ECN Implementations in the NS Simulator

Sally Floyd and Kevin Fall*

Lawrence Berkeley National Laboratory
One Cyclotron Road, Berkeley, CA 94720
floyd@ee.lbl.gov, fall@ee.lbl.gov

December 30, 1998

1 Introduction

This document illustrates the validation test for ECN (Explicit Congestion Notification) in the NS simulator [NS95].

Figure 1 shows a single simulation with two-way traffic, showing the use of the CE (Congestion Experienced) bit in the IP packet header, and the CWR (Congestion Window Reduction) and ECN-Echo bits in the TCP header. We refer to a packet with the CE bit marked as a *CE packet*. Similarly, a *CWR packet* denotes a packet carrying a CWR notification, and an *ECN-Echo ACK packet* denotes an ACK packet with the ECN-Echo bit set.

Figures 2 and 3 show that multiple CE packets in a window of data are considered as a single instance of congestion.

Figures 4, 5, 6, and 7 show that interspersed dropped packets and CE packets are treated as a single instance of congestion. While the behavior in these simulations is specific to Tahoe TCP, similar simulations in the test suite show that the same property holds for the Reno, NewReno, and Sack TCP implementations. Figures 8 and 9 are a separate simulation scenario showing that interspersed dropped packets and CE packets are treated as a single instance of congestion.

Figures 10, 11, and 12 show that interspersed dropped packets and CE packets are treated as a single instance of congestion even when the packet drops are sufficiently severe to result in a retransmit timeout. Figure 13 shows that retransmitted packets with the CE bit set are interpreted as a new instance of congestion.

2 Running the tests in NS

All of the tests in this document can be run in the NS simulator with the commands “test-all-ecn” and “test-all-ecn-ack” in the tcl/test directory.

To run only a single simulation from the file “test-suite-ecn.tcl”, use the command:

```
ns test-suite-ecn.tcl ecn_nodrop_tahoe
```

in the directory tcl/test. For each simulation, more detailed information about the simulation scenario can be found in the simulation files “test-suite-ecn.tcl” and “test-suite-ecn-ack.tcl”

3 Interpreting the diagrams

The top diagram in each figure shows the data packets transmitted in a single TCP session. For each packet, there is a mark when the packet arrives at the congested gateway, and a separate mark, possibly later, when the packet leaves the congested gateway. For each mark, the *x*-axis shows the time, and the *y*-axis shows the packet number mod 90. There is an “X” for each dropped packet, an “X” enclosed in a box for each CE packet, and a solid diamond for each CWR packet.

*This work was supported by DARPA under DARPA grant DABT63-96-C-0105. This is a revised version of a document that was first made available in October 1998.

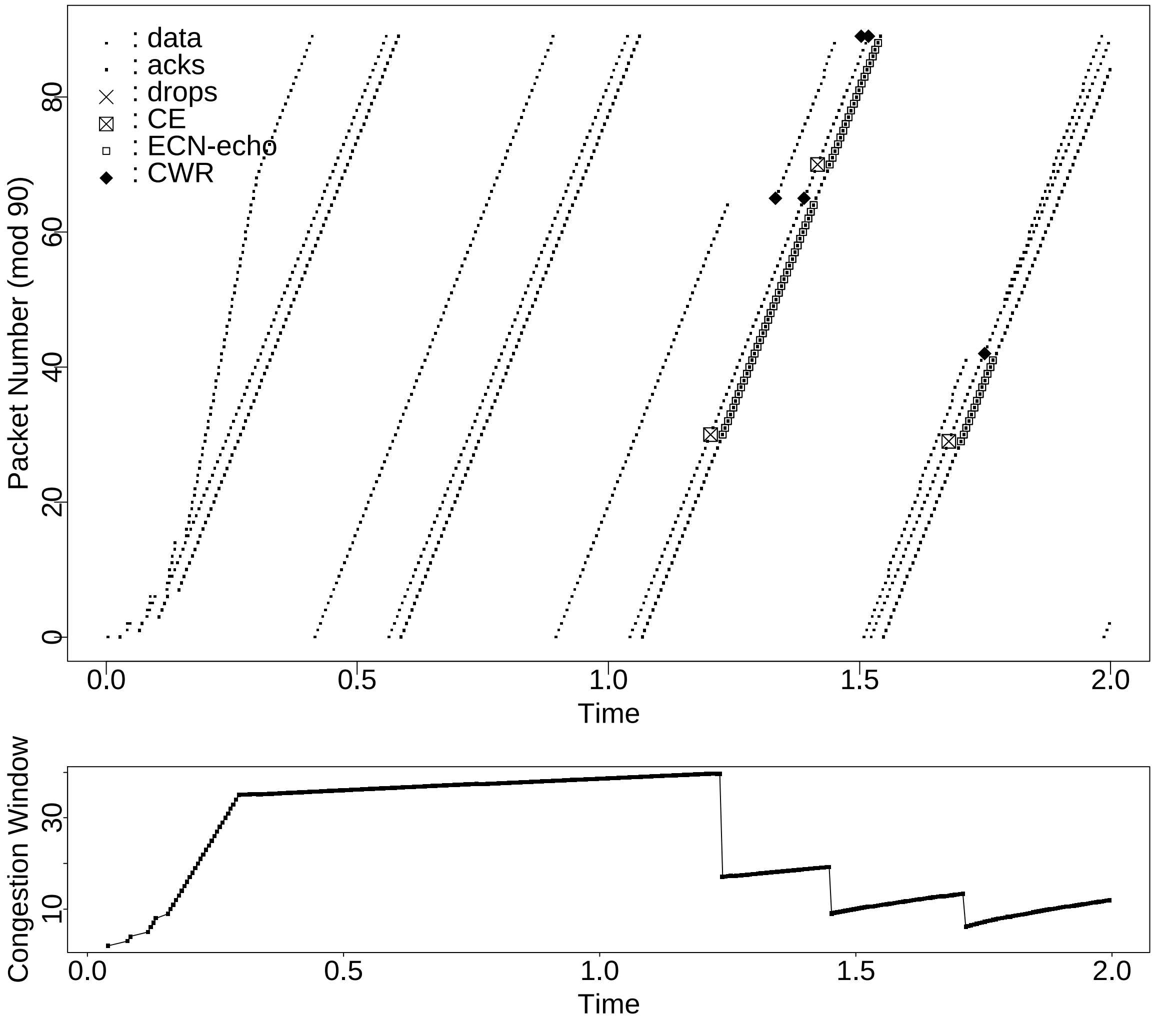

Figure 1: The test “ecn_ack in the file “test-suite-ecn-ack.tcl”.

In addition, Figure 1 includes ACK packets. There is a small dot for each ACK packet arriving at the congested gateway in the reverse direction, and a small open square for each ECN-Echo ACK packet.

The bottom diagram shows the TCP congestion window as a function of time.

4 An illustration of ECN-Echo packets

This test shows the data and ACK packets for an ECN-capable TCP connection. There are no packets dropped in this simulation. Three separate packets have the CE bit, at times 1.2, 1.4, and 1.7. The bottom diagram shows that the TCP sender cuts the congestion window in half after each CE packet.

When the TCP receiver receives a data packet with the CE bit set, the receiver sets the ECN-Echo bit on returning ACK packets. The receiver continues to set the ECN-Echo bit until it receives a data packet with the CWR bit set, indicating that the TCP sender has responded to congestion.

Test 1 uses Sack TCP, but Tahoe, Reno, and NewReno TCP show identical behavior for this scenario.

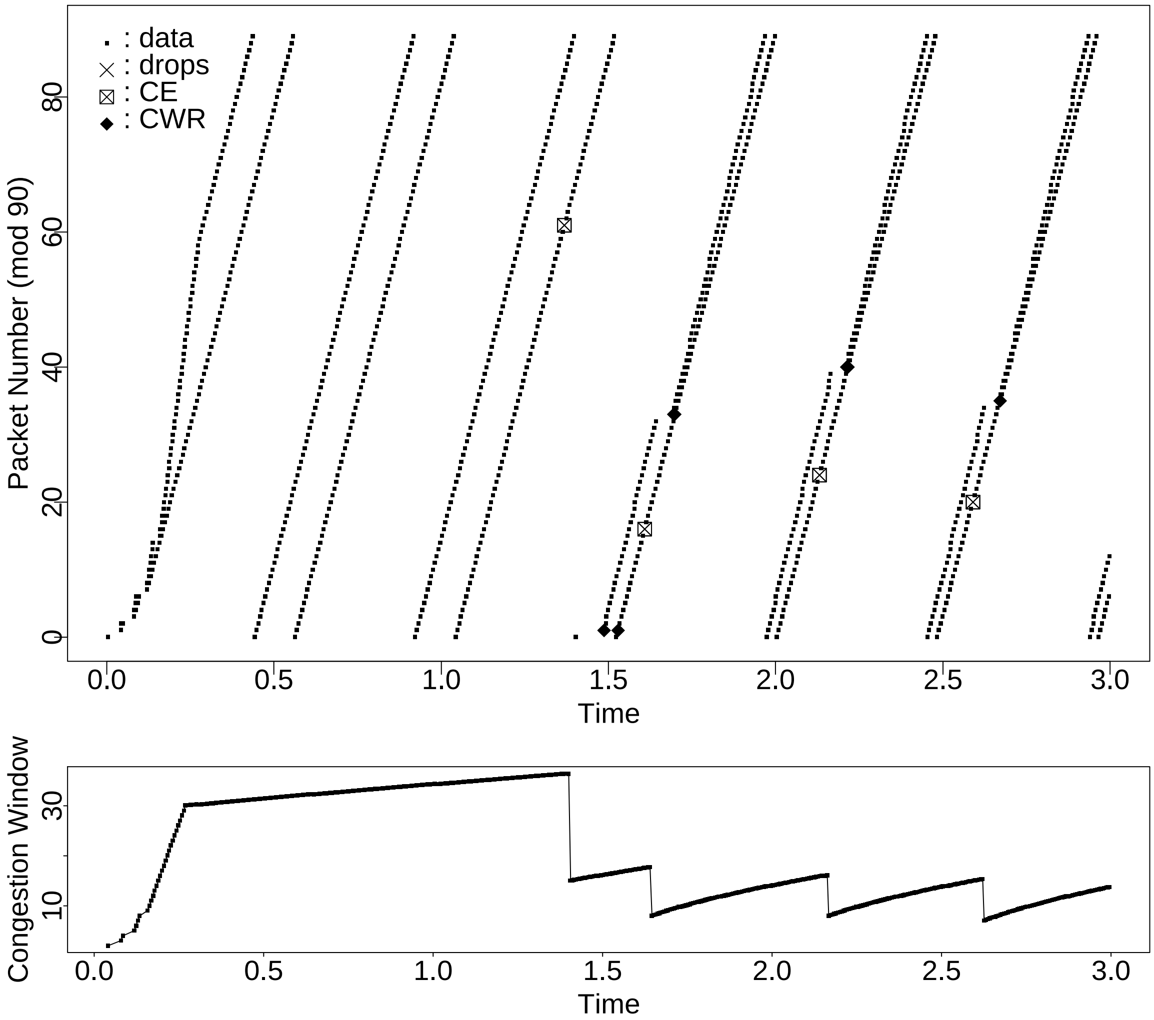

Figure 2: The test “ecn_nodrop_tahoe”.

5 Multiple CE packets

This test shows TCP in an environment with no packet drops but with occasional ECN packets with the CE bit set. Tests 2 and 3 use Tahoe TCP, but for each test Reno, NewReno, and Sack TCP show identical behavior.

For this simulation, when the first packet has its CE bit set at time 1.2, there is a substantial queueing delay, as illustrated by the time between that packet's arrival and departure from the congested gateway. At time 1.5, a CWR packet arrives at the congested gateway, indicating that the TCP sender has reduced its congestion window. This reduction in the congestion window can be seen in the bottom graph, and can also be seen visually in the reduction of the queueing delay at the congested gateway. For simplicity, these figures do not show the ACK packets.

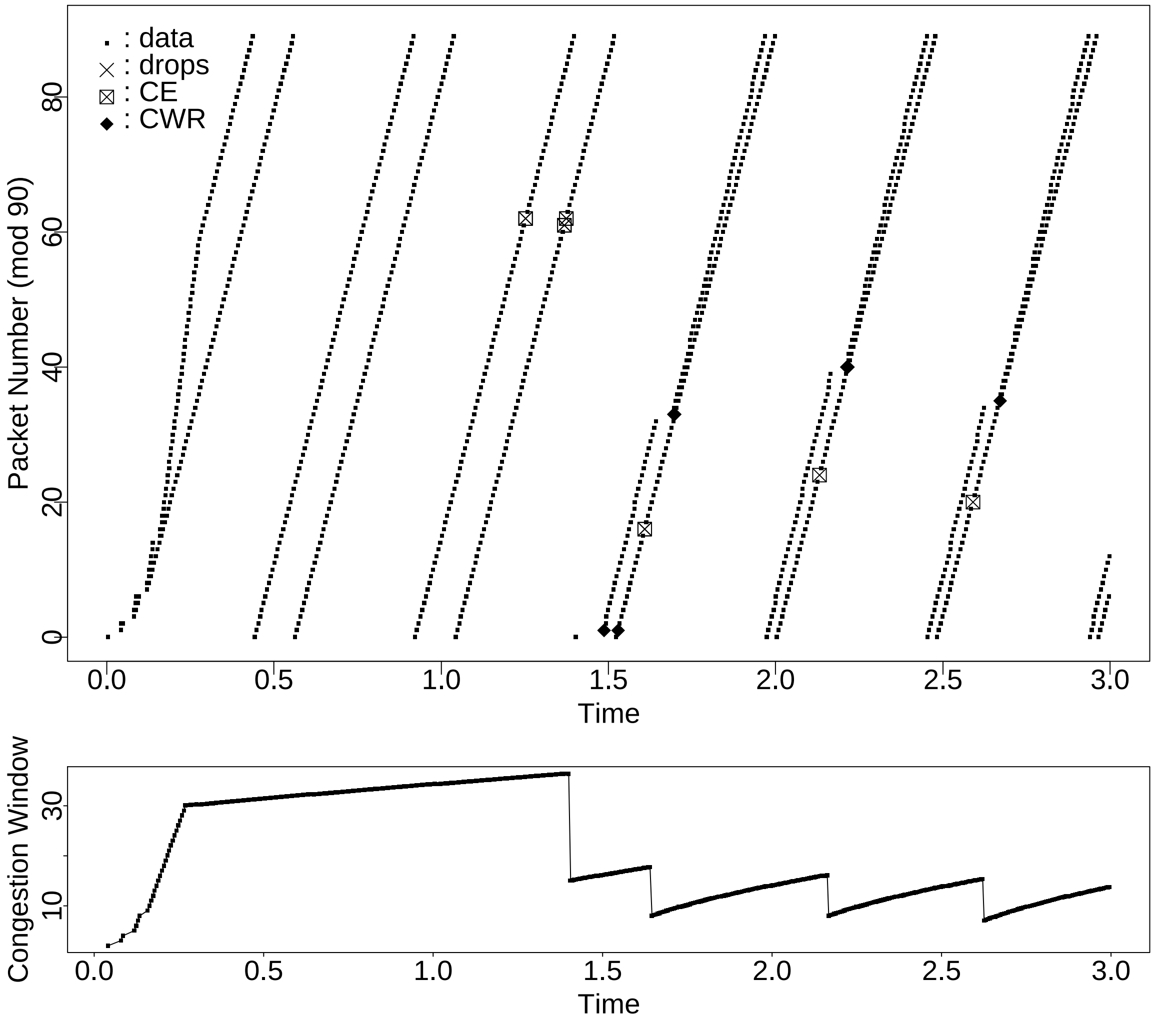

Figure 3: The test “ecn_twoecn_tahoe”.

This test shows that TCP considers two successive CE packets as a single instance of congestion. In particular, TCP cuts its congestion window in half only once in response to the two CE packets.. The tests in Figures 2 and 3 show the same evolution of the congestion window.

The first CE packet in this test, packet 242, has its CE bit set by the RED queue at the congested gateway. Packet 243, in contrast, arrives at the RED queue with its CE bit already set, from explicit “drop_pkt” and “markecn_” commands in the simulation script in the file “test-suite-ecn.tcl”.

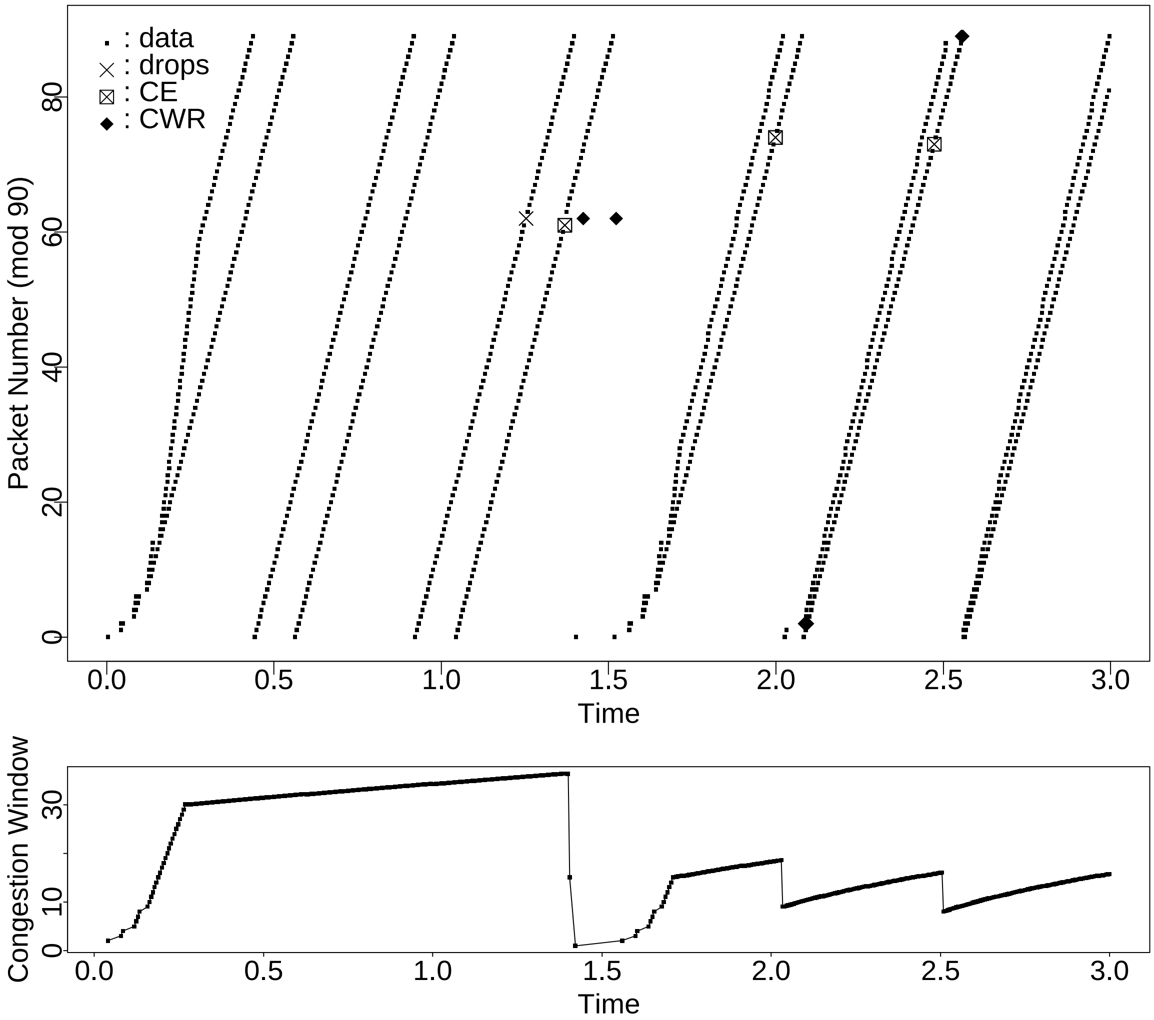

Figure 4: The test "ecn_drop_tahoe".

6 Interspersed dropped packets and CE packets

This test shows that TCP considers a CE packet followed by a dropped packet as a single instance of congestion. Figures 4 through 6 all show the same evolution of the congestion window, with a single Slow-Start to recover from a combination of CE packets and dropped packets at time 1.3. Figures 4 through 9 show behavior specific to Tahoe TCP. Similar simulations in the test suite show that Reno, NewReno, and Sack TCP implementations also respond to interspersed packet drops and CE packets as a single instance of congestion.

For this scenario, the TCP sender reduces the Slow-Start threshold *ssthresh*[1] and cuts its congestion window in half when it receives notification of the CE packet. The TCP sender subsequently Slow-Starts when it infers a packet loss from the receipt of three dup ACKs (duplicate acknowledgements). However, the Slow-Start is not accompanied by a second reduction ssthresh.

[1] The TCP sender reduces *ssthresh* to half the current congestion window or half the receiver's advertised window, whichever is smaller.

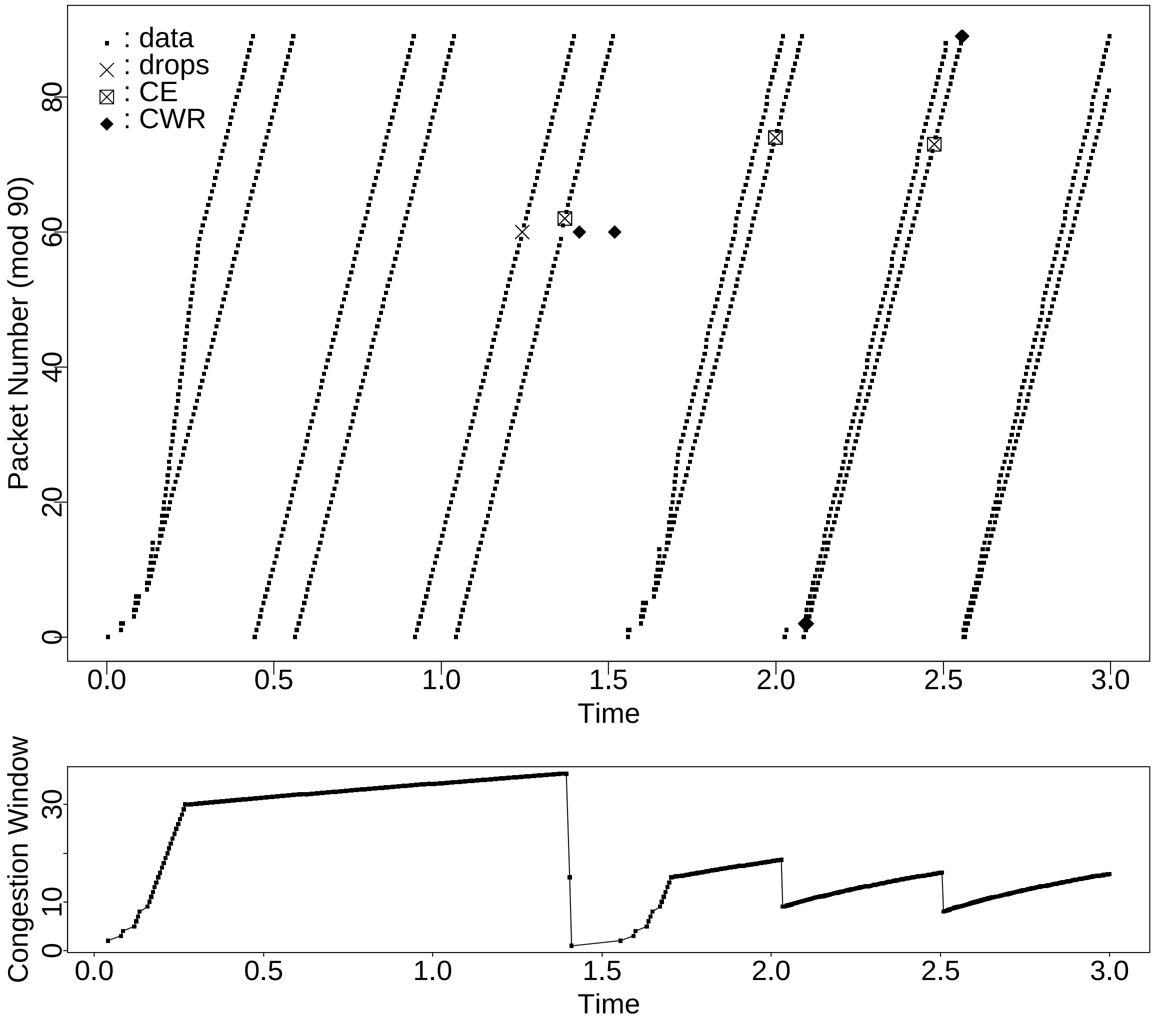

Figure 5: The test “ecn_drop1_tahoe”.

This test shows that TCP considers a dropped packet followed closely by a CE packet as a single instance of congestion.

The second dup ACK received by the TCP sender has the ECN-Echo bit set in the TCP header. When the TCP sender receives the ECN-Echo dup ACK, it reduces ssthresh and cuts its congestion window in half. When the TCP sender receives the third dup ACK, it retransmits the lost packet and Slow-Starts. However, the TCP sender does not reduce ssthresh a second time.

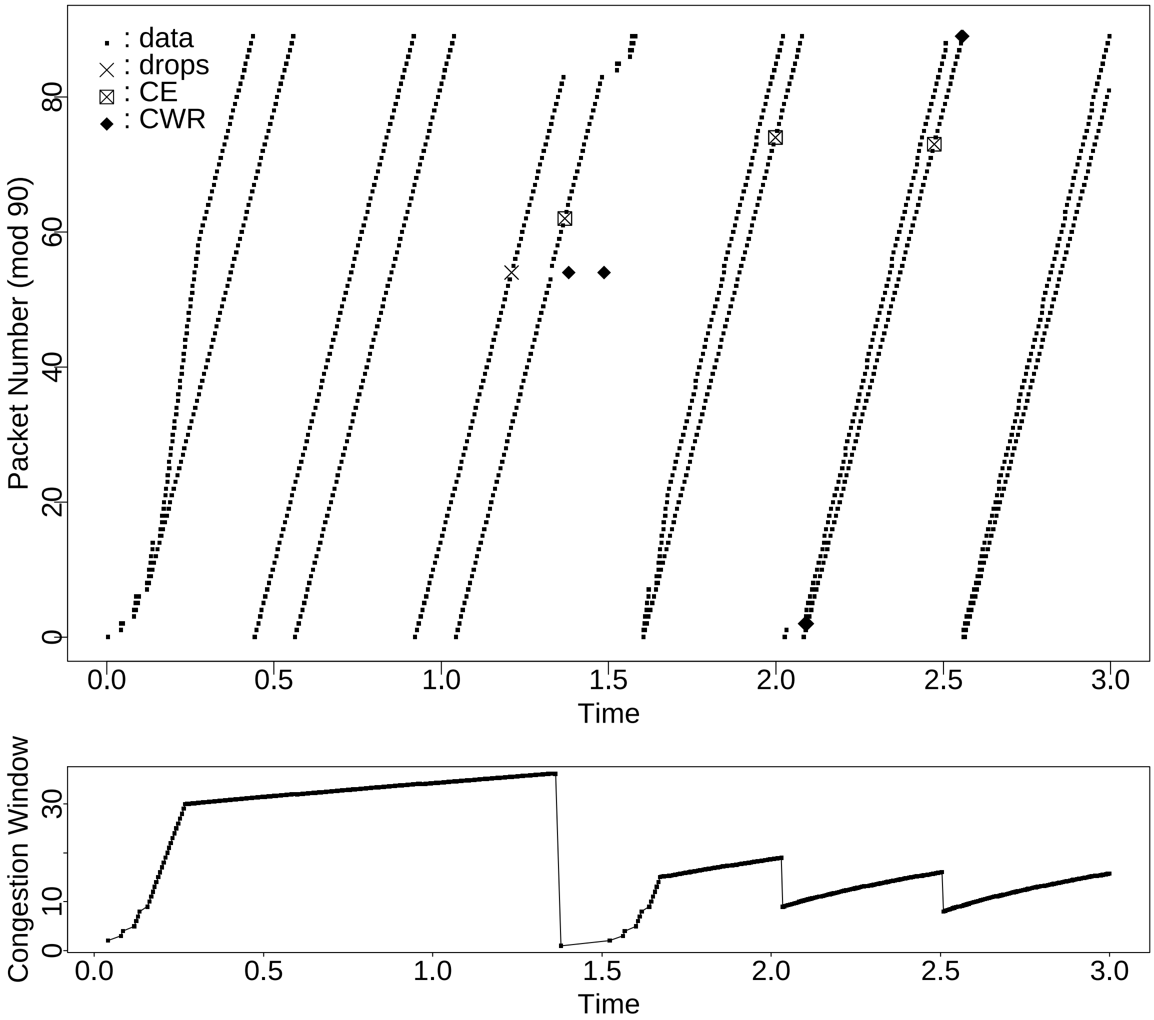

Figure 6: The test "ecn_drop2_tahoe".

This test shows that even when the dropped packet and the CE packet are separated by a number of intervening packets, Tahoe TCP considers the dropped packet and the following CE packet as a single instance of congestion. In this test the TCP sender learns of the CE packet after it has initiated Fast Recovery and Slow-Start.

The dropped packet and the CE packet in this scenario occur within a single window of data.

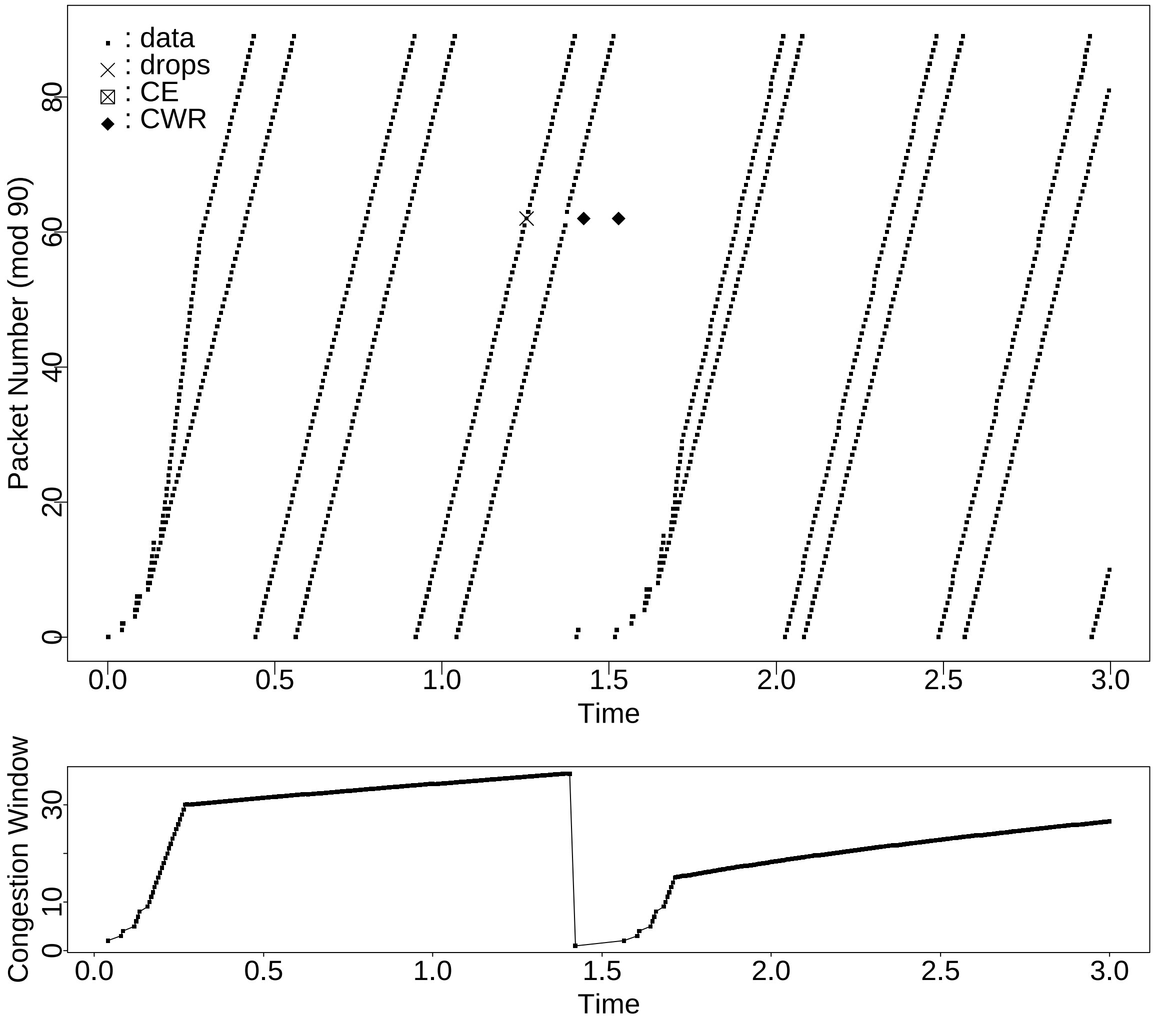

Figure 7: The test "ecn_noecn_tahoe".

This test shows a single dropped packet, without any accompanying CE packet. In this test RED queue management has been disabled, to show the test without any CE packets. Up until time 2.0, Figure 7 shows the same evolution of the congestion window as in Figures 4 through 6.

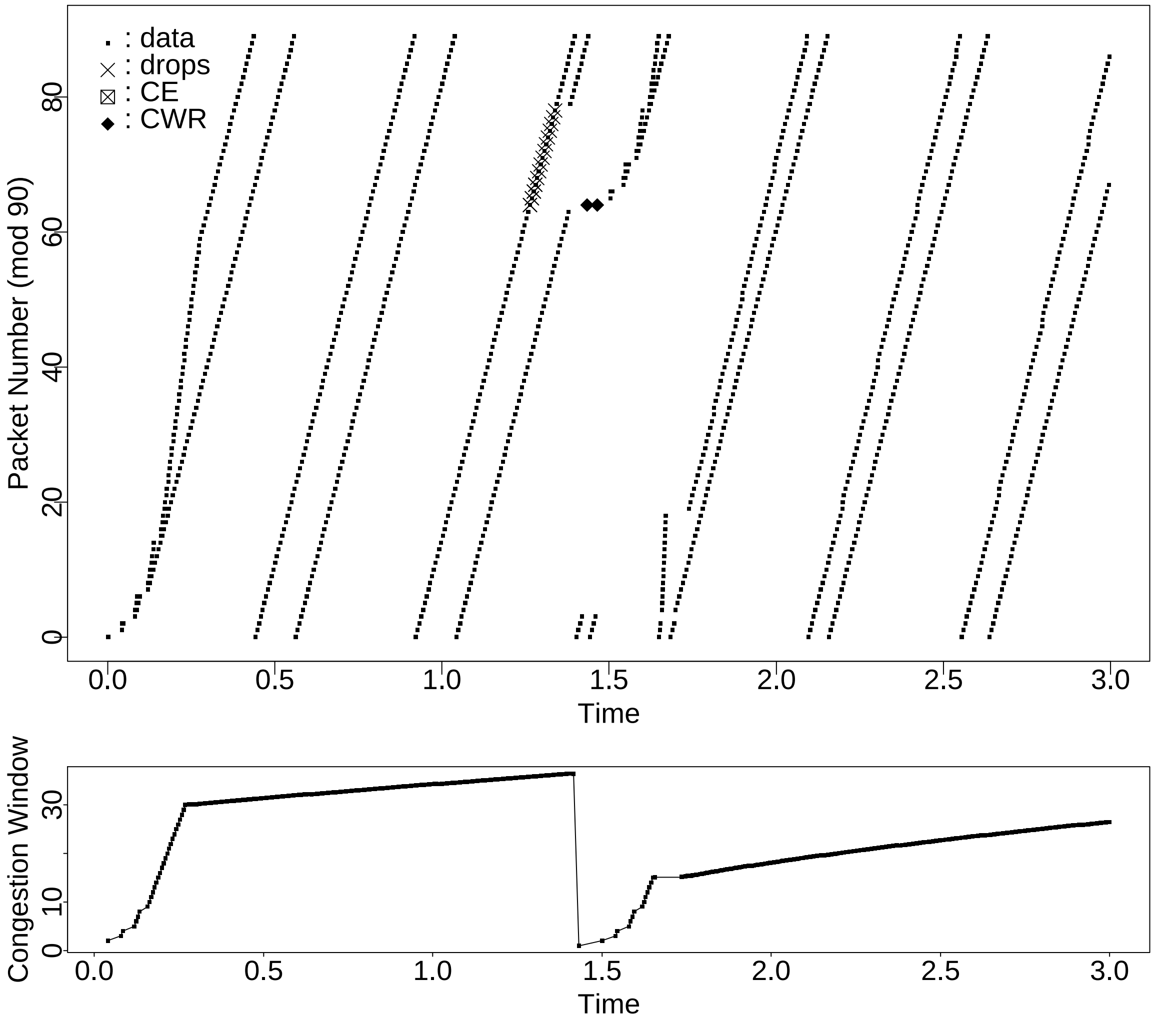

Figure 8: The test "ecn_bursty_tahoe".

Figures 8 and 9 show that interspersed dropped packets and CE packets are treated as a single instance of congestion. Again, while the behavior in these simulations is specific to Tahoe TCP, similar simulations in the test suite show that the same property holds for the Reno, NewReno, and Sack TCP implementations.

This test shows a burst of packet drops with no CE packets. The TCP sender recovers with a Fast Retransmit after receiving three dup ACKs. The Tahoe TCP sender reduces ssthresh and sets the congestion window to the default initial value (in this case, one packet) to enter Slow-Start. The burst of back-to-back packets sent at time 1.6, when the sender receives an ACK packet sharply advancing the cumulative acknowledgement number, is allowed because the experimental "maxburst" parameter to be turned off by default in NS.

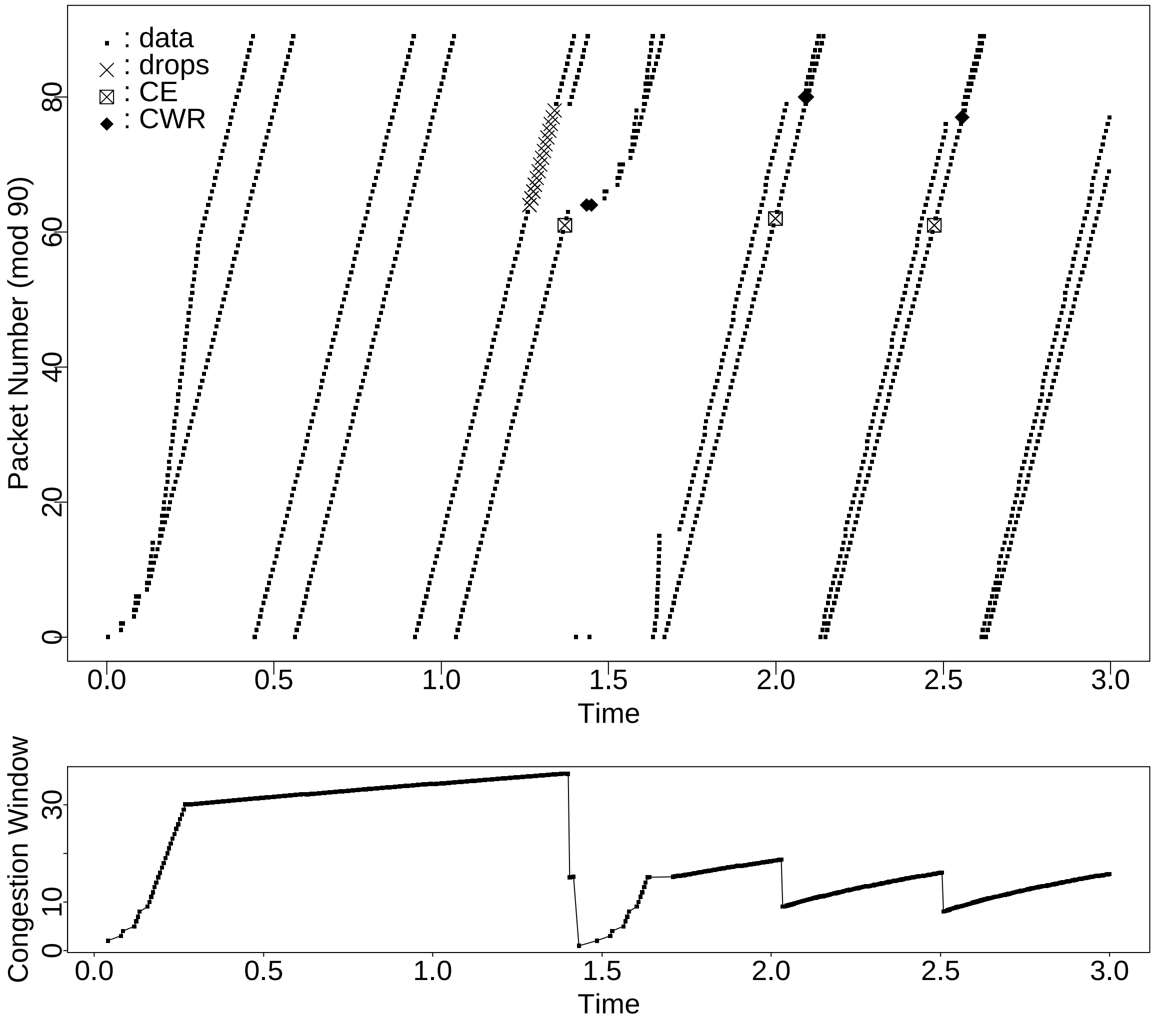

Figure 9: The test "ecn_burstyEcn_tahoe".

This test shows a CE packet followed by the same burst of packet drops as in Figure 8. When the TCP sender receives the ECN-Echo ACK acknowledging the data in the CE packet, it resets ssthresh to half of the old congestion (or to half of the receiver's advertised window, whichever is smaller), and cuts its congestion window in half. Therefore, when the TCP sender receives three dup ACKS, it does not reduce ssthresh again, but only resets the congestion window to one to initiate Slow-Start.

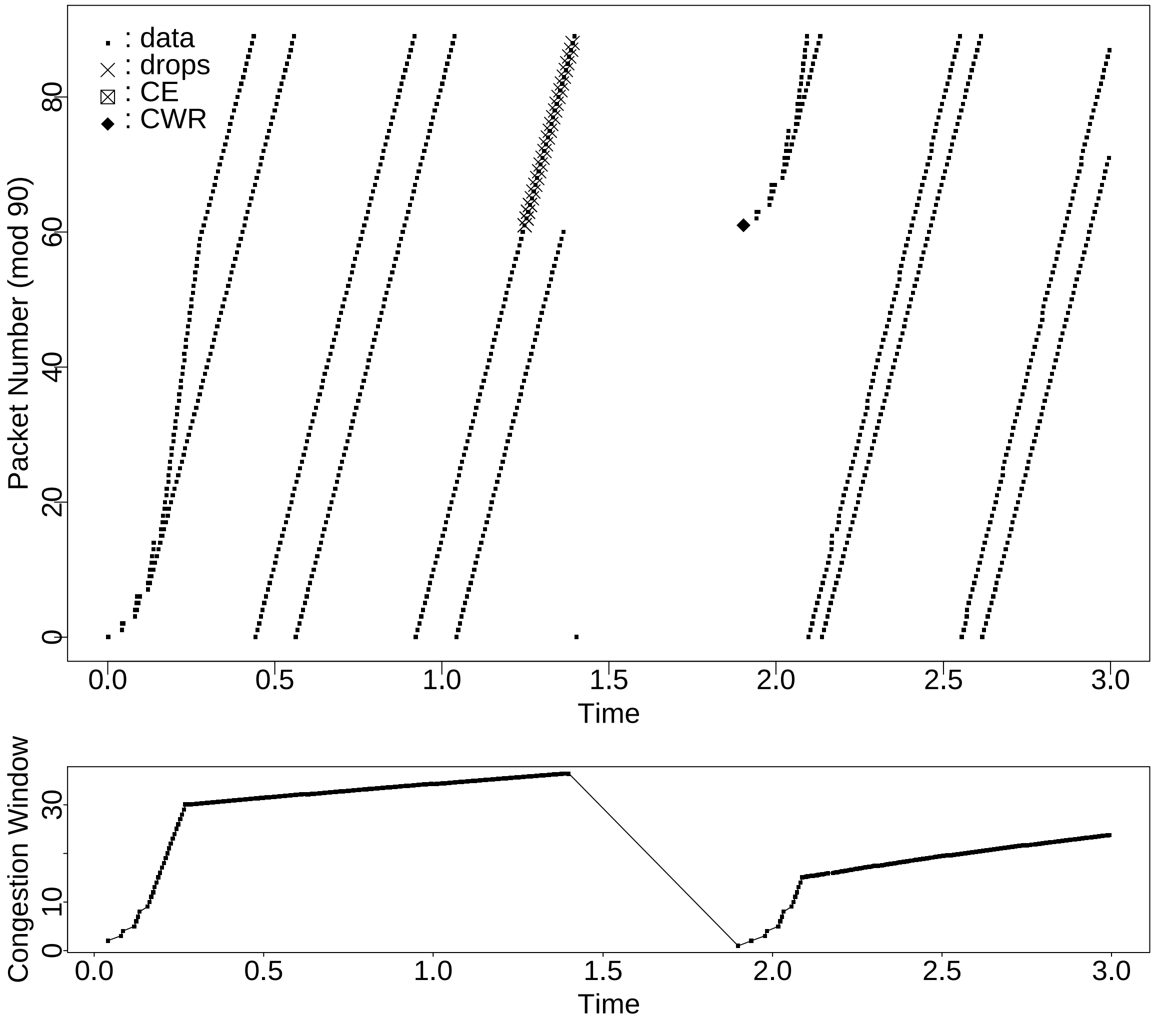

Figure 10: The test “ecn_timeout2_tahoe”.

7 Simulations with retreansmit timeouts

This test shows a large burst of packets being dropped, so that the TCP sender is forced to wait for a retransmit timeout.

Tests 10, 11, and 12 use Tahoe TCP, but in each case tests with Reno, NewReno, and Sack TCP show identical behavior.

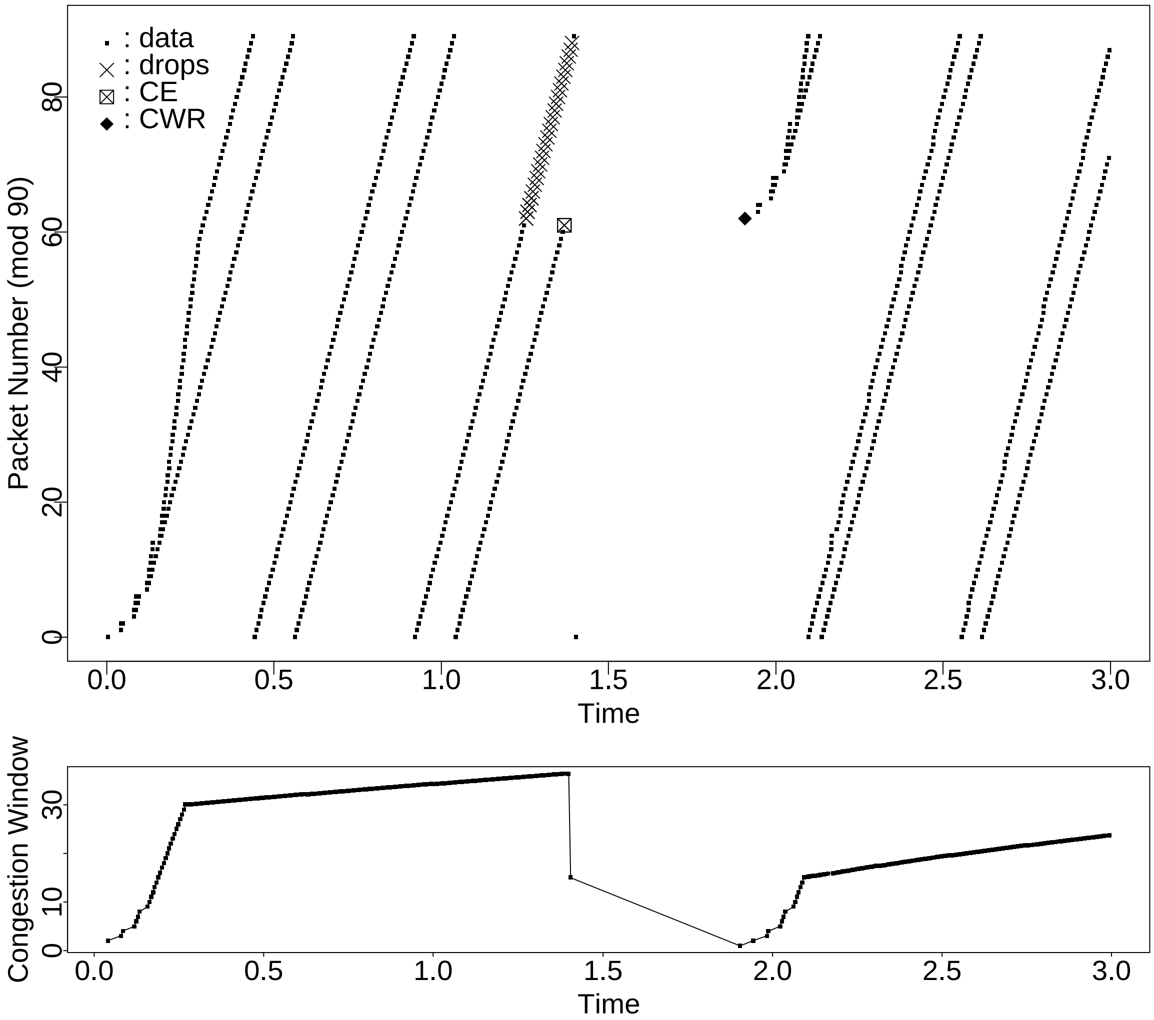

Figure 11: The test "ecn_timeout_tahoe".

This test is similar to Figure 10 except that a CE packet immediately precedes the burst of packet drops. The congestion window has a slightly different evolution from Figure 10 between times 1.4 and 1.9, but the two figures show the same evolution of the congestion window after time 1.9.

In this simulation, in response to an ACK packet with the ECN-Echo bit set that acknowledges the CE packet, TCP reduces ssthresh and reduces its congestion window by half. After the retransmit timeout, instead of reducing ssthresh again, the TCP sender sets ssthresh to the current congestion window. Thus, TCP is not penalized twice for the CE packet and the retransmit timeout.

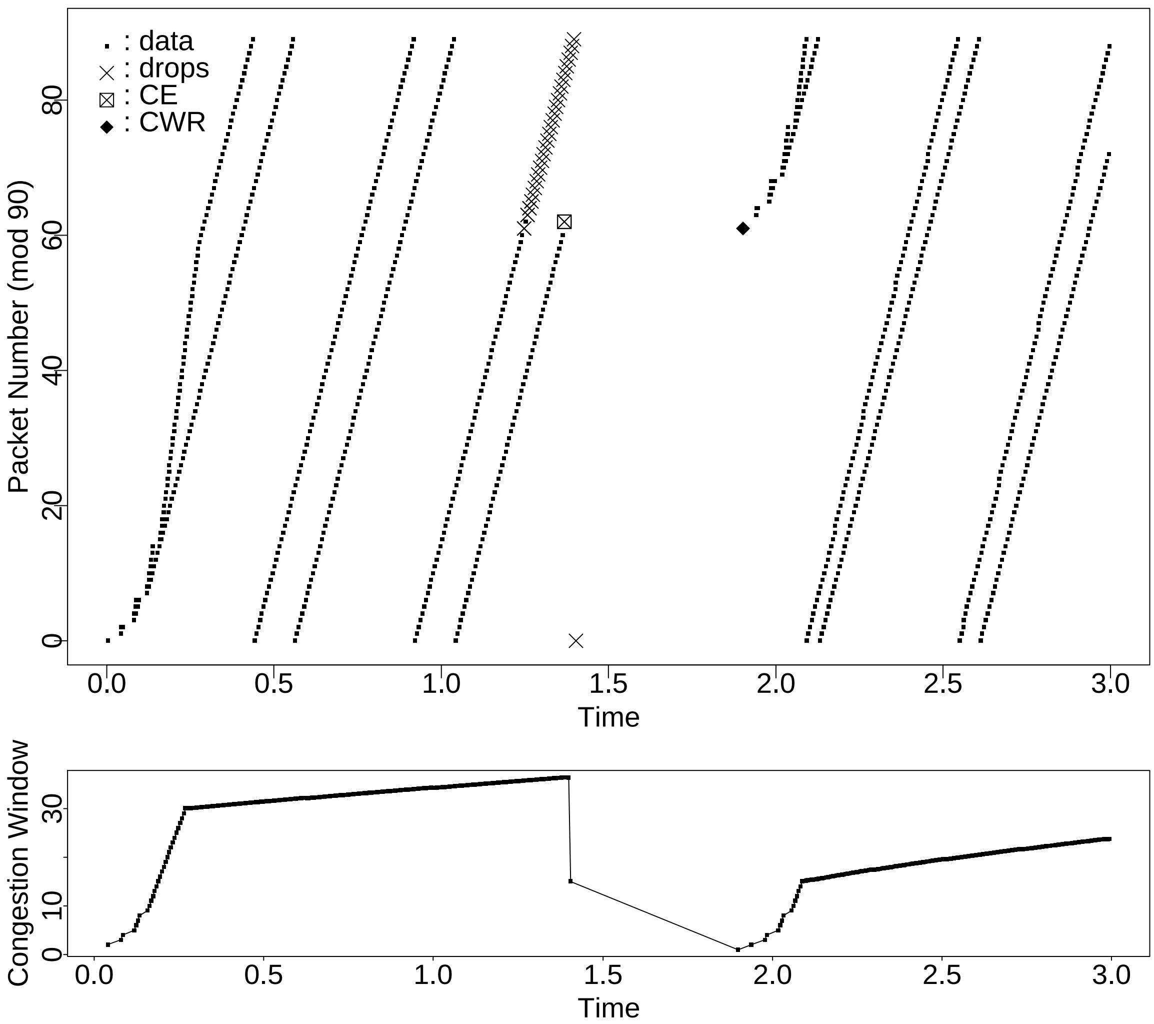

Figure 12: The test “ecn_timeout3_tahoe”.

This test is similar to Figure 10 except that a CE packet is interspersed among the burst of packet drops. Note that the congestion window has the same evolution as in Figure 11.

In this simulation, TCP reduces its congestion window by half in response to a duplicate ACK packet with the ECN-Echo bit set, sent in response to the CE packet. After the retransmit timeout, instead of setting the variable *ssthresh* to half the congestion window, the TCP sender sets ssthresh to the current congestion window.

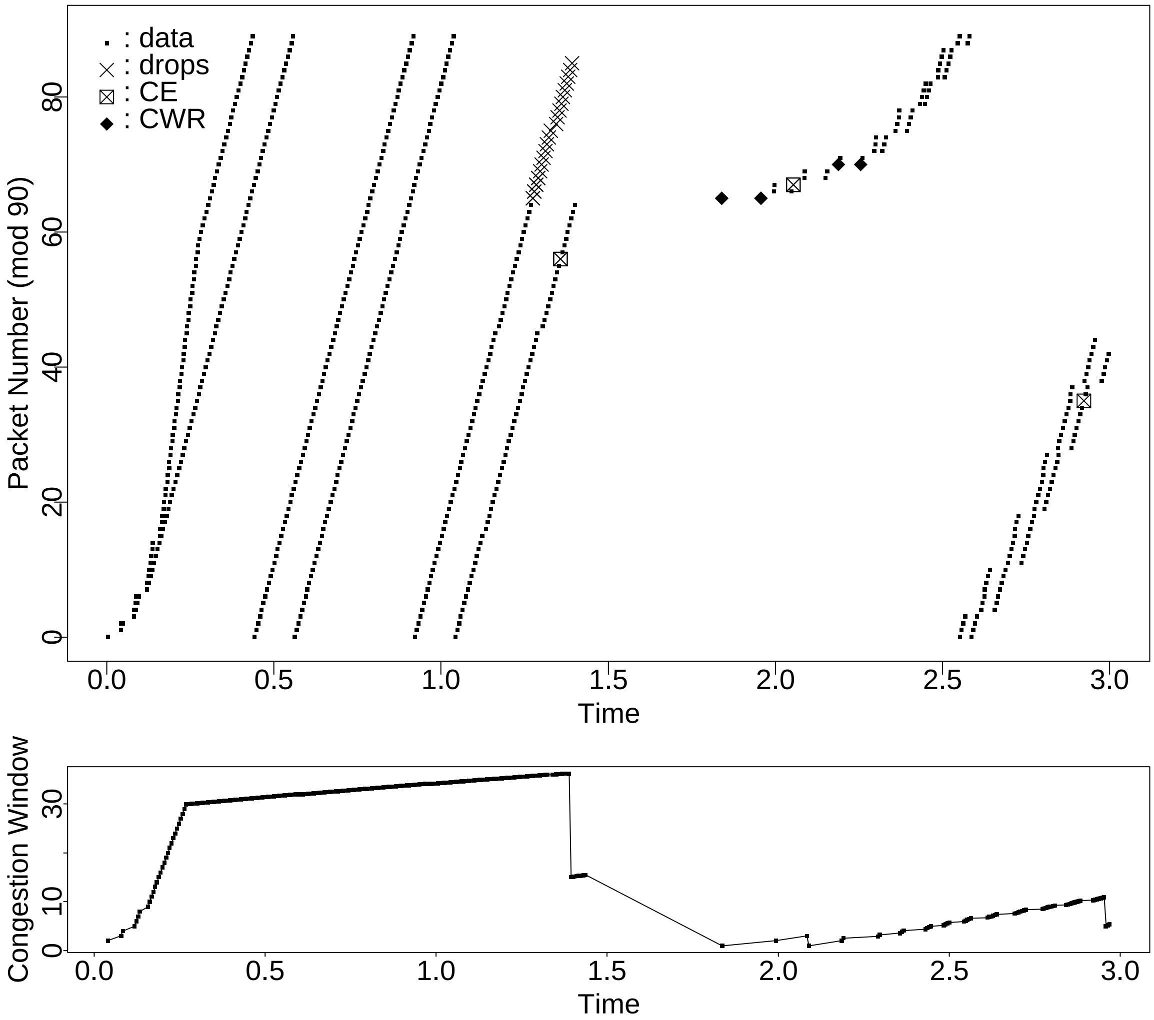

Figure 13: The test "ecn_timeout1_reno.

This test shows that a retransmitted packet with its CE bit set is interpreted by the TCP sender as a new instance of congestion. This test uses Reno TCP.

This test contains a second TCP connection whose packets are not shown in this figure. Because of congestion caused by the second TCP connection, a retransmitted packet from this TCP connection has its CE bit set at time 2.0.

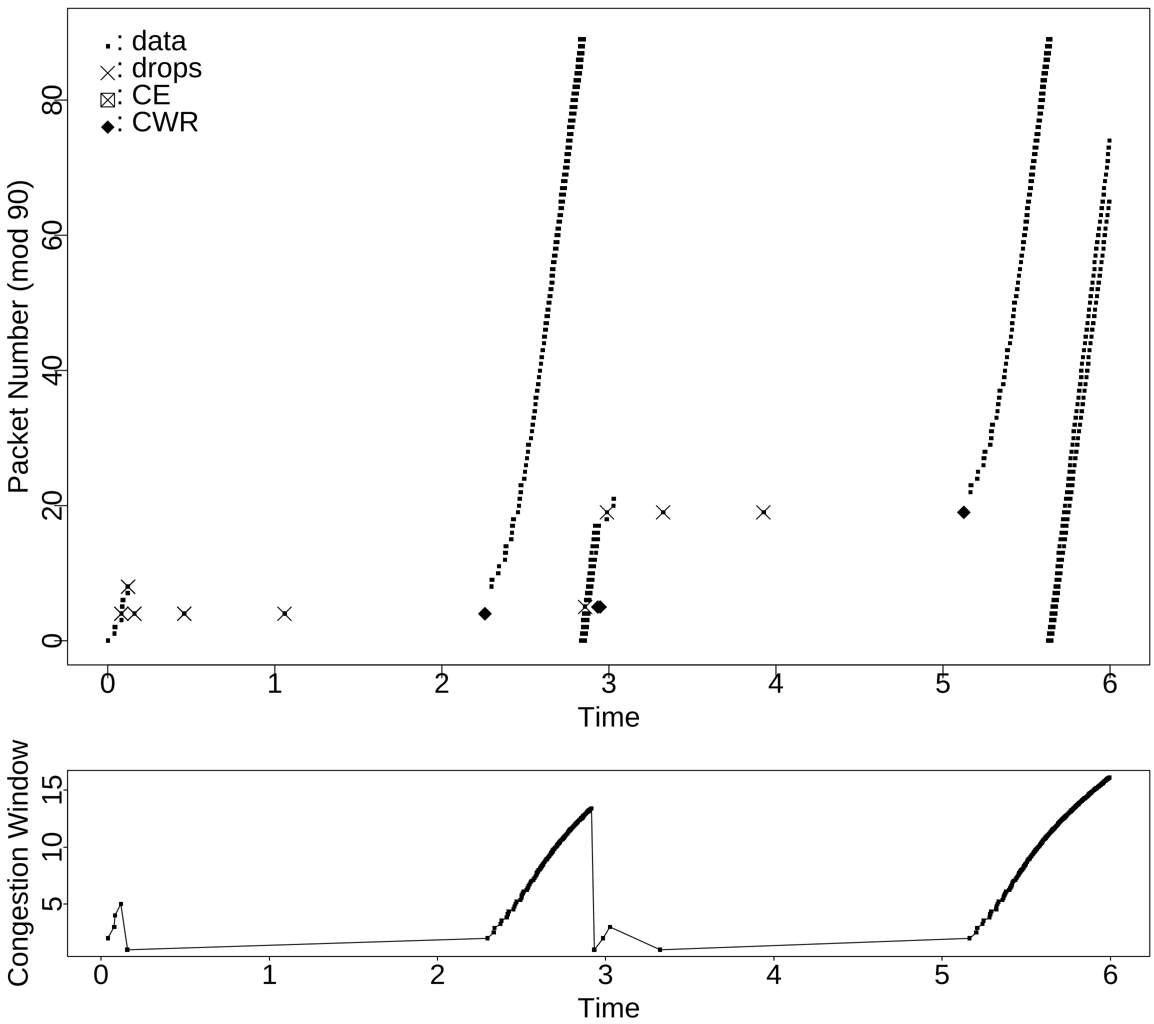

Figure 14: The test “ecn_smallwin_tahoe.

8 ECN with small congestion windows

This test shows Tahoe TCP when successive packets are dropped while the congestion window equals one packet. For the first such drop, a retransmit timeout is required for recovery (because the sender does not receive three duplicate acknowledgements to initiate Fast Retransmit). In successive drops, the retransmit timeout is backed-off, using exponential backoff. This exponential backoff of the retransmit timer is an essential component of TCP's congestion control procedures, allowing robust operation in environments where the available bandwidth is less than one packet per roundtrip time.

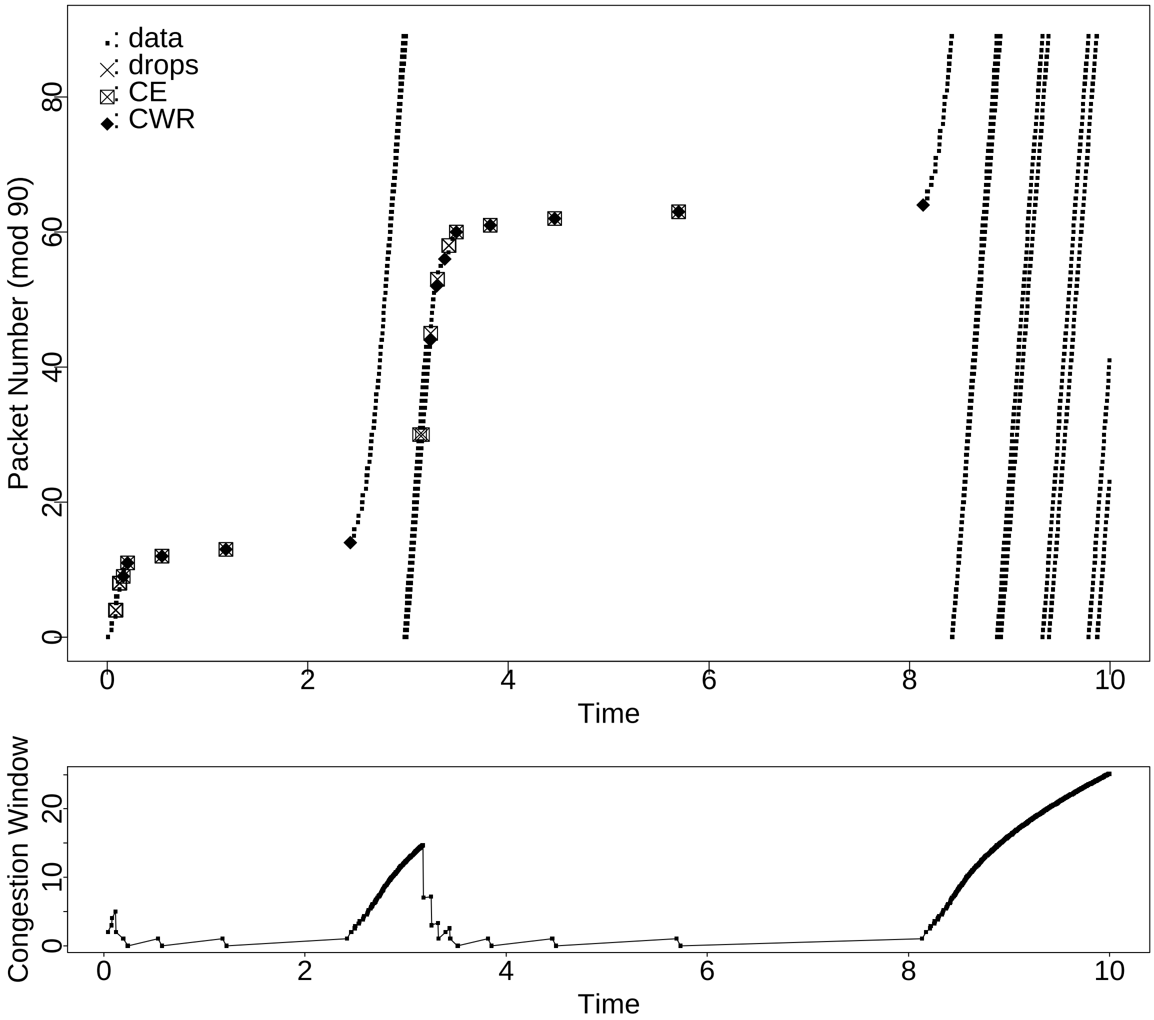

Figure 15: The test "ecn_smallwinEcn_tahoe.

This test is similar to that in Figure 14, except that no packets are actually dropped. Successive packets have the CE bit set while the congestion window equals only one MSS-sized packet. TCP's congestion window is bounded below by one MSS. When the TCP sender receives an ECN-Echo packet and the congestion window is already only one MSS-sized packet, the TCP sender refrains from sending a new packet, and sets the retransmit timer. This figure shows the exponential backoff of the retransmit timer with successive CE packets.

An alternative implementation would be simply to allow the congestion window to be smaller than one MSS and to send smaller packets, and not to set the retransmit timer unless the congestion window is small relative to the packet header size. While this approach merits investigation, we do not recommend it at this time, because this would allow ECN-Capable TCP to be substantially more aggressive than non-ECN TCP during times of high congestion.

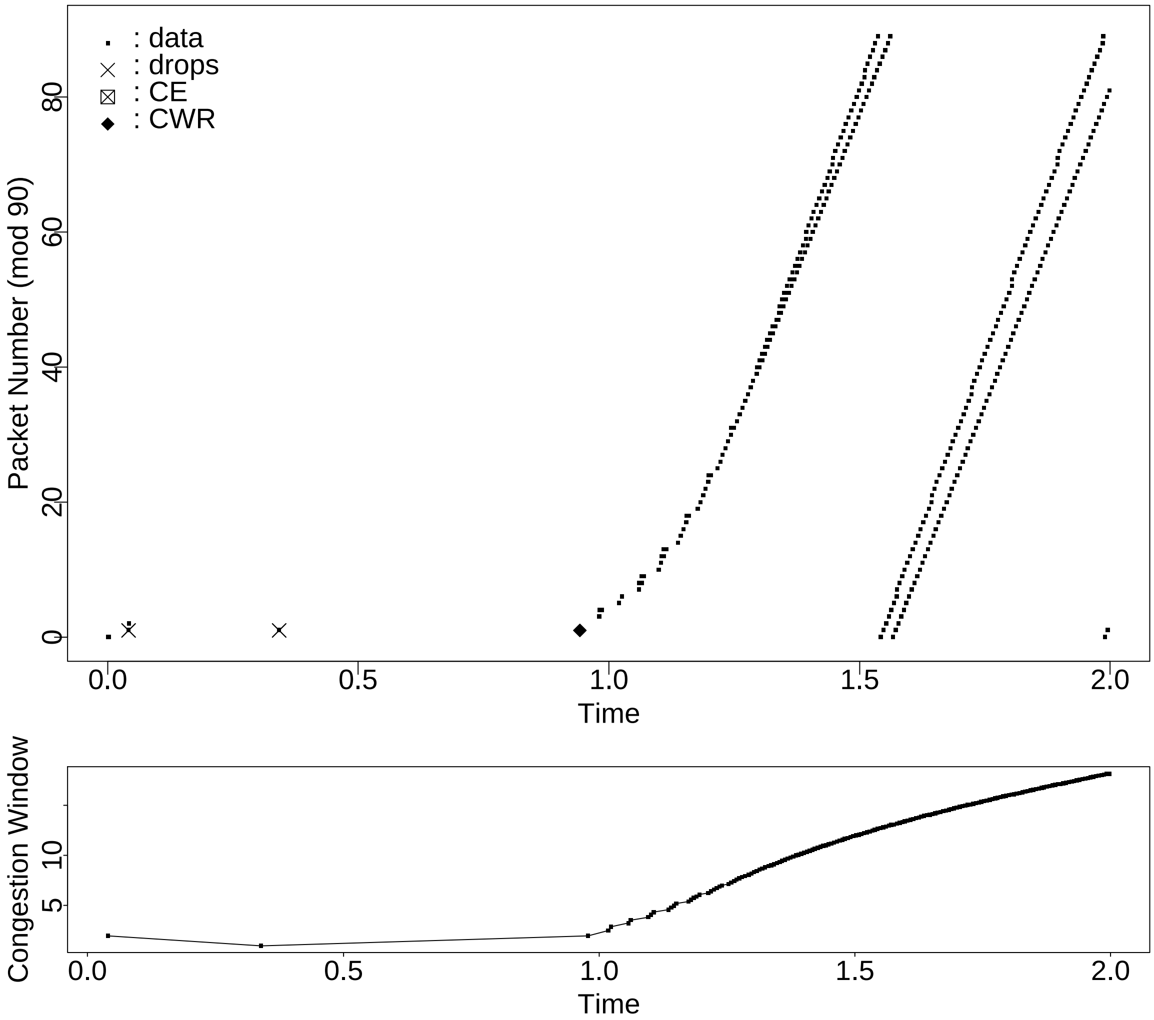

Figure 16: The test “ecn_secondpkt_tahoe.

If the first packet (i.e., the SYN or SYN/ack packet) of a TCP connection was dropped, the TCP connection would not yet have an estimate of the roundtrip time, and a (large) default value would be used for the retransmit timer. This test shows the second and fourth packets dropped from a TCP connection. Each packet drop is accompanied by a retransmit timeout.

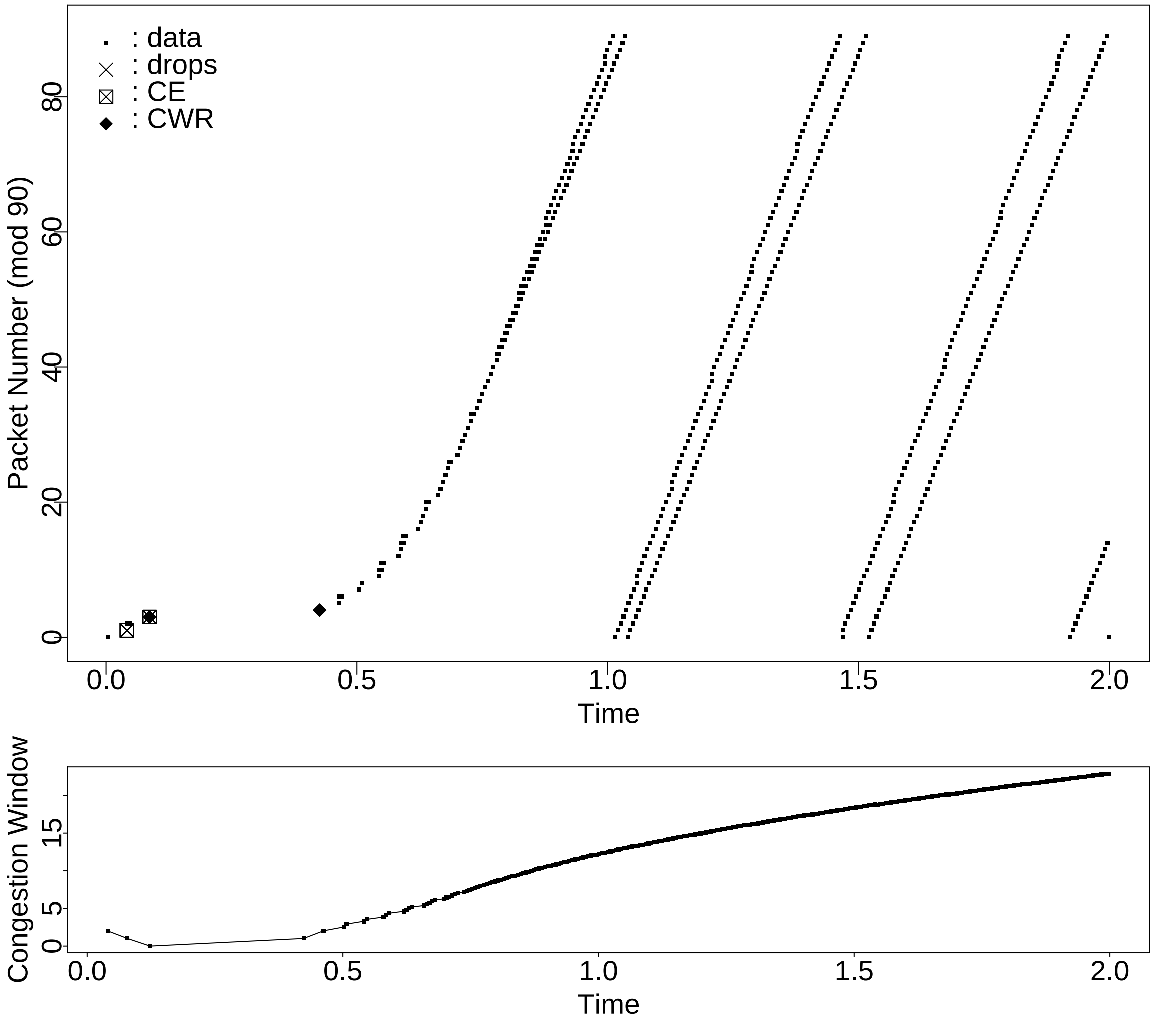

Figure 17: The test "ecn_secondpktEcn_tahoe.

In contrast to Figure 16, this test shows a TCP connection when the second and fourth packets have the CE bit set. After the first CE packet, there is no retransmit timeout, and the sender simply reduces the congestion window from two to one packets. Therefore, when congestion is encountered while the congestion window is small, an ECN-Capable TCP connection can have a significant advantage over a non-ECN-Capable connection.

If the router decides to indicate congestion by dropping or marking a packet that is the first packet (i.e., the SYN or SYN/ack packet) of the TCP connection, the TCP end-nodes would not yet have negotiated ECN-Capability, and that packet would not have the ECT bit set. Therefore, the router would have to drop rather than mark the packet, and the TCP sender would have to wait for the retransmit timer to expire. Thus, ECN-capability does not help response times for congestion encountered by the first packet of a TCP connection.

9 Conclusions

This note illustrates TCP's detailed response to ECN in the NS simulator. The response of TCP to ECN is described more generally in [ECN] and [Flo94].

References

[ECN] "The ECN Web Page,". URL http://www-nrg.ee.lbl.gov/floyd/ecn.html.

[Flo94] S. Floyd. "TCP and Explicit Congestion Notification,". *ACM Computer Communication Review*, 24(5):10–23, Oct. 1994.

[NS95] "NS (Network Simulator),", 1995. URL http://www-mash.cs.berkeley.edu/ns/.