Linux Network (TCP) Performance Tuning with Sysctl

Article Number: 348 | Rating: 5/5 from 1 votes | Last Updated: Fri, Aug 3, 2018 11:53 AM

Linux Network (TCP) Performance Tuning with Sysctl

16Udostępnij

Submitted by Sarath Pillai on Sun, 11/10/2013 - 19:06

Before the Linux kernel version 1.3.57, there was no mechanism other than recompiling the kernel, if you wanted to modify some system parameters.

And recompiling the kernel for each and every modification you needed was not at all a good idea. Simply because it didn’t offered flexibility, and it was not possible for a normal day to day user to sit and recompile, a kernel for modifying a value to his required one.

Hence there was a need to provide a user interface, using which a user can easily modify kernel parameters, at run time, without recompiling the kernel. Hence Sysctl was introduced. Before the introduction of sysctl in Linux. Almost all kernel parameters, were defined as constants. However using sysctl you can modify these constants to fit to your needs.

In this article we will be discussing some of the sysctl parameters, that affects the performance of network. Before getting into the details, let's first see some of the things that can be modified using sysctl.

- Device parameters
- Network parameters
- Firewall behavior
- File system
- NFS

- Processes
- Version details and much more..

A complete list of all sysctl parameters, can be found by running the below command.

?

```
[root@www kernel]# sysctl -a
```

Discussing all of them is beyond the scope of this article. However in the coming days I will surely write about a few more options available in sysctl.

We will be discussing Network related switches in sysctl, which on modification can result in speeding up network substantially.

Most of the Linux users out there are aware of the fact that, whatever you modify in sysctl, it ultimately modify some file in the /proc directory. On a running system you can redirect your required values to files in /proc file system, which will immediately get applied. However its better to always use sysctl to modify kernel parameters.

The primary objective of this article is to understand these network related options available in sysctl, and what exactly they do, as far as networking and communication is concerned. We will discuss what each of these options one by one, and understand how does they fit into the whole picture of networking in Linux.

Let's understand some basics of networking and TCP before we go ahead and fine tune these parameters on our Linux machine.

1. Round Trip Time

This is nothing but the amount of time it takes to send a packet to the receiver and the time took to get an acknowledgement from the receiver. Hence the round trip time is the amount of time it took to send an IP packet and then receive and acknowledgement from the other side. In networking this can be tested with a very highly used command called PING.

?

```
C:Userssarath>ping slashroot.in

Pinging slashroot.in [212.71.233.103] with 32 bytes of data:

Reply from 212.71.233.103: bytes=32 time=130ms TTL=51

Reply from 212.71.233.103: bytes=32 time=130ms TTL=51
```

The time = 130ms (milliseconds). Now, the round trip time for reaching [illegible] center in [illegible] from London it took you 90 milliseconds. [illegible] and then get an acknowledgement back

2.TCP and its connection Type

Transmission Control Protocol (TCP) is a connection-oriented protocol. Now [illegible] because [illegible] confirm that the data was delivered properly to the receiver and then also [illegible]

Read: How a TCP connection is Established

Once a reliable connection is established, then data can flow in both the direction. From sender to receiver and back from receiver to sender. The sender also needs to confirm the proper delivery of the data, by waiting for an acknowledgement from the receiver. Please remember the fact that "Who sends data has nothing to do with who originated the connection".

3. What are segments, packets, and frames

People use these terms (segments, packets, and frames), interchangeably in networking. However they are totally different. You might already know that in networking there are 5 different layers. The first one is Application layer, Transport layer, Network Layer, Data Link Layer, and the finally comes the Physical layer(This physical layer is the layer where data flows through the wire).

Whenever any application sends data to a remote server (For example, you are browsing this web page with your favorite web browser called Mozilla Firefox). While browsing, Application layer is taken care by your web browser and the operating system, and Transport layer is taken care by the TCP suite in your operating system, And network layer is where IP address and receiver address details comes (which is also taken care by the networking stack installed in your operating system), then comes the Data link layer where hardware part is involved like MAC address etc, and then comes the physical layer where the final data is crafted on the wire to be transmitted.

Hence whatever you send to the server from your system travels through these different layers (Each layer adds its own bit of information to the data submitted by the previous layer.).

- Data at Transport Layer is called Segments
- Data at Network Layer is called as Packets
- Data at the last layer is called as Frames(ready to be transmitted over wire)

Now we discussed previously that TCP is a reliable protocol. Its reliable because whatever you send, the receiver must acknowledge that it has got the thing you send. I must say that all the bytes you send must be acknowledged by the receiver. Now what if the receiver does not acknowledge?. If the receiver does not acknowledge, then the sender will resend the bytes those where unacknowledged.

It appears to be quite simple when we talk about sending and getting an acknowledgment back. However setting out to implement this in a reliable manner is a quite tough task, and you need to consider a lot of things.

Things like how much amount of data will be send on a continues base before the sender gets an acknowledgement back. How much data can the receiver handle before its finally being processed by the receiving application.

Those issues related to implementing such a reliable communication protocol is addressed by something called as Flow Control in TCP. Now let's understand what is Flow Control (Don't worry we will configure and fine tune our settings in Linux, once we understand these concepts.)

Flow Control in TCP

So the problem that will be addressed using Flow control in TCP is the proper amount of data that will be send and received. Data send by the sender must not be large enough to overwhelm the receiver.

Now such a control over communication is very much important because we have networks of different speed communicating with each other.

TCP uses something called Sliding window Protocol for managing this flow control. Its working is quite simple to understand. Both receiver and the sender will inform each other about the amount of data it can accept. Now the thing is how is this information shared with each other. There is a field in each TCP segment that is send and received called as "**receive window**" (Please note that we are talking about segments, hence its in the Transport layer). The receiver will mention the amount of data that it can accept or say willing to accept in "**receive window**" field.

The sender on seeing the receive window size mentioned in the the segment sent by the receiver, will make a note of it. Now the sender cannot send more than the receive window size mentioned by the receiver until they are acknowledged. Once the acknowledgement is received, and a new receive window value is send by the receiver, the sender can now send next set of data (again only that amount of data which the receiver has mentioned in the receiver window size.)

If a receiver sends a receive window size of 0, the sender cannot send any more data till an acknowledgement is received for its previous sent data and a new receive window size is send by the receiver.

The sender cannot send any more data until and unless a new receive window size is send by the receiver.

There is always a limitation of what you can include in an IP packet or a TCP segment. The limitation is because of the standard size alloted to each field in a TCP segment, as defined in the protocol specification. TCP is not a very new technology, and was made at the time when networks were really slow compared to the high speed networks we have today.

Hence there was a need to modify or say include and modify some additional features as far as performance is concerned. Hence RFC 1323 was born. It contains details about performance improvements in TCP.

The limitation was that the maximum receive window size that can be included in a TCP frame is 65,535 bytes. Now that is a very low number, if you take todays network speed into consideration.

The new modification came up with something called as window scaling, that increases the limit of receive window size from 65535 bytes to a maximum of 1,073,725,440 bytes (which is very close to 1 Giga byte). To understand this more closely let's dive into a little bit of calculation. This calculation is called as Bandwidth Delay Product.

Bandwidth Delay Product

The term bandwidth delay product in itself is quite self explanatory. Its the product of Bandwidth and Delay caused while communicating between two end points. Now let's see what's it.

The second value with which we will multiply the bandwidth is nothing but the delay caused in sending a packet and then getting an acknowledgement back. We saw how to determine that value in our Round Trip Time section.

Now if you have your bandwidth of 10Mbps, and has a RTT (Round Trip Time) of around 200ms for reaching your target receiver, then the Bandwidth delay product will be.

Bandwidth Delay Product = 2 x 106 b/s x 200 x 10-3 s = 244.14Kilobytes

You can calculate Bandwidth delay product from the link shown below.

Calculate Bandwidth Delay Product

The bandwidth delay product result shows the amount of bytes that must be transmitted, to efficiently use the connection speed. However as our operating system has this default value of 65535 bytes (65 Kilobytes)window size, the connection is not at all efficiently used.

If you calculate 244.14 - 65 = 179 KiloBytes is left unused. So the more Round Trip Time you have the more data needs to be send to fully utilize the link speed(because more delay means you need to send a little bit more data at once, to utilize the bandwidth.). Bandwidth Delay product will go on increasing if the latency (Round Trip Time) is more.

Hence to solve this problem we need to use a higher window size. As mentioned before performance improvements were brought to TCP with a new modification in the form of RFC 1323. Increasing the window size for performance is implemented in the form of something called as TCP Window Scaling.

> Using Window Scaling option in TCP will improve your network throughput and speed, if you have your Bandwidth Delay Product more than 65kilobytes.

Let's go ahead and modify our Linux system's TCP parameters, to by default use this option called TCP Window Scaling. As we discussed in the beginning of this article, enabling this option of window scaling is done by modifying sysctl.conf file.

?

```
net.ipv4.tcp_window_scaling = 1
```

Even after enabling window scaling option, the maximum amount of data that can be send to a receiver without getting an acknowledgement back depends on one more factor. Its called as receive window size. This is the maximum amount of data that a receiver can buffer before being processed by the receiving application.

Now if the receiver's receive window size is smaller, even after setting up window scaling option the sender can only send maximum data size equal to the receive window size configured at the receiver end.

Hence we need to modify the receive window size to a bigger maximum value. This configuration is also made using sysctl.conf file, with the below option.

?

```
net.core.rmem_max = 16777216
```

Apart from receive window size, the sender must also have a higher value in the maximum send window size. You might be thinking why there is a requirement of fixed value of send and receive window size. This is because the sender must keep track of the bytes its sending till it gets an acknowledgment back. Because if the acknowledgement doesn't come, then the sender has to resend the entire bytes (if its in buffer it can resend it. Hence this buffer data is not flushed until an acknowledgement comes.)

Modifying the maximum send window size, is also similar to the way we modified the maximum receive window size.

?

```
net.core.wmem_max = 16777216
```

Now other than the above mentioned maximum values of receive window size and send window size, there is one more setting that the operating system uses which sets these values for different conditions. Let's see that option (this is also set in sysctl.conf file). Let's see the receive window size values first.

?

```
net.ipv4.tcp_rmem = 4096 87380 16777216
```

There are three values in there.

- The first value is the minimum amount of receive window that will be set to each TCP connection, even if the system is under extreme high pressure.
- The default value allocated to each tcp connection
- The third one is the maximum that can be allocated to a TCP connection

> Please don't forget the fact that we are using window scaling option, hence the window size will be dynamic and will go on increasing till the maximum receive window size reaches

Similarly there is send window settings, which is shown below. (The three values in send window settings also denote the three things we discussed above)

?

```
net.ipv4.tcp_wmem = 4096        16384   16777216
```

Hence all these things combined together our sysctl.conf file will look something like the below.

?

```
net.ipv4.tcp_window_scaling = 1
net.core.rmem_max = 16777216
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096        16384   16777216
```

Once you have modified your sysctl.conf file with the above required settings, you can reload the configuration and make it permanent, by the below command.

?

```
sysctl -p /etc/sysctl.conf
```

Alternatively you can also modify the above values on the fly, by redirecting your required values to the required file in /proc. This can be done as shown below.

?

```
echo '16777216' > /proc/sys/net/core/rmem_max
```

?

```
echo '16777216' > /proc/sys/net/core/wmem_max
```

Hope this article was helpful in understanding some concepts behind tuning of TCP in Linux.

Posted - Fri, Aug 3, 2018 11:53 AM. This article has been viewed 11854 times.

Online URL: http://kb.ictbanking.net/article.php?id=348