Network Emulation with NetEm

Stephen Hemminger
Open Source Development Lab
shemminger@osdl.org

April 2005

Abstract

Many protocols and applications perform poorly when exposed to real life networks with delay and packet loss. Often, it is costly and difficult to reproduce Internet behavior in a controlled environment. There are tools available for testing, but they are either expensive hardware solutions, proprietary software, or limited research projects.

NetEm is a recent enhancement of the traffic control facilities of Linux that allows adding delay, packet loss and other scenario's. Documentation and discussion of NetEm is maintained at `http://developer.osdl.org/shemminger/netem`. NetEm is built using the existing Quality Of Service (QOS) and Differentiated Services (diffserv) facilities in the Linux kernel.

1 Introduction

Why take a perfectly good fast local area network (LAN) and make it slow and lossy? The main reason is to research protocols and applications that have to run over a Wide Area Network (WAN). The typical Ethernet network has a latency of 100 microseconds and can transfer 100's of megabits per second. A broadband connections are available at varying speeds from 128k to 4Mbits but can have a large latency (of up to 50 milliseconds). An application or protocol only designed for a LAN environment will be unusable when run over across the globe over the Internet.

The motivation behind NetEm is to provide a way to reproduce long distance networks in a lab environment. The first usage was to evaluate new TCP enhancements for Linux 2.6. TCP performance over high speed networks is under active research and the subject of many papers. As the total bandwidth delay product (BDP) becomes large, TCP congestion and buffering heuristics cause unstable response. Many of these researchers use the Web100 project [1] to examine and improve TCP congestion behavior. A major research center for these improvements is the Stanford Linear Accelerator Center (SLAC) testbed [2]. This testing showed the need for several enhancements to TCP/IP that should be integrated into the public kernel.

The existing Linux 2.4 TCP/IP used a TCP Reno [3] congestion control algorithm that becomes unstable over high BDP links. Several alternatives are available, and the following were judged to be stable and desirable to integrate into the generally used kernel:

- Vegas [4] avoids congestion using round trip time (RTT) to estimate connection bandwidth.
- Westwood+ [5] adjusts the congestion window based on measured bandwidth.
- BIC [6] sets the congestion window using binary search.

An additional receiver TCP enhancement that was validated with NetEm was automatic receiver buffer size tuning [7]. On low delay links, a small receive window is needed to reduce latency but over high delay links more buffering is required. The new enhancement adjusts the available receive window based on the sender response time.

A sample of the results of comparing TCP congestion algorithms is shown in Figure 1. The performance of single TCP flow tested us-

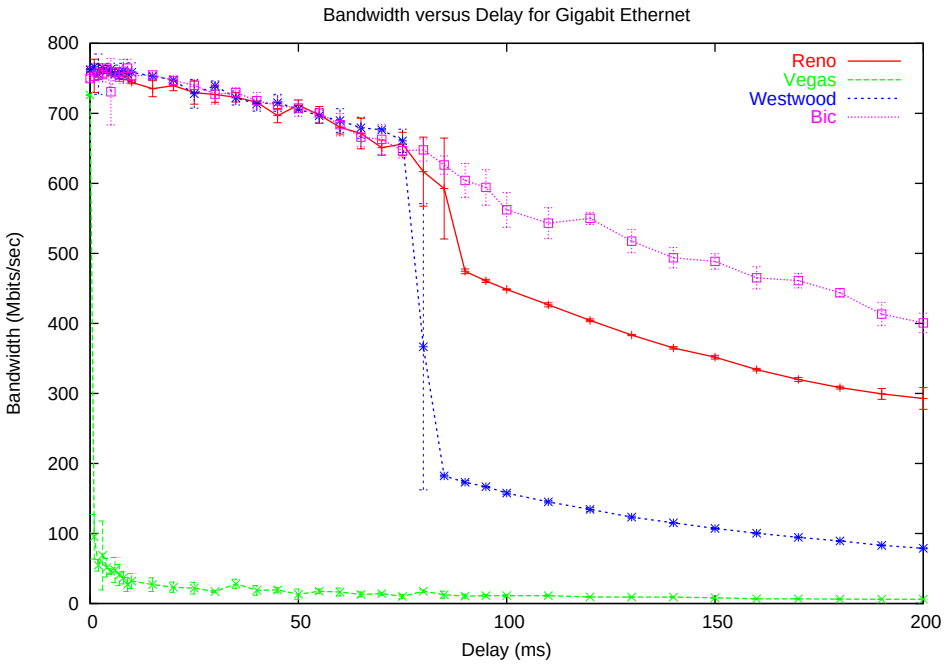


Figure 1: Performance comparison of TCP congestion algorithms

ing Iperf [1] was measured using NetEm to vary the delay [2].

1.1 Background

NetEm has evolved incrementally over the last year. The first version was in the Linux 2.6.7 kernel (called delay). The code was based on the TBF queuing discipline and supported specifying a constant delay. More features were integrated and interface refined based on the contributions of network developers and users.

An alternate to emulation of networks, is complete simulation in a virtual environment. Simulation is a more synthetic approach which involves making a model of the network protocols under test and applying synthetic data to the model. Simulation is more useful when developing a new protocol from scratch because the behavior is more reproducible and not influenced by real world timing details. The prototypical network simulator for research investigations of new protocols is ns-2[8].

An interesting hybrid approach is "umlsim" [9] which uses user-mode Linux (UML) to provide event-driven simulation. This allows testing the standard TCP/IP protocol stack using a pseudo-device that can simulate a network. While useful for testing (and patching) protocol behavior, umlsim is limited because the user-mode kernel runs in a virtual environment that does not have the same performance or timing as real hardware.

Many network emulators exist, but the two most similar in design to NetEm are Dummynet [10] and NIST Net. Dummynet is a standard part of FreeBSD and is implemented as part of the packet filtering mechanism (similar to netfilter). Dummynet is similar to NetEm because it does packet filtering on output. But it is completely self-contained and is not easily extended. Emulab [**?**] emulates complex networks with multiple machines and flows using Dummynet.

NIST Net is a Linux kernel extension that provides complex delay, loss, and other emulation options. Since NIST Net is public domain, many of these functions are re-used in NetEm. However unlike NetEm, NIST Net operates on

[1] http://dast.nlanr.net/Projects/Iperf/

[2] These tests showed problems with bandwidth estimation in the TCP Westwood and Vegas implementation that are being investigated

incoming packets before they reach the protocol stack and uses a high resolution hardware timing source. Like Dummynet, NIST Net does all it's own filtering and queuing.

One other approach that has been used, is emulating network delay using the "tuntap" device [11]. This allows testing network behavior without any kernel changes but the performance is limited because of the extra data copies and context switches. The application must also be able to get access to high resolution timers and real-time response. This solution was rejected as impractical for emulating behavior of high speed networks.

2 Design

NetEm consists of two portions, a small kernel module for a queuing discipline and a command line utility to configure it. The kernel module has been integrated in 2.6.8 (2.4.28) or later, and the command is part of the iproute2 package [12]. Communication between the command and the kernel is done via the Netlink socket interface [13]. Requests are encoded into a standard message format that is decoded by the kernel.

A Graphical User Interface (GUI) has been contributed [14]. This provides an interface similar to NIST Net and is built using PHP and a web interface. This project has exposed the need to have a simple API interface for controlling NetEm using sysfs.

The basic architecture of Linux queuing disciplines is shown in Figure 2. The queuing disciplines exist between the protocol output and the network device. The default queuing discipline is a simple packet FIFO queue.

A queuing discipline is a simple object with two key interfaces. One queues packets to be sent and the other releases packets to the network device. The queuing discipline makes the policy decision of which packets to send based on the current settings. Linux has a rich array of disciplines for queuing, prioritization, and rate control policies. More complex policies are implemented by nesting disciplines together in a manner similar to Unix pipes. For example, the priority queue discipline assigns incoming packets to nested FIFO disciplines based on priority. Each of these FIFO's can be replaced with other disciplines such as Token

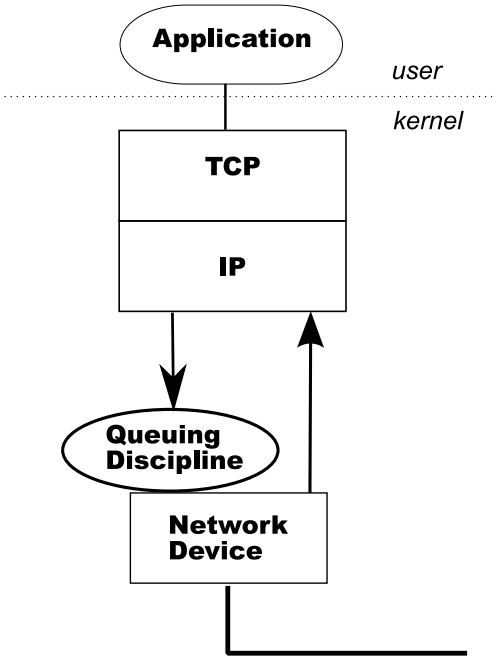


Figure 2: Linux queuing discipline

Buffer Filter (TBF), or Random Exponential Drop (RED).

Internally, NetEm is a classful queuing discipline with two packet queues. One is a private FIFO waiting queue, and the other is a nested queue discipline (typically a FIFO). The enqueue interface takes in packets and timestamps them with a send time, then places them in the holding queue. A timer moves packets from the holding queue to the nested discipline. The dequeue interface gets packets from the nested discipline.

2.1 Parameters

The user specifies the parameters to the network emulator as arguments to the "tc" command. With no additional parameters, NetEm behaves as a basic FIFO queue with no delay, loss, duplication, or reordering of packets.

2.1.1 Packet Delay

Networks do not exhibit constant delay; the delay varies based on other traffic flows contending for the same path. The resulting statistical distribution has one or more peaks and a long tail [15]. The delay parameters are de-

scribed by the average value (μ), standard deviation (σ), and correlation (ρ). By default, NetEm uses a uniform distribution ($\mu \pm \sigma$) but any distribution table can be used [3]. The random distribution is derived from a table that can be generated from a mathematical model or experimental data such as ping times. The iproute2 distribution includes tools to generate a normal distribution, Pareto distribution, and a sample based on experimental data.

2.1.2 Loss

Packet loss is implemented in NetEm by randomly dropping a percentage of the packets before they are queued. Loss is specified in the command interface as a percentage of packets to drop and a correlation value. Internally, th Linux kernel code should not use floating point; therefore the loss percentage is transferred as a scaled 32-bit number. Although under discussion, NetEm does not have the ability to modify packets because at the transport level corrupted packets would be equivalent to lost packets.

2.1.3 Duplication

Networks with reliable hardware don't duplicate packets; but with redundant routes and real hardware some duplication does occur. Duplicating packets with NetEm is done by randomly cloning packets before they are placed in the "waiting list" queue. Duplication is specified as a percentage and correlation value (the same as loss).

2.1.4 Reordering

Packet reordering occurs when packets traverse paths with differing delay. Some current high speed routing equipment use multiple buses and processes internally that create internal alternate paths. Packets are reordered when different processor and buses have varying delay.

NetEm has a simple form of packet reordering in the current implementation as of Linux 2.6.11. The user can specify a "gap" parameter that acts like a random security check at the airport, choosing 1 out of N packets to get additional delay. This is useful for functional testing of the reassembly logic of protocols.

A planned enhancement for NetEm is to provide more complete reordering implementation. The user should be able to specify probability (like duplication and loss), and packet distance [4].

2.2 Rate control

By default NetEm uses a FIFO queuing discipline for the outbound queue but other queues can be used. The queue management utilities and API specify the relationship between queues by numerical handles, for a more complete explanation see the Linux Advanced Routing and Traffic Control (LARTC [16]) guide.

3 Usage example

This section describes an example usage of NetEm. The section describes a test case that emulates the Internet connection between the authors DSL line and the conference website host `http://linux.conf.au`.

The first step in creating this emulation is to sample the net connection using ping. Ping measures the round trip time to a host using the ICMP echo request. This works for this basic experiment but does have flaws. Many Internet Service Providers use traffic control to rate limit ICMP requests, therefore any measurements of packet loss are invalid. Also, there are examples of network providers providing special case express service to ICMP requests to give better perceived customer responsiveness. A better method would be to run a tool like Iperf on both ends of the connection and measure the delay and packet loss using UDP.

An overnight sample of 50,000 pings had an average round trip time of (μ) 234.1 ms, with a deviation (σ) 4.7 ms and a correlation (ρ) of 28%. The resulting ping data was then processed to produce a distribution table [5]. Figure 3 shows a comparison of the measured distribution data with the resulting distribution

[3] NetEm does not yet have the correlation correction found in NIST net

[4] The existing gap implementation is equivalent to 100% probablity of reordering with a constant distance

[5] Distribution tables are text files that reside in `/usr/lib/tc` directory

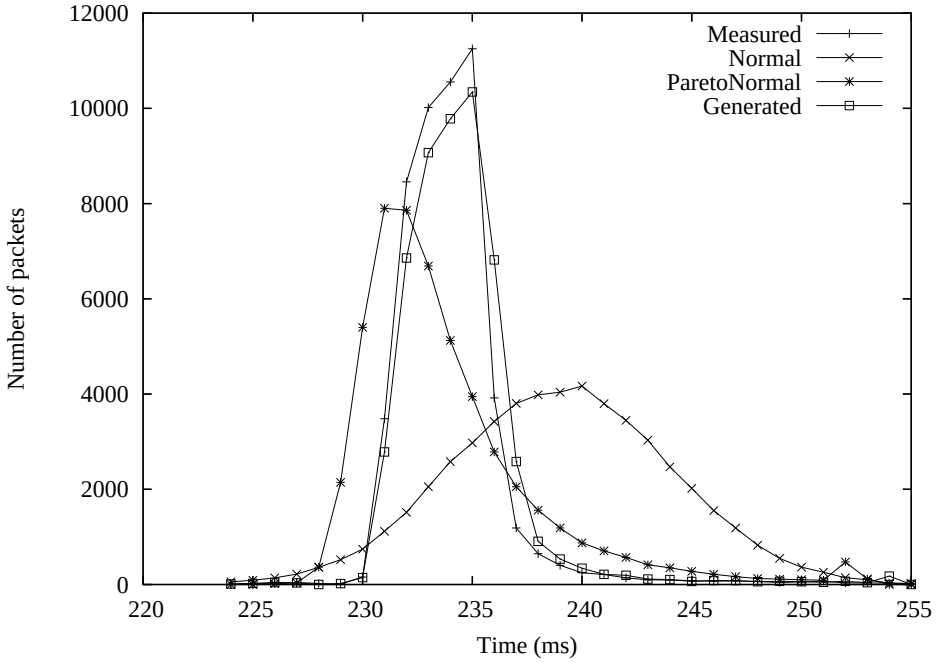


Figure 3: Linux.conf.au ping distribution

produced by NetEm.

The slowest path in the flow is the last-hop DSL connection that has a maximum bandwidth of 1Mbit. Several rate limiting queuing disciplines exist in Linux but the simplest to use is the Token Buffer Filter (TBF). The latency and burst size parameters control the size of the token buffer internal queue. This correspond to the overall buffering capacity of the network being emulated.

The commands to emulate the delay behavior with NetEm are:

```
tc qdisc add dev eth0 root netem handle 1:0 \
   delay 234ms 5ms 28% distribution linux.conf.au
tc qdisc add dev eth0 parent 1:1 handle 10: \
    tbf rate 1mbit latency 200ms burst 128k
```

This works, but affects all traffic that goes out over the network interface "eth0." When testing, it is useful to impact only some traffic (the test case), not all traffic. This is accomplished by using the traffic control filtering features of LARTC. A more complex example that uses the filtering and a priority queue to impact one one service is:

```
tc qdisc add dev eth0 root handle 1: prio
tc qdisc add dev eth0 parent 1:3 handle 30: \
   netem  delay 234ms 5ms 28% \
   distribution linux.conf.au
tc qdisc add dev eth0 parent 30:1 \
    tbf rate 1mbit latency 200ms burst 128k
tc filter add dev eth0 protocol ip parent 1:0 \
    prio 3 u32 match \
    ip dst 10.0.0.3/32 flowid 10:3
```

This creates a nested queue discipline structure as shown in Figure 4 The packets to be sent are filtered so that traffic to IP address 10.0.0.3 is prioritized into a separate queue, and that queue is rate limited and delayed.

4 Limitations

NetEm does its best to a single flow. However, real world networks are quite complex and the emulation inevitably breaks down in some circumstances. Linux timer granularity effects the real-time nature of NetEm, choice of Pseudo-Random Number Generator (PRNG) impacts emulation results, and network devices were not ready for the sudden burst of packets. The parameters that NetEm can control are not enough to describe an arbitrarily complex network with multiple levels of complexity.

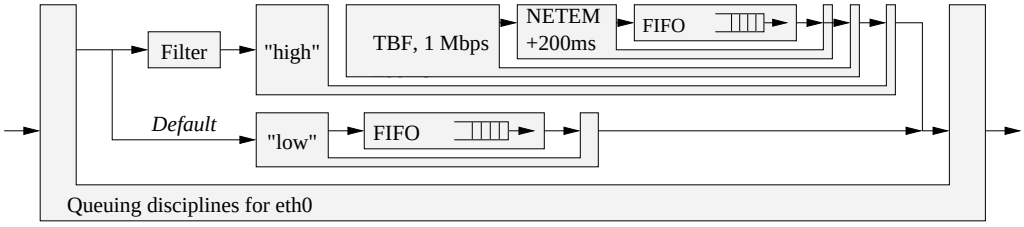


Figure 4: OSDL to linux.conf.au emulation

4.1 Timers

Linux is not a real-time system and this provides some constraints on the performance of a real-time simulator such as NetEm. Kernel timers are limited by the system time tick rate of 1000Hz (1ms) on Linux 2.6 The Linux 2.4 kernel uses a slower 100Hz clock (10ms). Therefore NetEm can not be used to emulate relatively short delay networks of less than 1ms.

This problem is not unique to NetEm, the rate control disciplines also suffer when running over high speed links. It is not possible to limit a 10Gigabit network to 100Mbit with accuracy without higher resolution timers.

NIST Net gets around this by programming one of the alternate time sources available on the PC architecture to provide a high speed clock. This has a performance impact because of the high interrupt load and, more importantly, is not portable to other architectures. There is ongoing work to provide higher resolution timers in Linux[17] that might be useful in the future.

4.2 Random numbers

Several sources of pseudo-random numbers are available in the kernel, but none of them are well-suited to good emulation. The cryptographically secure random number function `get_random_bytes()` cannot be used heavily because it relies on system events to provide entropy. It is intended for providing cryptographic keys and can block when low on entropy until entropy pool is replenished by more system events such as disk seeks, packet arrival, mouse movement, etc.

The networking code has the simple 32-bit PRNG function `net_random()` implemented as a linear congruential generator (LCG). LCG's are not useful for simulators because they produce patterns in the output that can influence results. A better alternative was found using a maximally equidistributed combined Tausworthe generator based on code from GNU Scientific Library 1.5. The decision was made to replace the `net_random()` code because all other places in the kernel would benefit from the faster code and better randomness.

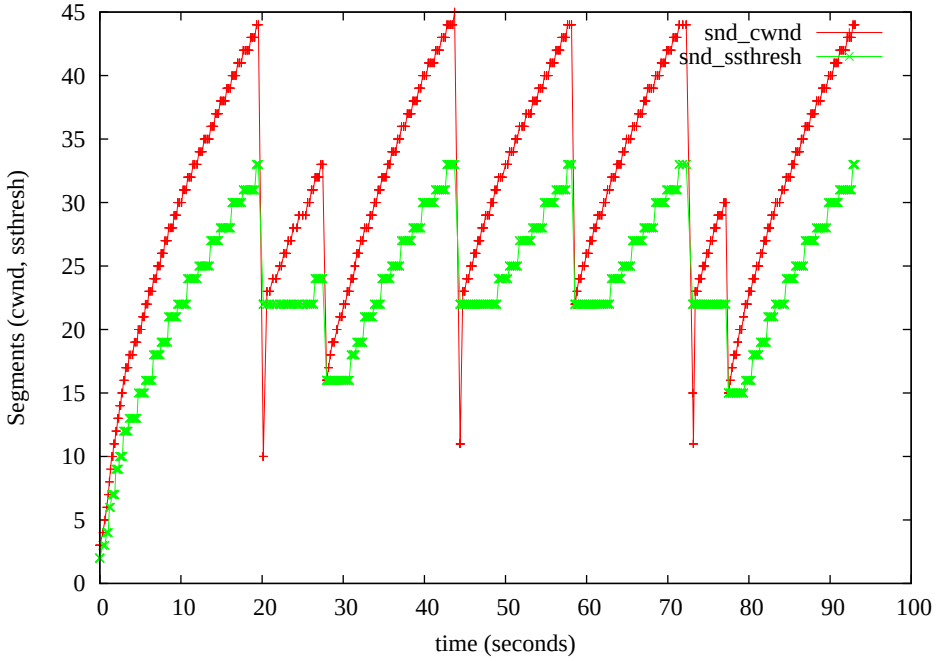
4.3 Network Drivers

Networking devices in Linux have a driver transmit ring that holds a reference to data ready for the hardware to process. This ring has a bounded size, limited by the availability of transmit control blocks. Under high load, NetEm will cause packet bursts to the device (every 1ms). The transmit ring must be sufficiently large to handle this burst or the device must flow control properly. Testing exposed several device drivers that did not flow control properly and would either stop transmitting or spin waiting for transmit ring slots.

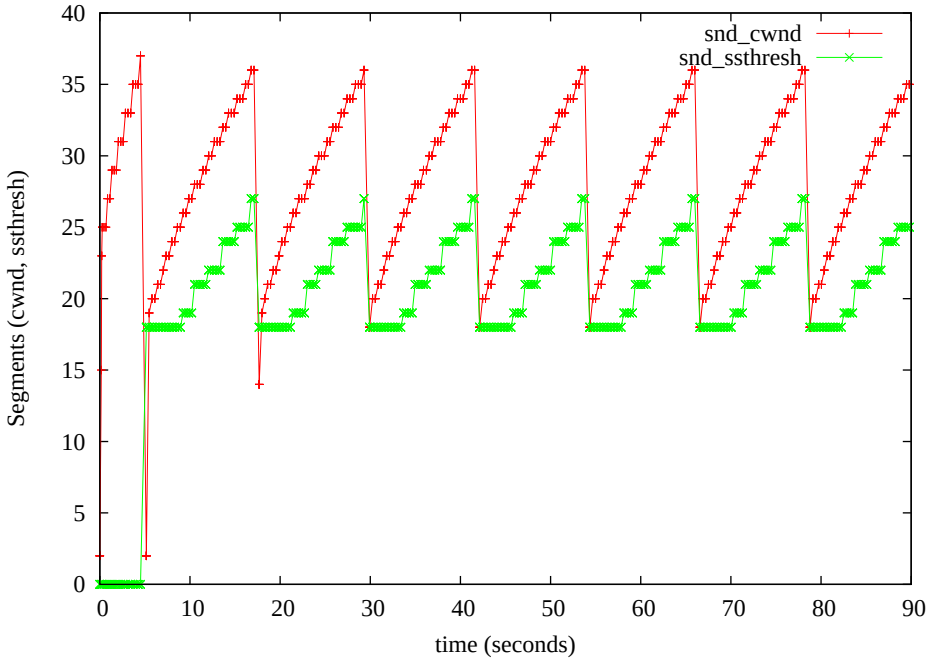
5 Evaluation

NetEm can't simulate the real Internet. The real Internet is very complex and always changing [15] and it is impossible to create one simple model.

Therefore a better question is: how well can NetEm recreate a typical connections behavior? This was tested by constructing a model of a 1Mbit DSL connection and comparing the TCP behavior a during a 90 second transfer. The internal state connection state was monitored using kernel probes [18] to capture the sequence number and congestion window variables.



(a) DSL



(b) NETEM

Figure 5: Comparison of TCP sequence and window over DSL vs NetEm

Figure 5 is a graphical representation of TCP Reno during the transfer. The basic characteristics of this link are a peak bandwidth of 1 Megabit and Round Trip delay of 50ms. The response of TCP over NetEm is very similar to the real DSL link. The spacing of the saw teeth is a function of the size of the router queues which can be adjusted by changing the size of the token buffer filter's parameters.

This result shows that NetEm can be used for test protocols, it not does mean that NetEm is sufficient to cover all types of Internet behavior. The risk is that NetEm reduces a complex multiple input system into a simplistic simulation.

6 Conclusion

NetEm has proved to be a useful tool for testing protocol behavior. It provides the necessary statistical options to emulate real world network response. The author developed it to validate BIC TCP and TCP Vegas for the 2.6 kernel; but many other developers are actively using for testing protocols and applications.

References

[1] Mathis, M.; Heffner, J.; Reddy, R. "Web100: Extended TCP Instrumentation for Research, Education and Diagnosis" *ACM Computer Communications Review, Vol 33, Num 3.* (July 2003) `http://web100.org/docs/mathis03web100.pdf`

[2] Bullot H. and Cotrell L. *TCP Stacks Testbed*, `http://www-iepm.slac.stanford.edu/bw/tcp-eval/`

[3] Floyd, S. and T. Henderson. (1999). RFC 2582 *The NewReno Modification to TCP's Fast Recovery Algorithm.*

[4] Brakmo L. and Peterson L. "TCP Vegas: New techniques for congestion detection and avoidance" *Proceedings of the SIGCOMM '94 Symposium* (Aug. 1994).

[5] Dell'Aera, A.; Grieco, L. A.; Mascolo S. "Linux 2.4 Implementatio of Westwood+ TCP with rate-halving: A Performance Evaluation over the Internet". *IEEE International Conference on Communications (ICC04), Paris, France.* (June 2004).

[6] Xu, L.; Harfoush, K.; and Rhee I. "Binary Increase Congestion Control for Fast Long-Distance Networks." *INFOCOM 2004.* `http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/`

[7] Heffner, J. *High Bandwidth TCP Queueing*, `http://www.psc.edu/~jheffner/papers/senior_thesis.pdf`

[8] ICB, LBNL, VINT. *The Network Simulator – ns-2*, `http://www.isi.edu/nsnam/ns/`

[9] Almesberger, W. (2004). *umlsim – Event-driven simulation for User-Mode Linux*, `http://www.almesberger.net/umlsim/`

[10] Rizzo, L. "Dummynet: a simple approach to the evaluation of network protocols." *ACM Computer Communication Review* (27 January 1997). `http://info.iet.unipi.it/~luigi/ip_dummynet`,

[11] Morton, A. *"Re: simulate delays and packet drops in tcp/udp"*, linux-net@vger.kernel.org (03 Sep 2002) `http://www.zip.com.au/~akpm/packet-delay.tar.gz`

[12] *IP routing utilities*, `http://developer.osdl.org/dev/iproute2`

[13] Salim J.; et al. (2003). RFC 3549 *Linux Netlink as an IP Services Protocol.*

[14] Earl, S. *PHPnetemGUI – a graphical interface for netem*, `http://www.smyles.plus.com/phpnetemgui/`

[15] Floyd, S.; Paxon, V. "Why we don't know how to simulate the Internet". In Proceedings of the 1997 Winter Simulation Conference (Dec. 1997)

[16] Hubert, B. et al. *Linux Advanced Routing & Traffic Control HOWTO*, `http://ds9a.nl/2.4Networking/`

[17] *High Resolution Posix timers*, `http://high-res-timers.sourceforge.net`

[18] Panchamukhi, P. *Kernel debugging with Kprobes*, http://www-106.ibm.com/developerworks/library/l-kprobes.html?ca=dgr-lnxw02Kprobe