Quantifying the Effects of Recent Protocol Improvements to Standards-Track TCP[*]

Michele C. Weigle, Kevin Jeffay, and F. Donelson Smith
Department of Computer Science, University of North Carolina at Chapel Hill
{mcweigle,jeffay,smithfd}@cs.unc.edu

Abstract

We assess the state-of-the-art in Internet congestion control and error recovery through a controlled study that considers the integration of standards-track TCP error recovery and both TCP and router-based congestion control. The goal is to examine and quantify the benefits of deploying standards-track technologies for Internet traffic as a function of the level of offered network load. We limit our study to the dominant and most stressful class of Internet traffic: bursty HTTP flows. Contrary to expectations and published prior work, we find that for HTTP flows (1) there is no clear benefit in using TCP SACK over TCP Reno, (2) unless congestion is a serious concern (i.e., unless average link utilization is above approximately 80%), there is little benefit to using Adaptive RED queue management, (3) above 80% link utilization there is potential benefit to using Adaptive RED with ECN marking, however, complex performance trade-offs exist and results are sensitive to parameter settings.

1. Introduction

Improvements to TCP's error recovery and congestion control/avoidance mechanisms have long been a mainstay of networking research. In this paper, we evaluate the performance of combinations of "standards-track" TCP error recovery and congestion control techniques. We consider standards-track TCP error recovery mechanisms to include TCP Reno fast retransmission [1] and TCP with selective acknowledgments (SACK) [2]. We consider standards-track TCP congestion control mechanisms to include congestion avoidance and fast recovery in TCP Reno and the router-based congestion control found in routers that support Adaptive Random Early Detection (ARED) [3] and Explicit Congestion Notification (ECN) [4].

Using *ns* simulations, we evaluated how well various pairings of Reno, SACK, drop-tail, and ARED (with both packet dropping and ECN marking) perform in the context of HTTP traffic. Our results provide an evaluation of the state-of-the-art in TCP error recovery and congestion control in the context of Internet traffic composed of "mice" and "elephants." We used bursty HTTP traffic sources generating a traffic mix with a majority of flows sending few segments (< 5), a small number sending many segments (> 50), and a number in the [5, 50] segment range. Our primary metric of performance was the response time to deliver each HTTP object requested along with secondary metrics of link utilization, router queue size, and packet loss percentage.

Section 2 explains our experimental methodology and Section 3 presents a summary of our main results. Complete experimental results, as well as a more detailed description of the experimental design, are presented in an available technical report [5].

2. Methodology

2.1 Experimental Setup

We ran simulations in *ns* with varying levels of two-way HTTP 1.0 traffic. These two-way traffic loads provide roughly equal levels of congestion on both the "forward" path (server-to-client) and "reverse" path (client-to-server). The following pairings of error recovery and queue management techniques were tested: Reno with drop-tail queuing in routers, Reno with ARED using packet drops, ECN-enabled Reno with ARED using ECN marking, SACK with drop-tail, SACK with ARED using packet drops, and ECN-enabled SACK with ARED using ECN marking. (SACK includes Reno congestion control.)

The HTTP traffic we generate comes from the PackMime model developed at Bell Labs [6]. The fundamental parameter of PackMime is the TCP/HTTP connection initiation rate (a parameter of the distribution of connection interarrival times). The model also includes distributions of the size of HTTP requests and the size of HTTP responses. Both the HTTP request size and response size distributions are heavy-tailed. There are a large number of small requests and a few very large requests. Almost 90% of the requests are under 1 KB and fit in a single packet. The largest request is almost 1 MB. 60% of the responses fit into one packet and 90% of the responses fit into 10 packets, yet the largest response size is over 100 MB. Using this distribution, we will have many short-lived transfers, but also some very long-lived flows.

In each experiment, we examine the behavior of traffic that consists of 250,000 HTTP connections, with a total simulated time of 40 minutes. In our implementation of PackMime in *ns*, one PackMime "node" represents a cloud of HTTP clients or servers. The traffic load is driven by the user-supplied connection rate parameter, which is the number of new connections starting per second. New connections begin at their appointed time, whether or not any previous connection has completed.

The network we simulate consists of two clouds of web servers and clients positioned at each end of a 10 Mbps bottleneck link (Figure 1). There is a 10 Mbps bottleneck link between the two routers, a 20 Mbps link between each PackMime cloud and its corresponding aggregation node, and a 100 Mbps link between each aggregation node and the nearest router. This configuration is designed to ensure that all congestion in the network occurs on the bottleneck link between the two routers.

The aggregation nodes in our simulations are *ns* nodes that we developed called DelayBoxes to delay packets in

[*] This work supported in parts by grants from the National Science Foundation (grants ITR-0082870, CCR-0208924, EIA-0303590, and ANI-0323648), Cisco Systems Inc., and the IBM Corporation.

In: Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of
Computer and Telecommunication Systems (MASCOTS), Orlando, FL, October 2003, pages 226-229

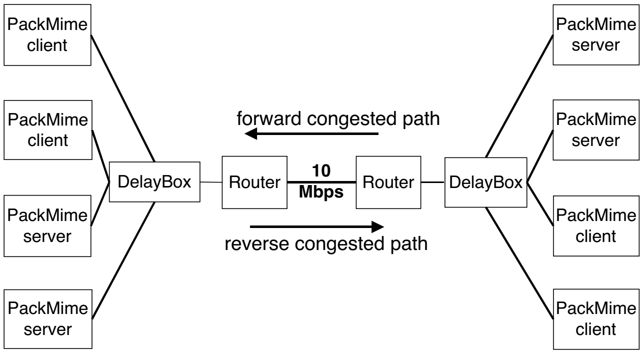

Figure 1: Simulated network environment

the simulation.[1] With DelayBox, packets from a TCP connection can be delayed before being passed on to the next node. This allows each TCP connection to experience a different minimum delay (and hence a different round-trip time), based on random sampling from a delay distribution. In our experiments, DelayBox uses an empirical delay distribution from the PackMime model. This results in minmum delays ranging from 1 ms to 3.5 seconds. The median delay is 54 ms, the mean is 74 ms, and the 90th percentile is 117 ms. Delays are assigned independently of request or response size and represent only propagation delay and do not include queuing delays.

For a target bottleneck bandwidth of 10 Mbps, we compute the bandwidth-delay product (BDP) to be 74 1,250-byte packets. In all cases, we set the maximum send window for each TCP connection to the BDP.

2.2 Queue Management Settings

Christiansen *et al.* recommend a maximum queue size between 1.25-2×BDP for reasonable response times for drop-tail queues [7]. The maximum queue buffer sizes tested in our drop-tail experiments were 1.5 and 2×BDP.

We ran sets of ARED experiments using the default ARED settings in *ns* (target delay = 5 ms) and with parameters similar to those suggested by Christiansen, min_{th} = 30 and max_{th} = 90, resulting in a target delay of 60 ms. The maximum router queue length was set to 5×BDP to ensure that there would be no tail drops, in order to isolate the effects of ARED.

2.3 Performance Metrics

In each experiment, we measured the HTTP response times (time from sending HTTP request to receiving entire HTTP response), link utilization, average loss rate, and average queue size. HTTP response time is our main metric of performance. We report the CDFs of response times for responses that complete in 1,500 ms or less. When discussing the CDFs, we discuss the percentage of flows that complete in a given amount of time.

2.4 Levels of Offered Load and Data Collection

The levels of offered load used in our experiments are expressed as a percentage of the capacity of a 10 Mbps link. We initially ran our network at 100 Mbps to determined the PackMime connection rates (essentially the HTTP request rates) that will result in average link utilizations (in both forward and reverse directions) of 5, 6, 7, 8, 8.5, 9, 9.5, 10, and 10.5 Mbps. The connection rate that resulted in an average utilization of 8% of the (clearly uncongested) 100 Mbps link was used to generate an offered load on the 10 Mbps link of 8 Mbps or 80% of 10 Mbps. Note that this 80% load (*i.e.*, the connection rate that results in 8 Mbps of traffic on the 100 Mbps link) will not actually result in 8 Mbps of traffic on the 10 Mbps link. The bursty HTTP sources will cause congestion on the 10 Mbps link and the actual utilization of the link will be a function of the protocol and router queue management scheme used.

Table 1: Summary of labels and abbreviations

Abbreviation	Description
DT-111q	Drop-Tail with 111-packet queue (1.5 x BDP)
DT-148q	Drop-Tail with 148-packet queue (2 x BDP)
ARED-5ms	Adaptive RED with 5 ms target delay
ARED-60ms	Adaptive RED with 60 ms target delay
ARED+ECN-5ms	Adaptive RED with ECN & 5 ms target delay
ARED+ECN-60ms	Adaptive RED with ECN & 60 ms target delay

3. Results

We first present the results for different queue management algorithms when paired with TCP Reno end-systems. Next, results for queue management algorithms paired with TCP SACK end-systems are presented and compared to the Reno results. Finally, the two best scenarios are compared. In the following response-time CDF plots, the response times obtained in a calibration experiment with the uncongested 100 Mbps link are included for a baseline reference as this represents the best possible performance. Table 1 lists the labels we use to identify experiments.

Figures 2-4 give summary network-centric performance measures.[2] For each experiment, Figure 2 shows the average link utilization, Figure 3 shows the average packet loss, and Figure 4 shows the average queue size at the bottleneck router.

3.1 Reno + DropTail

Figure 5 shows the CDFs of response times for Reno-DT-111q and Reno-DT-148q at 80% and 105% offered loads. There is little performance difference between the two queue sizes, though there is a crossover point in the response times. For simplicity, when comparing drop-tail to other queuing methods, we show only DT-148q results.

3.2 Reno + Adaptive RED

Response time CDFs for Reno-DT-148q, Reno-ARED-5ms and Reno-ARED-60ms are shown in Figures 6-7. At 80% load (Figure 6) the drop-tail queue performs no worse than ARED-60ms. There is a distinct crossover point between Reno-ARED-5ms and Reno-ARED-60ms (and Reno-DT-148q) at 400 ms. This points to a tradeoff between improving response times for some flows and causing worse response times for others. ARED-5ms has a larger percentage of responses completing in less than 400 ms, while there are a larger percentage of responses completing in less than 1,500 ms with ARED-60ms or Reno-DT-148q. As the load increases (Figure 7), the crossover

[1] Our implementations of PackMime and DelayBox are available at *http://www.isi.edu/nsnam/ns/ns-contributed.html.*

[2] For more legible plots, see [5].

remains near a response time of 400 ms, but the percentage of completed responses in that time decreases. Also, as load increases, the performance of ARED-5ms for longer responses is poor.

ARED-5ms keeps a much shorter average queue than ARED-60ms (Figure 4), but at the expense of longer response times for many responses. With ARED-5ms, there are also large numbers of packet drops (Figure 3). Many of the HTTP connections are very short-lived, often consisting of a single packet. When these flows experience packet loss, they are forced to suffer a retransmission timeout, increasing their HTTP response times. For ARED-5ms, the CDF of response times levels off after the crossover point and does not increase much until after 1 second, which is the minimum RTO in our simulations. This indicates that a significant portion of the flows suffered timeouts.

As congestion increases toward severe levels, the response time benefits from the drop-tail queue become substantially greater. Figure 7 shows that about 60% of responses (those taking longer than 300-400 ms to complete) have better response times with the drop-tail queue.

3.3 Reno + Adaptive RED + ECN

The full value of ARED is realized only when it is coupled with a more effective means of indicating congestion to the TCP endpoints than the implicit signal of a packet drop. ECN is the congestion signaling mechanism intended to be paired with ARED. Figures 8-9 present the response time CDFs for Reno-ARED and Reno-ARED+ECN with 5 and 60ms target delays. At 80% load, ARED+ECN-5ms delivers superior or equivalent performance for all response times. Further, ECN has a more significant benefit when the ARED target delay is small. As before, there is a tradeoff where the 5 ms target delay setting performs better before the crossover point and the 60 ms setting performs slightly better after the crossover point. The tradeoff when ECN is paired with ARED is much less significant. ARED+ECN-5ms does not see as much drop-off in performance after the crossover point. For the severely congested case, ECN provides even more advantages, especially when coupled with a small target delay.

3.4 SACK

The experiments described above were repeated by pairing SACK (which includes Reno congestion control) with the different queue management mechanisms. Figures 10-11 show a comparison between Reno and SACK error recovery based on response-time CDFs when paired with both drop-tail and ARED+ECN. Overall, SACK provides no better performance than Reno. When paired with ARED+ECN, SACK and Reno are essentially identical independent of load. When paired with drop-tail, Reno appears to provide somewhat superior response times especially when congestion is severe.

3.5 Drop-Tail vs. ARED+ECN

Here, we compare the performance of the two "best" error recovery and queue management combinations. Figures 12-13 present the response time CDFs for Reno-DT-148q and Reno-ARED+ECN-5ms. With these scenarios, the fundamental tradeoff between improving response times for some responses and making them worse for other responses is clear. Further, the extent of the tradeoff is quite dependent on the level of congestion. At 80% load, Reno-ARED+ECN-5ms offers better response-time performance for nearly 75% of responses but marginally worse for the rest. At levels of severe congestion, the improvements in response times for Reno-ARED+ECN-5ms apply to around 50% of responses, while the response times of the other 50% are degraded significantly. Reno-DT-148q and Reno-ARED+ECN-5ms are on the opposite ends of the queue-management spectrum, yet they each offer better HTTP response times for different portions of the total set of responses. Further complicating the tradeoff is the result that Reno-DT-148q gives higher link utilization (Figure 2) along with a high average queue size (Figure 4), while Reno-ARED+ECN-5ms gives low average queue sizes, but also lower link utilization.

4. Conclusions

Our main conclusions based on HTTP traffic and our performance metrics are:

- There is no clear benefit in using SACK over Reno, especially when the complexity of implementing SACK is considered. This result holds independent of load and pairing with queue management algorithm.
- As expected, ARED with ECN marking performs better than ARED with packet dropping, and the value of ECN marking increases as the offered load increases. ECN also offers more significant gains in performance when the target delay is small (5 ms).
- Below 80% offered load, SACK and Reno have identical performance, and ARED+ECN performs only marginally better than drop-tail.
- Unless congestion is a serious concern (*i.e.*, for average link utilizations of 80% or higher with bursty sources), there is little benefit to using ARED queue management in routers.
- ARED with ECN marking and a small target delay (5 ms) performs better than drop-tail with two times the bandwidth-delay product (BDP) queue size at offered loads having moderate levels of congestion (80% load). This finding should be tempered with the caution that, like RED, ARED is also sensitive to parameter settings.
- At loads that cause severe congestion, there is a complex performance trade-off between drop-tail with 2×BDP queue size and ARED with ECN at a small target delay. ARED can improve the response time of about half the responses but worsens the other half. Link utilization is significantly better with drop-tail.
- At all loads there is little difference between the performance of drop-tail with 2×BDP queue size and ARED with ECN marking and a larger target delay (60 ms).

In total, we conclude that for user-centric measures of performance, router-based congestion control has a greater impact on performance than protocol improvements for error recovery. However, for lightly to moderately loaded networks, neither queue management nor protocol improvements significantly impact performance.

5. References

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

[2] E. Blanton, M. Allman, K. Fall, and L. Wang, "A Conservative SACK-Based Loss Recovery Algorithm for TCP," IETF Internet Draft, Oct. 2002.

[3] S. Floyd, R. Gummadi, & S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active

Queue Management," *http://www.icir.org/floyd/papers/adaptiveRed.pdf*, 2001.

[4] K.K. Ramakrishnan and S. Floyd, "A Proposal to Add Explicit Congestion Notification to IP," RFC 2481, Experimental, Jan. 1999.

[5] M.C. Weigle. K. Jeffay, F.D. Smith, "Quantifying the Effects of Recent Protocol Improvements on Standards-Track TCP (Extended)," *http://www.cs.unc.edu/Research/dirt*, 2003.

[6] J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun, "PackMime: An Internet Traffic Generator," National Institute of Statistical Sciences Affiliates Workshop on Modeling and Analysis of Network Data, Mar. 2001.

[7] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith, "Tuning RED for Web Traffic," *IEEE Transactions on Networking*, vol. 9, no. 3, pp. 249–264, June 2001.

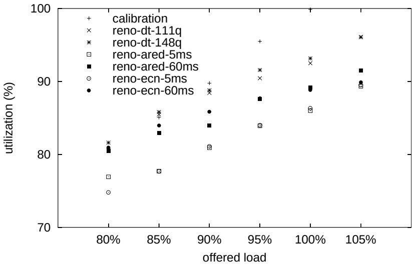

Figure 2: Average link utilization.

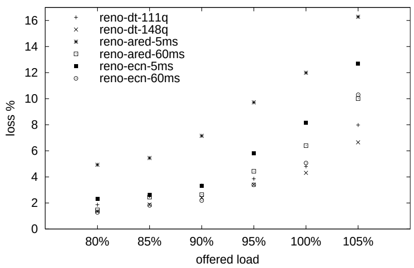

Figure 3: Average loss rate.

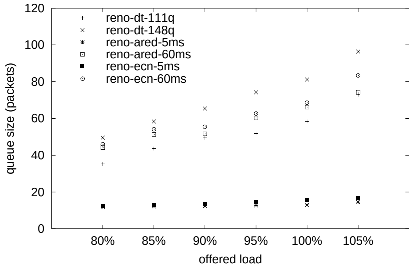

Figure 4: Average queue size.

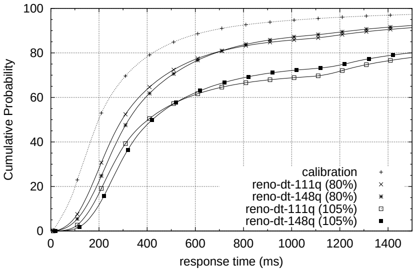

Figure 5: CDF of HTTP response times for Reno with drop-tail queuing.

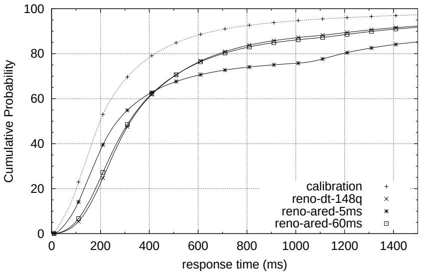

Figure 6: CDF of HTTP response times for drop tail and ARED at 80% offered load.

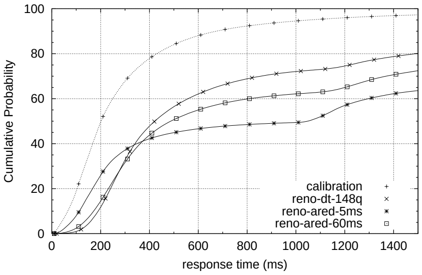

Figure 7: CDF of HTTP response times for drop tail and ARED at 105% offered load.

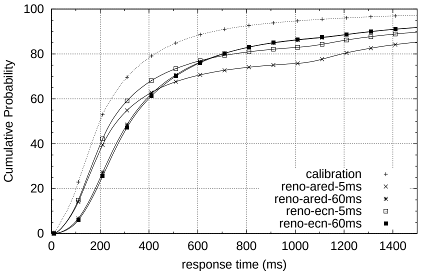

Figure 8: CDF of response times for ARED with and without ECN at 80% offered load.

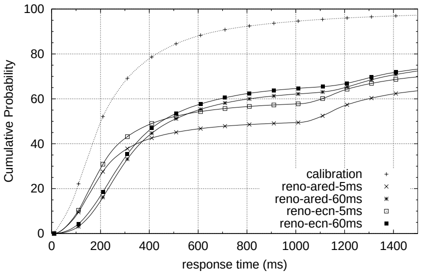

Figure 9: CDF of response times for ARED with and without ECN at 105% offered load.

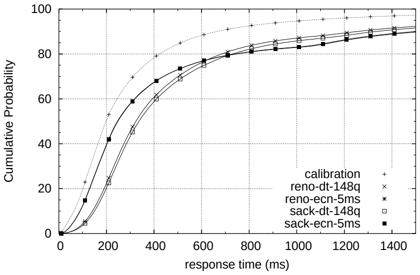

Figure 10: CDF of response times for drop-tail and ARED/ECN at 80% offered load.

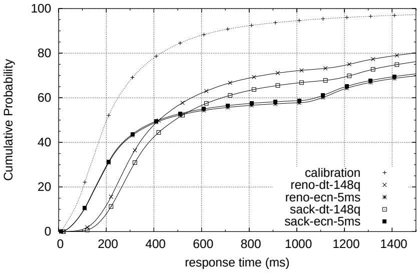

Figure 11: CDF of response times for drop-tail and ARED/ECN at 105% offered load.

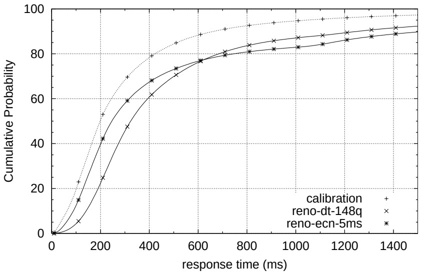

Figure 12: CDF of response times for best drop-tail and best AQM at 80% offered load.

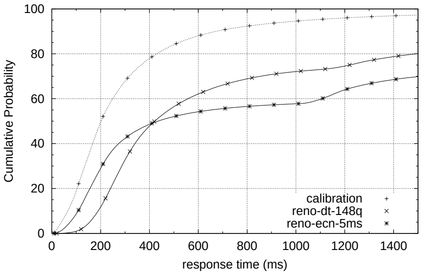

Figure 13: CDF of response times for best drop-tail and best AQM at 105% offered load.