Ns Simulator Tests for Random Early Detection (RED) Queue Management

Sally Floyd and Kevin Fall*

Lawrence Berkeley Laboratory
One Cyclotron Road, Berkeley, CA 94704
floyd@ee.lbl.gov

April 29, 1997

1 Introduction

This note shows some of the tests that I use to verify that the Random Early Detection gateway implementation in our simulator is performing the way that we intend it to perform.[1]

The input files in this document are in the format used by our old simulator, tcpsim. All of these tests can be run in our new simulator ns with the command "test-all-red", and the input files are available in "test-suite-red.tcl".

On each page, the graph shows the results of the simulation. For each graph, the x-axis shows the time in seconds. The y-axis shows the packet number mod 90. There is a mark 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. Some of the graphs show more that one active connection. In this case, packets numbered 1 to 90 on the y-axis belong to the first connection, packets numbered 101 to 190 on the y-axis belong to the second connection, and so on.

Below the graph is the input file for the simulator. The first part of the input file gives the simulator parameters that differ from the default parameters given in the standard input file. The second part of the input file defines the simulation network. The input file shows the edges in the network, with the queue parameters for the output buffers for the forward and/or backward direction for each link (if the output buffer does not use the default queue, which is an unbounded queue). Following the convention in our new simulator, the output buffer size includes a buffer for the packet currently being transmitted on the output link.

The third part of the file contains a line for each active connection, specifying the application (e.g., ftp, telnet), the transport protocol, and the variant of the transport protocol used by the receiver. Below the input file is a brief description of the behavior demonstrated in the test. Included are the commands for running this test in our network simulator *ns*. For more information about the simulator, see the ns web page [Ns], or the main document about simulator tests [F96].

2 Details of the RED simulations

The RED simulations in this section are run with the minimum threshold "minthresh" set to 5, the maximum threshold "maxthresh" set to 15, and the queue weight "q_weight" set to 0.002, as in the simulations in [FJ93].

The simulations in this section are also run with `linterm" set to 10, giving max_p set to 0.1. For the simulations in [FJ93], max_p, the maximum value for the packet dropping probability, is set to 0.02. This is equivalent to setting the parameter "linterm" to 50 in the simulator.

For most of the simulations in this note, the parameter "wait" is set to "true", resulting in a minimum spacing between packet drops that is a function of the drop parameter. For the simulations in [FJ93], the parameter "wait" is set to "false", giving the packet dropping algorithm described as "Method 2" in Section VII of [FJ93]. Simulations with "wait" set to "false" and "linterm" set to "50" should give similar results to simulations with "wait" set to "true" and "linterm" set to "25".

3 Acknowledgements

Our old simulator tcpsim is 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 new simulator ns [Ns], written largely by Steve McCanne, this has been rewritten embedded into Tcl, with the simulation engine implemented in C++.

*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.

[1] This is an updated version of a document that first appeared in October 1996.

References

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

[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.

[F94a] Floyd, S., "TCP and Explicit Congestion Notification", *ACM Computer Communication Review*, V. 24 N. 5, October 1994, p. 10-23. Available via http://www-nrg.ee.lbl.gov/nrg/.

[FJ93] Floyd, S., and Jacobson, V., *Random Early Detection Gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, p. 397-413. Available via http://www-nrg.ee.lbl.gov/nrg/.

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

4 RED (Random Early Detection) Gateways

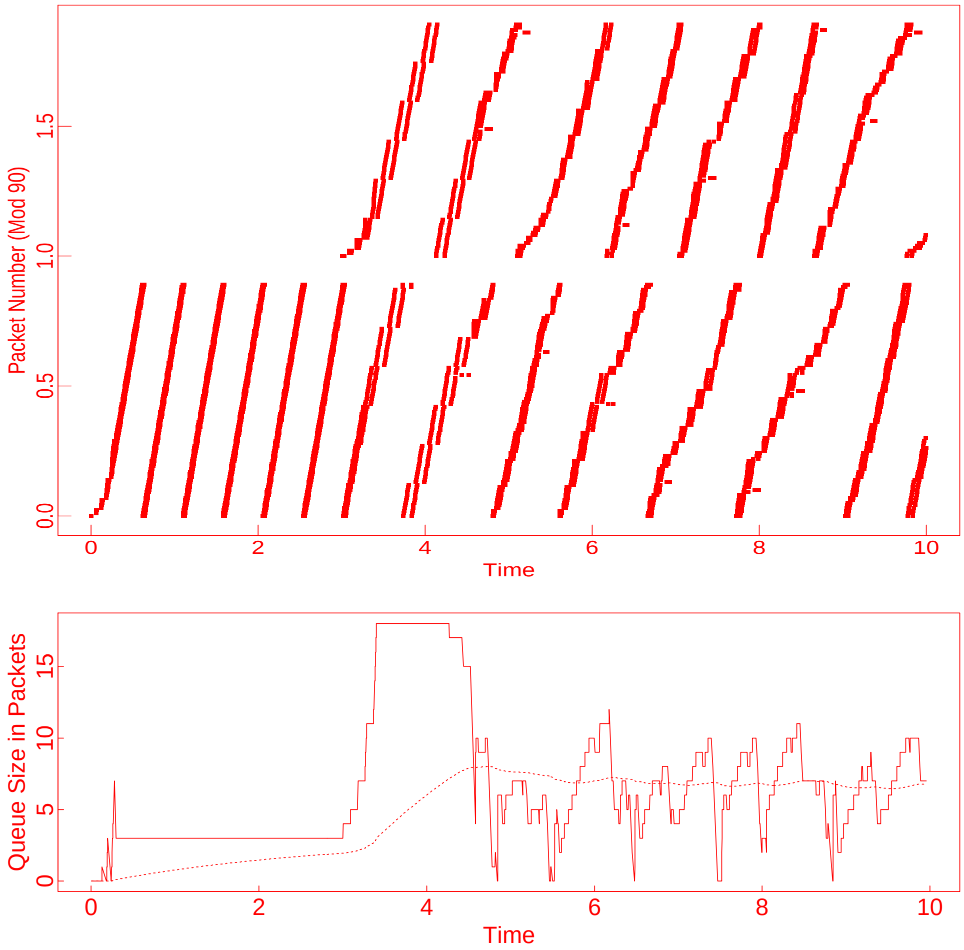

Figure 1: RED gateways.

```
renotcp [ window=15 ]
red [ min_thresh=5 max_thresh=15 q_weight=0.002 max_p=0.02 ]
edge s1 to r1 bandwidth 10Mb delay 2ms
edge s2 to r1 bandwidth 10Mb delay 3ms
edge r1 to r2 bandwidth 1.5Mb  delay 20ms
    forward [ queue-type=red queue-size=25 ]
    backward [ queue-type=red queue-size=25 ]
edge s3 to r2 bandwidth 10Mb delay 4ms
edge s4 to r2 bandwidth 10Mb delay 5ms
ftp conv from renotcp at s1 to sink at s3
ftp [start-at=3.0] conv from renotcp at s2 to sink at s3
```

Figure 1 shows a RED (Random Early Detection) gateway [FJ93]. The different parameters are discussed in [FJ93]. Because RED gateways use a randomized algorithm for determining which arriving packets to drop at the gateway, the results of the simulation will vary with different seeds for the pseudo-random number generator.

In the bottom graph, the solid line shows the queue (in packets) of packets waiting to be transmitted on the link from “r1” to “r2”. The dotted line shows the average queue size as calculated by the RED gateway. Note that in this simulation, no packets are ever dropped because of buffer overflow. This test can be run on ns-1 with the following command:

```
ns test-suite-red.tcl red1
```

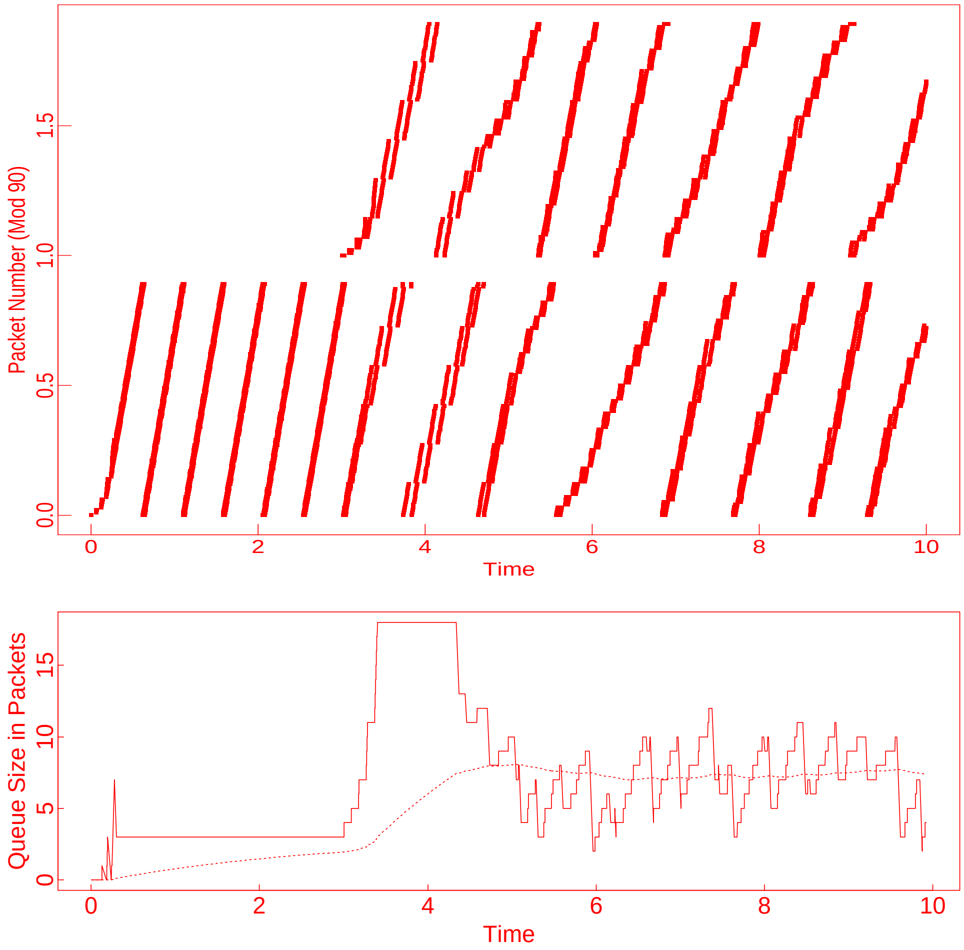

Figure 2: RED gateways with Explicit Congestion Notification.

```
renotcp [ window=15 ]
red [ min_thresh=5 max_thresh=15 q_weight=0.002 max_p=0.02 setbit=true ]
edge s1 to r1 bandwidth 10Mb delay 2ms
edge s2 to r1 bandwidth 10Mb delay 3ms
edge r1 to r2 bandwidth 1.5Mb  delay 20ms
    forward [ queue-type=red queue-size=25 ]
    backward [ queue-type=red queue-size=25 ]
edge s3 to r2 bandwidth 10Mb delay 4ms
edge s4 to r2 bandwidth 10Mb delay 5ms
ftp conv from renotcp [ecn=1] at s1 to sink at s3
ftp [start-at=3.0] conv from renotcp [ecn=1] at s2 to sink at s3
```

Figure 2 shows a simulation identical to that in Figure 1, except that the RED gateway and the TCP sources are using Explicit Congestion Notification [F94a]. That is, unless forced to drop a packet by a queue overflow, the RED gateway sets an ECN bit in packet headers rather than dropping the packet. The TCP source interprets an arriving packet with the ECN bit set as an indication of congestion. This test can be run on ns-1 with the following command:

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

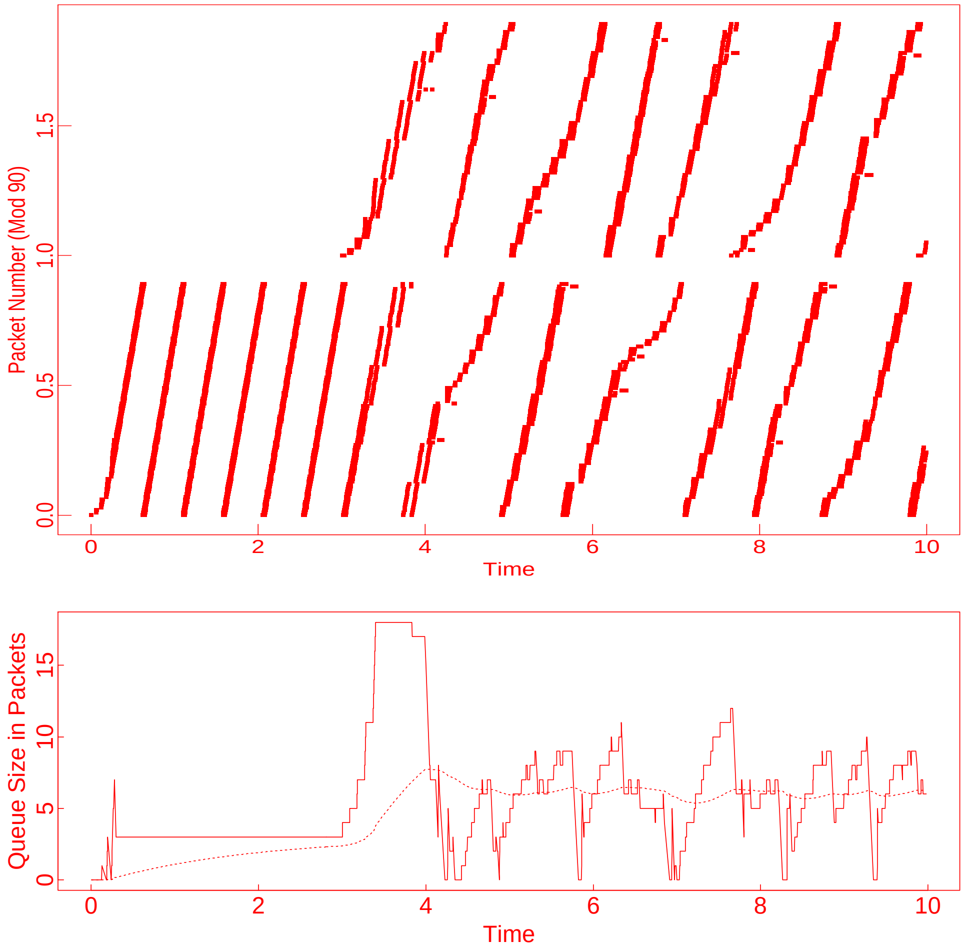

Figure 3: RED gateways, with a poor setting of parameters.

```
renotcp [ window=15 ]
red [ min_thresh=5 max_thresh=10 q_weight=0.003 max_p=0.02 ]
edge s1 to r1 bandwidth 10Mb delay 2ms
edge s2 to r1 bandwidth 10Mb delay 3ms
edge r1 to r2 bandwidth 1.5Mb  delay 20ms
    forward [ queue-type=red queue-size=25 ]
    backward [ queue-type=red queue-size=25 ]
edge s3 to r2 bandwidth 10Mb delay 4ms
edge s4 to r2 bandwidth 10Mb delay 5ms
ftp conv from renotcp at s1 to sink at s3
ftp [start-at=3.0] conv from renotcp at s2 to sink at s3
```

Figure 3 shows the same simulation as in Figure 1, but with a different set of parameters for the RED gateway. The parameter "q_weight" is higher, meaning that the calculated queue average puts more weight on current queue measurements. In addition, the parameter "max_thresh" is lower: the RED gateway tries to keep the average queue size between "min_thresh" and "max_thresh" during congestion. With these parameters, the performance of the RED gateway will generally suffer. This test can be run on ns-1 with the following command:

```
ns test-suite-red.tcl red2
```

5 RED gateways with two-way traffic

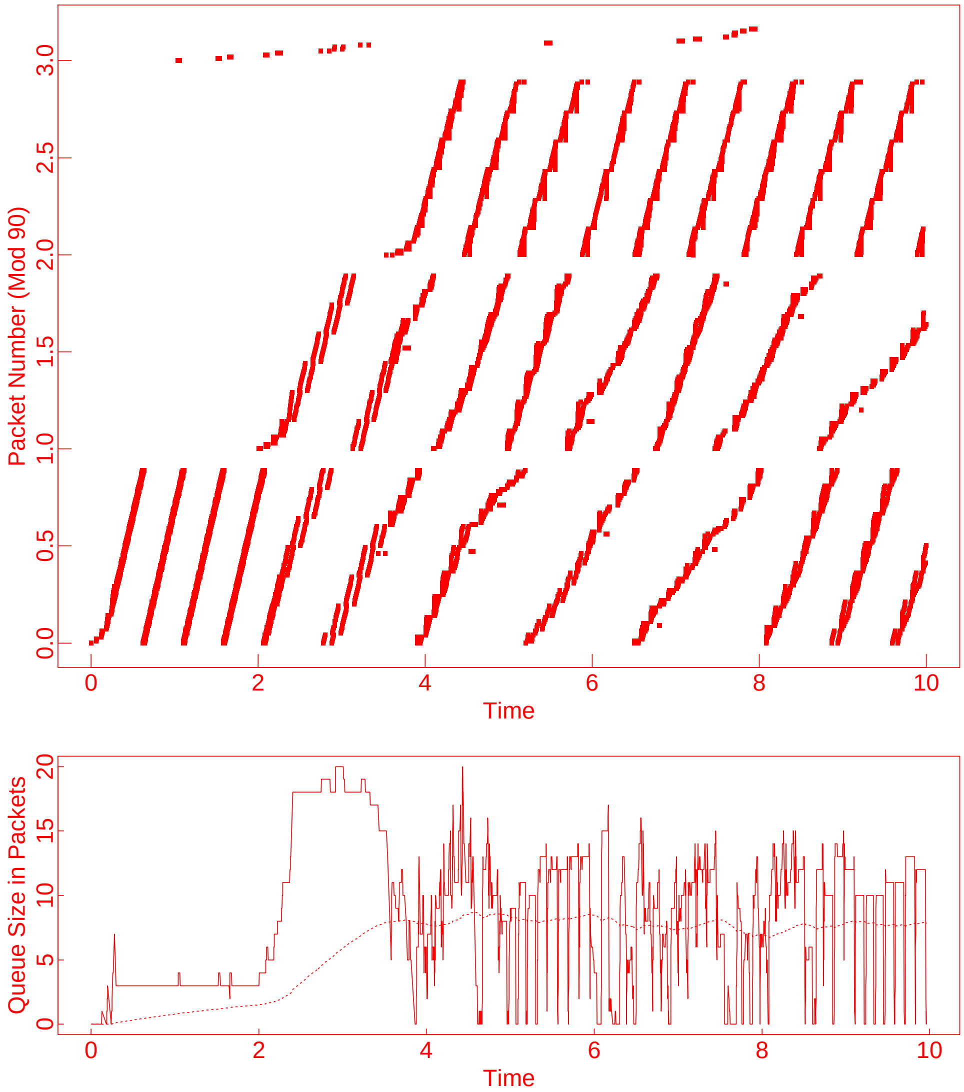

Figure 4: RED gateways, with two-way traffic.

```
[The network topology is the same as in previous simulations.]
ftp conv from renotcp at s1 to sink at s3
ftp [start-at=2.0] conv from renotcp at s2 to sink at s4
ftp [start-at=3.5] conv from renotcp at s3 to sink at s1
telnet [start-at=1.0 interval=0] conv from renotcp at s4 to sink at s2
```

Figure 4 shows a simulation with two-way traffic. The top graph shows packets transmitted on the link from “r1” to “r2”. For the bottom two connections, these packets are data packets. For the top two connections, these packets are returning ACK packets. The figure shows clear ACK-compression.

The narrow spikes in the queue size, which is measured in packets and not in bytes, are caused by bursts of ACK packets in the queue. Similarly, the quick decrease of the queue from 8 to 0 packets is caused when eight small ACK packets leave the queue back-to-back. This test can be run on ns-1 with the following command:

```
ns test-suite-red.tcl red_twoway
```

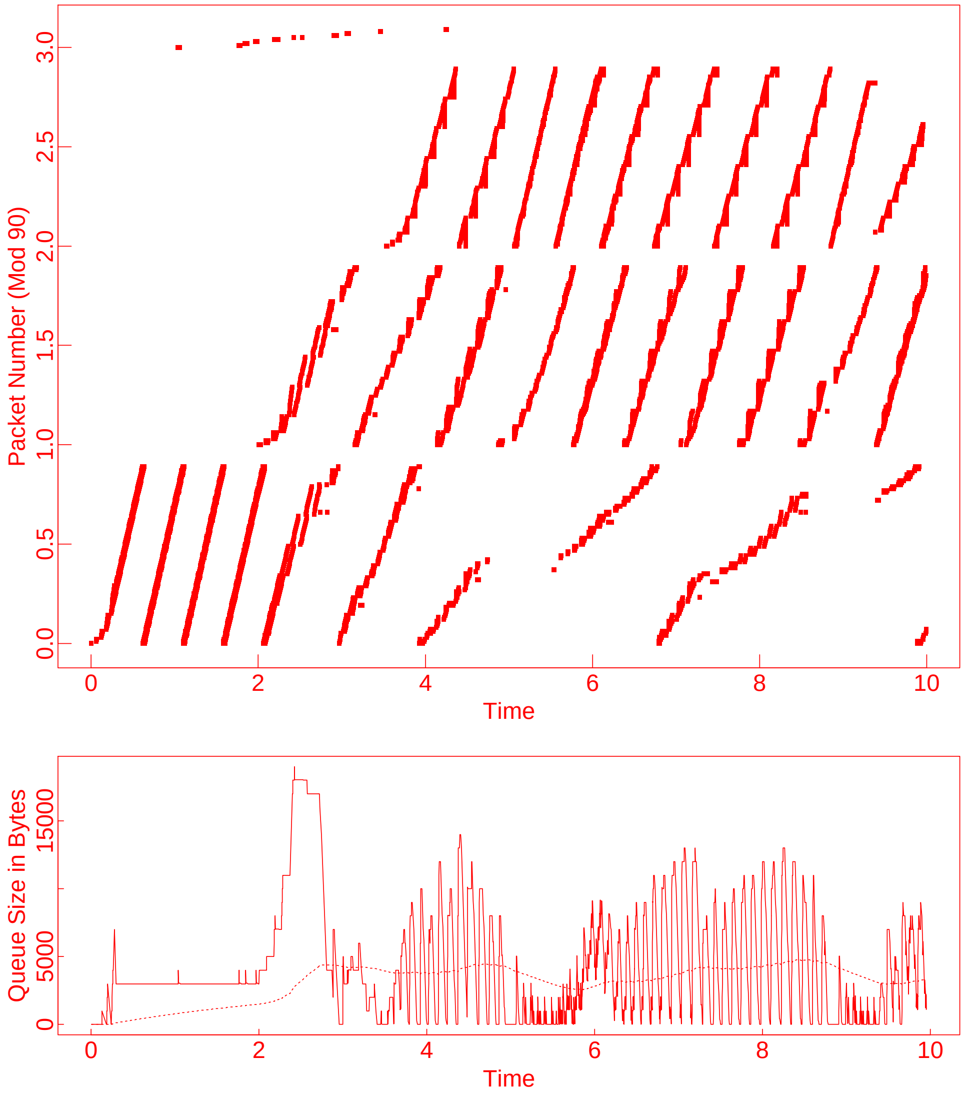

Figure 5: RED gateways, with two-way traffic and a queue measured in bytes rather than in packets.

```
[The network topology is the same as in the previous simulation,
except the queue uses byte-mode rather then packet-mode.]
[The traffic sources are the same as in the previous simulation.]
```

Figure 5 differs from Figure 4 only in that the queue is measured in bytes rather than in packets, and the packet-dropping probability is modified so that larger packets are more likely to be dropped than are smaller packets.

Note that with the packet-dropping probability proportional to the packet size in bytes, ACK packets are unlikely to be dropped. In addition, with the queue measured in bytes, the queue size has smaller swings, because ACK packets have a negligible influence on the queue size.

This test can be run on ns-1 with the following command:

```
ns test-suite-red.tcl red_twowaybytes
```

Setting the parameter "ns_link(queue-in-bytes)" to "true" results in a queue that is measured in bytes rather than in packets. Setting the parameter "ns_red(bytes)" to "true" results in a RED packet-dropping probability that is a function of the packet size in bytes. For this simulation, both "ns_link(queue-in-bytes)" and "ns_red(bytes)" are set to true.

6 Identifying high-bandwidth flows

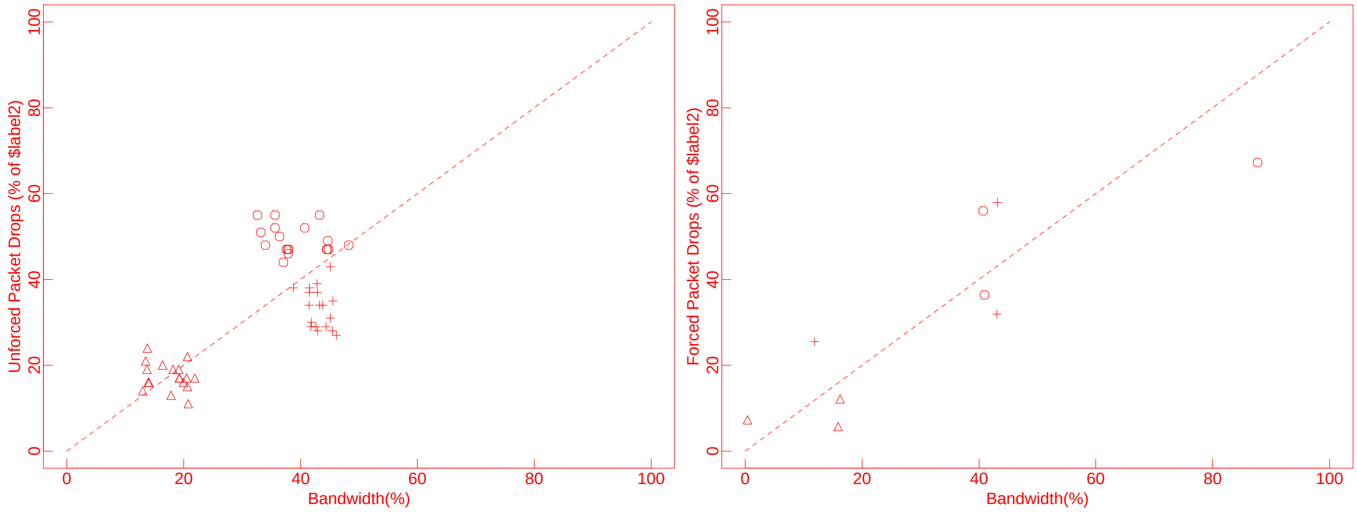

Figure 6: RED gateways, with per-flow information about bandwidth and packet drop rates.

```
renotcp [ window=100 ]
red [ min_thresh=5 max_thresh=15 q_weight=0.002 max_p=0.1 mean_pktsize=1000 ]
edge s1 to r1 bandwidth 10Mb delay 2ms
edge s2 to r1 bandwidth 10Mb delay 3ms
edge r1 to r2 bandwidth 1.5Mb  delay 20ms
    forward [ queue-type=red queue-size=100 bytes=true ]
edge s3 to r2 bandwidth 10Mb delay 4ms
edge s4 to r2 bandwidth 10Mb delay 5ms
ftp [start-at=1.0 pktsize=1000] conv from renotcp at s1 to sink at s3
ftp [start-at=1.2 pktsize=50] conv from renotcp at s2 to sink at s4
cbr [start-at=1.4 interval=0.003 pktsize=190] conv from udp at s1 to sink at s4
```

This simulation shows a congested link with three flows, The first, a TCP flow with 1000-byte packets and a RTT of 52 ms (excluding queueing delay), has an arrival rate just over 40% of the link bandwidth. The second, a TCP flow with 50-byte packets and a RTT of 56 ms, has an arrival rate around 40% of the link bandwidth. The third, a UDP flow with a CBR source that sends 190-byte packets every 3 ms, has an arrival rate just under 40% of the link bandwidth. The relative arrival rates of the three flows can be adjusted by modifying the packet size or roundtrip times for the TCP flows, or by modifying the packet size and arrival rate for the CBR flow.

The left graph in Figure 6 shows a mark for each flow for each set of 100 unforced packet drops, and the right graph in Figure 6 shows a mark for each flow for each set of 100 forced packet drops. We say that a packet drop is *forced* if the packet was dropped because either the buffer overflowed, or the average queue size exceeded the maximum threshold. Otherwise a packet drop is *unforced*.

The first flow is represented by squares, the second flow is represented by triangles, and the third flow is represented by plus marks. For each flow, the x-axis represents a flow's fraction of the aggregate arrival rate in bytes. For the left graph in Figure 6, the y-axis represents a flow's fraction of the packet drops; that is, the number of packets dropped from that flow as a percentage of the total number of packets dropped in that set. The dotted line shows where marks would lie if the fraction of dropped packets exactly equaled that flow's fraction of the arrival rate in bytes. A mark in the upper left quadrant would represent a flow whose fraction of marked packets was much higher than that flow's fraction of the aggregate arrival rate in bytes.

Figure 6 shows that, for flows that use a significant fraction of the link bandwidth, the fraction of unforced packet drops from a flow is a good indicator for that flow's fraction of the arrival rate in bytes. This simulations can be run on ns-1 with the following commands:

```
ns test-suite-red.tcl flows
```

```
ns test-suite-red.tcl flows1
```

The right graph in Figure 6 shows the flow statistics for each set of *forced* packet drops. For forced packet drops, a flow's fraction of packets dropped is not proportional to the flow's fraction of the arrival rate; instead, a flow's fraction of the total *bytes* of dropped packets is roughly proportional to the flow's fraction of the arrival rate. In the right graph of Figure 6, the y-axis shows a flow's byte-count of dropped packets as a percentage of the total byte-count of dropped packets. In some runs of the simulation, Note that for one set of 100 forced packet drops, all of a set of 100 forced packet drops are from the UDP flow. This generally only happens if both TCP flows have just backed off, waiting for retransmit timers to expire.