Performance Testing and Configurations

About

Discussion of testing performance of FreeSWITCH™ with links to test scenario open source projects.

Measures of Performance

When people say performance it can mean a wide variety of things. In reality performance typically comes down to two bottle necks which are SIP, and RTP. These typically translate into calls per second and concurrent calls respectively. Additionally, high volume systems might experience bottlenecks with database servers running out of connections or even bandwidth when looking up account or configuration data.

Calls per Second (CPS)

Since calls per second is simply a measure of how many calls are being setup and torn down per second the limiting factor is the ability to process the SIP messages. Depending on the type of traffic you have this may or may not be a factor. There are a variety of components that can contribute to this bottleneck, FreeSWITCH and it's libraries being only some of them.

Concurrent Calls

Using modern hardware concurrent calls is less a limit of SIP but rather the RTP media streaming. This can further be broken down to available bandwidth and the packets per second. The theoretical limit on concurrent calls through a gigabit Ethernet port would be around 10,500 calls without RTCP, assuming G.711 codec and the link-level overheads. Theory is great and all, but in reality the kernel networking layer will be your limiting factor due to the packets per second of the RTP media stream.

Multi-threaded

FreeSWITCH uses threading. In modern linux kernels threading and forking are very similar. Normally the 'top' utility only shows 1 FS process because top by default rolls up all the threads into one entry. The command 'top -H' will show individual threads.

'htop' by default shows you the individual threads.

In FreeSWITCH there are several threads running on just a base idle FreeSWITCH process. Each additional call leg is at least 1 more thread. Depending on which applications are active on a call, there could be more than 1 thread per call leg.

Configurations

Recommended Configurations

A 64-bit CPU running a 64-bit operating system and a 64-bit version of FreeSWITCH is recommended. A bare metal system provides consistent, predictable performance and most importantly for real–time applications like this, a reliable kernel clock for RTP packet timing. With a virtual machine it is difficult to determine where any problems might originate and improper propagation of the hardware clock through the VM host to the guest operating system is not always available so the RTP tests will be rendered meaningless.

Debian linux is the recommended OS, since that's the OS used by the core developers and therefore the best tested. It will work on some other operating systems though.

Recommended ULIMIT settings

The following are recommended ulimit settings for FreeSWITCH when you want maximum performance. Ulimit settings you can add to initd script before do_start().

```
ulimit -c unlimited # The maximum size of core files created.
ulimit -d unlimited # The maximum size of a process's data segment.
ulimit -f unlimited # The maximum size of files created by the shell (default option)
ulimit -i unlimited # The maximum number of pending signals
ulimit -n 999999    # The maximum number of open file descriptors.
ulimit -q unlimited # The maximum POSIX message queue size
ulimit -u unlimited # The maximum number of processes available to a single user.
ulimit -v unlimited # The maximum amount of virtual memory available to the process.
ulimit -x unlimited # ???
ulimit -s 240         # The maximum stack size
ulimit -l unlimited # The maximum size that may be locked into memory.
ulimit -a           # All current limits are reported.
```

Ethernet Tuning in linux

Beware buffer bloat

Prior to the bufferbloat guys coming in and talking to us there was a note in here that one should "set the buffers to maximum." That advice is WRONG on so many levels. To make a long story short, when you're doing real-time media like VoIP you *absolutely do not want large buffers*. On an unsaturated network link you won't notice anything, but when you have a saturated network the larger buffers will cause your RTP packets to be buffered instead of discarded.

So, what should your rx/tx queuelens be? Only you can know for sure, but it's good to experiment. Normally in linux it defaults to 1000. IF you are using a good traffic shaping qdisc (pfifo_fast or SFB or others) AND prioritizing udp/rtp traffic you can leave it alone, but it still is a good idea to lower it significantly for VoIP applications, depending on your workload and connectivity.

Don't use the default pfifo qdisc, regardless. It outputs packets in strict fifo order.

To see your current settings use ethtool:

ethtool settings

```
[root@server:~]# ethtool -g eth0
Ring parameters for eth0:
Pre-set maxima:
RX:             256
RX Mini:        0
RX Jumbo:       0
TX:             256
Current hardware settings:
RX:             256
RX Mini:        0
RX Jumbo:       0
TX:             128
```

These were the defaults on my Lenny install. If you needed to change it you can do this:

ethtool suggested changes

```
[root@server:~]# ethtool -G eth0 rx 128
[root@server:~]# ethtool -g eth0
Ring parameters for eth0:
Pre-set maxima:
RX:             256
RX Mini:        0
RX Jumbo:       0
TX:             256
Current hardware settings:
RX:             128
RX Mini:        0
RX Jumbo:       0
TX:             128
```

There is no one correct answer to what you should set the ring buffers to. It all depends on your traffic. Dave Taht from the Bufferbloat project reports that, based on his observations and experiences and papers such as http://www.cs.clemson.edu/~jmarty/papers/bittorrentBroadnets.pdf , that at present in home systems it is better to have no more than 32 unmanaged TX buffers on a 100Mbit network. It appears on my Lenny they are 32/64:

One man's buffer settings

```
[root@server:~]# ethtool -G eth0 rx 32 tx 32
[root@server:~]# ethtool -g eth0
Ring parameters for eth0:
Pre-set maximums:
RX:             256
RX Mini:        0
RX Jumbo:       0
TX:             256
Current hardware settings:
RX:             32
RX Mini:        0
RX Jumbo:       0
TX:             64
```

You'll note you can't with this driver reduce the TX buffer to a more optimum level!! This means that you will incur nearly a 10ms delay in the driver alone (at maximum packet size and load) on packets if you are on a 100Mbit network.

(similarly, a large TXQUEUELEN translates to lots of delay too)

On a gigibit network interface, the default TX queues and TXQUEUELEN are closer to usable, but still too large.

Having larger RX buffers is OK, to some extent. You need to be able to absorb bursts without packet loss. Tuning and observation of actual packet loss on the receive channel is a good idea.

And lastly, the optimum for TX is much lower on a 3Mbit uplink than a 100Mbit uplink. The debloat-testing kernel contains some Ethernet and wireless drivers that allow reducing TX to 1.

TCP/IP Tuning

For a server that is used primarily for VoIP, TCP Cubic (the default in Linux) can stress the network subsystem too much. Using TCP Vegas (which ramps up based on measured latency) is used by several FreeSWITCH users in production, as a "kinder, gentler" TCP for command and control functions.

To enable Vegas rather than Cubic you can, at boot:

```
modprobe tcp_vegas
```

```
echo vegas > /proc/sys/net/ipv4/tcp_congestion_control
```

--- Some interesting comments about tcp_vegas at http://tomatousb.org/forum/t-267882/

FreeSWITCH's core.db I/O Bottleneck

On a normal configuration, core.db is written to disk almost every second, generating hundreds of block-writes per second. To avoid this problem, turn /usr /local/freeswitch/db into an in-memory filesystem. If you use SSDs, it is CRITICAL that you move core.db to a RAM disk to prolong the life of the SSD.

On current FreeSWITCH versions you should use the documented "core-db-name" parameter in switch.conf.xml (simply restart FreeSwitch to apply the changes):

```
<param name="core-db-name" value="/dev/shm/core.db" />
```

Otherwise you may create a dedicated in-memory filesystem, for example by adding the following to the end of /etc/fstab

fstab Example

```
#
# Example of /etc/fstab entry (using default size)
#
tmpfs /usr/local/freeswitch/db tmpfs defaults 0 0
#
# To specify a size for the filesystem use the appropriate mount(1) option:
#
# tmpfs /usr/local/freeswitch/db tmpfs defaults,size=4g 0 0
#
```

To use the new filesystem run the following commands (or the equivalent commands for your OS):

```
mount /usr/local/freeswitch/db
/etc/init.d/freeswitch restart
```

An alternative is to move the core DB into an ODBC database, which will move this processing to a DBMS which is capable of handling large numbers of requests far better and can even move this processing onto another server. Consider using freeswitch.dbh to take advantage of pooling.

Stress Testing

KNOW

IF YOU DO NOT UNDERSTAND HOW TO STRESS TEST PROPERLY THEN YOUR RESULTS WILL BE WORTHLESS.

Using SIPp is part dark art, part voodoo, part Santeria. YOU HAVE BEEN WARNED

When using SIPp's uas and uac to test FreeSWITCH, you need to make sure there is media back and forth. If you just send media from one sipp to another without echoing the RTP back (-rtp_echo), FS will timeout due to MEDIA_TIMEOUT. This is to avoid incorrect billing when one side has no media for more than certain period of time.

See Also

FreeSWITCH performance test on PC Engines APU — Stanislav Sinyagin tests FreeSWITCH™ transcoding performance with only one test machine

Real-world observations — Post measurements of your experience at using FreeSWITCH™ in a real-world configuration, not a stress test.

SSD Tuning for Linux — special considerations for systems using Solid State Drives for storage

SIPp — Open source test toll and traffic generator for SIP

SIPpy Cup — Ben Langfeld contributes this scenario generator for SIPp to simplify the creation of test profiles and especially compatible media

check_voip_call — Henry Huang contributes this project to work with Nagios

http://www.bandcalc.com/ — Bandwidth calculator for different codecs and use cases

http://www.cs.clemson.edu/%7Ejmarty/papers/bittorrentBroadnets.pdf — Paper on buffer sizing based on bittorrent usage