Performance, validation and testing with the Network Simulation Cradle

Sam Jansen and Anthony McGregor
WAND Network Research Group
Waikato University
Hamilton, New Zealand

Abstract

Much current simulation of TCP makes use of simplified models of TCP, which is a large and complex protocol with many variations possible between implementations. We use direct execution of real world network stacks in the network simulator ns-2 to compare TCP performance between implementations and reproduce existing work. A project called The Network Simulation Cradle provides the real world network stacks and we show how it can be used for performance evaluation and validation. There are large differences in performance between simplified TCP models and TCP implementations in some situations. Such differences are apparent in some reproduced research, with results using the Network Simulation Cradle very different from the results produced with the ns-2 TCP models. In other cases, using the real implementations gives very similar results, validating the original research.

1. Introduction

The Transmission Control Protocol (TCP) is the ubiquitous transport protocol used on the Internet today. TCP has evolved significantly from its original specification in 1981 to include many extra mechanisms to handle situations where it was found to be inefficient [21]. Research continues with attempts to make TCP adapt to fast long distance networks quickly, handle erratic link layers (such as wireless links) and more. Much of the research employs network simulation to test new modifications to TCP or test TCP performance in a specific scenario.

Traditionally network simulators use simplified models of the various parts of the network that is to be simulated. This is true for TCP; the widely used network simulator ns-2 [25] includes simple models for TCP that are not designed to implement a specific implementation of TCP [8], do not perform full segmentation, only send data in one direction and do not implement a receivers advertised window (ns-2 also includes less abstracted "Full-TCP" models that allow bidirectional transfer of data, but these are not as well validated [7]). Despite this lack in functionality, the models are validated and allow useful evaluation of TCP congestion control algorithms [6] and have been used in many pieces of research.

The Network Simulation Cradle [16] (NSC) is a project that allows real world TCP implementations to be used in the place of the simplified TCP models present in ns-2. It allows a real TCP implementation to be used with only minor changes to the simulation script and only a small cost to performance. Simulation with NSC makes possible a range of performance evaluation of real world protocol implementations that would come at a large cost in time and resources without such a framework. It also works as a validation tool, as the real world TCP implementations can easily be used instead of the original ns-2 TCP models in an existing simulation. These ideas are explored in this paper.

This paper shows results generated from simulations with NSC. Section 3 presents NSC used as a tool for performance evaluation of TCP implementations. The TCP implementations in NSC and models in ns-2 are compared in some simple scenarios to show how NSC can be used to benchmark performance and test TCP implementations. Reproduced simulations are reported on in section 4. This shows NSC being used as a validation tool, providing further evidence of results or giving extra insight into a scenario. Related work that uses real world TCP implementations in simulation is reviewed briefly in section 5. The Network Simulation Cradle is described in the following section.

2. The Network Simulation Cradle

NSC is designed as two distinct objects that communicate through a C++ interface. There is an ns-2 *agent* implementing an ns-2 API which forms the transport protocol in the simulator: this means the agent will be connected to another agent and instructed to send data. The agent is responsible for managing and interacting with the other part of NSC, the shared library. The shared library contains the

network stack that is simulated as well as supporting code. NSC also has one other component that is used during the build process. A global parser programmatically changes references to global variables.

The C++ interface is designed to allow easy integration of simulators. As of NSC version 0.2.1[1] that is used in the simulations presented in this paper, only ns-2 is supported. Ns 2.29 is used in the simulations presented in this paper. There is interest in supporting OMNeT++ in the future. The architecture allows any network stack to be used given that the interface is implemented. The network stacks that are available are: Linux 2.6.10, FreeBSD 5.3 and OpenBSD 3.5.

Each network stack is contained in a shared library. The library contains a "cradle" that implements support functions for the network stack. Each stack requires some amount of support due to being removed from its original environment. In the case of all the stacks supported the network stack has been removed from a large monolithic operating system kernel. The facilities the operating system provide to the network stack are reimplemented in the cradle to allow the stack to function in user space and communicate with the simulator.

A program called the *globaliser* is used to allow multiple instances of each network stack to independently operate. The globaliser filters preprocessed C source code and modifies global variable declarations. This process programmatically makes the code re-entrant, allowing independent instances of the stack to run concurrently within the same process space. NSC scales up to thousands of instances of one complex network stack such as Linux 2.6.10 on a recent desktop computer.

A discussion of the architecture, implementation, performance and validity of NSC is presented in [16]. Further validation studies showing that NSC is able to produce packet traces that are very similar to real networks is shown in [17].

3. TCP performance comparisons

There is an enormous parameter space for TCP performance evaluation. There are many TCP options and parameters which may be tuned on a real system (Linux 2.6.12 has 45 kernel parameters relating to TCP), many possible network topologies and applications which may use TCP. There are many metrics which can be tested, The IRTF Transport Modeling and Research Group lists 11 metrics for evaluating congestion control algorithms [9] and many tools and characteristics to test with simulation or testbed studies.

In the following sections we do not attempt to provide a benchmark for TCP performance, but look at some sample situations where we analyse TCP implementations and models in some simple scenarios. Such testing traditionally requires large testbed networks and perhaps impractical amounts of time and resources.

[1] Available from: `http://research.wand.net.nz/software/nsc.php`

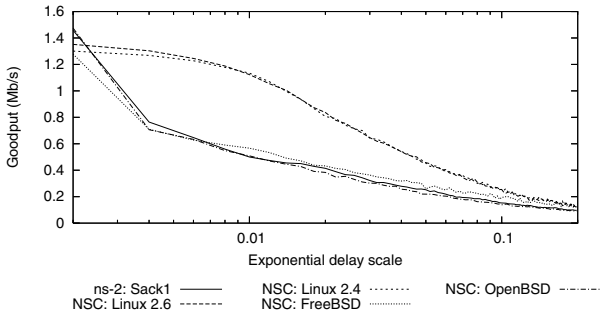


Figure 1. TCP goodput under packet reordering

The TCP implementations are setup to use their default parameters. MTU is set to 1500 bytes in all simulations.

3.1. Packet reordering

The results of a simulation designed to show how a mechanism implemented in one TCP/IP stack produces different performance results are shown in figure 1. This simulation scenario has substantial packet reordering due to packets being randomly delayed between the TCP source and sink. The TCP stream is limited by a bottleneck link of 4Mb/s and has a round trip time of 100ms. Data is transferred in one direction and packets travelling in the direction of the data are delayed by a exponential random variable. The scale factor (μ) of the exponential distribution is shown on the x-axis of the graph. At $\mu = 0$ no packet reordering is performed. Each point on the graph was generated from the mean of 20 simulation runs with differing random seeds. TCP goodput (the rate of bytes delivered to the application from TCP) is measured after 200 seconds of simulation time.

The ns-2 TCP model which uses TCP with selective and delayed acknowledgements produces very similar results to the FreeBSD and OpenBSD network stacks simulated with the Network Simulation Cradle. However, both versions of Linux have very different results.

The Linux TCP/IP stack has several mechanisms implemented to aid TCP performance during packet reordering [24]. Duplicate selective acknowledgements [12] (DSACK) help distinguish between packet loss and packet reordering. The Linux TCP/IP stack uses TCP timestamps to help detect spurious retransmissions similar to the TCP Eifel [14] algorithm.

3.2. Dumbbell topology tests

The simulation scenario presented in this section is an attempt to characterise how networking conditions in a constrained scenario affect TCP goodput. A dumbbell topology is used with flows F flowing in one direction, R in the opposite direction and flow M in the direction of F flows. The flows F and R have uniformly distributed RTTs in the interval $[8, 222]$ms. The number of flows in F is varied between 0 and 100 and R is varied between 0 and 5. The routers on the bottleneck link have queue sizes ranging from 6 to 50 packets. The bandwidth of the bottleneck link ranges from 512kb/s to 10Mb/s. Each set of parameters is simulated with 10 random seeds. Flow M is measured and has an RTT of 8ms. The TCP model is varied and goodput recorded after 200 seconds of simulation time.

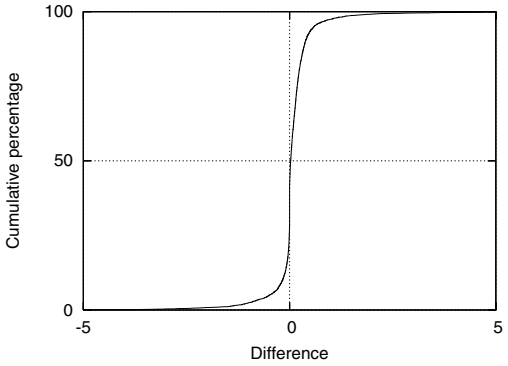
To simulate this range of parameters 112500 independent simulations were run. The simulations were spread over a set of 99 computers. A total of 4.98 CPU-years were spent simulating.

Figure 2 shows comparisons of TCP variants by plotting the difference in measured goodput as a cumulative percentage graph. A point at $x = 1$ on figure 2(a) means the Newreno ns-2 model was measured to have twice the goodput as the Sack ns-2 model.

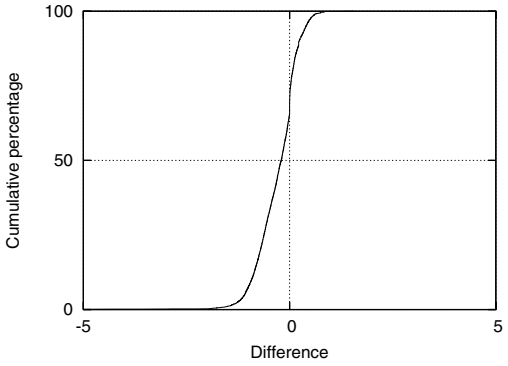
Figure 2(a) shows the comparison of the ns-2 TCP models for Newreno and Sack. The comparison does not favour either model largely, both attain more throughput than the other a small percentage of the time with a slight bias towards Newreno. Greater differences are shown in figures 2(b) and 2(c). Linux 2.4 simulated with NSC attains more goodput than FreeBSD much of the time. This is evident in figure 2(b). The difference between Linux 2.6 and FreeBSD is larger yet, with Linux 2.6 attaining more goodput than FreeBSD approximately 70% of the time, only a very small percentage of the time is more goodput recorded for FreeBSD than Linux 2.6.

These results show how that in even a very simplistic scenario there can be large differences in performance of the TCP implementations studied when using goodput as a metric. Not shown in figure 2 are graphs comparing the ns-2 models with the NSC models. These also show large differences, comparing ns-2's TCP with selective acknowledgements model to Linux 2.6 produces a graph similar to 2(c).

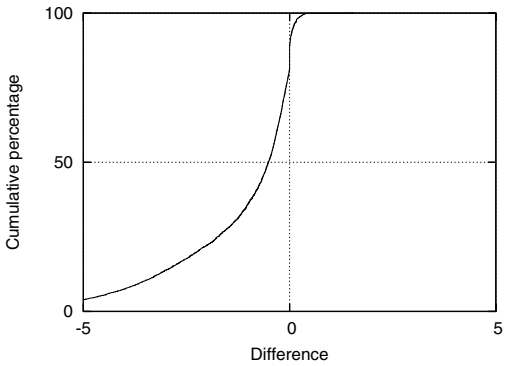
It is evident from viewing the raw data sorted by the difference in goodput that the largest differences are due to extreme circumstances: many flows with small queue sizes and small bandwidths. Often in such cases no goodput is recorded for the ns-2 TCP models of Newreno and Sack, as their connection establishment fails, where the real world implementations are able to connect and send data. Figure 3 shows a visualisation of the mean difference encountered as



(a) ns-2: Newreno vs. ns-2: Sack



(b) NSC: FreeBSD5 vs. NSC: Linux 2.4



(c) NSC: FreeBSD5 vs. NSC: Linux 2.6

Figure 2. TCP performance comparisons with cumulative graphs

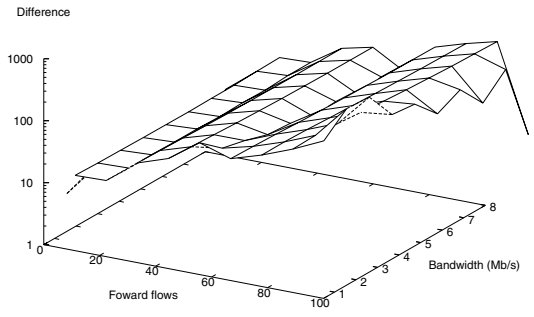


Figure 3. Mean goodput difference

flows and bandwidth is varied. This shows how at a low bandwidth and high amount of flows the difference is the greatest and there is a general trend towards higher differences as the number of flows is increased.

This method of generating large amounts of performance data of real world TCP implementations is something that is extremely difficult and resource intensive without a framework like the Network Simulation Cradle. The cradle, with its multiple TCP implementations, make comparative performance studies of TCP implementations over a range of networks and parameters simple.

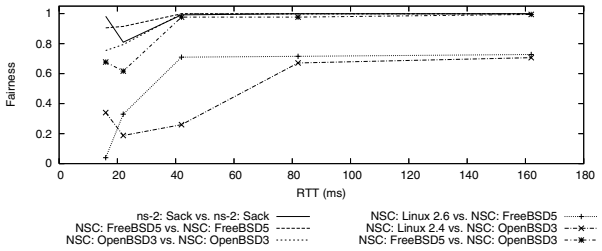
4. Reproduced simulations

This section shows the use of the Network Simulation Cradle in reproductions of simulations and experiments conducted in a range of TCP based research. The results in this section show that using real world TCP implementations in research is feasible for actual research undertaken with TCP simulation and more so that useful results and insights are possible from using such implementations.

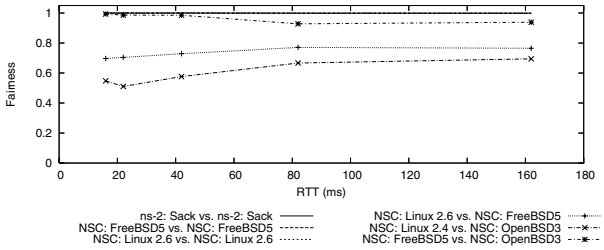
4.1. TCP fairness on high-speed networks

TCP over fast long distance networks is an active research area: TCP increases its window very slowly and is sensitive to packet loss, resulting in low link utilisation on many fast long distance networks. Various schemes have been invented to alleviate this problem, while remaining compatible with TCP. Examples include BIC-TCP [27], FAST TCP [18] and many others. These schemes often have problems with fairness (sometimes exacerbating TCPs inherent RTT unfairness) and convergence times vary [22].

These TCP modifications are tested in simulation, on testbeds and on the Internet. The simulations in this section reproduce experiments conducted on testbeds presented by



(a) 10Mb/s network



(b) 250Mb/s network

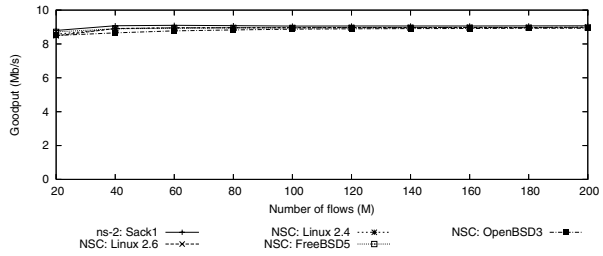
Figure 4. Fairness between two TCP flows

the Hamilton Institute technical report [22]. This report has been cited in research since being published [20, 15] and is noted as a reference for the IETF Transport Modeling Research Group led by Sally Floyd [10].

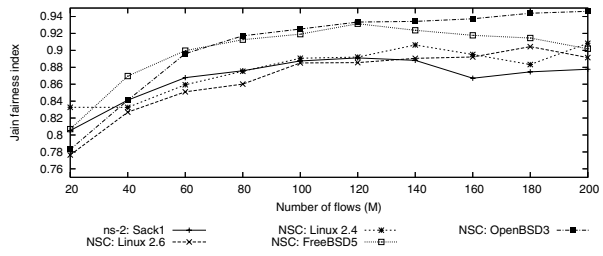
Presented in figure 4 are two graphs that reproduce the "standard TCP" results in figure 6 of [22] and show extra information gained from using the Network Simulation Cradle. The topology used in the experiments is a dumbbell topology. Path propagation delay (and hence round trip time) is varied and the fairness between two TCP flows is measured. The fairness is defined as the ratio of goodput achieved by the two flows after 60 seconds. The queue size is set to 20% of the bandwidth-delay product. Flow start time is jittered by up to one RTT and each set of parameters is simulated with 5 random seeds. The graphs show the mean fairness over the 5 simulations for each data point.

The baseline or "standard TCP" cases in figure 6 of [22] are reproduced in figure 4. The lines on the graphs without points show the results which agree. The fairness measured is near to 1, meaning the two TCP flows equally (fairly) share the link bandwidth. At low RTTs on figure 4(a) the fairness is less stable. As queue size is based on the bandwidth-delay product, when both the RTT and bandwidth are relatively low the queue size on the bottleneck router is also very low (as low as 3 packets when RTT is 16ms). With higher RTTs and/or a higher bandwidth the results are consistent.

These baseline results are expanded by comparing different TCP implementations against each other with the Network Simulation Cradle. The previous results discussed



(a) Goodput



(b) Fairness

Figure 5. 10Mb/s bottleneck, reverse traffic

are for when both TCP flows are from the same implementation. The results on the graphs in figure 4 with lines and points show standard TCP implementations compared against each other. Each of these results could be considered the fairness of standard TCP as used in [22].

Li, Leith and Shorten [22] compare new TCP variations such as BIC TCP and H-TCP against their standard TCP, which is the Linux 2.6.6 TCP implementation. The results in figure 4 show that there are large differences in fairness between standard TCP implementations, as much as between some of the high-speed TCP variants at 10Mb/s. Using real world network stacks in simulation means evaluating this situation is easy and not the prohibitive amount of work it is without NSC.

4.2. Congestion control comparisons

Grieco and Mascolo [13] compare Westwood+, New Reno and Vegas TCP congestion control algorithms and show that Westwood+ is friendly and improves utilisation of wireless links which are affected by non-congestion losses. They simulate a single bottleneck scenario and present goodput and fairness measures in figures 12 and 13 of [13]. They also present recorded goodput under the presence of multiple congestion points in figure 18.

4.2.1 Single bottleneck scenario

The scenario simulated is a simple single-bottleneck or dumbbell topology. A varying number of TCP flows, named M henceforth, send data in the forward direction (the direction the measured data travels), while 10 TCP flows send data in the reverse direction. All flows in M use the same TCP congestion control mechanism. Round trip times are uniformly distributed in the interval $[20 + 230/M, 250]$ms. M ranges from 10 to 200. Simulations last 2000s and the bottleneck link bandwidth is 10Mb/s.

Figure 5(a) shows the aggregate goodput for all M flows as M is increased. This result agrees with the results presented in [13] as once $M = 40$ the goodput levels out at approximately 9Mb/s. Figure 5(b) provides further insight into this result.

Grieco and Mascolo use the Jain Fairness Index to determine fairness between the flows in M. This index is defined in the following equation:

$$J_{FI} = \frac{(\Sigma_{i=1}^{M} b_i)^2}{M\Sigma_{i=1}^{M} b_i^2}$$

Where b_i is the goodput of the i^{th} connection and M are the connections sharing the bottleneck. The index belongs in the interval $[0, 1]$ where 1 is the fairest.

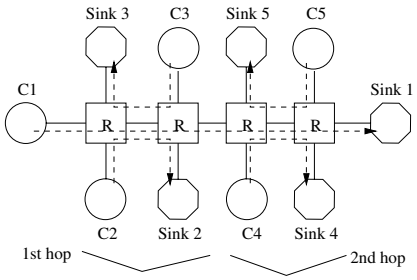
The Jain Fairness Index is plotted in figure 5(b). It is evident that while the TCP models achieve similar goodput, the fairness varies. The general trend of increasing fairness as M increases agrees with the results presented in [13]. This trend is explained by at lower values of M there is a greater variation of RTTs which increases TCPs unfairness.

The results in figure 5(b) further show the difference in TCP implementations. The ns-2 Sack TCP model creates results which are in the right ballpark but using ns-2 abstracted models does not give any knowledge on the range of values the real TCP implementations produce.

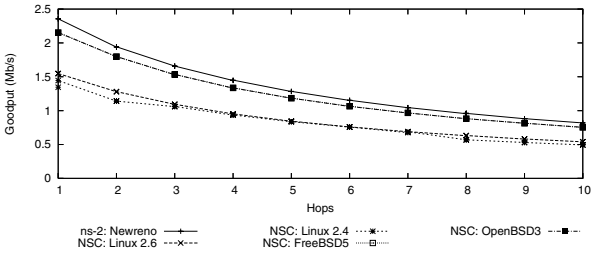
4.2.2 Multiple bottleneck scenario

Figure 6(a) shows the simulation scenario used by Grieco and Mascolo to evaluate the affect of multiple congestion points on TCP congestion control. The number of hops is varied and the goodput of the flow $C1$ is measured. The capacity of the entry/exit links is 100Mb/s with 20ms propagation delay. The capacity of the links connecting the routers is 10Mb/s with 10ms propagation delay. Router queue sizes are set to 125 packets. Simulations last 1000 seconds where the cross traffic is active all the time. The measured flow $C1$ begins after 10 seconds of simulated time. The flows generating cross traffic are controlled by the ns-2 Newreno TCP model.

The results of this simulation scenario are shown in figure 6(b). ns-2's Newreno agent achieves similar results to ns-2's Sack agent and the same trend as presented in figure 18 of [13]. Using real world TCP implementations shows a greater range of performance, with both versions of



(a) Multi-hop simulation scenario (adapted from [13])



(b) TCP goodput as number of congestion points is varied

Figure 6. Multi-hop simulations

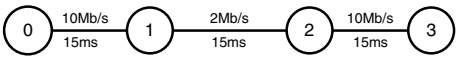
Linux measuring substantially less goodput than FreeBSD and OpenBSD.

4.3. Request latency for a SIP proxy

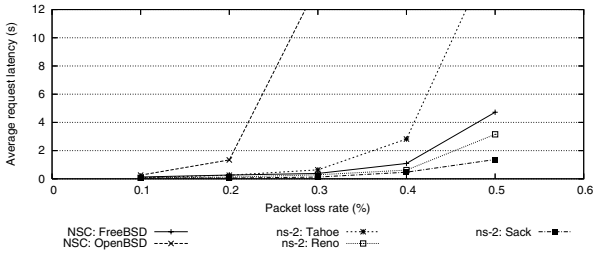
Lulling and Vaughan [23] simulated session initiation protocol (SIP) requests aggregated through a TCP proxy with different TCP variants. They compare Tahoe, Reno and Sack variants of TCP with the ns-2 simulator and show SIP request latency under unfavourable networking conditions such as that found on a best-effort network such as the Internet. The effect head of the line (HOL) blocking has on latency of SIP requests aggregated through one TCP stream is analysed.

Figure 7(a) shows the simulation topology used in the SIP simulations. Nodes 0 and 3 are the SIP proxies using the TCP variants studied. Traffic is generated using a stationary Poisson model to generate the arrival times of 512-byte session establishment requests at node 0. This models a SIP session establishment "INVITE" request arriving from a user to a SIP proxy. The requests are immediately forwarded to the proxy at node 3 and the arrival time recorded. SIP would usually respond with a "100 Trying" response, though this is not modelled here. The TCP MSS is set so a SIP message occupies one TCP segment. TCP delayed acknowledgements are disabled.

This simulation setup is used to test SIP request latency under varying loss conditions. Figure 7(b) shows the average request latency for increasing packet drop rates. The ns-2 TCP models for Tahoe, Reno and Sack are shown and



(a) Simulation scenario (adapted from [23])



(b) Average request latency for increasing loss rates

Figure 7. SIP proxy simulations

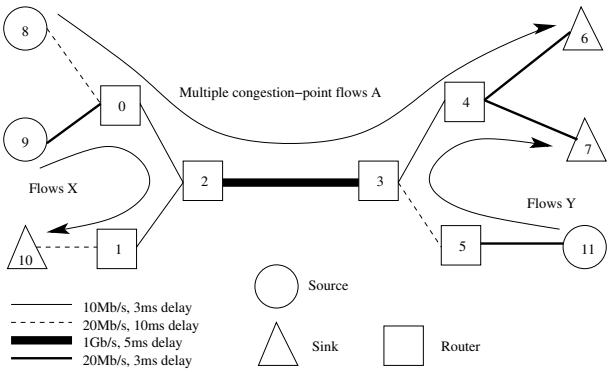
agree with the results presented in figure 9 of [23]. Included are results using the Network Simulation Cradle for the FreeBSD and OpenBSD TCP implementations. Both stacks use TCP selective acknowledgements and are therefore comparable with ns-2's Sack model. Linux is not included because delayed acknowledgements cannot be disabled in the Linux TCP stack — a socket option called QUICKACK disables delayed acks for only a short period, not the entire TCP connection duration.

Lulling and Vaughan analyse the delays under these loss conditions and check whether the latency is within a 2 second bound. This bound is due to ISDN switches used to interconnect within the public switched telephone network (PSTN) which may abandon a call if a reply from a setup attempt is not received with 2 seconds. They are able to conclude that TCP Sack is the only TCP variant that is able to satisfy this bound under all loss rates tested. As figure 7(b) shows, simulating with real world code provides extra insight into this scenario: FreeBSD has an average latency of over 4 seconds at a loss rate of 0.5% where OpenBSD has a very large latency once the loss rate is greater than 0.3%.

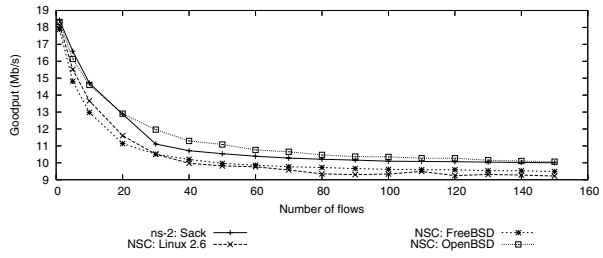
It is uncertain why such a low segment size was chosen for this simulation scenario as Internet MTUs are generally higher [5]. It is possible this is an attempt to reduce delay and jitter. When a higher MTU such as 1500 is used the request latency is much lower than presented in figure 7(b). Conversely, delayed acknowledgements are widely used in real TCP implementations [1] and increase the SIP request latency under random loss.

4.4. Performance over a complex topology

The simple dumbbell topology (otherwise known as a *barbell* topology) is often used when conducting simulation based research [11] even though they are often Internet studies and it is not clear such topologies represent Internet dynamics [11]. This is argued in [2] by analysis of Internet



(a) Simulation scenario (adapted from [2])



(b) TCP goodput over a multi-bottleneck topology

Figure 8. Multi-bottleneck simulations

measurements and creation of a multiple-bottleneck simulation topology that presents results differing largely from a dumbbell topology. The simulation topology studied is shown in figure 8(a).

The number of flows in X and Y of figure 8(a) is fixed at five each. The number of flows in A varies, as does the type of TCP source and sink used for the flows of A. The flows in X and Y use ns-2's Newreno TCP agent with a delayed acknowledgements enabled. Simulations last 300 seconds. Start times for all TCP streams are randomly distributed in the interval $[0, 10.0]$, goodput is measured from when all flows have completed connection establishment and the application has received data. Each set of simulation parameters is simulated 10 times with the random seed varied. The mean of the 10 runs is reported here.

Figure 8(b) presents the results of reproducing the simulation setup of figure 3 in [2]. The Network Simulation Cradle TCP stacks are used in addition to the ns-2 models used in the original study. The results presented here agree with the original research: as the number of flows in A increases the aggregate goodput decreases.

Anagnostakis *et. al.* [2] provide a thorough analysis of this result with different queueing mechanisms, queue sizes, TCP models and round trip times. The results of using the Network Simulation Cradle in figure 8(b) provide another level of validation for this experiment. At the same time this result is further evidence that the Network Simulation Cradle is valid.

There is at times large variation amongst TCP implementations and between real implementations and simulated abstractions, though this is not always true. The research presented in [2] was a pervasive result that was not dependent on the TCP variant or implementation. The results are reproduced independently and expanded on by simulating with real world TCP implementations. These extra simulations further serve to validate the original work.

5. Related work

Various simulators have used real world code to simulate TCP. The approaches are generally limited to one TCP implementation, often a version of BSD. x-sim [4], GloMoSim [28] and OppBSD [3] are all examples of this. The code is manually modified in OppBSD to support multiple instances. None of these approaches contain more than one TCP implementation or provide a methodology for including multiple implementations.

The Lunar [19] project modifies a Linux 2.4 series kernel to allow it to run in user-space with the goal of using it for TCP simulation. The approach to moving the kernel to user space is very similar to that used in the Network Simulation Cradle. This project again only supports one network stack.

NCTUns [26] is a simulator which attempts to make use of a real world network stack for simulation. NCTUns uses the local machines network stack via a tunnel network interface. Tunnel devices are available on most UNIX machines and allow packets to be written to and read from a special device file. To the kernel, it appears as though packets have arrived from the link layer when data is written to the device file. This means the packet will go through the normal TCP/IP processing. When a packet is read from the tunnel device, the first packet in the tunnel interfaces output queue is copied to the reading application.

Recent versions of NCTUns only support Linux and simulation machines are required for every different version of every operating system that is to be simulated. Simulation machines also require kernel patches.

6. Conclusions and future work

We present some results of simulations performed with ns-2 and the Network Simulation Cradle. Earlier work [16] described the implementation and some validation of NSC with only a small set of results. The results discussed here show NSC being used for validation and performance testing of TCP. We show TCP implementations differing amongst themselves and with respect to traditional abstracted simulation models. Simulations and testing done by previous researchers is reproduced with NSC, validating

past work and giving extra insight into the situations studied.

Our results show that simulating TCP with real world implementations is useful:

- there can be a wide variation between TCP implementations;
- TCP implementations differ substantially from simplified models; and
- simulating with real implementations can be used as a validation tool.

Future work could include further comparative performance testing of TCP implementations with NSC. It could be used for quick feedback on TCP performance, measuring metrics such as those defined by the Transport Modeling and Research Group [9]. Results can be gathered at little cost before conducting real network tests. There is also interest in extending the simulation cradle with additional network stacks such as the Open Solaris TCP/IP stack.

References

[1] M. Allman and A. Falk. On the effective evaluation of TCP. *SIGCOMM Computer Communications Review*, 29(5):59–70, October 1999.

[2] K. G. Anagnostakis, M. B. Greenwald, and R. S. Ryger. On the sensitivity of network simulation to topology. In *IEEE Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS)*, Washington, DC, USA, 2002. IEEE Computer Society.

[3] R. Bless and M. Doll. Integration of the FreeBSD TCP/IP-stack into the discrete event simulator OMNeT++. In *Winter Simulation Conference*, pages 1556–1561, December 2004.

[4] L. S. Brakmo and L. L. Peterson. Experiences with network simulation. In *ACM SIGMETRICS*, volume 24, pages 80–90, New York, NY, USA, May 1996. ACM Press.

[5] K. Cho, K. Mitsuya, and A. Kato. Traffic data repository at the WIDE project. In *USENIX, FREENIX Track*, pages 263–270, San Diego, CA, June 2000.

[6] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *SIGCOMM Computer Communications Review*, 26(3):5–21, July 1996.

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

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

[9] S. Floyd. Metrics for the evaluation of congestion control mechanisms. Internet Draft, October 2005.

[10] S. Floyd. The transport modeling research group (tmrg). http://www.icir.org/tmrg/, Accessed 2006.

[11] S. Floyd and E. Kohler. Internet research needs better models. *SIGCOMM Computer Communications Review*, 33(1):29–34, January 2003.

[12] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An Extension to the Selective Acknowledgement (SACK) Option for TCP. RFC2883, July 2000.

[13] L. A. Grieco and S. Mascolo. Performance evaluation and comparison of westwood+, new reno, and vegas tcp congestion control. *SIGCOMM Computer Communications Review*, 34(2):25–38, April 2004.

[14] A. Gurtov and R. Ludwig. Responding to spurious timeouts in TCP. In *IEEE INFOCOM*, volume 3, pages 2312–2322, 2003.

[15] S. Ha, Y. Kim, L. Le, I. Rhee, and L. Xu. A step toward realistic performance evaluation of high-speed TCP variants. In *Fourth International Workshop on Protocols for Fast Long-Distance Networks*, 2006.

[16] S. Jansen and A. McGregor. Simulation with real world network stacks. In *Winter Simulation Conference*, December 2005.

[17] S. Jansen and A. McGregor. Validation of simulated real world TCP stacks. Under submission, available `http://www.wand.net.nz/pubDetail.php?id=218`, 2006.

[18] C. Jin, D. Wei, and S. Low. Fast tcp: Motivation, architecture, algorithms, performance. In *IEEE INFOCOM*, 2004.

[19] C. C. Knestrick. Lunar: A user-level stack library for network emulation. Master's thesis, Virginia Tech, February 2004.

[20] K. Kumazoe, K. Kouyama, Y. Hori, M. Tsuru, and Y. Oie. Can high-speed transport protocols be deployed on the internet? : Evaluation through experiments on JGNII. In *Workshop on Protocols for Fast Long-Distance Networks*, 2006.

[21] S. Ladha, P. D. Amer, A. Caro, and J. R. Iyengar. On the prevalence and evaluation of recent TCP enhancements. In *IEEE Global Telecommunications Conference (GLOBECOM)*, volume 3, pages 1301–1307, 2004.

[22] Y.-T. Li, D. Leith, and R. N. Shorten. Experimental evaluation of TCP protocols for high-speed networks. Technical report, Hamilton Institute, NUI Maynooth, 2005.

[23] M. Lulling and J. Vaughan. A simulation-based performance evaluation of Tahoe, Reno and SACK TCP as appropriate transport protocols for SIP. *Computer Communications*, 27(16):1585–1593, October 2004.

[24] P. Sarolahti and A. Kuznetsov. Congestion control in linux TCP. In *Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference*, pages 49–62, Berkeley, CA, USA, 2002. USENIX Association.

[25] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/, Accessed 2006.

[26] S. Y. Wang, C. L. Chou, C. H. Huang, C. C. Hwang, Z. M. Yang, C. C. Chiou, and C. C. Lin. The design and implementation of the NCTUns 1.0 network simulator. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 42(2):175–197, 2003.

[27] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *IEEE INFOCOM*. IEEE, 2004.

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