Ns Simulator Tests for Reno FullTCP

Kevin Fall, Sally Floyd,[*,†] and Tom Henderson[‡]

July 25, 1997

This is a first rough draft of documentation on the simulation test suite for FullTCP. This document is very much in progress.

1 Introduction

This note shows the validation test suite for the implementation of Reno FullTCP in the ns simulators (ns-1 [Ns-1] and ns-2 [Ns-2]). This implementation was motivated in part by code contributed to ns-1 by Kathie Nichols (nichols@com21.com). The current implementation in ns-1 is from Kevin Fall, and is based primarily on the 4.4BSD TCP implementation. The code was ported by ns-2 by Tom Henderson.

FullTCP differs from the earlier TCP implementations in ns in the following ways:

- The use of SYN/FIN packets.
- Two-way data transfer in a single connection.
- Packets in a single connection could be variable sizes in bytes.
- Other?

This use of SYN packets can be of critical importance in simulations using many very short data transfers. This version of TCP currently sends data on the 3rd segment of an initial 3-way handshake.

Currently FullTCP is only implemented with Reno congestion control, but ultimately it should be available with the full range of congestion control algorithms (e.g., Tahoe, SACK, Vegas, etc.).

These tests can be run in the simulator ns-1 with the command "test-all-fulltcp"; the input files are available in "test-suite-fulltcp.tcl". These tests can be run in the newer simulator ns-2 with the command "test-all-fulltcp" in the directory tcl/test; the input files are available in "test-suite-full.tcl" in that directory.

On each page of the following section, a graph shows the results of each simulation in the test suite. For each graph, the x-axis shows the time in seconds. The y-axis shows the sequence number. There is a filled square on the graph for each packet as it arrives and departs from the congested gateway, and a "x" for each packet dropped by the gateway.

For more information about the simulator, see the ns web page [Ns-2], or the main document about simulator tests [F96].

2 Details of the FullTCP simulations

While FullTCP does not implement an entire reference TCP state machine, it attempts to closely emulate the behavior of a reference implementation. The following differences are known to exist:

- The sequence space does not wrap above 2^{31} bytes, and the simulation will not function correctly if that number is exceeded.
- This version of TCP currently sends data on the 3rd segment of the initial three way handshake (i.e., the third segment is an ACK with data), whereas many actual implementations don't send data until the 4th segment.
- There is no dynamic receiver's advertised window. The advertised window is simulated by simply telling the sender a bound on the window size (wnd_).
- In real TCP, a user process performing a read (via PRU_RCVD) calls tcp_output each time to (possibly) send a window update. Here we don't have a user process, so we simulate a user process always ready to consume the entire receive buffer.
- The state variables wnd_, wnd_init_, cwnd_, and ssthresh_ are in segment units. Sequence and ack numbers are in byte units.
- Some existing TCPs delay the ack for the first segment, which can cause connections to take longer to start up than if the first segment were acked immediately. This implementation acks the first segment immediately.

*This work was supported by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. In addition, this material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. DABT63-96-C-0054. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA.

†Kevin and Sally are both with the Network Research Group at Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, CA 94704. Their email addresses are kfall@ee.lbl.gov and floyd@ee.lbl.gov.

‡Tom Henderson is at the University of California at Berkeley.

The following defaults are configurable in the tcl/lib/ns-default.tcl file:

- "set segsperack_ 1": If set to 1, acknowledges every segment. If set to an integer greater than 1, permits the TCP to delay sending an acknowledgment until that number of segments has been received (the BSD default is 2).
- "set segsize_ 536": With 40 bytes of TCP/IP header, the packet size becomes 576 bytes.
- "set tcprexmtthresh_ 3": Waits for 3 duplicate acks before entering fast retransmit.
- "set iss_ 0": Sets initial sequence number to 0.
- "set nodelay_ false": When set to true, disables Nagle algorithm.
- "set data_on_syn_ false": If set to true, allows data to be piggybacked on a SYN.
- "set dupseg_fix_ true": If set to true, fixes a bug that causes duplicate segments with duplicate acks not to trigger a fast retransmit [F94].
- "set dupack_reset_ false": When true, sets the counter of consecutive duplicate acknowledgments to 0 when either a non-zero length segment or a segment with a window change is received during ACK procesing.
- "set interval_ 0.1": Sets the delayed ack interval to 100ms.

The trace file for FullTCP has slightly expanded entries. There are three fields included beyond the packet uid_: the ack number, the TCP flags value, and the TCP header length. To enable tracing of these values, the following line must be included in the tcl script: Trace set show_tcphdr_ 1. See tcl/test/test-suite-full.tcl for an example.

3 Acknowledgements

Our old simulator tcpsim was a version of the REAL simulator [K88a] built on Columbia's Nest simulation package [BDSY88a], with extensive modifications and bug fixes made by Steven McCanne and by Sugih Jamin. For the simulator ns-1 [Ns-1], written largely by Steve McCanne, this has been rewritten into Tcl, with the simulation engine implemented in C++. Ns-2 [Ns-2] is the successor to ns-1.

References

[Ns-1] Ns-1. Available via http://www-nrg.ee.lbl.gov/ns/.

[Ns-2] Ns-2. Available via http://www-mash.cs.berkeley.gov/ns/.

[BDSY88a] Bacon, D,. Dupuy, A., Schwartz, J., and Yemimi, Y., "Nest: a Network Simulation and Prototyping Tool", *Proceedings of Winter 1988 USENIX Conference*, 1988, pp. 17-78.

[F94] Floyd, S., "TCP and Successive Fast Retransmits". Technical report, October 1994. Available from http://ftp.ee.lbl.gov/floyd/.

[F96] Floyd, S., "Simulator Tests", URL "ftp://ftp.ee.lbl.gov/papers/simtests.ps.Z". This is an expanded version of a note that was first made available in October 1994.

[K88a] Keshav, S., "REAL: a Network Simulator", *Report 88/472*, Computer Science Department, University of California at Berkeley, Berkeley, California, 1988.

4 Reno FullTCP Simulations

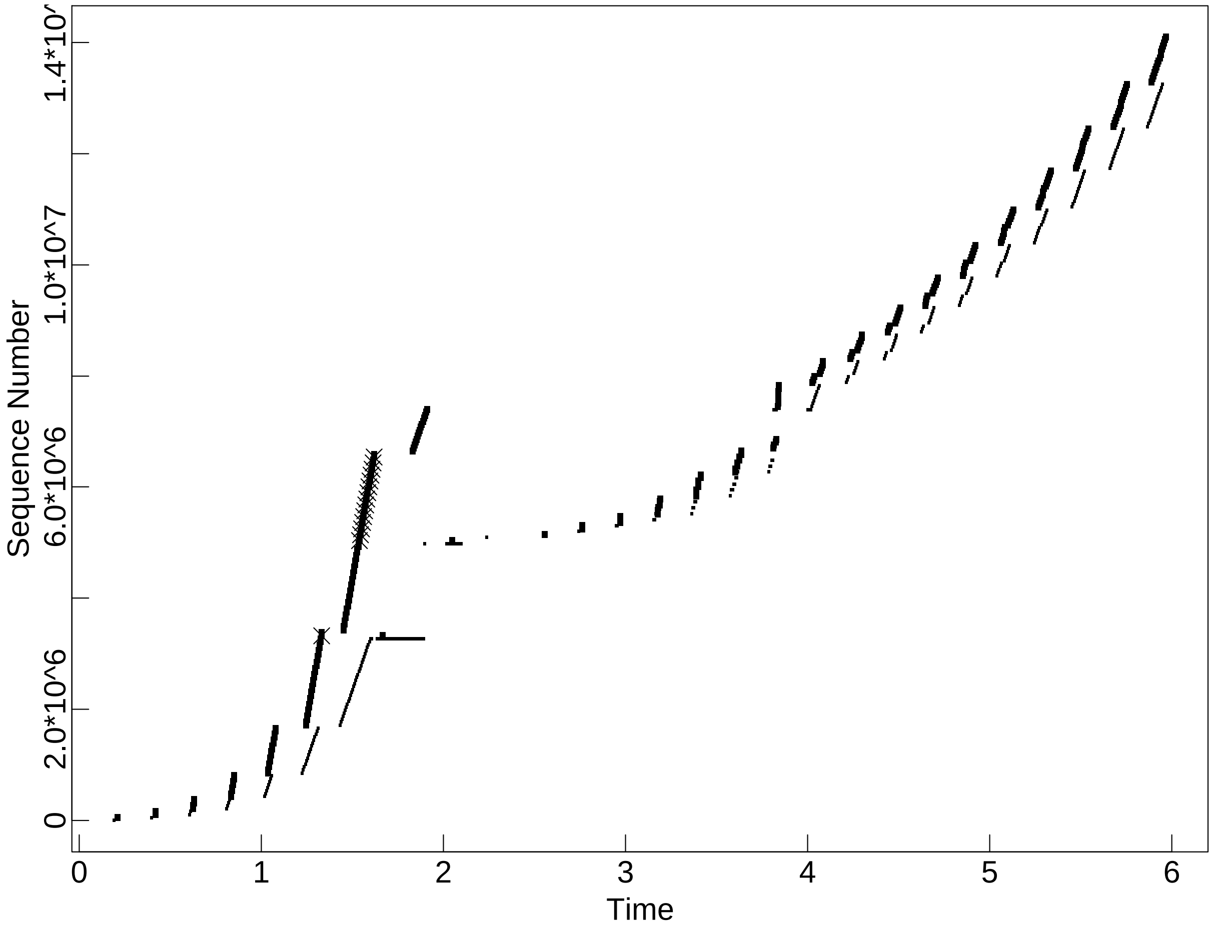

Figure 1: Test “full”.

This test shows a single TCP connection. Data transfer in the “forward” direction begins at time 0. There is no data transfer in the reverse direction.

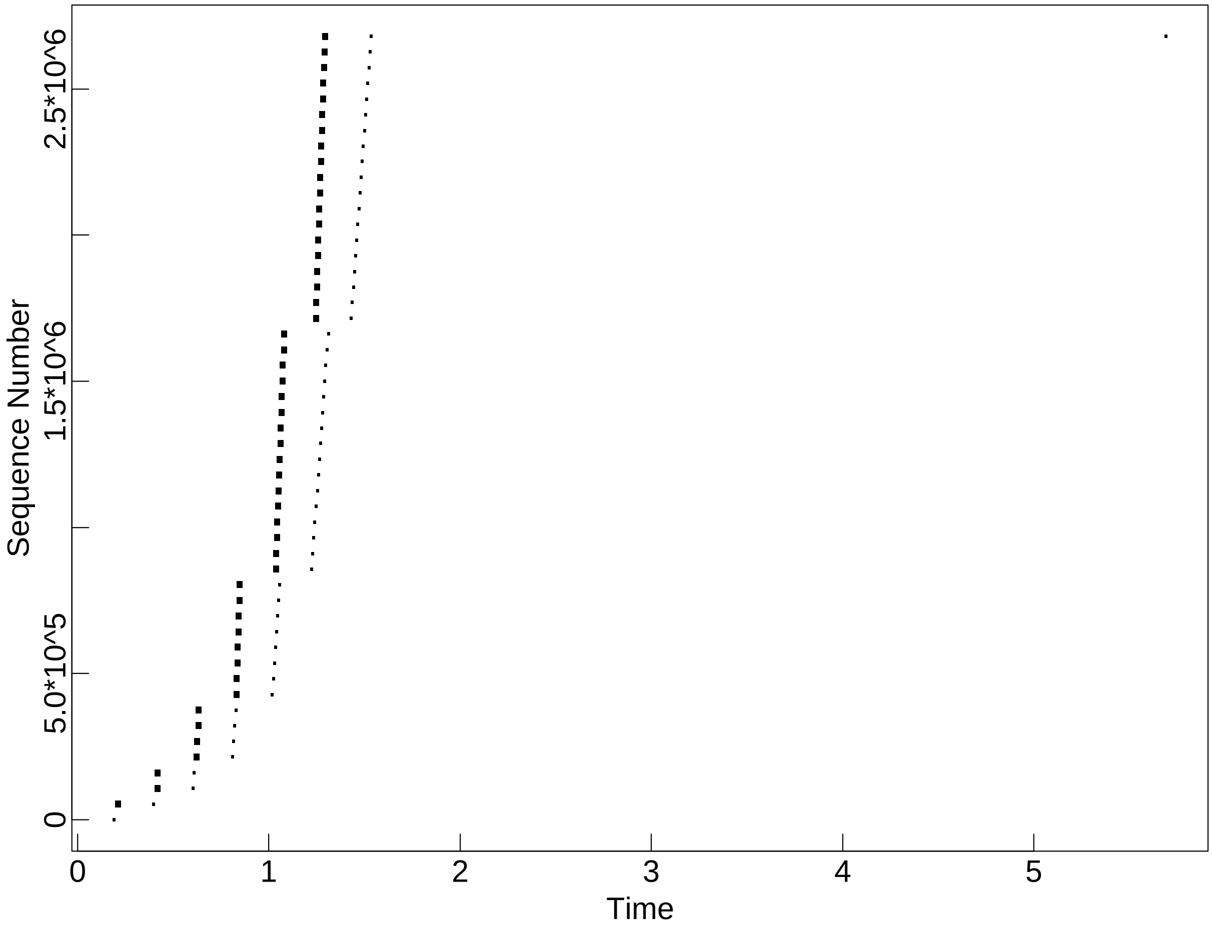

Figure 2: Test “close”.

This test shows a single TCP connection. Data transfer in the “forward” direction begins at time 0, and is limited to 50 packets. The Node A TCP closes the connection at time 5.5. There is no data transfer in the reverse direction.

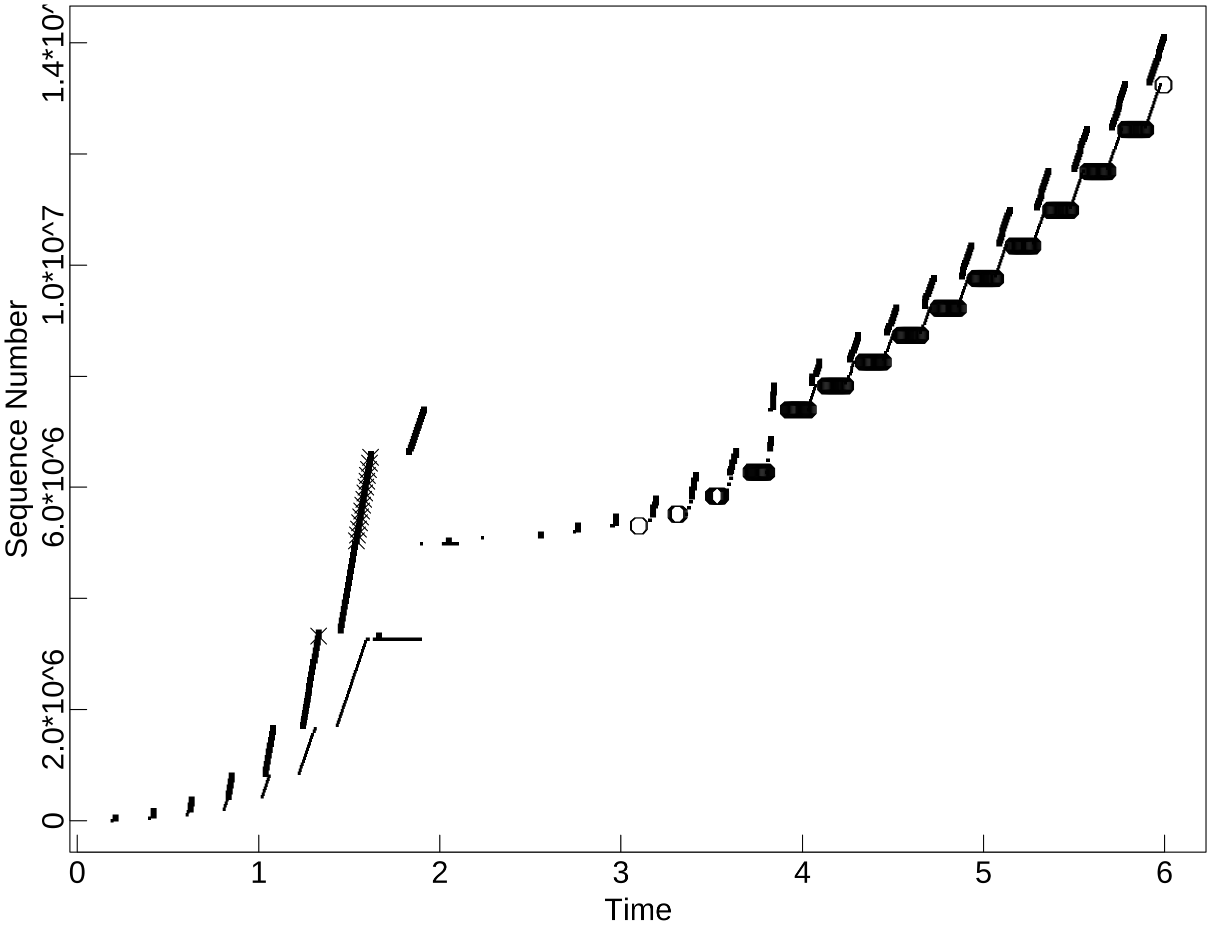

Figure 3: Test “twoway”.

This test shows the two-way transfer of data. Data transfer in the “forward” direction begins at time 0. Data transfer in the “reverse” direction begins at time 3.0.

The data packets leaving Node A are shown with squares, and pure acks (acks in packets without data) in the reverse direction are shown with dots. Acks in the reverse direction `piggy-backed' on data packets are shown by octagons.

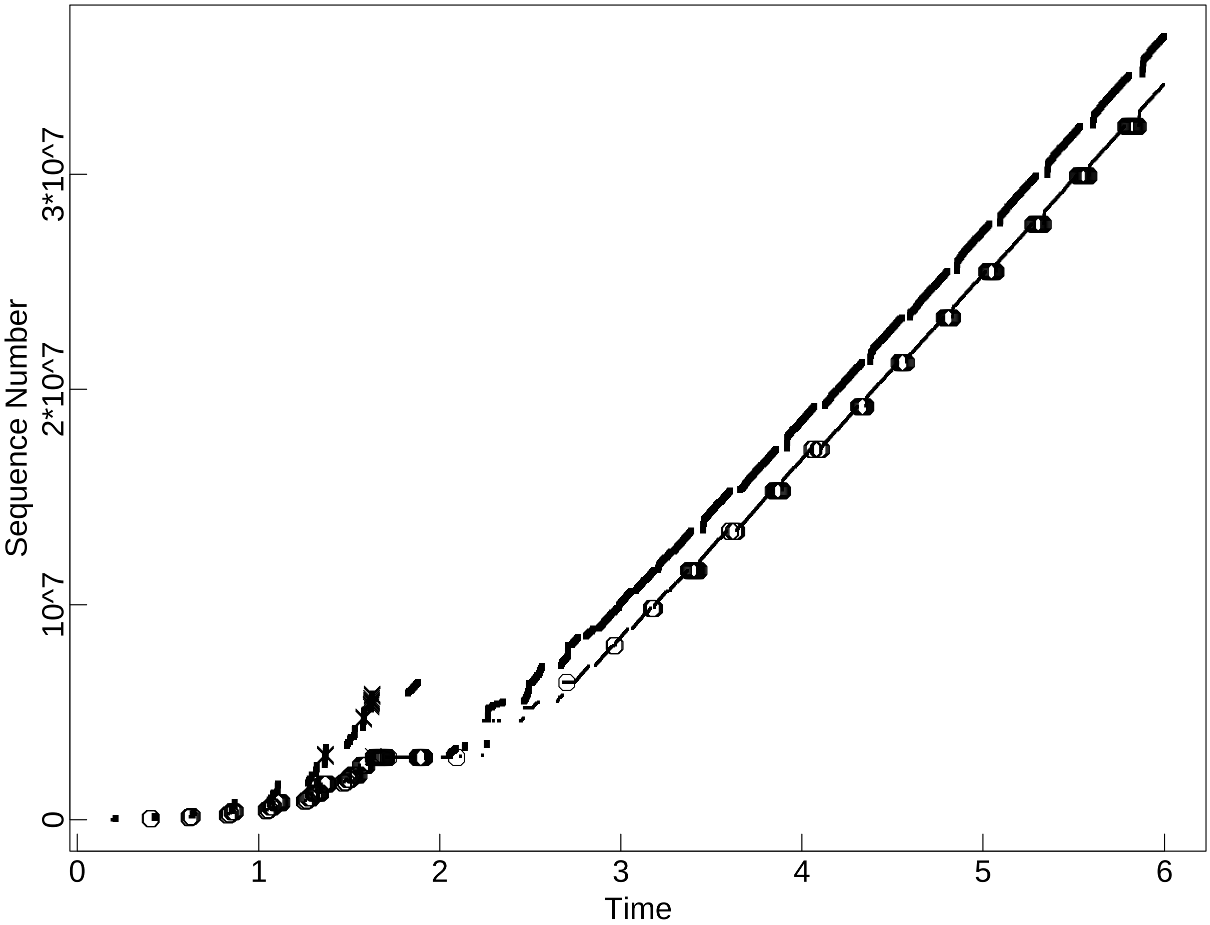

Figure 4: Test “twoway0”.

This test shows a variant on the two-way transfer of data. Data transfer in the “forward” and “reverse” directions both start at time 0.

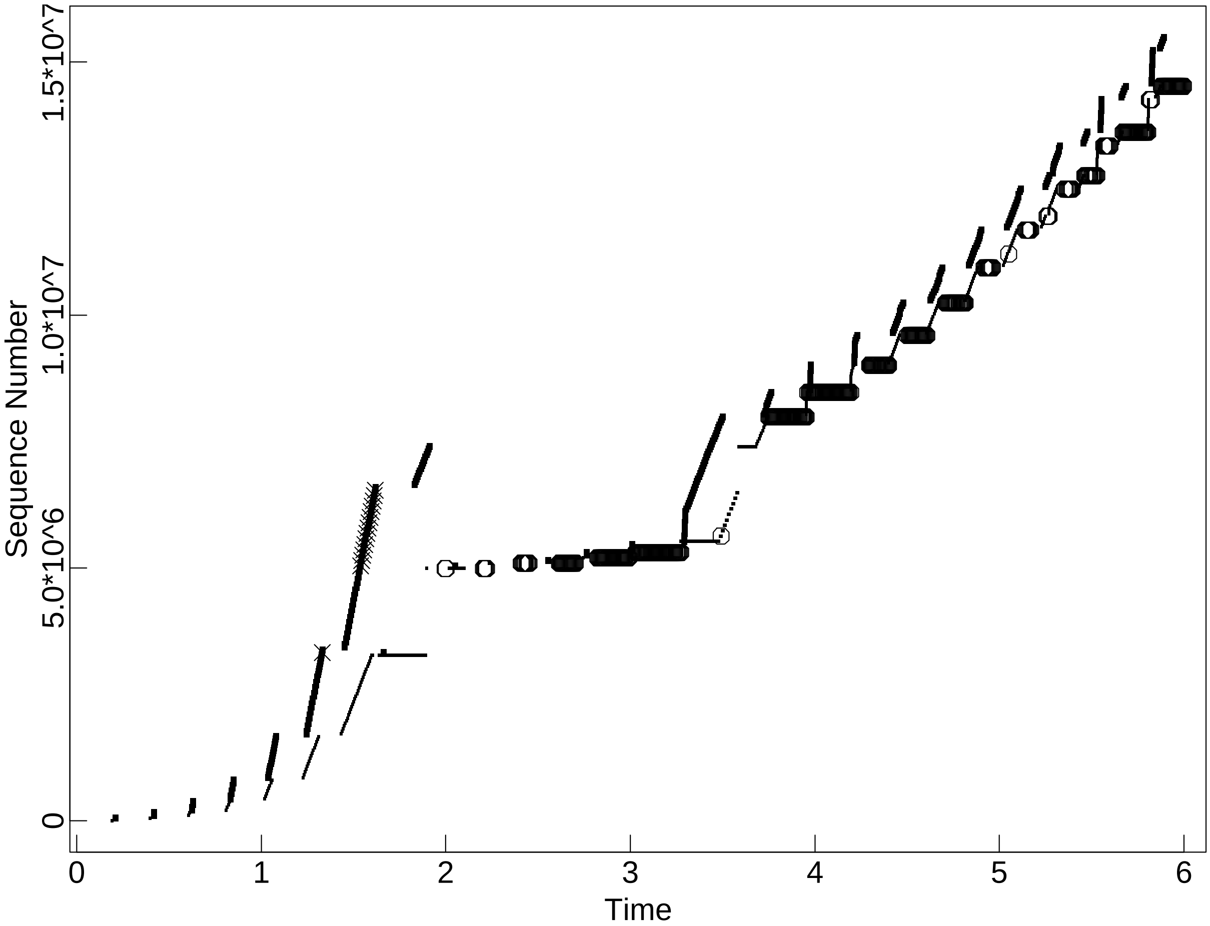

Figure 5: Test “twoway1”.

This test shows another variant on the two-way transfer of data. Data transfer in the “forward” direction begins at time 0. Data transfer in the “reverse” direction begins at time 1.9.

Note the ACK compression that results from the two-way transfer of data, with the resulting burstiness in the sending of packets at times 3.5, 4,5, and 5.5.

4.1 BSD-Compatibility Mode

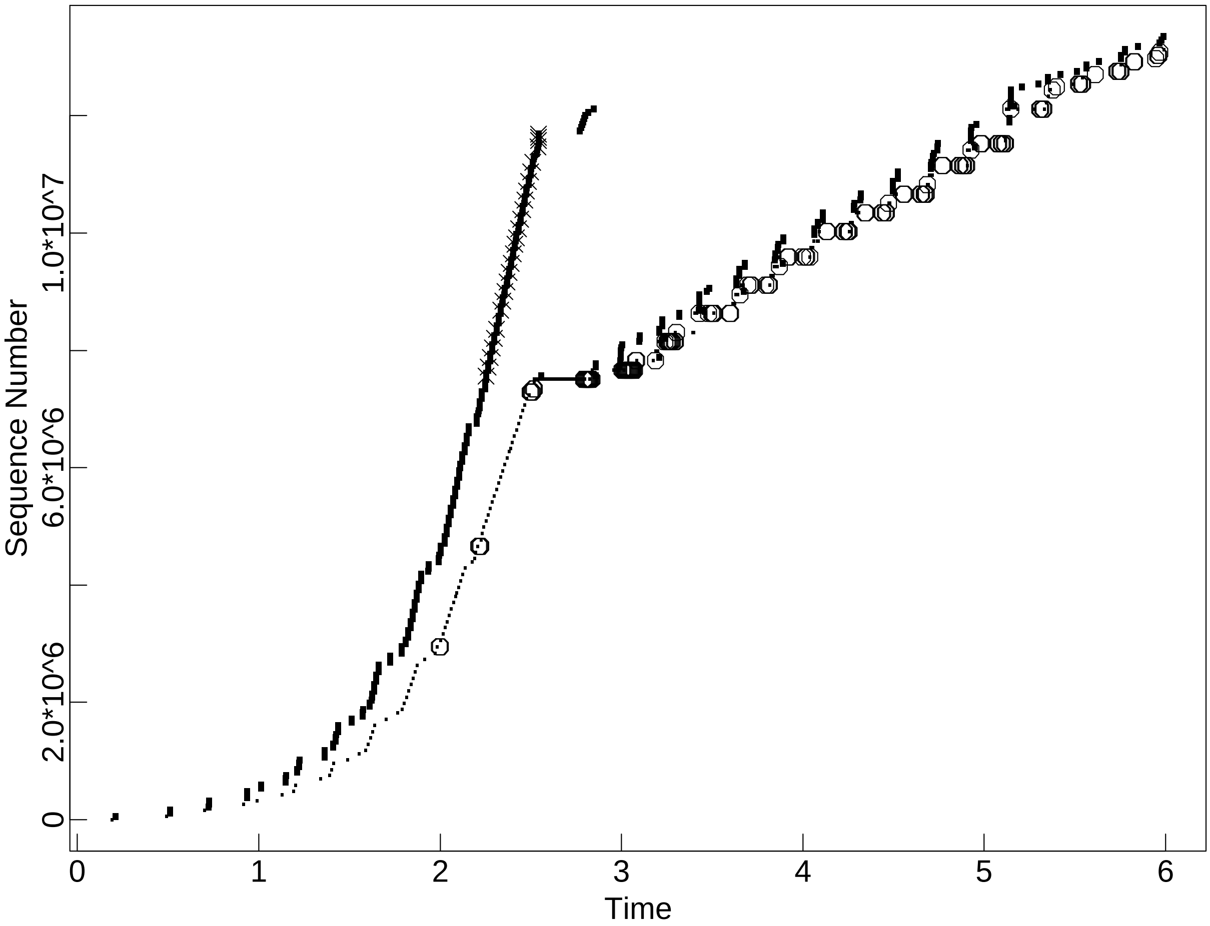

Figure 6: Test “bsdcompat_twoway1”.

This test is identical to test “twoway1” on the previous page, with the difference that the TCP implementation is closer to the actual implementation in 4.4 BSD. Thus, in bsd-compatibility mode FullTCPs uses delayed ACKs: “set segsperack 2”.

In bsd-compatibility mode various bug fixes in the simulator are turned off: Bsd-compatibility mode uses “set dupseg_fix false” to turn off a bug-fix that would mean that duplicate segments with dup acks don't trigger a fast retransmit. Bsd-compatibility mode uses “set bug-fix false” to turn off a bug-fix that eliminates unnecessary Fast Retransmits from one window of data [F94]. In addition, bsd-compatibility mode uses “set dupack_reset true” to zero the counter of duplicate acknowledgements in response to an incoming data packet or a change in the receiver's advertised window.

In addition, bsd-compatibility mode uses “set data_on_syn false”, which does not allow data in the SYN packet.

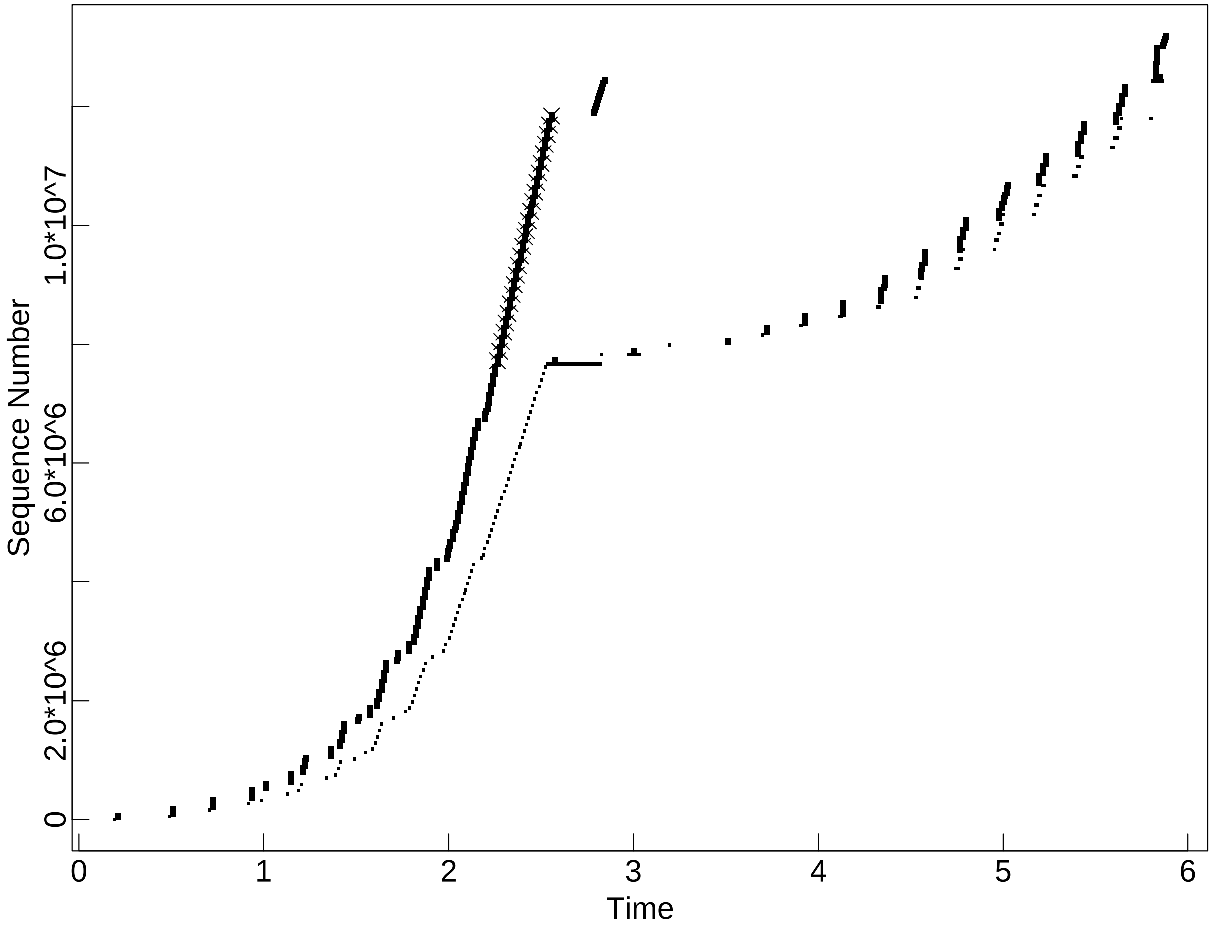

Figure 7: Test “oneway_bsdcompat”.

This test shows a TCP connection with one-way data transfer. This test is identical to that in Figure 1, except that the TCP is in bsd-compatibility mode.

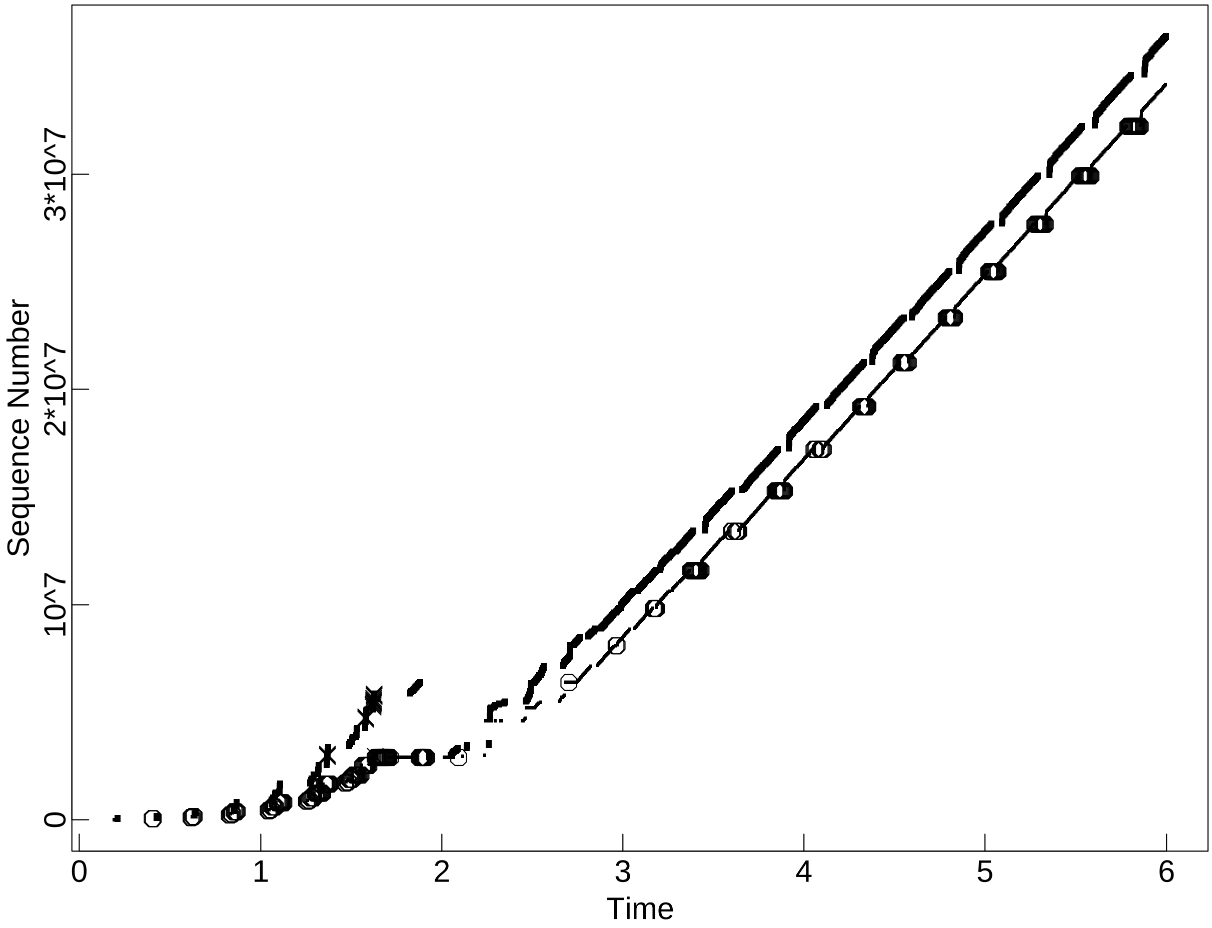

Figure 8: Test “twowayrandom”.

This test shows another variant on the two-way transfer of data. Data transfer in the “forward” direction begins at time 0. Data transfer in the “reverse” direction begins at a random time chosen between time 0 and time 6. This simulation will look slightly different every time that it is run.

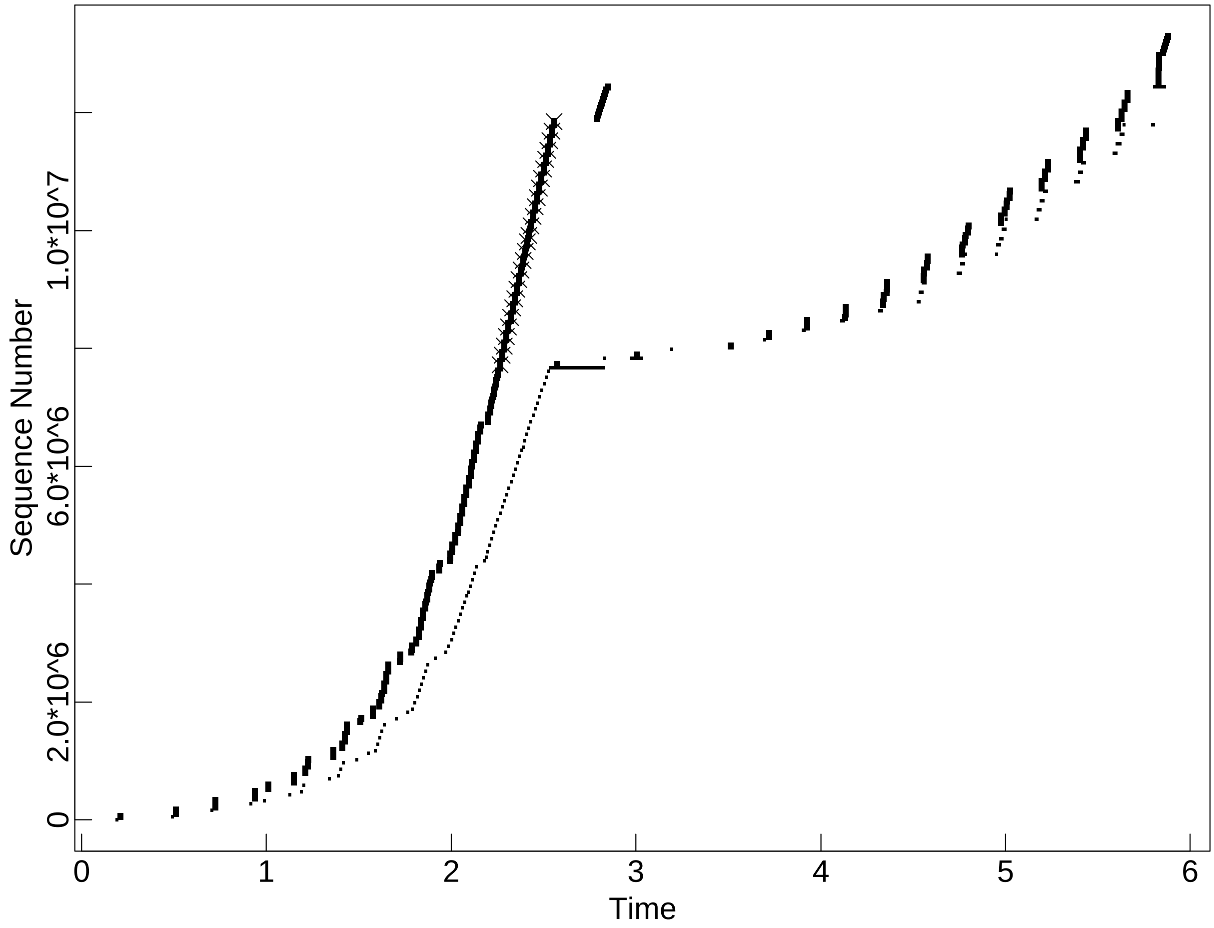

Figure 9: Test "delack".

This test shows a TCP connection with one-way data transfer. This test differs from Figure 1 only in that TCP uses a delayed-ack policy.