask ACK also reports the cumulative nonce for the packets received from the new range. In adherence to RFC 3448, our nonce scheme preserves the definition of a loss event as one or more packet losses within a single RTT. Furthermore, the scheme helps the sender to determine the receiver rate accurately even when congestion prevents the receiver from obtaining all data packets.

We also introduce a mechanism enabling the sender to reset the cumulative nonce explicitly. When the sender learns of a loss event, the sender puts a new nonce and sets the nonce reset flag in the header of the next data packet. After the packet arrives, the receiver has to consider the packet as the beginning of a new data range for feedback purposes.

When suggesting a sender-based variant of TFRC, RFC 3448 argues for feedback via a reliable delivery mechanism. We believe that adding a reliable feedback channel is an unnecessary burden. Therefore, RTFRC does not use retransmissions or correction codes for its feedback. Our experience shows that incorrect inference of loss events due to loss of feedback packets does not disrupt RTFRC performance.

4.2. Protecting RTT calculations

A misbehaving receiver can manipulate RTT calculations in TFRC by modifying the timestamp of a data packet or by overstating t_{delay}. To fend off the first type of attack, the sender stores timestamps locally instead of transmitting them to be echoed by the receiver. To protect against the second type of manipulation, we also eliminate the t_{delay} field from feedback by requiring the receiver to send a feedback packet only in immediate response to a data packet.

4.3. Protecting the receiver rate

Although our experiments do not confirm that a misbehaving receiver can gain any long-term advantage from manipulating the receiver rate, it is prudent to fix vulnerabilities before successful exploits are discovered. Luckily, protecting the receiver rate in RTFRC does not require extra communication: our mechanism for protecting the loss event rate provides the sender with sufficient information to compute the receiver rate in a straightforward and robust manner. Thus, we also move the computation of the receiver rate to the sender.

4.4. Summary of new design features

The main difference between TFRC and RTFRC lies in shifting the computation of the loss event rate and receiver rate from the receiver to the sender as well as in using a cumulative nonce over a potentially noncontiguous data range to verify feedback. Consequently, RTFRC has different formats for packet headers.

Minor changes include a reduction of the sender timeout value from 4 RTT to 2.5 RTT, providing RTFRC with a tighter control loop. The smaller timeout value reduces

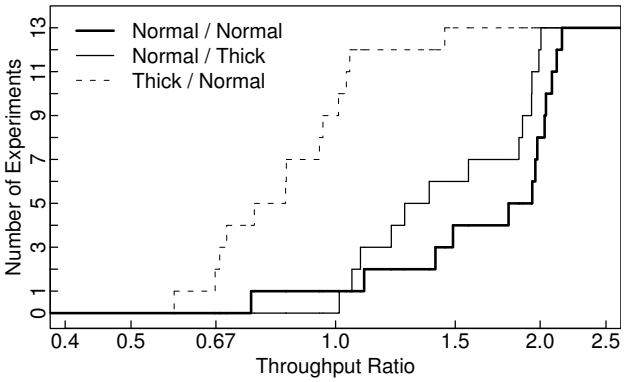


Figure 4. Thick feedback attack

the amount of delay that a misbehaving receiver can impose on feedback without causing a timeout. The particular value of 2.5 RTT is chosen to avoid spurious timeouts. The timeout value should be at least 2 RTT to tolerate loss of a single feedback packet. Furthermore, the timeout value should be large enough to cover variations in RTT. For environments with small propagation delays, variations in RTT can be comparatively large; adding a small constant (e.g. 10 ms) to the timeout value effectively eliminates spurious timeouts in such situations. Since adding a small constant does not undermine the robustness of RTFRC when propagation delays are large, we recommend this as a general rule for setting the sender timeout value in RTFRC.

5. Analysis of RTFRC resilience

In this section, we assess resilience of RTFRC to receiver misbehavior. Since RTFRC is made robust against the earlier presented attacks by its design, we now focus on new RTFRC-specific attacks.

5.1. Nonce guessing

Our nonce scheme prevents the receiver from concealing a loss event. Although the receiver can attempt to guess a nonce, the probability of guessing the nonce correctly equals $1/2^b$ where b is the nonce size in bits. The probability is small even for small values of b. Furthermore, to provide the receiver with a disincentive for nonce guessing, RTFRC gives a sender the option of terminating the connection if the receiver submits an incorrect nonce.

5.2. Thick feedback

Sending feedback at a higher rate presents a potential opportunity for increasing the data transmission rate. For example, since the sender doubles its transmission rate every RTT during slow start, more frequent feedback can increase the acceleration of transmission rate.

To evaluate whether and how thick feedback affects RTFRC throughput, we conduct ONL experiments with two RTFRC connections that run in parallel with five TCP NewReno connections. We examine three experimental configurations. In the first, both RTFRC connections provide feedback at the standard rate of one packet per RTT. In the second and third configurations, one RTFRC connection sends feedback at four times the normal rate while the other uses the standard feedback rate. The difference between the last two configurations is the order in which the normal-feedback and thick-feedback connections start. For each configuration, we repeat the experiment 13 times.

Figure 4 presents the ratio of steady-state throughputs of the earlier and later RTFRC connections. As with TFRC, the steady-state throughput in RTFRC depends on the initial state. When the thick-feedback connection starts after the normal-feedback connection, the throughput ratio is statistically similar to the results for the first configuration where both RTFRC connections send feedback at the normal rate. However, when the thick-feedback connection starts earlier, it acquires a smaller share of the bottleneck link capacity in the steady state, thereby punishing itself. In general, adoption of thick feedback tends to bring the throughput ratio closer to one. Therefore, thick feedback does not enable the RTFRC receiver to boost its network capacity consumption unfairly at the expense of well-behaving cross traffic.

5.3. Feedback timing

Thick feedback belongs to a larger class of attacks where the receiver violates feedback procedures, by sending spurious feedback packets, changing feedback timing, or not providing feedback at all. Although similar attacks can be targeted at TFRC, we discuss them in the context of RTFRC because a misbehaving TFRC receiver has easier means to manipulate the sender and thus lacks incentives for launching these less effective attacks.

RTFRC is immune to some feedback-timing attacks by its design. For example, nonce-based verification of feedback neutralizes early feedback that tries to confirm data that has not yet been delivered. Delaying or withholding feedback about packet losses allows the receiver to postpone the loss detection at the sender. Such loss concealment is only temporary since the sender times out if no feedback is provided. Furthermore, its short-term benefits are mitigated by the smoothness of RTFRC transmission. Additionally, delayed feedback inflates the RTT estimate at the sender and thereby further decreases the transmission rate; this detrimental effect offers a strong disincentive against the attack.

In connections where RTT varies a lot, more frequent feedback during low-RTT intervals and less frequent feedback during high-RTT intervals can decrease the sender

RTT estimate because the sender computes the estimate as a weighted moving average. In its turn, the deflated RTT estimate yields a higher transmission rate. To launch this attack, the receiver needs to track the RTT estimate reported by the sender in data packets. Also, the receiver needs to be sufficiently precise in estimating the true RTT and sufficiently prompt in adjusting the feedback frequency; the last two constraints appear to be difficult. In general, the selfish receiver in our RTFRC experiments were not able to translate any temporary advantages from feedback-timing attacks into tangible long-term benefits.

6. Conclusion

This paper investigated operation of TFRC in scenarios where the receiver is untrustworthy and can misbehave to receive data at an unfairly high rate at the expense of competing traffic. We identified and experimentally demonstrated vulnerabilities of TFRC to a selfish receiver's misbehavior. Then, we designed Robust TCP-Friendly Rate Control (RTFRC), a TFRC variant resilient to the identified receiver attacks. We also showed that additional attacks targeted directly at RTFRC are unable to compromise the protocol.

Acknowledgments

We would like to thank Michael Wilson for stimulating discussions over the design of RTFRC, Hariharan Iyer for invaluable help in analyzing data, and the anonymous ICAS/ICNS 2005 reviewer for detailed and insightful comments.

References

[1] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, April 1999.

[2] S. Choi, J. Dehart, R. Keller, F. Kuhns, J. Lockwood, P. Pappu, J. Parwatikar, W. D. Richard, E. Spitznagel, D. Taylor, J. Turner, and K. Wong. Design of a High Performance Dynamically Extensible Router. In *DARPA Active Networks Conference and Exposition*, May 2002.

[3] D. Ely, N. Spring, D. Wetherall, S. Savage, and T. Anderson. Robust Congestion Signaling. In *Proceedings IEEE ICNP 2001*, November 2001.

[4] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *SIGCOMM 2000*, pages 43–56, August 2000.

[5] S. Floyd, M. Handley, J. Padhye, and J. Widmer. TFRC, Equation-based Congestion Control for Unicast Applications: Simulation Scripts and Experimental Code. http://www.aciri.org/tfrc/, February 2000.

[6] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782, April 2004.

[7] M. Georg. Robust TCP-Friendly Rate Control Implementation. http://www.arl.wustl.edu/∼mgeorg/rtfrc/, May 2005.

[8] S. Gorinsky, S. Jain, and H. Vin. Multicast Congestion Control with Distrusted Receivers. In *Proceedings NGC 2002*, October 2002.

[9] S. Gorinsky, S. Jain, H. Vin, and Y. Zhang. Robustness to Inflated Subscription in Multicast Congestion Control. In *Proceedings ACM SIGCOMM 2003*, August 2003.

[10] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 3448, January 2003.

[11] V. Jacobson. Congestion Avoidance and Control. In *Proceedings ACM SIGCOMM 1988*, August 1988.

[12] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *19th ACM Symposium on Operating System Principles (SOSP 2003)*, October 2003.

[13] Open Network Laboratory. http://onl.arl.wustl.edu/, May 2005.

[14] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proceedings ACM SIGCOMM 1998*, September 1998.

[15] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP Congestion Control with a Misbehaving Receiver. *Computer Communication Review*, 29(5):71–78, October 1999.

[16] N. Spring, D. Wetherall, and D. Ely. Robust Explicit Congestion Notification (ECN) Signaling with Nonces. RFC 3540, June 2003.

[17] J. Widmer and M. Handley. Extending Equation-Based Congestion Control to Multicast Applications. In *Proceedings ACM SIGCOMM 2001*, August 2001.