Proceedings of the 2007 Winter Simulation Conference
S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.

VALIDATION OF SIMULATED REAL WORLD TCP STACKS

Sam Jansen
Anthony McGregor

WAND Network Research Group
University of Waikato
Private Bag 3105
Hamilton, NEW ZEALAND

ABSTRACT

The TCP models in ns-2 have been validated and are widely used in network research. They are however not aimed at producing results consistent with a TCP implementation, they are rather designed to be a general model for TCP congestion control. The Network Simulation Cradle makes real world TCP implementations available to ns-2: Linux, FreeBSD and OpenBSD can all be simulated as easily as using the original simplified models. These simulated TCP implementations can be validated by directly comparing packet traces from simulations to traces measured from a real network. We describe the Network Simulation Cradle, present packet trace comparison results showing the high degree of accuracy possible when simulating with real TCP implementations and briefly show how this is reflected in a simulation study of TCP throughput.

1 INTRODUCTION

Network protocols are often tested, developed and evaluated with the help of network simulators. Using simulation allows a researcher to have full control and transparent access to data in a complex distributed system, something that is difficult (and in some cases impossible) with real systems today. One of the protocols that is of great import is the Transmission Control Protocol (TCP), as it is used on the Internet and many other networks for transfer of reliable data.

For simulation results to be credible the simulation models in use must undergo *verification and validation*. Balci (1997) defines verification as substantiating that a model is built from a problem formulation accurately, where validation is substantiating that the model behaves with satisfactory accuracy within its domain. Carson (2002) and Sargent (2003) define the two terms to be similar and both note that sufficient accuracy is when a model can be used instead of a real system for purposes of experimentation and analysis.

In the context of simulation models for the Transmission Control Protocol (TCP), the models should be tested that they conform to specification (verification of the model) and that the model implementation produces results consistent with a real system (validation of the model).

The ns-2 (Information Sciences Institute 2004) simulator has a test suite that tests many facets of the simulator including the one-way TCP agents (Floyd 1997). The TCP tests cover a range of situations designed to provoke certain behaviour for each TCP variant. For example, the fast recovery mechanism of TCP Reno is tested with differing amounts of packet loss. A similar, though less thorough, set of tests exists for the bidirectional TCP agents (Fall, Floyd, and Henderson 1997). This type of testing is a verification that the models produce results consistent with specifications.

Floyd (1997) points out that the TCP models in the ns-2 simulator are not designed to model one specific real world TCP implementation but be a general model for experimenting with the underlying congestion control algorithms. The Network Simulation Cradle (Jansen and McGregor 2005) (NSC) uses real TCP implementation code in TCP models in simulation, allowing a different sort of validation to be used. The simulation model can be *directly compared* to a real machine: the output of the simulation model should be very close to that of a real machine given the same input.

The Network Simulation Cradle makes the OpenBSD, Linux and FreeBSD network stacks available as ns-2 TCP agents. These agents have the same interface as the original ns-2 TCP models, making it very easy to use NSC models in the place of or in addition to the existing models in a simulation script. This paper presents validation work done with the Network Simulation Cradle and the network simulator ns-2 (Information Sciences Institute 2004) and shows the degree of accuracy attained when using real world code for simulation of TCP. The results presented here expand on the previous work (Jansen and McGregor 2005) to provide further validation and show how accurate simulating with real world code is.

Bagrodia and Takai (1999) raise the question of whether a TCP model is correct with respect to actual TCP implementations and list two cases where validation was quite successful in their work with the GloMoSim (Zeng, Bagrodia, and Gerla 1998) simulator. Direct incorporation of the implemented protocol into the model allows the protocol model to be validated against an operational prototype. Comparison of independently developed models for a given protocol provide further validation information.

The first method is used in this paper, both at a micro level with comparisons of traces in section 3 and at a macro level where TCP performance is compared in section 4. First, a brief description of the Network Simulation Cradle is provided.

2 THE NETWORK SIMULATION CRADLE

NSC is an architecture that allows real world TCP implementations to be used as TCP simulation models. This is achieved by a runtime system that separates the TCP implementations for the simulator and bridges between the simulator and TCP implementations, a process used to integrate new TCP implementations and an off-line tool that modifies code to allow multiple independent instances of code to run within the same process.

The simulator includes a module that is responsible for loading TCP implementations and bridging between the abstractions used in the simulator and a consistent interface exposed from the support code for each TCP implementation. The TCP implementations are contained in shared libraries which contain code to provide the interface the simulator module expects.

The reader is referred to previous work (Jansen and McGregor 2005) for a more detailed explanation of the architecture of the NSC. The Network Simulation Cradle version 0.2.2 (NSC is available for download from `<research.wand.net.nz/software>`) is used in the experiments presented in this paper with ns-2 version 2.29. NSC 0.2.2 includes the network stacks Linux 2.4.28, Linux 2.6.10, FreeBSD 5.3 and OpenBSD 3.5.

3 TRACE COMPARISONS

The Network Simulation Cradle can produce packet trace files in the format used by tcpdump (Jacobson, Leres, and Mccanne 2005). Tcpdump captures packets from a network interface and optionally saves them to a file. A simulation can be modelled after a test network setup and tcpdump traces can be recorded at the same logical points in the two networks. The network trace from NSC and from a real machine can then be directly compared using trace analysis tools such as tcptrace (Ostermann 2005). Such a method of comparison is used in the experiments shown in this section.

3.1 Emulating with a Testbed Network

Building computer networks of varying topologies, varying link bandwidths and delays, possible packet loss, controlled router buffer sizes and differing TCP implementations is expensive and time consuming. This is one of the reasons simulation is performed; often it is impractical (or even impossible) to build networks to test a protocol or idea. Simulation of an entire network has many abstractions and needs to be validated against real systems, so a compromise often referred to as *emulation* is used. Network emulation is used here to mean a physical network which includes a device or set of devices that simulate part of the network. An example of this would be machine set to route packets between its network interfaces, delaying packets by 20ms. This machine would be simulating a long link in the network topology by adding the artificial delay.

3.2 The WAND Emulation Network

The WAND Network Research Group has a network of 24 machines available for testing. This network is called the WAND Emulation Network (Jones 2007). The machines in the WAND Emulation Network have multiple network interfaces. One network interface card is connected to a central server to form a control network. The other network interface card is connected to a patch panel which in turn is connected to a switch. Some of the machines have four Ethernet ports on their second card, allowing them to be used as routers. The machines are configured with a topology by changing connections on the patch panel. All machines are also connected to a terminal server to allow administration without relying on networking. This network is used for all real measurements presented in this paper.

Six machines are configured in a simple dumb-bell topology as shown in Figure 1. Two machines run FreeBSD 5.3 and use ipfw Dummynet (Rizzo 1997) to shape traffic. Due to the scheduling of packet delays with Dummynet (Dummynet processes delays on the software interrupt clock, which fires once every $1/HZ$ seconds; HZ is set to 1000 in these tests), the round trip time (RTT) on this network has some noticeable variation as presented in Table 1. The RTT of an equivalent network simulated with ns-2 is also shown. The results are gathered from 1000 pings on an otherwise unloaded network. A FreeBSD Dummynet router is configured to delay packets by 21ms in both directions and limit bandwidth to 2Mb/s.

The variation in timing on the testbed network shown in Table 1 means that there will be some small variation in timing between the simulated trace and the measured trace.

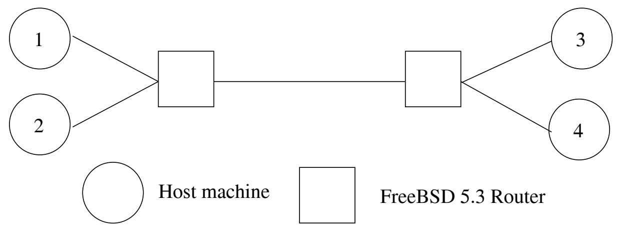


Figure 1: Emulation network setup.

Table 1: Emulation network RTT measurements.

	Round trip time (ms)				
Packet size	Min	μ	Max	σ	Simulated
84	43.0	43.6	49.9	0.588	43.1
1500	53.3	53.9	61.1	0.653	54.4

A direct binary comparison of the traces is therefore not useful. The sequence numbers and time used in the traces is also not synchronised. The tcpnorm (Jansen 2007) utility is used to normalise the traces and tcptrace (Ostermann 2005) is used to visualise them by making time-sequence graphs.

The bottom line on the tcptrace time sequence graphs seen later in this paper is the TCP sequence number which has been acknowledged to. The top line is the TCP acknowledgement number plus the receivers advertised window. This shows visually the window in which the data packets should be sent. Data packets are indicated by small black double-ended arrows. If the packet is a retransmission, it will have an "R" next to it. Selective acknowledgement blocks are shown by lines within the advertised window with an "S" next to them. If a data packet has the PUSH flag set a diamond will be drawn around the packet.

3.3 Connection Establishment

Figure 2 shows tcptrace graphs of TCP during connection establishment and slow start. For each operating system FreeBSD, Linux and OpenBSD a trace is measured on the emulation network and created in simulation. The traces are normalised with tcpnorm then graphed with tcptrace. The two graphs for each operating system are shown side by side. A Dummynet router limits bandwidth to 2Mb/s, delays packets in both directions by 21ms and has a queue length of 10 packets. The simulation scenario is configured to be equivalent.

Each of the pairs of graphs in Figure 2 are very close matches for each other. In addition to these graphs, each situation is analysed in detail using the textual output of tcpdump in the following sections.

3.3.1 FreeBSD

The two traces for FreeBSD are very close. The sequence and content of packets shown in Figures 2(a) and 2(b) are nearly identical except for the TCP timestamp option. The throughput measured on the emulation network is within 2% of the throughput measured in the ns-2 simulation. The TCP timestamp option differs by one often in the traces. The reason for this is that the timestamp counter is based on the ticks variable in the network stack which in this situation occurs once every 10ms. This timer starts counting when the machine boots up, so synchronising it between simulation and the real machine is not practical.

There are a small difference in timing of packets. This is due to the difference in round trip time and variation in timing found in the emulation network, as described in §3. The per-packet time difference is plotted in Figure 3. This graph shows how, in this case, the time differences accumulates over time (this is not always the case for other network stacks tested). This eventually leads to a slightly different ordering of packets, though the tcptrace graphs look similar.

3.3.2 Linux

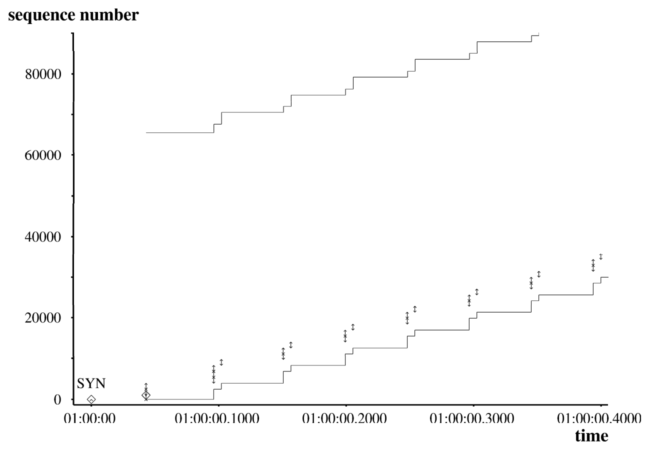
The traces for Linux look similar in Figures 2(c) and 2(d). The one notable difference is some of the data packets have diamonds around them meaning they have the PUSH flag set.

The PUSH flag in TCP was originally specified in RFC 793 to mean that when a receiving TCP sees the flag, it must not wait to receive any more data before passing the data to the receiving process. In practise, data is passed to the application as soon as possible irrespective of the PUSH flag and it is set by the sending network stack, rather than the application, in most recent TCP implementations.

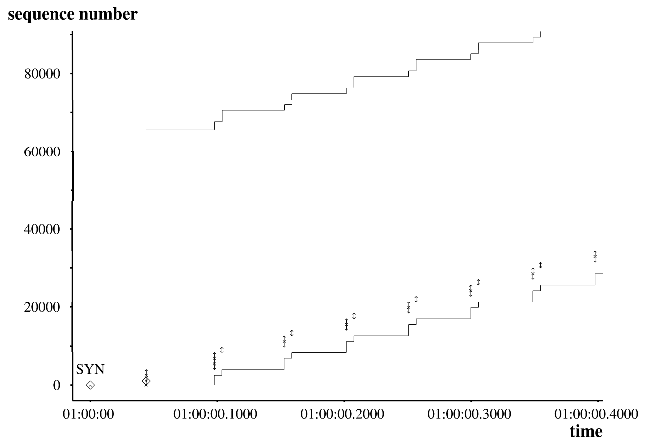
The interface between application and network stack is very different in simulation with ns-2 and on a real machine, so the model of the application is slightly different between the two. These differences result in the PUSH flag being set for extra packets in simulation.

The TCP timestamp option differs between the traces. The counter used for the timestamp is increased once every millisecond in the version of Linux studied. The packets are consistently between 0 and 3 milliseconds different in their timings and the TCP timestamp option reflects this.

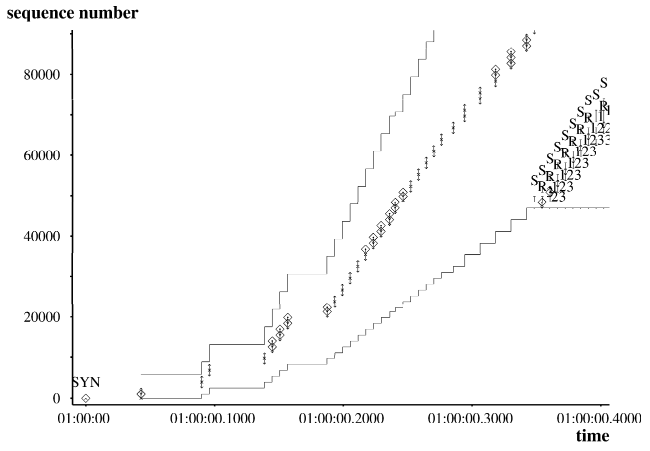
Linux 2.6 tunes the windows used in TCP based on the amount of memory available in the machine. The cradle code attempts to have reasonable defaults set that match up to the machines on the emulation network. The receivers advertised window grows dynamically and is additionally affected by the size of the packet structure allocated in the Ethernet driver. This is an example of subtle interactions coming from seemingly unimportant supporting code. We believe that validating real world code in simulation is important, because there is the possibility of many interactions such as this affecting results.



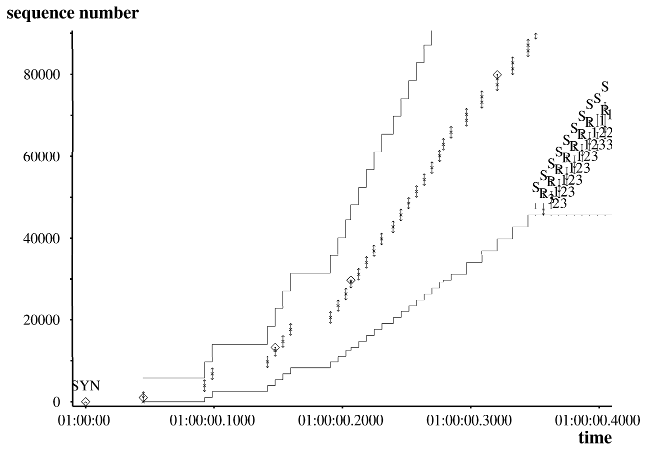
(a) Simulated FreeBSD



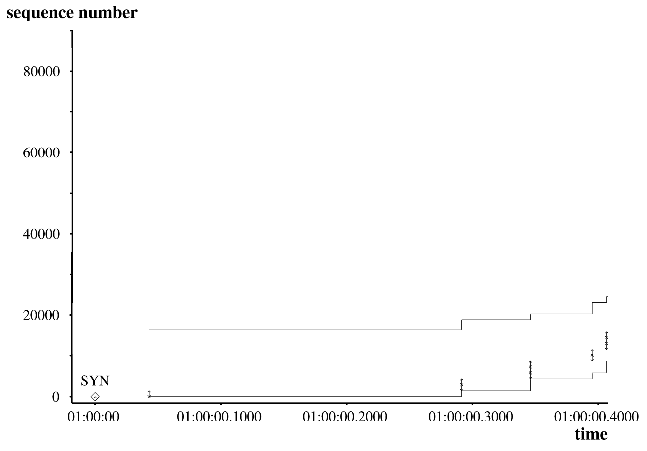
(b) Measured FreeBSD



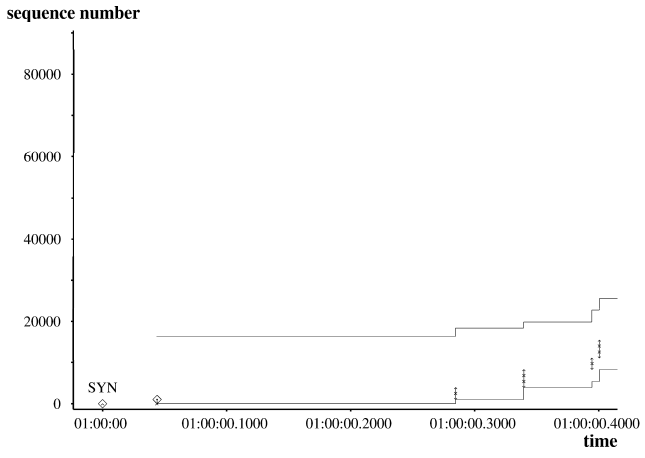
(c) Simulated Linux



(d) Measured Linux



(e) Simulated OpenBSD



(f) Measured OpenBSD

Figure 2: Simulated vs. measured connection establishment graphs.

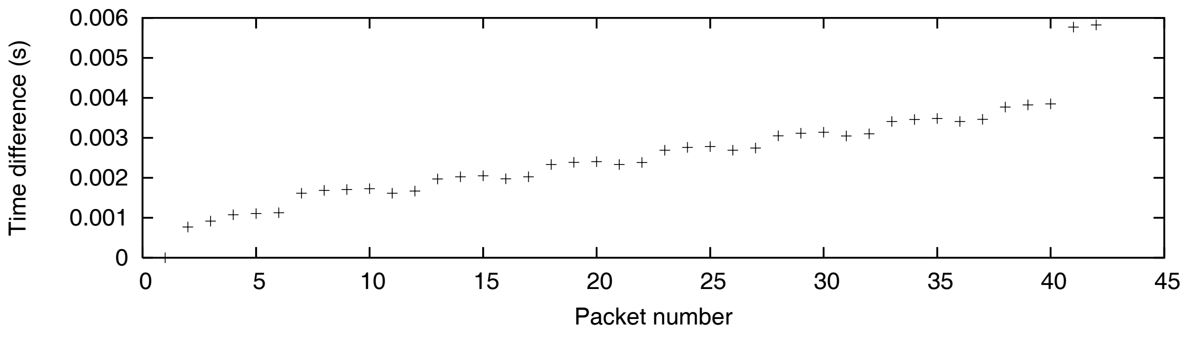


Figure 3: Time difference vs. packet number for FreeBSD traces.

When a packet is received in a network driver, the driver allocates a structure called an skbuff with enough space to hold the packet. It is up to the driver to select the space for the packet received, often there is extra slack space that is unused by the driver (but possibly used later by other sections of the network stack). This packet is then sent on to the network stack. When calculating the receivers advertised window, the size of the skbuff is checked as Listing 1 shows.

```
int incr;
/*
 * Check #2. Increase window, if skb
    with such overhead
 * will fit to rcvbuf in future.
 */
if (tcp_win_from_space(skb->truesize)
    <= skb->len)
         incr = 2*tp->advmss;
else
         incr = __tcp_grow_window(sk, tp
             , skb);

if (incr) {
         tp->rcv_ssthresh = min(tp->
             rcv_ssthresh + incr, tp->
             window_clamp);
         tp->ack.quick |= 1;
}
```

Listing 1: Linux 2.6 `tcp_grow_window` code.

In Listing 1 `skb->truesize` refers to the size of the skbuff allocated in the driver. To obtain the same traces on real machines and in simulation, the simulation driver code needs to allocate `skbuff` sizes in the same manner as the driver used on the real machine. The simulation driver allocates `skbuffs` similar to the *eepro100* driver used on the emulation network machines and is able to produce the same offered window sizes as those measured on the emulation network.

The traces are identical until the difference in PUSH flags save for the slight timing differences described above. On the real machines some data packets are generated later in the trace that are smaller than the MTU. This is due to application differences, the timing of when data is written to the TCP socket by the application is different between simulation and the real machine which results in this behaviour. The traces are very similar when visualised with tcptrace and the goodput (the amount of data received by the application layer from TCP) measured on the emulation network is within 2% of the goodput recorded in simulation.

3.3.3 OpenBSD

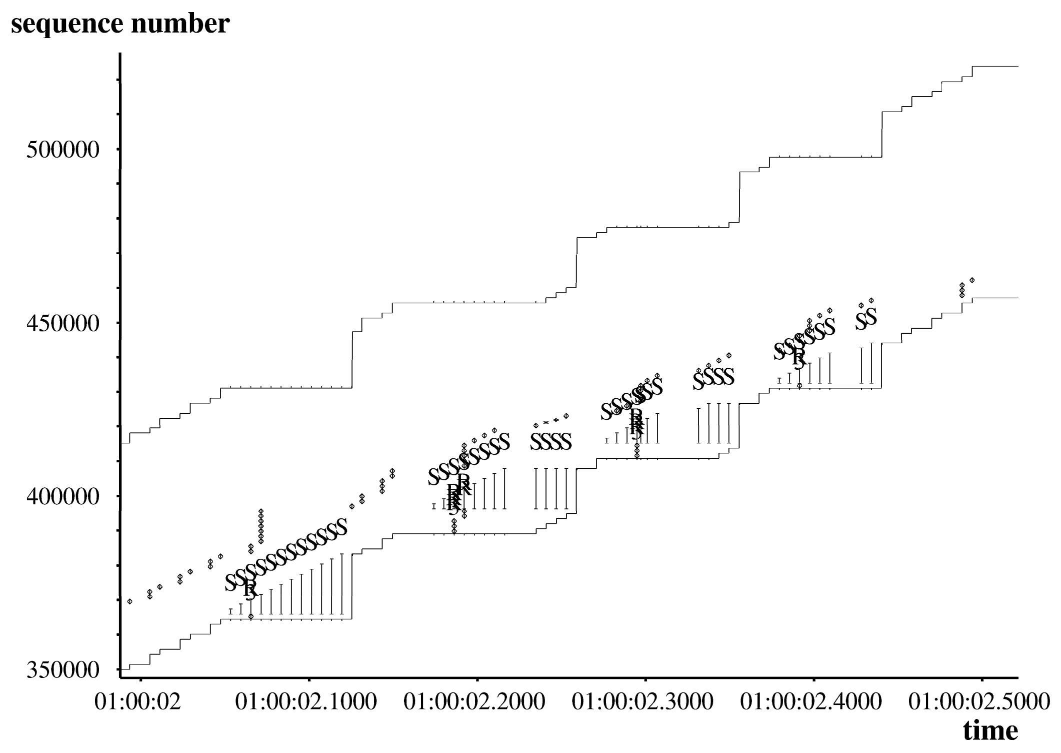
The sequence of packets shown in Figures 2(e) and 2(f) are very close matches. When the traces are analysed further it is evident some TCP timestamps vary between the traces by one. This occurs for the same reason it does in the FreeBSD trace and is described earlier.

There are less data packets in the graphs showing OpenBSD (Figures 2(e) and 2(f)) due to the OpenBSD sender only sending one initial data packet after the three-way handshake of TCP. The acknowledgement for this packet is not sent straight away by the other end of the connection due to the delayed acknowledgement mechanism: either the delayed acknowledgement timer must fire or two packets must arrive. This is one of the reasons for RFC 3390 (Allman, Floyd, and Partridge 2002) which increases the initial TCP window size. The version of OpenBSD tested does not implement RFC 3390 while the versions of Linux and FreeBSD studied here do. Figures 2(e) and 2(f) show a timer firing with the same duration in emulation and simulation: the acknowledgement is received which results in further data packets prior to time 01:00:00.3000. The acknowledgement is received at this time due to the delayed acknowledgement timer being set to 200ms. This verifies that this TCP timer is firing at the correct time.

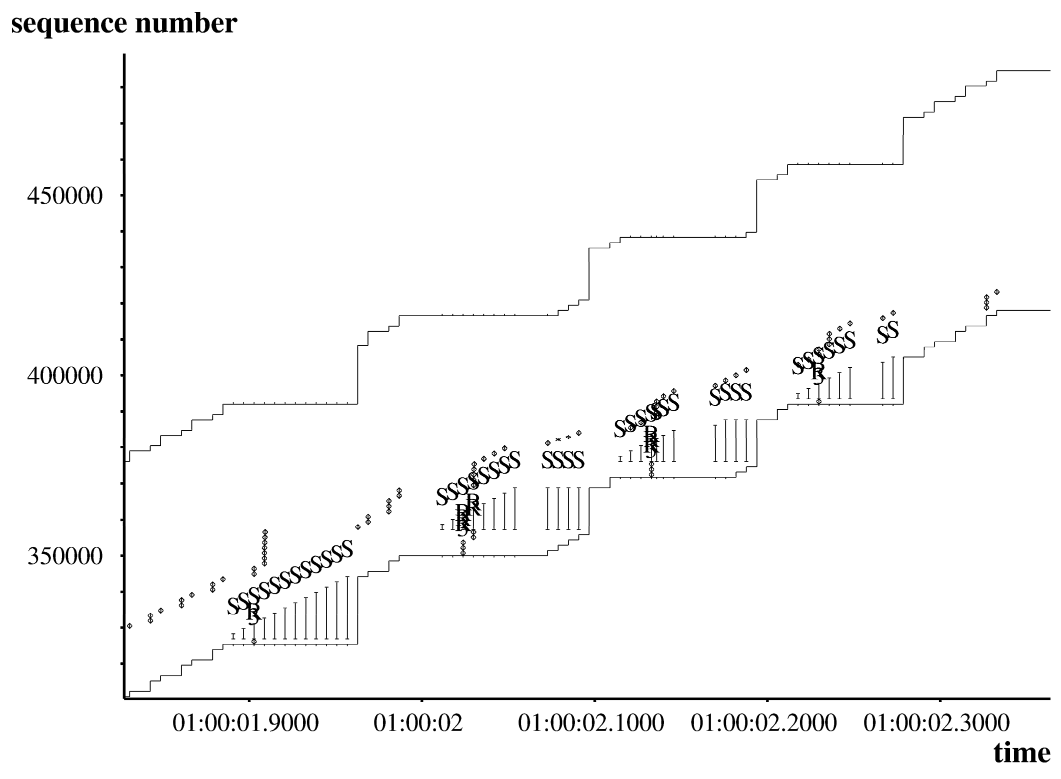
The timing difference of packets is similar to FreeBSD (see Figure 3). This eventually leads to a different sequence of packets, although an overall tcptrace graph of the connection looks nearly identical and the throughput recorded in simulation is within 2% of the throughput measured on the emulation network.

3.4 Congestion

Figures 4 and 5 are tcptrace graphs of TCP undergoing loss because it has overflowed the router queue size. The

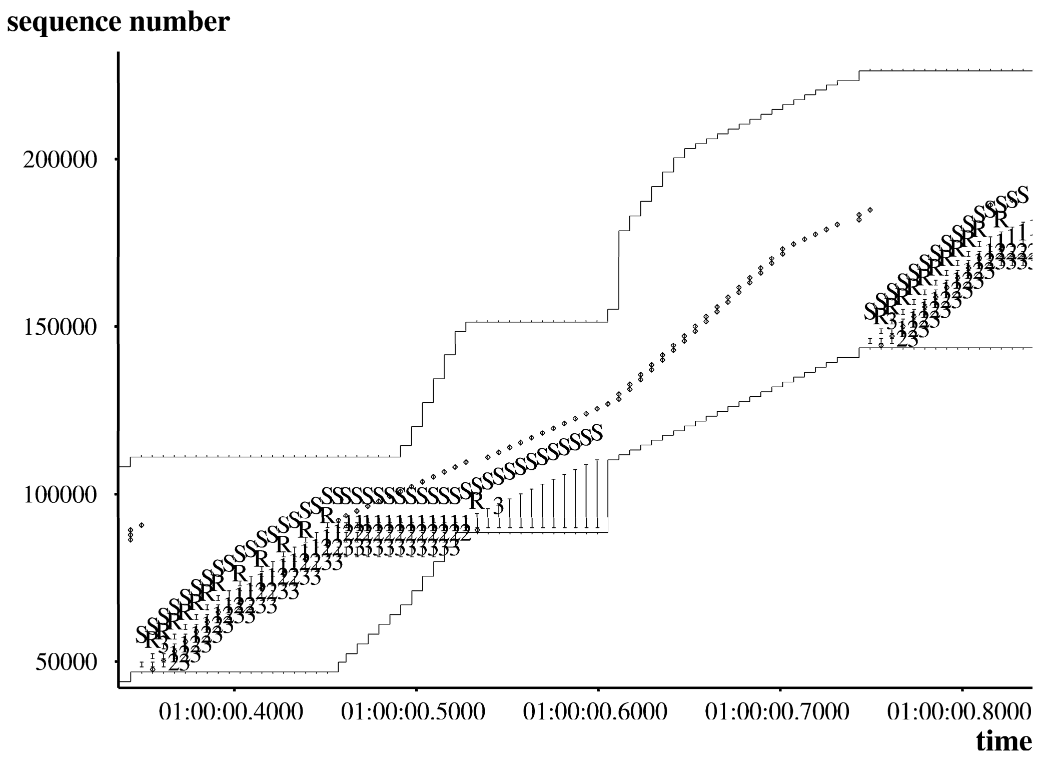


(a) Simulated FreeBSD

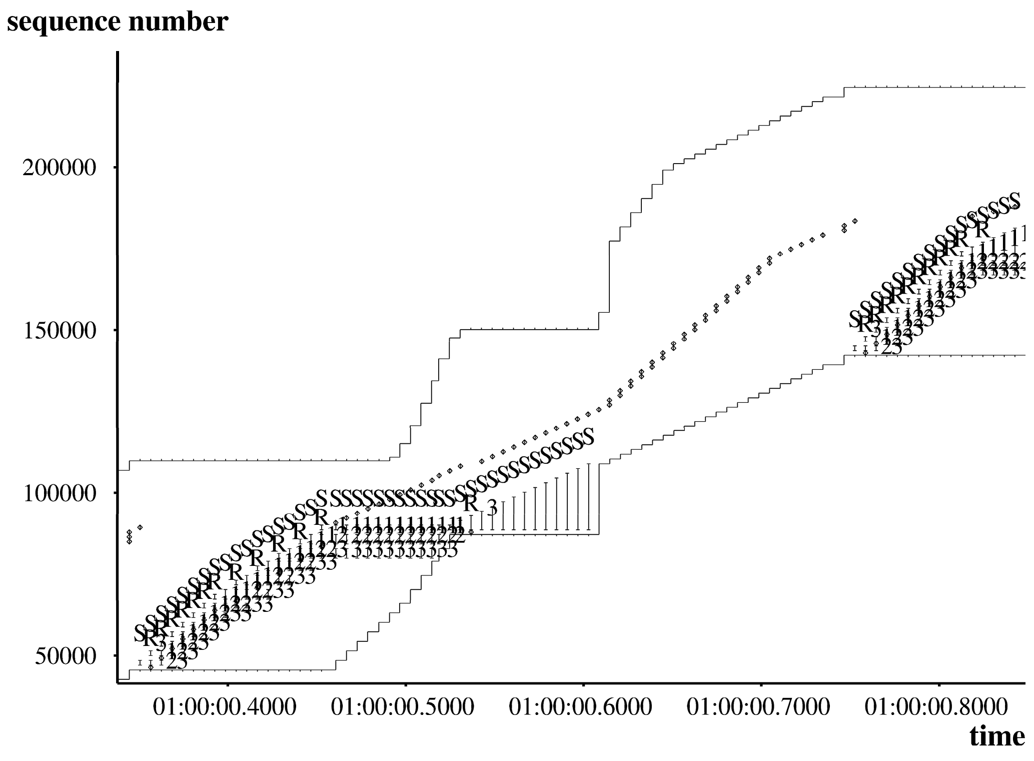


(b) Measured FreeBSD

Figure 4: Simulated vs. measured TCP packet loss response for FreeBSD.



(a) Simulated Linux



(b) Measured Linux

Figure 5: Simulated vs. measured TCP packet loss response for Linux.

scenario simulated and measured is the same as earlier, these graphs are produced from later in the connection.

3.4.1 FreeBSD

FreeBSD responds to the packet loss in the same manner in simulation and on the testbed network. Figure 4(a) and Figure 4(b) show the same selective acknowledgement ranges and bursts of data packets due to the loss. The difference in the two graphs is the time and sequence numbers shown on the axis. This is due to loss occurring slightly earlier on the emulation network. While the difference in network produces this discrepancy, the graphs show the algorithmic response of TCP is the same in simulation as it is on the real machines.

3.4.2 Linux

The graphs in Figure 5 do not show the TCP PUSH flag as previous graphs have. This makes the graphs easier to follow and compare. The two graphs have the same sequence numbers and time shown, unlike Figure 4, and are almost an exact match. The response to packet loss is the same with a simulated Linux TCP stack and one running on a real machine.

4 MEASURED TCP PERFORMANCE

Measurement studies have found the presence of random loss on the Internet (Zhang, Paxson, and Shenker 2000) and uniform random loss is used as a simple model for loss encountered on the Internet (Lakshman, Madhow, and Suter 2000, Padhye et al. 2000) (or other networks, for example, ATM networks, Romanow and Floyd 1995) in many simulation studies. This section presents a study of TCP performance under uniform random packet loss showing comparisons between ns-2 TCP models, NSC TCP implementations and measurements from a test network to validate the NSC TCP implementations.

The performance of TCP during varying uniform random loss rates is presented in Figure 7. Simulation results using ns-2 with its standard TCP models and with NSC TCP implementations are shown in Figure 7(a) and results measured from the WAND Emulation Network are shown in Figure 7(b). The TCP flow goes through a network with a round-trip time of 200ms and a bandwidth of 2Mb/s. Each point on the graphs is the mean of six runs of the same test. This scenario is depicted in Figure 6.

The goodput of Linux 2.6 is much higher during low loss than the other TCP implementations in simulation and measured on the testbed. An explanation is the use of BIC-TCP (Xu, Harfoush, and Rhee 2004) by default (Hemminger 2005) combined with auto-tuning buffer sizes with large maximums. BIC-TCP is a TCP enhancement design to make

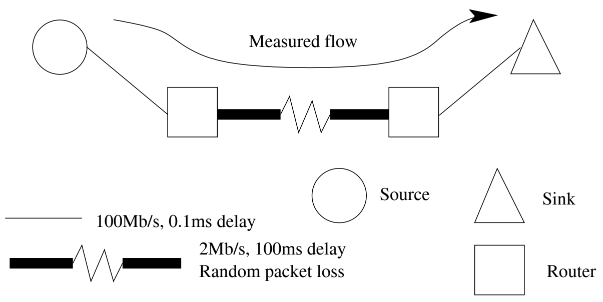


Figure 6: Simulation scenario for random loss scenario.

TCP more scalable so it is more efficient on high bandwidth delay product links. This congestion control algorithm is noticeable due to the large RTT in this experiment.

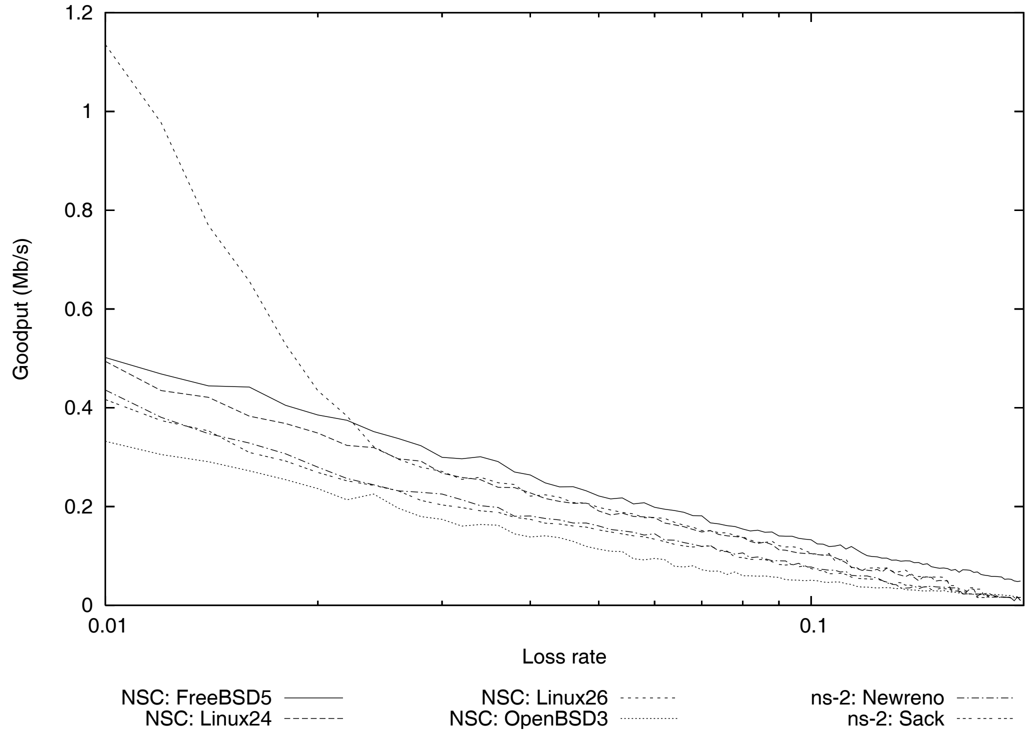
The simulation results of TCP implementations using the Network Simulation Cradle are consistent with measurements of the same implementations on the testbed network. The ns-2 TCP models have the same general trend as the real TCP implementations studied and fall in between the measurements for the real implementations.

5 CONCLUSION

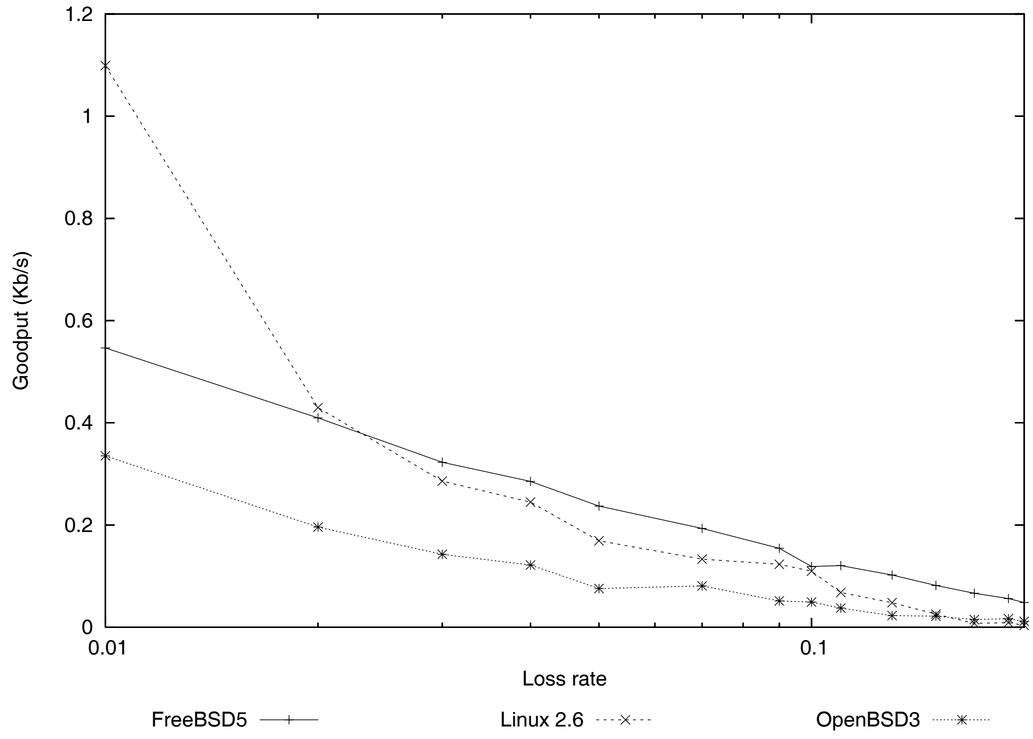
Using real world network stacks in simulation can produce very accurate results. The Network Simulation Cradle, a project that uses open source network stacks as TCP simulation models in ns-2, is able to produce packet traces that are nearly identical to traces collected from real machines.

Validating the Network Simulation Cradle resulted in several bug fixes in the implementation. Though other projects use real TCP implementations in simulation to have a full, valid, TCP implementation in simulation, we believe that the process used to extract the network stack and make it available to simulate can introduce error. Even though the network stacks in the Network Simulation Cradle have not had any lines of code modified and the process of modifying global variables is automated, we have still found subtle bugs that come from the support code that is required to allow the previously kernel-mode code to run in user-mode and in a simulator.

Visualising traces with tcptrace and comparing between a testbed network and equivalent simulation is a very helpful validation mechanism. The traces produced in both scenarios now match up to the naked eye and only small timing differences exist, save for the TCP PUSH flag in some situations. Checking traces during connection establishment, congestion and showing timers fire at the correct time gives a high level of confidence in the simulated TCP implementation. This is supported by our simulations of TCP under random loss, which agree between a measured test network and simulation.



(a) ns simulation



(b) Testbed measurements

Figure 7: TCP goodput vs. loss rate.

REFERENCES

Allman, M., S. Floyd, and C. Partridge. 2002, October. Increasing TCP's Initial Window. RFC3390.

Bagrodia, R., and M. Takai. 1999, May. Position paper on validation of network simulation models. In *DARPA/NIST Network Simulation Validation Workshop*.

Balci, O. 1997. Verification, validation and accreditation of simulation models. In *WSC '97: Proceedings of the 29th Winter Simulation Conference*.

Carson, J. S. 2002. Verification validation: model verification and validation. In *WSC '02: Proceedings of the 34th Winter Simulation Conference*, 52–58: Winter Simulation Conference.

Fall, K., S. Floyd, and T. Henderson. 1997. Ns simulator tests for reno fulltcp.

Floyd, S. 1997, May. Simulator tests. Technical report, Lawrence Berkeley Laboratory.

Hemminger, S. 2005, February. Network performance improvements in Linux 2.6. In *Linux World Expo*.

Jacobson, V., C. Leres, and S. Mccanne. Accessed 2005. tcpdump. http://www.tcpdump.org.

Jansen, S. Accessed 2007. tcpnorm. <www.wand.net.nz/~stj2/nsc/software.html>.

Jansen, S., and A. McGregor. 2005, December. Simulation with real world network stacks. In *WSC '05: Proceedings of the 37th Winter Simulation Conference*.

Jones, B. Accessed 2007. WAND emulation network. <www.wand.net.nz/~bcj3/emulation/>.

Lakshman, T. V., U. Madhow, and B. Suter. 2000, October. TCP/IP performance with random loss and bidirectional congestion. *IEEE/ACM Trans. Netw.* 8 (5): 541–555.

Ostermann, S. Accessed 2005. tcptrace. <www.tcptrace.org>.

Padhye, J., V. Firoiu, D. F. Towsley, and J. F. Kurose. 2000, April. Modeling TCP Reno performance: a simple model and its empirical validation. *IEEE/ACM Trans. Netw.* 8 (2): 133–145.

Rizzo, L. 1997. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review* 27 (1): 31–41.

Romanow, A., and S. Floyd. 1995, May. The dynamics of TCP traffic over ATM networks. *IEEE Journal on Selected Areas In Communications*.

Sargent, R. G. 2003. Verification and validation of simulation models. In *WSC '03: Proceedings of the 35th Winter Simulation Conference*, 37–48: Winter Simulation Conference.

The Network Simulator - ns-2 Accessed 2004. The network simulator - ns-2. <www.isi.edu/nsnam/ns/>.

Xu, L., K. Harfoush, and I. Rhee. 2004. Binary increase congestion control (BIC) for fast long-distance networks. In *IEEE Infocom*. IEEE.

Zeng, X., R. Bagrodia, and M. Gerla. 1998. Glomosim: A library for parallel simulation of large-scale wireless networks. In *Workshop on Parallel and Distributed Simulation*, 154–161.

Zhang, Y., V. Paxson, and S. Shenker. 2000. The stationarity of internet path properties: Routing, loss, and throughput. Technical report, ACIRI.

AUTHOR BIOGRAPHIES

SAM JANSEN is a PhD student studying at The University of Waikato supervised by Tony McGregor. The work described in part in this paper forms a part of his PhD research. He started his PhD in the WAND Network Research group in 2004. His email address is <sam@wand.net.nz> and his website is <www.wand.net.nz/~stj2/nsc>.

TONY MCGREGOR is an Associate Professor with the Department of Computer Science of The University of Waikato in New Zealand. His research areas are network measurement and simulation. He is a senior member of the Waikato University WAND network measurement group and leads the NLANR AMP active monitoring project. He teaches operating systems and computer networks.