Increasing the maximum number of TCP/IP connections in Linux

Asked 14 years ago Modified 2 months ago Viewed 547k times

254

I am programming a server and it seems like my number of connections is being limited since my bandwidth isn't being saturated even when I've set the number of connections to "unlimited".

How can I increase or eliminate a maximum number of connections that my Ubuntu Linux box can open at a time? Does the OS limit this, or is it the router or the ISP? Or is it something else?

linux networking linux-kernel

Share Follow

edited Dec 20, 2019 at 15:52

red0ct
4,665 3 17 43

asked Jan 4, 2009 at 7:35
jbu

2 @Software Monkey: I answered this anyway because I hope this might be useful to someone who actually is writing a server in the future. – derobert Jan 4, 2009 at 8:05

1 @derobert: I saw that +1. Actually, I had the same thought after my previous comment, but thought I would let the comment stand. – Lawrence Dol Jan 4, 2009 at 8:48

5 Answers

Sorted by: Highest score (default)

492

Maximum number of connections are impacted by certain limits on both client & server sides, albeit a little differently.

On the client side: Increase the ephermal port range, and decrease the `tcp_fin_timeout`

To find out the default values:

```
sysctl net.ipv4.ip_local_port_range
sysctl net.ipv4.tcp_fin_timeout
```

The ephermal port range defines the maximum number of outbound sockets a host can create from a particular I.P. address. The `fin_timeout` defines the minimum time these sockets will stay in `TIME_WAIT` state (unusable after being used once). Usual system defaults are:

- `net.ipv4.ip_local_port_range = 32768  61000`
- `net.ipv4.tcp_fin_timeout = 60`

This basically means your system cannot consistently guarantee more than `(61000 - 32768) / 60 = 470` sockets per second. If you are not happy with that, you could begin with increasing the `port_range`. Setting the range to `15000 61000` is pretty common these days. You could further increase the availability by decreasing the `fin_timeout`. Suppose you do both, you should see over 1500 outbound connections per second, more readily.

To change the values:

```
sysctl net.ipv4.ip_local_port_range="15000 61000"
sysctl net.ipv4.tcp_fin_timeout=30
```

The above should not be interpreted as the factors impacting system capability for making outbound connections per second. But rather these factors affect system's ability to handle concurrent connections in a sustainable manner for large periods of "activity."

Default Sysctl values on a typical Linux box for `tcp_tw_recycle` & `tcp_tw_reuse` would be

```
net.ipv4.tcp_tw_recycle=0
net.ipv4.tcp_tw_reuse=0
```

These do not allow a connection from a "used" socket (in wait state) and force the sockets to last the complete `time_wait` cycle. I recommend setting:

```
sysctl net.ipv4.tcp_tw_recycle=1
sysctl net.ipv4.tcp_tw_reuse=1
```

This allows fast cycling of sockets in `time_wait` state and re-using them. But before you do this change make sure that this does not conflict with the protocols that you would use for the application that needs these sockets. Make sure to read post "Coping with the TCP TIME-WAIT" from Vincent Bernat to understand the implications. The `net.ipv4.tcp_tw_recycle` **option is quite problematic for public-facing servers as it won't handle connections from two different computers behind the same NAT device**, which is a problem hard to detect and waiting to bite you. Note that `net.ipv4.tcp_tw_recycle` has been removed from Linux 4.12.

On the Server Side: The `net.core.somaxconn` value has an important role. It limits the maximum number of requests queued to a listen
ility, bump it up from default 128 to something like 128 to 1024. Now you can take
g variable in your application's listen call, to an equal or higher integer.

a role to play. Default values are 1000, so bump them up to 5000 or even more if your

/rc.local

acklog and `net.ipv4.tcp_max_syn_backlog`. Their default values are 1000 and 1024

Now remember to start both your client and server side applications by increasing the FD ulimts, in the shell.

Besides the above one more popular technique used by programmers is to reduce the number of *tcp write* calls. My own preference is to use a buffer wherein I push the data I wish to send to the client, and then at appropriate points I write out the buffered data into the actual socket. This technique allows me to use large data packets, reduce fragmentation, reduces my CPU utilization both in the user land and at kernel-level.

Share Follow

edited Sep 26, 2018 at 13:37 — Tombart — 29.5k 15 122 131

answered Oct 13, 2010 at 12:49 — mdk — 6,085 1 24 29

4 Brilliant answer! My problem was a bit different, ie I was trying to move session info from an application level session storage to redis via PHP. For some reason, I could not add more than 28230 sessions without adding lots of sleep in one go, with no errors seen either in php or on redis logs. We broke our heads on this for an entire day till I thought maybe problem is not with php/redis but in the tcp/ip layer connecting the two and came to this answer. Managed to fix the issue in no time after that :) Thanks a lot! – s1d Apr 1, 2013 at 7:10

34 Don't forget that we are always talking about IP+port. You can have "unlimited" sockets open to port XY from many different IPs. Limit of 470 applies to concurrent opened sockets to the same IP only. Another IP can have its own 470 connections to the same ports. – Marki555 Jan 22, 2014 at 20:15

6 @Marki555: Your comment is VERY CORRECT. Applications developed for generating and sustaining a large number of outbound connections, must have an "awareness" of available IPs for creating outbound connections, and must then appropriately bind to these IP addresses using some kind of a "round-robin algorithm", and maintain a "scoreboard". – mdk Jan 24, 2014 at 9:42

8 This answer has mistakes. First of, net.ipv4.tcp_fin_timeout is only for the FIN_WAIT_2 state (cs.uwaterloo.ca/~brecht/servers/ip-sysctl.txt). Secondly, as @Eric said, "470 sockets at any given time" is not correct. – Sharvanath Jun 30, 2014 at 5:37

Can you give some commentary regarding this answer? stackoverflow.com/a/2332756/582917 – CMCDragonkai May 15, 2015 at 6:36

@willbradley Don't use quote formatting for text that isn't quoted. – user207421 Nov 26, 2015 at 4:00

@EJP thanks, that was there for a long time, I was just cleaning it up: stackoverflow.com/revisions/3923785/2 – willbradley Dec 1, 2015 at 16:43

3 @mdk : I'm not clear with this calculation part `(61000 - 32768) / 60 = 470 sockets per second` . Can you please elaborate this? – Tom Taylor Dec 6, 2017 at 15:08

I'm testing a simple http server called on localhost, and I see that response time grows when there are more concurrent connections. Does any of this apply to the reason behind that? I'm running a "non-blocking" Akka Http server - I assume it runs on multiple threads/cores and even a trivial endpoint encounters response time degradation on minute "concurrent" connection changes - tested using "ab -c [1,2,3,4,5,10,20,50,100] -n 100000 (-k for all cases) localhost:11111/ping" (returning "returning code 200") – Avba Dec 9, 2018 at 19:02

I think the (61000 - 32768) / 60 = 470 sockets per second comes from: 61000 - 32768 is the default port range for random ports. How many ports can clients use 60 seconds is the time for TCP_WAIT (AFAIK it can't be changed) Therefore, a client could only hold ~30K connections in TIME_WAIT state per minute. Under 500 per second. – Radu Gheorghe Dec 13, 2018 at 9:57

Why should ulimit be increased on the client side also? – Jenna Kwon Oct 21, 2020 at 18:20

This is a great answer. The only thing I have to add is that you should not increase Linux defaults unless you really really need it and know exactly what you are doing. Some of them are calculated dynamically on kernel startup depending on your system resources and they are pretty relaxed as they are. Asking for even more has a potential cause all sorts of problems especially if you try to max out a few of them at once and actually try to use the extra resources. – mrKirushko Mar 1, 2021 at 17:44

@mdk could you please take a look at my question? stackoverflow.com/questions/73757107/socket-io-scaling-issue – Sujith S Manjavana Sep 25, 2022 at 14:55

70

There are a couple of variables to set the max number of connections. Most likely, you're running out of file numbers first. Check `ulimit -n`. After that, there are settings in `/proc`, but those default to the tens of thousands.

More importantly, it sounds like you're doing something wrong. A single TCP connection ought to be able to use all of the bandwidth between two parties; if it isn't:

- Check if your TCP window setting is large enough. Linux defaults are good for everything except really fast inet link (hundreds of mbps) or fast satellite links. What is your bandwidth*delay product?
- Check for packet loss using ping with large packets (`ping -s 1472` ...)
- Check for rate limiting. On Linux, this is configured with `tc`
- Confirm that the bandwidth you think exists actually exists using e.g., `iperf`
- Confirm that your protocol is sane. Remember latency.
- If this is a gigabit+ LAN, can you use jumbo packets? Are you?

Possibly I have misunderstood. Maybe you're doing something like Bittorrent, where you need lots of connections. If so, you need to figure out how many connections you're actually using (try `netstat` or `lsof`). If that number is substantial, you might:

- Have a lot of bandwidth, e.g., 100mbps+. In this case, you may actually need to up the `ulimit -n`. Still, ~1000 connections (default on my system) is quite a few.
- Have network problems which are slowing down your connections (e.g., packet loss)
- Have something else slowing you down, e.g., IO bandwidth, especially if you're seeking. Have you checked `iostat -x`?

Also, if you are using a consumer-grade NAT router (Linksys, Netgear, DLink, etc.), beware that you may exceed its abilities with thousands of connections.

I hope this provides some help. You're really asking a networking question.

Share Follow

edited May 5, 2021 at 10:56

jamix
5,318 5 26 34

answered Jan 4, 2009 at 8:02

derobert
49.1k 13 93 124

@derbert could you please take a look at my question? stackoverflow.com/questions/73757107/socket-io-scaling-issue – Sujith S Manjavana Sep 25, 2022 at 14:55

21

To improve upon the answer given by @derobert,

You can determine what your OS connection limit is by catting `nf_conntrack_max`. For example:

```
cat /proc/sys/net/netfilter/nf_conntrack_max
```

You can use the following script to count the number of TCP connections to a given range of tcp ports. By default `1-65535`.

This will confirm whether or not you are maxing out your OS connection limit.

Here's the script.

```
#!/bin/sh
OS=$(uname)

case "$OS" in
    'SunOS')
            AWK=/usr/bin/nawk
            ;;
    'Linux')
            AWK=/bin/awk
            ;;
    'AIX')
            AWK=/usr/bin/awk
            ;;
esac

netstat -an | $AWK -v start=1 -v end=65535 ' $NF ~ /TIME_WAIT|ESTABLISHED/ && $4 !~ /127\.0\.0\.1/ {
    if ($1 ~ /\./)
            {sip=$1}
    else {sip=$4}

    if ( sip ~ /:/ )
            {d=2}
    else {d=5}

    split( sip, a, /:|\./ )

    if ( a[d] >= start && a[d] <= end ) {
            ++connections;
            }
    }
    END {print connections}'
```

Share Follow

edited Nov 3, 2022 at 22:09

answered Oct 12, 2012 at 18:36

whitehat237
359 3 6

4 `which awk` is your friend to determine path to awk, SunOS has a link to it as well :) – Panagiotis Moustafellos Jul 9, 2014 at 9:39

3 @PanagiotisM. `which` relies on the program to be in `PATH` in which case you can just use `awk` instead of providing full path. (that said, I'm not sure if the solution in the script is any closer to perfection, but this is not what the script is about). – Michael Krelin - hacker Nov 28, 2014 at 11:35

8 I love how this script goes ballistic to determine `awk` location, but assumes that shell is always `/bin/bash` (pro tip: AIX5/6 doesn't even have bash by default). – kubanczyk Jun 29, 2016 at 7:32

me to have a correct `PATH` but a reasonable alternative could be `/usr/bin/env awk` and
t the location wrong on my Linux system. It's in `/usr/bin/awk` not `/bin/awk` – Wolph Apr 18,

r1642018 Oct 16, 2017 at 2:06

proc/sys/net/netfilter/nf_conntrack* – Goblinhack Apr 4, 2019 at 13:29

10

In an application level, here are something a developer can do:

From server side:

1. Check if load balancer(if you have),works correctly.
2. Turn slow TCP timeouts into 503 Fast Immediate response, if you load balancer work correctly, it should pick the working resource to serve, and it's better than hanging there with unexpected error massages.

Eg: If you are using node server, u can use toobusy from npm. Implementation something like:

```
var toobusy = require('toobusy');
app.use(function(req, res, next) {
  if (toobusy()) res.send(503, "I'm busy right now, sorry.");
  else next();
});
```

Why 503? Here are some good insights for overload: http://ferd.ca/queues-don-t-fix-overload.html

We can do some work in client side too:

1. Try to group calls in batch, reduce the traffic and total requests number b/w client and server.
2. Try to build a cache mid-layer to handle unnecessary duplicates requests.

Share Follow

answered Nov 19, 2014 at 21:26

Kev
111 1 3

1

im trying to resolve this in 2022 on loadbalancers and one way I found is to attach another IPv4 (or eventualy IPv6) to NIC, so the limit is now doubled. Of course you need to configure the second IP to the service which is trying to connect to the machine (in my case another DNS entry)

Share Follow

answered Oct 6, 2022 at 20:46

Filip Strapina
21 4

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.