State of the union in TCP land

Eric Dumazet @ Google

Netdev 0x18, Jul 2024

Agenda

- Overall health status
- TCP-AO
- Better cache locality
- usec resolution in TCP Timestamps
- Better backlog processing under pressure
- TCP Swift
- AVX experimentation
- TCP OOOED
- tcp_auto_rbuf_rtt_thresh_us

linux TCP is still very active

- MPTCP got ~200 patches in last year.
- TCP got ~370 patches in last year.

Huge thanks to awesome contributors, reviewers and maintainers :

Kuniyuki Iwashima, Dmitry Safonov, Jakub Kicinski, David S. Miller, Paolo Abeni, Geliang Tang, Jason Xing, Neal Cardwell, Matthieu Baerts, and others.

Numbers, numbers.

Thanks to TSO and HW-GRO, we reach line-rate on 200Gbit NIC even with a single TCP flow, with a reasonable MSS (8K), even on an ARM core.

-> ~380,000 packets per second (without BIG-TCP)

This includes the additional overhead of header splitting.

TCP-AO in linux-6.7

TCP-AO (RFC 5925) has finally been merged in linux-6.7

Joint work from Dmitry Safonov, Francesco Ruggeri, Salam Noureddine @Arista.com

This is supposedly used by BGP daemons.

Better memory locality

In linux-6.8, a series of patches reorganised socket fields (and other related networking structures) to reduce the number of cache lines used in TCP rx and tx paths.

This is especially useful for hosts dealing with a lot of concurrent sockets and RPC-like traffic.

https://lwn.net/Articles/951321/

(Thanks to Coco Li, Wei Wang, Daniel Borkmann, ...)

usec resolution in TCP timestamps

In linux-6.7, we made possible to use usec instead of ms in the TCP TS values.

1) better observability of delays in networking stacks.

2) better disambiguation of events based on TS val/ecr values.

3) building block for congestion control modules needing usec resolution.

ip route add 10/8 ... features tcp_usec_ts

Initial idea from Van Jacobson

defer / suppress regular ACK while processing socket backlog

In linux-6.7, TCP is able to better cope with pressure on the socket backlog.

This was driven by optimizations for TCP-direct ML workloads on ARM cores.

Performance of a single TCP flow on a 200Gbit NIC (with copies)

- Throughput is increased by 20% (100Gbit -> 120Gbit).
- Number of generated ACK per second shrinks from 240,000 to 40,000.
- Number of backlog drops per second shrinks from 230 to 0.

More ACK suppression in the future ?

We already have SACK suppression which show great benefit whenever some packets are dropped, especially for wifi networks.

Idea is to generalize the idea to normal ACK.

IETF TCPM group is working on a standard : TCP ACK Rate Request Option

https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-ack-rate-request-05

Note that linux could adopt a different strategy (no additional options)

Release socket lock earlier

In BH handler, calling expensive wakeups (EPOLLIN , EPOLLOUT) could be done after the socket lock has been released.

Similarly in recvmsg() we probably could send potential ACK packets after the socket lock has been released, because traversing the whole stack up to ndo_start_xmit() gives more opportunity for BH handler to feed the backlog queue, thus increasing recvmsg() latency at release_sock() time.

issues fixed recently

TCP_USER_TIMEOUT bug (CVE-2024-41007), landing in 6.10 and stable kernels.

DSACK undo in bug fast recovery.

Fastopen bug vs ‘partial’ SYNACK

Busy-polling - more to come

https://netdevconf.info//2.1/slides/apr6/dumazet-BUSY-POLLING-Netdev-2.1.pdf
Slow progress on this front, but people want to use busy-polling.

“Break the pipe”

Wei Wang implemented part of the ideas exposed in 2017.
In order to better use the cpus dedicated to napi-poll, we return early from tcp_sendmsg() after skbs have been put in the socket write queue.

TCP-Swift

Kevin Yang (yyd@google.com) plans to upstream TCP-Swift congestion control in Q3 2024.

Swift is a delay-based congestion control algorithm that uses AIMD based on network_rtt measurements.

It is designed for datacenter networks where a round-trip propagation delay is usually around 100us or less.

https://dl.acm.org/doi/10.1145/3387514.3406591

Status on MOVNTDQA attempts

Presented at netdev 0x16.

Idea was to not populate cpu caches at recvmsg() time for the source buffer,

in a similar (but reverse) way than (ethtool -K eth1 tx-nocache-copy on)

Unfortunately, Jeffrey Ji was not able to demonstrate significant gains.

TCP Out Of Order Early Delivery (OOOED)

TCP sockets need a big RCVBUF value if they want to deal with packet losses without stalling too much.

When a packet is lost, a bubble is built, accumulating packets in kernel memory, thus increasing memory costs, until the hole is repaired.

When the hole is repaired, the application receives an EPOLLIN and might have to transfer megabytes of data in a big burst, which:

- increases memory costs in the kernel, and max latency.
- defeats page pool recycling strategies used in some NIC drivers.
- defeats DDIO on various cpu.
- Wastes cpu cache for non zero copy modes.

I

OOOED : Proposal

Instead of delivering incoming data in the legacy way (monotonically increasing copied sequence number), allow for the application to get a copy (or pages/iovecs for zero copy modes), from packets in Out Of Order receive queue (and/or from normal receive queue).

Design a new API where the application explicitly signals to the kernel that it is okay for the kernel to deliver chunks, and copy them in a non sequential way (as soon as they are received by TCP stack)

For each chunk of data copied, kernel would have to give to the application a 2-tuple:
<SEQ-GAP, SIZE>

SEQ-GAP is basically the delta between packet SEQ and tp->copied_seq.

After an skb payload has been consumed, we can remove the payload from the skb, and keep skb in OOO queue for normal TCP SACK processing.

OOOED : API changes

1) New recvmsg() flag.
   Report the 2-tuple(s) to satisfy a recvmsg() in CMSG message.
   Cons: slows down fast path with extra checks.

2) New getsockopt()
   (Like we did for TCP_ZEROCOPY_RECEIVE)

   Allow for a mix of copy/zerocopy, and/or TCPDirect use case, where the 2-tuples need to be expanded to something like:
   <SEQ-GAP, DMABUFF cookie, size>

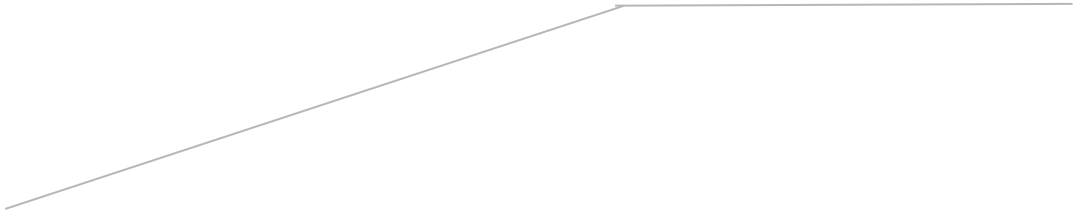
   We can decide if the copy can be done ‘in-place’ in the user provided buffers, leaving holes untouched (and completed later) or if the application prefers receiving chunks in arbitrary locations. (Application would have to deal with a traditional iovec model in order to consume the received data)

tcp_auto_rbuf_rtt_thresh_us

Currently, we cap sk->sk_rcvbuf at sysctl_tcp_rmem[2] which is a constant for all TCP connections.

This value usually makes sense with long RTT connections but is way too large for small RTT connections.

Idea: Make the actual value a function of RTT.



tcp_auto_rbuf_rtt_thresh_us

```
if (tp->srtt_us)
	rtt = min(tp->srtt_us, tp->rcv_rtt_est.rtt_us) >> 3;
else
	rtt = tp->rcv_rtt_est.rtt_us >> 3;

if (rtt >= net->ipv4.sysctl_tcp_auto_rbuf_rtt_thresh_us) {
	rcvbuf_cap = net->ipv4.sysctl_tcp_rmem[2];
} else {
	rcvbuf_cap = ((s64)net->ipv4.tcp_rcvbuf_cap_slope * rtt) >> 8;
	rcvbuf_cap += net->ipv4.sysctl_tcp_rmem[1];
}
return rcvbuf_cap;
```