Ns Version 1 Simulator Tests for Class-Based Queueing

Sally Floyd*

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

April 28, 1997

1 Introduction

This note shows simulation acceptance tests for CBQ (class-based queueing) as implemented in the ns simulator [MF95]. An introduction to CBQ is available in [FJ95].

The simulator implements three separate algorithms for link-sharing described in [FJ95]. These are Formal, Top-level, and Ancestor-Only link-sharing. The simulator implements two different scheduling algorithms within classes of the same priority level, weighted round-robin (WRR) and packet-by-packet round-robin (PRR). The weighted round-robin scheduling algorithm is described in Appendix A of [FJ95].

2 Formal, Top-level, and Ancestor-Only link-sharing

This section shows simulations that use Formal, Top-level, and Ancestor-Only link-sharing. For the scenario in these simulations, Top-level link-sharing performs slightly better than Ancestor-Only link-sharing, and Formal link-sharing performs slightly better than Top-level link-sharing. However, for this scenario all three link-sharing algorithms give reasonable performance.

The simulations in Figures 1 through 4 essentially reproduce the simulations shown in Figure 11-13 in [FJ95]. These tests are run in `ns` with the following respective commands:

```
ns test-suite-cbq.tcl cbqTL
ns test-suite-cbq.tcl cbqAO
ns test-suite-cbq.tcl cbqFor
ns test-suite-cbq.tcl cbqForOld
```

The reader is referred to the file `test-suite-cbq.tcl` for more details of the simulation set-up.

The simulation scenario is given in Figure 6 in [FJ95], and the link-sharing structure for the congested link is given in Figure 8 in [FJ95]. These simulations are discussed further in Section 5.1 of [FJ95].

The simulations with the old version of Formal link-sharing implement an obsolete version of Formal link-sharing that is not discussed in [FJ95].

*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.

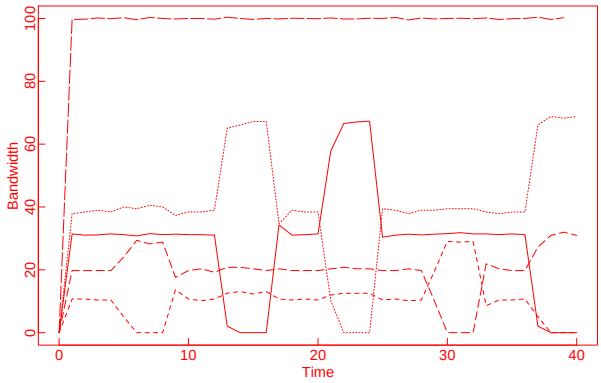

Figure 1: WRR, Top-level link-sharing.

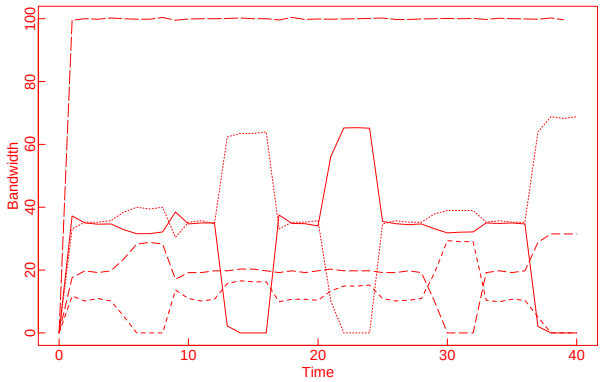

Figure 2: WRR, Ancestor-Only link-sharing.

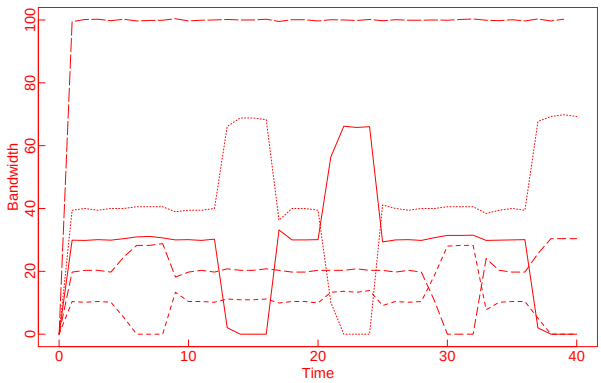

Figure 3: WRR, Formal link-sharing.

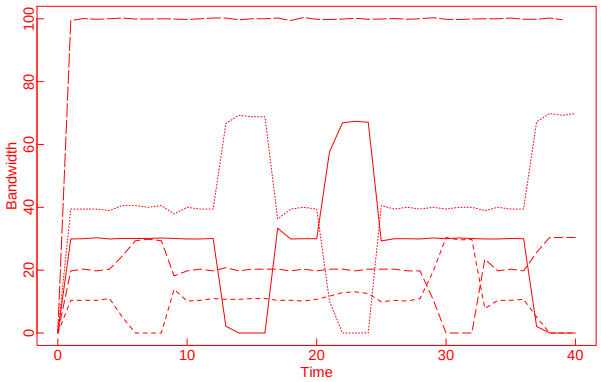

Figure 4: WRR, Formal link-sharing, old version.

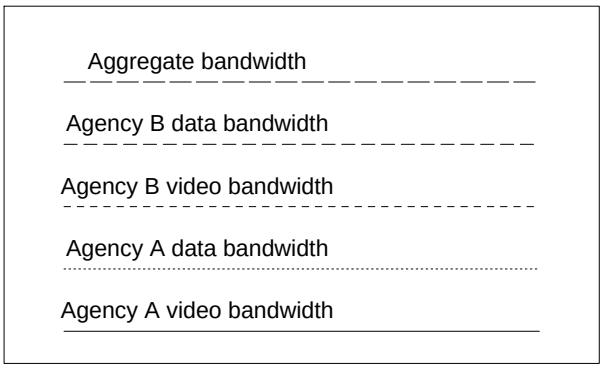

Figure 5: Legend.

2.1 Differences between the three link-sharing algorithms

The simulations in this section are designed to illustrate the possible weaknesses of Ancestor-Only link-sharing.

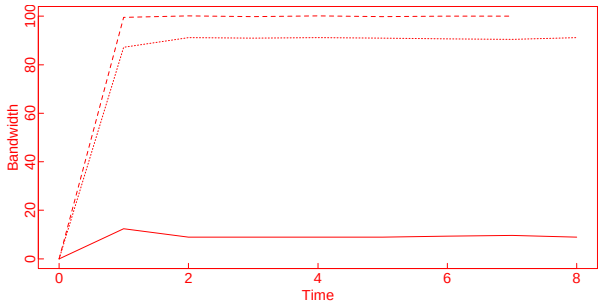

Figure 6: Ancestor-Only link-sharing.

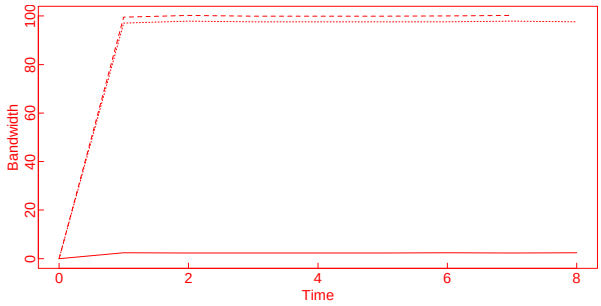

Figure 7: Top-Level link-sharing

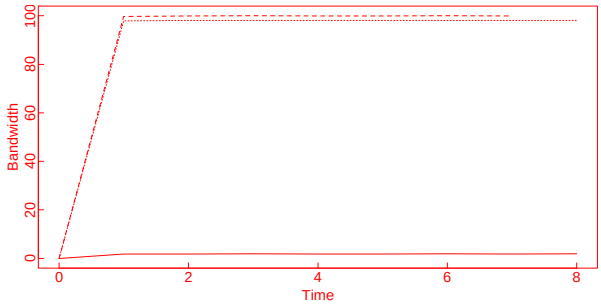

Figure 8: Formal link-sharing

These figures show a link that is shared by two classes, and explore the bandwidth actually received by a priority-one class that is allocated a very small fraction of the link bandwidth. Class A, at priority-one, is allocated 1% of the link bandwidth, and Class B, at priority-two, is allocated 99% of the link bandwidth. The data source for Class A is a CBR flow that sends 190-byte packets every 0.001 seconds. The data source for Class B is a CBR flow that sends 500-byte packets every 0.002 seconds.

Figures 6 through 10 compare Ancestor-Only, Top-Level, and Formal link-sharing. With both Top-Level and Formal link-sharing, the higher-priority class is properly restricted to a small fraction of the link bandwidth. In contrast, with Ancestor-Only link-sharing the higher-priority class receives

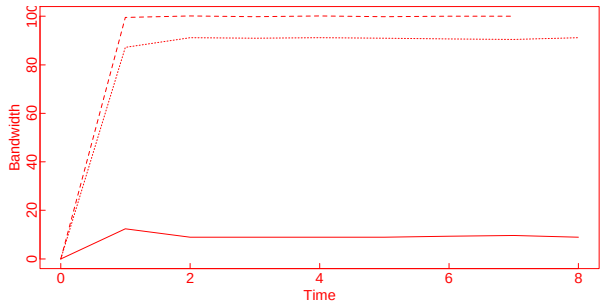

Figure 9: Ancestor-Only link-sharing with `maxIdle` for the root class set to 0.005 seconds instead of 0.000002 seconds.

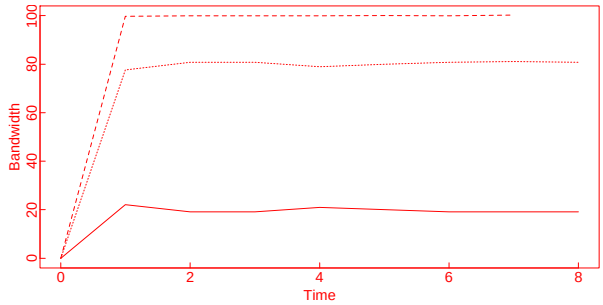

Figure 10: Ancestor-Only link-sharing with bandwidth allotment for the root class of 0.99 instead of 0.98 of the the link bandwidth.

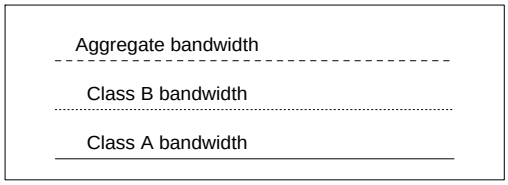

Figure 11: Legend.

more than 10% of the link bandwidth. The exact bandwidth received by the higher-priority class depends on the exact CBQ paramaters for the root class. Figures 6, 9, and 10 differ only the CBQ parameters for the root class in the link-sharing structure.

With Top-Level or Formal link-sharing, Class A is only allowed to send packets when it is underlimit or when Class B is satisfied. With Ancestor-Only link-sharing, Class A is allowed to send packets when either it or the root class is underlimit, regardless of the status of Class B.

These tests are run in `ns` with the following respective commands:

```
ns test-suite-cbq.tcl cbqTwoAO
ns test-suite-cbq.tcl cbqTwoTL
ns test-suite-cbq.tcl cbqTwoF
ns test-suite-cbq.tcl cbqTwoAO2
ns test-suite-cbq.tcl cbqTwoAO3
```

3 Round-robin scheduling algorithms

The simulations in this section explore the differences between the packet-by-packet round robin and the weighted round-robin scheduling algorithms.

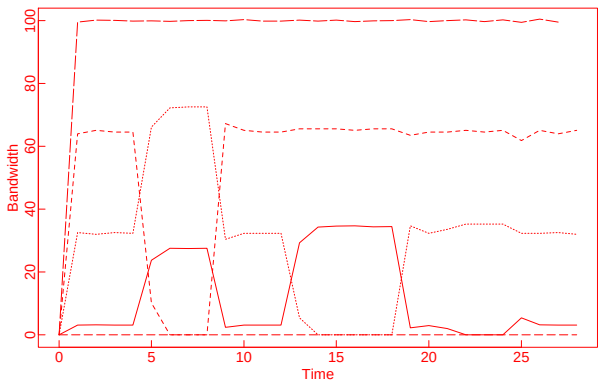

Figure 12: PRR

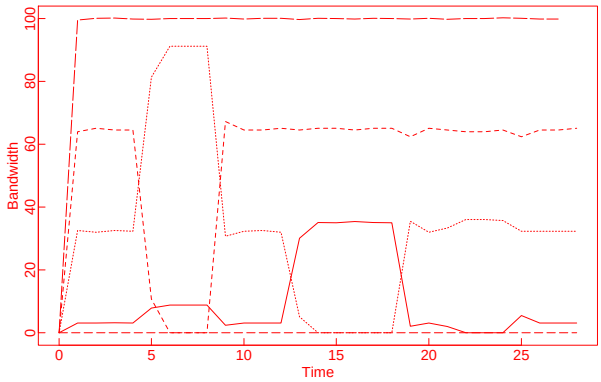

Figure 13: WRR

Figures 12 and 13 reproduce Figure 10 from [FJ95]. The simulation scenario is described in Section 5.1 of [FJ95]. Figure 12 uses PRR, and Figure 13 uses WRR. The two simulations differ in the distribution of "extra" bandwidth to the high-priority classes when the lower priority class has no data to send. When the ftp class has no data to send, with weighted round-robin the extra bandwidth is distributed to the audio and video classes in proportion to their allocated bandwidth. With packet-by-packet round-robin the extra bandwidth is distributed equally between the audio and video classes.

These simulations can be run in `ns` with the respective commands:

```
ns test-suite-cbq.tcl cbqPRR
ns test-suite-cbq.tcl cbqWRR
```

4 Sensitivity to parameters in PRR

The simulations in this section show the sensitivity of packet-by-packet round robin to the CBQ parameters. This sensitivity to parameters is not shared by the weighted round-robin scheduling algorithm.

4.1 Formal link-sharing

These tests show PRR.

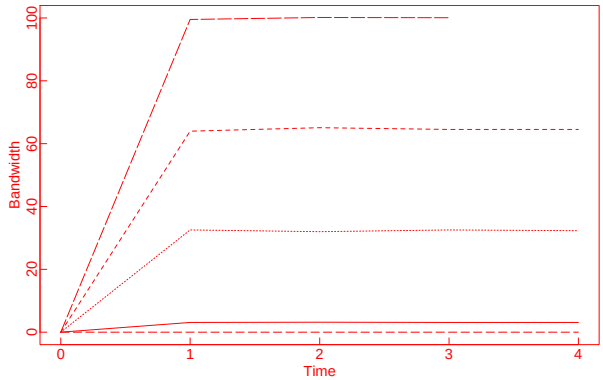

Figure 14: PRR.

Figure 14 shows a link shared by three classes. Class B, at priority 1, is allocated 32% of the link bandwidth. Class A, also at priority 1, is allocated 3% of the link bandwidth. Class A is not allowed to borrow bandwidth from the root class. Class C, at priority 2, is allocated 65% of the link bandwidth.

Figure 14, shows the results with packet-by-packet round-robin scheduling. Class A is correctly restricted to at most 3% of the link bandwidth. This test does not work correctly in the early version of the distributed code, because in that code the variable `avgidle` has a lower bound of zero and if `extradelay` is not set, Class A gets more bandwidth than intended (at the expense of Class C).

This simulation is run in `ns` with the command:

```
ns test-suite-cbq.tcl cbqMin1
```

The results are essentially the same with Ancestor-Only link-sharing.

5 The extradelay parameter

Figures 15 and 16 show that the parameter `extradelay` functions as intended in the simulator. These figures show simulations with different values for extradelay (which determines the steady-state burst size). This parameter is discussed further in [Flo95], where it is called the *offtime* parameter.

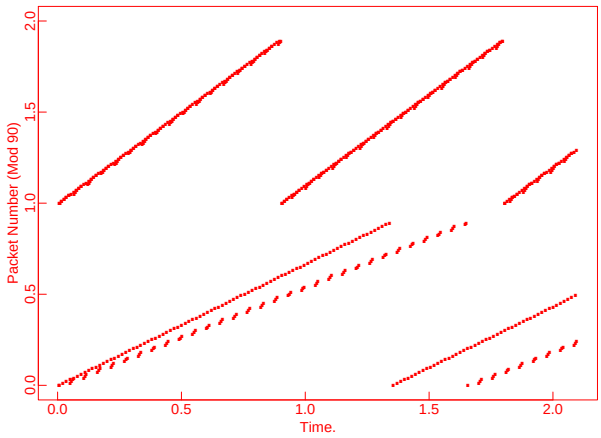

Figure 15: Set extradelay for a steady-state burst of 2 packets.

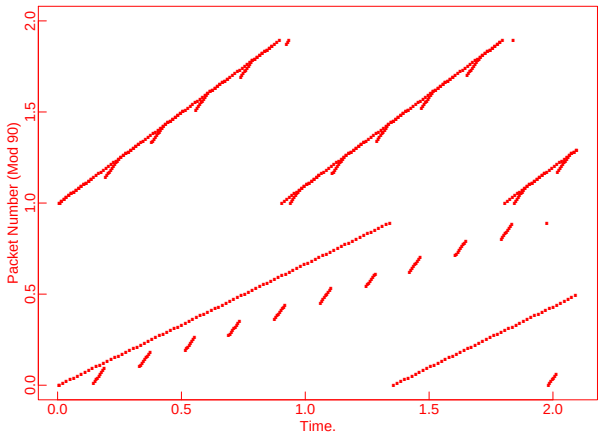

Figure 16: Set extradelay for a steady-state burst of 8 packets.

In these figures, the bottom row shows the packets for Class A, and the top row shows the packets for Class B. The x-axis shows time, and the y-axis shows a linear function of the packet number `mod 90`. There is a mark when the packet arrives at the congested gateway, and another mark when the packet departs the congested gateway.

These simulations are run in `ns` with the respective commands:

```
ns test-suite-cbq.tcl cbqExtra1
ns test-suite-cbq.tcl cbqExtra2
```

6 The `maxidle` parameter

Figures 17 and 18 show that the parameter `maxidle` functions as intended in the simulator. The `maxidle` parameter is discussed further in [Flo95]. These figures show simulations with different values for maxidle (which determines the maximum number of back-to-back packets). The `maxidle` parameter serves a similar function as does the bucket size in a token bucket.

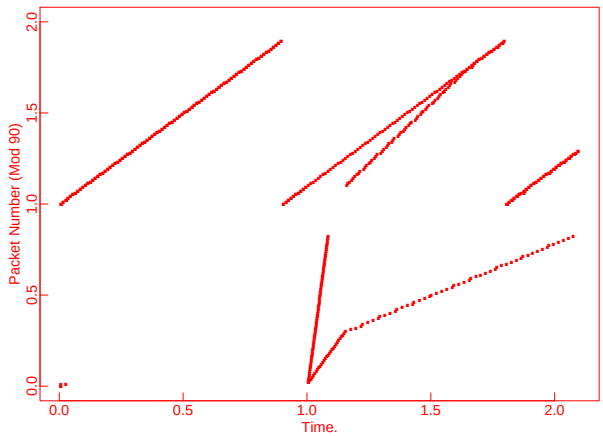

Figure 17: Set maxidle to enable 50 back-to-back packets.

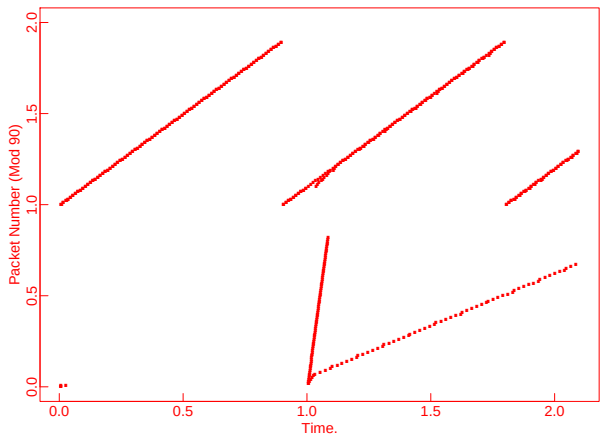

Figure 18: Set maxidle to enable 5 back-to-back packets.

In these figures, the bottom row shows the packets for Class A, and the top row shows the packets for Class B. These simulations are run in ns with the following respective commands:

```
ns test-suite-cbq.tcl cbqMax1
ns test-suite-cbq.tcl cbqMax2
```

7 Acknowledgements

Kevin Fall has made this document consistent with the current CBQ code base in ns version 1. He has also implemented a new version of CBQ for ns version 2 [McC97].

References

[FJ95] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4), 1995.

[Flo95] S. Floyd. Notes on class-based queueing: Setting parameters. *Unpublished draft*, Jul. 1995. URL ftp://www-nrg.ee.lbl.gov/floyd/papers.html.

[McC97] S. McCanne. Ns-2 (network simulator version 2), 1997. URL http://mash.cs.berkeley.edu/ns.

[MF95] S. McCanne and S. Floyd. Ns (network simulator), 1995. URL http://www-nrg.ee.lbl.gov/ns.