FreeNAS - Bug #43558

Relax the TCP reassembly queue length limit to improve performance

08/29/2018 11:50 AM - John Kutta

Status:	Done		
Priority:	No priority		
Assignee:	Ryan Moeller		
Category:	OS		
Target version:	11.2-U2		
Seen in:	11.1-U6	**Needs Merging:**	No
Severity:	Medium	**Needs Automation:**	No
Reason for Closing:		**Support Suite Ticket:**	n/a
Reason for Blocked:		**Hardware Configuration:**	
Needs QA:	No	**ChangeLog Required:**	No
Needs Doc:	No		

Description

See discussion on forums at

https://forums.freenas.org/index.php?threads/11-1u6-update-transfer-aborts-on-smb-share-mac-client.69553/
https://forums.freenas.org/index.php?threads/network-issues-after-upgrading-to-11-1-u6.69506/

Two different reports of erratic network issues between FreeNAS and MacOS clients after upgrading to U6, observed in SMB and SCP timeouts and/or network performance dropping to zero, sometimes recovering before dropping to zero again.

Symptoms verified on MacOS 10.13.6 and 10.12.6.

Disappear after rollback to FreeNAS to 11.1-U5, everything else equal.

Related issues:

Related to FreeNAS - Bug #41028: Patch FreeBSD CVE-2018-6922 "Resource exhaus...	**Done**
Related to FreeNAS - Bug #49576: Apple OSX backup very slow, eventually seems...	**Closed**
Related to FreeNAS - Bug #78222: Remove obsolete net.inet.tcp.reass.maxqueuel...	**Ready for Testing**
Has duplicate FreeNAS - Bug #67672: Relax the TCP reassembly queue length lim...	**Closed**
Has duplicate FreeNAS - Bug #71459: TCP Traffic Stops after Hundreds of DupAC...	**Closed**
Copied to FreeNAS - Bug #63180: Relax the TCP reassembly queue length limit t...	**Done**

Associated revisions

Revision 91085034 - 12/04/2018 12:28 PM - Ryan Moeller

[sysctl.conf] Raise max TCP segment queue length

Out-of-order TCP segments are queued in the reassembly queue until the missing in-sequence segments arrive. As a DoS mitigation, the length of this queue is limited, with a default of 100. Since TCP allows a whole window to be in-flight at once, there can be up to 1448 to 11586 segments arriving before the sender expects an ACK, depending on the configured max receive buffer size and negotiated window size. The limit of 100 is not usually an issue in wired networks, because segments seldom arrive out of order. However, reordering and/or losses are more frequent with wireless networks, so the low default limit can cause a lot of segments to be discarded.

Raise the limit to 1448, which allows a full window to be queued in the default configuration, and greatly improves the behavior in configurations where the max receive buffer length has been raised to 16 MiB (for example by the autotune script).

Ticket: #43558

Revision 97233341 - 12/05/2018 02:05 PM - Ryan Moeller

[loader.conf] Raise max TCP segment queue length

Out-of-order TCP segments are queued in the reassembly queue until the missing in-sequence segments arrive. As a DoS mitigation, the length of this queue is limited, with a default of 100. Since TCP allows a whole window to be in-flight at once, there can be up to 1448 to 11586 segments arriving before the sender expects an ACK, depending on the configured max receive buffer size and negotiated window size. The limit of 100 is not usually an issue in wired networks, because segments seldom arrive out of order. However, reordering and/or losses are more frequent with wireless networks and more complex networks, so the low default limit can cause a lot of segments to be discarded.

Raise the limit to 1448, which allows a full window to be queued in the default configuration. Behavior in configurations where the max receive buffer length has been raised to 16 MiB (for example by the autotune script) is also improved, while still mitigating DoS.

Ticket: #43558

Revision 6f79caa8 - 12/05/2018 05:55 PM - Ryan Moeller

tkt-43558: [loader.conf] Raise max TCP segment queue length (#2185)

Out-of-order TCP segments are queued in the reassembly queue until the missing in-sequence segments arrive. As a DoS mitigation, the length of this queue is limited, with a default of 100. Since TCP allows a whole window to be in-flight at once, there can be up to 1448 to 11586 segments arriving before the sender expects an ACK, depending on the configured max receive buffer size and negotiated window size. The limit of 100 is not usually an issue in wired networks, because segments seldom arrive out of order. However, reordering and/or losses are more frequent with wireless networks and more complex networks, so the low default limit can cause a lot of segments to be discarded.

Raise the limit to 1448, which allows a full window to be queued in the default configuration. Behavior in configurations where the max receive buffer length has been raised to 16 MiB (for example by the autotune script) is also improved, while still mitigating DoS.

Ticket: #43558

Revision 74812ca8 - 12/06/2018 10:02 AM - Ryan Moeller

tkt-43558: [loader.conf] Raise max TCP segment queue length (#2185)

Out-of-order TCP segments are queued in the reassembly queue until
the missing in-sequence segments arrive. As a DoS mitigation, the
length of this queue is limited, with a default of 100. Since TCP
allows a whole window to be in-flight at once, there can be up to
1448 to 11586 segments arriving before the sender expects an ACK,
depending on the configured max receive buffer size and negotiated
window size. The limit of 100 is not usually an issue in wired
networks, because segments seldom arrive out of order. However,
reordering and/or losses are more frequent with wireless networks
and more complex networks, so the low default limit can cause a lot
of segments to be discarded.

Raise the limit to 1448, which allows a full window to be queued
in the default configuration. Behavior in configurations where the
max receive buffer length has been raised to 16 MiB (for example by
the autotune script) is also improved, while still mitigating DoS.

Ticket: #43558

Revision 56cfbbe4 - 12/10/2018 09:44 AM - Ryan Moeller

tkt-43558: [loader.conf] Raise max TCP segment queue length (#2185) (#2206)

Out-of-order TCP segments are queued in the reassembly queue until
the missing in-sequence segments arrive. As a DoS mitigation, the
length of this queue is limited, with a default of 100. Since TCP
allows a whole window to be in-flight at once, there can be up to
1448 to 11586 segments arriving before the sender expects an ACK,
depending on the configured max receive buffer size and negotiated
window size. The limit of 100 is not usually an issue in wired
networks, because segments seldom arrive out of order. However,
reordering and/or losses are more frequent with wireless networks
and more complex networks, so the low default limit can cause a lot
of segments to be discarded.

Raise the limit to 1448, which allows a full window to be queued
in the default configuration. Behavior in configurations where the
max receive buffer length has been raised to 16 MiB (for example by
the autotune script) is also improved, while still mitigating DoS.

Ticket: #43558

Revision 01896354 - 12/10/2018 10:51 AM - Ryan Moeller

tkt-43558: [loader.conf] Raise max TCP segment queue length (#2185)

Out-of-order TCP segments are queued in the reassembly queue until the missing in-sequence segments arrive. As a DoS mitigation, the length of this queue is limited, with a default of 100. Since TCP allows a whole window to be in-flight at once, there can be up to 1448 to 11586 segments arriving before the sender expects an ACK, depending on the configured max receive buffer size and negotiated window size. The limit of 100 is not usually an issue in wired networks, because segments seldom arrive out of order. However, reordering and/or losses are more frequent with wireless networks and more complex networks, so the low default limit can cause a lot of segments to be discarded.

Raise the limit to 1448, which allows a full window to be queued in the default configuration. Behavior in configurations where the max receive buffer length has been raised to 16 MiB (for example by the autotune script) is also improved, while still mitigating DoS.

Ticket: #43558

Revision 31b73df4 - 01/09/2019 04:19 AM - Ryan Moeller

Relax the TCP reassembly queue length limit

Ticket: #43558

History

#1 - 08/29/2018 12:23 PM - Dru Lavigne

- Private changed from No to Yes

- Reason for Blocked set to Need additional information from Author

John: please reboot into U6, reproduce, create a debug (System -> Advanced -> Save debug), and attach that to this ticket. You can then reboot back into U5 while the dev investigates.

#2 - 08/29/2018 12:38 PM - John Kutta

I had a look at what that (System -> Advanced -> Save debug) does and although I'd like to help I am not sure I'd like to upload my entire log folder with rolled history including auth etc, on principal grounds if nothing else. Could we narrow it down a little in terms of what you're looking for? Also, seeing as there are other reports it looks like this could be relatively easy for you to reproduce in your own QA environment with a Mac client?

#3 - 08/29/2018 12:41 PM - Dru Lavigne

- Assignee changed from Release Council to Alexander Motin

- Reason for Blocked deleted (Need additional information from Author)

Alexander: can you let John know which freenas-debug switches would be useful to get you started in your investigation?

#4 - 08/30/2018 07:56 AM - Alexander Motin

John, I understand your worries, but can only say that we are marking all tickets with sensitive data as private and delete the data when ticket is closed.

The problem is that so far we have no any information about the problem, so I can't even say whether it is a network issue, file system issue or disk hardware/driver issue. There was, for example, a change in handling of Ethernet jumbo frames, but without debug we don't know whether you even use them. If it is a networking issue, we don't even know what NIC you are using.

What's about `freenas-debug` switches, as I understand `freenas-debug` does not include system logs itself. It just collects different configs and output of different tools. Logs are collected by `ixdiagnose` wrapper around of it. While I would prefer to have everything, including logs, running `freenas-debug -A` and uploading /var/tmp/fndebug content would be a good start.

#5 - 08/30/2018 07:57 AM - Alexander Motin

- Status changed from Unscreened to Blocked

- Reason for Blocked set to Need additional information from Author

#6 - 08/30/2018 12:14 PM - John Kutta

- File debug.tar.gz added

Ok, so here's what I did. I hope it will help.

1. Running 11.1-U5, ran concurrent SMB and SCP transfers of large files from the Mac to FreeNAS a few times, no issues and saturated the gigabit line (as I'm used to).

2. Rebooted into 11.1-U6, did the same thing. The transfer stops abruptly about 1.5GB in. At this point I run freenas-debug -A as you asked for. The transfers subsequently stall and time out.

/var/tmp/fndebug uploaded.

The Mac btw is running MacOS 10.13.6 (latest stable).

#7 - 08/31/2018 07:38 AM - Alexander Motin

- Reason for Blocked changed from Need additional information from Author to Waiting for feedback

I am not completely certain, but one of changes in U6 was introduced limit on size of TCP reorder queue to limit potential of DoS attacks. In provided debug I see that out of 3534794 received TCP packets 12850 were reordered, and 10360 were dropped, possibly due to this limit. Too many bursty drops of incoming packets may significantly affect TCP throughput. To check this guess you may try to increase net.inet.tcp.reass.maxqueuelen sysctl value from its default 100 to something much higher. Aside of slightly opening DoS vulnerability there should be no side effect from setting it very high at least for experiment. Please let us know how will it go.

Are you accessing your server locally on LAN, or there is some complicated topology in between?

#8 - 08/31/2018 10:14 AM - John Kutta

Alexander, I think you nailed it. Under 11.1-U6 I set sysctl net.inet.tcp.reass.maxqueuelen=16384 and rebooted. Have been hammering it pretty hard since with the same Mac client as before and transfers are now as stable as they ever were.

To your question; network topology is not particularly complex - a few switches and a VLAN trunk at one segment along the way. No router involved.

I'll keep that setting for now as a workaround. Clearly a default of 100 is too restrictive at least In my case and empirically MacOS clients are more likely to cause problems.

#9 - 08/31/2018 10:18 AM - Alexander Motin

- Related to Bug #41028: Patch FreeBSD CVE-2018-6922 "Resource exhaustion in TCP reassembly" added

#10 - 09/04/2018 07:37 AM - Dru Lavigne

- Status changed from Blocked to Unscreened

- Reason for Blocked deleted (Waiting for feedback)

#11 - 09/04/2018 12:26 PM - Alexander Motin

- Assignee changed from Alexander Motin to Ryan Moeller

Ryan, could you look on how the reorder queue works, how expiration is handled in it, and why 100 items is so insufficient? Would some bigger number help, or it is a design issue?

#12 - 09/06/2018 12:36 PM - Ryan Moeller

- Status changed from Unscreened to Screened

- Severity changed from New to Medium

FreeNAS is configured to allow up to a 2 MiB TCP receive window. The receive window size starts at 64 kiB and auto-scales up, or can be explicitly set by an application. A queue length of 100 at 1500 MTU allows for less than 150 kiB of that to be queued before packets start being discarded.

The receive queue is considered full when the length reaches or exceeds min(windowsize / mss, maxqueuelen). Raising the max queue length to a value high enough queue the configured max window size restores the original behavior, where the max queue length is determined by the receive window size of the connection.

The TCP reassembly code is in need of a rewrite. The current implementation is inefficient, which is part of the reason such a low limit was placed on the queue length. It addresses concerns about CPU usage while doing TCP reassembly. The ability to raise or lower this limit is provided as a short-term workaround to control performance.

With the default value of net.inet.tcp.recvbuf_max=2097152 and 1500 MTU (1460 MSS), setting net.inet.tcp.reass.maxqueuelen=1437 or higher will restore the previous behavior.

#13 - 09/06/2018 09:20 PM - Ryan Moeller

- Status changed from Screened to Blocked

- Reason for Blocked set to Other: make a note in comments

12-CURRENT has recently received a significant rewrite of the TCP reassembly code [1], addressing the previously mentioned need. There is also work in progress to fix some outstanding issues with the old reassembly queue code in the stable branches [2].

I am reaching out to the author of the patch that introduced the queue length limit (he is also the author of the stable fixes) to discuss the issue we have with 11-stable.

[1]: https://reviews.freebsd.org/D16626
[2]: https://reviews.freebsd.org/D16471

#14 - 10/04/2018 01:38 PM - Ryan Moeller

- Related to Bug #49576: Apple OSX backup very slow, eventually seems to stop and not to proceed added

#15 - 11/20/2018 03:41 PM - Ryan Moeller

- Status changed from Blocked to In Progress

I have received no response from the author of this code nor the thread I started on the net mailing list; the following is my personal analysis.

The TCP reassembly queue length limit effectively imposes a tradeoff between

- additional CPU time spent processing a long queue
- discarding potentially good packets and causing retransmissions, which means throughput and latency suffer

For our application, the possibility of higher CPU load is more tolerable than a performance hit by default.

The plan is to raise net.inet.tcp.reass.maxqueuelen high enough that the previous behavior is restored for any reasonable configuration of the system. This should resolve the related performance issues that have been encountered.

The max queue length value I am proposing is 11587. This is based on 1448 byte max segment size (MSS) and a 16 MiB receive window. By default, the max receive window is 2 MiB, but the Autotune script will raise the max to 16 MiB if enabled. 1448 is the MSS when the MTU is 1500 and net.inet.tcp.rfc1323=1 (the default). 16 MiB / 1448 B = 11586 and some change, so I'm rounding up to 11587.

This should be a sufficiently high limit to make additional manual tuning unnecessary. Even outside of these tuning values, it is unlikely so many out-of-order segments would ever be in the queue at once, unless the network is under attack by some adversary or faulty hardware or misconfigured/buggy software.

#16 - 11/21/2018 09:30 AM - Alexander Motin

I don't feel very good, knowing we may have to sequentially iterate 12000 packets for each received one. I propose we increase it from present 100 to 1000 and see whether anybody complain about that. It should cover TCP window sizes of over a megabyte, that should be enough for most of links. Even if we ever announce 16MB receive window, it does not mean sane sending side will grow that big sending window.

#17 - 11/21/2018 11:05 AM - John Kutta

Sorry to butt in, but "see whether anybody complain" doesn't seem very robust... in that case, wouldn't it be more appropriate to test this in a lab environment, with a couple if different types of different clients and different load scenarios? I had assumed you did that sort of soak and regression testing ahead of any release to the public in any case.

Also, the behaviour when the server hits the cap and starts to throw away packets is potentially catastrophic, depending on the protocol of course, but SMB for one does not like this at all and it leads to scenarios where some clients have to be rebooted before they recover. Surely in this scenario an increased CPU load is preferred, over saving those CPU cycles (likely to the effect that the CPU only idles instead) at the expense of killing clients?

Or in other words; the fact that the implementation is inefficient, and has the CPU potentially iterating sequential lists, is unfortunate... but surely the solution isn't to start to throw away packets, as long as the CPU still has cycles to apply. Clearly the implementation needs improving.. and that's also the real solution.

Makes sense..?

#18 - 11/21/2018 11:18 AM - Alexander Motin

The problem is that there is no obvious solution or obvious answer, only trade offs. Everything depends on specific environment and how pathological it is. So far you are the only who reported about this problem AFAIK, so there is a chance that if we reduce the scale of the problem by 10 times, we may not hear about it ever again, and after migration to FreeBSD 12 next year problem disappear completely. We do run our own tests with different clients, but our performance engineers haven't reported about this problem.

#19 - 11/21/2018 11:25 AM - Alexander Motin

John, it could be useful to hear whether limit of 1000 is sufficient for your case.

#20 - 11/21/2018 11:39 AM - John Kutta

Hi, no there's been more people with the problem on the forums, but the discussion usually links back to the original thread which in turn concludes that I will raise a ticket (this one). I take your point about trade-offs though, i.e. how far are you prepared to load the CPU vs starting to throw away packets with the potential consequence of killing clients. I would have thought the latter is one to avoid in almost any scenario?

Already noted above that a default value >=1437 will restore previous behaviour (i.e. any version prior to 11.1-U5), surely you do not want to go below this at least, particularly if as you mention FreeBSD 12 will bring a more efficient implementation which will bring the CPU load down.

I note that the upstream change which introduces this is concerned about DoS attacks; "An attacker who has the ability to send TCP traffic to a victim system can degrade the victim system's network performance and/or consume excessive CPU by exploiting the inefficiency of TCP reassembly handling, with relatively small bandwidth cost. [...] As a workaround, system administrators should configure their systems to only accept TCP connections from trusted end-stations, if it is possible to do so. For systems which must accept TCP connections from untrusted end-stations, the workaround is to limit the size of each reassembly queue."

If FreeNAS was an Internet appliance (e.g. web server) I would be more sympathetic to taking the cautious approach... however a "rock-solid" NAS appliance I would argue should weigh things differently.. not my call of course but my 5c worth.

As for myself, I've already set my sysctl and will keep it as such, as my server is not exposed to unknown clients on the Internet... but I do expect it to be rock-solid on my own LAN against any type of client I throw at it and even with net.inet.tcp.reass.maxqueuelen at a very high value (16k) I've not seen SMB transfers load my CPU (i3) to any meaningful degree... the current default of 100 however makes SMB with Mac clients impossible.. and my topology is not particularly exotic, a few switches in a straightforward VLAN configuration.

#21 - 11/21/2018 11:43 AM - John Kutta

Ok I will see if I can try with 1000 - it's quite cumbersome for me though to take everything down and restart etc.

Best would be of course if your test engineers could reproduce the problem and take it from there.... ;-)

#22 - 11/21/2018 11:58 AM - Ryan Moeller

There is also at least one other report linked to this ticket, and I have seen a few reports in the wild. That said, I agree with the suggestion to start at 1000 and see the impact. I think it should be sufficient for non-pathological cases, and this was my original plan exactly.

The trouble is, I haven't figured out the specific conditions necessary to reproduce the issue in the lab. Without understanding the cause of the out-of-order segments, I'm not able to predict how bad we can expect it to get in a non-broken network, and I'm not able to do extensive testing to validate.

I was surprised that our performance tests didn't find any problems when they tested a macOS SMB client. I have been able to measure discarded packets coinciding with transfer stalls on my own network. It is likely dependent on particular hardware or network setup.

#23 - 11/21/2018 02:46 PM - John Kutta

Understood. But how are you going to see the impact of 1000, as you say, if you can’t reproduce nor model this behaviour? Or can you? Not quite sure what the last paragraph on the previous comment means - implies you **have** seen the impact of the default setting in tests? Conversely, have you seen CPU spikes attributable to queue length in previous versions (11.1-U5 and prior) that would have you tune it down to avoid those, and indeed default to a more restrictive setting that is more prone to throwing away packets instead?

Sorry and I’ll stop now - but I can’t really see the rationale for tightening from previous defaults - all seems a bit anectodal on the back of a FreeBSD developer who doesn’t respond to questions relating to this patch he did - and as mentioned I see throwing away packets as a Very Bad Thing in a file server, to be avoided at almost any cost, including accepting that there are inefficient algorithms that cause excessive CPU use.

Also, I have seen no evidence or suggestion on what the queue length needs to be before CPU usage becomes a real problem (DoS). As mentioned I ramped up the setting massively in order to be sure to eliminate the problem - have no way of monitoring the queue size in practical use but I have never managed to load my i3 with more than 15%ish while saturating 1Gbit with SMB traffic.

#24 - 11/21/2018 05:25 PM - Ryan Moeller

To clarify, I've experienced discarded segments and the impact on performance at home, but the engineers in the lab doing performance testing to verify the issue did not find a performance problem, and I don't have a clear explanation for why other than perhaps it is dependent on additional factors such as network equipment, the particular hardware, drivers, etc. I only have one Apple system at home so I can't really do much to test those theories.

The uncertainty about how to actually measure the impact of this tuning consistently and accurately is part of why I leaned toward just bypassing the whole mechanism, but it would be better to find a more practical limit that stays out of the way of real-world usage while still keeping the system protected against faulty/bizarre network equipment, misconfigured clients, etc.

> default to a more restrictive setting that is more prone to throwing away packets instead?

The goal is to pick a default that does not result in packets being thrown away under normal circumstances. It is not typical for an entire window to be so severely rearranged that the first expected segment arrives after most of the other segments. Traffic over a LAN usually arrives relatively in-order.

> I have also seen no evidence or suggestion of that the queue length should be before CPU usage becomes a real problem (DoS).

It depends on several factors. This is very much at the core of why it is hard to pick a value. If your CPU is fast, you're using jumbo frames, your load is coming from a small number of clients, you aren't actively monitoring transfers, then you might not notice or care and will have no problem with the limiter removed. If you have a heavily loaded server with a large number of clients concurrently throwing tons of out-of-order segments at you with a large window size and small MTU, it might add up to be a real problem.

I am of course doing what I can to test that we pick a reasonable default, but whatever choice is made, we will still want to keep an eye out for feedback from the community to gauge the impact.

#25 - 11/21/2018 09:30 PM - Ryan Moeller

I have found some interesting results.

I cleared my TCP stats, fired off a loop to poll the discarded count and the out-of-order count from netstat, and started an rsync of my FreeBSD ISO collection over SMB. I started at maxqueuelen=100 to get a baseline, and things seemed fine for a while, until I started playing music on YouTube. Almost immediately the counters started to periodically climb. Most of the time everything is fine, but every few seconds the counters creep up and the transfer rate drops.

When I bumped the reassembly queue length to 1000, the counters stopped rising. Now I've been running transfers over wifi at maxqueuelen=1000 for a few hours and it seemed rock solid there, until I started walking around with my laptop and streaming music again. Now the counters have started climbing periodically again.

My recvbuf_max is at the default 2097152.

It appears the reordering is significantly worse with other network activity competing for the network (I don't have a wired port to test if it is dependent on wifi specifically).

#26 - 12/05/2018 02:30 PM - Ryan Moeller

The compromise I have opted for is to set the default as net.inet.tcp.reass.maxqueuelen=1448 = ⌊2 MiB / 1448 B⌋ to restore the unrestricted behavior for default configurations, while retaining DoS mitigation for configurations tuned for larger receive buffers (for example by the autotune script).

Master PRs:
https://github.com/freenas/freenas/pull/2185
https://github.com/iXsystems/truenas/pull/248

11.2 PRs:
https://github.com/freenas/freenas/pull/2206
https://github.com/iXsystems/truenas/pull/254

#27 - 12/05/2018 05:55 PM - Bug Clerk

- Status changed from In Progress to Ready for Testing

#28 - 12/05/2018 05:55 PM - Bug Clerk

- Target version changed from Backlog to 11.3

#29 - 12/06/2018 08:41 AM - Alexander Motin

Ryan, lets backport it to 11.2-U1 and may be 11.1 too.

#30 - 12/06/2018 10:07 AM - Bug Clerk

- Status changed from Ready for Testing to In Progress

#31 - 12/09/2018 11:10 AM - Dru Lavigne

- File deleted (debug.tar.gz)

#32 - 12/09/2018 11:11 AM - Dru Lavigne

- Subject changed from Network issues after upgrading to 11.1-U6 to elax the TCP reassembly queue length limit to improve performance

- Reason for Blocked deleted (Other: make a note in comments)

#33 - 12/09/2018 11:11 AM - Dru Lavigne

- Target version changed from 11.3 to 11.2-U2

- Private changed from Yes to No

#34 - 12/10/2018 09:44 AM - Bug Clerk

- Status changed from In Progress to Ready for Testing

PR: https://github.com/freenas/freenas/pull/2206

#35 - 12/10/2018 10:03 AM - Dru Lavigne

- Subject changed from elax the TCP reassembly queue length limit to improve performance to Relax the TCP reassembly queue length limit to improve performance

- Needs Doc changed from Yes to No

- Needs Merging changed from Yes to No

#37 - 12/10/2018 10:19 AM - Ryan Moeller

- Copied to Bug #63180: Relax the TCP reassembly queue length limit to improve performance added

#38 - 01/04/2019 08:54 AM - Ryan Moeller

- Copied to Bug #67672: Relax the TCP reassembly queue length limit to improve performance added

#39 - 01/08/2019 04:22 AM - Dru Lavigne

- Status changed from Ready for Testing to In Progress

- Needs Merging changed from No to Yes

#41 - 01/08/2019 04:24 AM - Dru Lavigne

- Copied to deleted (Bug #67672: Relax the TCP reassembly queue length limit to improve performance)

#42 - 01/08/2019 04:24 AM - Dru Lavigne

- Has duplicate Bug #67672: Relax the TCP reassembly queue length limit to improve performance added

#43 - 01/08/2019 02:56 PM - Ryan Moeller

- Status changed from In Progress to Ready for Testing

- Needs Merging changed from Yes to No

#45 - 01/09/2019 03:46 PM - Dru Lavigne

- Status changed from Ready for Testing to In Progress

#47 - 01/09/2019 04:29 PM - Ryan Moeller

- Status changed from In Progress to Ready for Testing

#51 - 01/21/2019 09:03 PM - Dakota Schneider

John Kutta wrote:

> Also, the behaviour when the server hits the cap and starts to throw away packets is potentially catastrophic, depending on the protocol of course, but SMB for one does not like this at all and it leads to scenarios where some clients have to be rebooted before they recover. Surely in this scenario an increased CPU load is preferred, over saving those CPU cycles (likely to the effect that the CPU only idles instead) at the expense of killing clients?
>
> Or in other words; the fact that the implementation is inefficient, and has the CPU potentially iterating sequential lists, is unfortunate... but surely the solution isn't to start to throw away packets, as long as the CPU still has cycles to apply. Clearly the implementation needs improving.. and that's also the real solution.
>
> Makes sense..?

I would like to confirm that I am such a user whose experience is catastrophically affected by this issue. See forum thread for reference: https://forums.freenas.org/index.php?threads/11-2-rc2-network-issue-over-wifi-tcp-traffic-stops-after-hundreds-of-dupacks-from-freenas.71553/ I even reported a bug for it before discovering this one and verifying that it's actually the same underlying issue: https://redmine.ixsystems.com/issues/71459

In summary: with the queue at default 100 items and delayed ack turned on, I am easily able to saturate the TCP reassembly queue when tending data to the server (over SMB, iperf, or apparently anything over TCP). This causes a massive flood of DUP ACK packets to be sent to my client, which can actually overwhelm the network throughput on my client and interrupt other data streams.

If I increase the queue size to 16384, or disable delayed ack the issue is mitigated. I have also just tested the proposed queuelen 1448, with default recvbuf_max=2097152. Unfortunately I must report that this is still a performance bottleneck: I get a less catastrophic number of dup ack packets, but it still has an impact on performance, bringing SMB write throughput down from ~109MB/sec to 100MB/sec. With size 16384 I still get dup ack, but performance is better. For reference this is over Ethernet (Thunderbolt Ethernet adapter). The dup ack packets are still making a mess in this scenario.

#52 - 01/21/2019 10:43 PM - Ryan Moeller

- Has duplicate Bug #71459: TCP Traffic Stops after Hundreds of DupACKs from FreeNAS added

#53 - 02/07/2019 01:06 PM - Dru Lavigne

- Status changed from Ready for Testing to Done

- Needs QA changed from Yes to No

#55 - 03/01/2019 09:32 AM - Ryan Moeller

- Related to Bug #78222: Remove obsolete net.inet.tcp.reass.maxqueuelen tunable added