Identifying the TCP Behavior of Web Servers

Jitendra Padhye and Sally Floyd
ACIRI (AT&T Center for Internet Research at ICSI)

Preliminary Version, July 17, 2000

Abstract

This paper describes a tool for characterizing the TCP behavior of a remote host on the Internet. The tool allows the researcher to manipulate TCP communication with a remote TCP. The specific goal in building this tool was to answer the question "What fraction of web servers use NewReno instead of Reno or Tahoe TCP congestion control mechanisms, for TCP connections with non-Sack-enabled clients". The more general goal was to provide a tool for efficiently probing the TCP congestion control behaviors of remote hosts in the Internet. This paper describes both TBIT (the TCP Behavior identification Tool) and our specific results about the TCP congestion control behaviors of major web servers. Both TBIT and the experimental results are available from the TBIT Web Page [TBI]. The tool uses some of the code developed by the Sting project [Sav99].

1 Introduction

There are a range of TCP congestion control behaviors in deployed TCP implementations, include Tahoe [Jac88], Reno [APS99], NewReno [FH99], and Sack TCP [MMFR96], which date from 1988, 1990, 1996, and 1996, respectively. These different variants of TCP congestion control are described and illustrated in detail in [FF96]. The preferred TCP is Sack (Selective Acknowledgement) TCP, but a TCP connection cannot use the SACK option unless both end nodes are SACK-enabled. Of Tahoe, Reno, and NewReno TCP, NewReno has significantly better performance when multiple packets are dropped from a window of data. We recently asked the question "What fraction of the non-SACK TCP flows in the Internet use NewReno instead of Reno congestion control mechanisms (and thereby avoid unnecessary retransmit timeouts when multiple packets are dropped from a window of data)?" [Flo00]. This paper presents a tool for answering this and related questions about TCP behavior, and gives results from experiments to answer this question with regard to the TCP behavior of some of the major web sites in the Internet. One finding of our experiment is that 62% of the 136 web servers tested used NewReno TCP congestion control mechanisms, instead of those of Tahoe or Reno, and 41% advertised the 'SACK-permitted' option.

One reason for asking the question "Who uses NewReno TCP?" is to better understand the migration of new congestion control mechanisms to the public Internet. A second reason to ask the question is to discourage network researchers from extensive investigations of the negative impacts of Reno TCP's poor performance when multiple packets are dropped from a window of data, if in fact Reno TCP is already being replaced by NewReno TCP in the Internet. A third reason to investigate the TCP congestion control mechanisms actually deployed in the public Internet is that it is always useful to occasionally step back from our models, analysis, and simulations, and look at the Internet itself.

It is possible to determine from passive traces whether a remote TCP supports the TCP Sack option, simply by observing whether TCP SYN packet includes the "SACK-permitted" option. This has been used by Mark Allman in [All00] to show that the fraction of web clients advertising SACK capability at a particular web server has risen from 8% in December 1998 to 40% in February 2000. This has also been used by Anja Feldman to show that at a dialup modem pool in AT&T WorldNet, 58% of the TCP connections advertise "SACK permitted" on the SYN packet, and 5% of the TCP connections advertise "SACK permitted" on the returning SYN ACK (acknowledgement). The asymmetry is due to the fact that web clients are currently more likely to be SACK-capable than are web servers.

In the absence of SACK, the TCP congestion control mechanisms used by a remote host are likely to be either Tahoe, Reno, or NewReno. While these all share the same underlying congestion control behaviors of Slow-Start, and of Additive Increase and Multiplicative Decrease (AIMD) of the congestion window in congestion-free and congested round-trip times respectively, Tahoe, Reno, and NewReno differ in some of the details of the response to one or more packets dropped from a window of data, and these differences can have a significant effect on the overall performance. The different varieties of TCP are not signaled in packet headers; the only way to determine which is being used by a particular host is to observe a trace of a TCP connection that contains packet drops eliciting the desired behavior.

Inferring the congestion control behavior of a remote host can be done to some extent by actively initiating a transfer

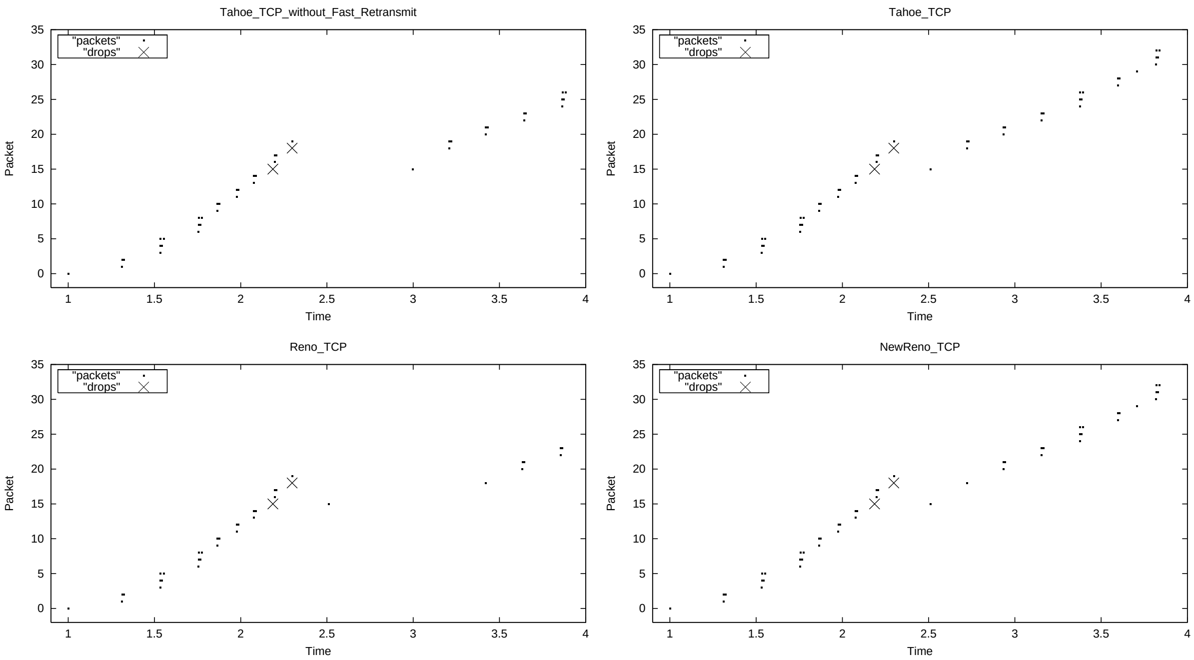

Figure 1: Simulations of Tahoe, Reno, and NewReno TCP

of data from a remote host over a TCP connection, and then passively monitoring the connection's congestion control responses to packet drops on congested links within the network, if in fact packets are dropped. Passive monitoring was used by Paxson in [Pax99], which used the *tcpanaly* program to monitor TCP traces, and to infer information about packet loss patterns, out-of-order delivery, and the durations of congestion periods.

However, it is difficult to determine TCP congestion control behaviors through passive monitoring, because one has to wait for the desired pattern of packet drops to occur. (For example, one might like to observe the response when a specific pattern of multiple packets are dropped from a window of data, or when a retransmitted packet is itself dropped.) Thus, to determine which web servers use Tahoe, Reno, or NewReno TCP, we built TBIT to allow us to control the receipt and sending of TCP packets at the local host, thus introducing specific packet drops at the host itself.

Ours is of course not the only tool that actively elicits TCP behaviors. The NMAP tool uses TCP/IP stack fingerprinting [Fyo98], among other techniques, in order to determine the operating system of a remote machine. The TCP behaviors investigated in NMAP include the response to a FIN probe or to a TCP SYN packets with a bogus flag in the TCP header, the detection of patterns in initial sequence numbers, the use of the Don't Fragment bit, the values in the ACK field, and the TCP options advertised in the SYN packet.

Section 2 describes a simple test for distinguishing between Tahoe, Reno, and NewReno congestion control mechanisms. Section 3 describes the TBIT tool for conducting these tests. Section 4 gives the experimental results from applying this test to popular web servers.

2 A test for distinguishing between Tahoe, Reno, and NewReno TCP

This section describes a procedure that could be used to distinguish between Tahoe, Reno, and NewReno TCP, and illustrates this procedure in the NS simulator.

Tahoe TCP differs from Reno and NewReno even in scenarios where a single packet is dropped from a window of data. After a packet drop, Tahoe TCP reduces the slow-start threshold *ssthresh* and sets the congestion window *cwnd* to 1, entering Slow-Start. Tahoe TCP enters Slow-Start after a packet drop regardless of whether the packet drop was detected by Fast Retransmit or a Retransmit Timeout. In contrast, Reno and NewReno TCP use the Fast Recovery procedure, and do not enter Slow-Start after a single packet is dropped from a window of data.

The simulations in Figure 1 show Tahoe, Reno, and NewReno TCP responding to two packets dropped from a single window of data. For these simulations, the maximum window is five packets, the TCP receiver uses delayed acknowledgements, and packets 13 and 16 are dropped in the simulator. Each graph shows a single simulation, with time on the x-axis and the packet number on the y-axis. These simulations can be run in the NS simulator with the command "test-all-testReno" in the directory “tcl/test”.

For the simulation in the upper left hand corner with Tahoe TCP without Fast Retransmit, the TCP sender detects the first packet loss with a Retransmit Timeout, and retransmits packet 13. When the acknowledgement arrives at the TCP sender acknowledging all packets up to and including packet 15, the TCP sender increases its congestion window from one to two packets, and sends packets 16 and 17. We note that packet 17 is retransmitted unnecessarily, as it had already been received at the TCP receiver. Tahoe TCP without Fast Retransmit in this scenario is characterized by the long delay of a Retransmit Timeout before the retransmission of packet 13, and the unnecessary retransmission of packet 17.

In the simulation in the upper right corner, of Tahoe TCP *with* Fast Retransmit, the TCP sender detects the packet loss after the receipt of three dup ACKs (duplicate acknowledgements), but otherwise behaves exactly as did Tahoe TCP without Fast Retransmit. With Fast Retransmit, the TCP sender infers a packet loss from the receipt of three dup ACKs, and responds by retransmitting the lost packet. Tahoe TCP with Fast Retransmit in this scenario is characterized by the absence of a Retransmit Timeout before the retransmission of packet 13, and the unnecessary retransmission of packet 17.

Reno and NewReno TCP have the same behavior when at most one packet is dropped from a window of data, and differ only in their response to multiple packets dropped from a window of data. That is, when multiple packets are dropped from a window of data, Reno TCP requires either a Retransmit Timeout or multiple Fast Retransmits to recover, while NewReno TCP can generally recover with a single Fast Retransmit.

The lower left corner of Figure 1 shows the simulation with Reno TCP. The TCP sender retransmits packet 13 after receiving three dup acks, but does nothing when it receives a “partial ACK”, the subsequent ACK acknowledging everything up to and including packet 15. After a Retransmit Timeout, the TCP sender retransmits packet 16, with a congestion window of one packet, and then waits for the acknowledgement, which acknowledges everything up to and including packet 17. Reno TCP in this scenario is characterized by a Fast Retransmit for packet 13, a Retransmit Timeout for packet 16, and no unnecessary retransmit of packet 17.

In contrast, the lower right corner of Figure 1 shows the simulation with NewReno TCP. It is identical to Reno TCP up to the retransmission of packet 13. However, when the NewReno sender receives the partial ACK acknowledging all packets up to packet 15, the sender infers that packet 16 was lost, and retransmits it, without waiting for either a Retransmit Timeout or another Fast Retransmit. NewReno TCP in this scenario is characterized by a Fast Retransmit for packet 13, no additional Fast Retransmits or Retransmit Timeouts, and no unnecessary retransmit of packet 17.

Thus, given a transfer from a web server to a client of at least 18 packets, a TCP receiver that does not advertise the “SACK-permitted” option, and the ability to drop specific packets from the TCP connection, we have a simple test that can determine whether the web server at the remote host is using Tahoe, Reno, or NewReno TCP.

3 Description of TBIT

This section gives a brief description of the TBIT code that we used to transfer the procedure described in the section above to an experiment with an arbitrary web server in the Internet.

TBIT depends upon establishing, maintaining and manipulating TCP connections at the user level. An TBIT process fabricates TCP packets and uses raw IP sockets to send them to a remote host. The kernel is prevented from seeing the response TCP packets generated by the remote host by using appropriate firewall rules. At the same time, Berkeley Packet Filter (BPF) services are used to deliver these packets to the TBIT process. In short, the TBIT process establishes and maintains a TCP connection with the remote host entirely at the user level.

An alternate approach to actively eliciting and identifying TCP behavior might have been to use a standard TCP at the web client to request a web page from the server, and to use a tool in the network along the lines of Dummynet [Riz97] to drop specific packets from the TCP connection. A more complex alternative would have been to use a simulator such as NS in emulation mode to drop specific packets from the TCP connection.

However, for our tests we needed to ensure that we would receive a significant number of packets (at least 18) in a single transfer. Rather than search for large objects at each web site, the easiest way to do this is to control the TCP sender’s packet size in bytes, by specifying a small MSS (Maximum Segment Size) at the TCP receiver. In our tests, we specified an MSS of 100 bytes, and a receiver’s advertised window of 5*MSS.

Most of the TBIT code related to establishing and maintaining user level TCP connections is derived from the TCP-based network measurement tool *Sting* [Sav99]. In this section, we focus on describing the TBIT methodology for detecting TCP variants. The interested reader may consult [Sav99] for details on how to establish and maintain a TCP connection at user level.

In the following description, we assume that the remote host is running a web server. This assumption is not required for TBIT to function, as long as there is some process on the remote host that is willing to establish a TCP connection with the TBIT process and send it a sufficient number of data packets. We assume the existence of a web server only for the ease of description.

In our experiments, the TBIT process begins by sending a SYN packet to the remote web server. For the NewReno experiment, the SYN packet does not contain the “SACK permitted” option. The MSS is set to a small value to ensure that several packets will be exchanged between TBIT and the remote web server, and the receiver window size is set to 5*MSS.

Once the SYN packet is sent, the TBIT process waits for up to two seconds to get a SYN-ACK in return. If no SYN-ACK is received within two seconds, the SYN packet is sent again. If no SYN-ACK is received after three attempts, the TBIT process terminates.

In response to the SYN-ACK, TBIT sends the following request to the web server:

```
GET /
HTTP/1.0
User-Agent: Mozilla/4.0 (compatible;
  MSIE 5.0; Windows NT; DigExt; TBIT)
```

The entire request is sent in a single TCP packet. If no acknowledgment is received within 2 seconds, the request is sent again. After three unsuccessful attempts, the TBIT process terminates.

If an acknowledgment is received, the TBIT process starts receiving the data packets from the remote web server. Acknowledgments for these data packets are generated in a manner similar to a normal TCP receiver, except that certain packets are deliberately dropped. The entire exchange of packets is carefully monitored to infer the version of the congestion control algorithm running on the remote web server.

Separate experiments were run to test whether each web server supports timestamps and the SACK option. In the experiment to test which web servers use SACK, the TBIT process sends a SYN packet to the remote web server with a "SACK permitted" option, and observes whether the returning SYN-ACK from the server also has the "SACK permitted" option set. We have not yet run any tests to explore the congestion control behavior of the SACK-enabled TCP connections, but we have already run across one behavior that varies among different deployed SACK implementations.

Consider a SACK TCP connection, where both the SYN and the SYN-ACK advertised the "SACK permitted" option, but the TCP data receiver doesn't include the SACK option in any of its acknowledgements. (We inadvertently used this scenario in the first incarnation of our experiments.) Consider a dup ACK with no SACK block in a SACK-enabled connection. Some SACK-enabled web servers will consider such a non-SACK dup ACK as a valid dup ACK for the purposes of Fast Retransmit, and other SACK-enabled web servers will not. (A TCP sender in a SACK-enabled connection could reason that if the data receiver is SACK-capable, then the dup ACK with no SACK block indicates that the receiver had not received any data above the Acknowledgment Number reported in the TCP header, and that in this case the non-SACK dup ACK is not an indication of a packet drop.)

4 Experimental Results

To test where web servers were using Tahoe, Reno, or NewReno TCP for non-SACK-enabled TCP connections, we implemented the procedure described above, requesting an object from the web site, and dropping the 13th and 16th packets at the TCP receiver. To make sure that the TCP transfer was sufficiently long to take at least 18 packets, the TCP receiver requested a packet size of 16 bytes. We also delayed the sending of each acknowledgement packet by 10 ms, to restrict the transfer rate.

In our experiment, we tested the congestion control mechanisms of the top web servers listed from the 100hot web site at URL "http://www.100hot.com/directory/100hot/". We note that this is a commercial web site, and the only assumption that we are making about their list of "The Web's 100 most popular sites" is that this list is likely to contain some selection of high-traffic web servers in the Internet.

Figure 2 shows a sampling of our experimental results. Detailed results of the experiments are given on the TBIT web page [TBI]. The server identified as Tahoe in Figure 2 is identified by NMAP as running Linux 2.0.35-38. Correlating NMAP with TBIT suggests that Linux 2.1.122 machines use NewReno.

Many of the entries on the list map to more than one web server. Of the 136 individual web servers that we successfully tested from this list, 84 used NewReno TCP, 31 used some variant of Reno, and 21 used some variant of Tahoe. Figure 2 shows experiments illustrating Tahoe without Fast Retransmit, Tahoe with Fast Retransmit, Reno, and NewReno. 56 of the 136 servers advertised the 'SACK-permitted' option. Detailed results of the experiments are given on the TBIT web page [TBI].

The experiments with the NewReno test were carried out as follows.

- A list of host names was obtained from the web page: "http://www.100hot.com/directory/100hot/".
- A list of all IP addresses associated with each host was generated using DNS lookup.
- For each <*hostname, ip_address*> pair, the experiment described above was carried out at three different times between 4:40 PM and 7:40 PM on July 15th, 2000.
- The result was considered valid if and only if both the following conditions were satisfied.
  - At least two experiments were successfully completed.
  - All successful experiments returned the same answer.
  - The tests for the SACK-permitted option and the timestamp option also completed successfully (meaning that the same answer was returned at least twice).

There is no transparent web caching at our service provider. However, many busy web servers employ transparent caching

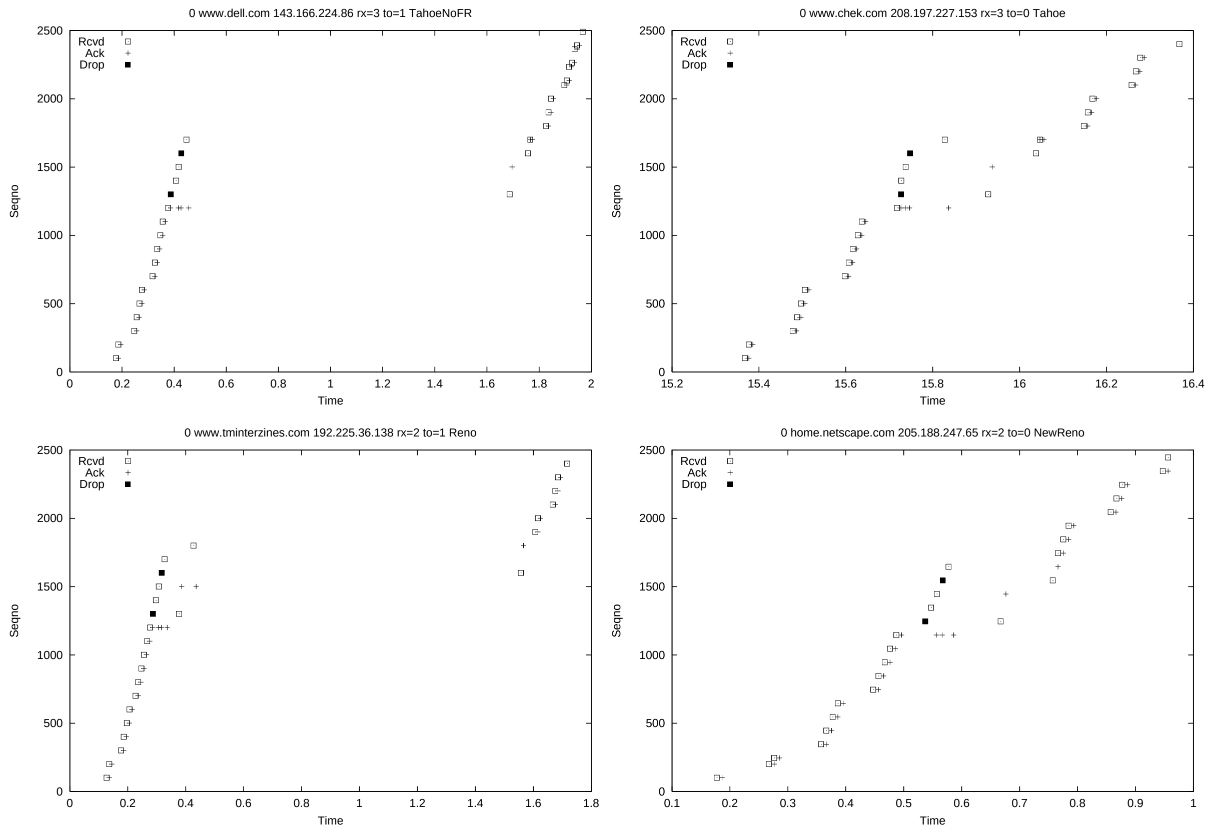

Figure 2: Web servers with Tahoe without Fast Retransmit (upper left), Tahoe (upper right), Reno (lower left), and NewReno (lower right)

near their source. Also, some large websites use TCP-level load balancing between several hosts that respond to the same IP address. These mechanisms imply that if the experiments are carried out again at different time and/or from a different client address, the results may not be the same in all cases.

One of the surprises in the experiment was the number of web servers that didn't use Fast Retransmit, or at least, that didn't use Fast Retransmit for the scenario for our test. Some of the web servers labeled as Tahoe without Fast Retransmit could be using Tahoe, Reno, or NewReno with a 'dupack threshold' higher than three. (The dupack threshold is the number of duplicate acknowledgements required to trigger Fast Retransmit.) However, when we investigated further, by seeing the web server's response to a single packet dropped from a window of five packets, most of these servers did not Fast Retransmit in this case either, but waited for a Retransmit Timeout. Some of the servers classified by our program at Tahoe without Fast Retransmit are at www.microsoft.com, and advertise the SACK-permitted option. We are investigating this behavior further.

The only other unexpected TCP behavior from our experiment is that of the Aggressive Reno and Aggressive Tahoe TCPs, which send two packets instead of one after a Fast Retransmit. To illustrate, the top graph in Figure 3 shows Reno TCP with a pattern of two Fast Retransmits rather than the Fast Retransmit followed by a Retransmit Timeout illustrated in the lower left corner of Figure 2. The bottom graph in Figure 3 shows one of the web servers with Aggressive Reno TCP. The server identified as Aggressive Reno in Figure 3 is reported by NMAP as running Digital UNIX OSF1.

We plan to run additional experiments to verify that the TCP implementations in web servers use conformant TCP, in that they halve the congestion window by half in response to a packet drop. We have not explicitly tested for this in our current test.

5 Conclusions

We have described TBIT, a TCP Behavior Identification Tool, and described its use to identify whether the TCP stack in a web server is using Tahoe, Reno, or NewReno congestion control mechanisms for TCP connections with non-SACK-enabled browsers. Both TBIT and our experimental results are available from the TBIT web page [TBI].

We plan to run additional experiments with TBIT to ran-

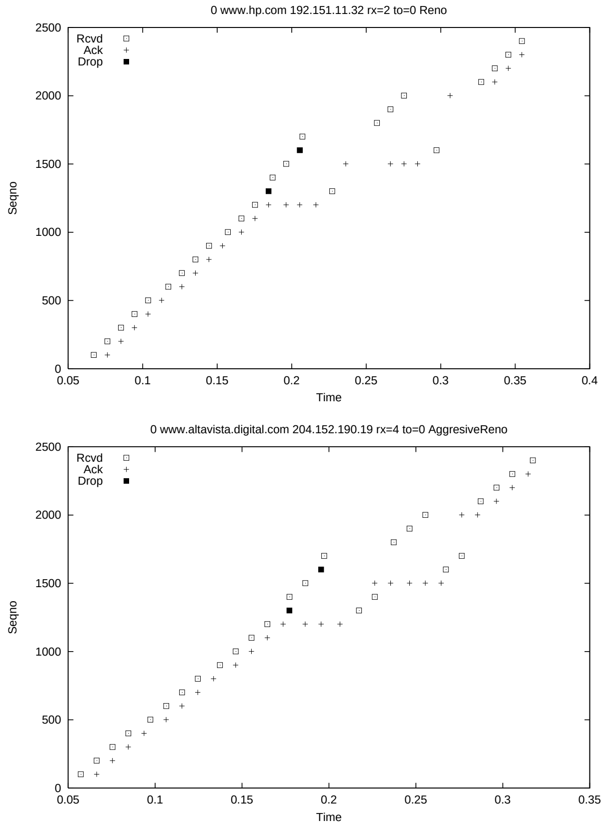

Figure 3: Reno (top) and Aggressive Reno (bottom)

domly probe the IP address space for web servers, and to test other aspects of the Slow-Start and congestion control behavior of TCP implementations.

One possible use of TBIT would be to create a general series of tests that can test a TCP implementation for conformance in a range of aspects, in addition to the congestion control behavior.

6 Acknowledgements

We thank Mark Handley for helpful discussions, and Stefan Savage for his immensely-helpful Sting code.

References

[All00] M. Allman. "A Web Server's View of the Transport Layer". In preparation. URL "http://roland.grc.nasa.gov/ mallman/tcp-opt-deployment/", June 2000.

[APS99] M. Allman, V. Paxson, and W. Stevens. "TCP Congestion Control". RFC 2581, Apr. 1999.

[FF96] K. Fall and S. Floyd. "Simulation-based Comparisons of Tahoe, Reno, and Sack TCP". *ACM Computer Communication Review*, Jul. 1996. URL http://www-nrg.ee.lbl.gov/nrg-papers.html.

[FH99] S. Floyd and T. Henderson. "The NewReno Modification to TCP's Fast Recovery Algorithm", "Apr. 1999. RFC 2582.

[Flo00] S. Floyd. "Questions Web Page", 2000. URL "http://www.aciri.org/floyd/questions.html".

[Fyo98] Fyodor. "Remote OS detection via TCP/IP Stack FingerPrinting", Dec. 1998. URL "http://www.insecure.org/nmap/nmap-fingerprinting-article.html".

[Jac88] V. Jacobson. "Congestion Avoidance and Control". *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 314–329, 1988. An updated version is available via ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z.

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. "TCP Selective Acknowledgment Options". RFC 2018, Apr. 1996.

[Pax99] V. Paxson. "End-to-end Internet Packet Dynamics". 7(3):139–152, Jun. 1999.

[Riz97] L. Rizzo. "Dummynet: a Simple Spproach to the Evaluation of Network Protocols". *ACM Computer Communication Review*, 27(1), Jan. 1997.

[Sav99] S. Savage. "Sting: a TCP-based Network Measurement Tool". *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems*, pages 71–79, Oct. 1999.

[TBI] "The TBIT Web Site". URL "http://www.aciri.org/tbit/".