enfabrica

Can the Linux networking stack be used with very high speed applications?

SEPTEMBER 2022

Introduction

enfabrica

Set out to answer a fundamental question:

How does the Linux networking stack scale as the line rate increases?

Stated another way: **When 400G and 800G become common, will the S/W be ready?**

- Get out in front on what is needed and start working on solutions
- Changes to Linux take time - way too much time (e.g., XDP H/W hints)

Scale "**up**" (pushing a single flow to line rate) is as important as scale "**out**" (multiple flows to reach line rate)

- e.g, Machine Learning apps

Created a custom setup to investigate what is needed to scale S/W to higher line rates

- Able to push a single flow to over ~~670~~ 782 Gbps and more than 31M pps

Test Setup

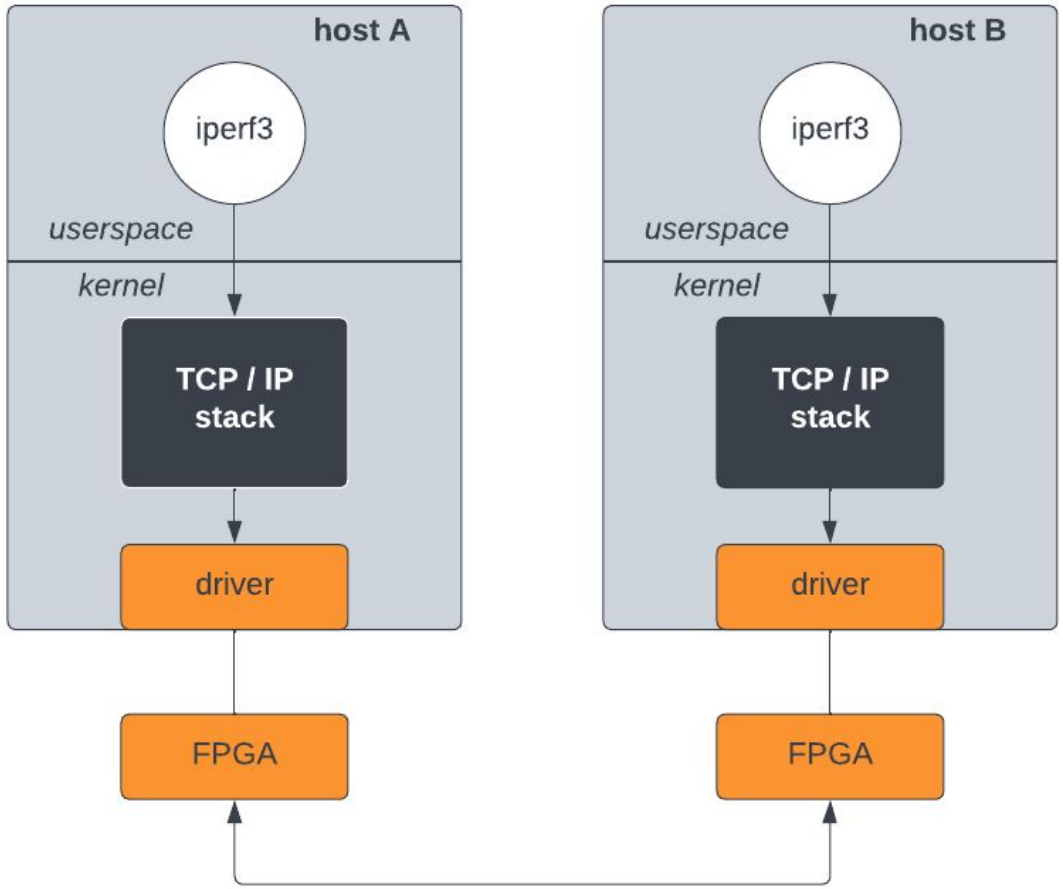
Off-the-shelf servers

Ryzen 9 Zen 3 cpu (5950x 16-core), 5GHz
128GB, DDR4, 3200 MT/s

Ubuntu 20.04 OS

Unmodified 5.13.19 kernel

Xilinx FPGA - VCU1525



Getting Around Current Physical Limits

PCI gen3 x16 (128 Gbps cap)

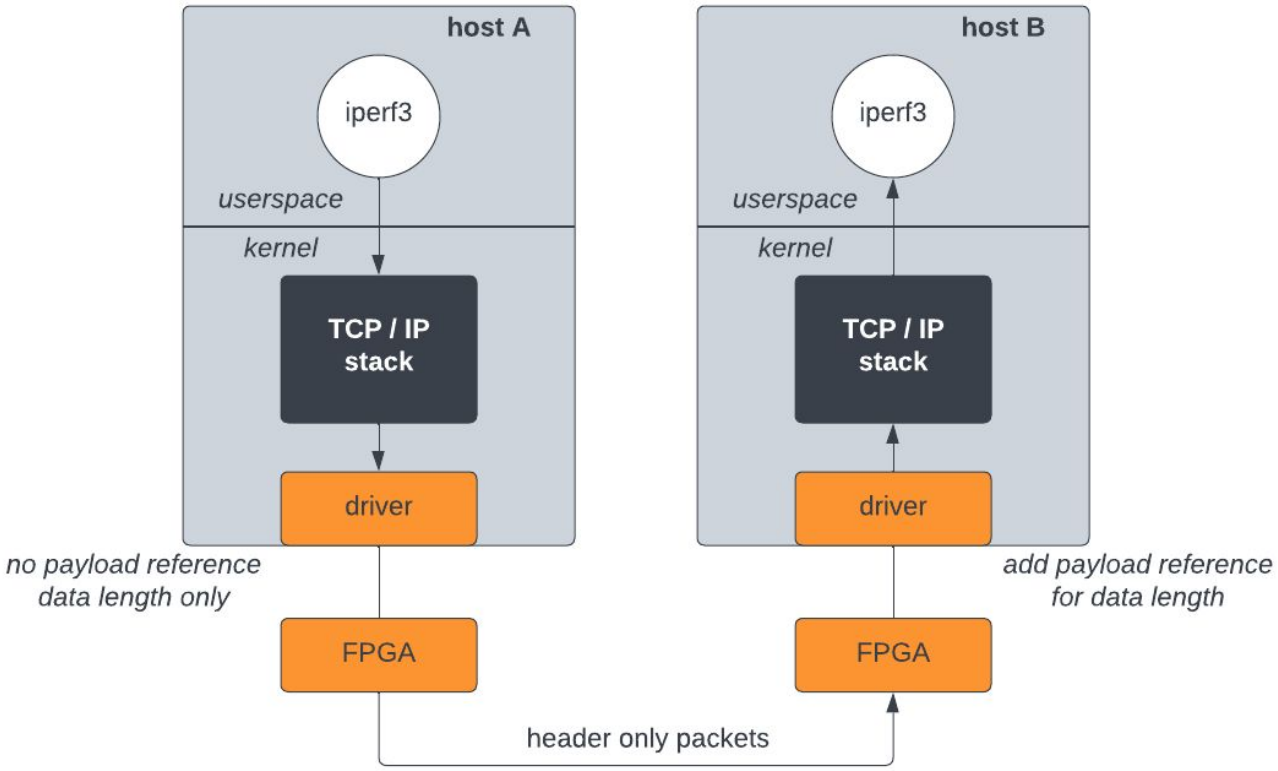
2 QSFP28 100Gbps interfaces

Idea: Payload in benchmark app is meaningless. Why send it?

- Drop payload over the wire

Key Point: **Software stack sees actual packet size and rate that TCP wants to send**

No modification to kernel. Payload games confined to driver and FPGA user logic



Zerocopy / Direct Data Placement

enfabrica

memcpy - never going to happen

- Speed limited to ~30Gbps with memcpy
- ZC / DDP type scheme is a requirement for high speed networking

Existing Linux Zerocopy APIs

- **Tx**: fairly easy to use, but has its overhead
  get_user_pages (and variants) plus reaping completions (recvmsg syscalls)

- **Rx**: very limited and tricky to use
  Requires a specific MTU size and header split such that payloads are exactly PAGE_SIZE
  Side band with memcpy for data less than PAGE_SIZE

Key Point: **Modern workloads need hardware to land data in application buffers**

Modified iperf3 to avoid memcpy

enfabrica

TX: Added support for ZC API (--zc_api option)

- Good enough for these tests to show intent - avoiding memcpy from userspace
- Extra CPU cycles for page pinning and completions is a factor, but not limiting one

Rx ZC API is too limiting and not usable for generic testing

Mimic **intent** of ZC API on Rx by dropping data

- --rx_drop option to iperf3 to use MSG_TRUNC with recvmsg
- Packets and data traverse networking stack as usual, attach to socket, process wakeup
- MSG_TRUNC drops payload to avoid memcpy to userspace

Net result is avoiding memcpy while still using existing socket APIs

Memory Management

enfabrica

400G at 4096 L2 MTU = 12+M pages per second

- Actual number depends on MTU and how S/W and H/W handle posted buffers

FPGA driver managed pages for packets similar to other high speed NIC drivers

- Page per packet to keep it simple across MTUs
- Max pps in tests 31+M pps means 31+M pages per second handled by driver
- Even split page for 1500 MTU means 16.5M pages per second

Avoid system page allocators

- Page pool infrastructure as the base layer
- Driver managed per-cpu cache on top (preferred allocation if available)

skb recycling via napi cache

- Use napi_build_skb over build_skb

Key Point: **Current buffer management scheme has too much overhead**

Reducing Packet Rate Handled by S/W

enfabrica

Amortize S/W stack overhead with more data per packet

- FIB lookup, socket lookup, tc and netfilter hooks, etc

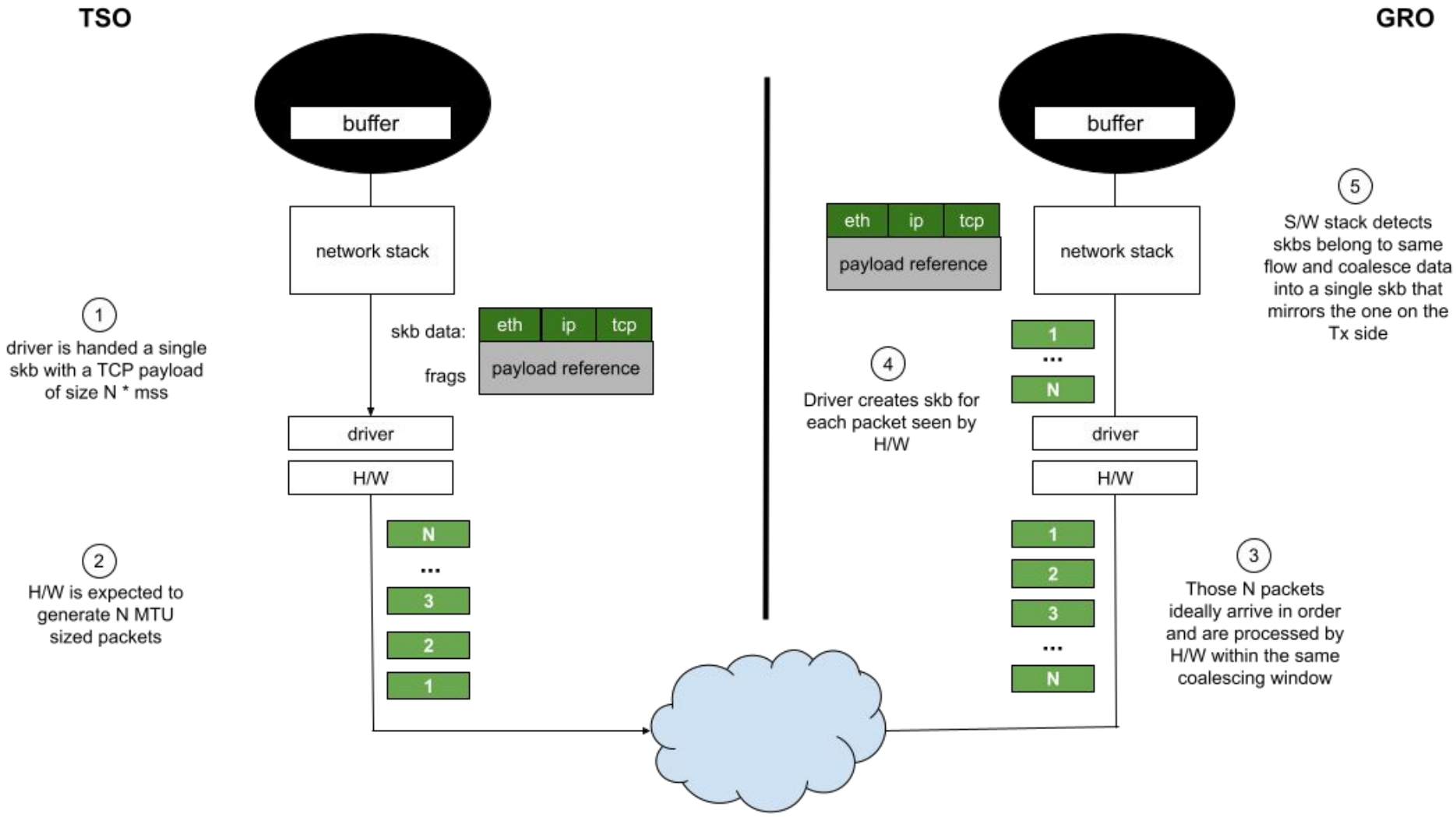
MTU

- More payload per packet on the wire

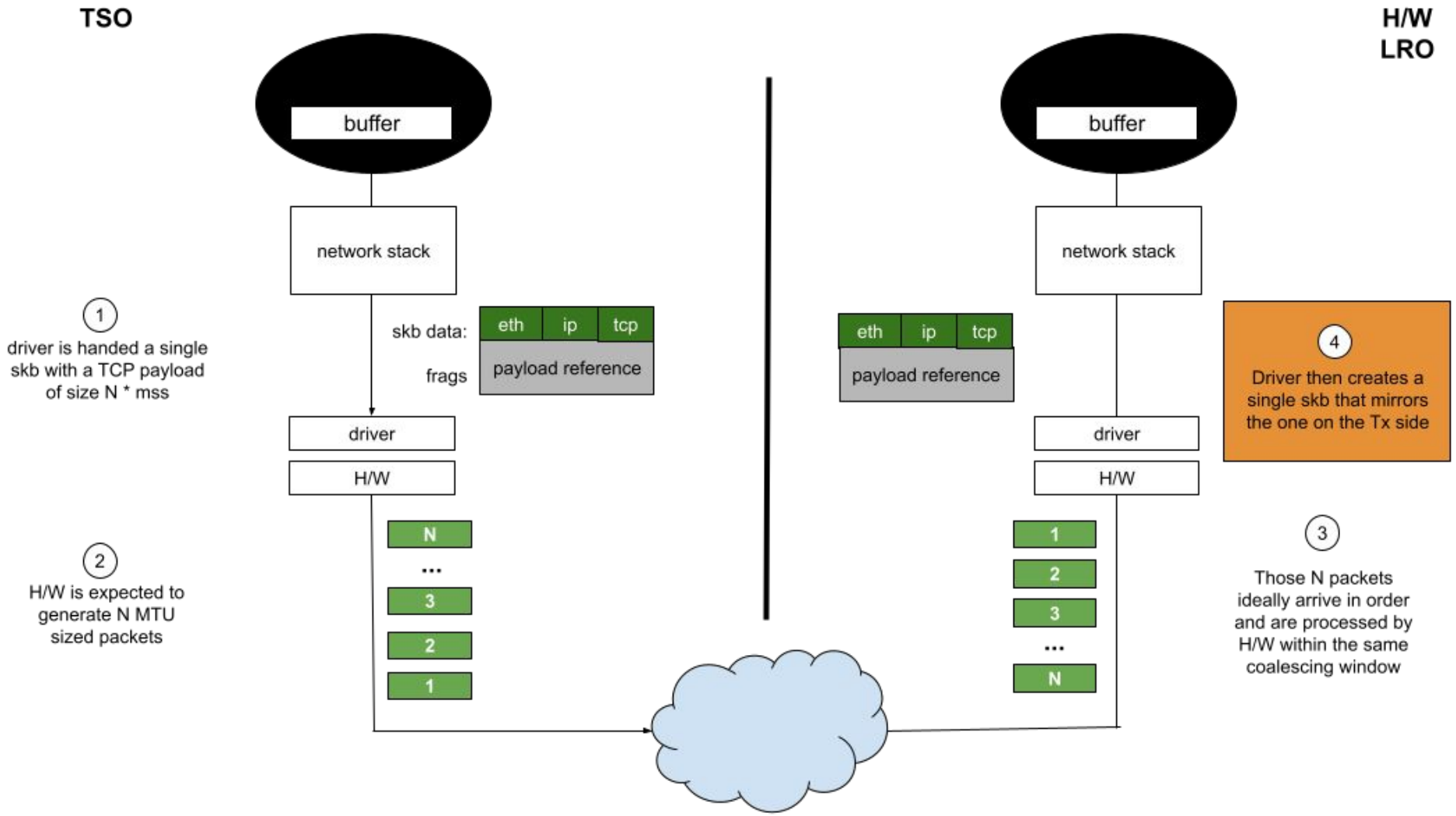
TSO into S/W GRO or H/W LRO

- Goal is pushing effective MTU seen by S/W up to 64kB

TSO -> S/W GRO

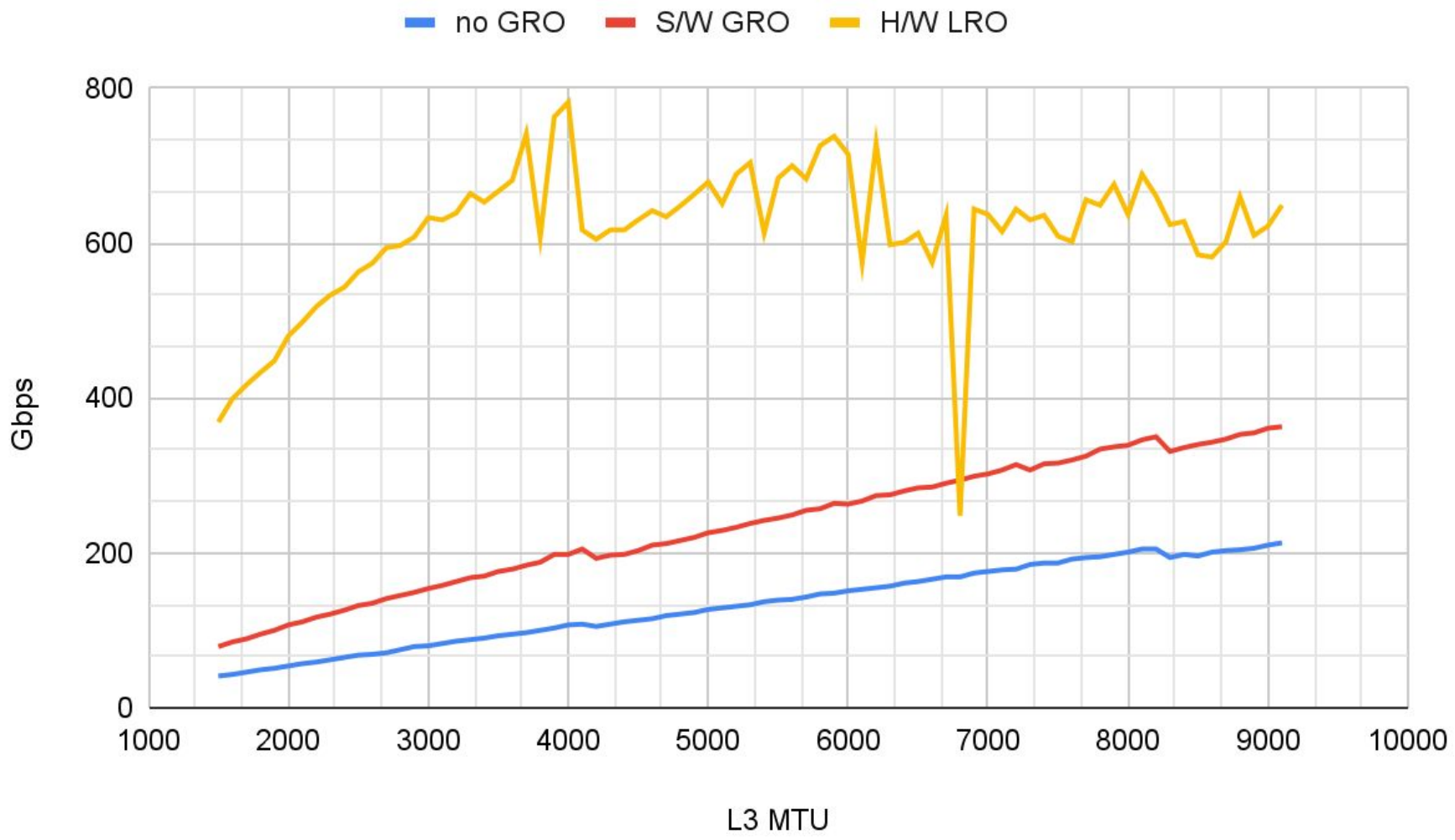


TSO -> H/W LRO

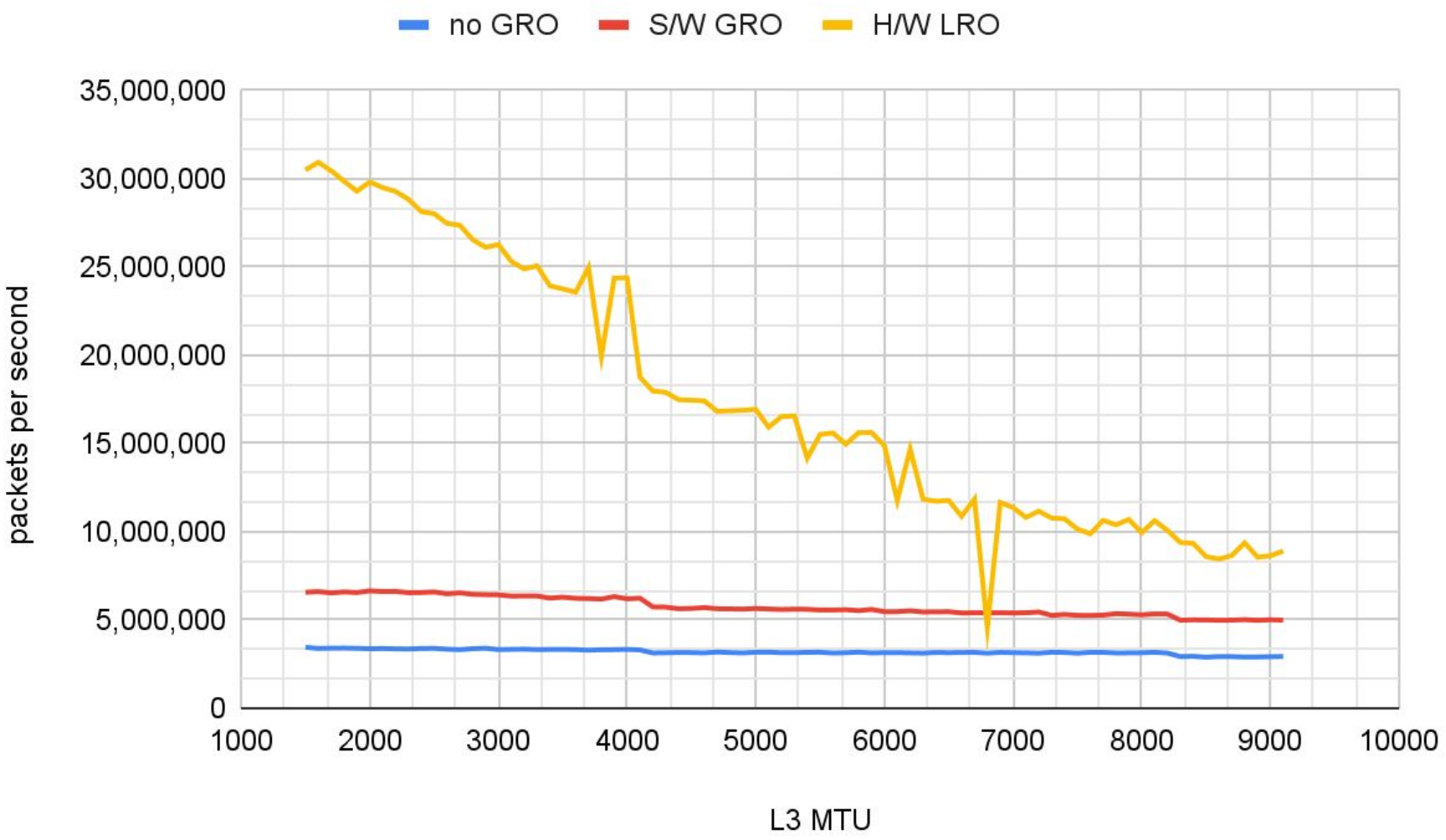


Data Rates on the Wire with TSO/*RO

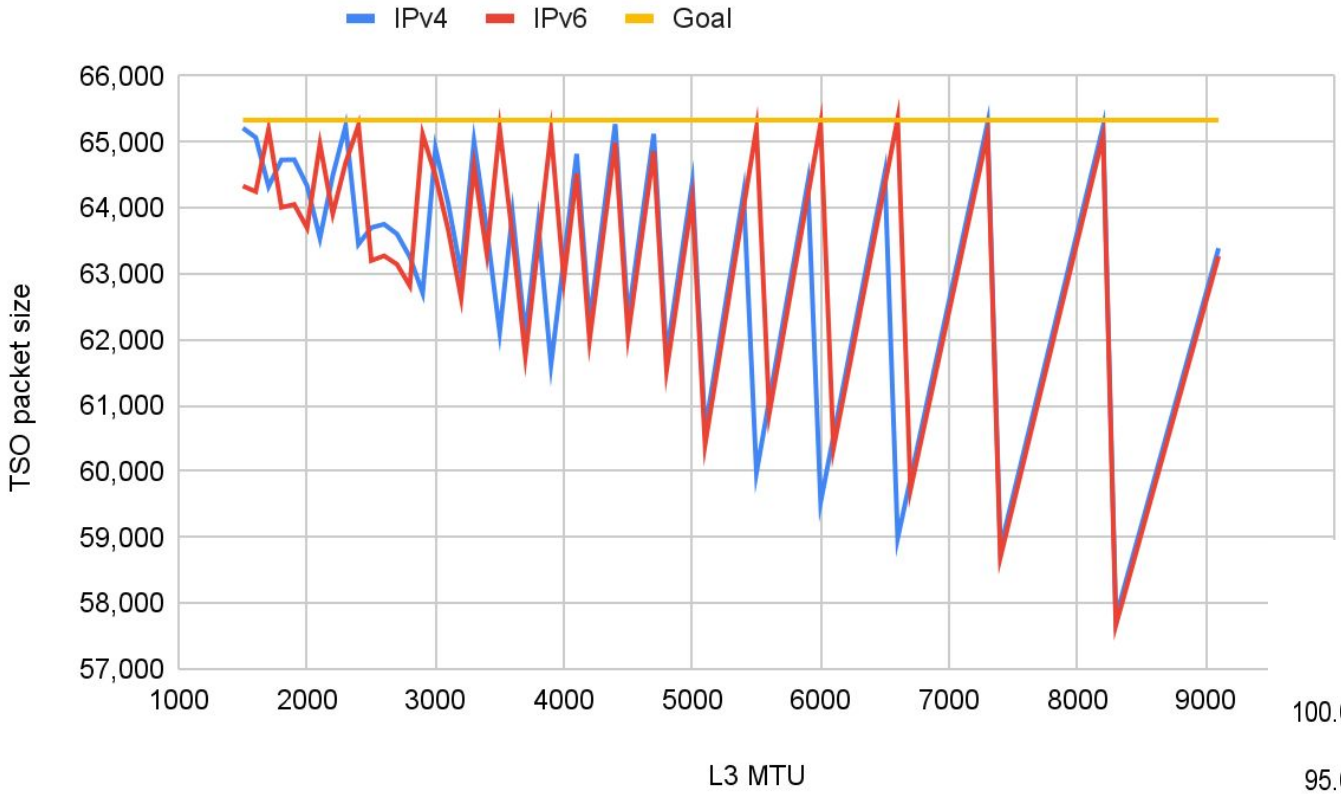
enfabrica

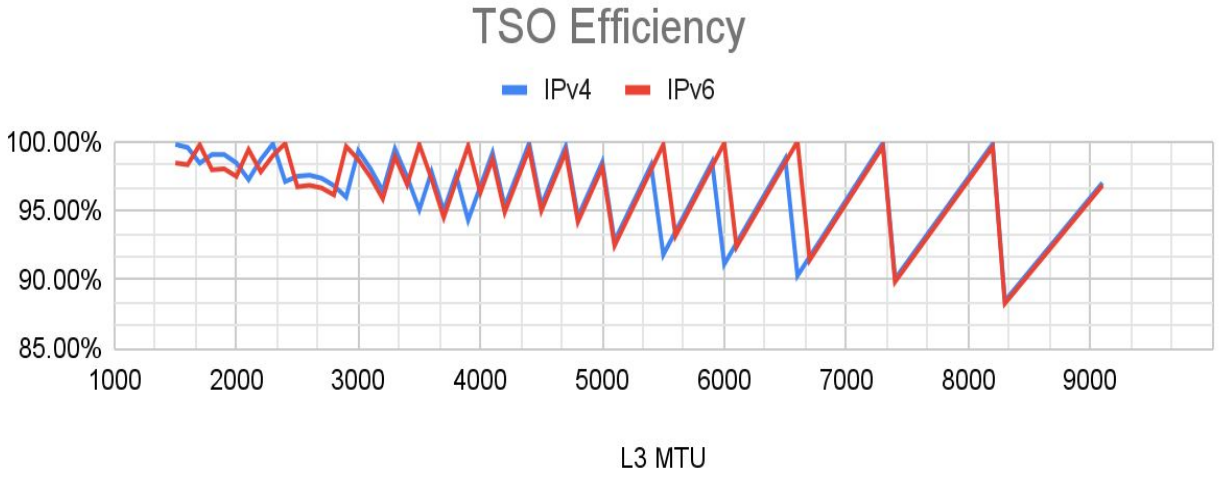


Packet Rate on the Wire with TSO/*RO



MTU / TSO Trade-off





Reducing Packet Rate Handled by S/W

enfabrica

non-GRO/LRO case shows maximum pps for S/W stack: ~3M pps

- increasing MTU alone is insufficient

MTU affects TSO goal which affects S/W packet rate

GRO helps but S/W analysis of packet headers wastes CPU cycles

LRO:

- @1500 MTU: 31.7M pps on the wire -> 720k pps into software stack
- @9100 MTU: 8.3M pps on the wire -> 1.2M pps into software stack

Key Point: **Need a solid, robust H/W based LRO scheme to scale up**

Socket Buffers and Syscalls

enfabrica

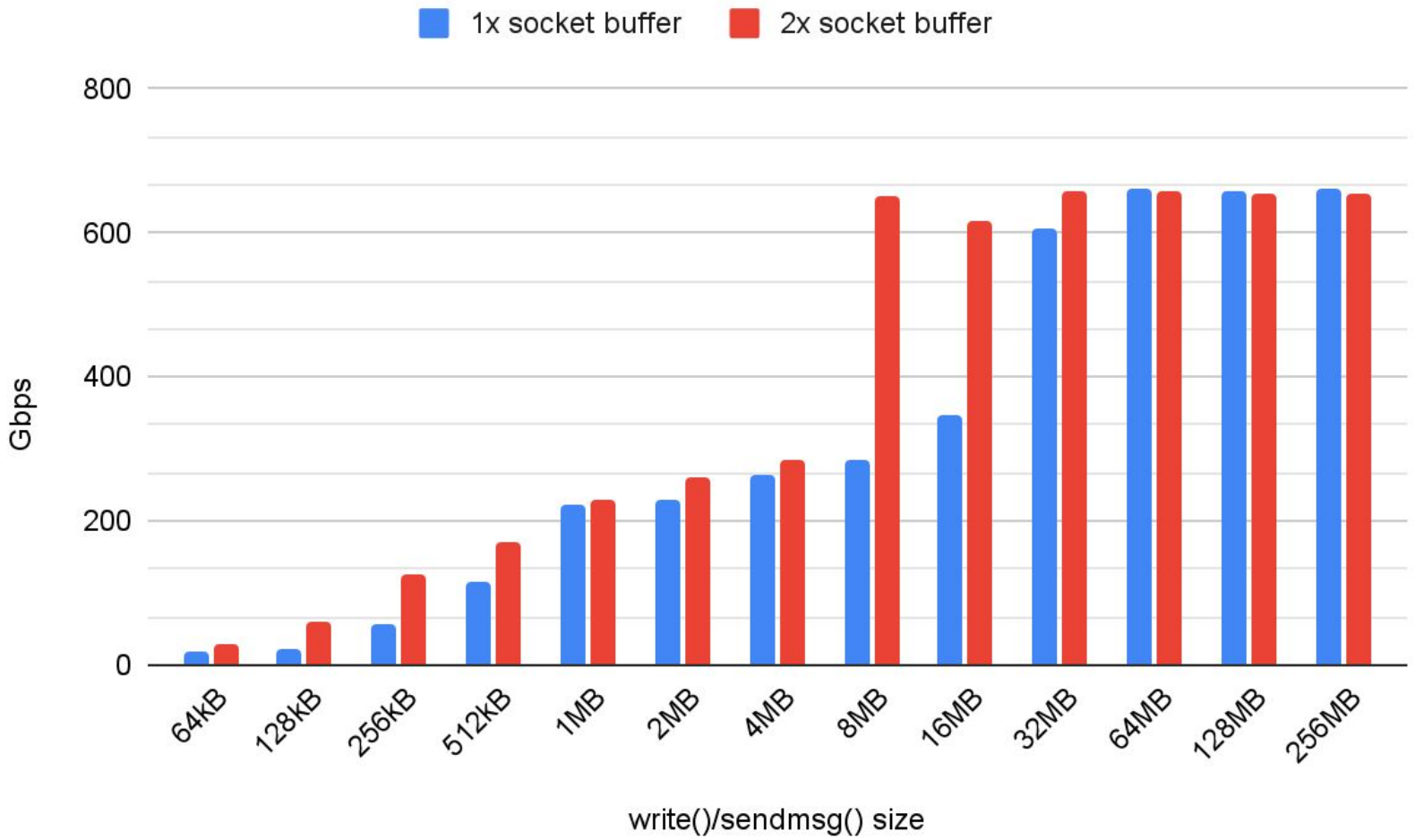
Keep the Tx pipeline primed

Reduce the overhead of using sockets to get data to/from H/W

More data per recvmsg /sendmsg syscall

More data queued up in socket buffers

-l and -w args to iperf3



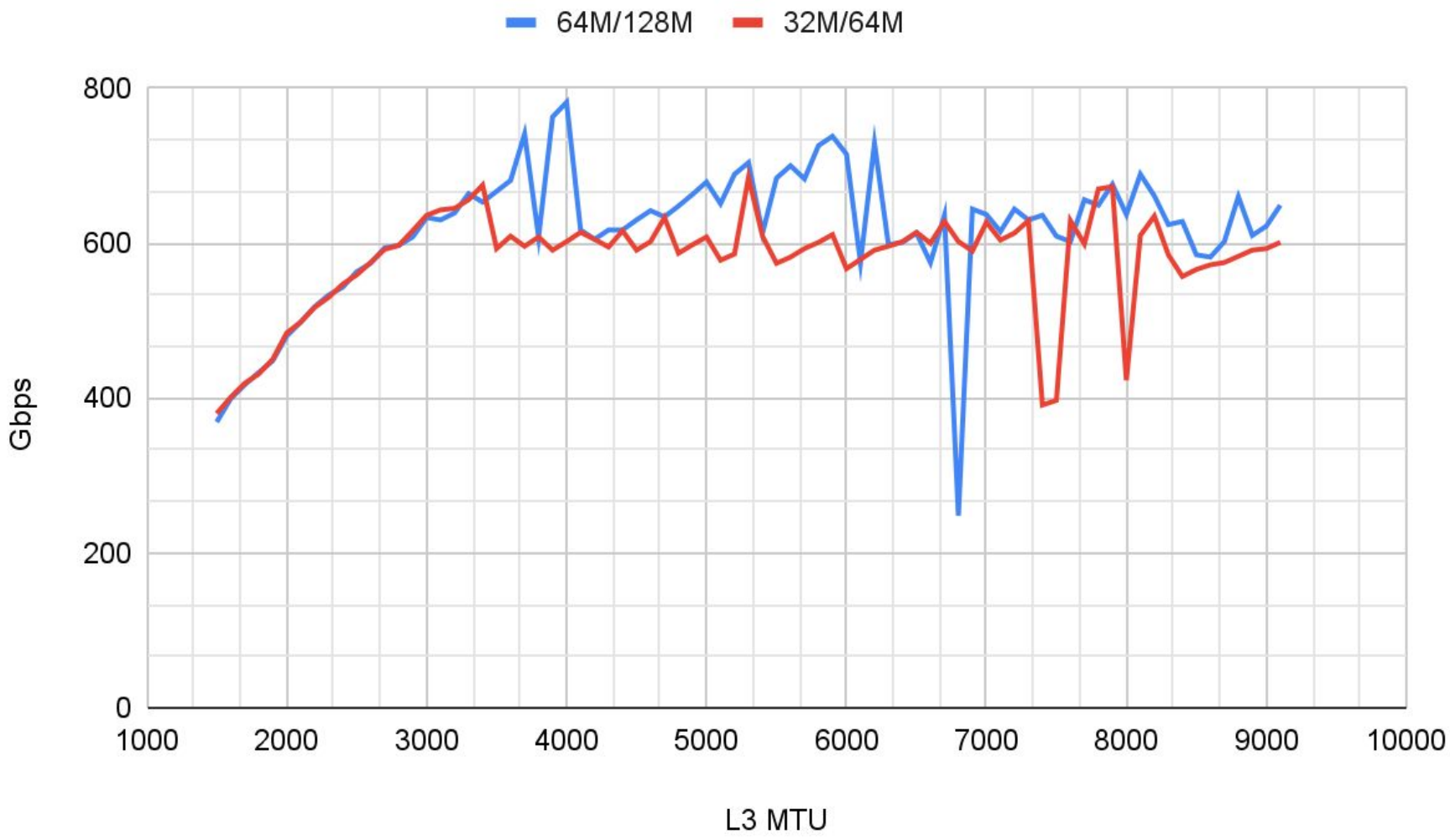
Socket Buffers and Syscalls

enfabrica

Comparison of 64M per read/write, 128M socket buffer vs 32M per read/write, 64M socket buffer

Key Point: **Need to manage datapath without system calls**

- io_uring does this via user-kernel queues, but it too is not sufficient



Hugepages

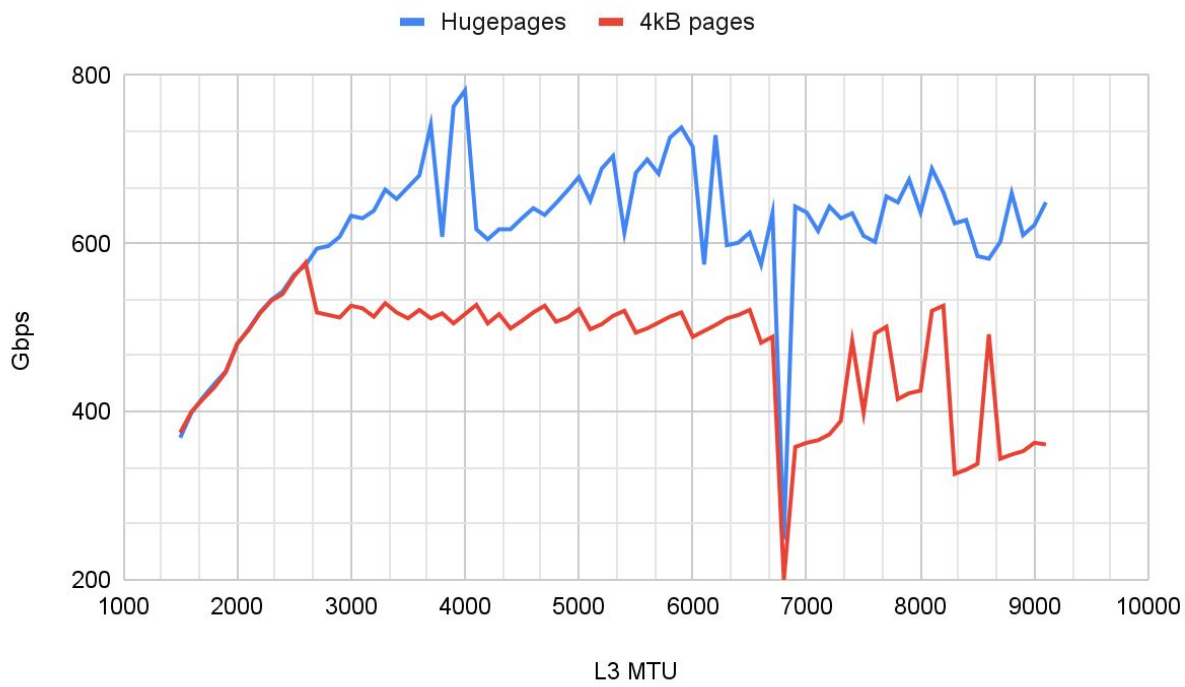
Tx ZC API iov is passed to driver and then H/W

- 4kB page size = 17 fragments per TSO skb
- 2MB hugepage = 2-3 fragments per skb
- non-ZC path for TCP has 2-3 fragments per skb

More fragments per skb == more overhead

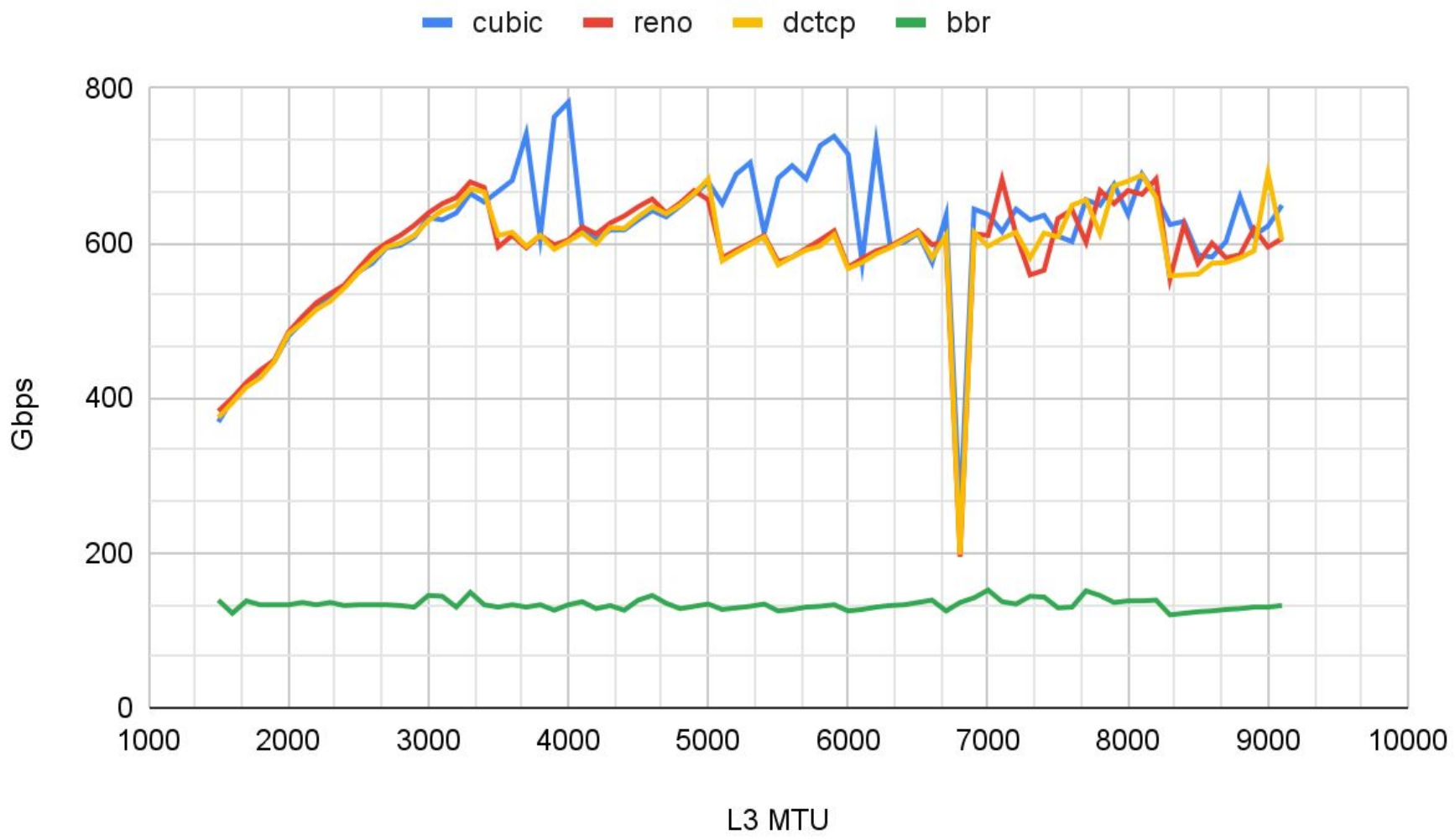
- internal_get_user_pages_fast and skb_release_data become prominent in perf profiles

Key Point: **Reduce the overhead of the buffer representation going through the networking stack.**



Congestion Control Algorithm

enfabrica



Hardware Considerations - CPU

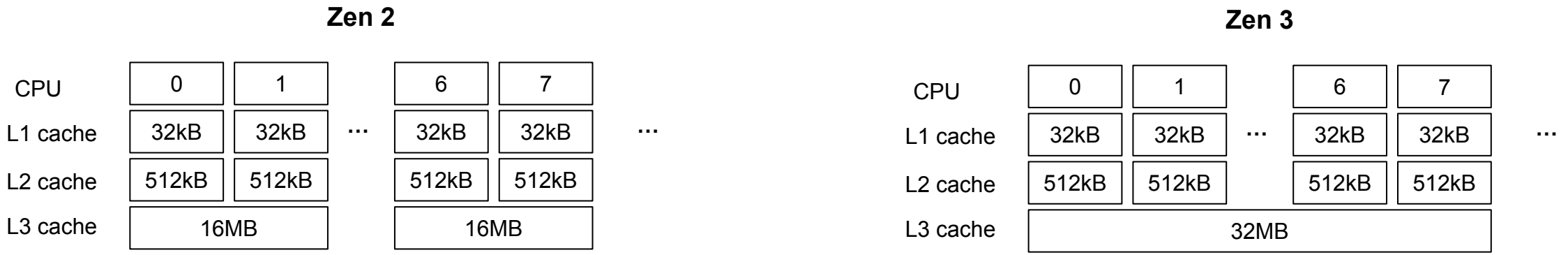
enfabrica

Speed is important

- CPU bound processing packets

L3 cache and task placement

- packet processing and application need to share L3 cache
- pinning means disabling irqbalance (static configuration)
- some CPU architectures make that more difficult



Hardware Considerations

enfabrica

Memory Speed

- Epyc based servers with memory speed from 2400 MT/s to 3200 MT/s
- Ryzen servers with 3200 MT/s
- Populate ALL slots

NIC - bumped ring size of 8192

- irq driven system - too many packets land before irq handler runs
- See drops today with 200G nics and 4096 ring size

Scale Out

enfabrica

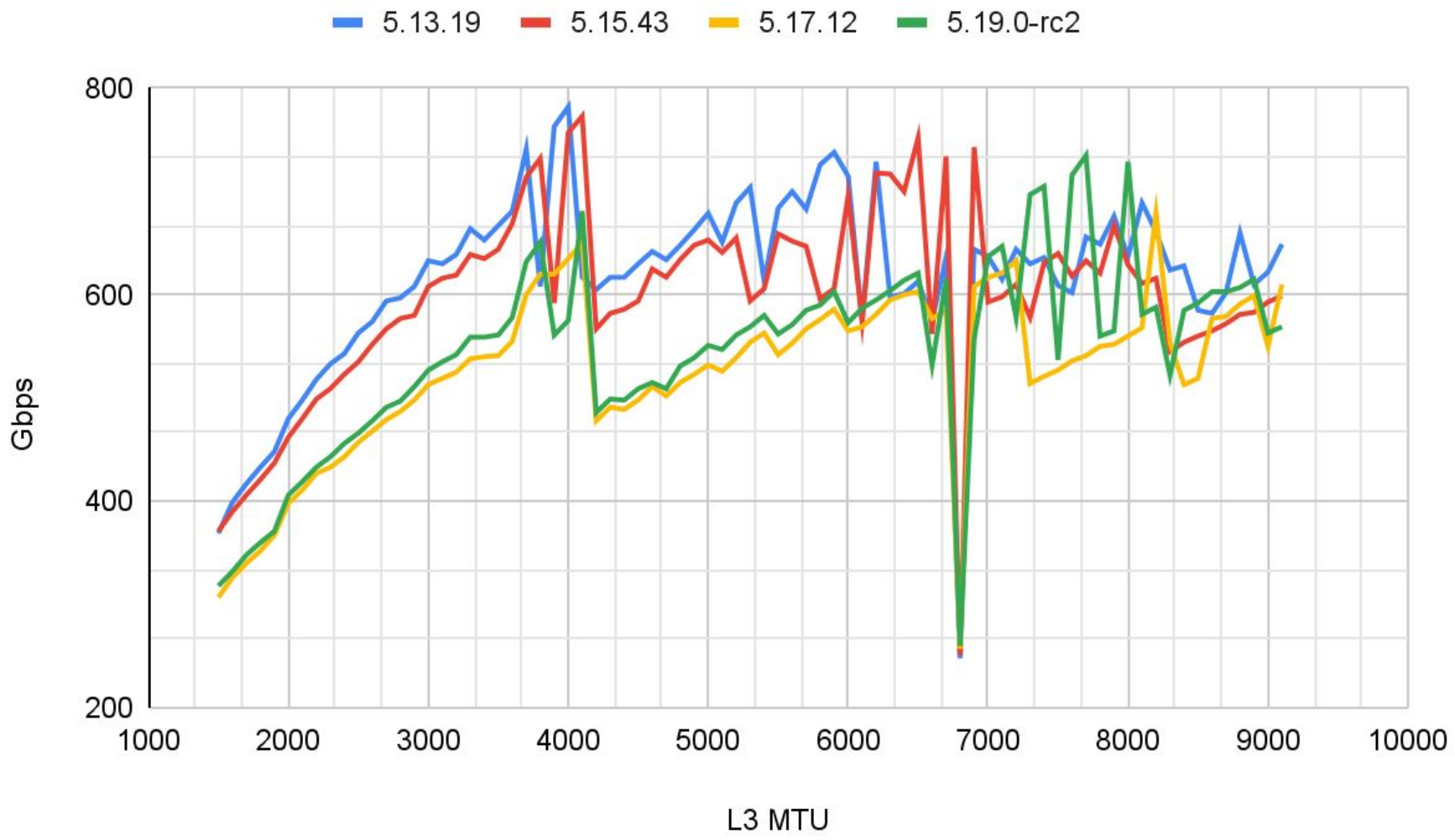
VCU 1525 has 2 MACs

Isolated resources - task placement, device queues

- 700+ Gbps for each of 2 streams (750G on one, 750G on the other) at 4000 MTU

Kernel Regressions

enfabrica



Misc. Tidbits

enfabrica

Ubuntu based systems: Disable CONFIG_INIT_ON_ALLOC_DEFAULT_ON

- page clearing on alloc is cpu intensive (i.e., performance killer)

Be weary of SMIs

Any packet socket running on the system kills performance

- e.g., tcpdump, lldpd
- all packets are cloned

Key Point: **Application wanting high performance can be impacted by random events**

Summary

enfabrica

Linux TCP stack can scale to high bit rates, but bottlenecks with socket API need to be removed

Must have a scheme where hardware places packet payload directly in application buffers

Better memory / buffer management scheme

Must have a solid LRO scheme from H/W - S/W GRO will not cut it

Reduce / Eliminate system calls

Simpler representation of application memory in skb as it traverses the networking stack

Resource isolation allows scale out and up

Followup

enfabrica

Be sure to catch the next talk at netdev 0x16

Thank You

Results with ConnectX-6

enfabrica