RED with ACK Spoofing

Zhong Xu
Dept. of Electrical Engineering
University of California, Riverside
Riverside, CA 92521
zhong@cs.ucr.edu

Mart Molle
Dept. of Computer Science & Engineering
University of California, Riverside
Riverside, CA 92521
mart@cs.ucr.edu

Abstract

We propose a new congestion signaling method to support Active Queue Management in IP networks called *ACK* (acknowledgement) *Spoofing*. ACK Spoofing enables the router to reduce the transmission rate for any responsive TCP source *without* needlessly reducing goodput (by dropping data packets in advance of buffer overflow), nor relying on TCP implementations to support the packet marking required by Explicit Congestion Notification (ECN). Instead, ACK Spoofing only uses the available fast retransmit and fast recovery mechanisms in current TCP implementations. The basic idea of ACK Spoofing is that a router can remotely trigger the fast retransmit and fast recovery mechanisms in current TCP implementations to reduce their respective transmission rates by sending *multiple locally-created duplicate ACK packets* to the TCP source. The only real difficulty is determining the correct state information (i.e., ack number and advertised window) to include in the spoofed ACKs, for which we present our solution.

ACK Spoofing can be combined with any AQM algorithms, including Random Early Detection (RED). Using simulation, we compare RED/Spoofing with Tail Drop, RED, and RED/ECN. Our results show that RED/Spoofing performs as well as RED/ECN, even though it avoids ECN's deployment problem. In addition, RED/Spoofing out-performs RED, in terms of packet loss rate and fairness, and significantly reduces the queueing delays in comparison to Tail Drop.

1 Introduction

Currently, most routers in the Internet use Tail Drop as their queue management algorithm. Tail Drop has served the Internet well for many years, but it has two major drawbacks [1]: (i) *Lock-out problem.* In some cases, a few flows may unfairly occupy a very large portion of bandwidth. This is caused by synchronization or other timing effects. (ii) *Full-queue problem.* Tail Drop provides no indication of congestion to the TCP sources before the queue overflows, so large queue sizes may persist for long periods.

In response to these problems, a more proactive approach to buffer management called Active Queue Management (AQM) has been proposed. In this case, the router signals congestion by selectively dropping a few packets *before* the queue overflows, which causes the TCP senders to reduce their sending rates upon detection of the packet losses. AQM algorithms could solve the lock-out problem due to their random nature in dropping

packets, and reduce the queueing delay by forcing TCP senders to decrease their sending rates before congestion occurs.

RED [2] is one well-studied AQM algorithm. RED modulates its packet drop probability in proportion to its current estimate of the queue size, relative to a target range. Using RED can significantly reduce the average queue size and queueing delay at a router. However, in some scenarios, RED's improvement on fairness and packet loss rate is not significant[1][3][4]. There are some modified RED algorithms to address these problems, such as Self-configuring RED[5], SRED[6] and FRED[3].

Recently, ECN has been proposed as an alternative to packet dropping for signaling congestion [7]. With the ECN mechanism, the AQM algorithm will simply mark some ECN control bits (formerly reserved flag bits in the headers) instead of dropping the "victim" packet. Thereafter, the TCP receiver must look for ECN markings in the incoming packets and, if any are found, forward them to the TCP sender through the ACK stream. The performance of ECN was first studied in combination with RED, where it was found to give significant reduction in packet losses, but only a minor increase in the link effective throughput[8]. Several other AQM algorithms have subsequently adopted the ECN mechanism, such as BLUE[9], REM[10], etc.

Unfortunately, ECN faces some serious deployment problems because most current TCP implementations do not support the ECN mechanism. Due to the huge number and variety of Internet nodes, it would be unrealistic to expect them all upgrade to an ECN-capable TCP/IP implementation simultaneously. Thus, ECN capabilities must be deployed incrementally. Unfortunately, incremental deployment of ECN would create severe unfairness problems during the transition period. If ECN-compatible flows are mixed with ECN-incompatible flows, the ECN-compatible flows will be punished in bandwidth allocation, because ECN-incompatible flows will ignore the congestion information contained in ECN bits and occupy unfair portion of bandwidth. As a consequence, a question arises:

Can we achieve the benefits of ECN congestion signaling without the need to modify current implementations of TCP?

In this paper, we propose a new congestion signaling method, which is only based on currently available mechanisms of TCP, and can be combined with RED algorithm. Nowadays, most of TCP implementations are based on TCP Reno release. TCP Reno has fast retransmission and fast recovery mechanisms incorporated. With these two mechanisms, the TCP sender will shrink its congestion window size, not only when there is a packet loss but also when the sender receives multiple duplicate *acknowledgement packets* (ACKs). The basic idea of our method is that the router artificially generates multiple duplicate ACKs to trigger the sender to reduce its sending rate, when there is an imminent congestion. We call this congestion signaling mechanism *ACK Spoofing*.

The rest of this paper is organized as follows. In section 2, we discuss ACK Spoofing, enhancement mechanism and RED/Spoofing algorithm in detail. In section 3, we present and analyze the simulation results, and compare the performance of RED with ACK Spoofing (RED/Spoofing), RED with ECN (RED/ECN), RED (with Packet Dropping) and Tail Drop algorithms. Then, we briefly discuss some implementation issues of RED/Spoofing in section 4. Finally, we conclude this paper in section 5.

2 RED with ACK Spoofing

2.1 Principles of ACK Spoofing

Most current TCP implementations are based on the TCP Reno release, which incorporates the fast retransmit and fast recovery mechanisms. These mechanisms cause the TCP sender to reduce its congestion window size by half after receiving multiple duplicate ACKs. This duplicate-ACK response property forms the basis of *ACK Spoofing*: the router tells the TCP source to reduce its transmission rate by sending it multiple artificially generated duplicate ACK packets, called *spoofing ACKs*.

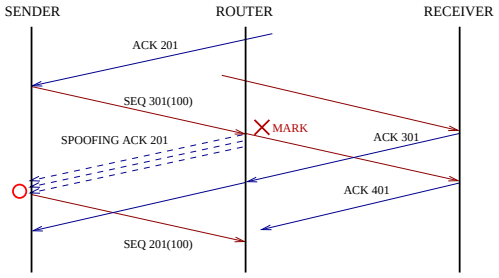


Figure 1: Basic ideas of ACK Spoofing

Fig. 1 illustrates the basic procedure of ACK Spoofing. If RED decides to mark a particular packet [1] (SEQ 301, say), it generates multiple spoofing ACKs and sends them to source of the associated TCP flow. Note that the ack number carried by the spoofing ACKs must be the highest ack number [2] yet seen by the router for this flow (i.e., ACK 201), rather than the sequence number of the marked packet (SEQ 301). Smaller ack numbers would cause the TCP sender to ignore the spoofing ACKs (delayed, out-of-order), while larger ack numbers would compromise the reliability of the TCP session.

Upon receiving the spoofing ACKs, the TCP source will be tricked into immediately reducing its sending rate and retransmitting the "missing" packet (SEQ 201). However, since no packet was actually dropped, the retransmission is just a needless duplicate that can be discarded at the router if we like.

2.2 Partial Per-Flow Maintenance

To generate spoofing ACKs, the router needs to include some state variables obtained from real ack packets travelling over the reverse path, i.e., ack number (*th_ack*) and advertised window size (*th_win*). The most obvious way to gather these values would be for the router to maintain per-flow state information about every active flow all the time — which would cause considerable overhead and possibly reduce its throughput. We will consider these implementation issues in more detail in section 4. At this point, however, we will use Fig.2 to show that ACK Spoofing only requires the router to track the state variables for a given flow for a short time period after one of its packets is targetted for marking/dropping. We call this requirement *partial per-flow maintenance*.

[1] Packet marking doesn't necessarily mean setting ECN bits. Within this paper, it refers to any congestion signaling action without dropping the packet.

[2] Using modulo arithmetic to handle wrap around of ack and sequence numbers, of course.

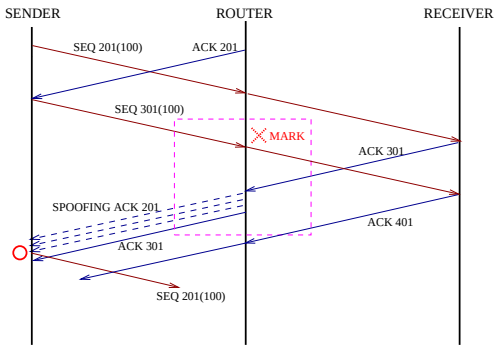


Figure 2: Procedure of ACK Spoofing

Suppose RED decides to mark packet 301 in fig.2, but the router has *not* kept track of the most recent ack number and advertised window values for the associated flow. In this case, the router "sets" the spoofing state for this flow, triggering a search for the next ACK on the reverse path. As soon as it finds a suitable ack packet, the router extracts the necessary information to create the spoofing ACKs, and "clears" the spoofing state for this flow.

Since the router generates the spoofing ACKs before forwarding the "real" ACK packet that just arrived on the reverse path, we can perform the following optimization. By sending the spoofing ACKs before the "real" ACK and setting the spoofed ack number to one MSS below the real ack number, we might be able to trick the TCP source into reducing its congestion window (because of the spoofing ACKs) but without sending a wasted duplicate packet (because of the real ACK).

2.3 RED/Spoofing Algorithm

Original RED uses packet dropping to signal congestion. It drops incoming packets with a dynamic dropping probability, which generally increases with the growth of queue size estimation. For algorithm details, please refer to original publication [2]. To combine RED with ACK Spoofing, we only need to make two modifications: (1) If the packet dropping is caused by full queue, add an ACK Spoofing in addition to packet dropping; (2) Otherwise, change packet dropping to ACK Spoofing.

2.4 Dealing with Asymmetric Paths

With ACK Spoofing, the router must acquire the *th_ack* and *th_win* values in real packets from TCP receivers. Otherwise, the router has no way to generate spoofing ACKs. Unfortunately, the path for a flow in two directions may differ. V. Paxson [11] showed that there were 30% – 50% of paths visiting at least one different hop in two directions in 1994 and 1995. Among these asymmetric paths, majorities (2/3) differ at only one hop and others (1/3) at two or three hops. Besides, routing paths may change during the lifetimes of flows.

To handle this situation, we can treat the flows with asymmetric paths differently. The main idea behind the mechanism is that, for one flow, the router will not send spoofing ACKs but drop the packets on congestion, if the router detects an asymmetric path for this flow. One method to find an asymmetric path is for the router to count the number of spoofing events it generates for the flow or the number of data packets

of the flow it receives since the spoofing status has been set. Once the counted number exceeds a given threshold, the router will treat the flow as a flow with asymmetric path at this hop, and use packet dropping instead of ACK Spoofing to signal congestion. On the other hand, the status of asymmetric path will be cleared once the router receives a real packet from the TCP receiver.

3 Performance Evaluation

In this section, we will present the simulation results and compare the performance of RED/Spoofing with its counterparts. Performance metrics used in this section include: (1) Goodput, which represents effective link throughput acknowledged by receivers and doesn't include throughput wasted by dropped packets and useless retransmits; (2) Average queue size, which stands for average queueing delay; (3) Packet loss rate, which is presented in packets/second; and (4) Fairness and stability, which evaluate whether algorithms allocate bandwidth fairly and how severe their bandwidth sharing fluctuates. Our simulator is written in C/C++ with CSIM18[12], and includes a faithful representation of TCP Reno, which mimics the model implementation by Stevens[13].

3.1 Performance of RED/Spoofing

In this part, we compare the performance of RED/Spoofing with Tail Drop, RED and RED/ECN in terms of goodput, packet loss rate and queueing delay. The simulation network is illustrated in Fig.3, where all link speeds are 100Mbps and values shown on the links are propogation times in milliseconds. Simulation parameters are given in Table 1. In our experiments, we set up four groups of TCP flows, and each group has 10 individual TCP flows. Thus, we have simulated a total of 40 TCP flows — they are 10 flows from H0 to H5, 10 flows from H1 to H4, 10 flows from H2 to H4, and 10 flows from H3 to H5 respectively. Here, the bottleneck link is the link from R2 to R3. Hence, we focus on the simulation results for router R2 and the link from R2 to R3.

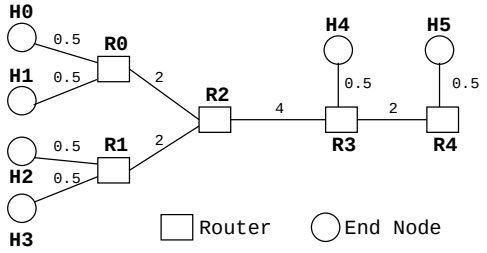
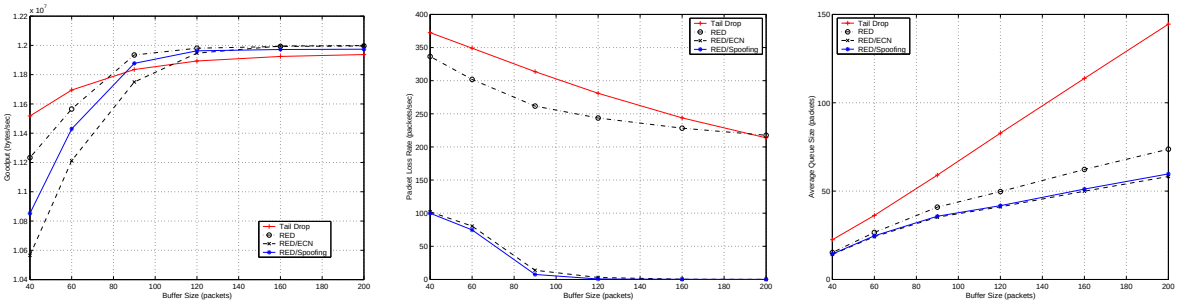


Figure 3: The simulation network 1

Table 1: Simulation parameters

Buffer Size	**40**	**60**	**90**	**120**	**160**	**200**
min_th	5	10	10	10	10	10
max_th	30	50	72	90	120	150
AQM	**RED**		**RED/ECN**		**RED/Spoofing**	
max_p	0.02		0.10		0.05	

Simulation results of various queue management algorithms with different queue buffer sizes are illustrated in Fig.4. Each point in the graphs represents the average of 30 experiment results.



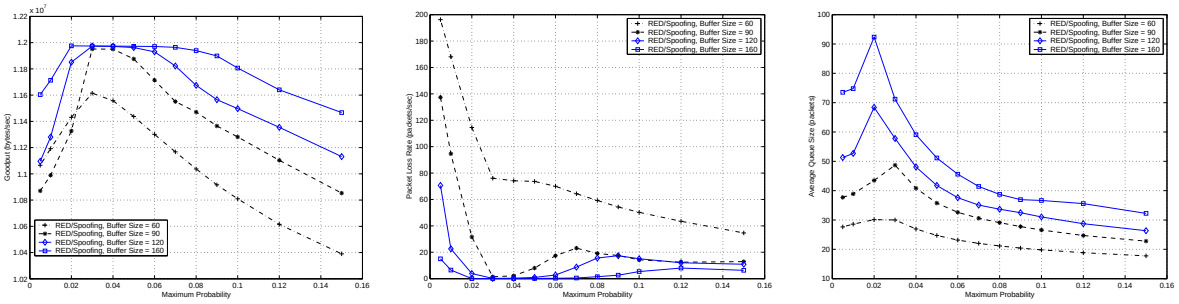
(a) Goodput (b) Packet Loss Rate (c) Average Queue Size

Figure 4: Performance comparison of RED/Spoofing, RED/ECN, RED and Tail Drop

Although Tail Drop could yield high throughputs even with very small buffer sizes, it has worst performance in terms of packet loss and average queueing delay. Compared to Tail Drop, RED gains around 25% – 50% improvement on average queue size and a small improvement on packet loss rate, while it maintains comparable goodputs. On the other hand, in comparison to RED, both RED/Spoofing and RED/ECN significantly reduce the packet loss rates, while maintaining slightly better average queue sizes and comparable goodputs. Although RED/Spoofing has similar performance to RED/ECN, we think that RED/Spoofing is more promising because it is compatible with existing TCP implementations.

3.2 Parameter Sensitivity

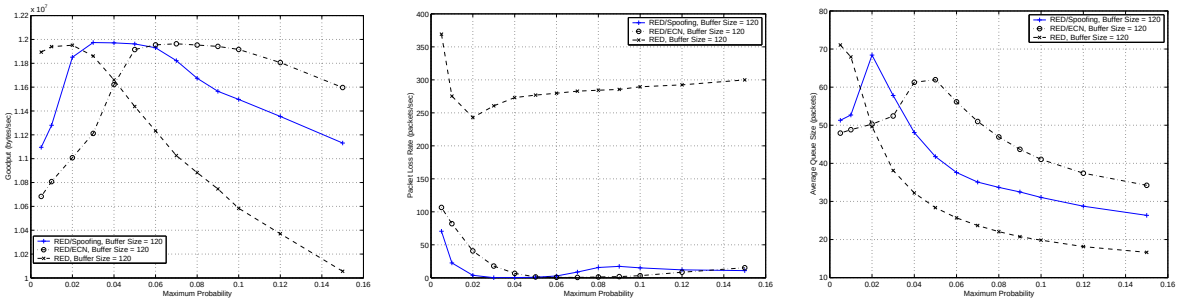
In this part, we study impacts of parameters on the performance of RED/Spoofing. Among four RED parameters, max_p(maximum probability) plays an important role, and therefore we focus on the impacts of max_p. We use same network and thresholds as in last subsection. Simulation results are given in Fig.5. Also, we compare the parameter sensitivities of RED/ECN and RED/Spoofing, and simulation results are given in Fig.6.



(a) Goodput (b) Packet Loss Rate (c) Average Queue Size

Figure 5: Performance of RED/Spoofing with different maximum probabilities

For RED/Spoofing, the performance varies with different maximum probabilities. There exists an optimal area(for example, 0.03-0.05 for our experiments), within which



(a) Goodput (b) Packet Loss Rate (c) Average Queue Size

Figure 6: Performance comparison of RED/Spoofing and RED/ECN with different maximum probabilities

the algorithm has best overall performance. Furthermore, there are trade-offs between three performance metrics. For example, changing max_p to improve goodput may probably increase the average queue size but decrease the packet loss rate. How to choose best parameters needs further investigation.

Similar conclusion can be drawn for RED/ECN and RED. Therefore, the high parameter sensitivity of RED/Spoofing should not originate from ACK Spoofing but from RED algorithm itself.

3.3 Fairness and Stability

In this experiment, we use the simulation network 2(see Fig. 7), where link R0-R1 is the bottleneck. We set up two groups of TCP flows. First group contains 10 TCP flows from H0 to H2 and these TCP flows start at time 0. The other group contains 10 TCP flows from H1 to H3, but these flows start at time 2000ms. TCP flows in the second group will compete for bandwith with flows in the first group. After they reach fair sharing of bandwith, bandwidth of flows may oscillate.

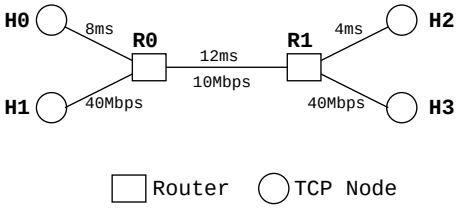
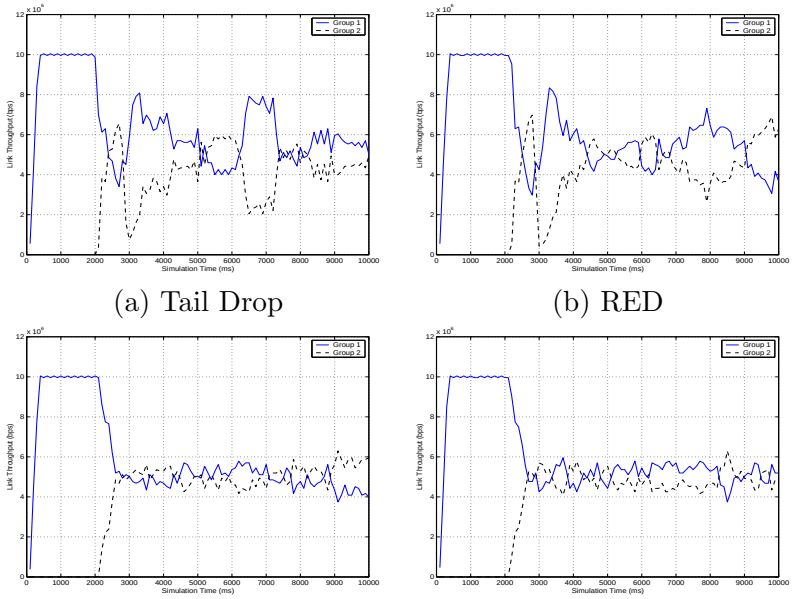


Figure 7: The simulation network 2

Fig.8 illustrats the dynamics of bandwidth sharing among two groups with different algorithms on link R0-R1. From the graphs, we can find that all four algorithms basically converge to the fair bandwidth sharing. Among these algorithms, convergence speeds of Tail Drop and RED are a little bit faster. However, Tail Drop's stabilities are the worst one. Its bandwidth sharing fluctuates severely. During the interval (6500ms, 7200ms), its aggregate bandwidth of group 1 is roughly 3.5 times of that of group 2. Although RED is a little bit better than Tail Drop in this aspect, its oscillation of bandwidth is still very large. On the other hand, RED/ECN and RED/Spoofing are much more stable than



(a) Tail Drop (b) RED

(c) RED/ECN (d) RED/Spoofing

Figure 8: Convergence and stability of queue management algorithms

Tail Drop and RED. In average, the average difference between aggregate bandwidths of two groups is about 10% of the bandwidth total.

The possible reason for this is that their signaling latencies are different. Tail Drop and RED use packet dropping as their congestion signaling method, while others use ECN or ACK Spoofing. With packet dropping, TCP senders may need to wait for timeouts for detecting the packet losses. But with ECN or ACK Spoofing, the latency from packet marking to sender's reaction should be less than one round-trip time. Also, with packet dropping, the sender may decrease its rate too much due to slow start. Therefore, faster signaling and moderate rate adjustment make RED/Spoofing and RED/ECN better in terms of fairness and stability.

4 Implementation Issues

In high-speed networks, routers must handle traffic at gigabit-per-second or even higher rates. This requires routers be able to forward millions of packets per second. The agorithm RED/Spoofing uses most recent connection information for generating spoofing ACKs, and requires routers to maintain partial flow information when necessary. Such a maintenance and corresponding matching/updating operations should not impose much computing load on routers. Otherwise, router's processing throughput may be hurt.

Hashing tables and caches are popular techniques used in IP routing table lookup[14]. We can apply these techniques in software to perform ACK Spoofing state maintenance too. Also, the state maintenance is easier than routing table lookup in that the searching key of flow information is of fixed length. A TCP flow can be uniquely determined by following information: sender's IP address & port number, and receiver's IP address & port number. When we determine the searching key, one requirement is to make records randomly scattered. Due to the hierarchical nature of Internet's IP addresses, high bits of IP addresses are not suitable for the searching key. On the other hand, port number is

not a good choice for searching key, because most TCP communications use only a few popular ports(like ports reserved for http and ftp). So, we suggest to use low 8-16 bits of two IP addresses as the searching key.

An alternative hardware-based approach is considered in [15]. In this case, we recognize the similarity between wire-speed address filtering of 48-bit MAC addresses in LAN bridges (aka "layer 2 switches") and the IP-layer reverse ACK matching problem based on 96-bit "keys" (representing the 4-tuple of two 32-bit IP address and two 16-bit port numbers), and show that the same content addressible memory (CAM) modules used by LAN bridges could be used to pick out the necessary ACKs from the reverse data stream travelling through the given port to support ACK spoofing.

5 Conclusions

With the rapid growth of Internet traffic, network congestion is becoming a significant problem. Deploying AQM algorithms in combination with packet dropping would be very easy to accomplish, while offering a significant performance improvement by combining high throughput with low queueing delays. However, current algorithms exhibit serious fairness and stability problems. On the other hand, deploying AQM in combination with ECN would solve all of the performance problems associated with the packet dropping signaling, but is impractical because it requires modifications to the installed base of TCP implementations. Thus, AQM in combination with ACK Spoofing represents an important advance in practical TCP congestion control methods because it offers both the performance of ECN and backwards compatibility with existing TCP implementations. Indeed, using a detailed simulation model to compare RED with ACK Spoofing against Tail Drop, RED and RED/ECN, we show that RED/Spoofing significantly out-performs Tail Drop and RED in terms of the packet loss rate, queueing delay or fairness, and offers similar performance to RED/ECN while completely avoiding ECN's compatibility problem.

References

[1] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jaconson, G. Minshall, C. Patridge, L. Peterson, K. Ramakrishnan, S.Shenker, J. Wroclwaski, and L. Zhang, "Recommendations on queue management and congestion avoidance in the internet," *RFC 2309*, April 1998, http://www.ietf.org/rfc/rfc2309.

[2] Sally Floyd and Van Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, August 1993.

[3] Dong Lin and Robert Morris, "Dynamics of random early detection," *Proceedings of ACM SIGCOMM'97*, October 1997.

[4] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons not to deploy red," *Proceedings of IWQoS'99*, March 1999.

[5] Wu-Chang Feng, Dilip D. Kandlur, Debanjan Saha, and Kang G. Shin, "A self-configuring red gateway," *Proceedings of INFOCOM'99, New York*, March 1999.

[6] Teunis J. Ott, T.V. Lakshman, and Larry Wong, "Sred: Stablized red," *Proceedings of IEEE INFOCOM'99, New York*, March 1999.

[7] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ecn) to ip," *RFC 3168*, September 2001.

[8] Sally Floyd, "Tcp and explicit congestion notification," *ACM Computer Communication Review*, vol. 24, no. 5, October 1994.

[9] Wu-Chang Feng, Dilip D. Kandlur, Debanjan Saha, and Kang G. Shin, "Blue: A new class of active queue management algorithms," *Technical Report, CSE-TR-387-99, University of Michigan*, April 1999.

[10] Sanjeewa Athuraliya and Steven H. Low, "Optimization flow control, ii: Random exponential marking," *Preprint, http://www.ee.mu.oz.au/staf/slow/research*, May 2000.

[11] Vern Paxson, "End-to-end routing behavior in the internet," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, October 1997.

[12] Mesquite Software, *CSIM18 Documentation: User Guides*, http://www.mesquite.com/htmls/guides.htm.

[13] Gary R. Wright and W. Richard Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*, Addison-Wesley, 1995.

[14] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner, "Scalable high speed ip routing lookups," *Proceedings of ACM SIGCOMM'97, Nice, France*, September 1997.

[15] Mart Molle and Zhong Xu, "Short-circuiting the congestion signaling path for aqm algorithms using reverse flow matching," *Technical Report, Dept. of Computer Science, Univ. of California, Riverside*, 2003.