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. 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 these 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. Linux 2.4 TCP/IP used the TCP Reno [1] congestion control algorithms that becomes unstable as the total bandwidth delay product (BDP) becomes large. The the Stanford Linear Accelerator Center (SLAC) TCP/IP testbed [2] explored the response and fairness of several alternative congestion control schemes. The several of these were incorporated into the production Linux kernel:

- TCP Vegas [3] avoids congestion using round trip time (RTT) to estimate connection bandwidth.
- TCP Westwood+ [4] adjusts the congestion window based on measured bandwidth.
- BIC TCP [5] sets the congestion window using binary search.
- Automatic receiver side buffer tuning [6].

A sample of the results of comparing TCP congestion algorithms is shown in Figure 1. The performance of single TCP stream was tested using Iperf[1] and NetEm to vary the delay. These tests showed problems with bandwidth estimation in the TCP Westwood and Vegas implementation that are being investigated.

NetEm has evolved incrementally over the last year. The initial version (which was called "delay") was added to the Linux 2.6.7 kernel and only supported specifying a constant delay. After more features were added, the name was changed to "netem" to better reflect the purpose and scope.

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

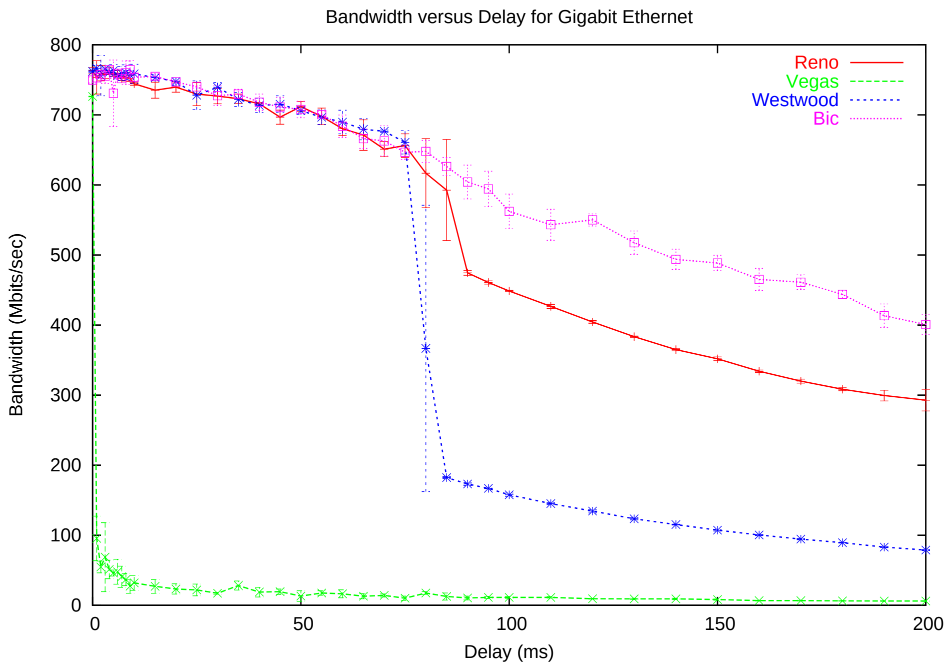


Figure 1: Performance comparison of TCP congestion algorithms

1.1 Related Work

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[7].

An interesting hybrid approach is "umlsim" [8] 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 [9] and NIST Net. Dummynet is a standard part of FreeBSD and is implemented as part of the packet filtering mechanism (similar to netfilter). NetEm and Dummynet both act on packets on the output flow before being sent to the network interface. But it is completely self-contained and can be used to implement differentiated service on FreeBSD but it is not as easily extended as the Linux qdisc architecture. Emulab [10] 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 incoming packets before they reach the protocol stack. Like Dummynet, NIST Net does all it's own filtering and queuing.

One other approach that has been used, is emulating network delay using a simulated network device to transfer packets to an user mode process. A set of test tools using the "tuntap" device is available [11]. These tools allow testing network behavior without any kernel changes but the performance is limited because of the extra data copies and context switches.

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) and the command is part of the iproute2[2] package. Communication between the command and the kernel is done via the Netlink socket interface [12]. Requests are encoded into a netem specific message format and sent/received down the netlink socket.

A Graphical User Interface (GUI)[3] is also available. It provides an interface similar to NIST Net and is built using the Apache web server and PHP scripting language.

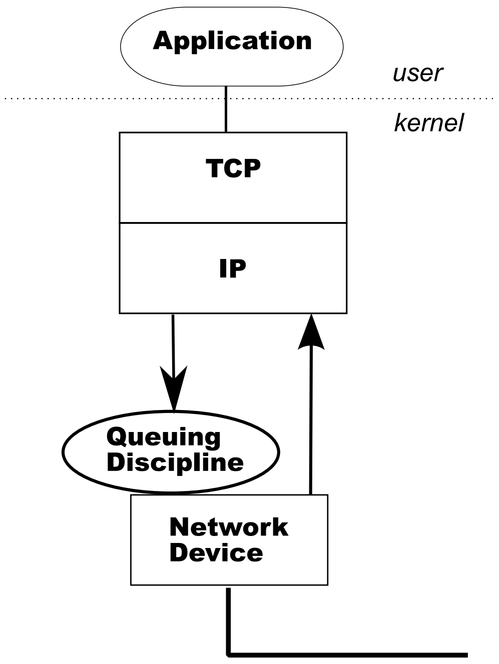


Figure 2: Linux queuing discipline

The basic architecture of Linux queuing disciplines is shown in Figure 2. One (or more) 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 its current state and parameters. Linux already has a rich array of disciplines for queuing, prioritization, and rate control policies.

A classful queuing discipline is a discipline that contains other nested disciplines. More complex policies are implemented by nesting disciplines together in a manner similar to pipes. For example, a priority queue discipline can have multiple sub-queues. Those sub-queues can be simple FIFO's or more complex disciplines such as as Token Buffer Filter (TBF), or Random Exponential Drop (RED).

Internally, NetEm has two queues: one is a private holding 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 for transmit. 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 similar to a FIFO queue with no loss, duplication, or reordering of packets.

2.1.1 Packet Delay

NetEm accepts both constant and random delay parameters. 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 [13]. NetEm describes delay in four possible parameters: the average value (μ), standard deviation (σ), correlation (ρ), and the statistical distribution table. The random value is derived from a table that can be generated from a mathematical model or experimental data such as ping times. By default, NetEm uses a uniform distribution ($\mu \pm \sigma$) but any distribution conversion table. The iproute2 distribution includes tools to generate a normal distribution, Pareto distribution, and a sample based on experimental data.

[2]`http://developer.osdl.org/dev/iproute2`

[3]`http://www.smyles.plus.com/phpnetemgui/`

The correlation parameter controls the relationship between successive pseudo-random values. The correlated pseudo-random number generator uses scaled math to produce a correlated random number x_i based on the pseudo-random values r_i:

$$x_i = r_i * (1 - \rho) + \rho * x_{i-1}$$

This works but is not accurate, it produces less than the desired variance. NIST net adds an empirically derived "correction" factor, but NetEm does not. Better alternatives are being examined for a future version.

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 the correlation between successive random numbers. The command then translates that value to a scaled 32-bit number because it is slow and difficult to use floating point in the kernel. This makes the smallest possible (non-zero) value for loss $2.3e - 10$.

2.1.3 Duplication

Networks with reliable hardware shouldn't duplicate packets; but with real networks redundant routes and bad connections some duplication does occur. To emulate this behavior, NetEm can randomly copy packets before they are placed in the "waiting list" queue. Duplication is specified as a percentage of packets to duplicate and a correlation value (the same as loss).

2.1.4 Reordering

Packet reordering occurs when multiple packets traverse paths with differing delay. Some high speed routing equipment use multiple buses and processors that can create internal alternate paths. This can cause successive packets to pass each other as one is processed by the busier processor than the other.

NetEm has a simple form of packet reordering in the current implementation (2.6.10). 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. Unfortunately, the "ungapped" packets are passed through with no delay therefore this simple reordering mechanism is not useful.

A planned enhancement for NetEm is to provide more complete reordering implementation. The user should be able to specify probability ($\mu\rho$) and packet distance (gap). The existing gap implementation is equivalent to 100% probablity of reordering with a constant distance.

2.2 Rate control

By default, NetEm uses a FIFO queuing discipline for the outbound queue but other policies can be used. The queue management utilities and API specify the relationship between queues by numerical handles, this will be demonstrated in the next section. For a more complete explanation see the Linux Advanced Routing and Traffic Control (LARTC [14]) 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 is a "good enough" estimate for this basic experiment but ISP's may treat ICMP traffic specially by imposing rate limits and/or preferential service. 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. This was possible without recompiling the "tc" command because the tables are not built into the command, instead they are text files

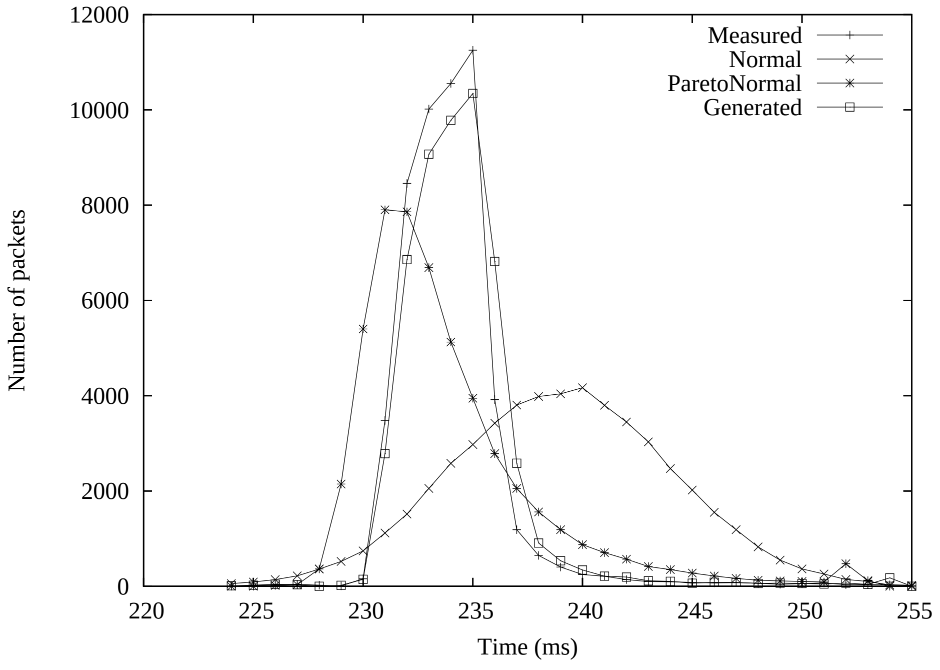


Figure 3: Linux.conf.au ping distribution

in `/usr/lib/tc` directory. Figure 3. shows a comparison of the measured distribution data with the resulting distribution produced by NetEm.

The slowest path in the route to the conference is the last-hop DSL connection that has a maximum bandwidth of 1Mbit. There are several rate limiting queuing disciplines to choose from 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. These parameters 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 \
   latency 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 latency 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 can model a single path through a network, but 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 may not handle the sudden burst of packets.

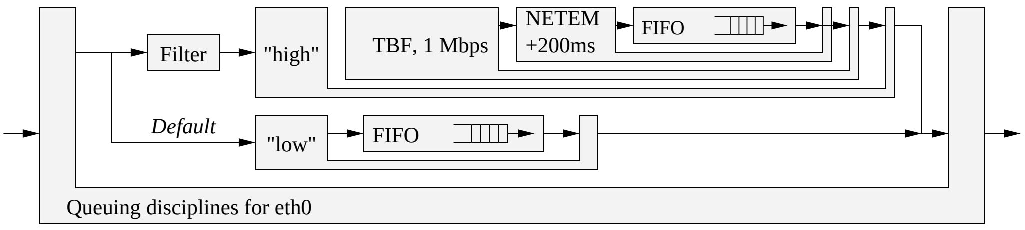


Figure 4: OSDL to linux.conf.au emulation

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.[4] 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[5] 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 the entropy pool is replenished by more external 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]The Linux 2.4 kernel uses a slower 100Hz clock (10ms).

[5]`http://high-res-timers.sourceforge.net`

4.3 Network Drivers

Most 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 and 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 parameters available are not enough to describe an arbitrarily complex network with multiple levels of complexity. The real Internet is very complex and always changing [13] and it is impossible to create one simple model.

Therefore a better question is: how well can NetEm recreate a typical connection's behavior? This was tested by constructing a model of a 1 Mbit DSL (50ms delay) connection and comparing the TCP behavior a during a 90 second transfer. The internal TCP state was monitored using kernel probes [15] to capture the sequence number and congestion window variables.

Figure 5. is a graphical representation of TCP Reno during the transfer. The `snd_cwnd` line shows the congestion window that was used on each output packet. `snd_ssthresh` is the slow start threshold. The congestion window grows until packet loss and then after loss is reset to the slow start threshold. The response of TCP over NetEm is close the real DSL link, and could be improved with more tuning of the parameters. 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 internal queue.

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

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

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

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

[5] 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/`

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

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

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

[9] 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`,

[10] White B. et al. (2002). "An Integrated Experimental Environment for Distributed Systems and Networks" *USENIX Dec 2002.*

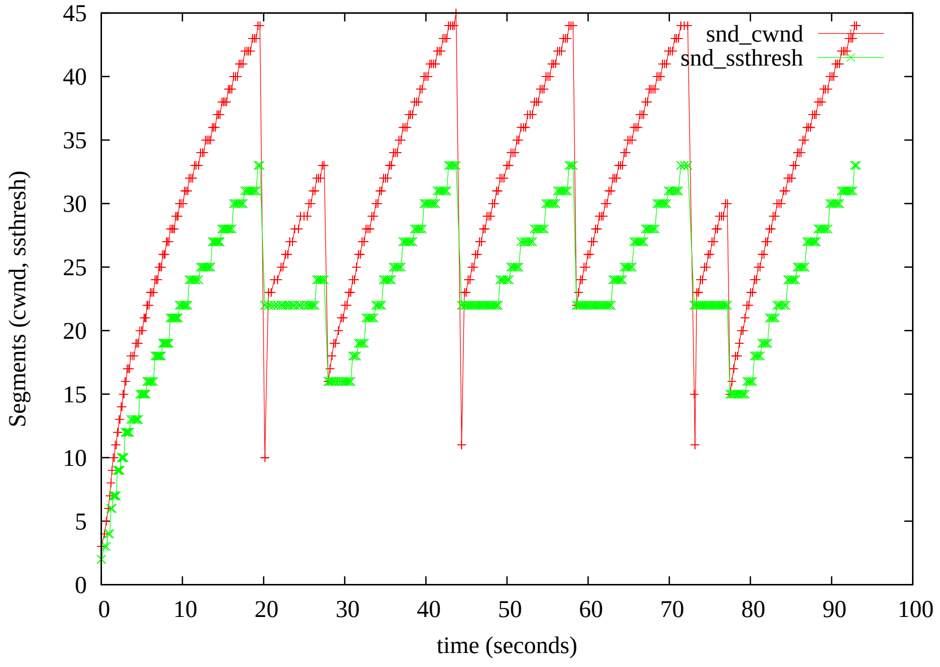
[11] Morton, A. *"Re: simulate delays and packet drops in tcp/udp"*, linux-net@vger.kernel.org (03 Sep 2002) `http://www.zip.com.au/~akpm/packet-delay.tar.gz`

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

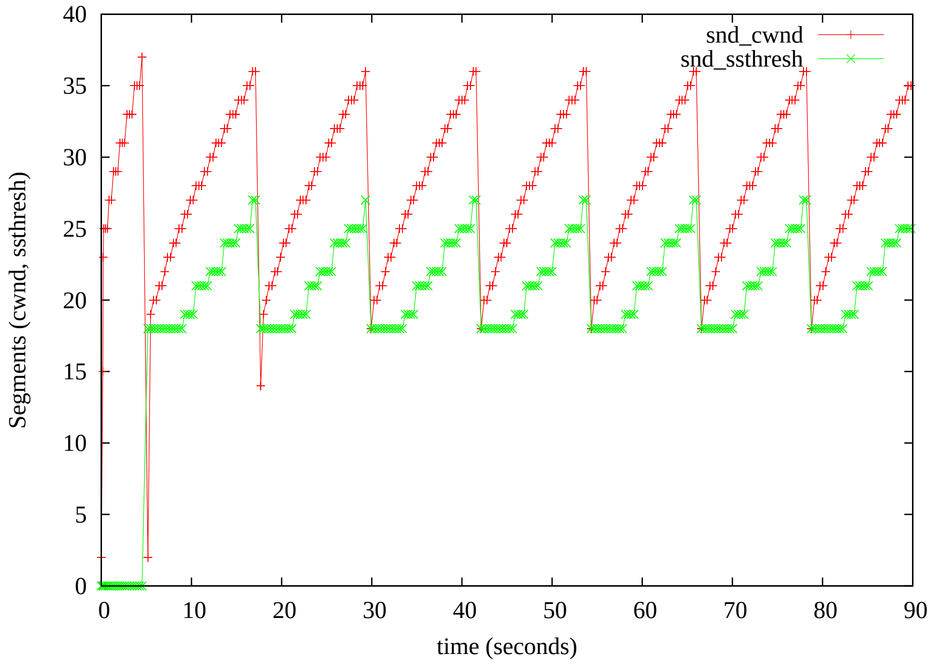
[13] Floyd, S.; Paxon, V. "Why we don't know how to simulate the Internet". In Proceedings of the 1997 Winter Simulation Conference (Dec. 1997)

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

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



(a) DSL



(b) NETEM

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