Drinking From The Host Packet Fire Hose

Netdev conf 0x18

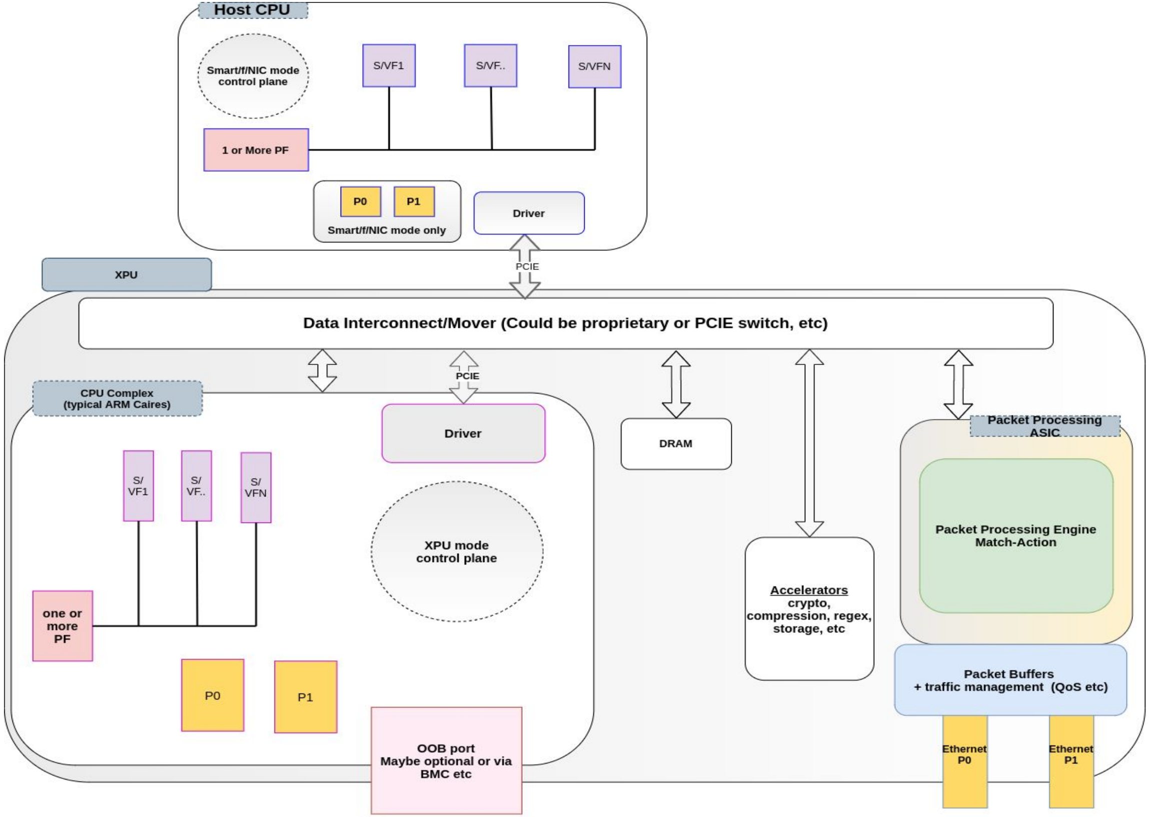
Jamal Hadi Salim, Nabil Bitar, Pedro Tammela
Work Done At Bloomberg

Agenda

- Lessons Learnt From xPUs
- Overcoming The Host Message Interface Costs
    - Discussion on PPS reduction Techniques
    - Discussion on Data Movement Reduction Techniques
- Experiments And Results
    - Setup And Traffic Patterns Used
    - Experiments
- Quarks And Challenges
- Takeaways
- Future Work

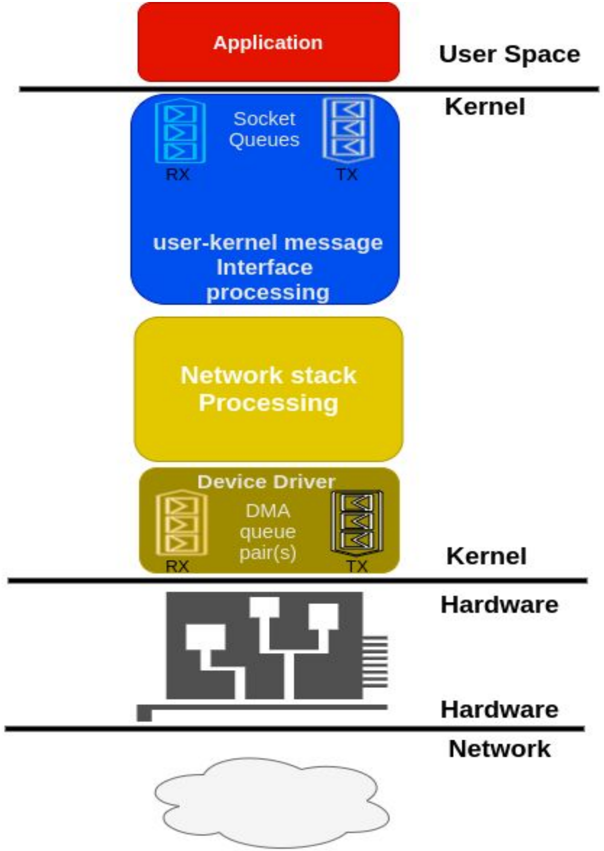
Lessons Learnt from xPUs

Infrastructure/Data Processing Unit (XPU) NIC Overview



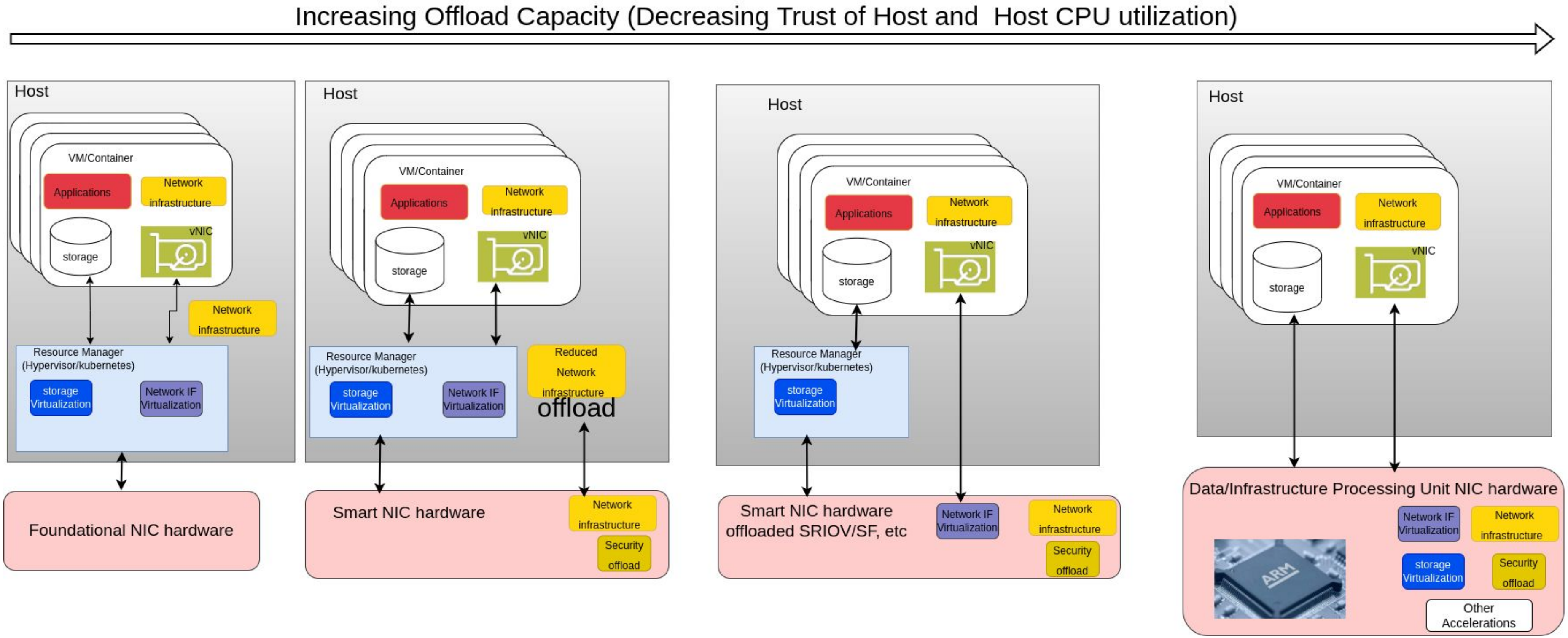
Host CPU
Smart/f/NIC mode control plane
S/VF1
S/VF..
S/VFN
1 or More PF
P0
P1
Smart/f/NIC mode only
Driver
PCIE
XPU
Data Interconnect/Mover (Could be proprietary or PCIE switch, etc)
PCIE
CPU Complex (typical ARM Caires)
Driver
S/ VF1
S/ VF..
S/ VFN
XPU mode control plane
one or more PF
P0
P1
OOB port Maybe optional or via BMC etc
DRAM
Accelerators crypto, compression, regex, storage, etc
Packet Processing ASIC
Packet Processing Engine Match-Action
Packet Buffers + traffic management (QoS etc)
Ethernet P0
Ethernet P1

Host Application: Network Processing Split



- Net Infrastructure(I) (yellow) - per packet processing
  - Message packetization
    - “Super-packet“ Segmentation/Reassembly (TSO/GSO/GRO)
  - Anything L2-4: ACLs, LB, QoS
  - Protocol processing (eg IP, TCP, TLS)
    - Header level + Protocol processing
- Application infrastructure(AI) (blue) - per msg processing
  - System call overhead
  - Data movement/Memory transfer + associated costs, locks, etc
- Application(A) (red)
  - Whatever CPU left is available for applications
  - Goal is to optimize so more resources available to application

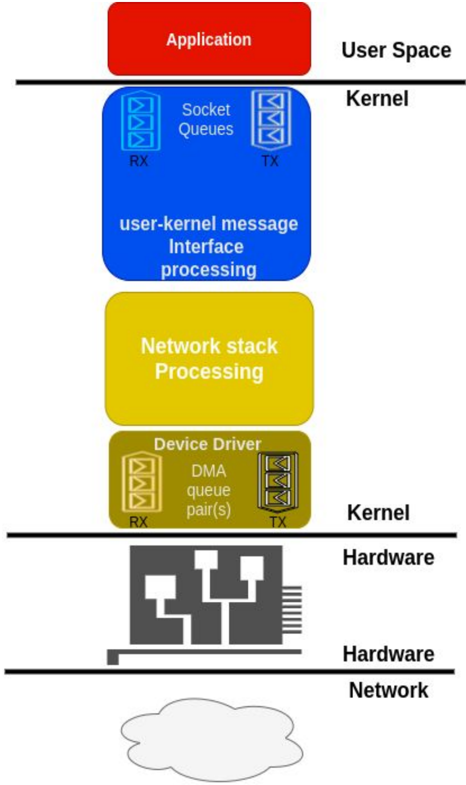
Desire On Where Host Application CPU Expenditure Goes



General I/DPU Offload Effect On Host CPU

On ingress, you hit I -/-> AI -> A and on egress A -> AI -> I

- Ingress, Offloading of I proved useful
    - We experimented with ACLs and TLS
        - 10-95% improvement on host CPU utilization (traffic dependent)
- Egress, you hit AI first then I
    - We experimented with TLS, more improvement to I
        - Further 5-15% CPU utilization improvement (traffic dependent)



Packet/Flow Patterns Affecting Performance

- Per packet processing overhead
  - Influenced by packets/sec
  - ‘How big / small the packets are?’
- Per flow state overhead
  - Influenced by flows/sec
  - ‘How long / short the flows are?’

Lesson: Effect of Application message size on host cpu+offload

Network stack infrastructure cost is per packet (header)

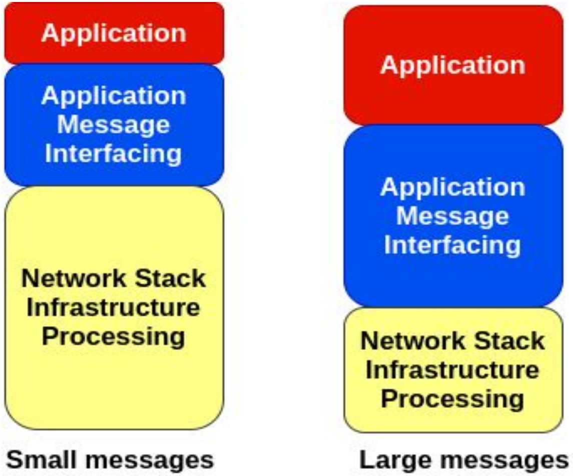
Meaning: The **higher** the pps, the more visible the cpu cost

@64B at 25G is 35Mpps, means I exercised 35M/second/direction

@1.4KB the rate is in the ~1.6Mpps, @4K ~500Kpps,@9KB ~250Kpps

=> **I bottleneck is reduced proportional to the packet size**

Lesson: Reduce the PPS, reduce per packet/header processing.


Application
Application Message Interfacing
Network Stack Infrastructure Processing
Small messages
Application
Application Message Interfacing
Network Stack Infrastructure Processing
Large messages

Offload of Infra implies more host resources released and available to applications and interfacing

=> Consequences: more payload data coming into the host kernel

- Bulk data apps like works well (e.g. Storage, ML training, file serving, backups, long lived RPCs)

Lesson: Effect of flow sizes on host cpu+offload

Flow sizes influence state setup/termination cost(STC)

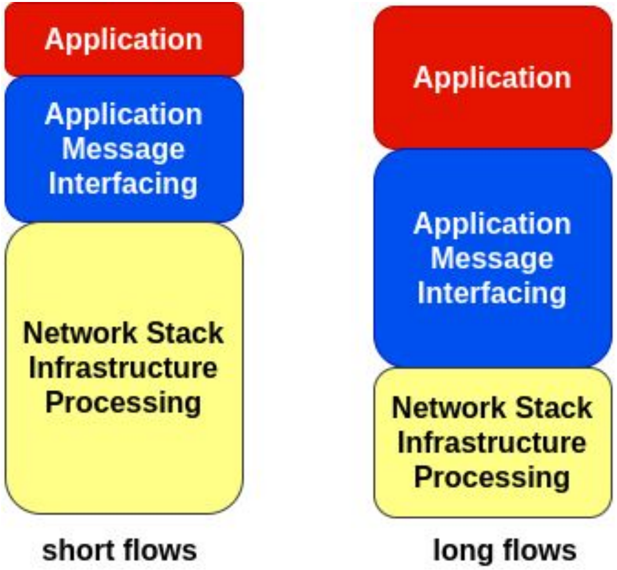
TCP costs: 5 packets (3 pkts setup, 2 to teardown)

Shorter TCP flows (less packets) => high STC/sec

(E.g. Bursty traffic)

Stateful flow offload gains for short flows less prominent

- Flow duration vs h/w setup time


Application
Application Message Interfacing
Network Stack Infrastructure Processing
short flows
Application
Application Message Interfacing
Network Stack Infrastructure Processing
long flows

Lesson: Most advantageous for longer flows

(E.g. Storage, ML training, file serving, backups, long lived RPCs)

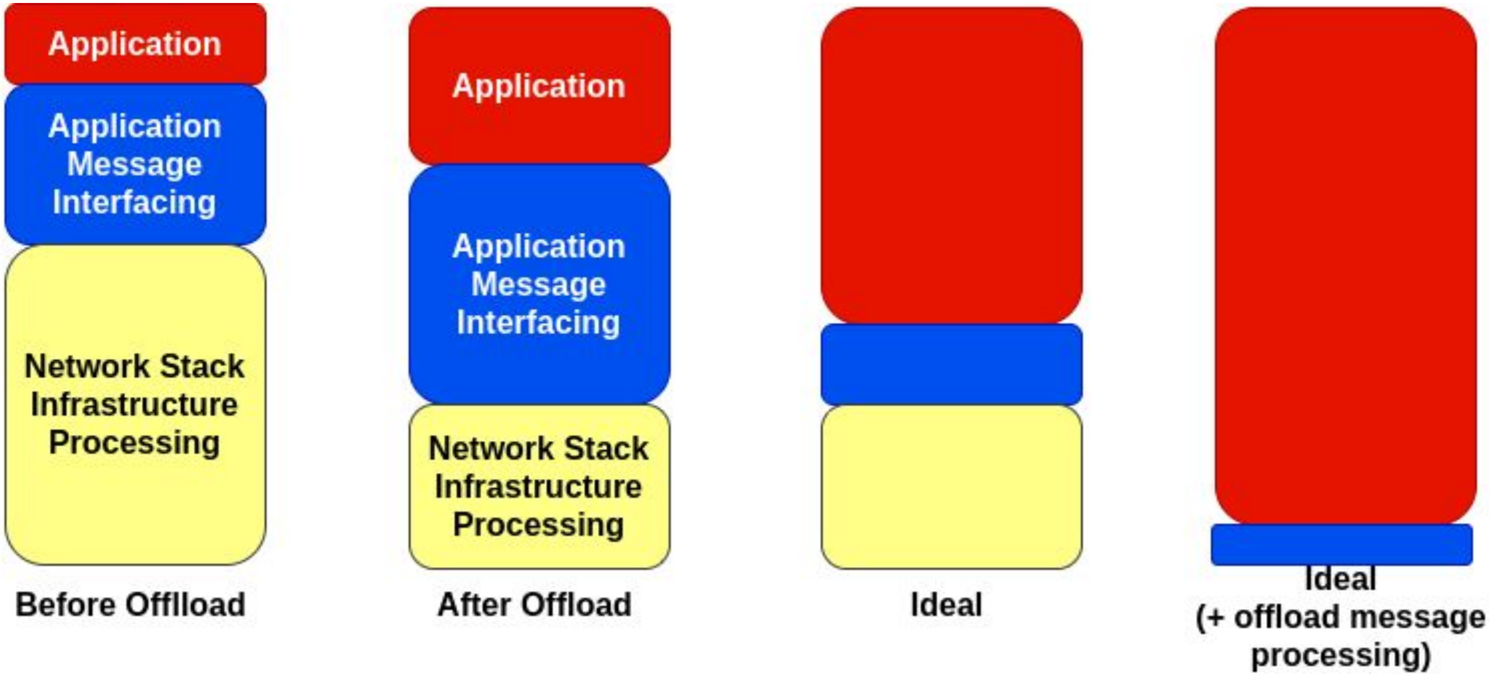
- Very amenable to offload
  - Amortize the cost of setup/teardown

Increasing Port Speeds : Host Networking Bottlenecks Taxonomy

CPU time for infrastructure and message interface work compete with application

- Increase In Data Throughput (bits/sec) view
    - Resources bottlenecks: PCI bus bandwidth, memory transactions throughput
- Increase in Packet Throughput (packets/sec) view
    - Resource bottlenecks: CPU capacity, PCI setup cost, memory access latency

Desire On Where Host Application CPU Expenditure Goes


Application
Application Message Interfacing
Network Stack Infrastructure Processing
Before Offlload
Application
Application Message Interfacing
Network Stack Infrastructure Processing
After Offload
Ideal
Ideal
(+ offload message processing)

Summary: Where Offload Helps

- Offloads help to reduce PCI traffic as well as CPU cycles spent on host
- Offloads are useful for high pps compute bound infrastructure processing
  - Transactions on headers, therefore cost/packet processed same regardless of payload size
  - Even then: offload is useful if you have CPU resource challenges
    - Example: If you are running at 20% cpu utilization, saving another 2-3% needs to be weighed against operational effort/capex cost
    - Cloud vendors pack workloads on hosts; meaning higher resources utilization
      - The value is: “keeping the servers working longer” or “use less machines in DC”
- Offloads are useful for memory bound infrastructure processing
  - Table lookups at low latency and high throughput
  - Stateless transactions (ACLs, LB decisions)
  - Reduces memory pressure on host CPU’s mem controller
    - Being able to drop packets early without copying to host
    - Being able to preprocess data and place directly into application or device memory

Overcoming The Host Message Interface Costs

Motivation: Why Is This More Relevant Now?

Exponential growth of network traffic => more data coming into/out of the host

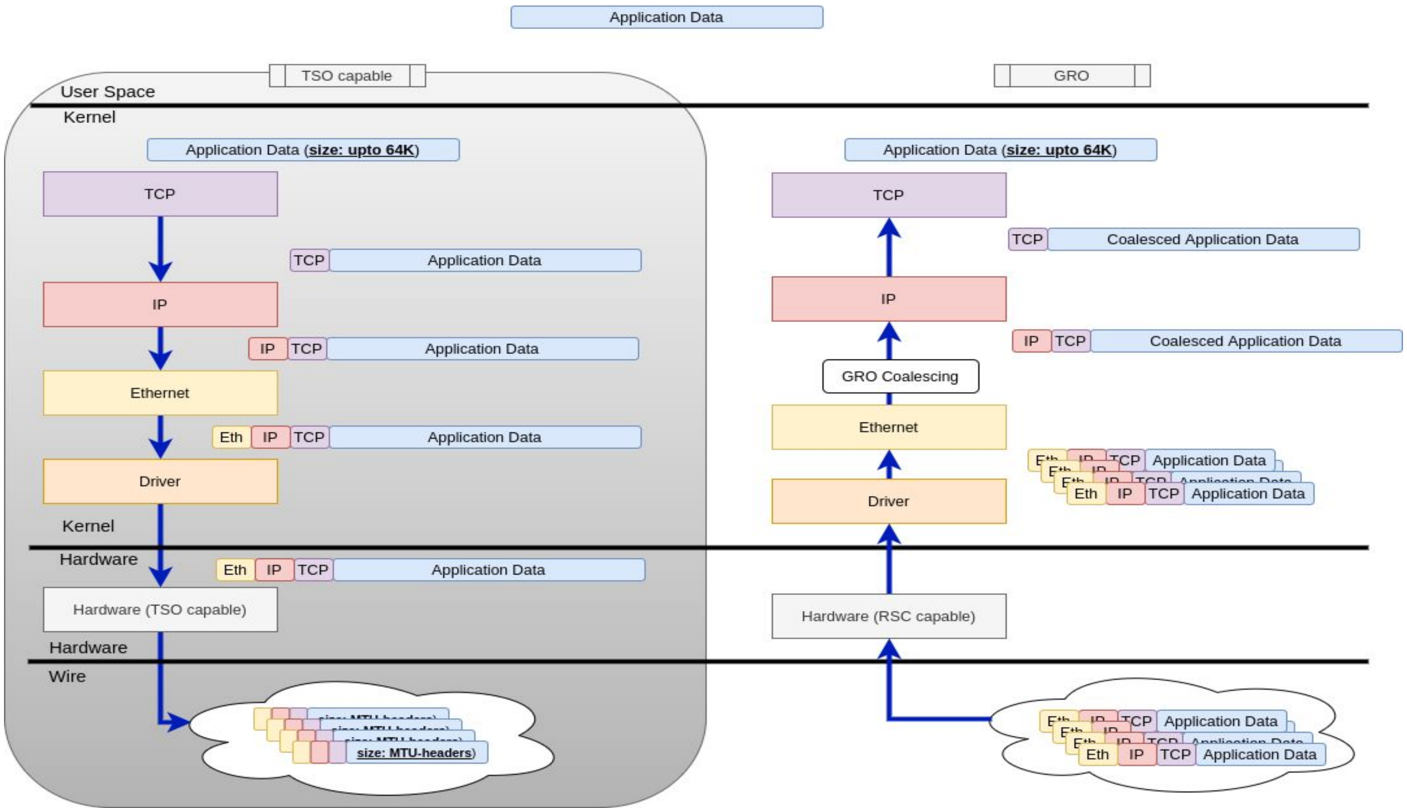
- Primarily driven by **ML** training
  - 800 Gbps ratified (NICs appearing on RFPs and POCs..)
  - Talk of 1-1.6 Tbps ports (Broadcom claim)

PPS Reduction Techniques

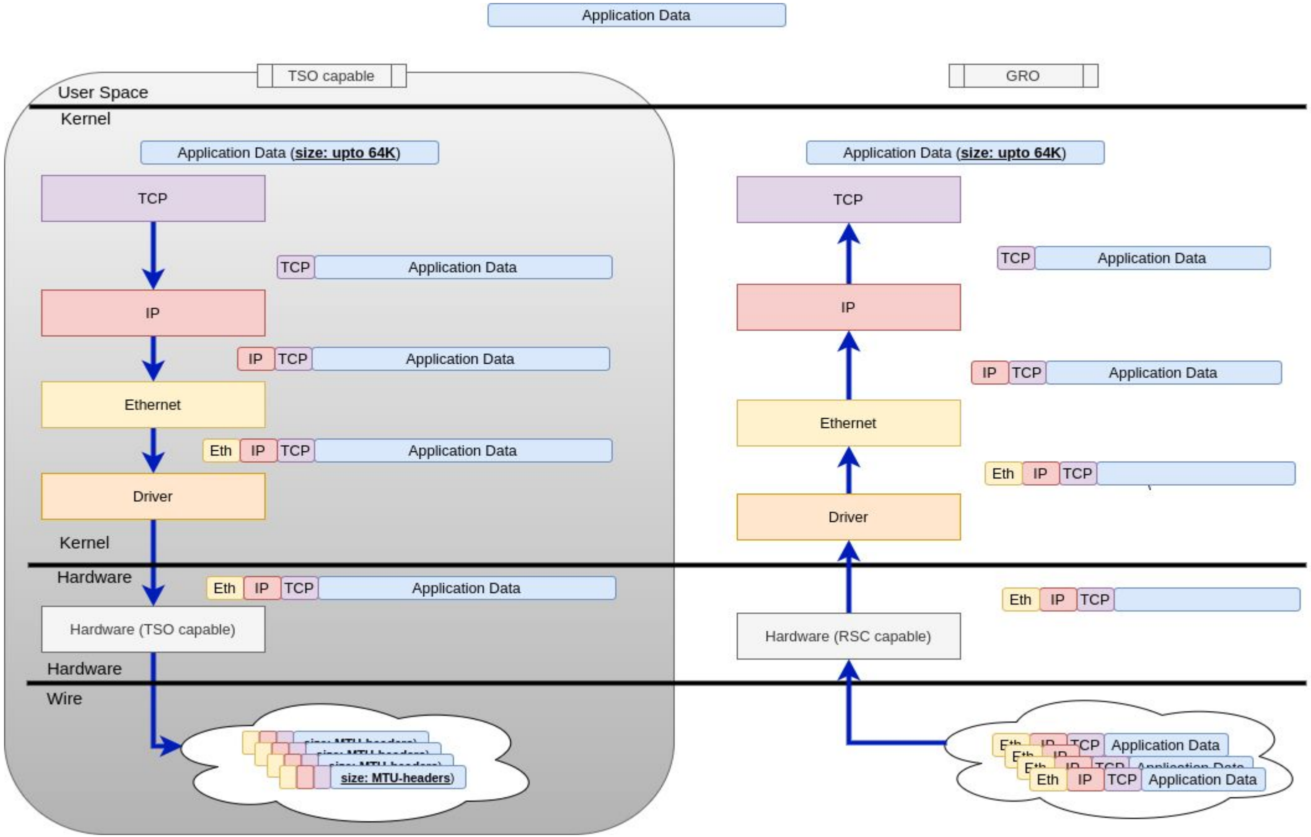
Solutions Not Requiring Application Change

- App Message packetizations: Large payloads SG with single header
  - GRO/RSC (offload* packet->message coalescing)
  - TSO (offload message->packet segmentation)
  - Big TCP (glorified TSO/GRO)
- Basic offload of per packet processing
  - Checksums
- Reduced PPS approaches
  - Increase MTU
    - At 4K MTU, 100Gbps is 2.25Mpps
- Cache hit improvements
  - Interrupt moderation
  - NAPI
  - Batching of packets

TSO And S/W GRO (PPS reduction)


Application Data
TSO capable
GRO
User Space
Kernel
Application Data (size: upto 64K)
TCP
TCP Application Data
IP
IP TCP Application Data
Ethernet
Eth IP TCP Application Data
Driver
Kernel
Hardware
Eth IP TCP Application Data
Hardware (TSO capable)
Hardware
Wire
size: MTU-headers
Application Data (size: upto 64K)
TCP
TCP Coalesced Application Data
IP
IP TCP Coalesced Application Data
GRO Coalescing
Ethernet
Eth IP TCP Application Data
Driver
Hardware (RSC capable)
Eth IP TCP Application Data

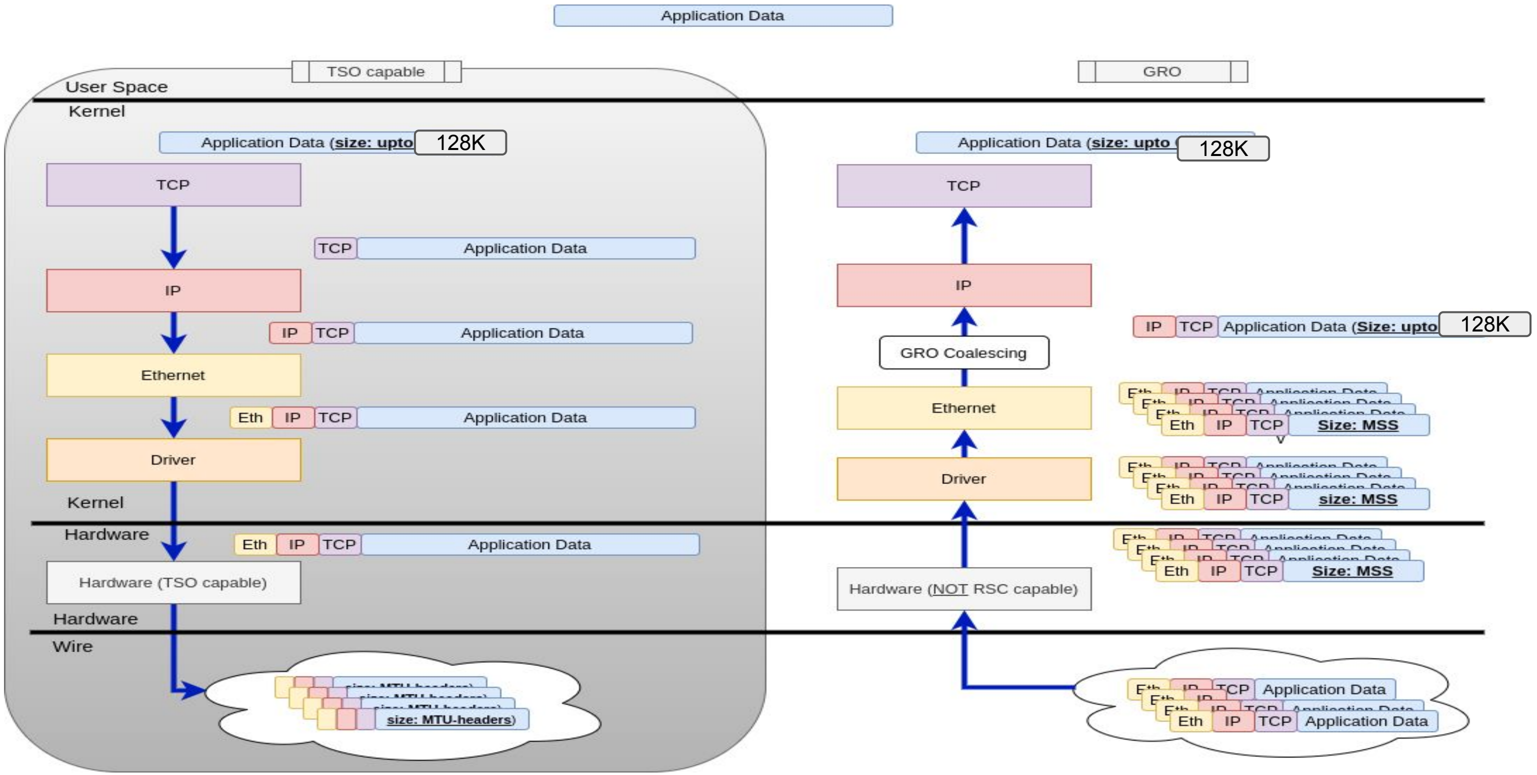
TSO And RSC (PPS reduction)



NICs known to support RSC

- Intel ixgbe(10G)
  - Known to be buggy
- Intel IPU (Multiple 100G)
  - We share results
- There could be others

Big TCP



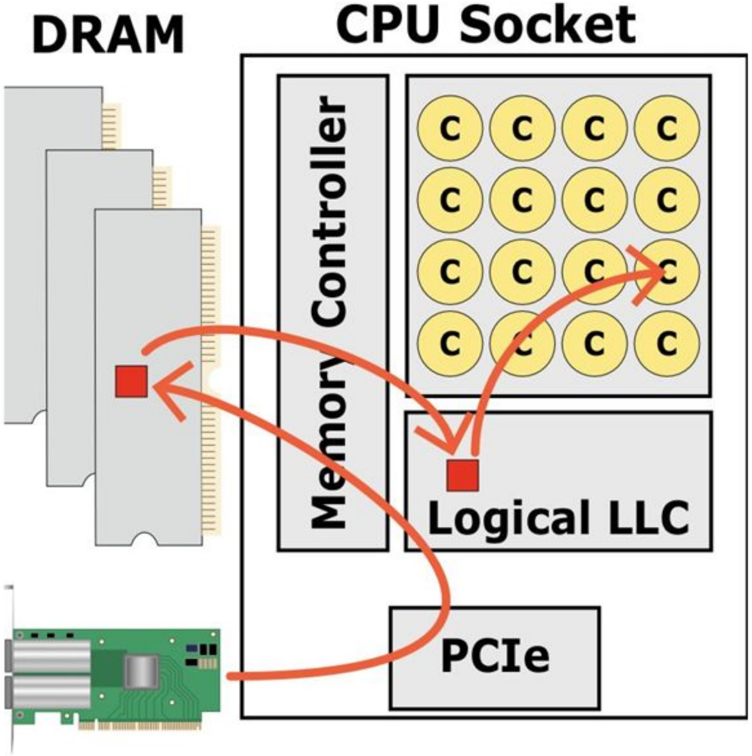
See: https://netdevconf.info/0x15/session.html?BIG-TCP

Cons on ‘Reduced PPS Techniques’

- Not suitable for low latency in general
- Require network tuning for optimal performance
  - Either MTU size or sysctl knobs
- Opportunistic TSO/RSC/GRO
  - When the network goes berserk

Data Movement Reduction Techniques Via Copy Elimination

Device To/From Application

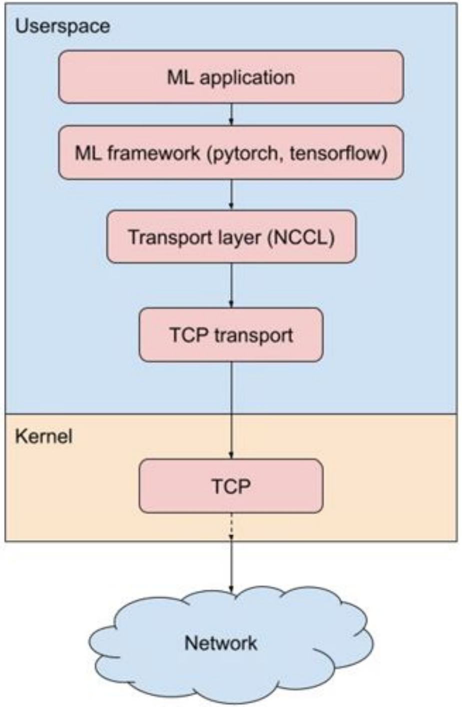


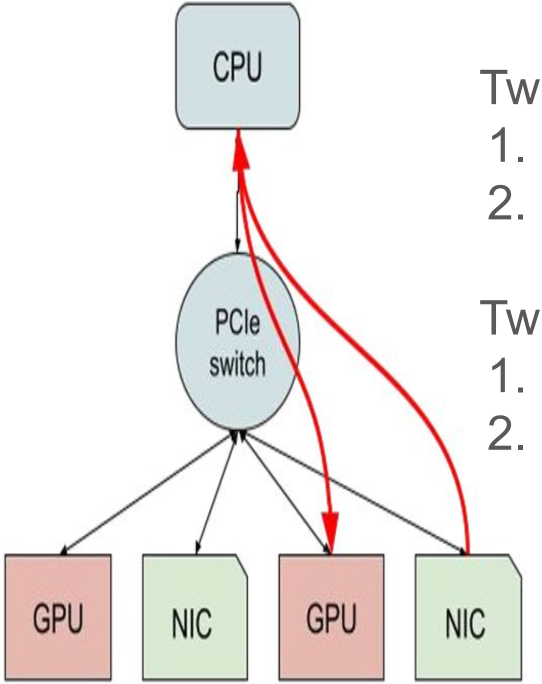
A. System call overhead
B. RX: Two copies
   1. From NIC to host kernel
   2. From kernel to user space

Reverse is true for TX direction

Diagram Source: Netdev conf 0x17 talk: **Fast ZC Rx Data Plane using io uring**

Current Anatomy Of A Server Doing ML Training





Two copies on incoming

1. From NIC to host kernel
2. From kernel to GPU

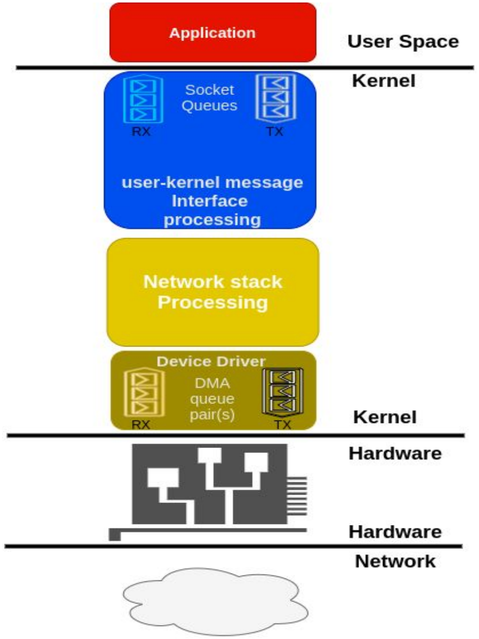
Two copies on outgoing

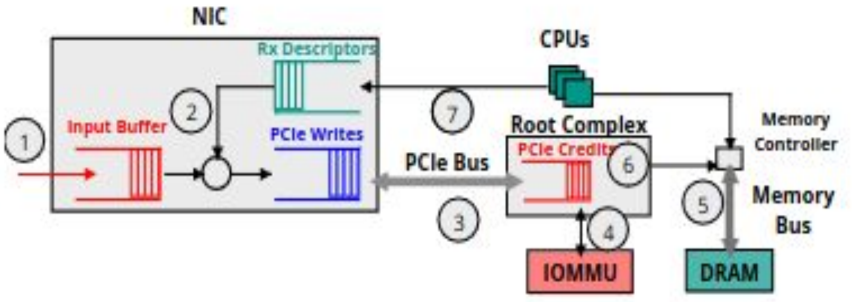
1. From GPU to kernel
2. From Kernel To NIC

Google Cloud A3 VMs (8 H100 GPUs)
Nvidia DGX H100 Systems (8 H100 GPUs)

Netdev conf 0x17: **Device Memory TCP {El-Masry et al}**

Host RX Packet Flow





1. Host driver allocates "DMA-able" memory and specifies it in Rx descriptors (step 0)
2. Packet arrives at NIC, enqueued in NIC buffer (step 1)
3. NIC fetches an Rx descriptor which specifies where to DMA
4. NIC instantiates PCIe write transaction (credit based)
5. The root complex translates virtual to physical memory address for the Rx descriptor address using IOMMU
6. PCIe root complex DMA's the data to the host memory over the memory bus
7. Root complex replenishes NIC PCIe Credits
8. NIC interrupts the host CPU
9. Driver invoked initiates stack processing of the protocol headers and replenishes the rx descriptors
10. Application notified and data copied over to user space

NIC Diagram from: Rachit A. et al: {netdev 0x17, sigcomm 2023, hotnets 2022}

Application Changes Required: Zero Copy RX/TX

- Total Kernel bypass: DPDK and friends
    - Our target application requires TCP/IP stack which is bypassed
        - Resolvable by running alternative(to kernel) network stack
            - Tend to be very limited. Linux net stack took many years to get to its stability
- Semi-kernel bypass: AF_XDP
    - Has option to allow some traffic to be exercised by kernel processing for the network stack
        - But such apps are still going via traditional in-efficient kernel path
        - So not useful for our goal
- Sendfile
    - zero copy transmits, mostly used in files->sockets
- recvmsg/sendmsg enhancements or alternatives
    - RX Requires header and payload separation
        - Headers handled by the stack
        - Payload goes directly to application
        - Page flipping
    - For TX side requires scatter-gather support in HW (available in most drivers)
    - Two approaches (msg zc and io uring)
        - The page pinning

Cons on Zero Copy

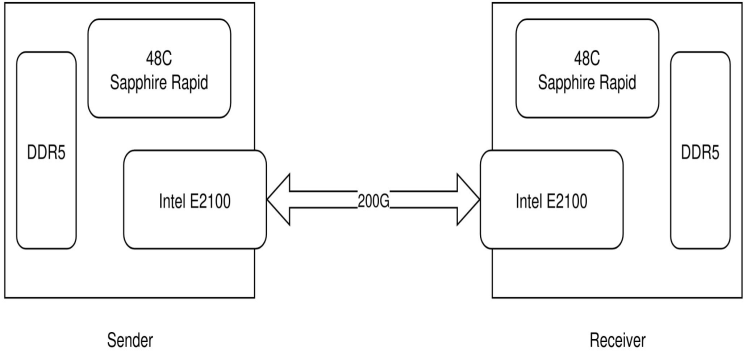
- Kernel based offload of L7 processing breaks down
    - Example KTLS falls apart because it operates on payload and would need to be done in user space
    - If h/w is not capable of checksum offload and the kernel computes it instead
        - HW csum offload must have feature
- Past studies have shown that “Page Flipping” is only beneficial with “large enough” payload sizes
- Requires non trivial application changes
- Requires network tuning for optimal performance
    - Either MTU size or sysctl knobs

Setup And Target Traffic Patterns

Traffic Patterns Used

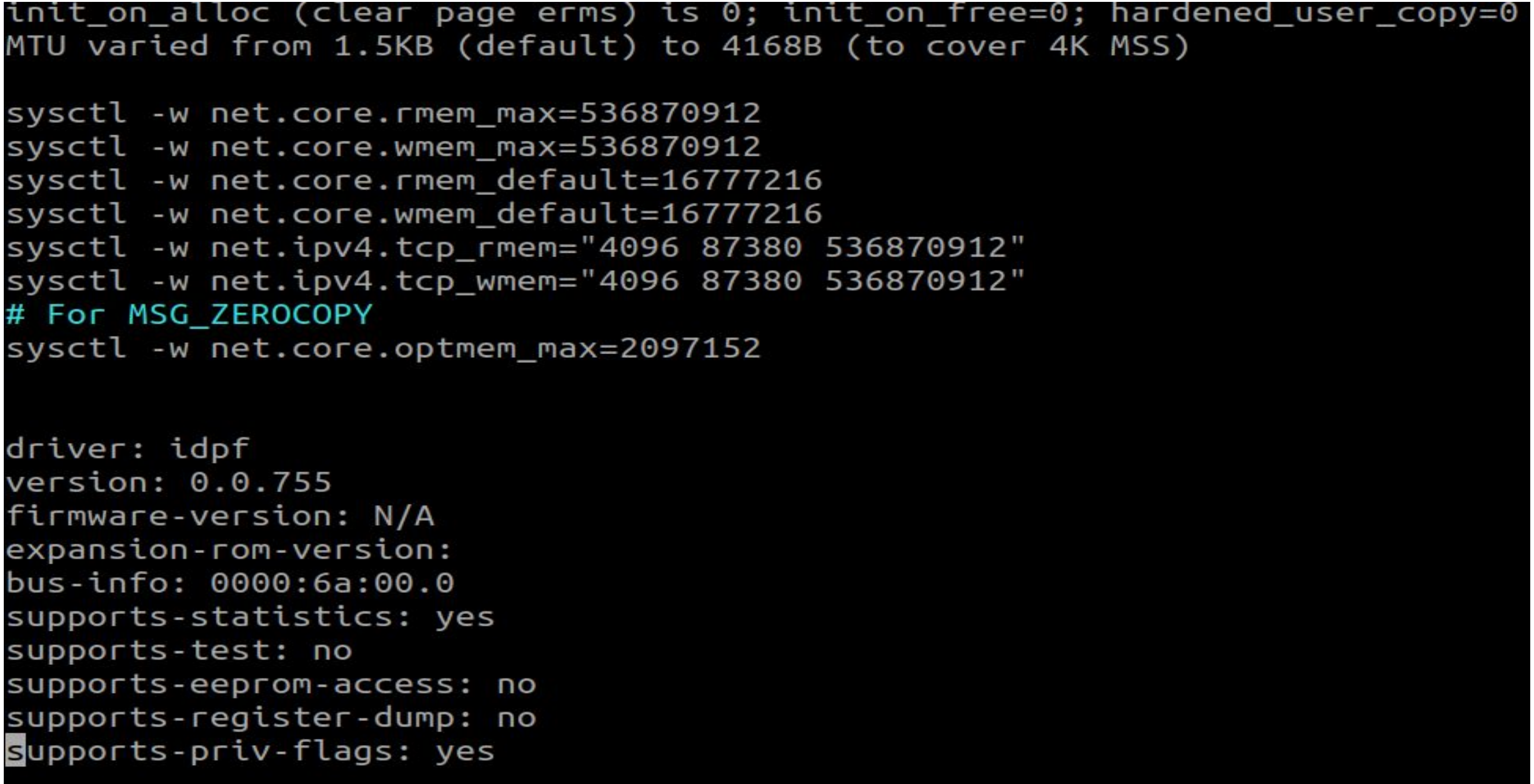
- Saturate the link capacity
  - 16 flows over two links
- Asymmetric Client-Server
  - IPv6
  - Very long flow analogous to Netperf TCP_STREAM
  - Tests the transmit processing on sender side and receiver processing on receiver side
  - 512KB app messages
- TCP_RR for transactional BIG TCP
  - IPv6
  - Using Netperf TCP_RR

Setup



- CPU: Intel(R) Xeon(R) Platinum 8468 48C
  - Hyper-Threading off
    - Use only 16 cores on each
      - IRQ and Applications bound to cpuset 0-15
  - Dual port connection back to back
    - Both active
    - SG, checksum offload, TSO
    - Ring size: RX:2048 TX:512
- RAM: DDR5 128GB 4800 MT/s
- Intel IPU E2100
  - 200G on configuration 2x100G
  - Header Split on
- Test duration: 100 seconds (typically > 1TB of data exchanged, so any glitches even out)
  - Repeated 3 times and averaged
- Governor mode: starts in powersave mode but ramps up within 100 seconds
- External power measurement

General settings



```
init_on_alloc (clear page erms) is 0; init_on_free=0; hardened_user_copy=0
MTU varied from 1.5KB (default) to 4168B (to cover 4K MSS)

sysctl -w net.core.rmem_max=536870912
sysctl -w net.core.wmem_max=536870912
sysctl -w net.core.rmem_default=16777216
sysctl -w net.core.wmem_default=16777216
sysctl -w net.ipv4.tcp_rmem="4096 87380 536870912"
sysctl -w net.ipv4.tcp_wmem="4096 87380 536870912"
# For MSG_ZEROCOPY
sysctl -w net.core.optmem_max=2097152


driver: idpf
version: 0.0.755
firmware-version: N/A
expansion-rom-version:
bus-info: 0000:6a:00.0
supports-statistics: yes
supports-test: no
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: yes
```

Setup

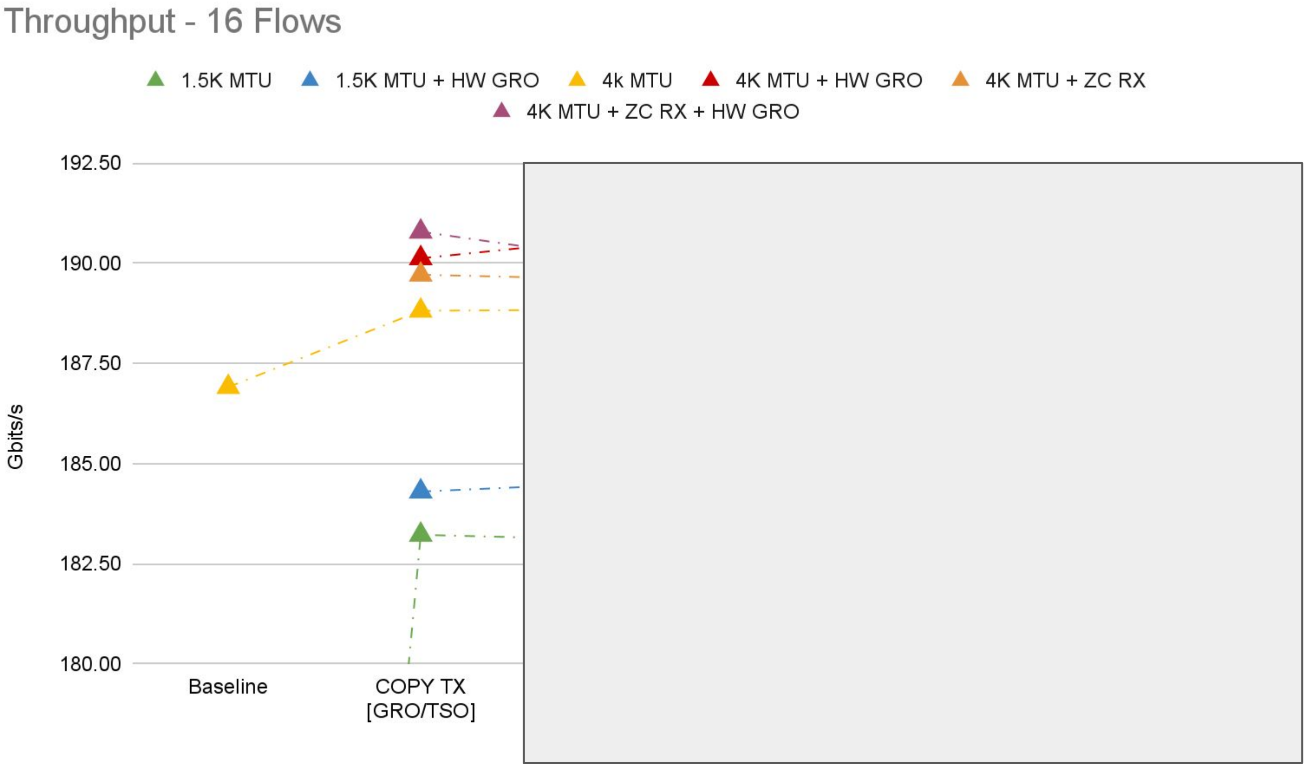
- Interested in 3 metrics
  - Throughput, Power Consumption and CPU Usage
  - Results computed based on these 3 metrics
- Power consumption measured by an external device
  - Server as a ‘black box’
  - Baseline power around 200W
- Networking bound to 16 Cores
  - Programs bound to the same 16 Cores
  - 16 flows per run
    - RSS (toeplitz) distribution
    - Boils down to about 8 flows per port
- We use our own home-brewed test program
  - Heavily modified version of tcp_mmap selftest

Experiments

Experiment Matrix

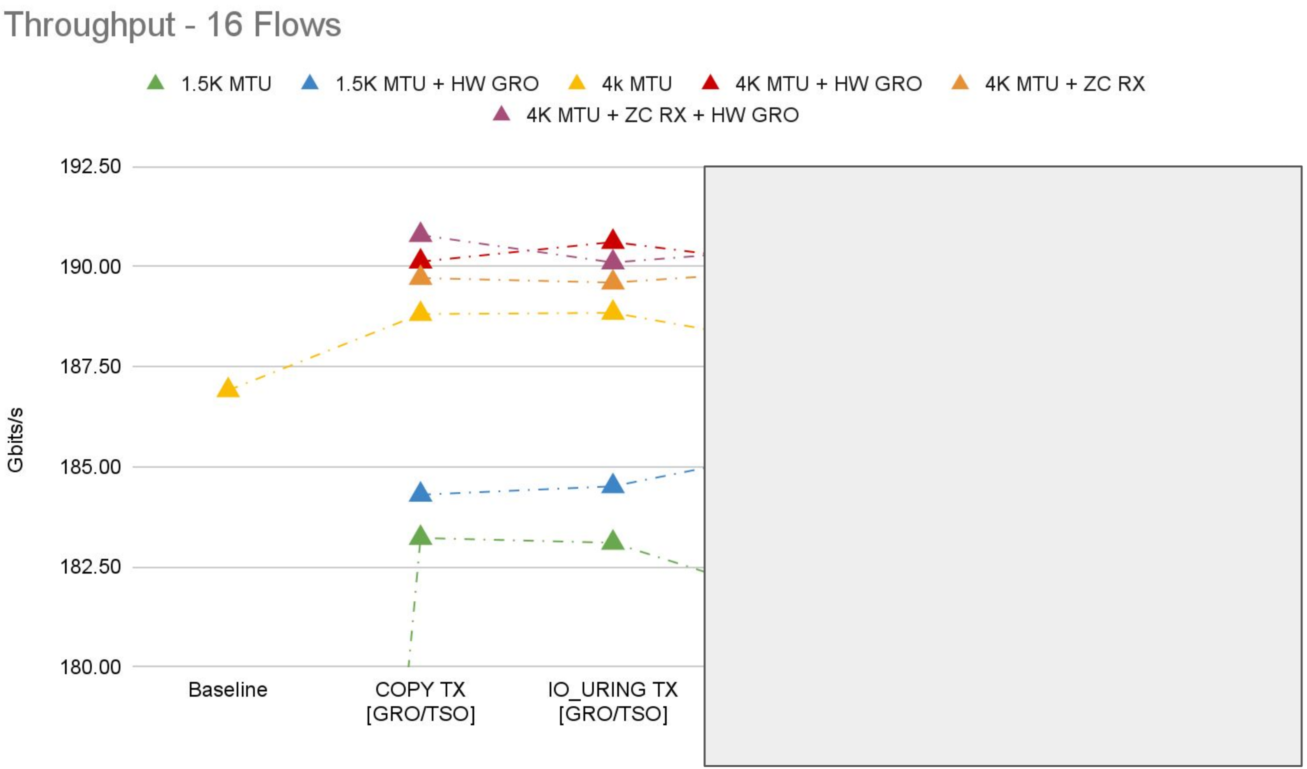
- MTU varied
  - 1.5K
  - 4168B (4K MSS)
- Receiver side setup
  - Baseline: No GRO/TSO {both MTU values}
  - SW GRO + HW TSO {both MTU values} - 64KiB, and 128KiB for bigTCP
  - HW GRO + HW TSO {both MTU values}
  - Zero Copy {only 4K MSS to fit on a memory page}
- Sender side {both MTU values}
  - Baseline: No GRO/TSO
  - GRO _ HW TSO - 64KiB, and 128KiB for bigTCP
  - IO_URING
  - IO_URING {zero copy}
  - Sendfile {zero copy}
  - MSG_Zerocopy

Throughput

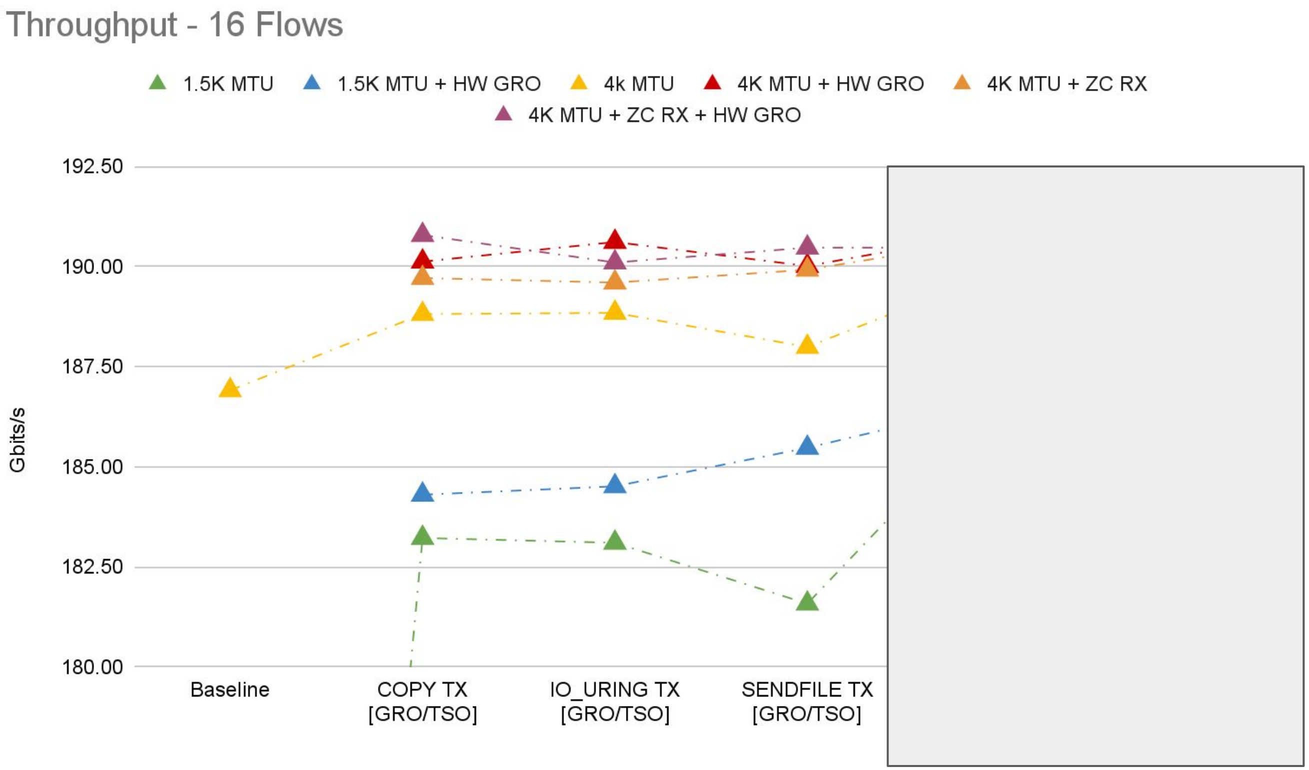

Throughput - 16 Flows
1.5K MTU
1.5K MTU + HW GRO
4k MTU
4K MTU + HW GRO
4K MTU + ZC RX
4K MTU + ZC RX + HW GRO
192.50
190.00
187.50
185.00
182.50
180.00
Gbits/s
Baseline
COPY TX [GRO/TSO]

Throughput

HW GSO aka TSO (always)

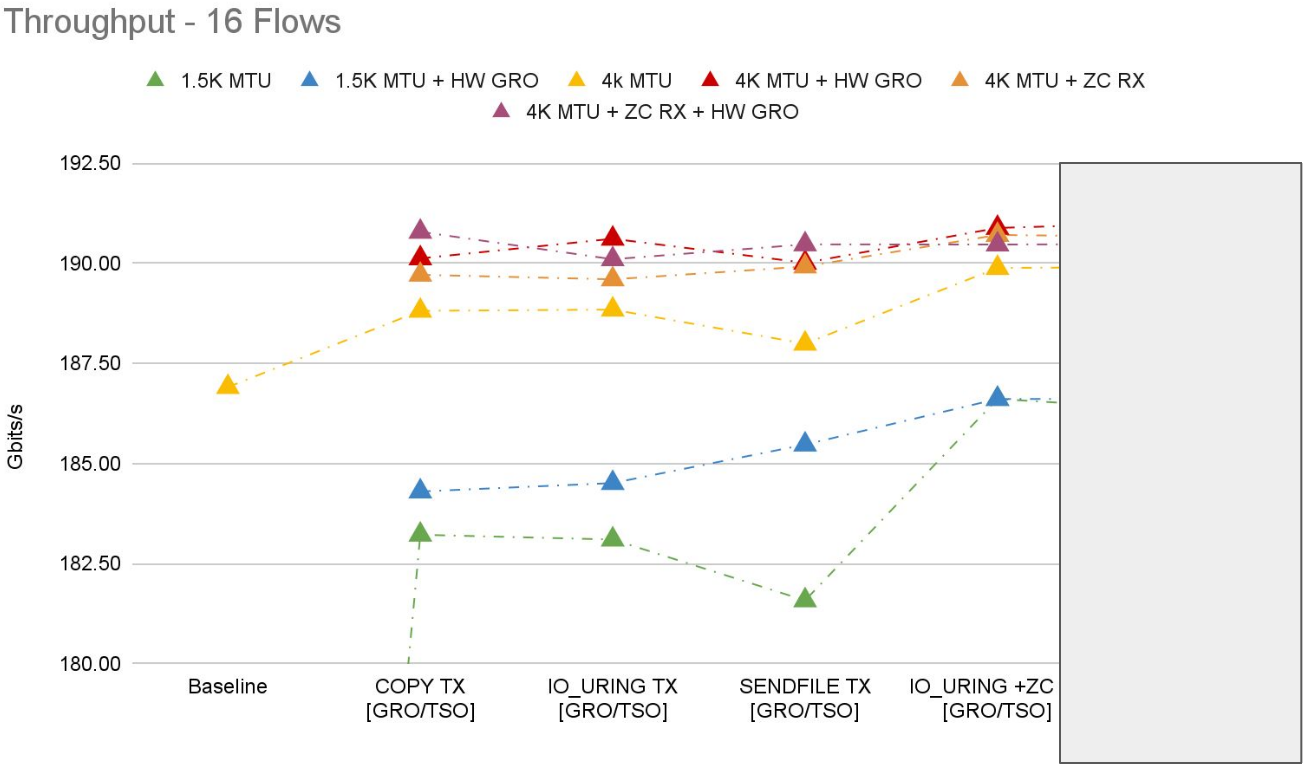


Throughput

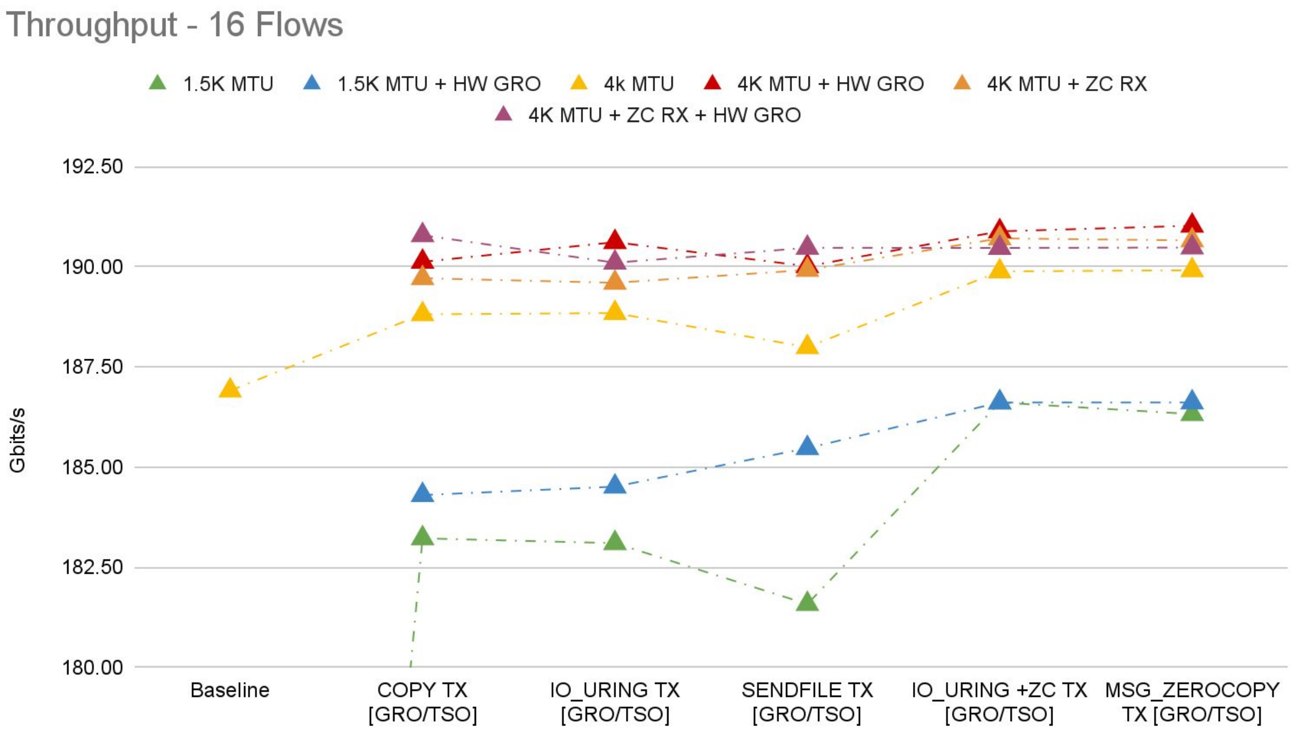


Throughput - 16 Flows
1.5K MTU
1.5K MTU + HW GRO
4k MTU
4K MTU + HW GRO
4K MTU + ZC RX
4K MTU + ZC RX + HW GRO
192.50
190.00
187.50
185.00
182.50
180.00
Gbits/s
Baseline
COPY TX [GRO/TSO]
IO_URING TX [GRO/TSO]
SENDFILE TX [GRO/TSO]

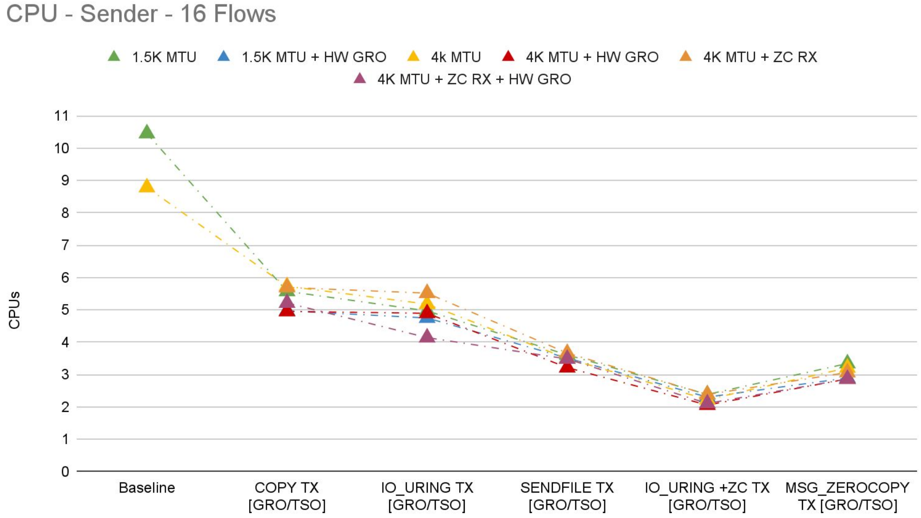
Throughput


Throughput - 16 Flows
▲ 1.5K MTU
▲ 1.5K MTU + HW GRO
▲ 4k MTU
▲ 4K MTU + HW GRO
▲ 4K MTU + ZC RX
▲ 4K MTU + ZC RX + HW GRO
192.50
190.00
187.50
185.00
182.50
180.00
Gbits/s
Baseline
COPY TX [GRO/TSO]
IO_URING TX [GRO/TSO]
SENDFILE TX [GRO/TSO]
IO_URING +ZC [GRO/TSO]

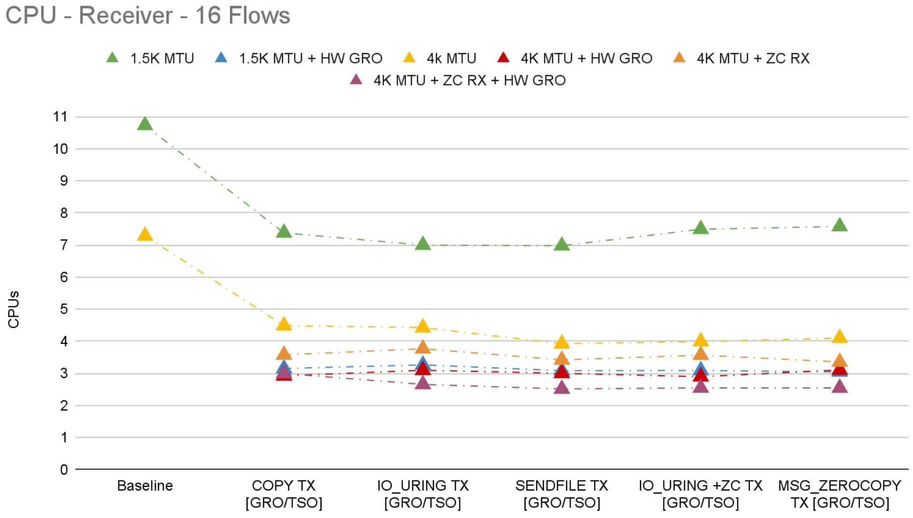
Throughput


Throughput - 16 Flows
▲ 1.5K MTU
▲ 1.5K MTU + HW GRO
▲ 4k MTU
▲ 4K MTU + HW GRO
▲ 4K MTU + ZC RX
▲ 4K MTU + ZC RX + HW GRO
192.50
190.00
187.50
185.00
182.50
180.00
Gbits/s
Baseline
COPY TX [GRO/TSO]
IO_URING TX [GRO/TSO]
SENDFILE TX [GRO/TSO]
IO_URING +ZC TX [GRO/TSO]
MSG_ZEROCOPY TX [GRO/TSO]

CPU Usage

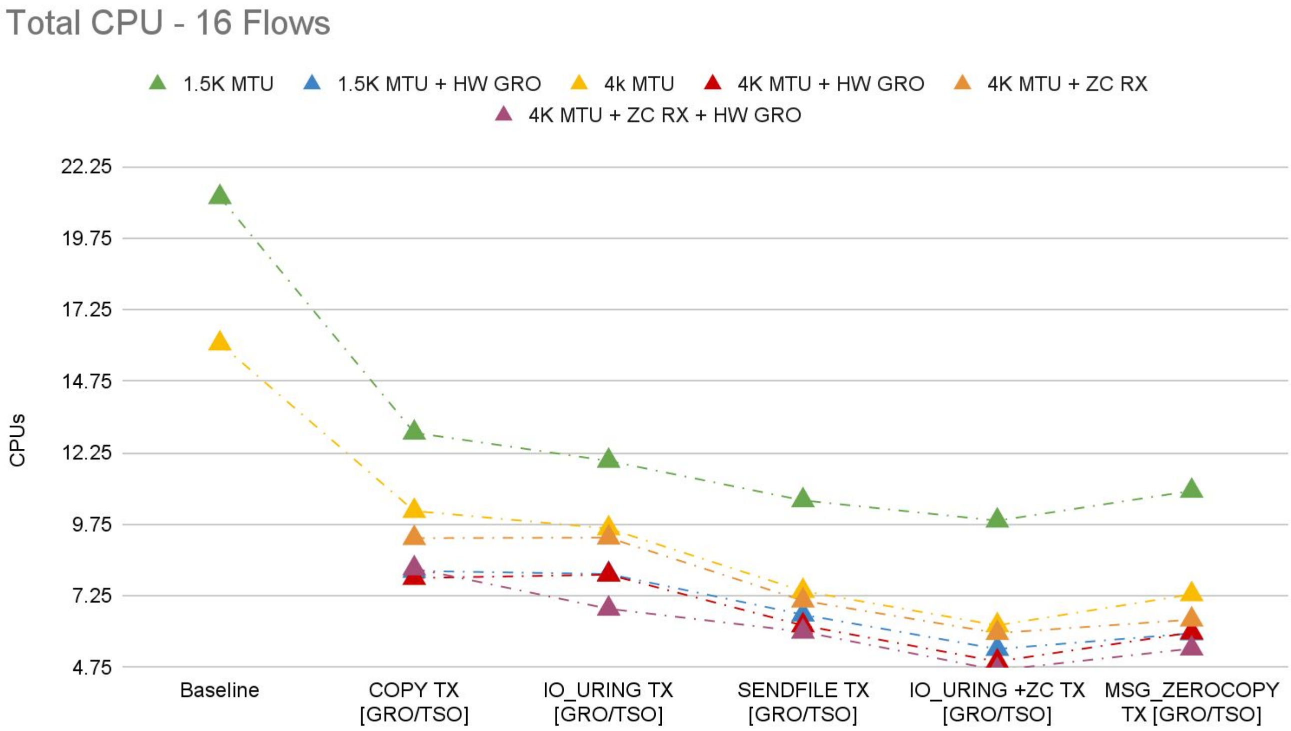


CPU - Sender - 16 Flows
1.5K MTU
1.5K MTU + HW GRO
4k MTU
4K MTU + HW GRO
4K MTU + ZC RX
4K MTU + ZC RX + HW GRO
CPUs
0
1
2
3
4
5
6
7
8
9
10
11
Baseline
COPY TX [GRO/TSO]
IO_URING TX [GRO/TSO]
SENDFILE TX [GRO/TSO]
IO_URING +ZC TX [GRO/TSO]
MSG_ZEROCOPY TX [GRO/TSO]

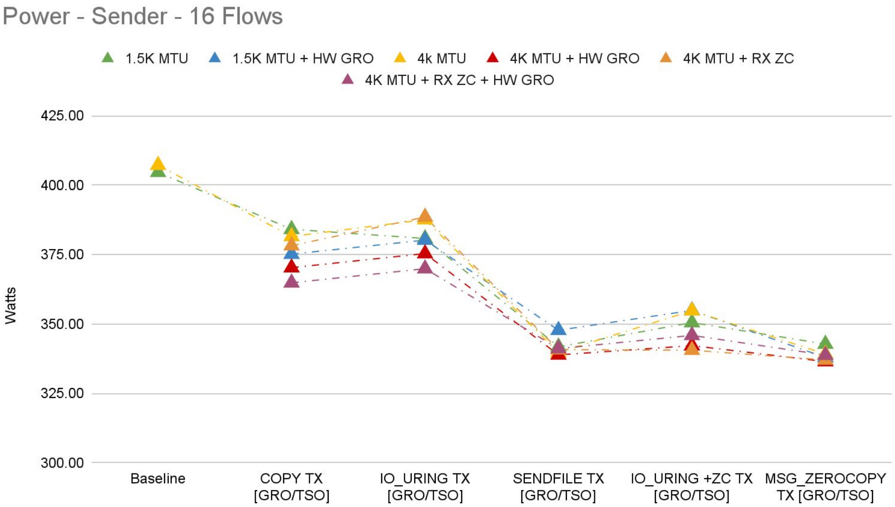


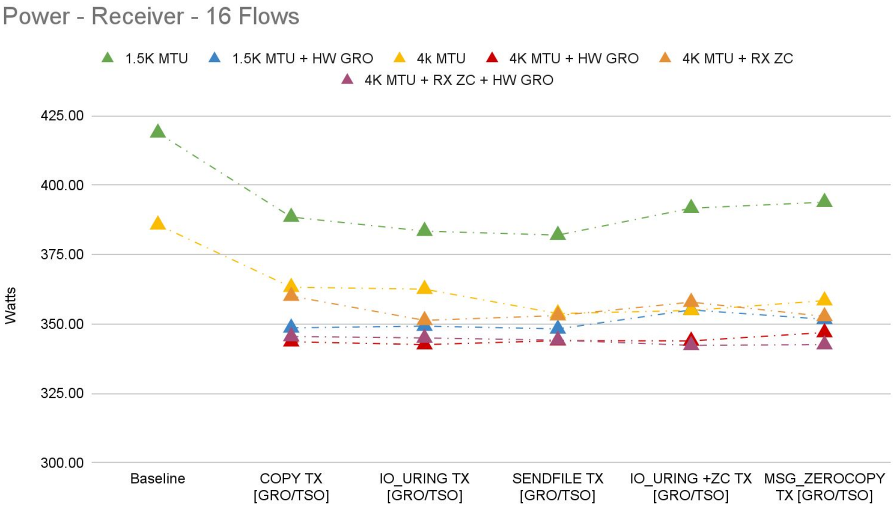
CPU - Receiver - 16 Flows
1.5K MTU
1.5K MTU + HW GRO
4k MTU
4K MTU + HW GRO
4K MTU + ZC RX
4K MTU + ZC RX + HW GRO
CPUs
0
1
2
3
4
5
6
7
8
9
10
11
Baseline
COPY TX [GRO/TSO]
IO_URING TX [GRO/TSO]
SENDFILE TX [GRO/TSO]
IO_URING +ZC TX [GRO/TSO]
MSG_ZEROCOPY TX [GRO/TSO]

Total CPU Usage

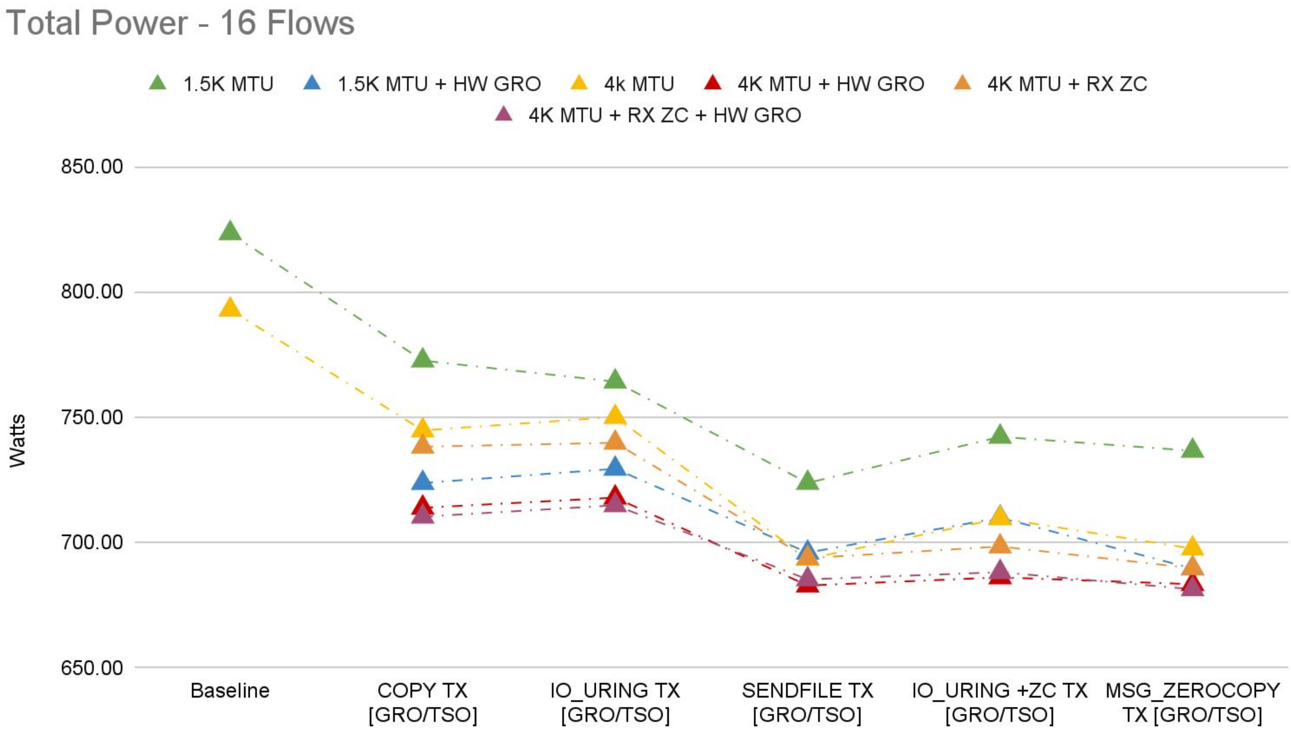


Power Usage


Power - Sender - 16 Flows
1.5K MTU
1.5K MTU + HW GRO
4k MTU
4K MTU + HW GRO
4K MTU + RX ZC
4K MTU + RX ZC + HW GRO
Watts
425.00
400.00
375.00
350.00
325.00
300.00
Baseline
COPY TX [GRO/TSO]
IO_URING TX [GRO/TSO]
SENDFILE TX [GRO/TSO]
IO_URING +ZC TX [GRO/TSO]
MSG_ZEROCOPY TX [GRO/TSO]


Power - Receiver - 16 Flows
1.5K MTU
1.5K MTU + HW GRO
4k MTU
4K MTU + HW GRO
4K MTU + RX ZC
4K MTU + RX ZC + HW GRO
Watts
425.00
400.00
375.00
350.00
325.00
300.00
Baseline
COPY TX [GRO/TSO]
IO_URING TX [GRO/TSO]
SENDFILE TX [GRO/TSO]
IO_URING +ZC TX [GRO/TSO]
MSG_ZEROCOPY TX [GRO/TSO]

Total Power Usage



ROI Performance Metrics

When comparing different results for throughput, it is often hard to say which results gives you the best return on investmen(ROI). We came up with two ROI formulas:

$$\frac{T^n}{\sum C_i}$$

T=Throughput achieved in Gbp/s. n=2

C=compute cost to achieve the throughput. Summed across all CPUs

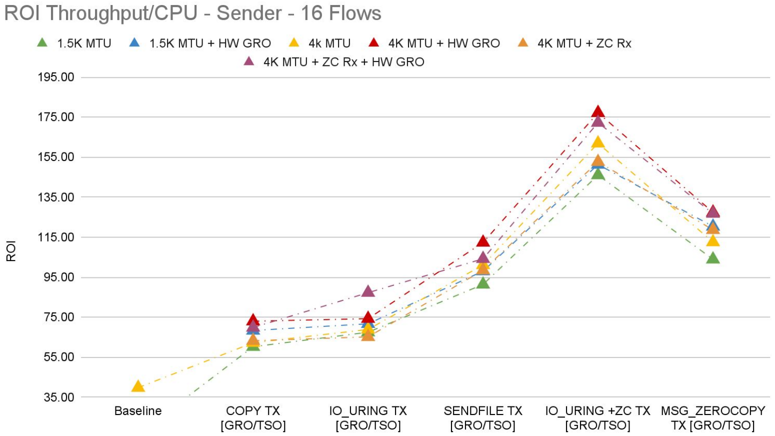
$$\frac{T^n}{Pm}$$

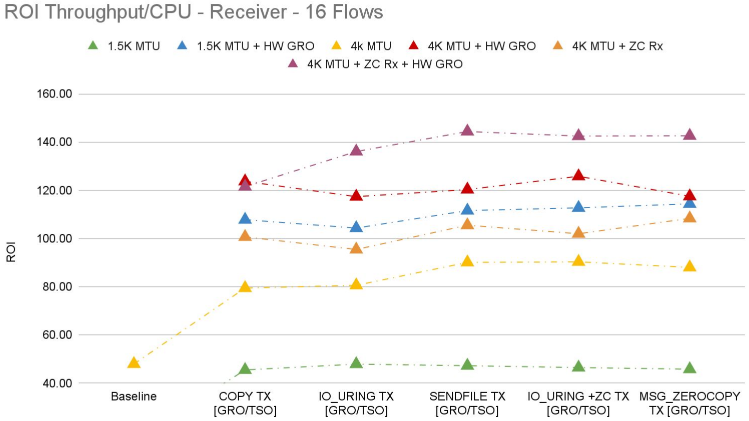
T=Throughput achieved in Gbp/s. n=2

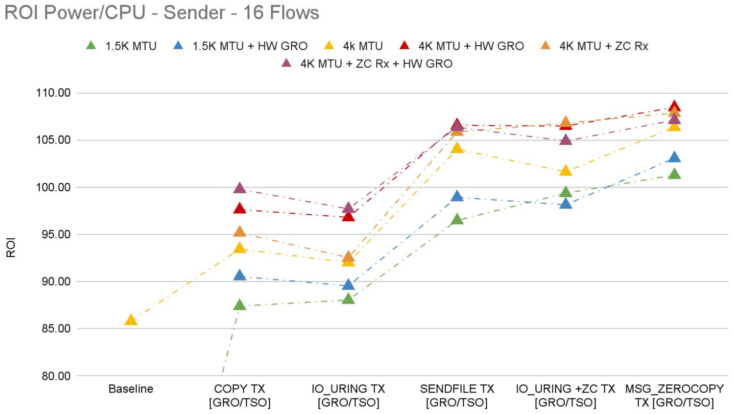
P=Power consumed by in watts. m=1

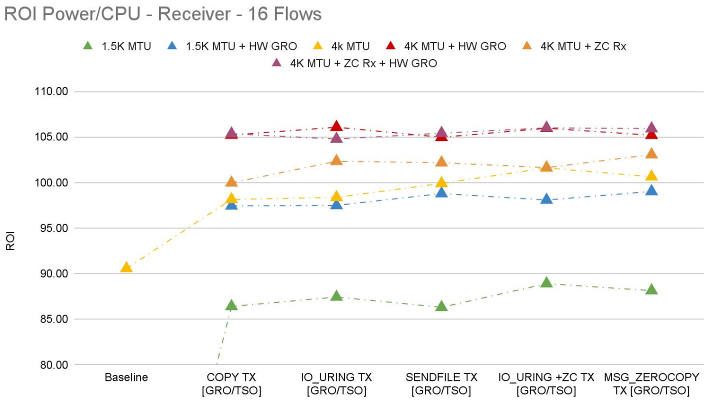
We pick n=2 to emphasize throughput as the goal. So 10 gbps using 200% cpu is considered to be better ROI than 5 gbps using 100% cpu

ROIs (The higher the better)

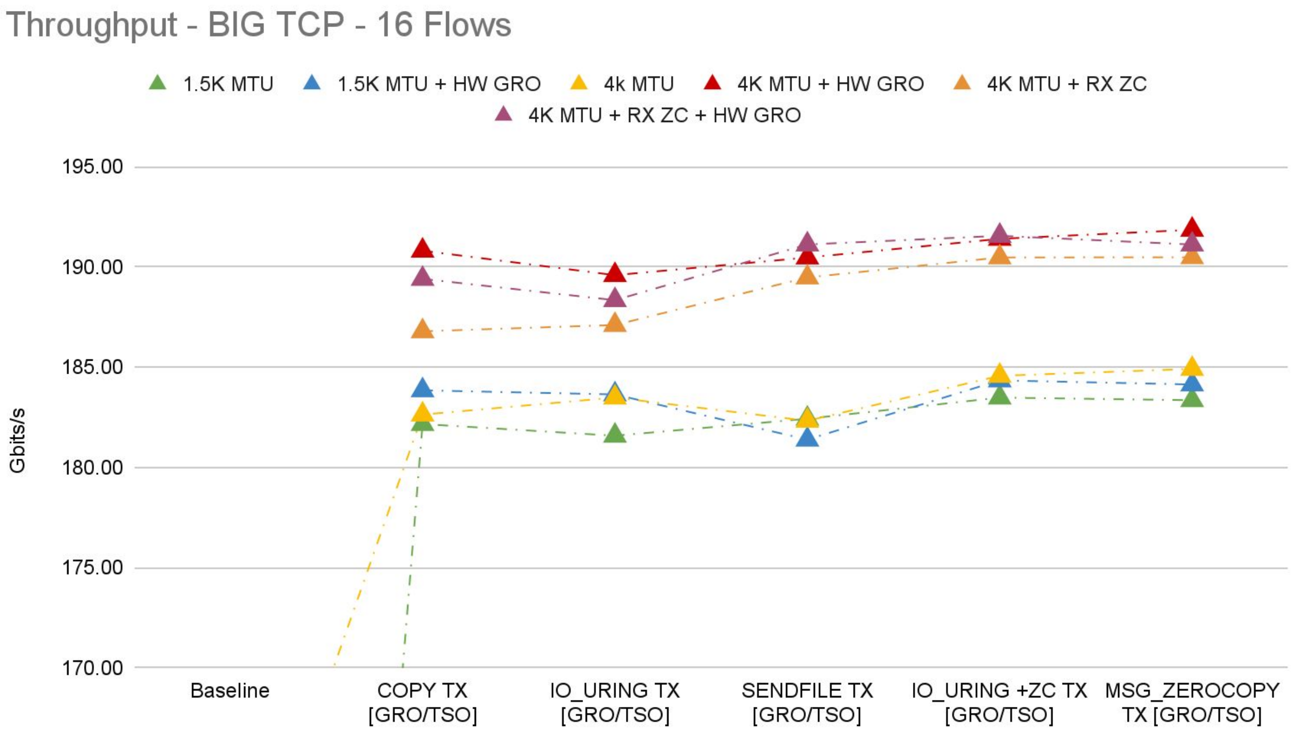




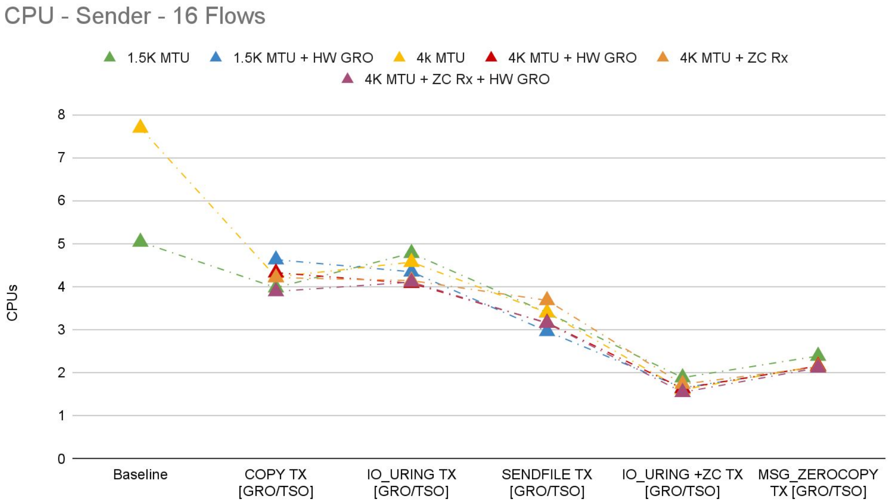


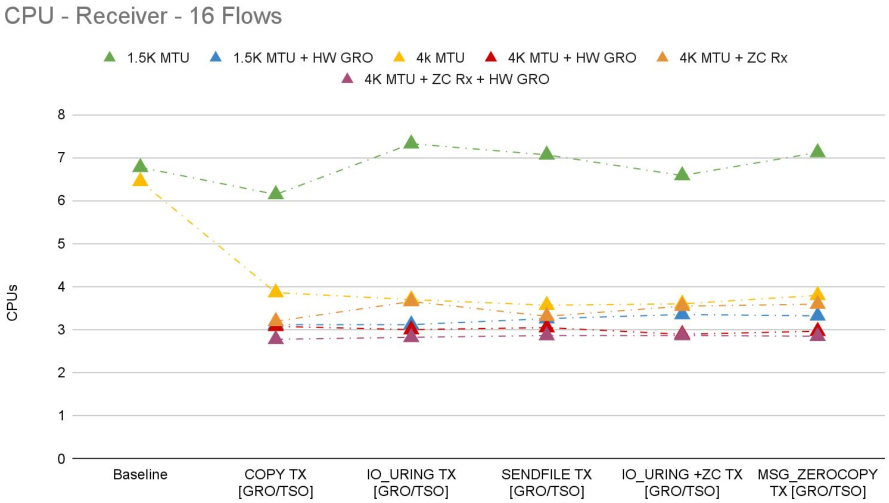


Throughput +BIGTCP

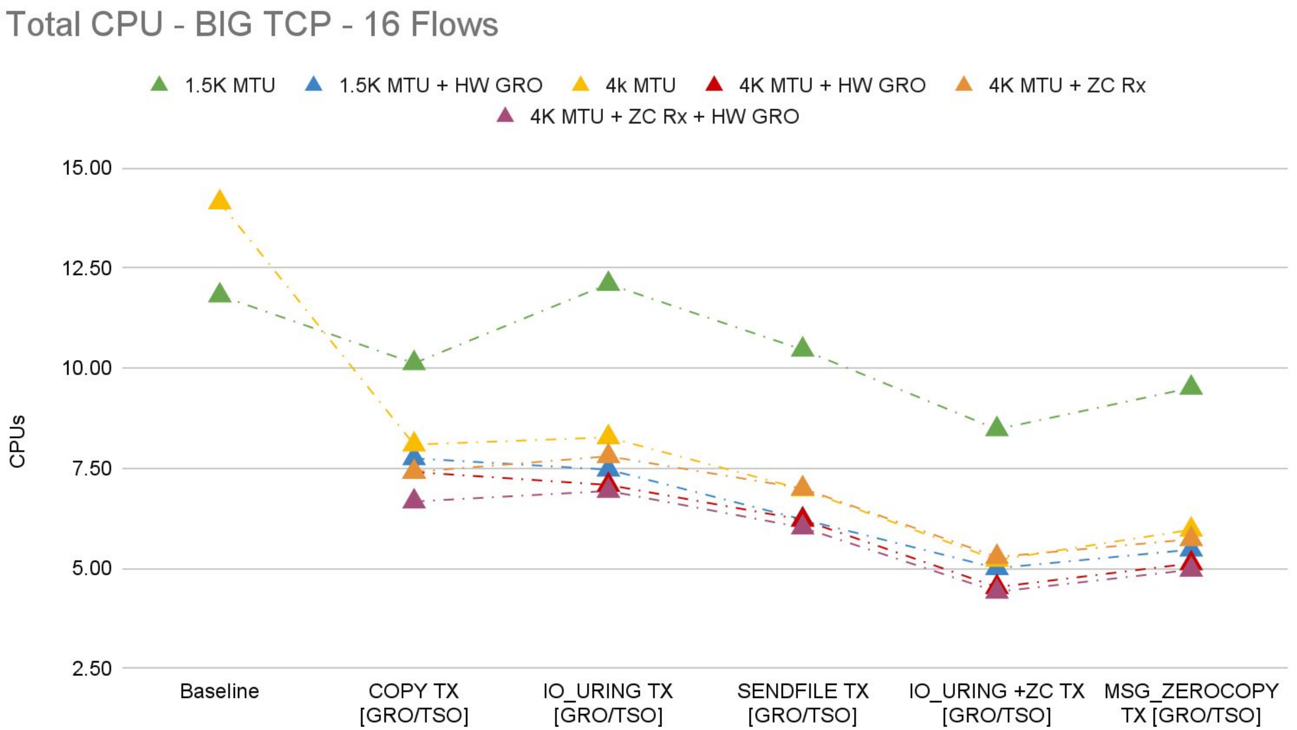


CPU Usage + BIGTCP

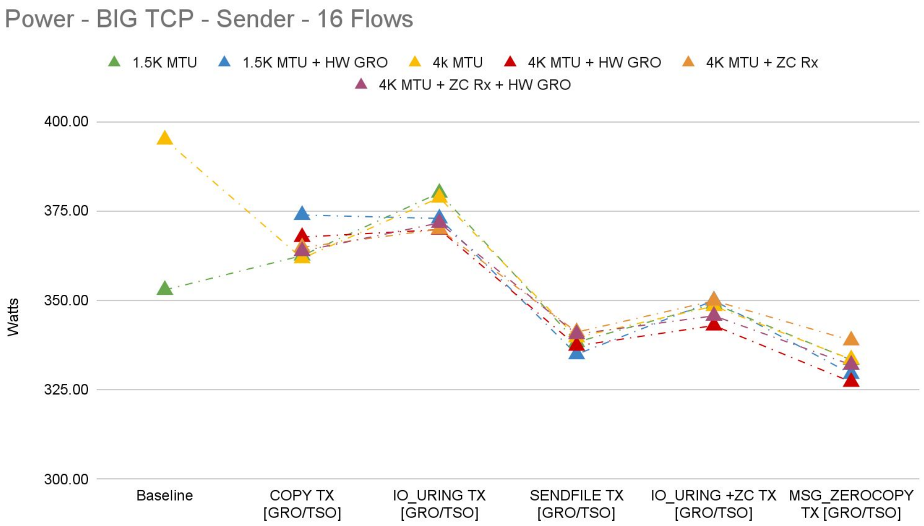


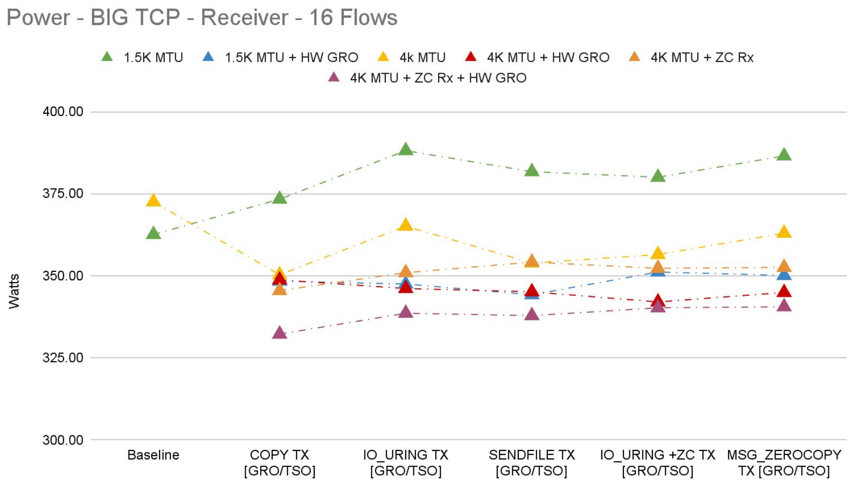


Total CPU Usage +BIGTCP

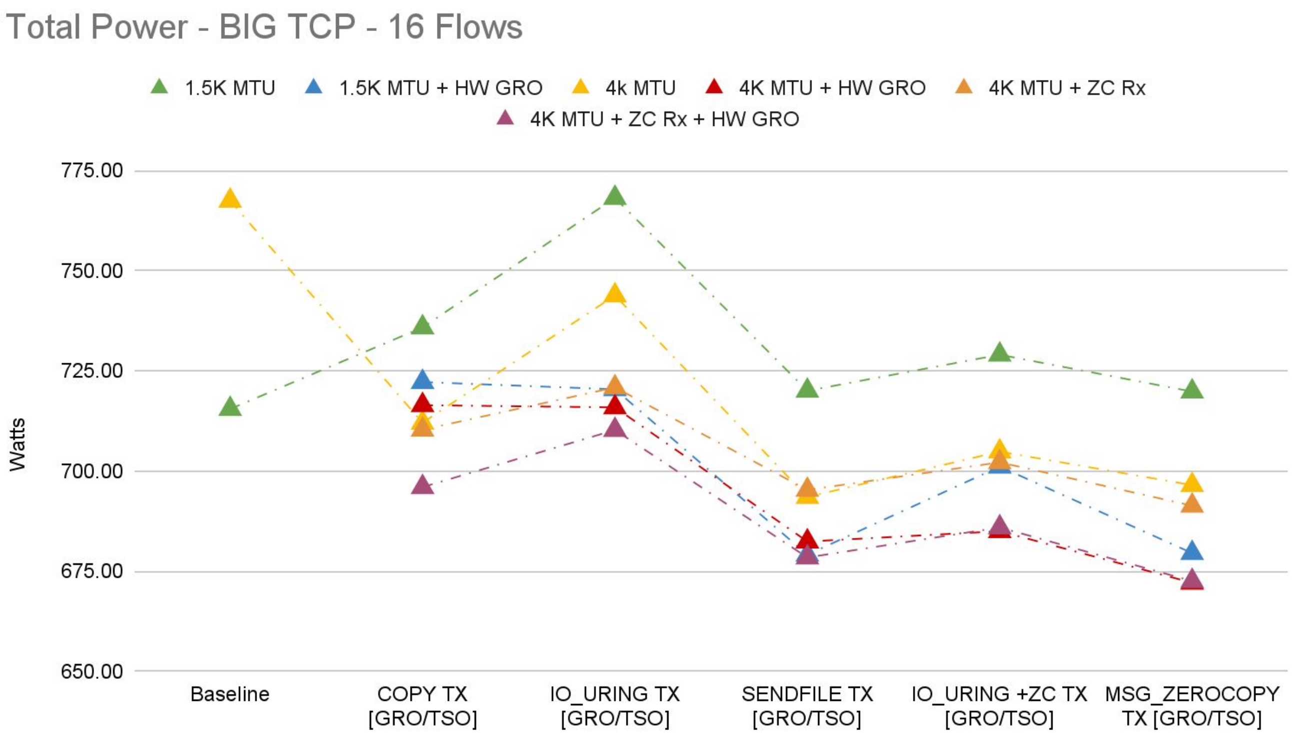


Power Usage +BIGTCP

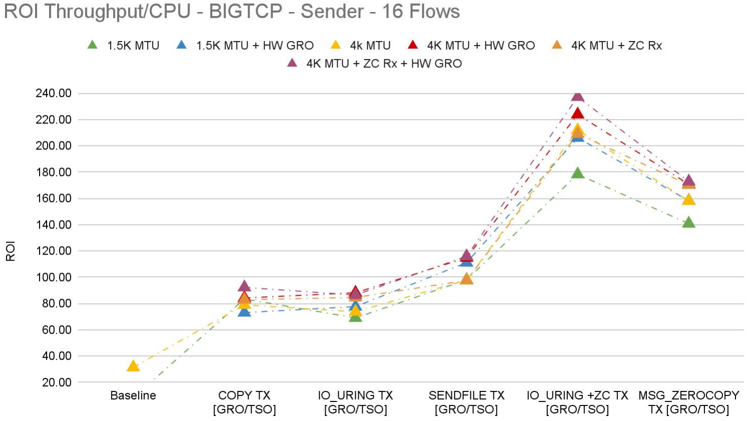


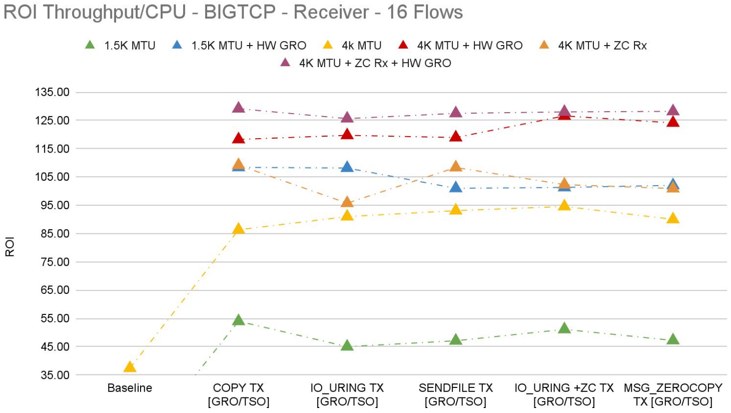


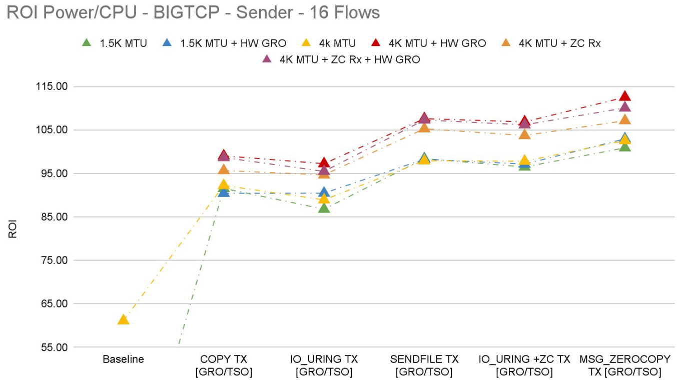
Total Power Usage +BIGTCP

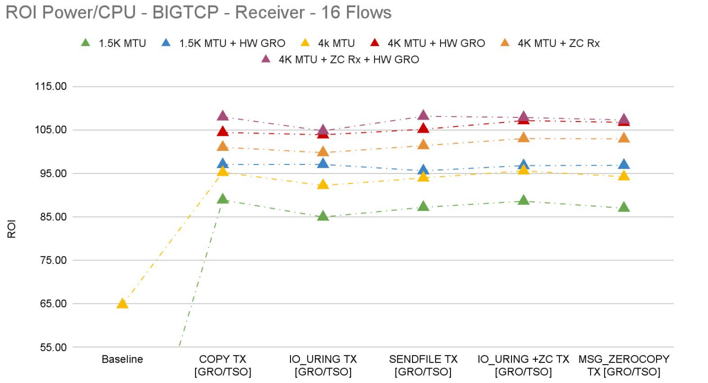


ROIs +BIGTCP


ROI Throughput/CPU - BIGTCP - Sender - 16 Flows
▲ 1.5K MTU
▲ 1.5K MTU + HW GRO
▲ 4k MTU
▲ 4K MTU + HW GRO
▲ 4K MTU + ZC Rx
▲ 4K MTU + ZC Rx + HW GRO
ROI
240.00
220.00
200.00
180.00
160.00
140.00
120.00
100.00
80.00
60.00
40.00
20.00
Baseline
COPY TX [GRO/TSO]
IO_URING TX [GRO/TSO]
SENDFILE TX [GRO/TSO]
IO_URING +ZC TX [GRO/TSO]
MSG_ZEROCOPY TX [GRO/TSO]


ROI Throughput/CPU - BIGTCP - Receiver - 16 Flows
▲ 1.5K MTU
▲ 1.5K MTU + HW GRO
▲ 4k MTU
▲ 4K MTU + HW GRO
▲ 4K MTU + ZC Rx
▲ 4K MTU + ZC Rx + HW GRO
ROI
135.00
125.00
115.00
105.00
95.00
85.00
75.00
65.00
55.00
45.00
35.00
Baseline
COPY TX [GRO/TSO]
IO_URING TX [GRO/TSO]
SENDFILE TX [GRO/TSO]
IO_URING +ZC TX [GRO/TSO]
MSG_ZEROCOPY TX [GRO/TSO]

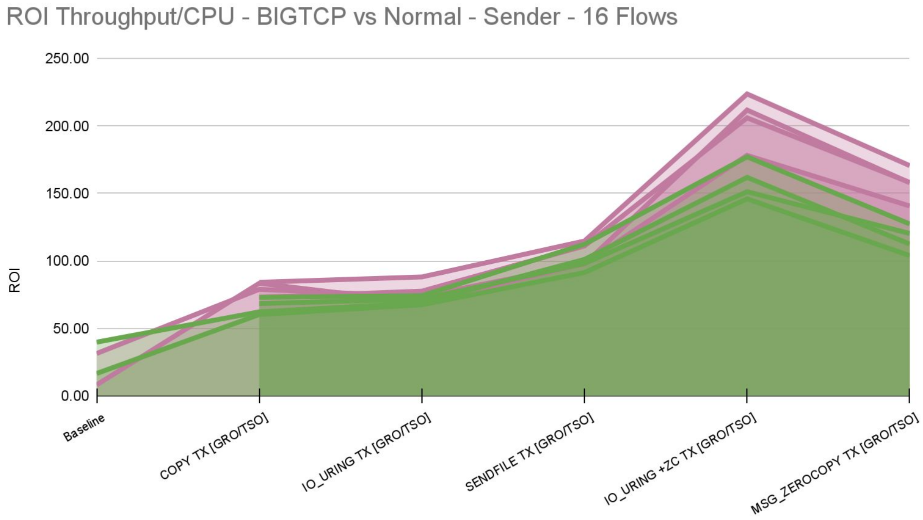

ROI Power/CPU - BIGTCP - Sender - 16 Flows
▲ 1.5K MTU
▲ 1.5K MTU + HW GRO
▲ 4k MTU
▲ 4K MTU + HW GRO
▲ 4K MTU + ZC Rx
▲ 4K MTU + ZC Rx + HW GRO
ROI
115.00
105.00
95.00
85.00
75.00
65.00
55.00
Baseline
COPY TX [GRO/TSO]
IO_URING TX [GRO/TSO]
SENDFILE TX [GRO/TSO]
IO_URING +ZC TX [GRO/TSO]
MSG_ZEROCOPY TX [GRO/TSO]


ROI Power/CPU - BIGTCP - Receiver - 16 Flows
▲ 1.5K MTU
▲ 1.5K MTU + HW GRO
▲ 4k MTU
▲ 4K MTU + HW GRO
▲ 4K MTU + ZC Rx
▲ 4K MTU + ZC Rx + HW GRO
ROI
115.00
105.00
95.00
85.00
75.00
65.00
55.00
Baseline
COPY TX [GRO/TSO]
IO_URING TX [GRO/TSO]
SENDFILE TX [GRO/TSO]
IO_URING +ZC TX [GRO/TSO]
MSG_ZEROCOPY TX [GRO/TSO]

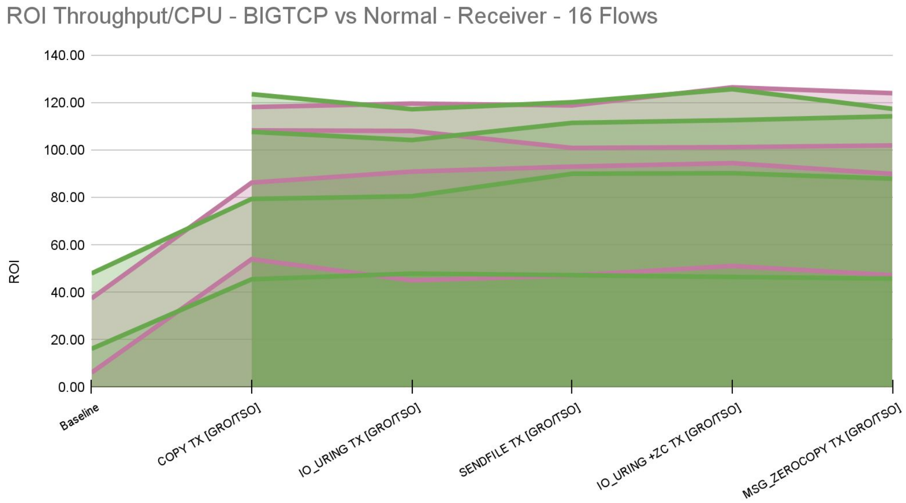
ROIs Diff

🟪 BIG TCP

🟩 NORMAL

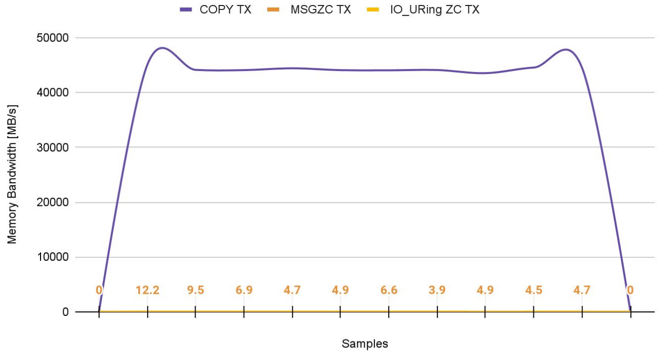


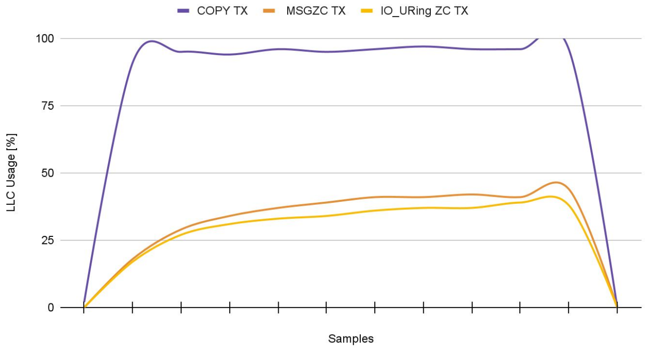
ROI Throughput/CPU - BIGTCP vs Normal - Sender - 16 Flows

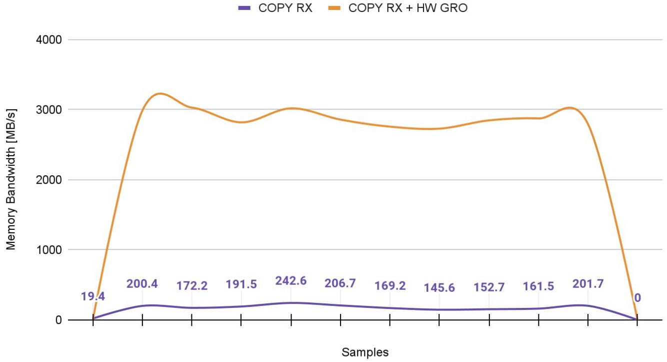


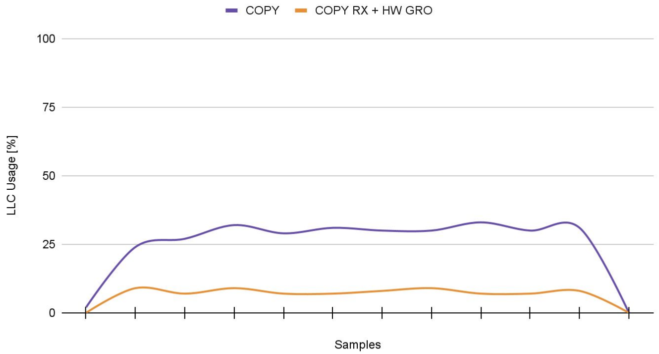
ROI Throughput/CPU - BIGTCP vs Normal - Receiver - 16 Flows

1.5K MTU - Effect on Memory

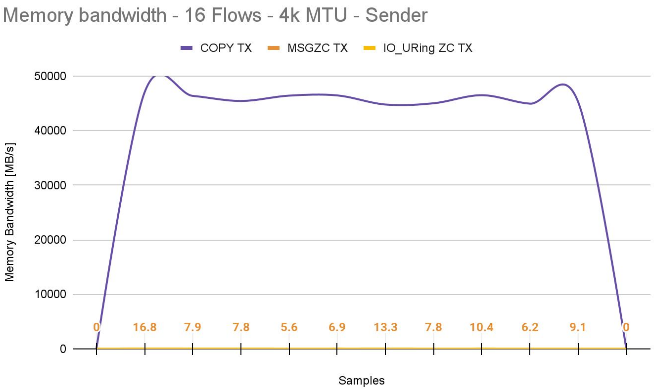

Memory bandwidth - 16 Flows - 1.5k MTU - Sender

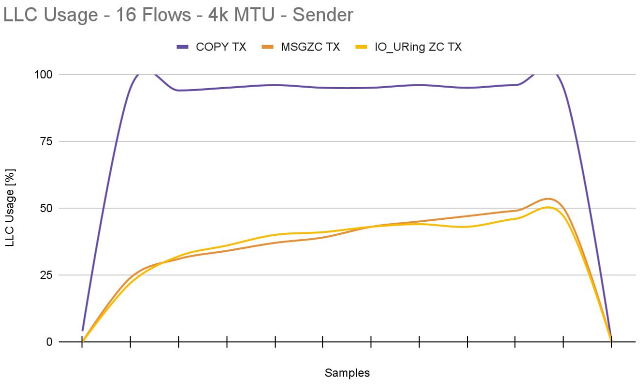

LLC Usage - 16 Flows - 1.5k MTU - Sender

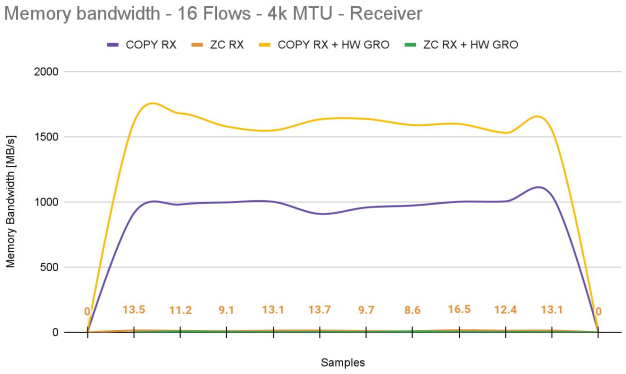

Memory bandwidth - 16 Flows - 1.5k MTU - Receiver

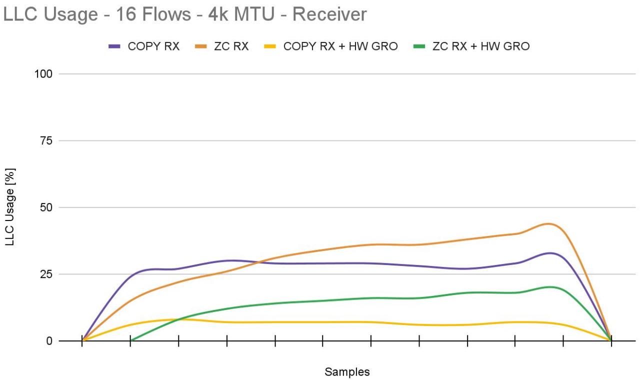

LLC Usage - 16 Flows - 1.5k MTU - Receiver

4K MTU - Effect on Memory


Memory bandwidth - 16 Flows - 4k MTU - Sender

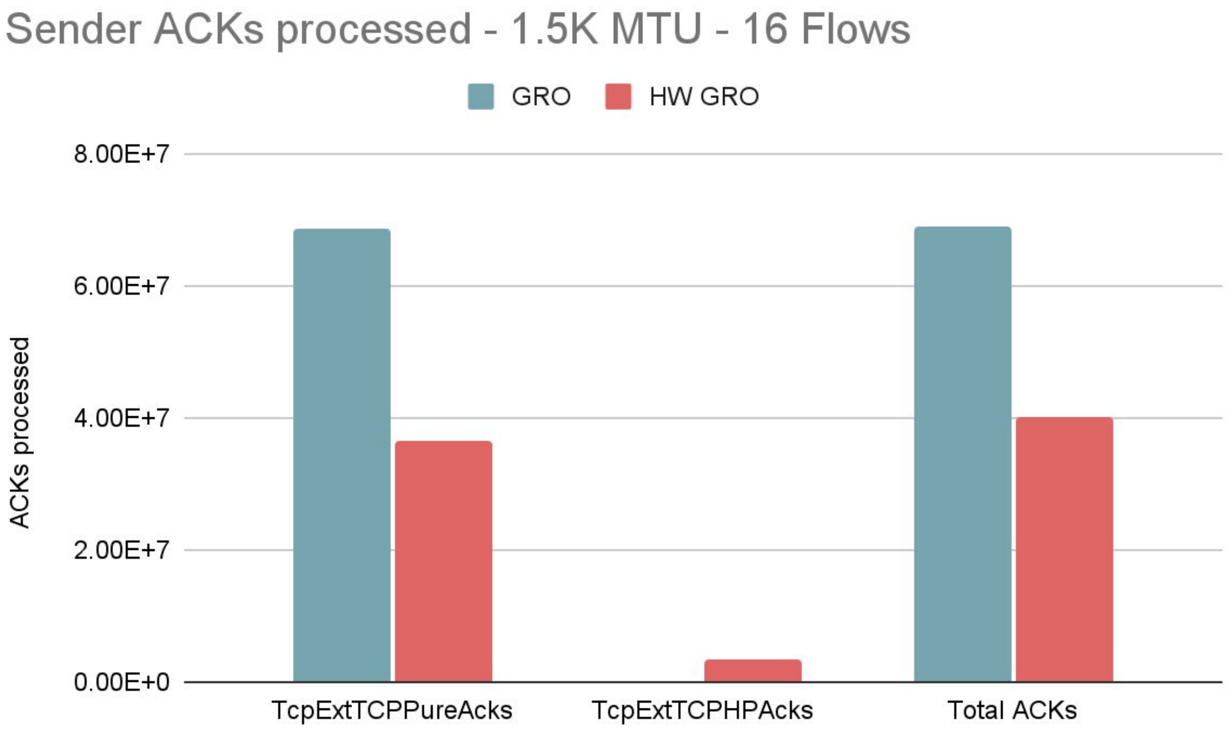

LLC Usage - 16 Flows - 4k MTU - Sender


Memory bandwidth - 16 Flows - 4k MTU - Receiver


LLC Usage - 16 Flows - 4k MTU - Receiver

Quarks And Challenges

Mystery: HW GRO Test From Sender perspective



IO_URing

- In our tests with IO_URing we saw suspicious results and interactions so we decided to contact the authors
    - Pavel and David were quick to respond and help us understand the issues
- The authors cooked two patches
    - One removes iowait accounting which interacts with sleep states
        - Reason for increased power glitch
        - Still incoming to mainline
    - The other improves the ZC notification handling for small messages
        - Merged
            - https://lore.kernel.org/io-uring/171383162913.28674.10529865050261202154.git-patchwork-notify@kernel.org/T/#t

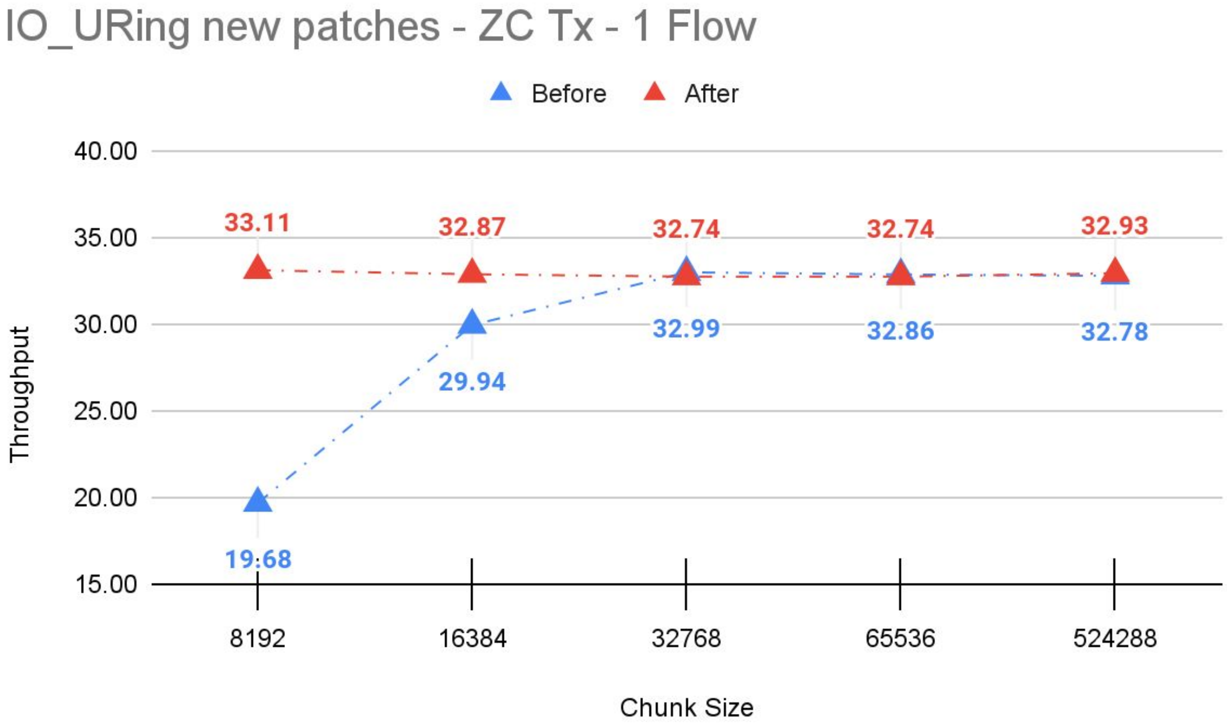
IO_URing - Removing iowait accounting

CPU	POLL	C1	C1E	C6
0	**0**	0.01	57.06	26.14
1	**13.09**	6.68	41.16	19.2
2	**11.47**	5.41	43.19	45.38
3	**1.01**	1.23	58.13	20.56
4	**7.07**	4.23	40.04	37.66
5	**0**	0.01	51.67	30.73
6	**20.91**	9.43	27.83	13.34
7	**12.81**	3.16	40.48	42.64
8	**0.02**	0	38.25	73.54
9	**21.96**	10.02	23.68	46.14
10	**15.1**	8.6	33.94	45.59
11	**0**	0.01	33.22	47.47
12	**11.98**	5.41	33.17	30.66
13	**22.29**	9.16	23.86	44.8
14	**14.34**	6.04	35.44	44.07
15	**11.31**	5.67	32.77	57.62

Improved idle states
Reduced power consumption

CPU	POLL	C1	C1E	C6
0	**0**	0	0.58	67.21
1	**0.47**	1.56	49.8	27.06
2	**0.98**	1.4	42.03	29.86
3	**1.56**	1.66	49.61	22.85
4	**0.82**	1.61	30.51	33.81
5	**0.02**	0.04	19.82	63.87
6	**0**	0	0.67	99.29
7	**0**	0	0.6	82.31
8	**0.99**	1.61	50.61	36.51
9	**0**	0	0.32	83.25
10	**0.06**	0.11	25.89	52.4
11	**0.47**	1.59	40.61	53.88
12	**0.9**	1.72	46.9	44.29
13	**1.11**	1.16	42.77	42.01
14	**0.79**	1.67	42.32	52.22
15	**0**	0	0.58	89.64

IO_URing - Improved notification handling



Takeaways And Future Work

Takeaways

- Bigger MTUs correlates to better metrics
- HW GRO boosts 1.5K and 4K MTUs considerably
  - 1.5K MTU + HW GRO gets close to plain 4K MTU
  - 4K MTU + HW GRO improves things even further
  - Note: The sending side gets a small boost
- IO_URing ZC notification processing shaves a lot of CPU cycles when compared to MSG_ZEROCOPY
  - Yet it loses on the Power ROI
  - Looking forward to test both the MSG_ZEROCOPY and the IO_Uring notification patches once merged
- BIGTCP benefits mostly in the presence of the Sender ZEROCOPY routines
  - On Receiver it doesn’t seem to make a difference

Future Work

- Test with 8K MSS
- Segments from 17->45 for BigTCP
- Test with single flow
- Test with many flows on same tx/rx ring
- Modification to MSG_ZEROCOPY to reduce notifications overhead
    - https://netdevconf.info/0x18/sessions/talk/a-new-lightweight-zero-copy-notification-mechanism-in-linux.html
- IO_URING
    - RX ZC
    - Other issues mentioned once patches make it
- Busy epoll
    - https://netdevconf.info/0x18/sessions/tutorial/real-world-tips-tricks-and-notes-of-using-epoll-based-busy-polling-to-reduce-latency.html
- IOTLB effect
    - https://netdevconf.info/0x18/sessions/talk/characterizing-iotlb-wall-for-multi-100-gbps-linux-based-networking.html