tcp_testing

TCP testing in Linux is an continuous process. These tests are concerned with the behavior and correctness of the TCP protocol in the Linux kernel.

More will be added to this page soon.

Contents

Tools

Many tools are available for building TCP tests, but some of the basic ones are:

- **Iperf**
- a simple client/server test application.
- **TcpProbe**
- kernel module that creates a way to see TCP endpoint state changes
- **Netem**
- emulate long delays, packet loss etc.

Tuning

When running TCP experiments over high Bandwidth Delay Product [http://en.wikipedia.com/wiki/Bandwidth-delay_product] (BDP) you probably want to change the maximum available TCP window size. TCP tuning is done via wikipedia:sysctl [http://en.wikipedia.com/wiki/sysctl] parameters documented in Net:Ip-sysctl (or Documentation/networking/ip-sysctl.txt) see for more info. At a minimum increase tcp_rmem[2] for receiver and tcp_wmem[2] for sender to twice the BDP.

If the test involves repeated connections, you should also turn off the route metrics:

```
sysctl -w net.ipv4.tcp_no_metrics_save=1
```

Normally Linux will remember the last slow start [http://en.wikipedia.com/wiki/Slow-start] threshold (ssthresh). This modifies the result of the second test, and causes errors.

Experiments

Congestion Control

The purpose of congestion control tests is to observe how the congestion window changes with different network conditions. These tests have lots of possible variables and there is no one “right answer”.

Example

A basic test would be:

1. Start *iperf* server on the receiver

```
iperf -s
```

1. Insert *tcp_probe* module (as root) on sending machine and filter for iperf port. You can change the mode to allow non-root user access

```
modprobe tcp_probe port=5001
chmod 444 /proc/net/tcpprobe
```

1. Capture tcp probe output (on sender) and place in background

```
cat /proc/net/tcpprobe >/tmp/tcpprobe.out &
TCPCAP=$!
```

1. Run *iperf* test on sender for 15minutes

```
iperf -i 10 -t 300 -c //receiver//
```

1. Kill capture process

```
 kill $TCPCAP
```

The tcp probe capture file will contain one line for each packet sent.

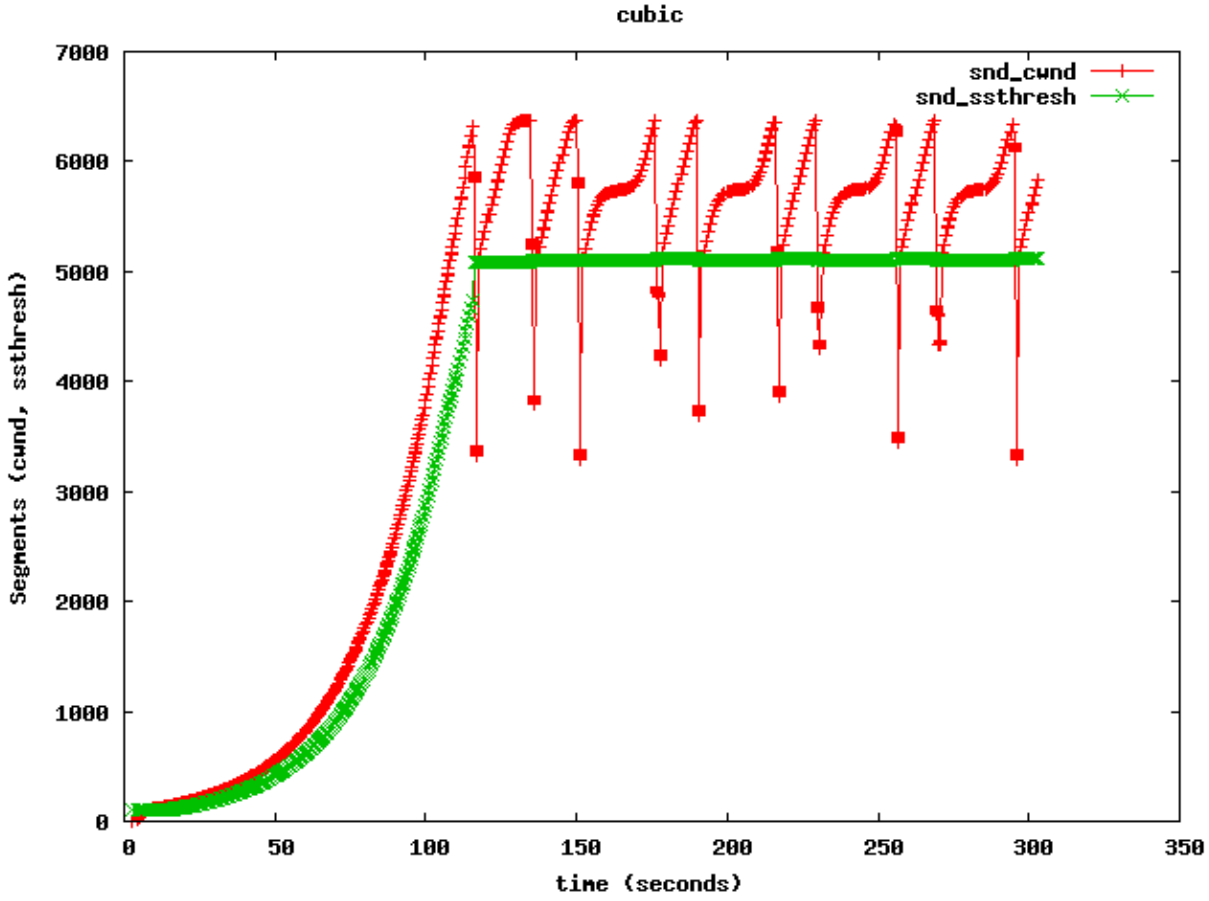
```
0.073678 10.8.0.54:38644 192.168.1.42:5001 24 0xb6b19bb 0xb6b19bb 2 2147483647 5792
^        ^               ^                 ^  ^         ^         ^ ^          ^
|        |               |                 |  |         |         | |          +- [9] Send window
|        |               |                 |  |         |         | +------------ [8] Slow start threshold
|        |               |                 |  |         |         +-------------- [7] Congestion window
|        |               |                 |  |         +------------------------ [6] Unacknowledged sequen
|        |               |                 |  +---------------------------------- [5] Next send sequence #
|        |               |                 +------------------------------------- [4] Bytes in packet
|        |               +------------------------------------------------------- [3] Receiver address:port
|        +----------------------------------------------------------------------- [2] Sender address:port
+-------------------------------------------------------------------------------- [1] Time seconds

* The value of slow start threshold here is (-1) which means it hasn't been determined yet
* Time since Tcp probe was loaded.
```

This text file can be easily filtered and modified with standard tools such as awk [http://en.wikipedia.com/wiki/A_wk] and perl [http://en.wikipedia.com/wiki/Perl] . A common usage is to make a plot of congestion window and slow start threshold over time using gnuplot [http://en.wikipedia.com/wiki/Gnuplot] .

```
$ gnuplot -persist <<"EOF"
set data style linespoints
show timestamp
set xlabel "time (seconds)"
set ylabel "Segments (cwnd, ssthresh)"
plot "/tmp/tcpprobe.out" using 1:7 title "snd_cwnd", \
     "/tmp/tcpprobe.out" using 1:($8>=2147483647 ? 0 : $8) title "snd_ssthresh"
EOF
```

The result should look something like this:



A set of scripts to run test like this and plot are available [1] [http://developer .osdl.org/shemminger/tcp/tcppro be-utils.tar.gz]

Variations

There are many possible factors that can be evaluated in any test. For a good discussion of the issues see the paper: On the effective evaluation of TCP [http://www.sigcomm.org/ccr/archive/1999/oct99 /ccr-9910-allman2.html] and Experimental evaluation of TCP protocols for high-speed networks [http://www.hamilton.ie/net/eval/T oNfinal.pdf] .

- **Algorithim**
- Linux provides many congestion control algorithms now.
- **BDP**
- TCP needs to adapt to a wide range of band-width delay products
- **Loss**
- TCP should recover from moderate loss, but since most congestion control algorithms use loss to estimate BDP; they have trouble distinguishing random packet loss from router queue overflow packet drop.
- **Fairness**
- *Define fairness and how to measure it*
- **Background traffic**
- *Describe how to add background traffic via harpoon here*

Performance

On Linux TCP should be able to fully saturate the network given a sufficiently fast CPU and bus for a single connection. Some useful tools are:

- netperf [http://www.netperf.org/netperf/NetperfPage.htm l] allow for measuring CPU utilization
- Apache benchmark [http://httpd.apache.or g/docs/2.0/programs/ab.html] for measuring lots of web connections
- curl-loader [http://curl-loader .sourceforge.net/] web traffic generation (HTTP/S, FTP/S) with 10-20K and more real HTTP sessions

See also Performance Testing .

Protocol Conformance

There are (expensive) TCP protocol validation suites.

Fuzz testing [http://en.wikipedia.com/wiki/Fuzz_testing] is a useful way to find unexpected bugs in the TCP processing.ISIC [http://isic.sourcefor ge.net] is a suite of utilities to exercise the stability of an IP Stack and its component stacks (TCP, UDP, ICMP et. al.)

External Links

- Web100 project [http://web100.or g/]
- Protocols for Long Distance Networks conference 2007 [http://wil.cs.caltech.edu/pfldnet2007/] 2006 [http://www .hpcc.jp/pfldnet2006/technical.html]
- Hamilton Institute [http://www .hamilton.ie] Experimental testing [http://www .hamilton.ie/net/eval/]
- NCSU [http://ncsu.edu] developers of CUBIC [http://www .csc.ncsu.edu/faculty/rhee/export/bitc p/index_files/Page815.htm] and BIC [http://www .csc.ncsu.edu/faculty/rhee/export/bitc p/], experimental results [2] [http://netsrv .csc.ncsu.edu/highspeed] [3] [http://netsrv .csc.ncsu.edu/convex-ordering]
- Stanford Linear Accelerator Testbed experiments [http://www-iepm.slac.stanford.edu/bw/tcp-eval/] and report [http://dsd.lbl.gov/DIDC/PFLDnet2004/papers/Bullot.pdf]
- Stephen Hemminger development tests [http://developer .osdl.org/shemminger/tcp] results 2.6.19-rc3 [http://developer .osdl.org/shemminger/tcp/2.6.19- rc3] 2.6.19-rc4 [http://developer .osdl.org/shemminger/tcp/2.6.19 -rc4/dsl] pre-2.6.22 [http://developer .osdl.org/shemminger/tcp/net-2.6 .22]

---

networking/tcp_testing.txt · Last modified: 2016/07/19 01:22 (external edit)