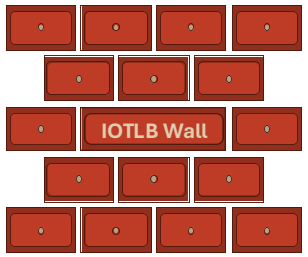
Characterizing IOTLB Wall for Multi-100-Gbps Linux-based Networking

Alireza Farshin (NVIDIA)* and Luigi Rizzo (Google)



* Work was done at KTH Royal Institute of Technology as part of Google PhD Fellowship

Higher Bandwidth Demand

Online Services

AI

VR & AR

VM Hosting

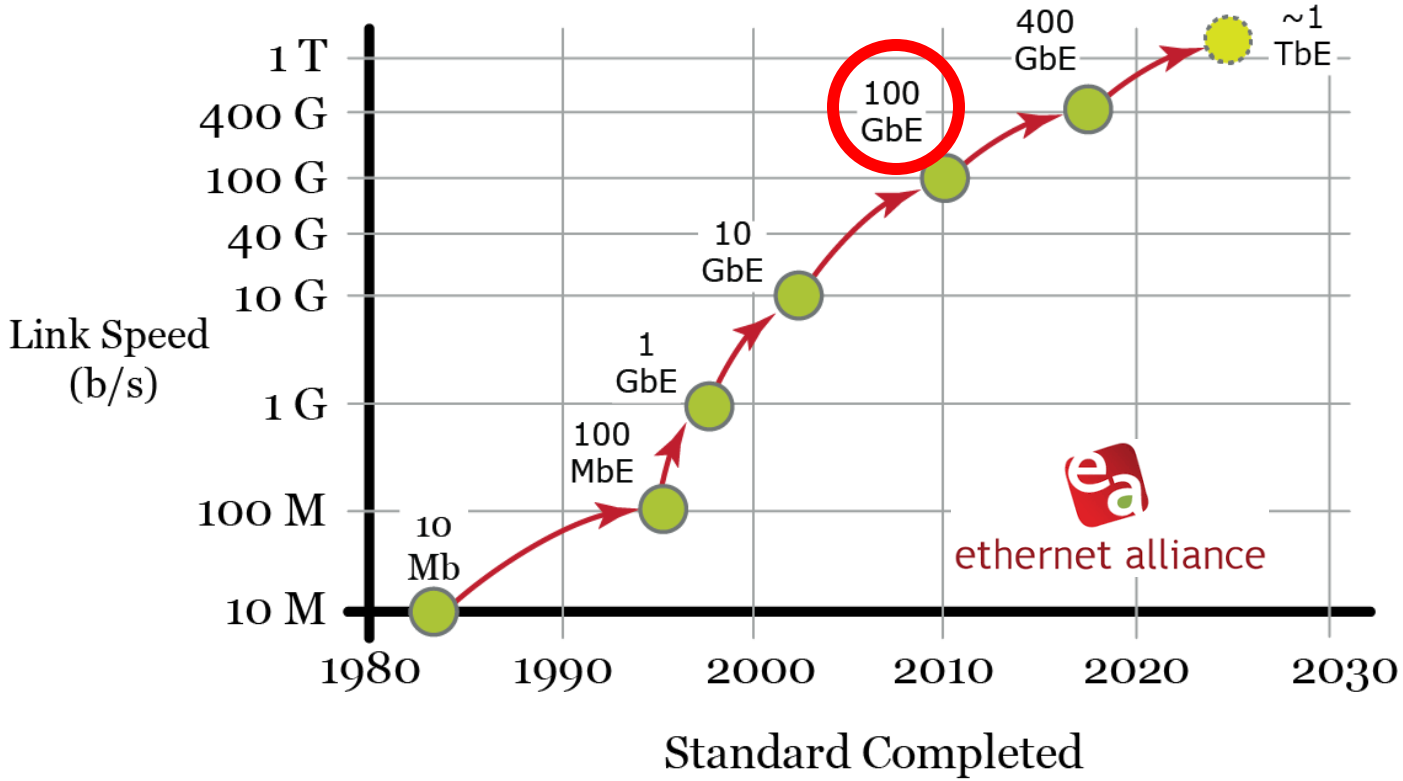
Industrial Digitalization

5G/6G

Link Speeds Move Quickly to 1 Tbps

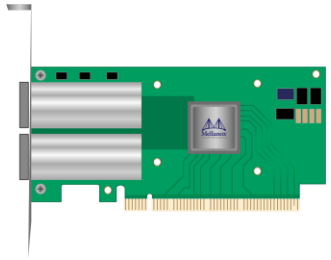
- Inter-arrival time = 10x faster than memory access latency
- Communication between different system components could become a bottleneck

Every 6.72 ns a new (64-B+20-B*) packet arrives at 100 Gbps



* 7B preamble + 1B start-of-frame delimiter + 12B inter-frame gap = 20 B

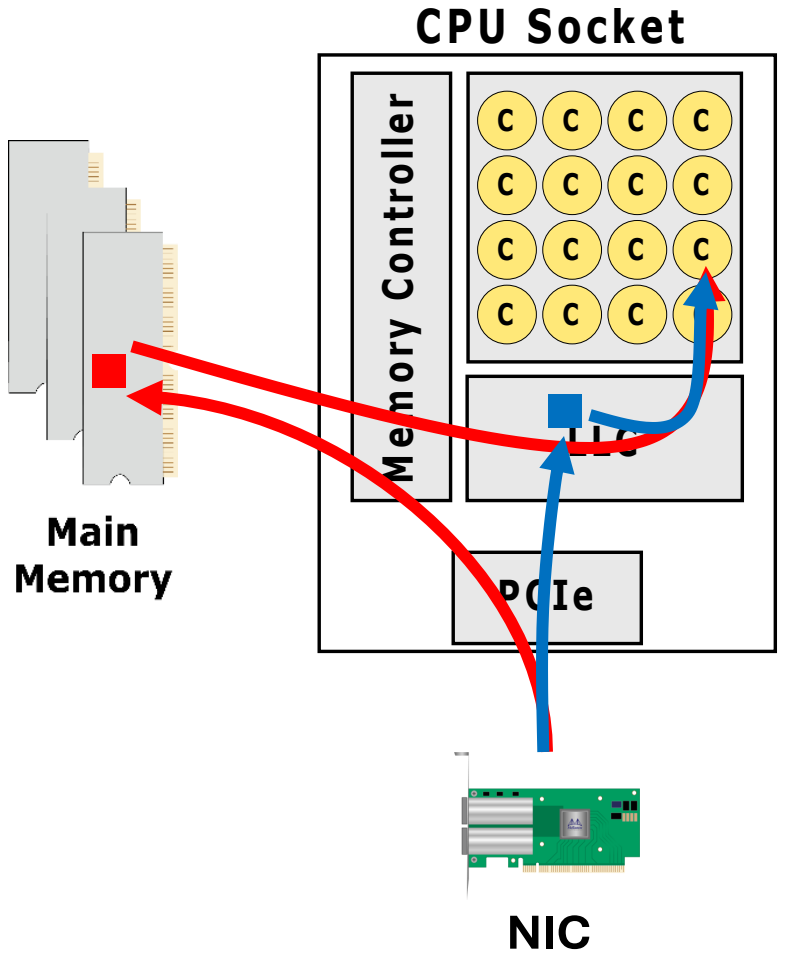
What are the Communication Bottlenecks when Transferring Packets between NIC and CPU



NIC **CPU**

Data Path between NIC and CPU

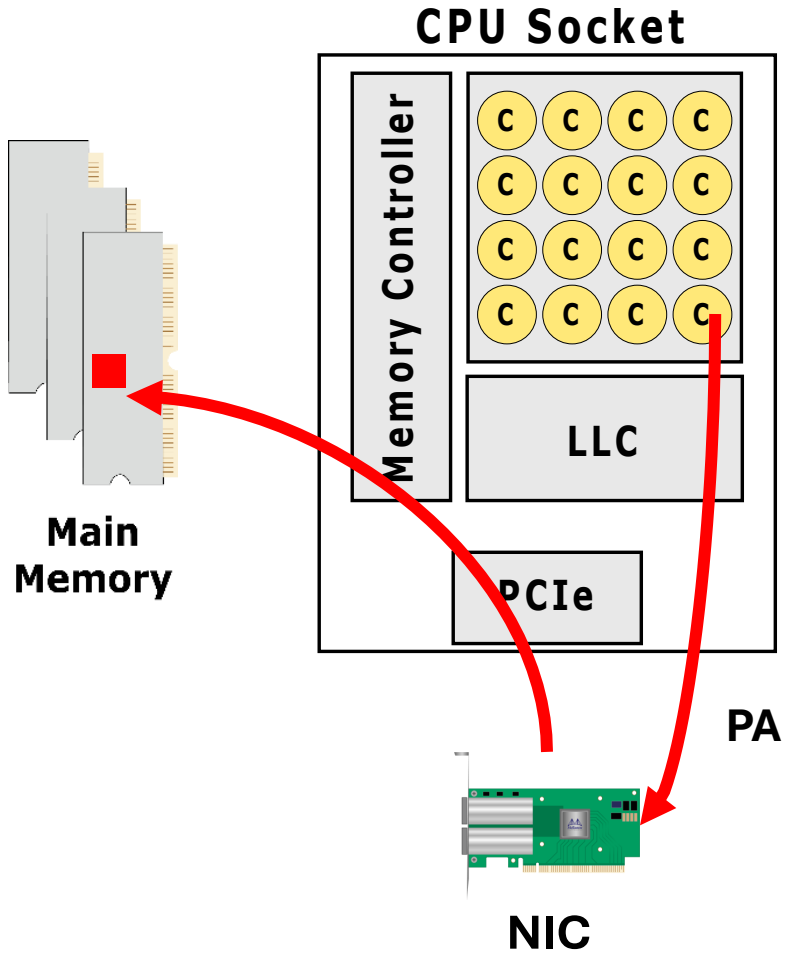
- NIC access memory through the PCIe bus
- On some architectures, NIC can access Last Level Cache (LLC) to reduce latency
  - DDIO* on Intel Xeon processors
- CPU later access the data



* Data Direct I/O Technology

IOMMU*

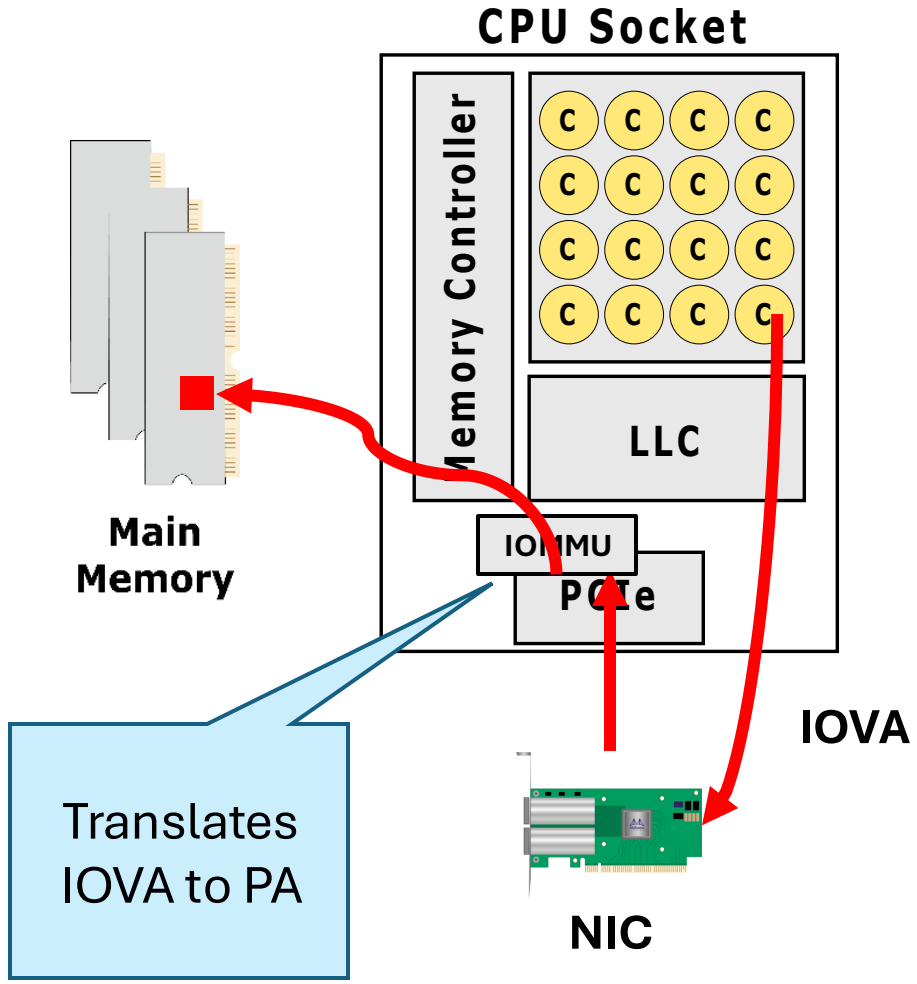
- Without IOMMU, CPU provides the Physical Address (PA) of the buffers to the I/O device



* I/O Memory Management Unit

IOMMU*

- With IOMMU, CPU provides I/O Virtual Address (IOVA) of the buffer to the I/O device
  - Restrict DMAs to specific regions
  - Provide I/O security
  - Facilitate virtualization and backward compatibility
- IOMMU translates IOVA to PA on every I/O request
  - Like MMU, a cache, called IOTLB**, is used to accelerate translations

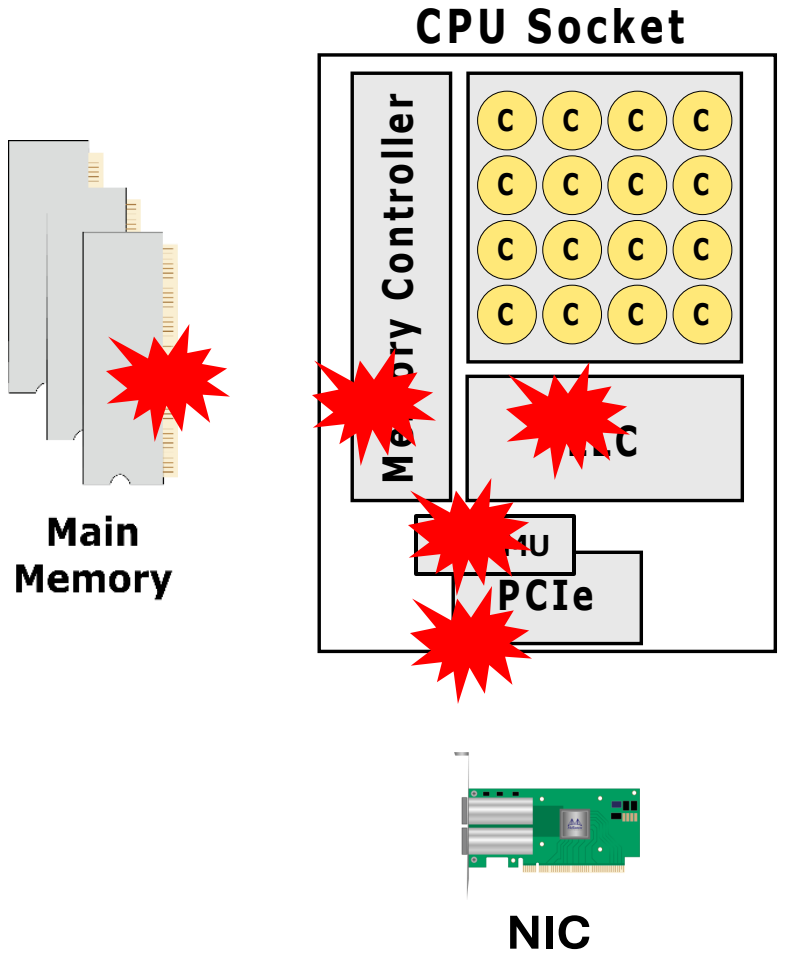


* I/O Memory Management Unit

** I/O Translation Lookaside Buffer

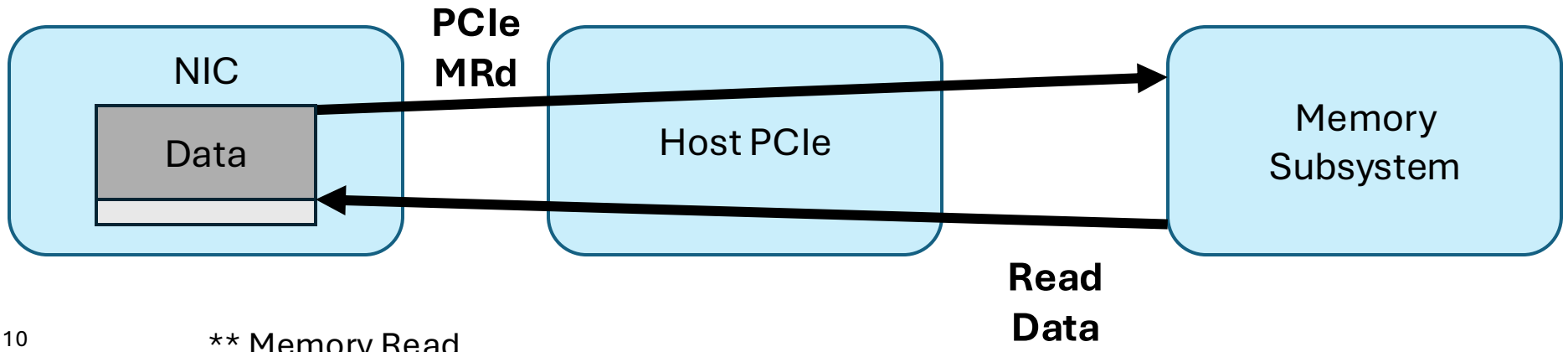
Bottlenecks – Bandwidth

- All components in the path must sustain line rate
- Short-term variability can be absorbed by on-NIC buffers
- PCIe bus
  - PCIe 4.0 supports 16 Gbps per lane
  - 16 lane → 256 Gbps
  - PCIe 5.0 doubles the rate
- Memory interconnect or DDIO
- Memory
- IOMMU/IOTLB



Bottlenecks – Bandwidth Delay Product

- Each boundary has limited buffering
  - Each transaction keeps buffer busy for some time (T) until it is completed
  - Throughput ≪ buffer size / T
- PCIe can buffer around 32-64 KiB* per direction
  - For PCIe MRd**, the read buffer (completion buffer) stays busy for the time between a request is sent and the data is received
    - 64 KiB worth of completion buffers allow only 2.6 µs read latency at 200 Gbps



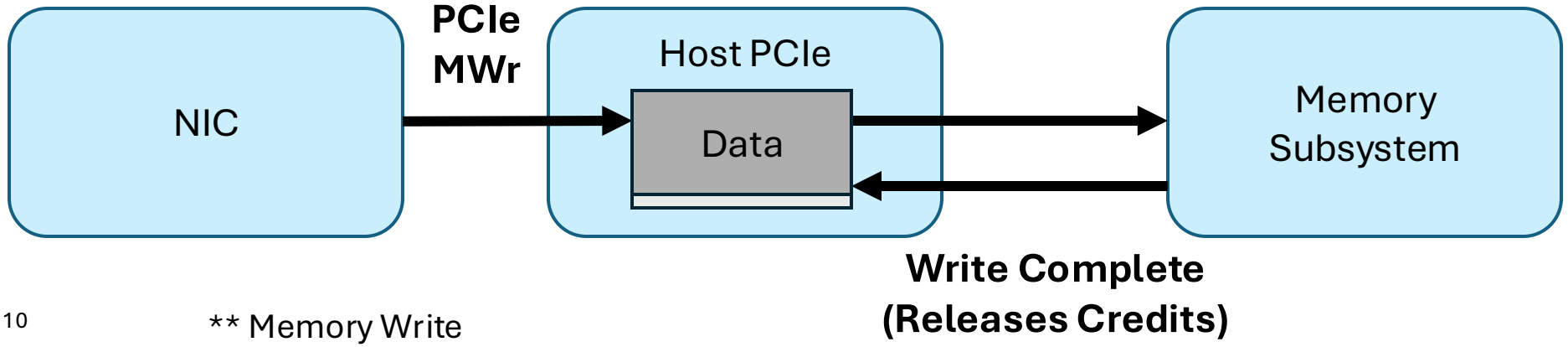
* Ki (kibi) = 2^{10}

** Memory Read

Bottlenecks – Bandwidth Delay Product

- Each boundary has limited buffering
  - Each transaction keeps buffer busy for some time (T) until it is completed
  - Throughput ≪ buffer size / T

In this talk, we focus on IOMMU/IOTLB

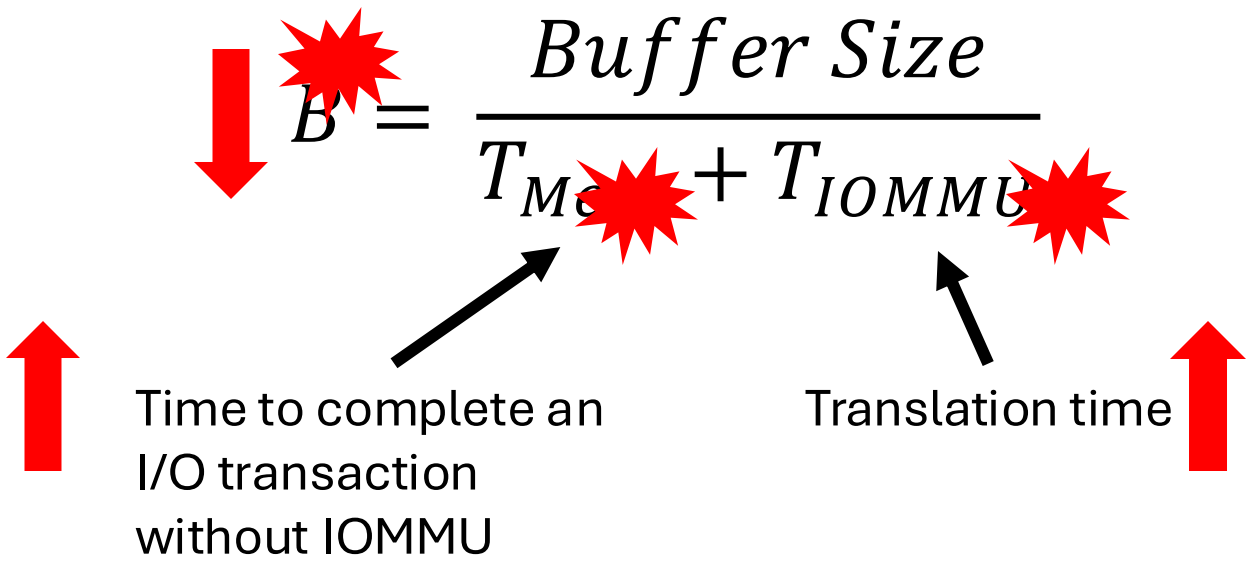


* Ki (kibi) = 2^{10}

** Memory Write

Modeling Performance Bottlenecks

- I/O subsystem has an upper bound for the number of outstanding DMA transactions
- This implicitly creates a throughput bottleneck (B)
  - Ideally, B >> Data path rate (i.e., NIC and PCIe)



IOMMU Performance Overheads

1. IOVA Allocation/Deallocation

2. IOTLB Invalidations/Flushing
   - Ensure higher degree of security

Addressed by previous works; Typically show up as additional CPU load

3. IOTLB Misses
   - Traverse the I/O page table to resolve a miss, causing memory accesses and increases T_{IOMMU}

Our focus

Factors Affecting IOTLB Misses

1. Size and management policy of IOTLB
   - Hardware dependent
   - Different vendors have different implementations (undocumented)

2. Memory request pattern
   - Affected by system configurations (e.g., MTU size, packet rate, drop rate)
   - Offloading features (e.g., LRO, TSO)
   - Buffer management (e.g., Page Pool)
   - Mapping size (4-KiB, 2-MiB, and 1-GiB pages)*

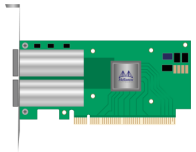
* Ki (kibi) = 2^{10}, Mi (mebi) = 2^{20}, Gi (gibi) = 2^{30}

Impact of IOMMU on Throughput

iPerf Client

iPerf Server

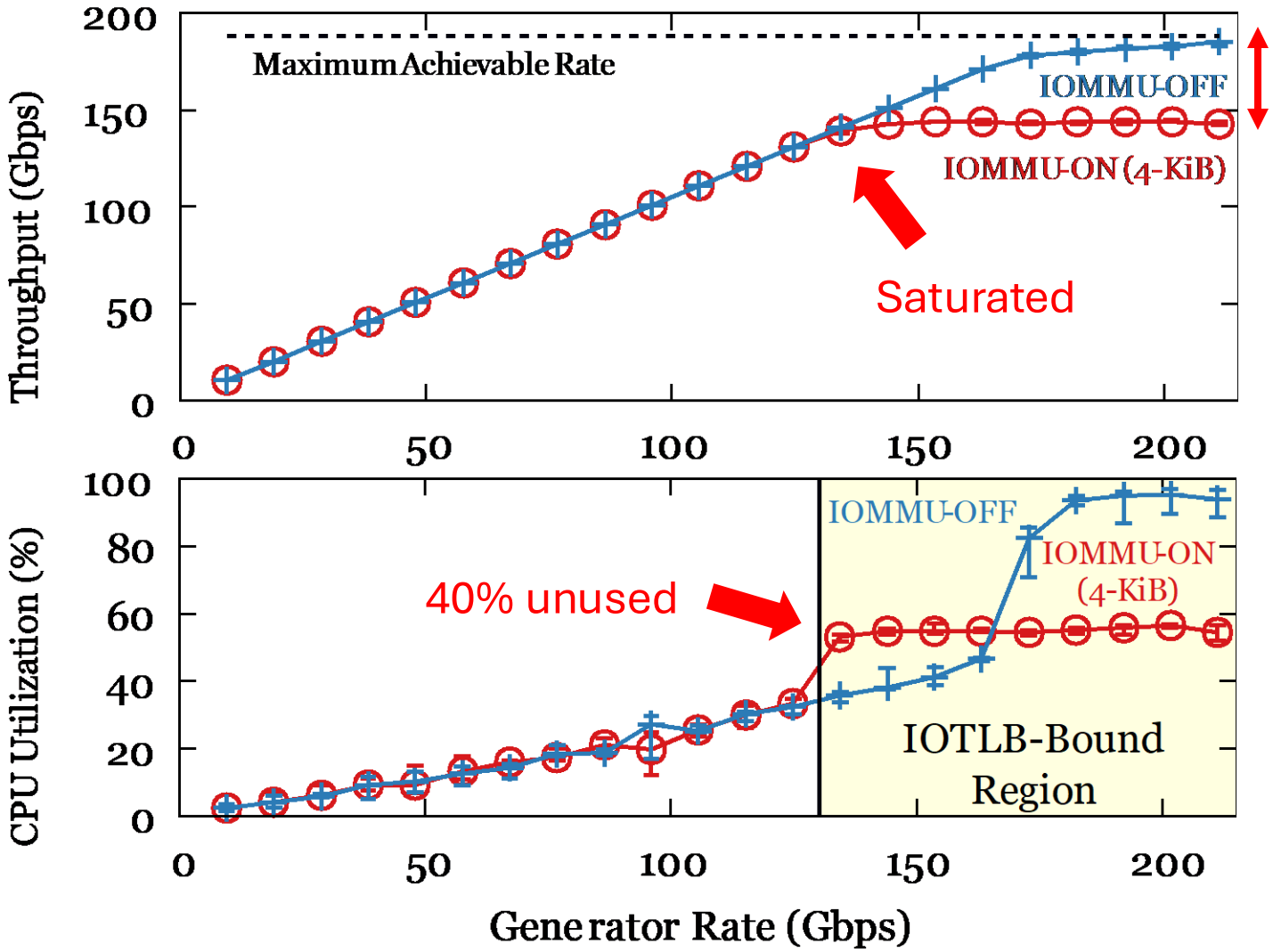
Intel Xeon Gold 6346
(Ice Lake)



NVIDIA/Mellanox
ConnectX-6

* It uses 32 threads, 384 TCP connections, 128 KiB messages, and MTU=1500 (MSS=MTU-52) – Linux kernel v5.15

Impact of IOMMU on Throughput



Enabling IOMMU on an iPerf receiver causes a significant throughput drop

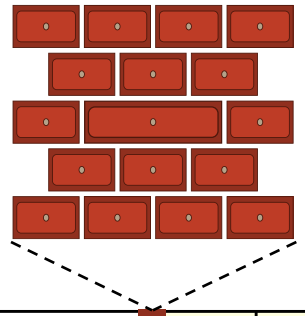
Despite having enough computation power, IOMMU restrains the system from achieving higher throughput

Measuring IOTLB Misses

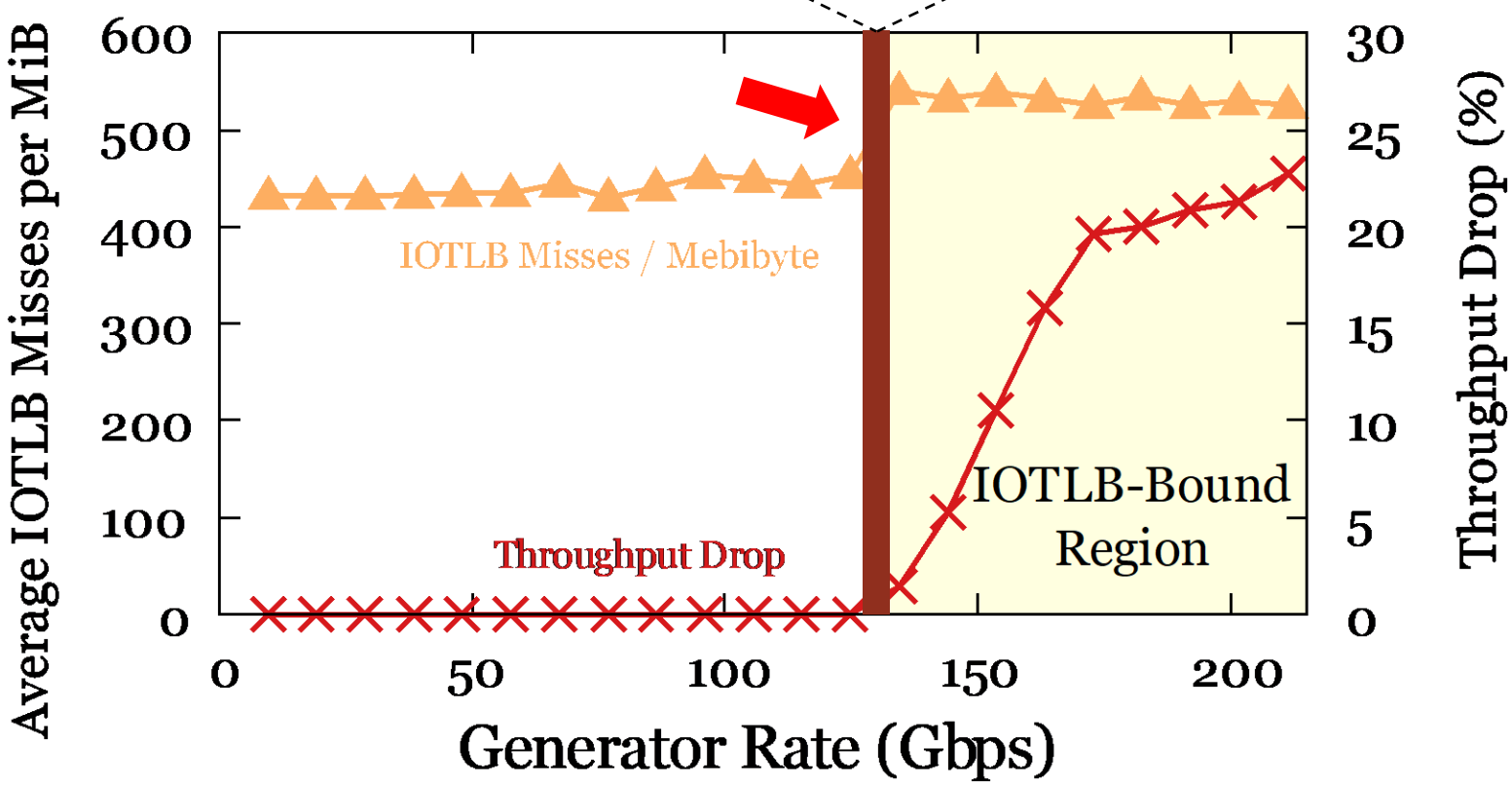
- We report the number of IOTLB misses per unit of data (MiB)
- It makes it possible to compare the IOTLB misses across different rates and configurations

IOTLB Misses Per MiB at 200 Gbps

IOTLB Wall



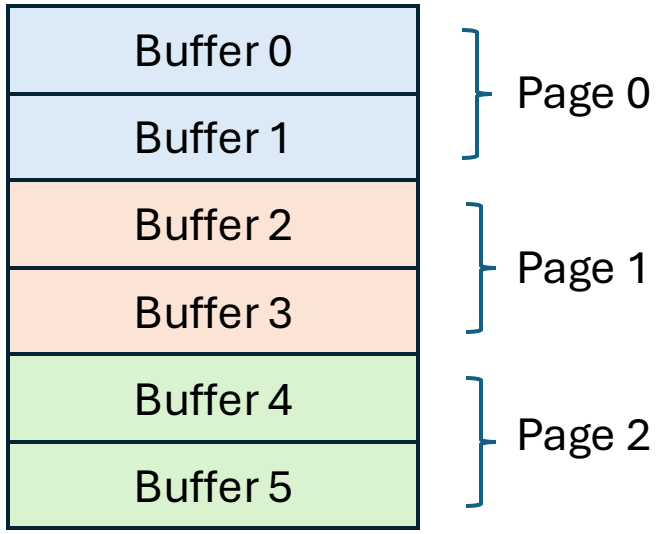
Excessive IOTLB misses builds up a wall, causing throughput drop



We see a sudden jump in the number of IOTLB misses per MiB

IOTLB Wall – 2-KiB Buffers

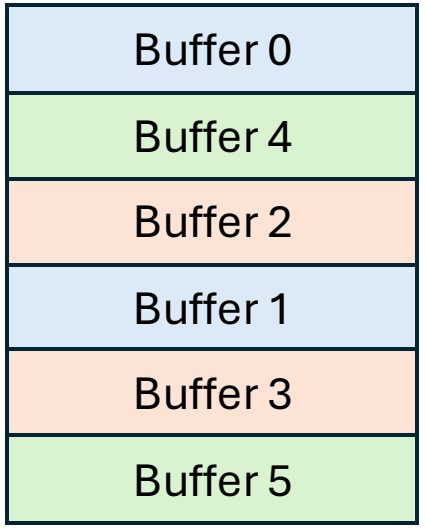
- MTU = 1500, MSS = 1448
- 1 MiB requires 2^{20}/ 1448 ≈ 725 RX buffers
- Each buffer uses half a 4-KiB page → ~362 x 4-KiB pages*
- Initially, the RX descriptors are contiguous
  - Only accessing the first buffer causes an IOTLB miss



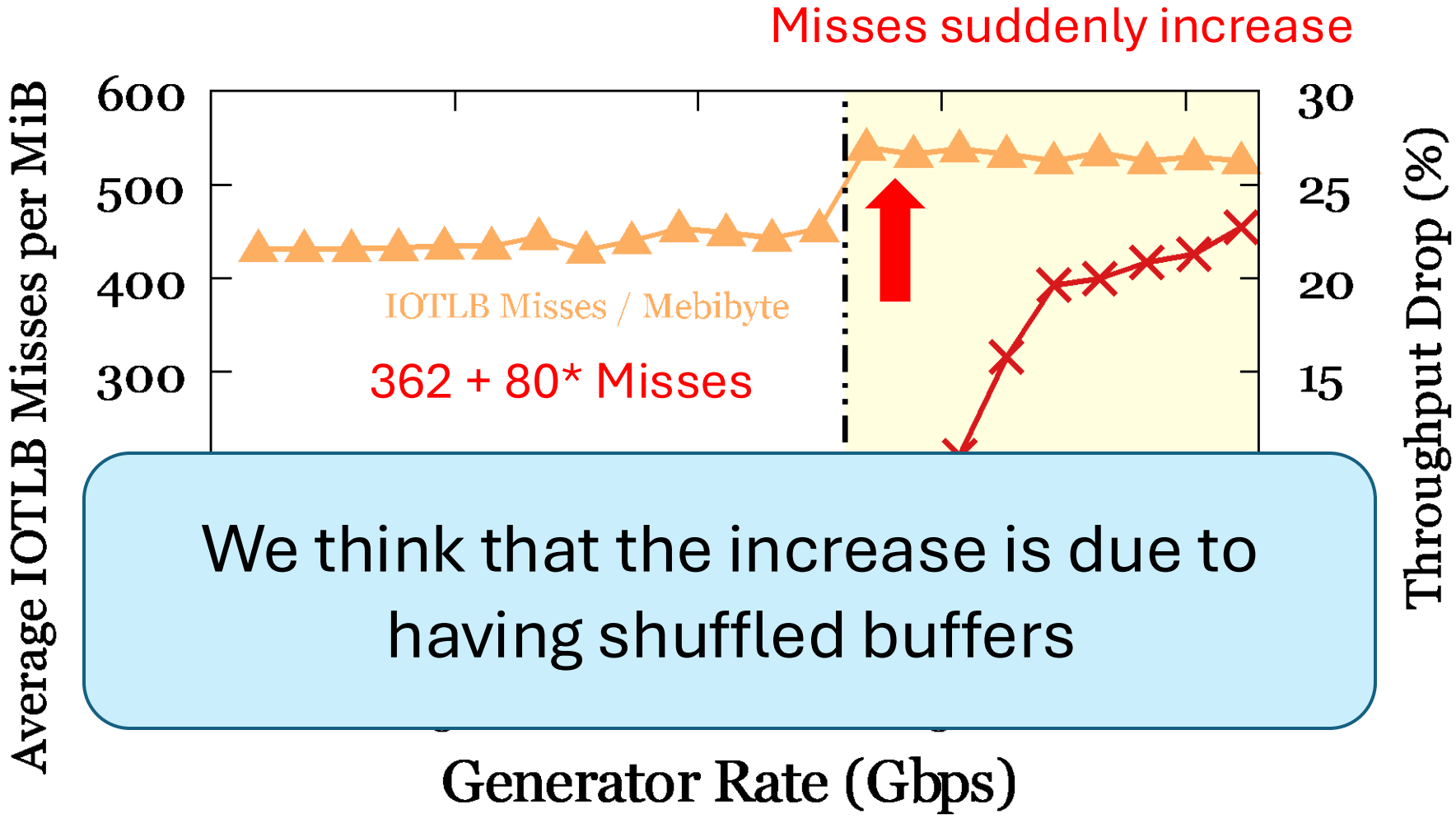
* If the device driver does not allocate any other pages

IOTLB Wall – 2-KiB Buffers

- Later, due to packet drops and slow buffer recycling,
  - Buffers are shuffled, which causes an additional IOTLB miss



IOTLB Wall – 2-KiB Buffers

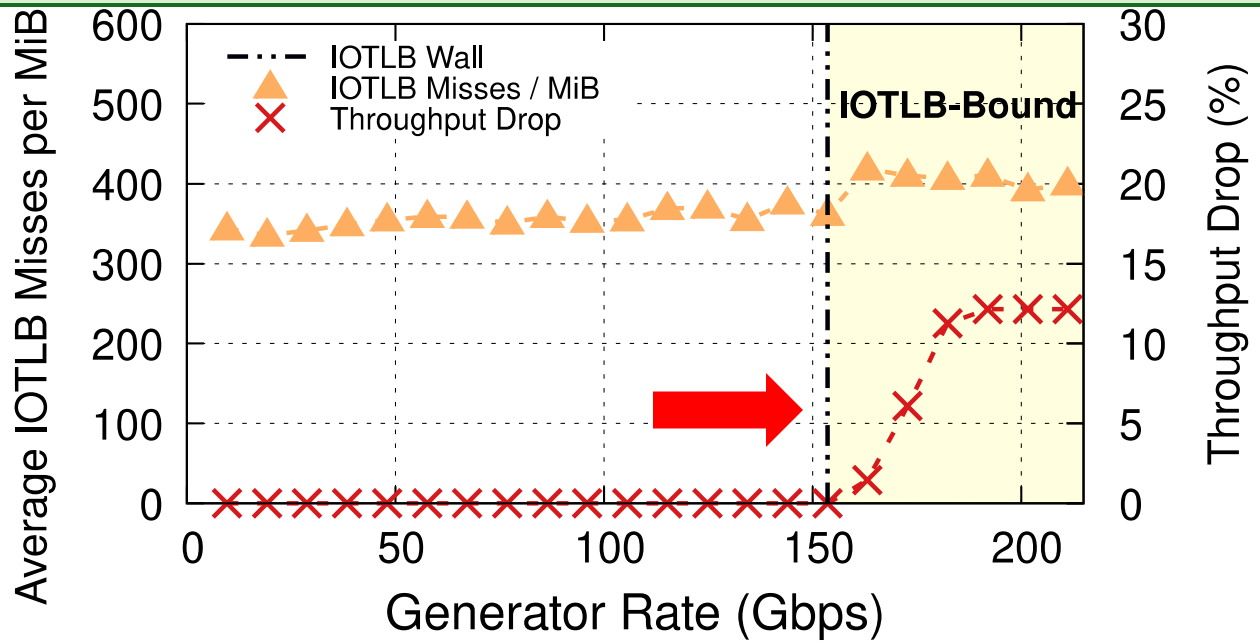


* Due to slow recycling and having multiple queues

IOTLB Wall – 4-KiB Buffers

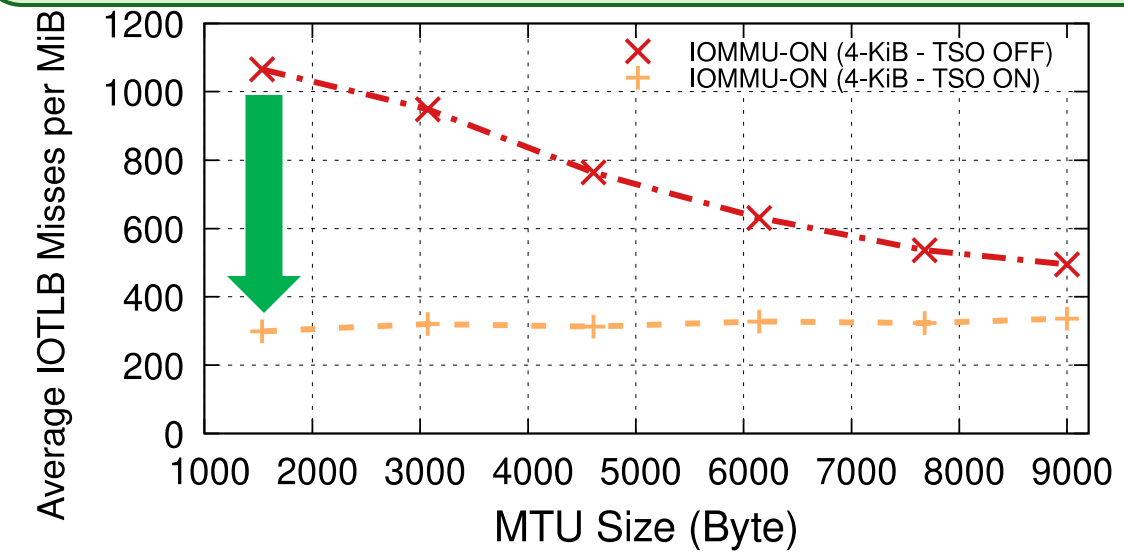
A modest decrease in the number of IOTLB misses per MiB can shift the rate at which throughput becomes a bottleneck

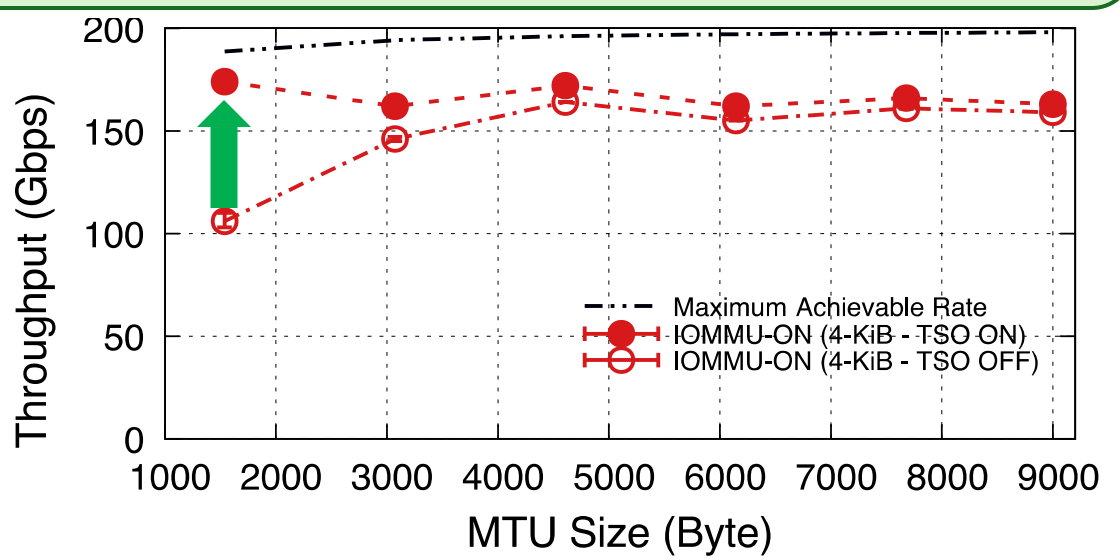
- Fewer IOTLB misses (420 vs. 365)
- IOTLB wall happens at a higher rate (130 vs. 150)



Impact of Offloading Features – TSO*

TSO reduces the number of IOTLB misses per MiB and the throughput drop due to IOTLB wall

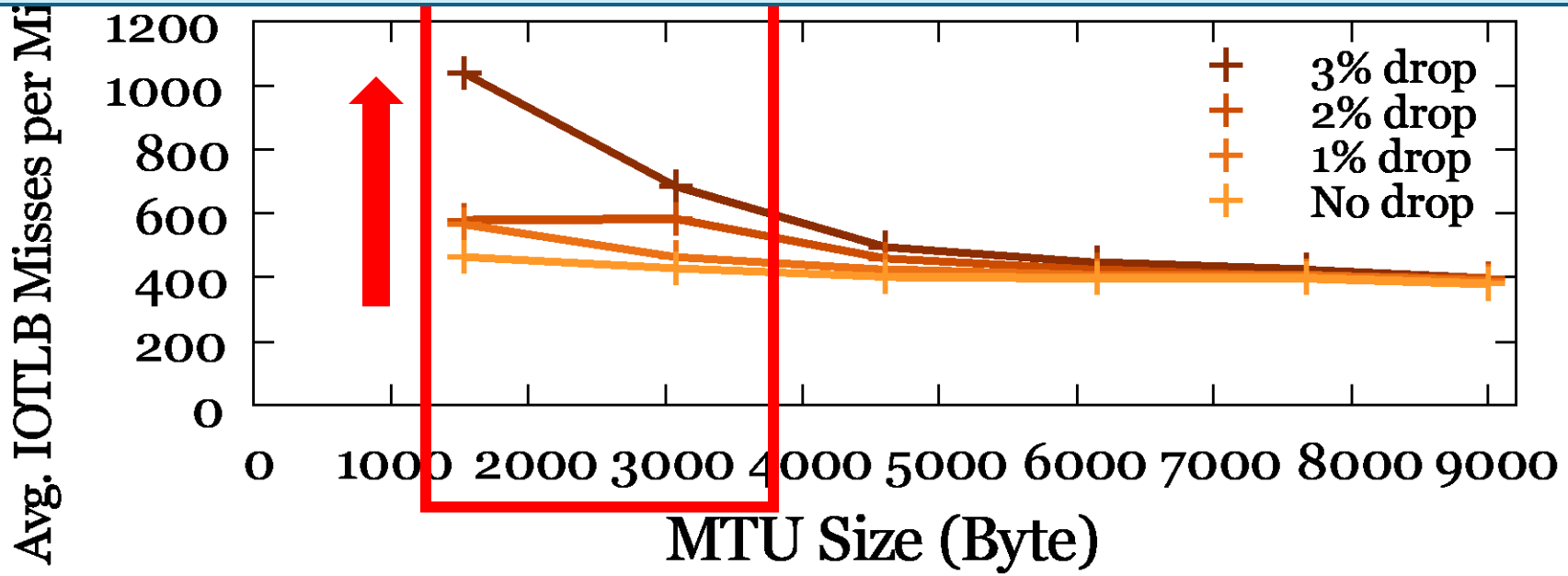




* TCP Segmentation Offload (TSO)

Impact of Packet Drops

Split pages are shuffled by the packet drops and TCP re-transmission, causing more IOTLB misses

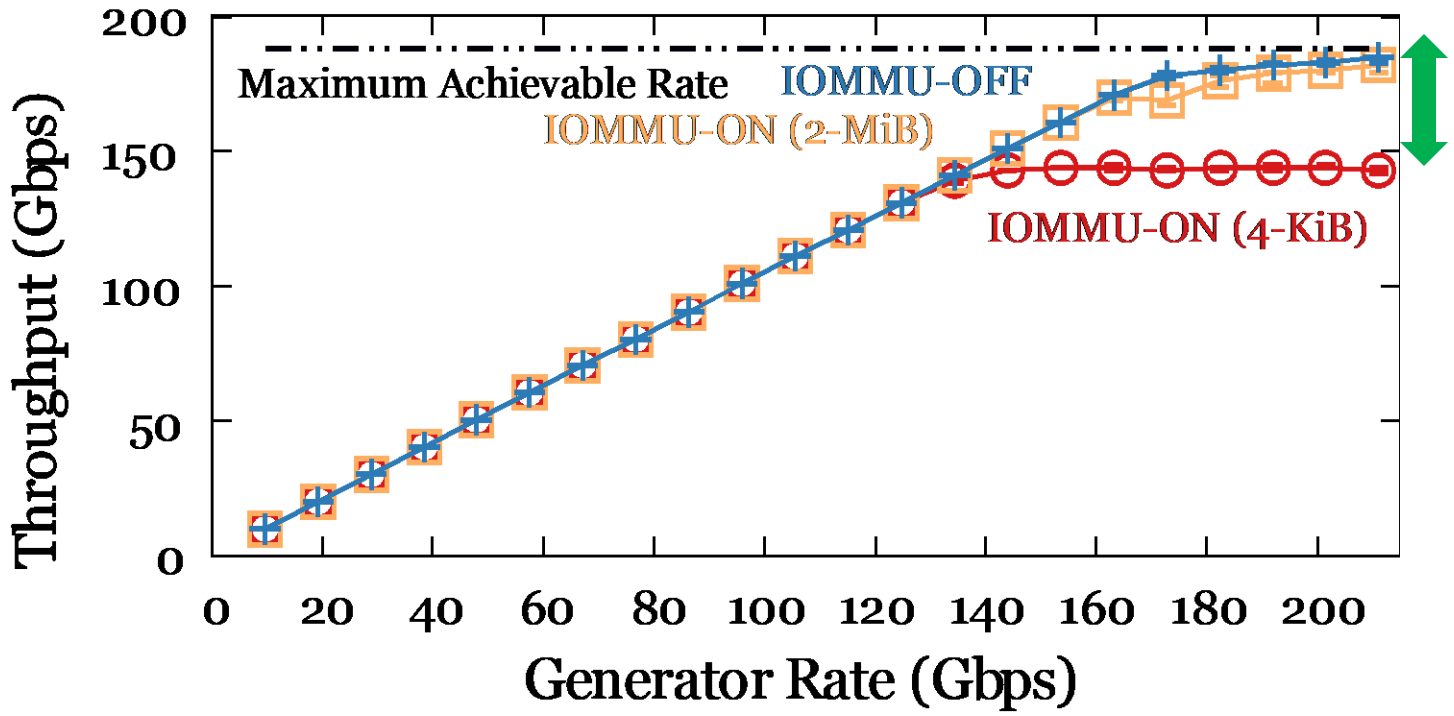


* Luigi patch: https://lists.bufferbloat.net/pipermail/bloat/2021-October/016693.html

How to Mitigate IOTLB Wall?

- Use larger mappings (e.g., 2-MiB and 1-GiB)

  Using 2-MiB huge pages* recovers the throughput drop caused by IOMMU



* We modified Page Pool API to use 2-MiB pages and allocate 512 x 4-KiB pages when performing bulk allocation.

Using Larger Mappings – Challenges (1/3)

- Allocation and CPU cost
  - Allocating 512 x 4-KiB physically contiguous pages are more difficult
  - Compaction (coalescing) 512 pages is significantly more expensive
  - It may cause tremendous memory fragmentation in long-running systems

Using Larger Mappings – Challenges (2/3)

- Memory stranding
  - Possible to reserve a few GB of memory based on BDP* at boot time
  - We noticed existing drivers continually allocate pages due to slow recycling, so it may be difficult to operate with a fixed-size page pool

* Bandwidth Delay Product

Using Larger Mappings – Challenges (3/3)

- Locality and buffer management
  - It is much more difficult to ensure locality with larger mappings as they are split into smaller chunks (e.g., 512 x 4-KiB)
  - More severe buffer shuffling

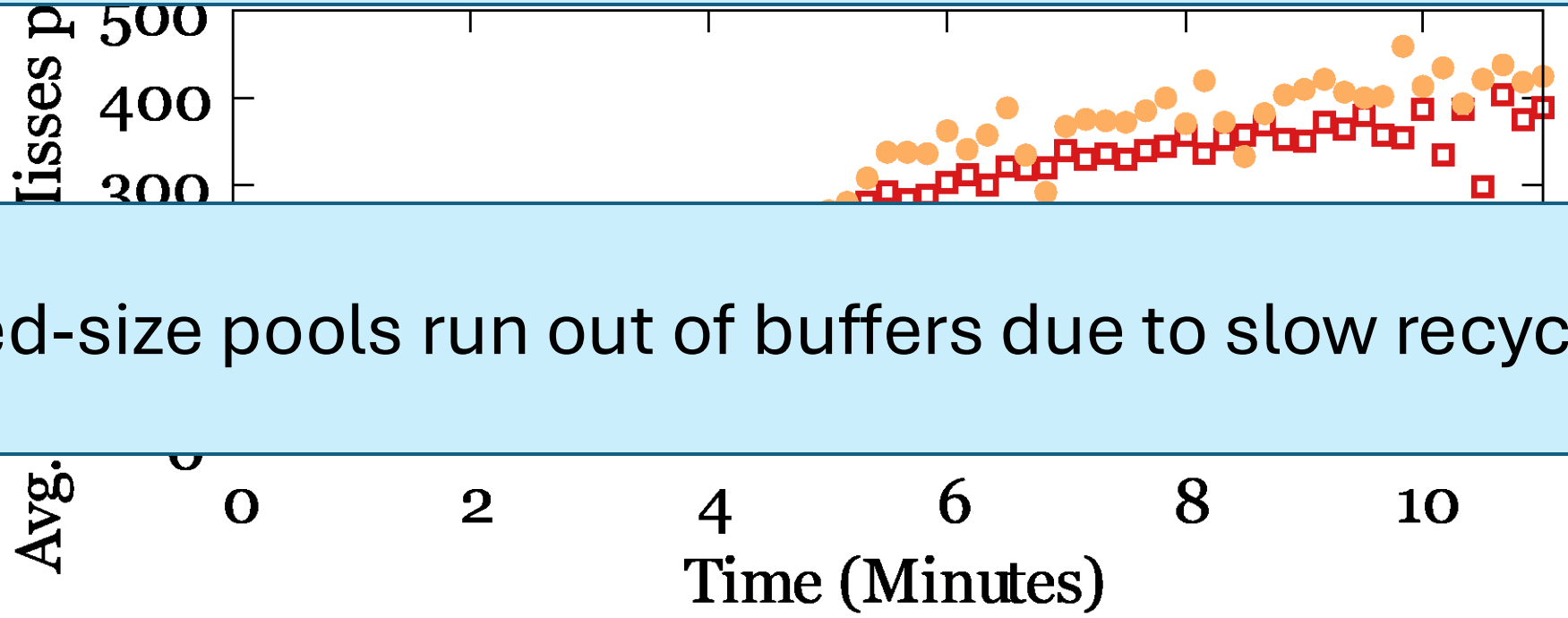
Buffer Shuffling with Larger Mappings

- Continual allocation
  - No buffer recycling

- Using fixed-size (pre-allocated) pool
  - 256 x 2-MiB huge pages
  - 512 x 2-MiB huge pages

Buffer Shuffling with Larger Mappings

IOTLB misses increase significantly over time when using fixed-size pool due to buffer shuffling



Fixed-size pools run out of buffers due to slow recycling

Conclusion

- Shifting toward high link speeds could introduce new bottlenecks in the system
- We modeled these bottlenecks and characterized IOTLB wall at 200 Gbps
- Supporting the upcoming 200/400-Gbps networking with larger IOTLB mappings demands fundamental changes in Linux kernel memory management and I/O management

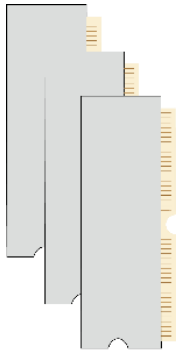
github.com/aliireza/iommu-bench

Backup Slides

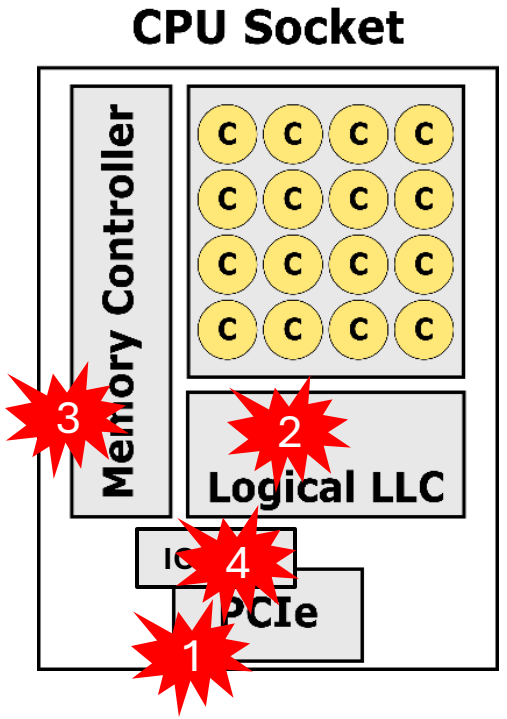
Bottlenecks

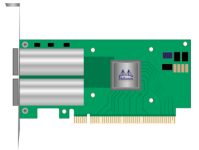
1. PCIe (PCIe-Bench [SIGCOMM'18])
2. DDIO (ddio-bench [ATC'20])
3. Memory bandwidth (Host Interconnect Congestion [HotNet'22])
4. IOMMU/IOTLB (DAMN [ASPLOS'18], iommu-bench [PeerJCS'23])

These problems will continue to grow with higher bandwidth



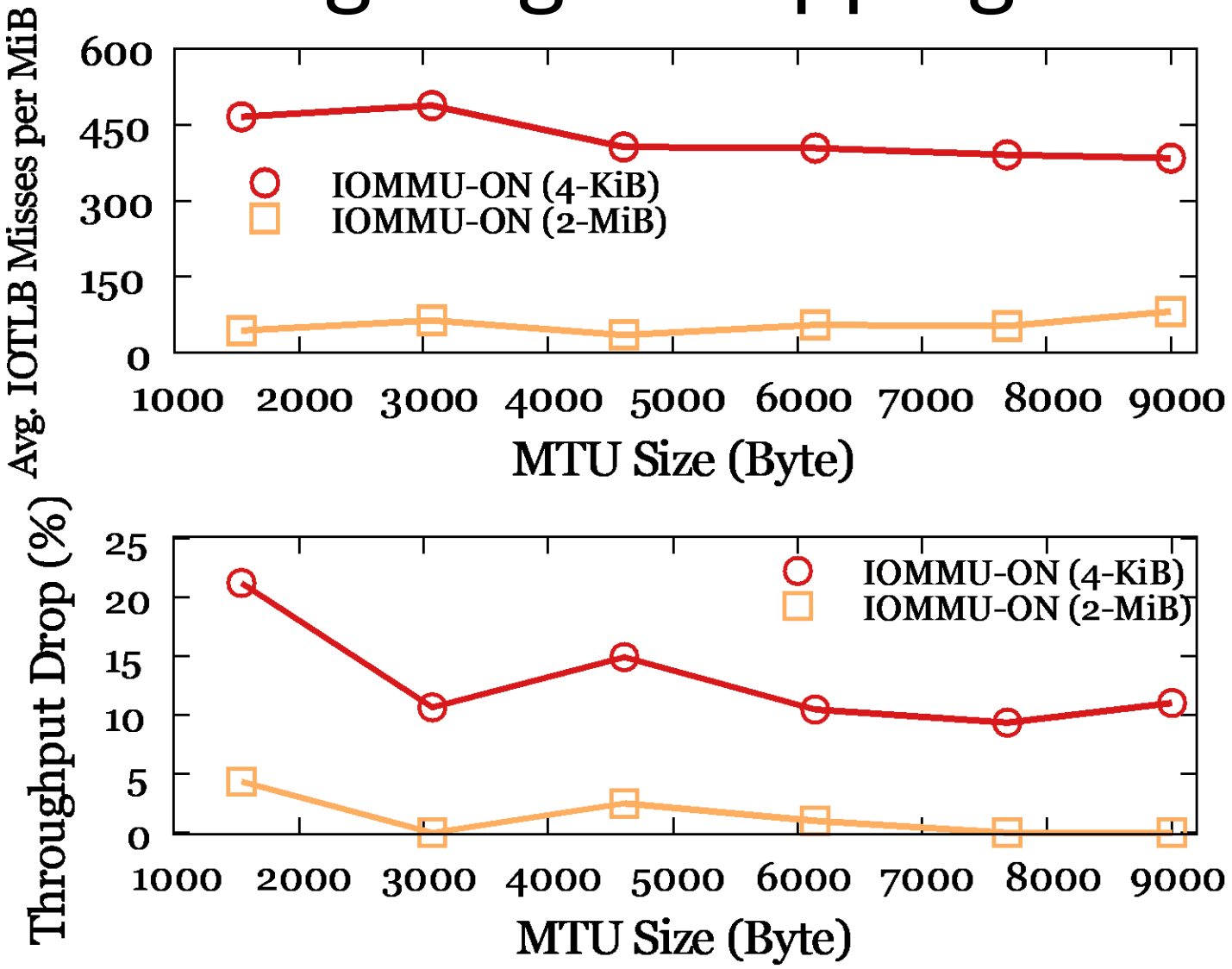
Main Memory



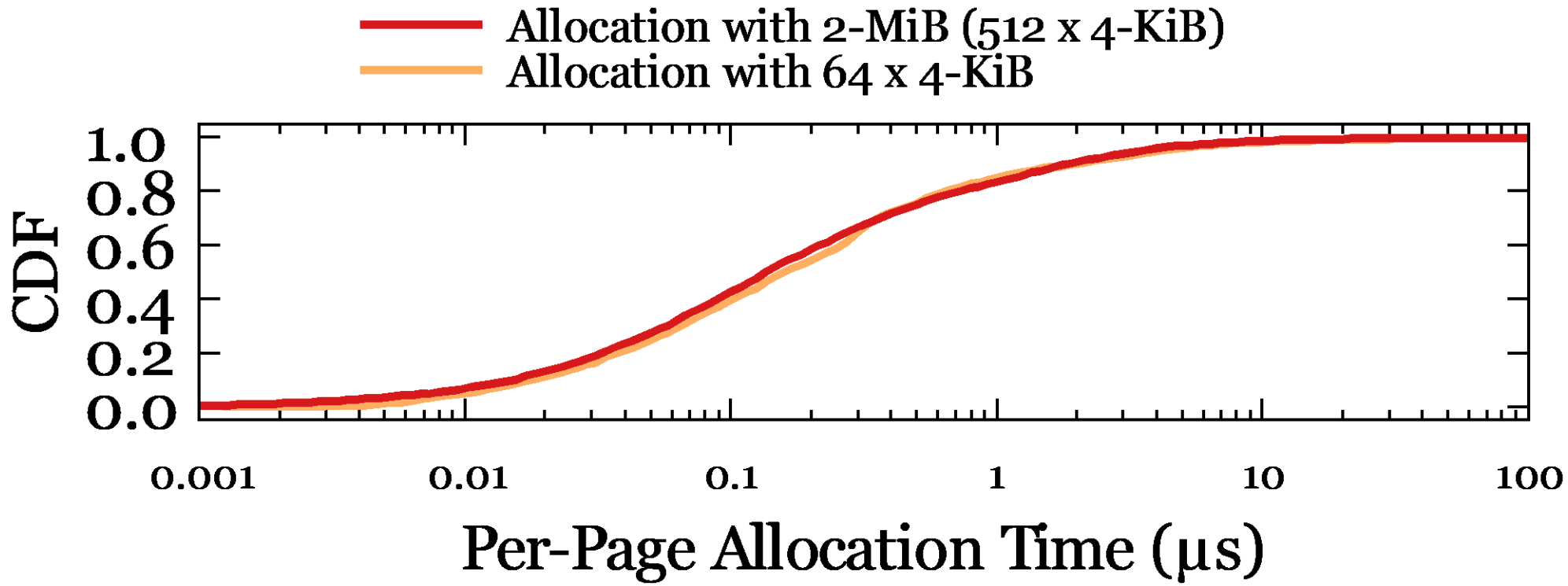


NIC

Benefits of Using Larger Mappings



Per-Page Allocation Cost



IOMMU – Prior Works (1/2)

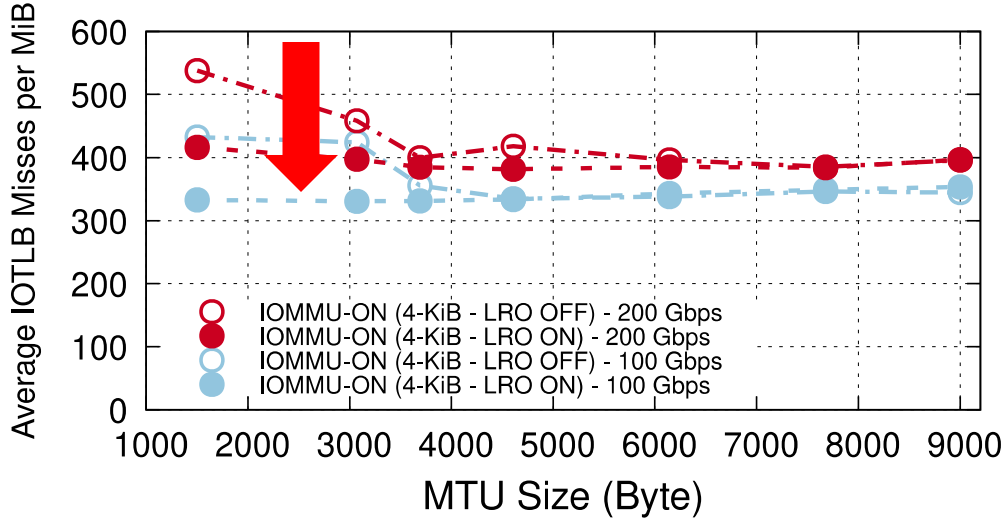
- **Utilizing the IOMMU Scalably [ATC'15]**
  - Introduces deferred IOTLB invalidation and then optimizes the implementation of dma_map() and dma_unmap() to minimize the locks and waiting time to allocate an IOVA. They introduce a cache for recently freed IOVA to avoid accessing the red-black tree holding pairwise-disjoint ranges of allocated virtual I/O page numbers.
- **DAMN [ASPLOS'18]**
  - Present a memory allocator to provide both security and performance. It uses permanently mapped buffers for IOMMu to prevent performing extra map/unmap. It is similar to our solution, but it focuses on managing 4-KiB buffers and requires changes to the page data structure.

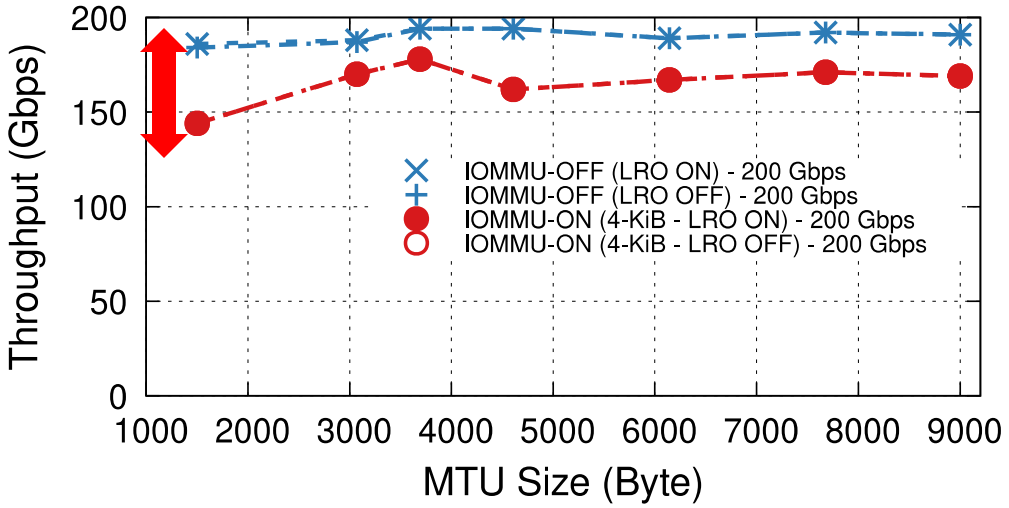
IOMMU – Prior Works (2/2)

- **rIOMMU [ASPLOS'15]**
  - Introduces a flat table to improve the performance of IOMMU, which is based on the characteristics of circular ring buffers.
- There are more works that focus on DMA attacks (e.g., sub-4-KiB vulnerabilities) and mapping VM pages into the IOMMU hardware.

Impact of Offloading Features – LRO*

LRO reduces the number of IOTLB misses per MiB, but not enough to overcome the IOTLB wall in our testbed





* Large Receive Offload (LRO)

Other Analysis - Takeaways

- AMD EPYC 74F3 (3G-Milan) vs. Intel Xeon Gold 6346 (Ice Lake)
  - IOMMU imposes a lower overhead on AMD EPYC, but it cannot achieve line rate for MTUs smaller than 3000 bytes
- Intel E810 vs. NVIDIA/Mellanox ConnectX-6 at 100 Gbps
  - NVIDIA/Mellanox results in a slightly smaller number of IOTLB misses