Chapter 10

Simulating Random Scenarios

10.1 Introduction

The performance of the rate-based congestion framework using TRUMP/RBCC has been compared against TCP Reno and TCP Vegas in a set of hand-designed network scenarios with specific congestion problems. It may be argued that this small set of scenarios does not fully test the framework, and that a wider range of scenarios should be tried.

In this chapter I discuss the testing of the framework on a set of 500 randomly generated network scenarios. Both TCP Reno and TCP Vegas are also simulated on these scenarios to allow for comparisons of simulation results.

10.2 Scenario Generation

The 500 network scenarios were pseudo-randomly generated[1] by a small Perl script. Each scenario contains 9 routers, 15 sources and 15 destinations. The 9 routers are randomly interconnected to form a single network, and the 15 sources and destinations are randomly connected to the routers.

Each source transmits 1,500 packets of 1,500 octets to its destination, and the start time for each source is distributed evenly over the interval 0:100 seconds.

In order to increase the probability of network congestion, the overall bit rates and latencies for the links in each scenario were chosen as follows. There is a probability of 0.6 that a link is *fast*: fast links have capacities spread evenly over 1,000,000:10,000,000 bits per second and latencies spread evenly over 100:10,000 microseconds. If a link is not fast, it is *slow*: slow links have capacities spread evenly over 32,000:250,000 bits per second and latencies spread evenly over 10:5,000 milliseconds.

All other network scenario parameters are the same as the values given in Section 9.1. Details on how to obtain the 500 generated network scenarios are given in Appendix G.

[1] The 4.4BSD `rand()` function was used, which conforms to the ANSI C standard. For each scenario generated, it was seeded with the current time in seconds since 1970. There was a delay of at least 100 seconds between the generation of each scenario.

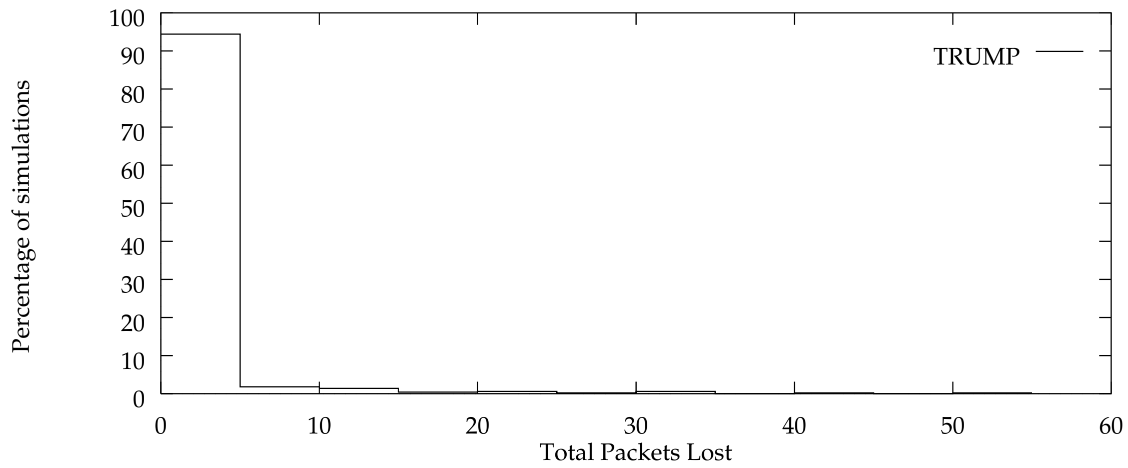

Figure 55: Total Packets Lost by TRUMP/RBCC

The distribution of total packets lost per simulation by TRUMP/RBCC is shown in Figure 55. This is an excellent distribution for packet loss, skewed towards 0 packets lost per scenario. The median of the distribution is actually 0 packets lost. In fact, 458 out of 500 scenarios simulated lost *no* packets whatsoever. Out of the minimum 22,500,000 packets admitted into the network, TRUMP/RBCC lost only *551* packets, a loss of roughly 0.002%

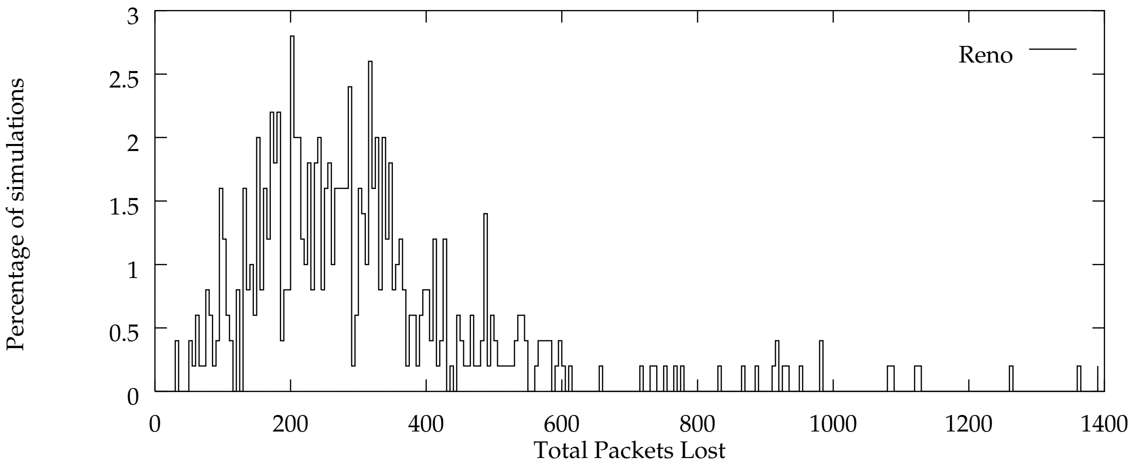

Figure 56: Total Packets Lost by TCP Reno

TCP Reno, on the other hand, shows a very disappointing distribution of packet loss. In fact, all of the 500 simulated scenarios lose packets with TCP Reno. The median of the distribution is 274 packets lost, with 90% of the distribution in the range 75:547 packets lost. TCP Reno loses 155,833 packets in total over the 500 scenarios. Clearly, TRUMP/RBCC provides much better congestion control than TCP Reno, based on packet loss alone.

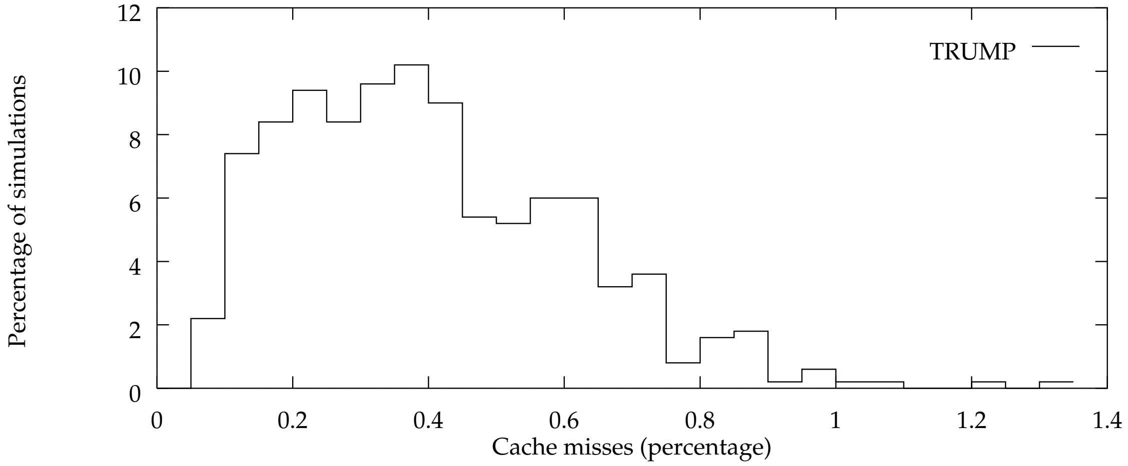

Figure 63: Percentage of Cache Misses

10.12 Summary of Results

In order to confirm that the simulation results of TRUMP/RBCC on a set of hand-picked network scenarios were not abnormal, TRUMP/RBCC was simulated on 500 pseudo-randomly generated network scenarios. As well, TCP Reno and Vegas were simulated on these scenarios to allow for comparisons of simulation results.

For each simulation, several key results were abstracted: this includes average router queue lengths, packets lost, link utilisation, throughput of slowest source and end-to-end variance. These were presented as a set of distributions for the 500 simulated scenarios.

TCP Reno was shown to be a poor network congestion scheme. Router queue lengths were very high, and packets were lost in all scenarios. In most scenarios, the packet loss was high. Link utilisation was very poor for the most congested link, and the sources generally achieved a lower effective bit rate than with TRUMP/RBCC. Finally, end-to-end variance and inter-packet spacing was high, which makes TCP Reno unsuitable for time-sensitive network traffic.

The congestion control of TCP Vegas is much better than TCP Reno, as packet loss is very much improved. However, in many respects, the performance of Vegas is similar to Reno: high router queue lengths, poor link utilisation and poor end-to-end variance. One serious drawback with TCP Vegas is its apparent unfair treatment of long path traffic flows with regards to available bandwidth: this problem alone should rule out the replacement of TCP Reno with TCP Vegas in the current Internet.

The results show that TRUMP/RBCC provides a very good congestion control scheme indeed. Router queue lengths were close to the ideal value of 1, even for severely congested routers. In over 90% of the scenarios simulated, no packets were lost, and very few packets were lost in other scenarios. Link utilisation for the most congested link was close to full utilisation. End-to-end variance and inter-packet spacing changes were very suitable for time-sensitive network traffic. Finally, with implementation techniques such as congestion field caching in routers, the overhead to implement the rate-based congestion control framework using TRUMP/RBCC appears to be low.