

Contents

- [1 Performance Testing](#)
- [2 Measures of Performance](#)
 - ◆ [2.1 Calls per Second \(CPS\)](#)
 - ◆ [2.2 Concurrent calls](#)
- [3 Configurations](#)
 - ◆ [3.1 Recommended hardware/OS](#)
 - ◆ [3.2 Recommended ULIMIT settings](#)
 - ◆ [3.3 Recommended SIP settings](#)
 - ◆ [3.4 Ethernet tuning in Linux](#)
 - ◆ [3.5 TCP/IP Tuning](#)
 - ◆ [3.6 FreeSWITCH's core.db I/O bottleneck](#)
- [4 Stress Testing](#)
- [5 Some real-world results](#)
- [6 Also See](#)

Performance Testing

This section will provide links to specific tests performed by the core developers and community.

[Long term performance testing \(chevymanjosh\)](#)

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.

Calls per Second (CPS)

Since the 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 bottle, FreeSWITCH and it's libraries being only some of them.

Concurrent calls

Performance_testing_and_configurations

Using modern hardware concurrent calls doesn't seem to be a limit of SIP but purely the RTP. Which can further be broken down to the actual volume of the bandwidth and the packets per second. The theoretical limit on concurrent calls with a gigE port would be around 10,500 calls without RTCP, assuming G.711, 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 seconds RTP generates.

Configurations

Recommended hardware/OS

A 64-bit CPU running a 64-bit OS and a 64-bit version of FreeSWITCH is recommended.

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

Recommended ULIMIT settings

The following are recommended ulimit settings for FreeSWITCH when you want maximum performance.

```
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 8192 # The maximum stack size
ulimit -l unlimited # The maximum size that may be locked into memory.
ulimit -a # All current limits are reported.
```

Note that the maximum stack space will allocate 8MB of virtual RAM space to every thread. On 32-bit this can waste large amounts of space and with enough threads you can run out of memory, so the following is recommended on 32-bit:

```
ulimit -s 240 # The maximum stack size (optimized for 32-bit)
```

Note this is virtual, not physical, RAM. On 64-bit the same amount of space is allocated, but it is not noticeable due to the huge amount of space addressable in 64-bit.

Recommended SIP settings

```
Turn off every module you don't need
Turn presence off in the profiles
libsofia only handles 1 thread per profile, so if that is your bottle neck use more profiles
mod_cdr_csv is slower than mod_xml_cdr
Reports of running more than a single instance of FreeSWITCH has helped.
Disable console logging when not needed - loglevel 0
```

Ethernet tuning in Linux

NOTE - 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:

```
[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:                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:

```
[root@server:~]# ethtool -G eth0 rx 128
[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:                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:

```
[root@server:~]# ethtool -G eth0 rx 32 tx 32
[root@server:~]# ethtool -g eth0
Ring parameters for eth0:
Pre-set maximums:
```

Performance_testing_and_configurations

```
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, large TXQUEUELEN's translate out to lots of delay too)

On gigE, the default TX queues and TXQUEUELEN are more correct, but still overlarge.

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 receives 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 <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 coe.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" />
```

Performance_testing_and_configurations

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

```
#
# 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.

Stress Testing

IF YOU DO NOT UNDERSTAND HOW TO STRESS TEST PROPERLY THEN DON'T BOTHER
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.

Some real-world results

If you want to see FreeSWITCH get down and dirty with 10001 calls then watch Tony's 2009 ClueCon [video](#).

Please go to the [real-world results page](#) to see what people around the globe do.

Also See

If you are using SSDs you may find [SSD Tuning for Linux](#) helpful.