Congestion Control with a Misbehaving Receiver: Robust TFRC and Other Protocols

Manfred Georg, Sergey Gorinsky

Abstract— **This paper examines the operation of TFRC (TCP-Friendly Rate Control) in scenarios where the receiver is untrustworthy. By misbehaving, a TFRC receiver can obtain a higher data rate at the expense of competing traffic. We identify and experimentally confirm several such attacks and designed Robust TCP-Friendly Rate Control (RTFRC), a TFRC variant which is resilient to receiver misbehavior. We also show that additional attacks that are based on feedback timing and targeted directly at RTFRC are unable to compromise the protocol. We discuss existing and propose new techniques for protecting congestion control protocols from receiver misbehavior in general. The discussion includes analysis of what level of protection is feasible with different amounts of feedback. Finally, we explore methods that compress feedback reports without undermining their verifiability.**

I. 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], [2] exhibits two features that are detrimental to multimedia applications. First, the use of retransmissions to provide in-order reliability introduces extra delay. Second, the high variability of TCP transmission rates over short timescales undermines streaming audio and video. TFRC (TCP-Friendly Rate Control) [3], [4] was designed to address these concerns. It 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 [5].

A misbehaving receiver can lie to the sender of a TFRC session in order to maximize its throughput at the expense of competing traffic. Specific attacks by the receiver include manipulating the loss event rate, Round Trip Time calculation, and reported received rate. Furthermore, the Internet architecture contains no safeguards against attacks by a selfish receiver. In this paper, we explore how to protect TFRC and similar protocols from receiver misbehavior. In particular, we present Robust TCP-Friendly Rate Control (RTFRC) [6], a robust version of TFRC. While our original proposal of RTFRC sketches its main design features, this paper validates properties of the protocol more thoroughly and extends the advocated protection techniques to a general class of congestion control protocols.

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 [7], [8]. Protection of TCP from receiver misbehavior relies on 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. There are also attacks in Explicit Control Protocol (XCP) which are difficult to protect against because of router participation in the protocol [9].

Protecting TFRC against receiver misbehavior is inherently more difficult than in TCP. Unlike TCP, TFRC separates reliability from congestion control and does not retransmit lost data segments, causing cumulative nonces to not be 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.

Datagram Congestion Control Protocol (DCCP) [10], [11], [12] partially addresses the receiver misbehavior vulnerabilities of TFRC. Similarly to TFRC, DCCP does not support reliable delivery. For congestion control, DCCP offers the user a choice of multiple profiles. In particular, the chosen congestion control can be TCP-like or TFRC-like. To protect itself against receiver misbehavior, DCCP employs a nonce bit by default. The protection is similar in general but less robust than our approach in RTFRC. The other differences between TFRC and DCCP's TFRC-like profile are unimportant to the problem studied in this paper.

The rest of the paper is organized as follows. Section II describes TFRC. Section III presents our threat model and experimentally demonstrates vulnerabilities of TFRC to selfish receiver misbehavior in a real network. Section IV introduces RTFRC, our robust version of TFRC which is resilient to the identified receiver attacks. Section V analyzes the protection offered by RTFRC through experiments in a real high-speed network. Section VI discusses general approaches to protecting protocols from receiver misbehavior and analyzes how small feedback summaries can be used most effectively for such protection. Section VII provides the paper with a summary.

II. 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 [3]. In order to share the network capacity fairly with TCP despite different behavior in response to congestion, a TFRC connection determines its transmission rate based on the TCP throughput equation [13]:

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

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 the undesirable delay caused

This work was performed at the Applied Research Laboratory, Department of Computer Science and Engineering, Washington University in St. Louis, One Brookings Drive, St. Louis, MO 63130-4899, USA by Manfred Georg (mgeorg@cse.wustl.edu) and Sergey Gorinsky (gorinsky@arl.wustl.edu).

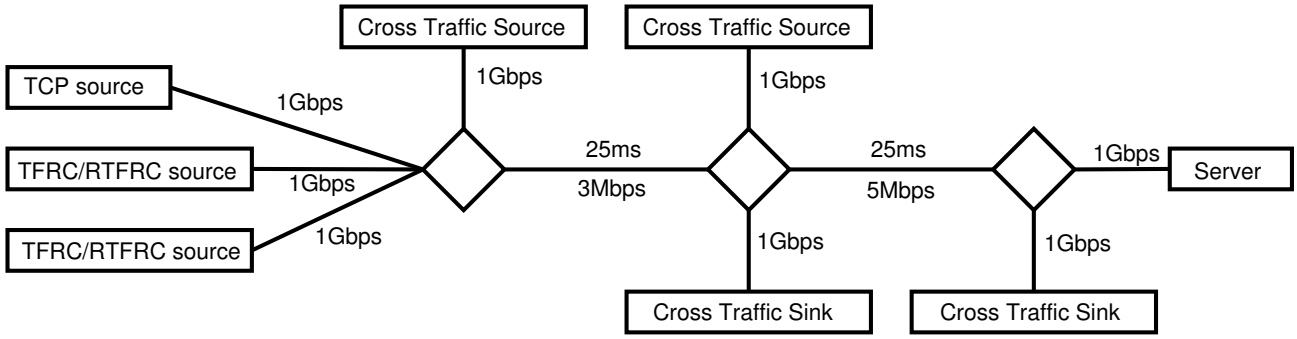


Fig. 1. Experimental topology.

by retransmission of lost packets, TFRC offers no support for reliable delivery.

As a result of TFRC being developed in conjunction with TCP-Friendly Multicast Congestion Control (TFMCC) [14], the role of the receiver in TFRC is more prominent than in TCP. For example, the receiver measures the loss event rate and includes it in feedback packets, allowing the sender to compute the transmission rate. Feedback also echoes the timestamps of data packets, thereby relieving the sender from storing these values. In TFMCC, due to the asymmetry of multicast communications, minimizing the sender involvement is a rational design choice. Furthermore, assuming a trustworthy environment, there was no reason to change this split of responsibilities between sender and receiver when TFRC was developed from TFMCC. However, a misbehaving receiver can easily exploit this design by sending misleading feedback packets.

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.

III. Evaluation of vulnerabilities in TFRC

While the original TFRC design assumes trustworthy participants, this assumption of universal trust is not 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.

A. 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" [4]. 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 [15], [16]. 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.

B. Experimental methodology

To evaluate vulnerabilities of TFRC to receiver misbehavior, we conduct experiments in the ONL (Open Network Laboratory), a network testbed built around extensible two-gigabit routers [17], [18]. The ONL enables an experimenter to configure network parameters such as topology, link capacities, buffer sizes, and queuing disciplines. We experiment in a topology with two core links, as shown in Figure 1. The first link has a capacity of 3 Mbps while the capacity of the second one is 5 Mbps. The second link becomes a bottleneck only when additional traffic traverses it. All link buffers are FIFO (First-In First-Out) and Droptail. The propagation delay of every link, in both directions, is 25 ms and is implemented by hardware in the routers. The bottleneck buffer sizes are set to 37.5 kB, corresponding to one bandwidth-delay product.

We allocate the first 10 seconds of each experiment to stabilize the network conditions before any misbehavior is introduced. After the misbehavior starts, we allow an additional 5 seconds to re-stabilize the network conditions and only then begin recording results. The measurements last for 60 seconds. The first set of our experiments involves one well-behaving TFRC connection, one misbehaving TFRC connection, and a variable number of TCP SACK connections [19], [20]. All connections transmit in parallel and terminate at one host. The well-behaving and misbehaving TFRC connections transmit from two different computers while all the TCP connections originate from a third host. We use a standard implementation for TFRC [21] but implement RTFRC from scratch [22]. In subsequent sets of experiments, we introduce various types of cross traffic on the core links.

C. 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 effective attack 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 2 shows an instance of the attack. Two TFRC flows compete with five TCP flows. After 10 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.

We generalize the above to a set of experiments where the TFRC receiver underreports its loss event rate to varying degrees. Figure 3 shows the throughput observed by the two TFRC and the average of 5 TCP flows in these experiments. For each configuration, the experiment is repeated 10 times, and the results are summarized with box plots. The boxes

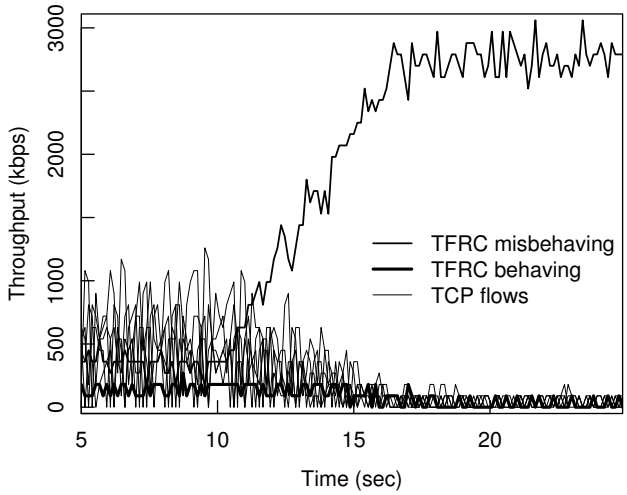


Fig. 2. Underreporting loss event rate by a factor of 32 starting at time 10

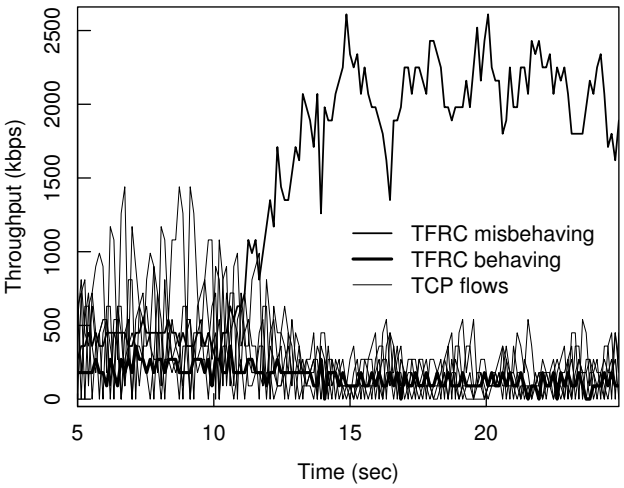


Fig. 4. Underreporting RTT by a factor of 64

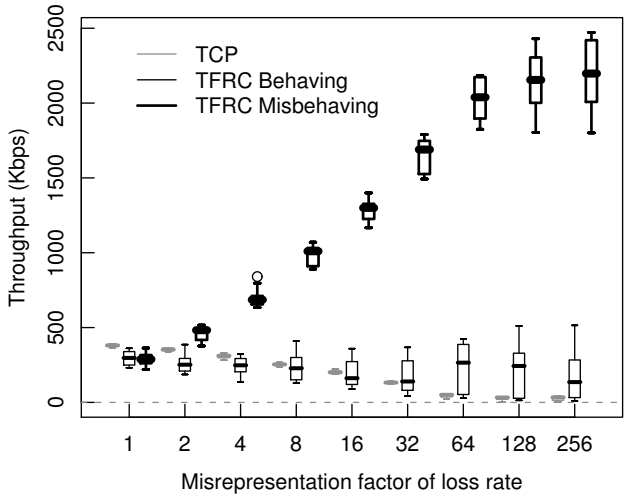


Fig. 3. Underreporting loss event rate

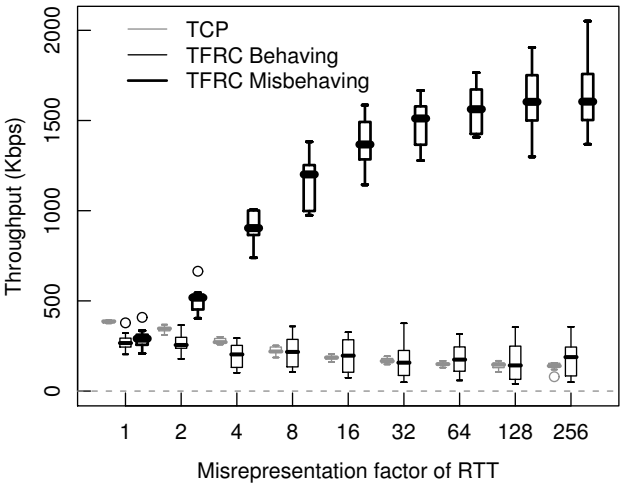


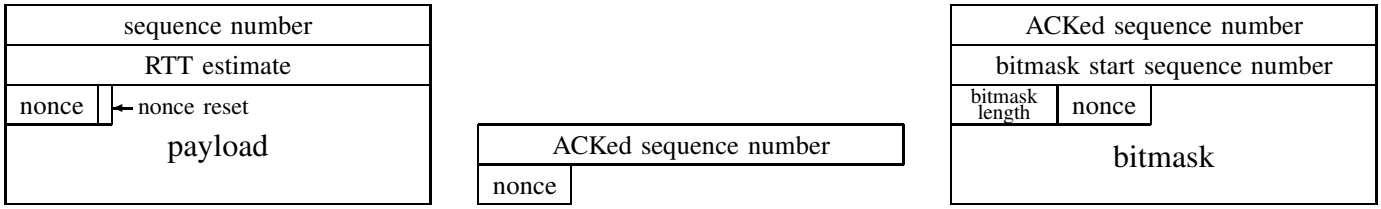
Fig. 5. Underreporting RTT

extend from the first to the third quartile, and the whiskers extend to the furthest data point within half an interquartile range of the box; outliers are plotted individually as circles. To improve readability, the boxes depicting the results for the TCP and misbehaving TFRC connections are staggered slightly along the horizontal axis of the graph. The height of the TCP boxes is much smaller since the displayed values are averages for the 5 TCP flows. The variability in throughput is greater during underreporting because the link is being utilized more aggressively, which causes more severe loss events where all connections backoff dramatically. Figure 3 confirms that understating the loss event rate is a potent attack that enables the misbehaving TFRC receiver to boost its connection throughput significantly (by a factor of 5) while almost starving the parallel well-behaving TCP and TFRC traffic. The differences in throughput with no underreporting are caused because TCP is able to recover better from losses at low capacity.

Manipulating RTT calculations is another potent way of deceiving 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 misrepresent 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 with 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, since behavior of the sender under a negative RTT estimate is not specified by TFRC but depends on the implementation. Figure 4 shows an attack where the receiver distorts RTT calculations by overstating t_{delay}. The misbehavior starts 10 seconds into the experiment and results in an RTT estimate that is one 64th of the actual value. Once again, the receiver increases the transmission rate at the expense of competing traffic; however, the attack is not as effective and predictable as a direct attack on the loss event rate.

Figure 5 shows the throughput observed by a misbehaving TFRC flow in a series of experiments where the misbehaving TFRC receiver distorts the RTT estimate by a factor plotted on the horizontal axis of the graph. Once again, we run the experiment 10 times for each configuration and summarize the results in box plots. The plots reveal that the RTT attack is also dangerous and allows the misbehaving TFRC receiver to obtain more than half of the bottleneck link capacity.

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,



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

Fig. 6. Packet header formats in RTFRC.

we were unable to translate potential transient benefits from manipulating the receiver rate into any noticeable long-term advantage for a misbehaving receiver. However, manipulating this parameter value enables the TFRC receiver to keep the packet loss rate above 50%, which might be used to launch a powerful denial of service attack.

IV. Design of Robust TFRC

The previous section 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) protocol 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. The source code of our application-level RTFRC implementation is available online [22].

A. Protecting the loss event rate

In TFRC the loss event rate is computed by the receiver and sent explicitly in feedback. Verifying the loss event rate sent by an untrustworthy receiver is difficult both due to its complex definition and the aggregate nature of loss events. In RTFRC, we simplify the situation by moving the computation of the loss event rate from the receiver to the sender; this idea is briefly mentioned in RFC 3448 [4].

With the sender computing the loss event rate, the main challenge lies in verifying whether the receiver has received data packets. To achieve this goal, we design a cumulative nonce mechanism similar to those used for robust TCP [7], [8], [23]. In particular, since TFRC does not support reliable in-order delivery, we adopt a scheme that works despite loss of nonces [23]. 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 6a 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. The first, shown in figure 6b is a normal ACK which cumulatively acknowledges all data up to a sequence number, with a cumulative nonce which ensures that the receiver cannot conceal loss of a data packet. When packets are not received in order or when a loss occurs, RTFRC uses bitmask ACKs. To acknowledge an incontiguous 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 6c 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 creates 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.

B. 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.

C. 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, our mechanism for protecting the loss event rate also allows the sender to robustly compute the receiver rate. Thus, we move the computation of the receiver rate to the sender.

D. 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 incontiguous data range to verify feedback. 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 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

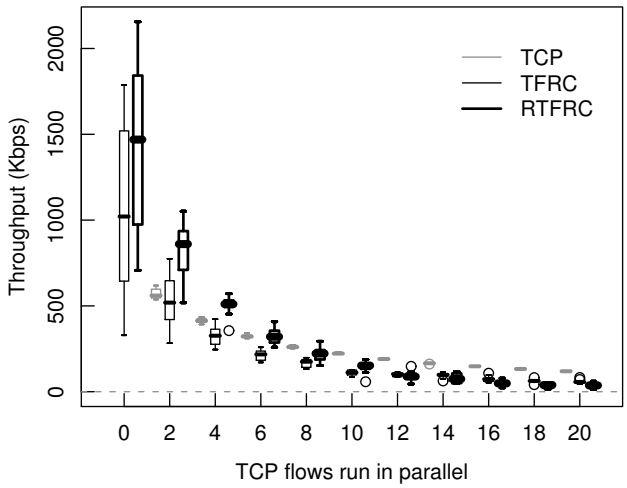


Fig. 7. RTFRC and TFRC with TCP cross traffic: all flows traverse both core links.

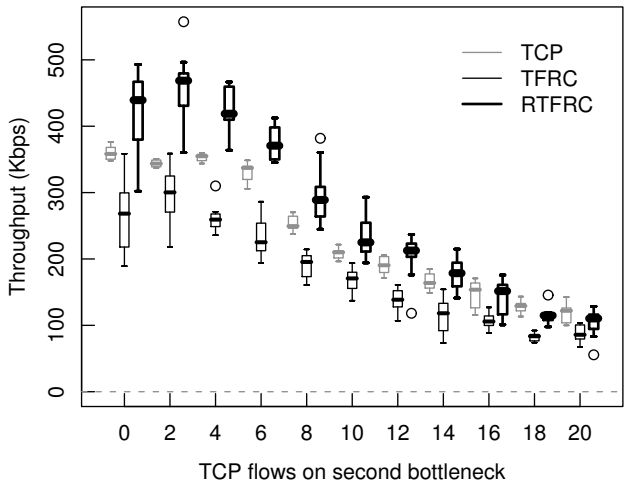


Fig. 8. RTFRC, TFRC, and 5 TCP flows on both core links when additional TCP flows congest second core link.

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.

E. Performance of RTFRC

Our transformation of TFRC into RTFRC is geared toward improving the latter's security without undermining its performance and fairness. To verify this we conduct three series of experiments with one well-behaving RTFRC connection, one well-behaving TFRC connection, and different types of well-behaving TCP cross traffic. As before, we always repeat every experiment 10 times in each configuration and summarize the results with box plots.

Due to implementation differences RTFRC has a different average throughput than TFRC. The cause of this difference was not isolated, but might arise from the use of a b value of 1 in RTFRC. This number represents the number of TCP packets acknowledged by a single feedback packet in the throughput equation; the reference TFRC implementation uses a value of 2. Additional differences occur in keeping information about loss events, and from history discounting, the standard TFRC implementation has several choices for these parts of the algorithm; we use their default configuration.

First, we conduct an experimental set with the same cross-traffic pattern as in Section III, meaning that the TCP cross traffic runs in parallel with TFRC and RTFRC along the two core links. We vary the number of parallel TCP connections from 0 to 20. Figure 7 shows that as the number of flows grows, the average throughput of the RTFRC connection declines proportionally to the fair rate of the first core link and stays close to the average throughput of TFRC and TCP connections.

The second series of experiments keeps the number of parallel TCP connections on both core links at 5 but introduces from 0 to 20 extra TCP cross traffic connections exclusively on the second core link. Figure 8 plots the average throughput of TFRC, RTFRC, and five TCP flows sharing the first core link. The graph shows that as the number of extra TCP flows increases, moving the bottleneck for the TFRC and RTFRC connections from the first core link to the second core link, RTFRC remains fair to TFRC and TCP.

Our last set of experiments in well-behaving RTFRC environments examines the impact of cross traffic in the reverse direction. In addition to the original five TCP, one TFRC, and one RTFRC flows running in parallel, we also launch extra TCP cross traffic on the reverse direction of the second core link. We vary the number of TCP flows on the reverse direction from 0 to 50. Figure 9 demonstrates that creating congestion on the reverse path of the RTFRC connection does not dramatically disrupt RTFRC efficiency and its fairness to TFRC and TCP. In general, RTFRC and TFRC are unaffected by congestion on the reverse path while TCP, which requires more acknowledgment packets and is more directly affected by the RTT of the connection reduces its throughput moderately.

V. Evaluating RTFRC specific attacks

Since RTFRC is made robust against the earlier presented attacks by design, this section focuses on new RTFRC-specific attacks.

A. Resilience to 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 is small ($1/2^k$ where k is the nonce size in bits). 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.

B. Resilience to excessive 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 as long as it receives feedback acknowledging all data, more frequent feedback might increase the acceleration of the transmission rate.

To evaluate whether excessive feedback affects RTFRC throughput, we conduct ONL experiments in the same manner and topology as presented earlier, where two RTFRC connections run in parallel with 5 TCP connections. We examine the behavior of RTFRC when the feedback rate is varied. Figure 10 plots the throughput of a excessive feedback RTFRC flow to a normal RTFRC flow and an average of the 5

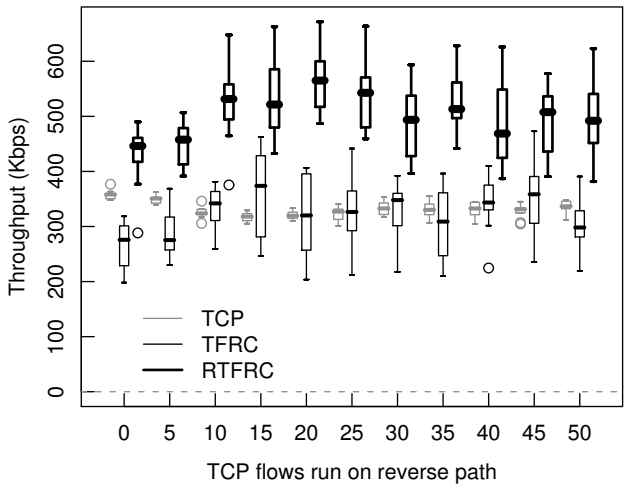


Fig. 9. RTFRC, TFRC, and 5 TCP flows when additional TCP flows congest the reverse path.

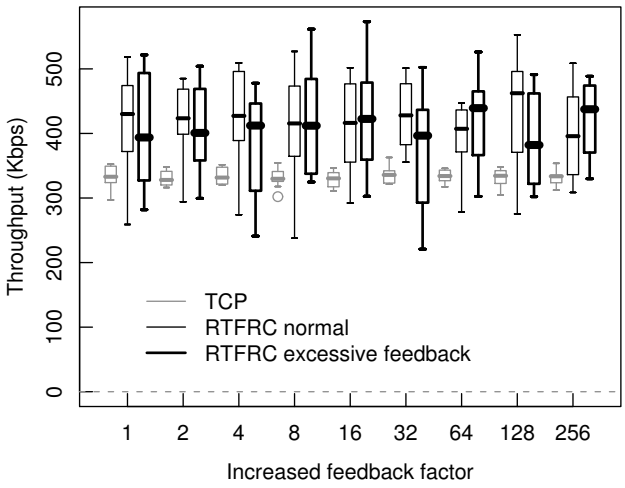


Fig. 10. Excessive feedback attack.

background TCP flows. The excessive feedback flow reports feedback more frequently than normal by a factor plotted on the x-axis. A comparison of these results with Figure 7 shows that the observed values and their variation are not unusual. It is obvious from these experiments that excessive feedback does not enable the RTFRC receiver to boost its network capacity consumption unfairly at the expense of well-behaving cross traffic.

C. Feedback timing

Excessive 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 of manipulating the sender.

RTFRC is immune to some feedback-timing attacks by its design. For example, a feedback packet cannot be sent before a packet has been delivered since it must include its nonce value. 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 decreases the transmission rate; this detrimental effect offers a strong disincentive against the attack.

The RTT value is computed as a weighted moving average of the RTT value included in packets. In connections with widely varying RTT, a receiver can artificially decrease the RTT estimate by providing more frequent feedback during low-RTT intervals and less frequent feedback during high-RTT intervals. This 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 are difficult. In general, we were unable to translate any temporary advantages from feedback-timing attacks into tangible long-term benefits for a selfish receiver. Furthermore, it is unclear how to completely eliminate this vulnerability: simply scaling the extent of the update by how long has elapsed since the last update could lead to unstable situations. Since this attack is not disastrous, we prefer to ignore it and keep the protocol as simple as possible.

VI. Discussion

In the previous sections, we focused on protecting TFRC. Now, we extend our study to discuss the general problem of protecting congestion control protocols against a misbehaving receiver. Our main objective is to provide efficient and fair transmission of data. Achieving this goal involves two tasks: (1) accurate measurement of the network state and (2) effective enforcement of fair bandwidth sharing among competing flows. We start by considering both router-supported and end-to-end approaches to solving the general problem. Then, we analyze the degree of protection feasible with limited congestion-control feedback from the receiver.

A. Router-centric techniques

A router is able to directly monitor the congestion state of the network. However, it is resource intensive to observe all flows to determine which flows are responsible for congestion. Fair queuing schemes such as WDRR [24], WF2Q [25], SCFQ [26], and SRR [27] place flows in independent queues to ensure that they receive a fair share of the bandwidth as determined by an idealized Generalized Processor Sharing (GPS) [28] schedule. The only circumvention of this technique is to open a number of parallel connections, a vulnerability fundamental to flow-based identification and out of the scope of this investigation. Another practical concern about fair queuing schemes is their requirement to maintain per-flow state. Due to this overhead, such schemes are not widely deployed.

XCP [29], RCP [30], and SFQ [31] represent alternative router-centric approaches that do not require per-flow state in routers. To avoid this prohibitive overhead, SFQ approximates WDRR by using a fixed number of aggregating queues instead of a separate queue for each flow. In XCP and RCP, each router monitors its traffic and inserts explicit congestion information directly into forwarded packets. Since the router does not transmit the congestion feedback information directly to the sender but relays the information through the receiver, a misbehaving receiver can manipulate or simply ignore feedback

to deceive the sender to transmit at an arbitrary rate. Hence, these protocols need an additional mechanism for protection against receiver misbehavior. One straightforward fix of this problem is to require routers to send their congestion feedback directly to the sender; however, the possibility of asymmetric paths complicates this situation. Furthermore, direct feedback to the sender from each router on a path will cause an implosion effect where multiple control packets are sent in response to a single data packet. A cryptographic approach of signing control information (e.g., by using an intricate nonce scheme such as Fanfare [32]) also requires a substantial amount of overhead. Despite these obstacles, router-centric approaches are inherently more effective since they do not assume trustworthiness of end hosts.

B. End-to-end techniques

End-to-end approaches to protecting against receiver misbehavior must determine the state of the network through external measurements. Generally, this task takes the form of measuring the loss event rate. In such settings, it is common to define fairness in terms of normal TCP behavior: anything that behaves like TCP is considered to be fair. This definition lacks rigor since TCP has many variants which behave differently. However, it simplifies determination of a fair rate. Modeling the behavior of TCP under various network conditions is an area of ongoing research [33], [13]. As in TFRC, one can use the TCP throughput equation to translate a loss event rate directly into a fair rate.

The chief challenge with the reformulated problem is that only the receiver has all the information needed to compute the loss event rate correctly. As in our RTFRC solution, the sender can include a nonce in each data packet and require that the receiver returns the nonces to the sender. Also, a single cumulative nonce can compactly represent multiple nonces when the network is not congested. However, when losses do occur, it becomes difficult to have concise feedback that enables the sender to determine exactly the loss event rate at the receiver. One straightforward solution is to require the receiver to echo every received nonce; this is what RTFRC does. This feedback can be sent either in individual ACK packets for each nonce or, with a significant reduction in overhead, by aggregating multiple nonces into a single ACK (RTFRC does the latter). In both cases, the amount of information included in acknowledgments is directly proportional to the amount of sent data; hence, both approaches are scalable. For most scenarios this constitutes an acceptable solution.

To facilitate deployment of resilient congestion control protocols, Kuzmanovic and Knightly propose that packets be intentionally discarded as a backward-compatible alternative to nonce-based protection [34]. Unfortunately, this degrades protocol performance. Even if calculation of the transmission rate ignores purposeful losses, it will still introduce introduce substantial jitter due to packet retransmission. Furthermore, waiting for retransmissions can cause buffers to fill. The proposal also includes a sender-specific change that emulates the behavior of the receiver in order to verify that the receiver is acting as expected. Sadly, this method of verification is at least as computationally expensive as performing calculations at the sender in the first place. Furthermore, the two main advantages of receiver centric protocols: that the sender is simplified and that information only available to the receiver can be incorporated in the protocol are both violated.

C. Protection with small feedback summaries

In network settings with asymmetric connections, multicast sessions, and servers with many connections, it might be useful to decrease the congestion feedback overhead. Although RTFRC aggregates its feedback for many data packets into a single ACK, the nonce for every packet is still communicated individually, without any compression. On the other hand, a compressed summary of nonces has the potential of reducing the amount of feedback to less than a linear function of the number of data packets.

An interesting research objective is to find an algorithm allowing the sender to verify the loss rate for n sent packets based on receiver feedback of less than $O(n)$ bits. A general technique such as the Shamir thresholding scheme [35] or various error correction codes can be used to verify a specific loss rate. However, verification of an arbitrary loss rate is a more difficult challenge.

By only verifying the loss rate within a factor of 2, we can acheive constant feedback size, $O(1)$. Include $\log_2 n + 1$ independent nonce bits in the data packets, assuming that n is a power of 2. For the sequence of first bits in the nonces, include no redundancy. For the sequence of the i-th bit in the nonces, include redundancy in the amount of 2^{i-1} using an error correction code. The above can be accomplished through the use of a Hamming code with a predetermined number of bits known beforehand [36]. If no packets are lost, then a hash of the first nonce bits can be used to verify this fact. If more than 2^i but less than 2^{i+1} packets are lost, then the receiver can reconstruct the sequence of i-th bits in the nonces and send a hash of this sequence back to the sender. If more than 2^{i+1} packets are lost, the sequence of i-th nonce bits cannot be reconstructed because there is not enough redundancy. Therefore, the solution enables verification of the loss rate within a factor of 2 and minimizes the size of the feedback. For multicast connections, precomputed hashes can be used to verify reports from each receiver quickly. This technique can be easily generalized to verify the loss rate to within a different factor k simply by increasing the number of bits sent; the generalization requires $O(\log_k n)$ control bits in the forward direction and $O(1)$ for feedback.

When emulating TCP, one is commonly concerned about the loss event rate, not the packet loss rate. Generally, enough time passes between loss events to verify each individually. To do this with minimal overhead, logically split a stream of packets into sections. In each packet, include a piece of the aggregate nonce for its section. If each section covers half a RTT, then a loss event is detected by incorrect aggregate nonces for two or less consecutive sections. This is reasonably close to the original definition of a loss event as one or more packet losses within one RTT. A closer approximation of this definition can be obtained by decreasing the duration for each section of packets.

The two previous schemes can be combined to allow the sender to measure the loss event rate with a certain granularity and verify it to within a factor of 2. Since loss events do not always cause packet losses in consecutive sections, a misbehaving receiver can underreport the loss rate slightly so that enough sections still pass verification at the sender. Nevertheless, this technique can prevent the most egregious misbehavior and is useful for occasionally verifying that the receiver is not grossly underreporting its loss event rate. The amount of feedback bandwidth required to implement this technique is smaller than that used to maintain a TCP

connection.

VII. Conclusion

This paper investigated operation of TFRC in scenarios where the receiver is untrustworthy. The modification of feedback packets allows a misbehaving receiver to easily gain an unfairly high transmission rate at the expense of competing traffic. We experimentally demonstrated the effectiveness of attacks based on manipulating the loss event rate and RTT. Then, we designed Robust TCP-Friendly Rate Control (RTFRC), a TFRC variant resilient to the identified receiver attacks. We also showed that RTFRC has similar properties to TFRC in well-behaving environments and is resilient to additional attacks targeted specifically at it. We extended the protection techniques used in RTFRC to create a general family of congestion control protocols. Finally, we presented several protection methods with compressed feedback from the receiver and analyzed the degree of resilience offered by these methods.

References

[1] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581, April 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2581.txt

[2] V. Jacobson, "Congestion Avoidance and Control," in *ACM SIGCOMM*, August 1988.

[3] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," in *ACM SIGCOMM*, August 2000, pp. 43–56.

[4] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 3448, January 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3448.txt

[5] 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.

[6] M. Georg and S. Gorinsky, "Protecting TFRC from a Selfish Receiver," in *IEEE International Conference on Networking and Services*, October 2005.

[7] D. Ely, N. Spring, D. Wetherall, S. Savage, and T. Anderson, "Robust Congestion Signaling," in *IEEE ICNP*, November 2001.

[8] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, "TCP Congestion Control with a Misbehaving Receiver," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 5, October 1999, pp. 71–78.

[9] C. Wilson, C. Coakley, and B. Y. Zhao, "Fairness Attacks in Explicit Control Protocol," in *IWQoS*, 2007.

[10] S. Floyd, E. Kohler, and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)," RFC 4342, March 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4342.txt

[11] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC 4340, March 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4340.txt

[12] ——, "Designing DCCP: Congestion Control Without Reliability," in *SIGCOMM*, 2006.

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

[14] J. Widmer and M. Handley, "Extending Equation-Based Congestion Control to Multicast Applications," in *ACM SIGCOMM*, August 2001.

[15] S. Gorinsky, S. Jain, and H. Vin, "Multicast Congestion Control with Distrusted Receivers," in *NGC*, October 2002.

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

[17] 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.

[18] "Open Network Laboratory," http://onl.arl.wustl.edu/.

[19] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP," RFC 2883, July 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2883.txt

[20] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, October 1996. [Online]. Available: http://www.ietf.org/rfc/rfc2018.txt

[21] 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.

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

[23] N. Spring, D. Wetherall, and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces," RFC 3540, June 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3540.txt

[24] M. Shreedhar and G. Varghese, "Efficient Fair Queueing using Deficit Round Robin," in *ACM SIGCOMM*, August 1995, pp. 231–242.

[25] J. C. Bennett and H. Zhang, "WF2Q: Worst-case Fair Weighted Fair Queueing," in *IEEE INFOCOM*, March 1996, pp. 120–128.

[26] S. Golestani, "A Self-Clocked Fair Queuing Scheme for Broadband Applications," in *IEEE INFOCOM*, June 1994, pp. 636–646.

[27] S. Ramabhadran and J. Pasquale, "Stratified Round Robin: A Low Complexity Packet Scheduler with Bandwidth Fairness and Bounded Delay," in *ACM SIGCOMM*, August 2003, pp. 239–249.

[28] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks – The Multiple Node Case," *IEEE Transactions on Networking*, vol. 2, no. 2, 1994, pp. 137–150.

[29] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *ACM SIGCOMM*, August 2002.

[30] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor Sharing Flows in the Internet," in *International Workshop on Quality of Service*, June 2005.

[31] P. E. McKenney, "Stochastic Fairness Queueing," in *IEEE INFOCOM*, June 1990, pp. 733–740.

[32] E. Shi, B. Parno, A. Perrig, Y.-C. Hun, and B. Maggs, "FANFARE for the Common Flow," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-05-148, June 2005.

[33] Q. He, C. Dovrolis, and M. Ammar, "Prediction of TCP Throughput: Formula-based and History-based Methods," in *SIGMETRICS*, June 2005.

[34] A. Kuzmanovic and E. Knightly, "A Performance vs. Trust Perspective in the Design of End-Point Congestion Control Protocols," in *IEEE ICNP*, October 2004.

[35] A. Shamir, "How to Share a Secret," *Communications of the ACM*, vol. 22, no. 11, November 1979, pp. 612–613.

[36] R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell System Technical Journal*, vol. 26, no. 2, April 1950, pp. 147–160.

Manfred Georg is a Ph.D. student at Washington University in St. Louis. He received a dual B.S. in Mathematics and Computer Science from Colorado State University in 2003 and a M.S. in Computer Science from Washington University in St. Louis in 2007. His main area of research is computer vision with emphasis on manifold learning techniques. His work has appeared in various conferences and journals in diverse fields including CVPR, IWQoS, ICNS, Annual Meeting of the American Radium Society, and the Journal of Information and Software Technology. He has had internships at MeVis in Bremen, Germany and AT&T labs in New Jersey. He serves on the Technical Program Committee of ICNS, ICAS, ICABS, and SOAS.

Sergey Gorinsky is a native of Skhodnya, USSR and currently works as an Assistant Professor at the Applied Research Laboratory in the Department of Computer Science and Engineering at Washington University in St. Louis. He received the Ph.D. and M.S. degrees from the University of Texas at Austin, USA and Engineer degree from Moscow Institute of Electronic Technology, Zelenograd, Russia. Prof. Gorinsky's primary research interests are in computer networking and distributed systems. His research contributions include multicast congestion control resilient to receiver misbehavior, analysis of binary adjustment algorithms, and network service differentiation based on performance incentives. Prof. Gorinsky's work appeared at top conferences and journals such as ACM SIGCOMM, IEEE INFOCOM, and IEEE/ACM Transactions on Networking. He has been serving on the Technical Program Committees (TPCs) of IEEE INFOCOM 2006, 2007, 2008, 2009, ICNP 2008, and other networking conferences. He co-chaired the TPCs of High-Speed Networks (HSN 2008) at IEEE INFOCOM 2008 and the Symposium on Future Internet Architectures and Protocols (FIAP 2008) at ICCCN 2008. He is serving as a TPC Vice-Chair of ICCCN 2009 and as a Vice-Chair of the IEEE Communications Society (ComSoc) Technical Committee on High-Speed Networking (TCHSN).