Advanced Computer Networks

Data Center Network Applications (I)

https://pages.cs.wisc.edu/~mgliu/CS740/F25/index.html

Ming Liu
mgliu@cs.wisc.edu

Outline

- Last lecture
  - Endhost Network Stack in Data Center Networks (II)

- Today
  - Data Center Network Applications (I)

- Announcements
  - In-class Exam 11/20/2025
  - Project Presentation on 12/04/2025 and 12/09/2025

Which application does this paper target?

Which application does this paper target?

Video Streaming

NETFLIX hulu HLS

Disney+ peacock

Which application does this paper target?

Video Streaming @Server Side

NETFLIX
hulu
HLS
Disney+
peacock

Video Streaming: The Nextflix Example

- Netflix maintains its own CDN infrastructure
  - PoPs and data centers
- System architecture
  - FreeBSD OS
  - Nginx web server
  - HTTP —> HTTPS
- Client view: rate-adaptive long-lived TCP flows
  - Adaptive bitrate (ABR) algorithm

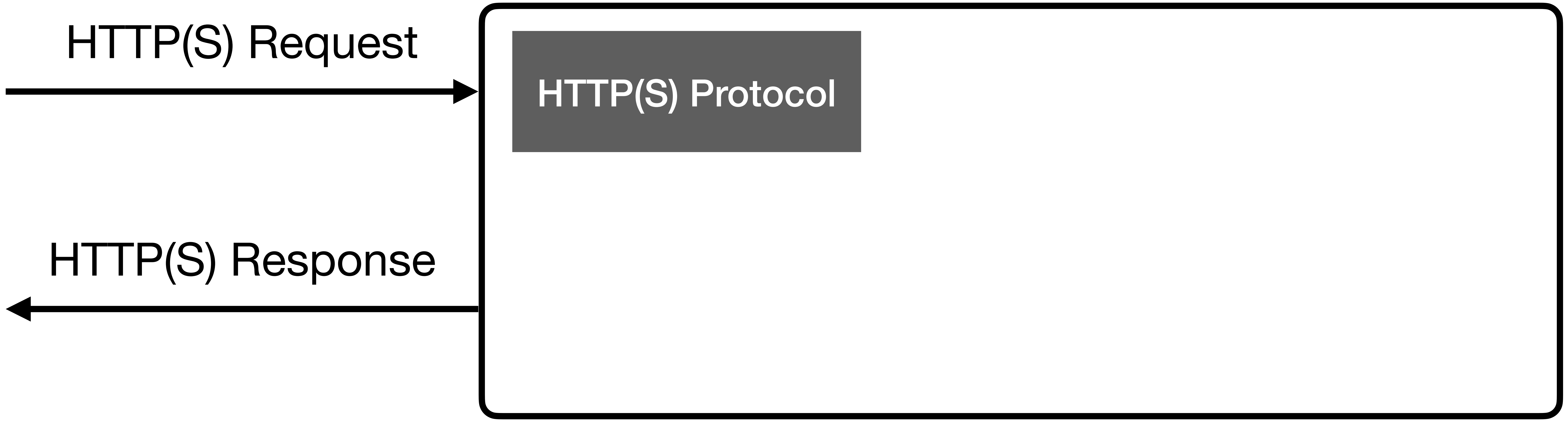
Data Path of Video Streaming

- Nearly read-only when serving
  - Updates are scheduled

HTTP(S) Request
HTTP(S) Response

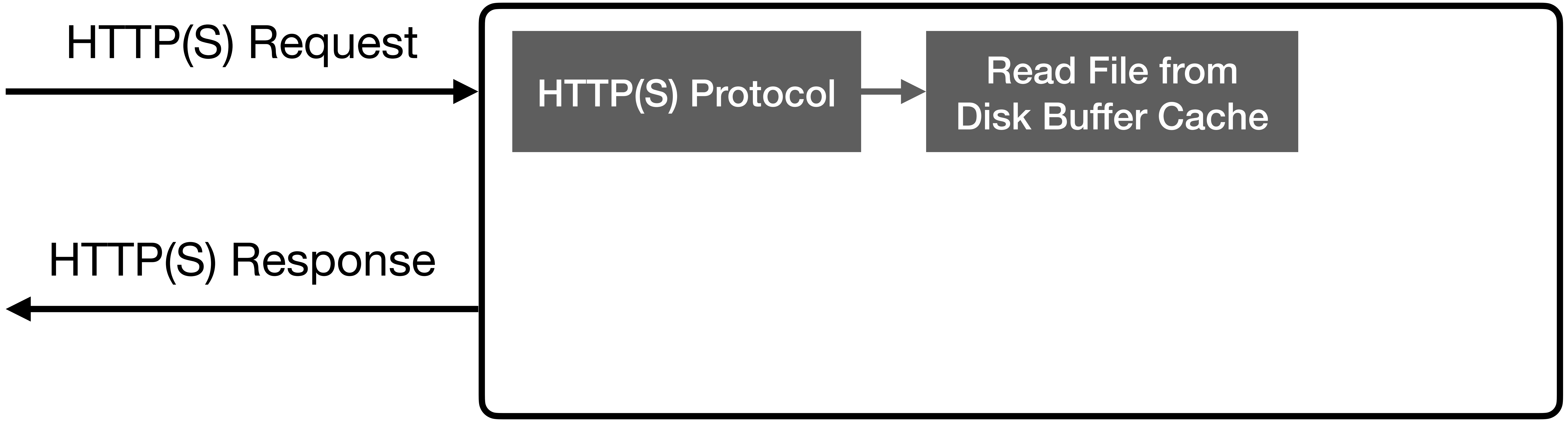
Data Path of Video Streaming

- Nearly read-only when serving
    - Updates are scheduled



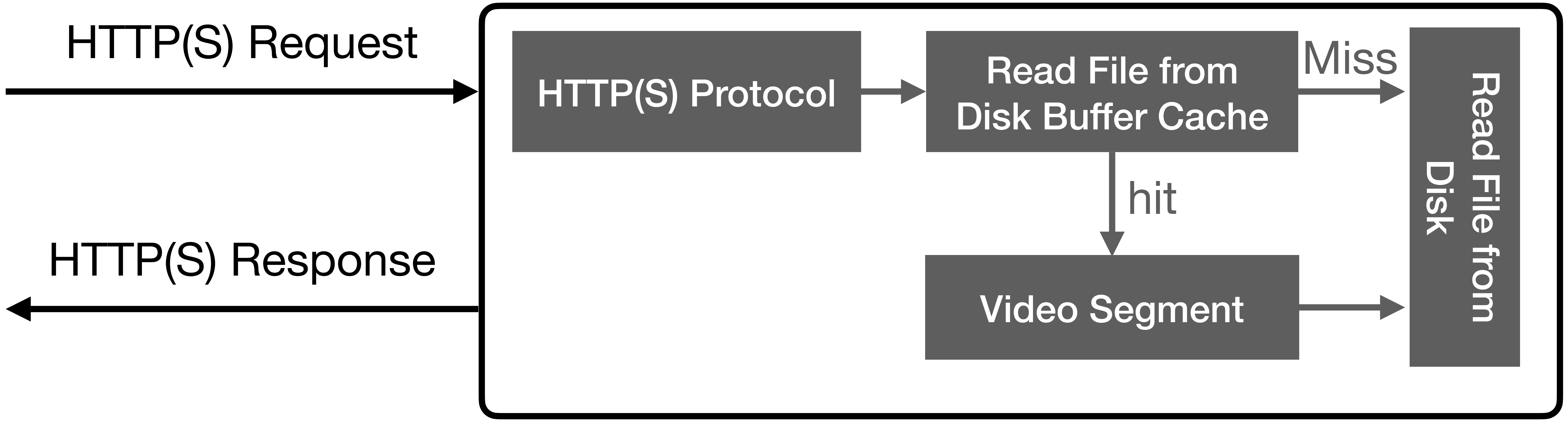
Data Path of Video Streaming

- Nearly read-only when serving
  - Updates are scheduled



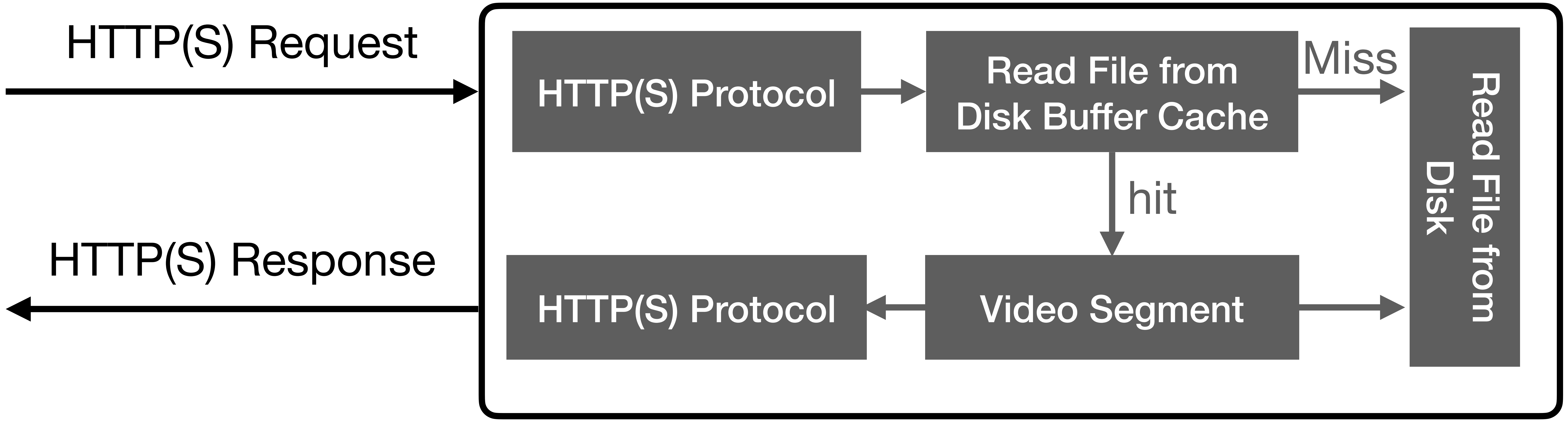
Data Path of Video Streaming

- Nearly read-only when serving
  - Updates are scheduled
- Use `sendfile` for zero-copy
  - Map a file page to `sf_buf` and attach an `mbuf` header
  - Enqueue it to the socket buffer



Data Path of Video Streaming

- Nearly read-only when serving
  - Updates are scheduled
- Use `sendfile` for zero-copy
  - Map a file page to `sf_buf` and attach an `mbuf` header
  - Enqueue it to the socket buffer



Data Path of Video Streaming

- Nearly read-only when serving
  - Updates are scheduled
- Use `sendfile` for zero-copy
  - Map a file page to `sf_buf` and attach an `mbuf` header
  - [illegible]

Looks simple! So what is the problem?

HTTP(S) Response
HTTP(S) Protocol
Video Segment
from

The Video Streaming Performance Problem

- I/O throughput is unable to saturate!

- BBC iPlayer streaming service achieved 20Gb/s in Dec. 2015
  - Two Intel Xeon E5-2680v3 processor (24 cores)
  - 512GB DDR4
  - 8.6TB RAID array of SSDs
  - 40Gbps NIC

The Video Streaming Performance Problem

- I/O throughput is unable to saturate!
- BBC iPlayer streaming service achieved 20Gb/s in Dec. 2015
  - Two Intel Xeon E5-2680v3 processor (24 cores)
  - 512GB DDR4
  - 8.6TB RAID array of SSDs
  - 40Gbps NIC

How can we do better?

The Netflix Ad-Hoc Optimizations

- Issue #1: synchronous `sendfile`
    - Blocked I/Os, hindering event-driven applications
    - Low hit rate in the disk buffer cache (less than 10%)

The Netflix Ad-Hoc Optimizations

- Issue #1: synchronous `sendfile`
  - Blocked I/Os, hindering event-driven applications
  - Low hit rate in the disk buffer cache (less than 10%)

- Solution #1: asynchronous `sendfile`
  - System calls return immediately before the I/O is completed
  - The socket starts to transmits only when all inflight I/Os are ready

The Netflix Ad-Hoc Optimizations

- Issue #2: Virtual Memory (VM) scaling
  - VM allocations are blocked when VM pages are exhausted
  - Streaming processes are stalled

The Netflix Ad-Hoc Optimizations

- Issue #2: Virtual Memory (VM) scaling
  - VM allocations are blocked when VM pages are exhausted
  - Streaming processes are stalled

- Solution #2: Optimized VM subsystem
  - Virtual NUMA domains with managed cores and memory
  - Proactively reclaim under a low threshold
  - Batched re-enqueue reclaimed pages

The Netflix Ad-Hoc Optimizations

- Issue #3: high ingress packet rates (incast)
  - LRO (large receive offload) becomes ineffective under interleaved flows
  - Cause more CPU cycles

The Netflix Ad-Hoc Optimizations

- Issue #3: high ingress packet rates (incast)
  - LRO (large receive offload) becomes ineffective under interleaved flows
  - Cause more CPU cycles

- Solution #3: RSS-assisted TCP LRO
  - Sort incoming TCP packets into buckets based on their RSS hash and the time at the end of the interrupt

The Netflix Ad-Hoc Optimizations

- Issue #4: slow HTTPS streaming
  - Zero-copy becomes useless since data needs to fall back to userspace
  - POSIX `reads` and `writes` are required again

The Netflix Ad-Hoc Optimizations

- Issue #4: slow HTTPS streaming
  - Zero-copy becomes useless since data needs to fall back to userspace
  - POSIX `reads` and `writes` are required again

- Solution #4: In-kernel TLS
  - Userspace TLS session management
  - Bulk data encryption is encoded into the `sendfile` pipeline
  - In-place encryption is not allowed as this invalidates the cache entry
  - `sendfile` clones the data to another buffer and performs encryption

Performance of Netflix Ad-Hoc Optimizations

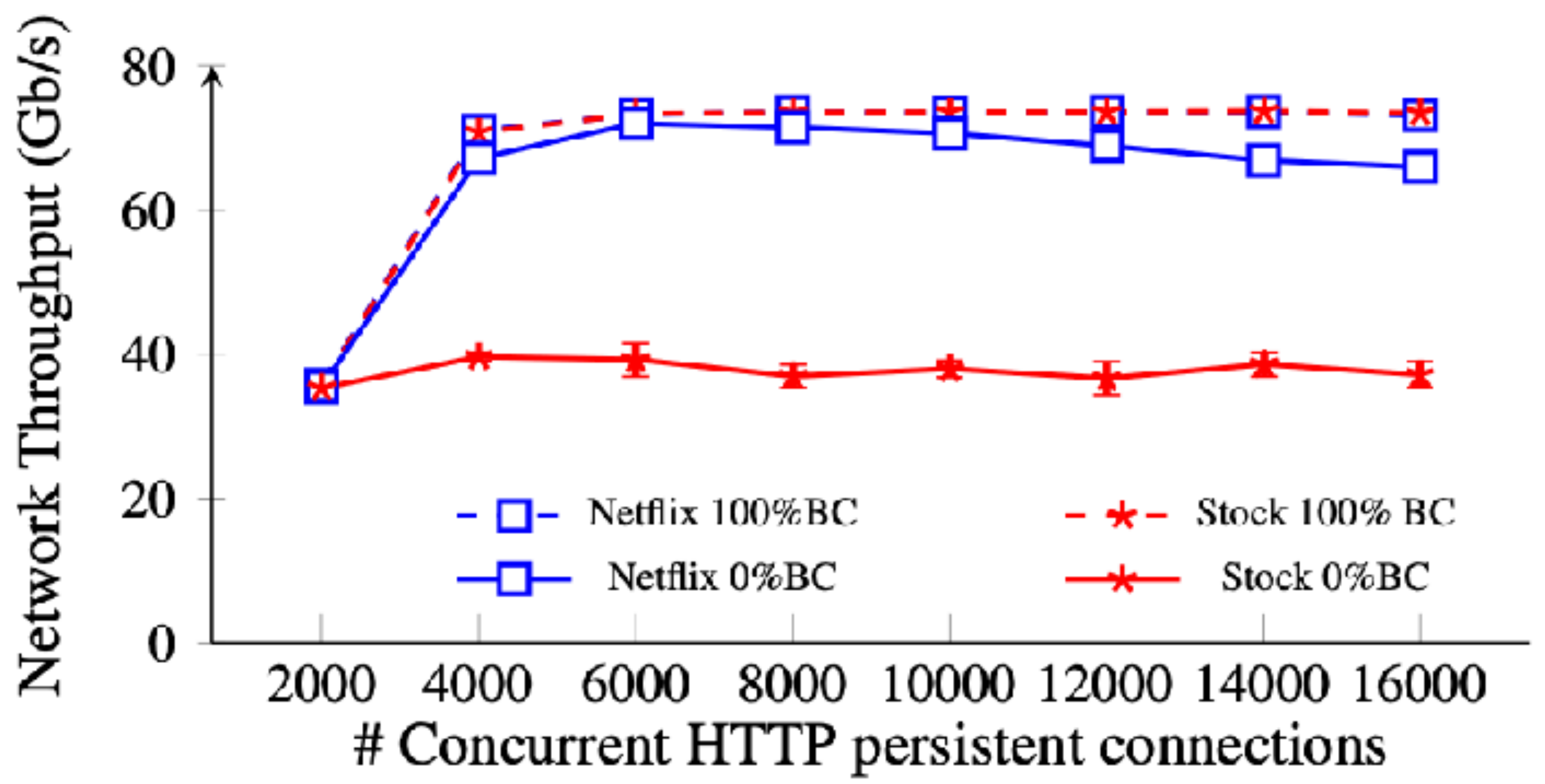
- Experimental setup
  - Intel Xeon E5-2667-v3 8-core CPU
  - 128GB RAM
  - Two dual-port Chelsio T580 40GbE adapters
  - Four P3700 NVMe disks with 800GB

- Comparison
  - Netflix stack
  - Stock nginx/FreeBSD

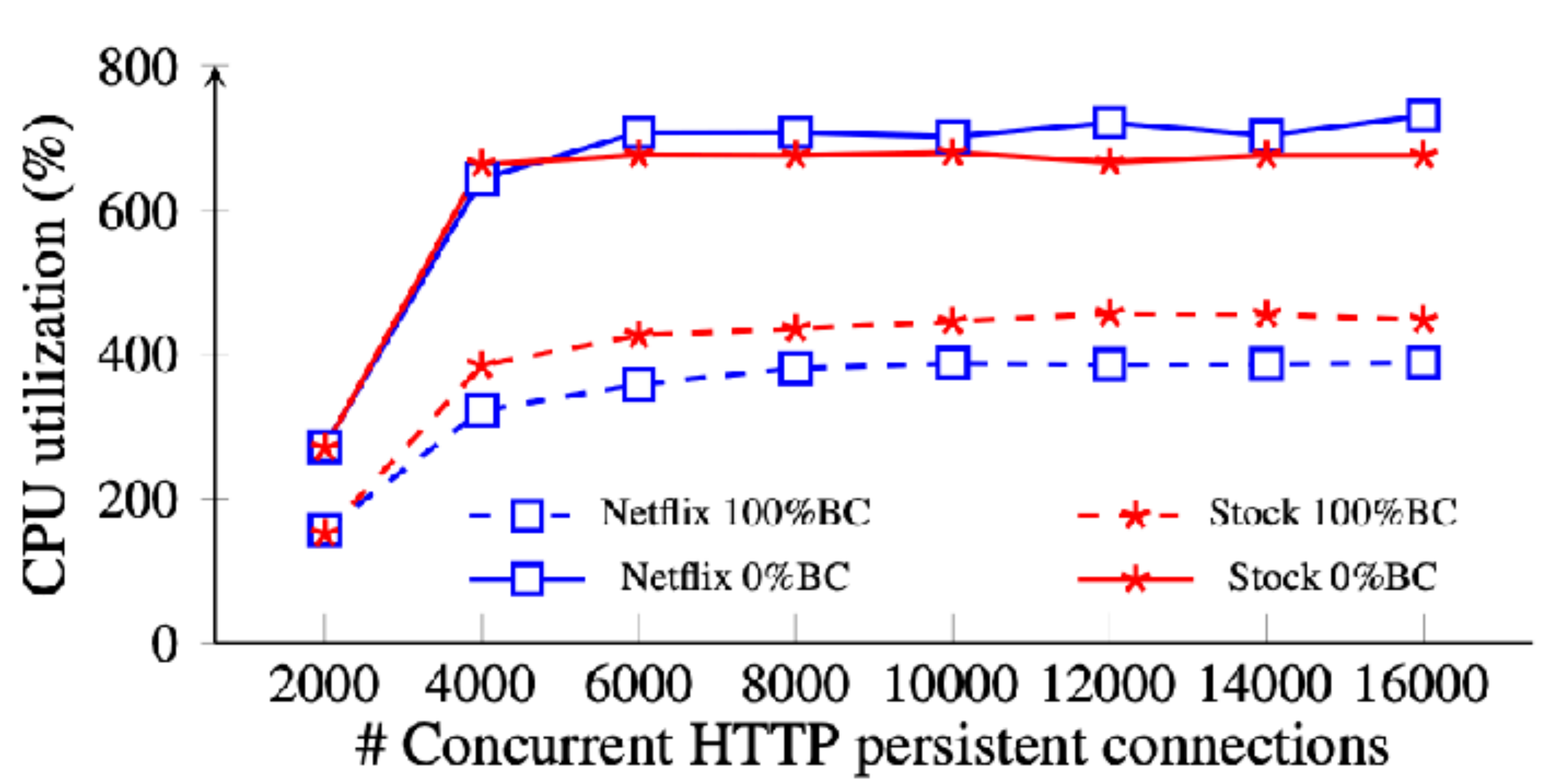
Netflix Ad-Hoc Optimizations are Effective

- Double throughput from 39Gb/s to 72Gb/s when from SSDs
  - 8 cores are nearly saturated

Plaintext



(a) Network throughput (Error bars indicate the 95% CI)

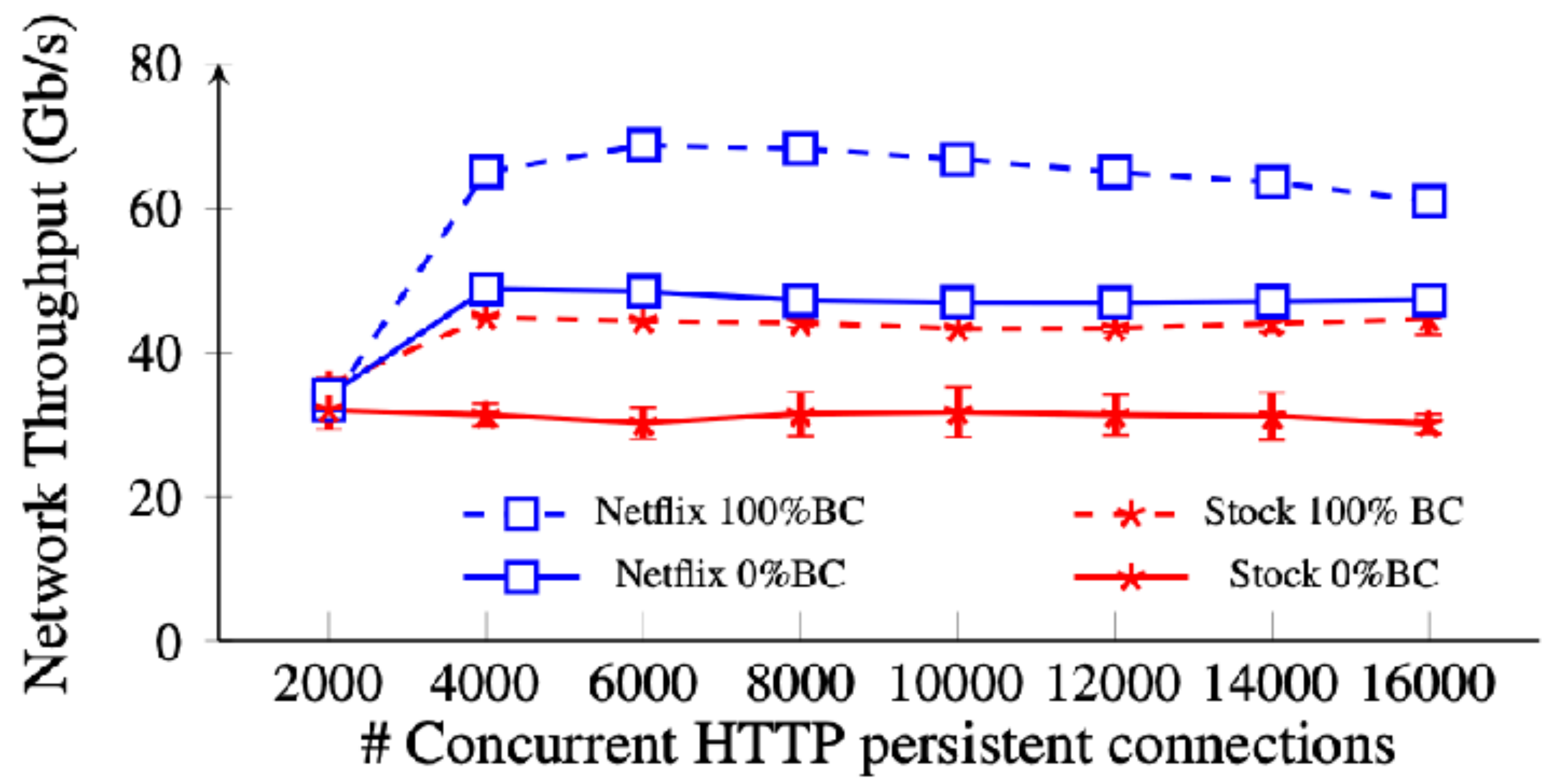


(b) CPU utilization

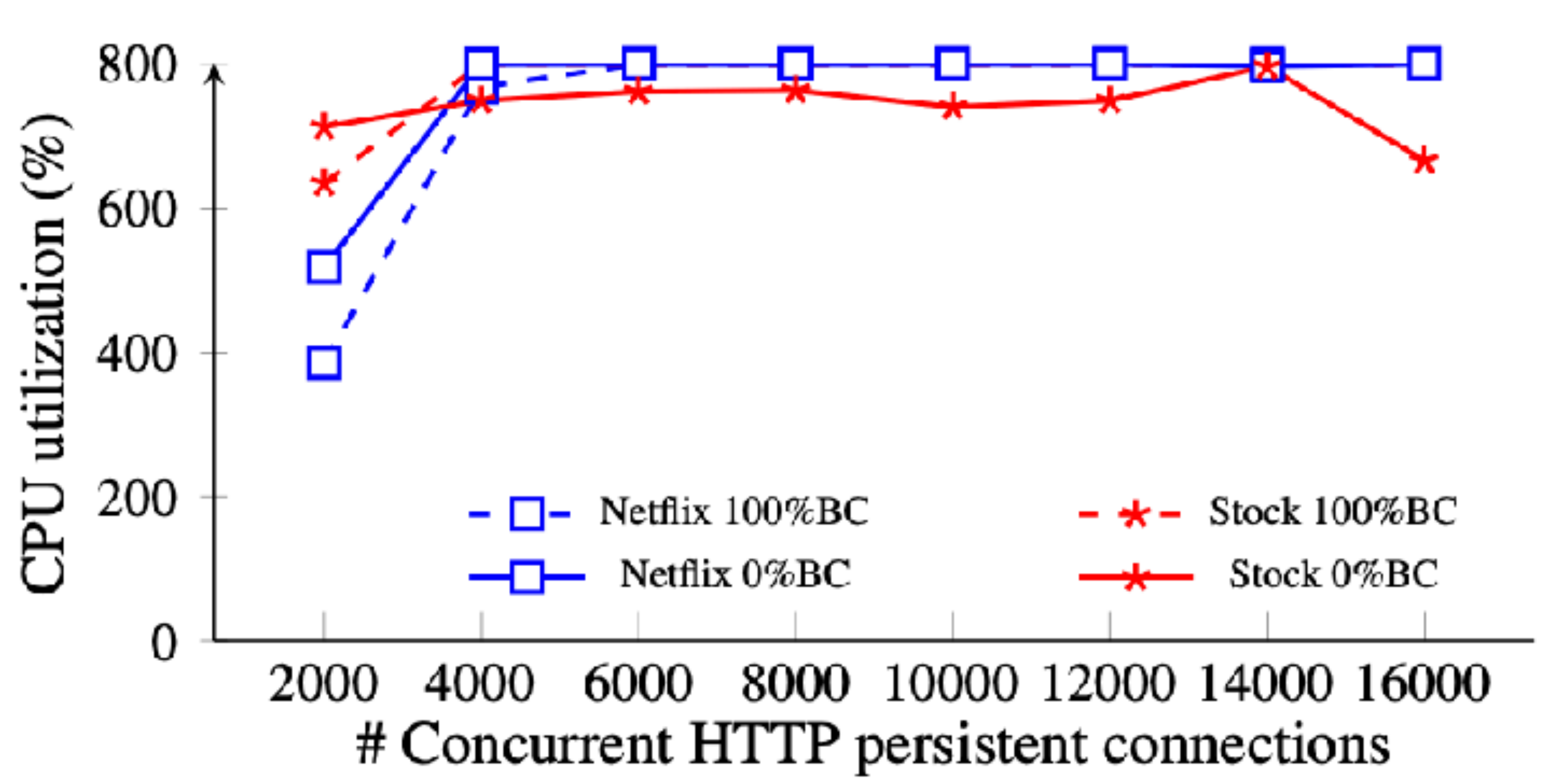
Netflix Ad-Hoc Optimizations are Heavy

- Throughput drops to 47Gb/s when serving from SSDs
  - Memory accesses are becoming a bottleneck

Encryption



(a) Network throughput (Error bars indicate the 95% CI)



(b) CPU utilization

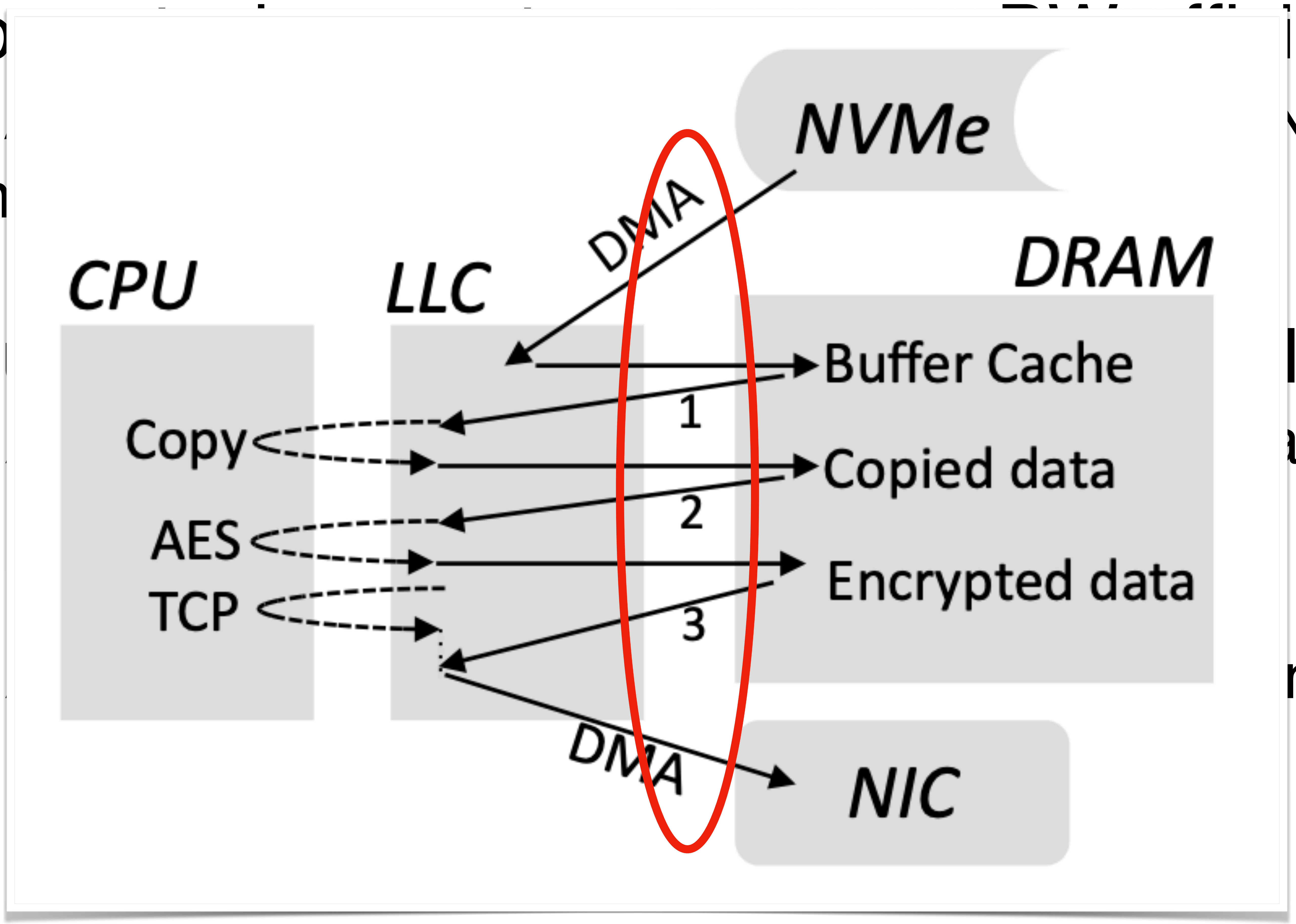
Why Memory Becomes the Bottleneck

- Asynchronous stack cannot use memory BW efficiently
  - Ideally, DMA data from the SSD to LLC, then DMA it to NIC
  - Not touching memory

- Data gets flushed to memory when not immediately used
  - Data is DMAed from the SSD to the disk buffer cache, landing in the LLC
  - The Kernel copies the buffer, loading it into the LLC
  - The Kernel encrypts the data, re-lading into the LLC
  - Data is DMAed to the NIC, requiring it to be loaded again

Why Memory Becomes the Bottleneck

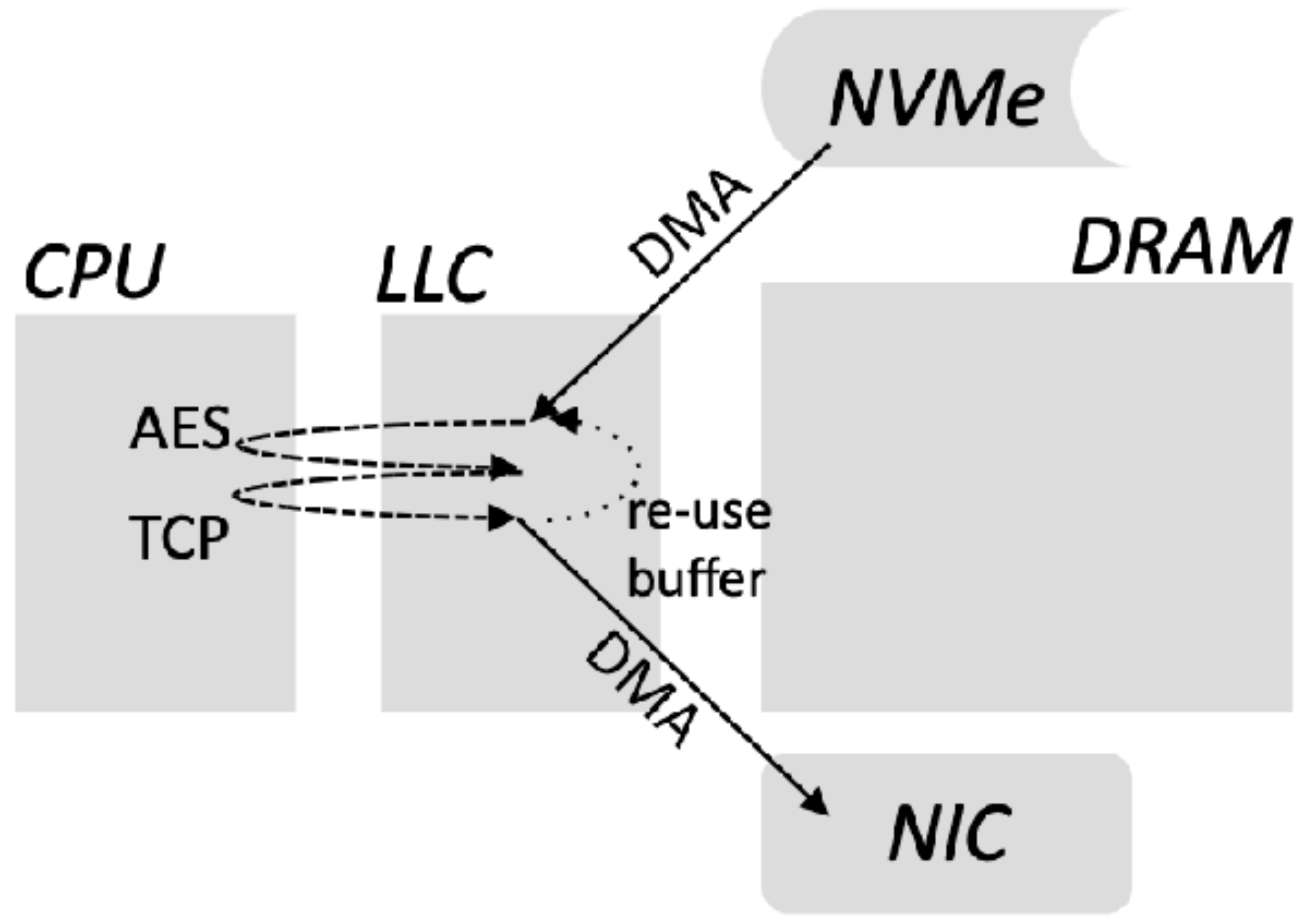
- Asynchrono   iently
    - Ideally, DM   NIC
    - Not touchin
- Data gets fl   ly used
    - Data is DM   anding in the LLC
    - The Kernel
    - The Kernel
    - Data is DM   n



How does the paper solve it?

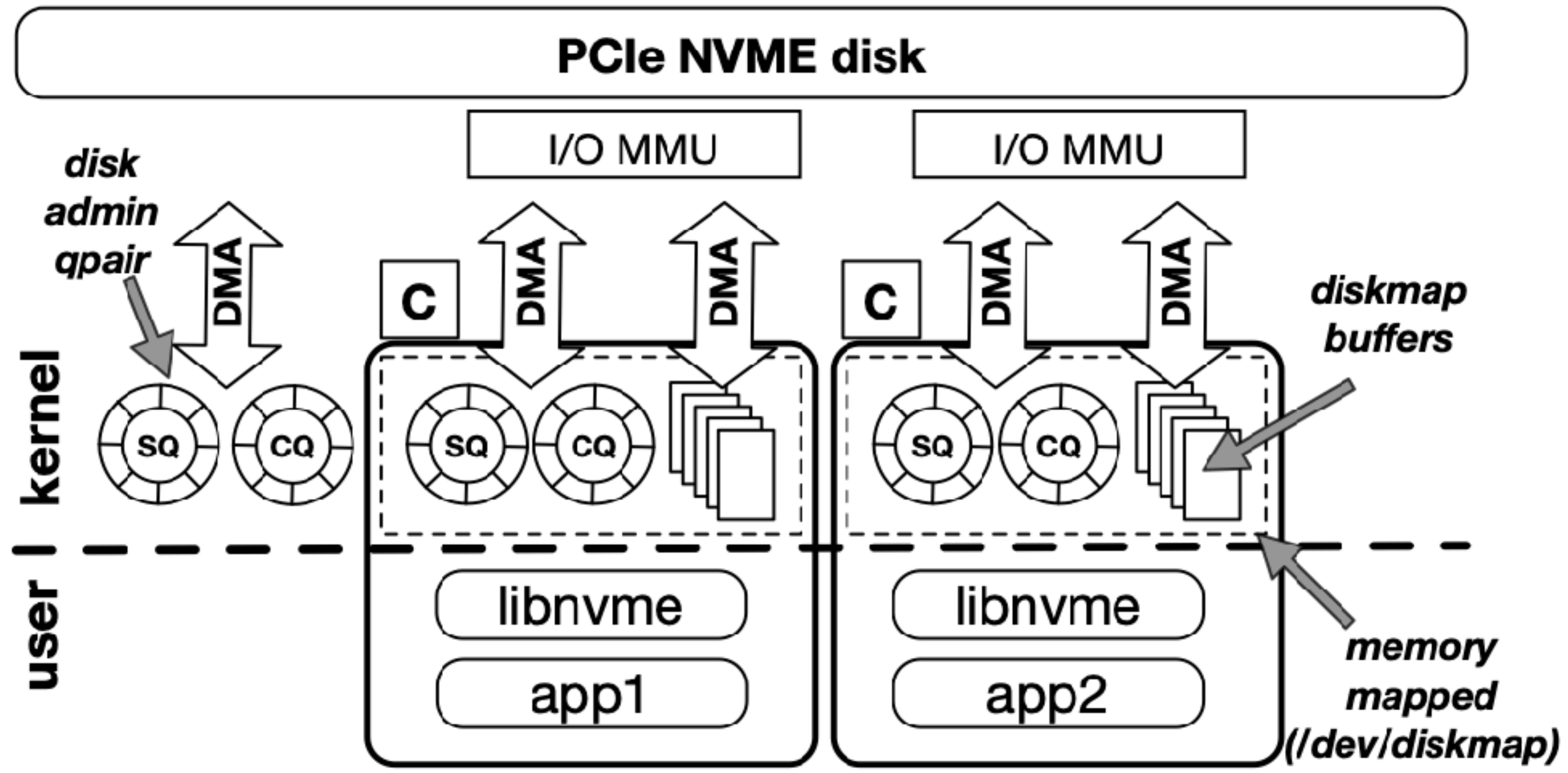
Key Idea

- Fetch the data from storage just-in-time
  - Free up congestion window when a TCP ACK arrives
  - Fetch data from SSD to fill the window
  - Put data into LLC
  - Encryption, protocol processing, and data transmission happen in LLC



Atlas Storage Stack

- diskmap: a kernel-bypass design
  - Admission queue: in-kernel
  - Submission/Completion queue: kernel-bypass

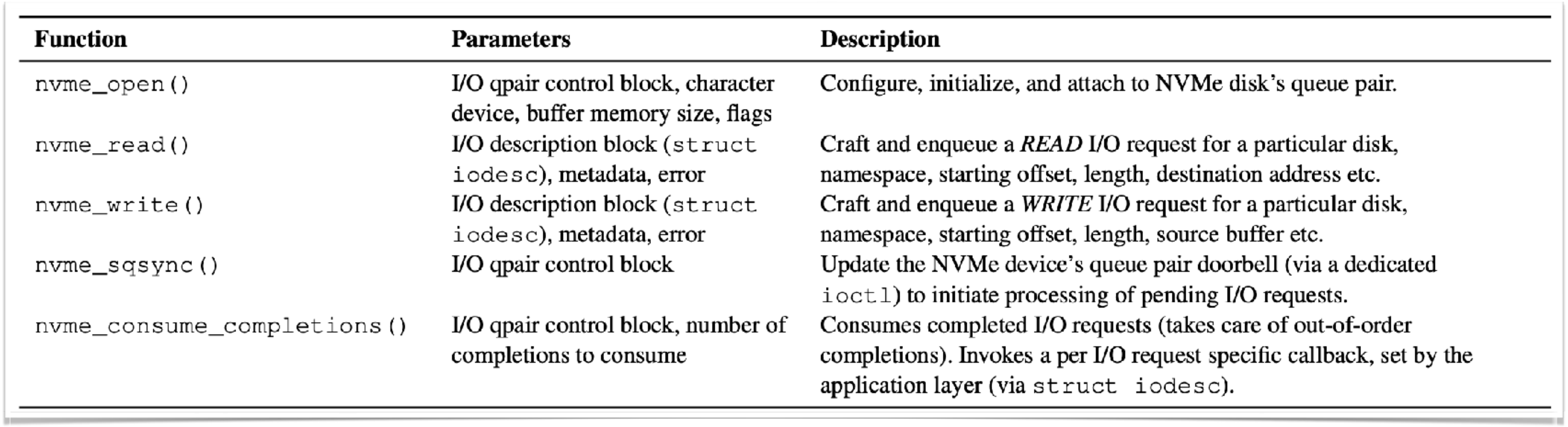


Diskmap Data Path

- #1: Apps map the shared memory into its virtual address space
  - Issue `ioctl` to indicate the disks and queue pairs
- #2: Apps call `nvme_read` or `nvme_write` to submit I/Os
  - I/O requests are enqueued into the submission queue
- #3: Apps invoke a system call to update the disk doorbell
  - Process all pending I/O commands
- #4: Apps register a callback function when I/Os complete
  - Non-blocking and event-driven model

Diskmap Data Path

- #1: Apps map the shared memory into its virtual address space
  - Issue `ioctl` to indicate the disks and queue pairs

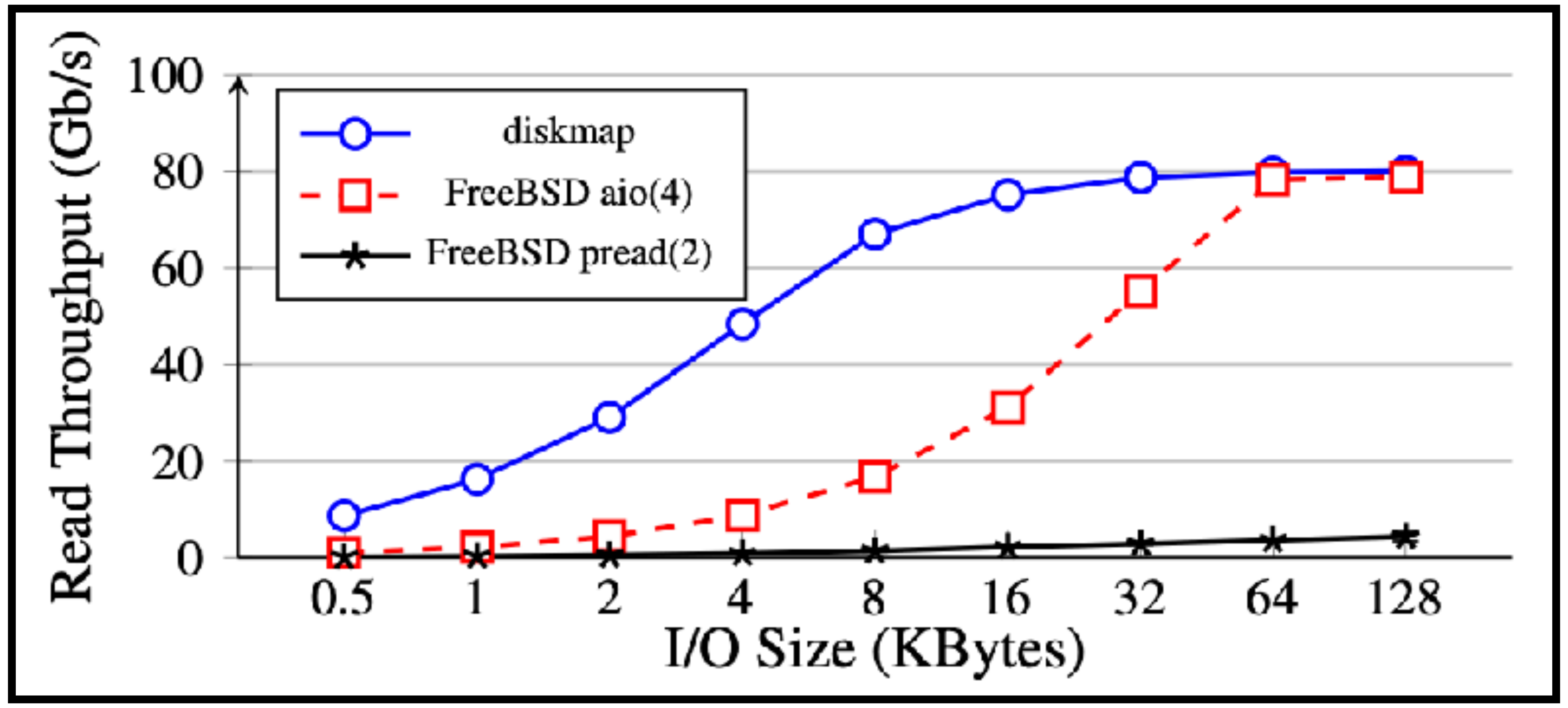


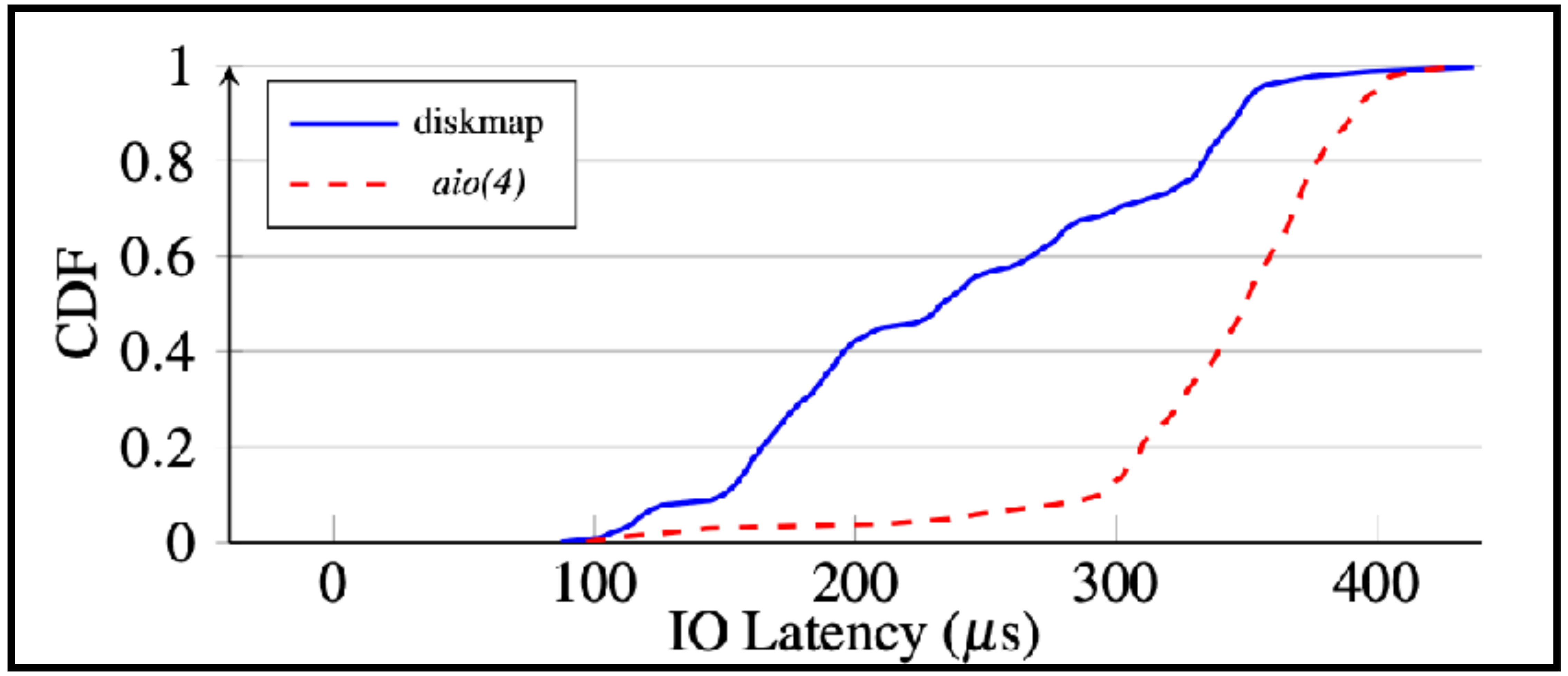
Function	Parameters	Description
`nvme_open()`	I/O qpair control block, character device, buffer memory size, flags	Configure, initialize, and attach to NVMe disk's queue pair.
`nvme_read()`	I/O description block (`struct iodesc`), metadata, error	Craft and enqueue a *READ* I/O request for a particular disk, namespace, starting offset, length, destination address etc.
`nvme_write()`	I/O description block (`struct iodesc`), metadata, error	Craft and enqueue a *WRITE* I/O request for a particular disk, namespace, starting offset, length, source buffer etc.
`nvme_sqsync()`	I/O qpair control block	Update the NVMe device's queue pair doorbell (via a dedicated `ioctl`) to initiate processing of pending I/O requests.
`nvme_consume_completions()`	I/O qpair control block, number of completions to consume	Consumes completed I/O requests (takes care of out-of-order completions). Invokes a per I/O request specific callback, set by the application layer (via `struct iodesc`).

  - Non-blocking and event-driven model

Diskmap Performance

- Faster than Aio and pread
  - Batching can amortize the system call overheads

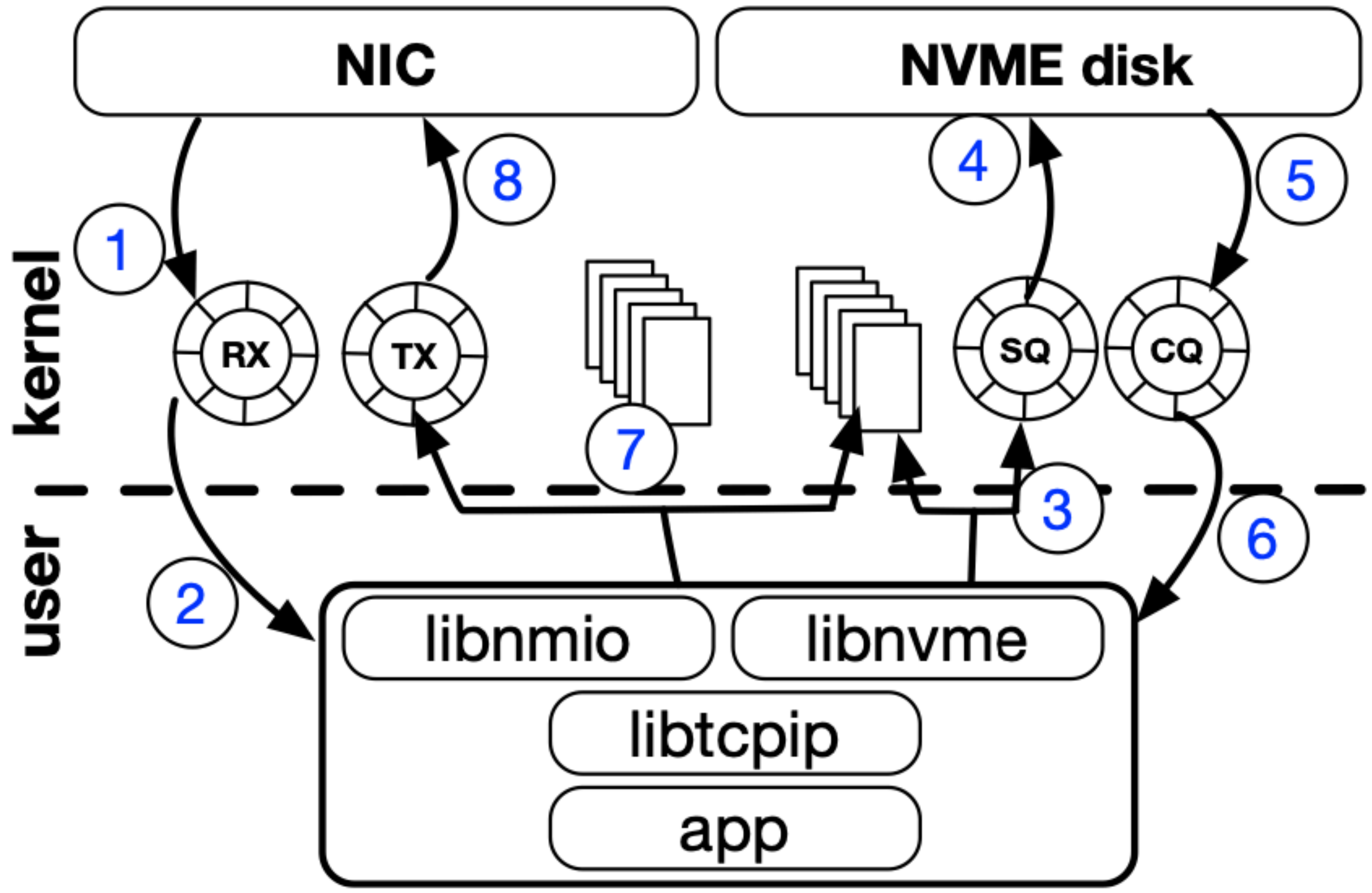




Atlas Network Stack

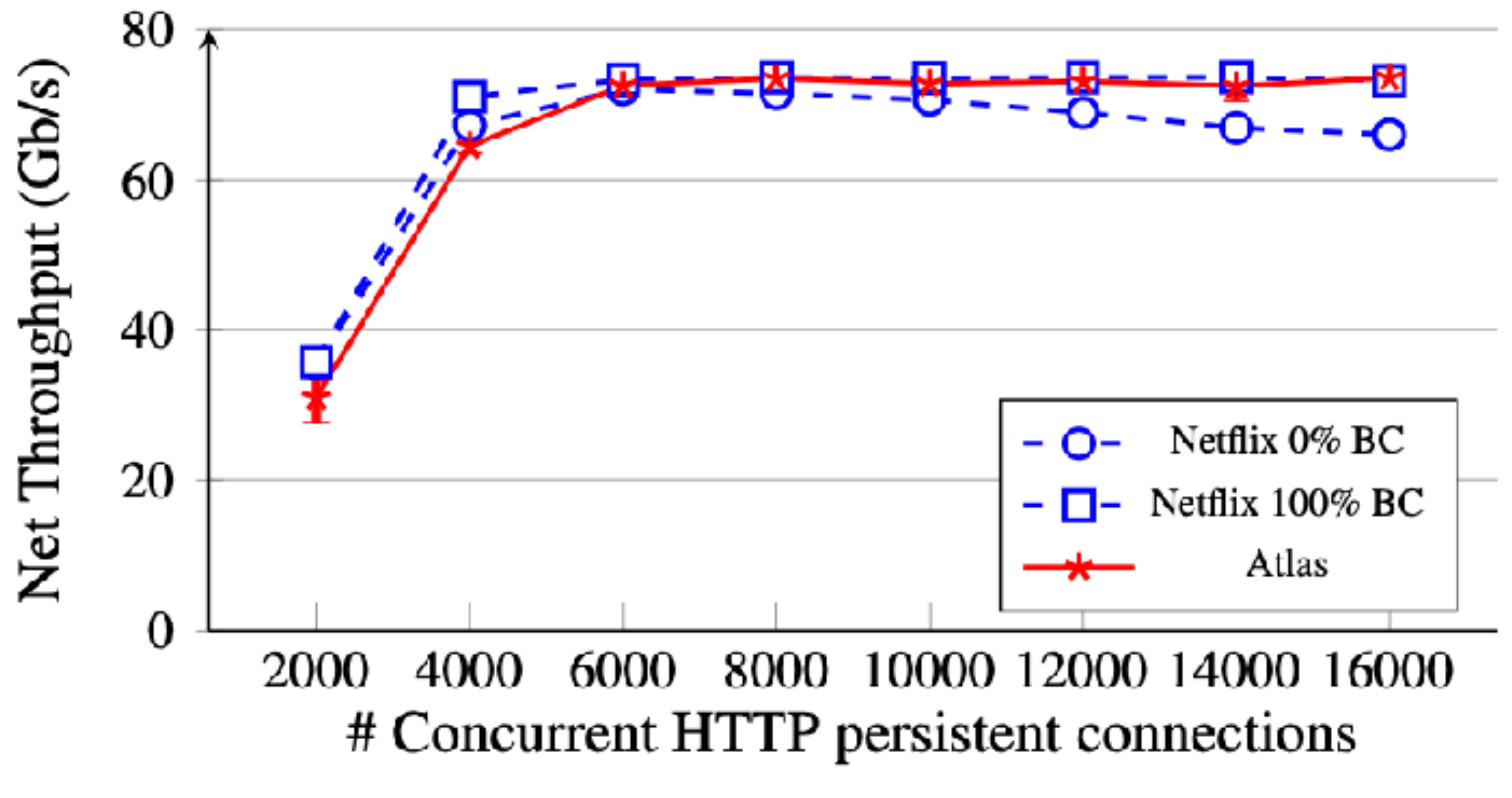
- Integrate the network stack into the diskmap path
  - Based on Sandstorm (SIGCOMM’14)
- Introduce a `tcpip_sendfile` call
  - Instruct the network stack to attach the in-memory object representation
- On-demand data fetching
  - Maxout NVMe drive performance with 8+KB blocks
- Use AES GCM for encrypted traffic
  - Retransmitted data must be re-fetched from disks

Atlas Overview

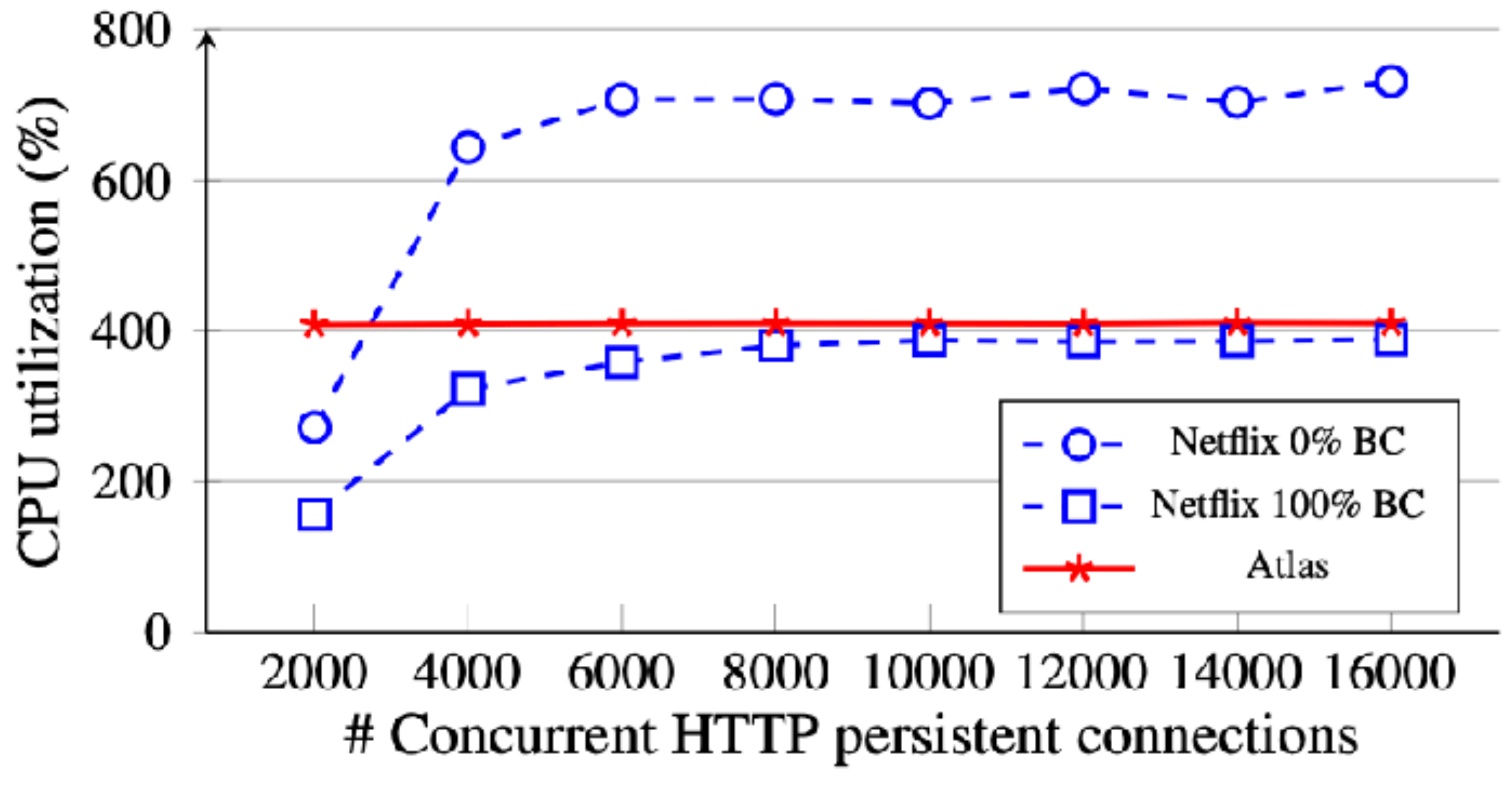


Atlas Performance under Plaintext

- Same throughput with fewer CPU cores



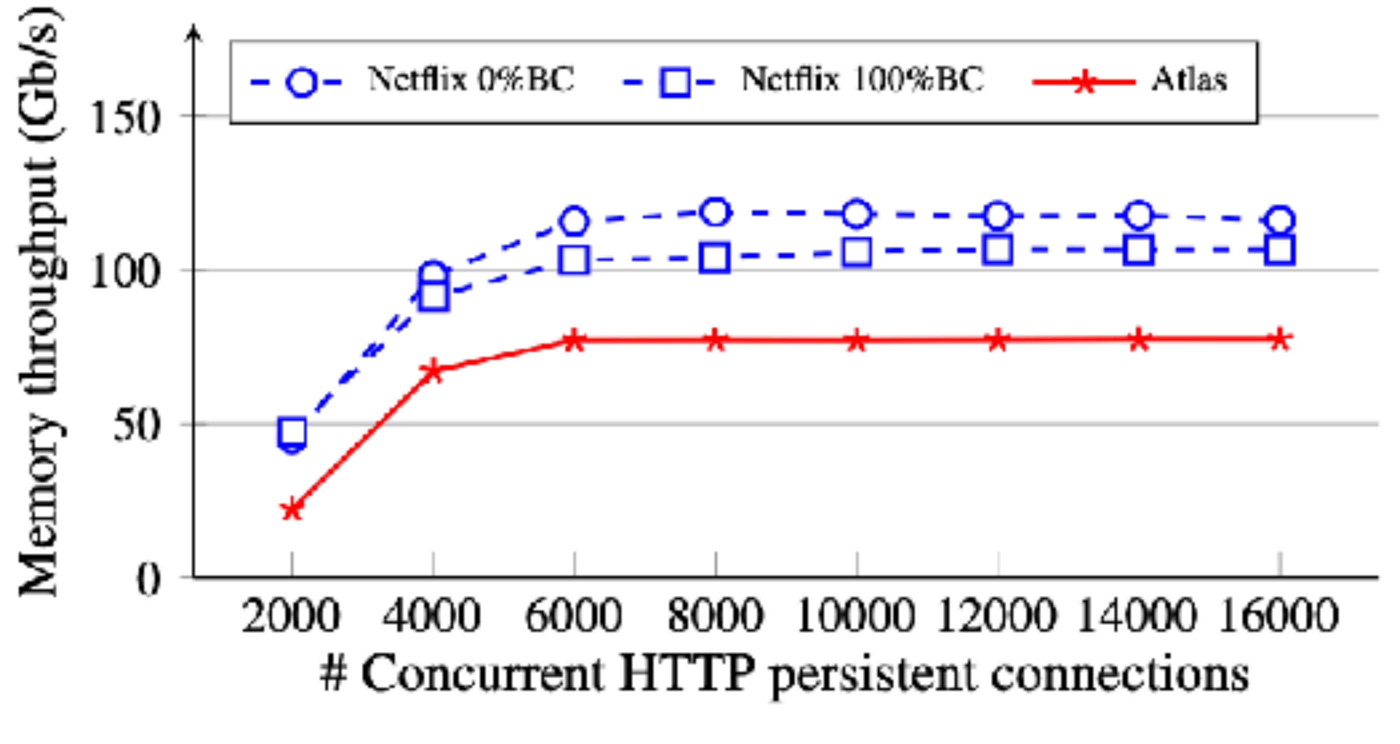
(a) Network throughput (Error bars indicate the 95% CI)



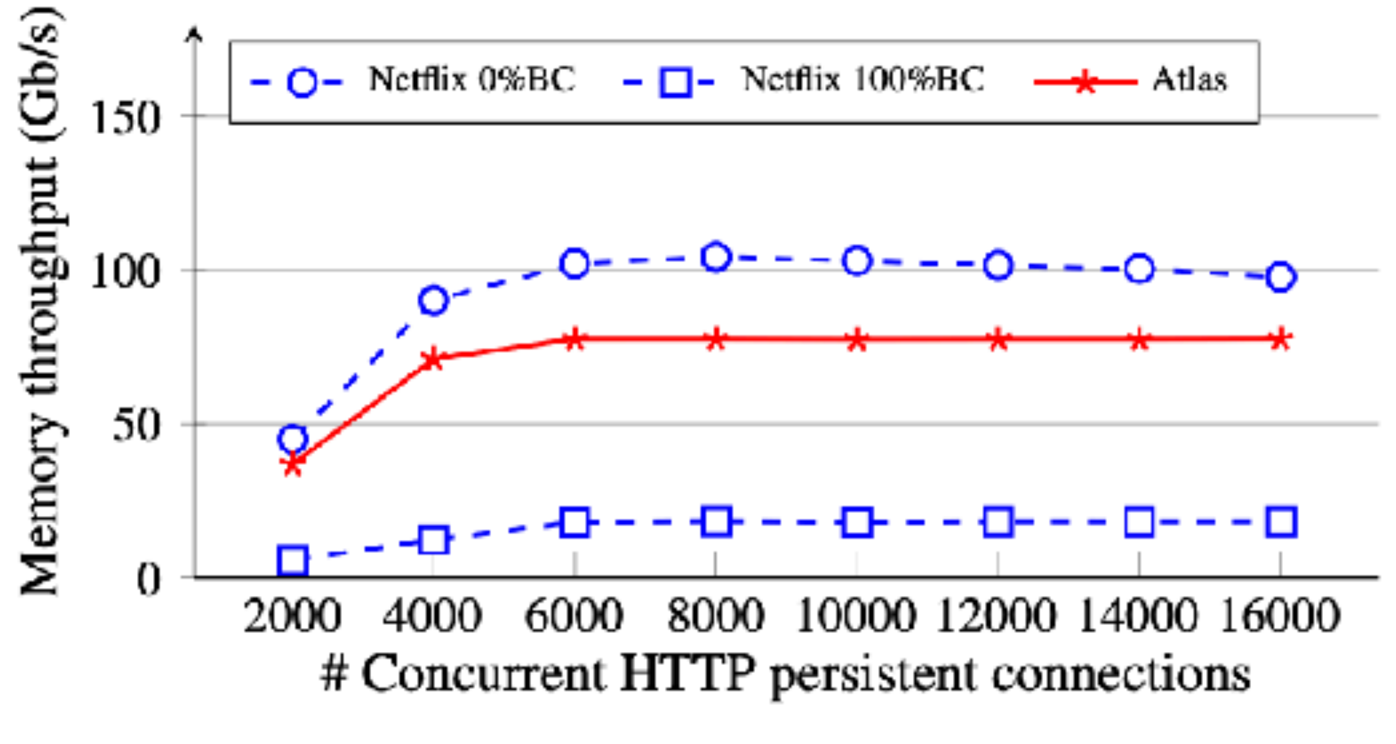
(b) CPU utilization (Average)

Atlas Performance under Plaintext (Cont'd)

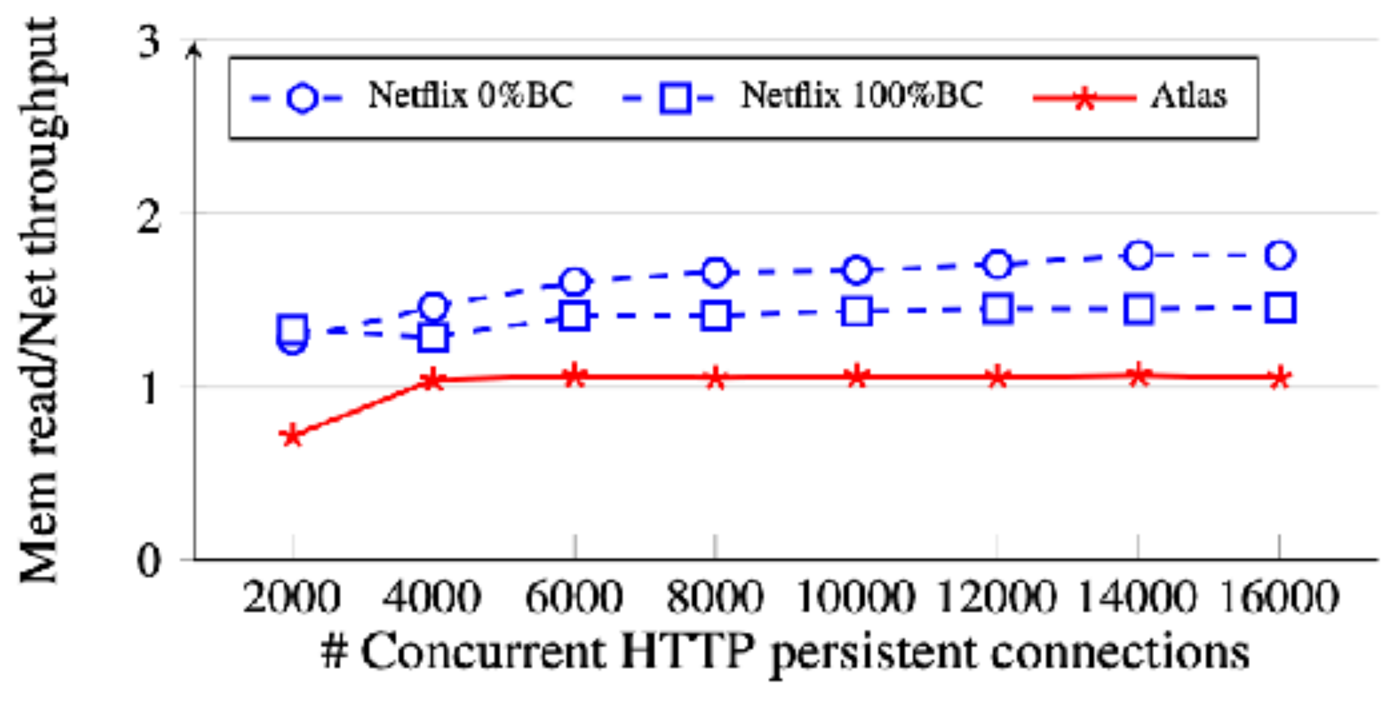
- Fewer memory accesses



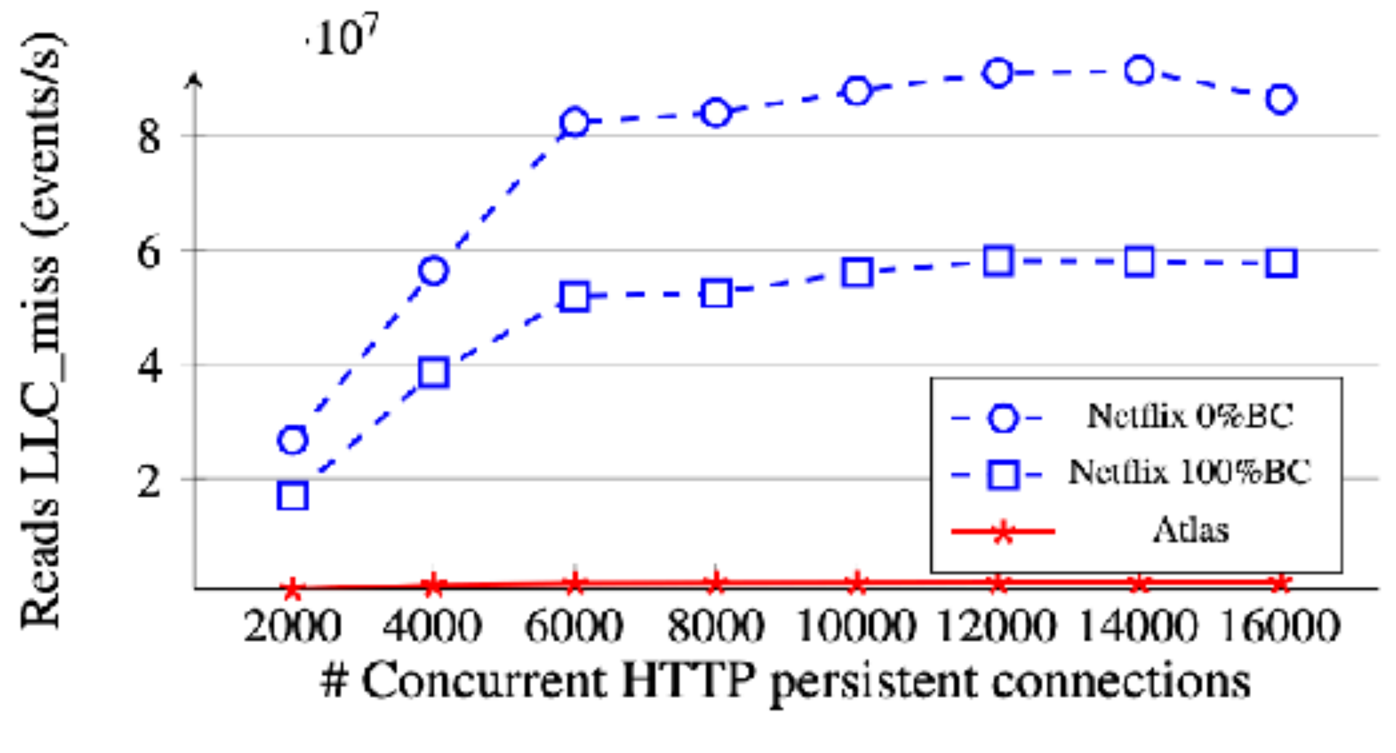
(c) Memory READ



(d) Memory WRITE



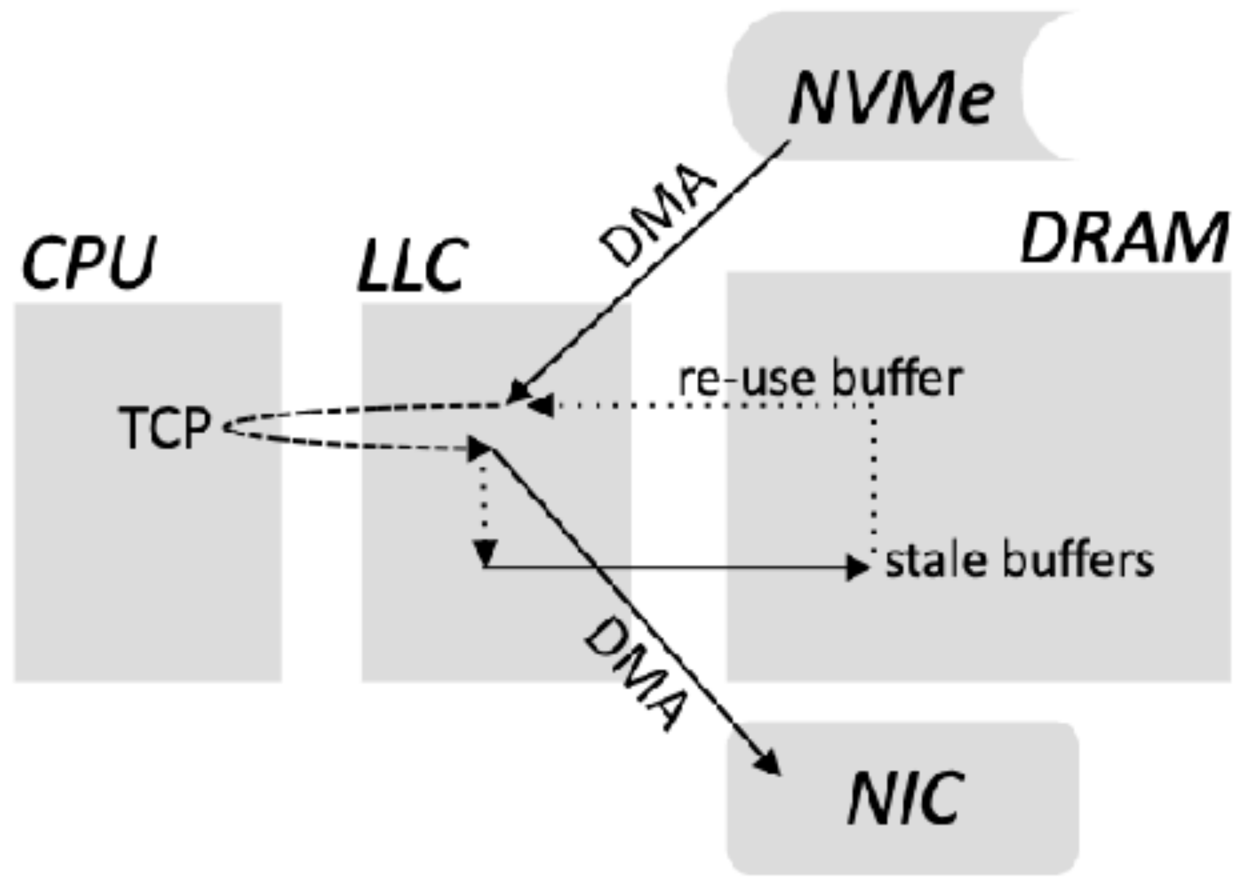
(e) Memory READ-Network Throughput Ratio



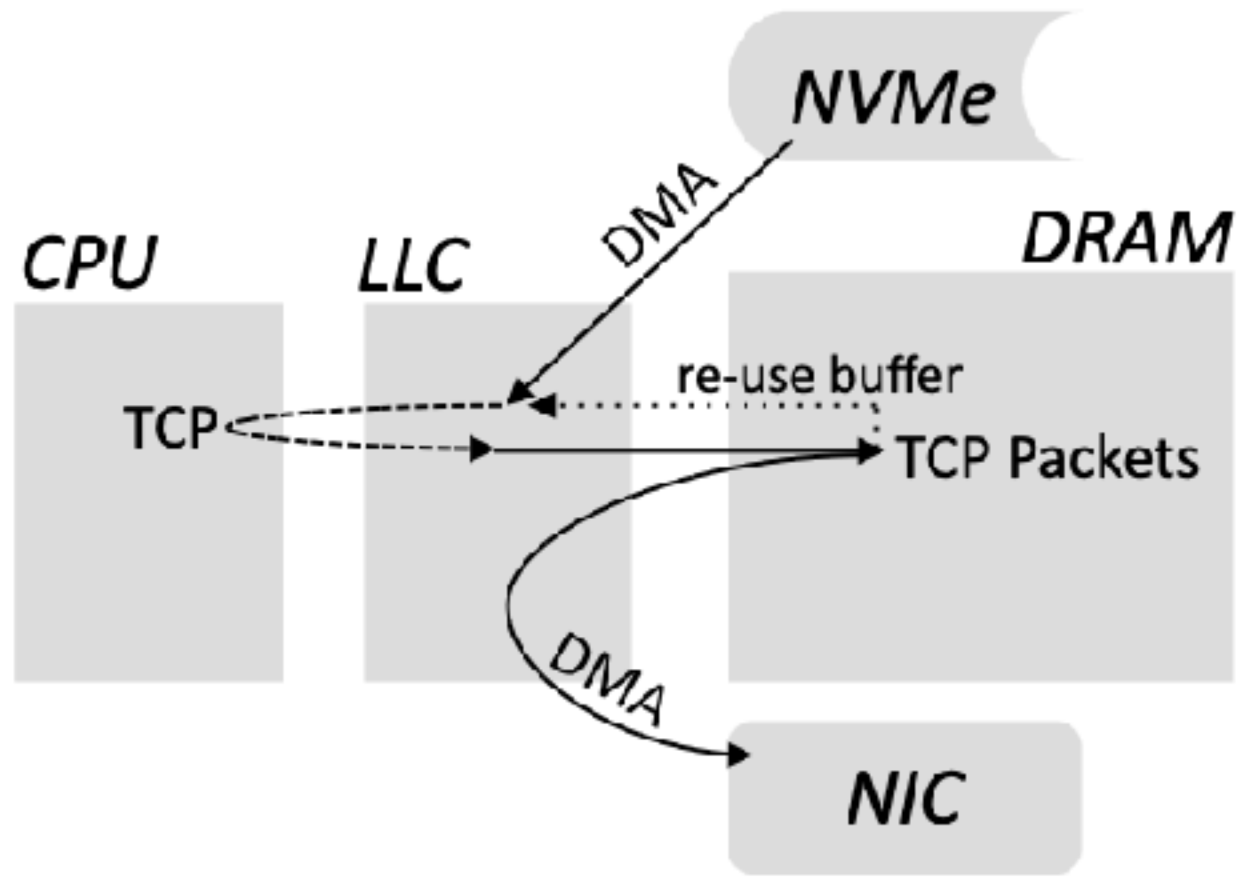
(f) CPU reads served from DRAM due to LLC misses

But, anomalies still happen

- Delayed notifications incur extra memory writes
- Contentions cause cache misses



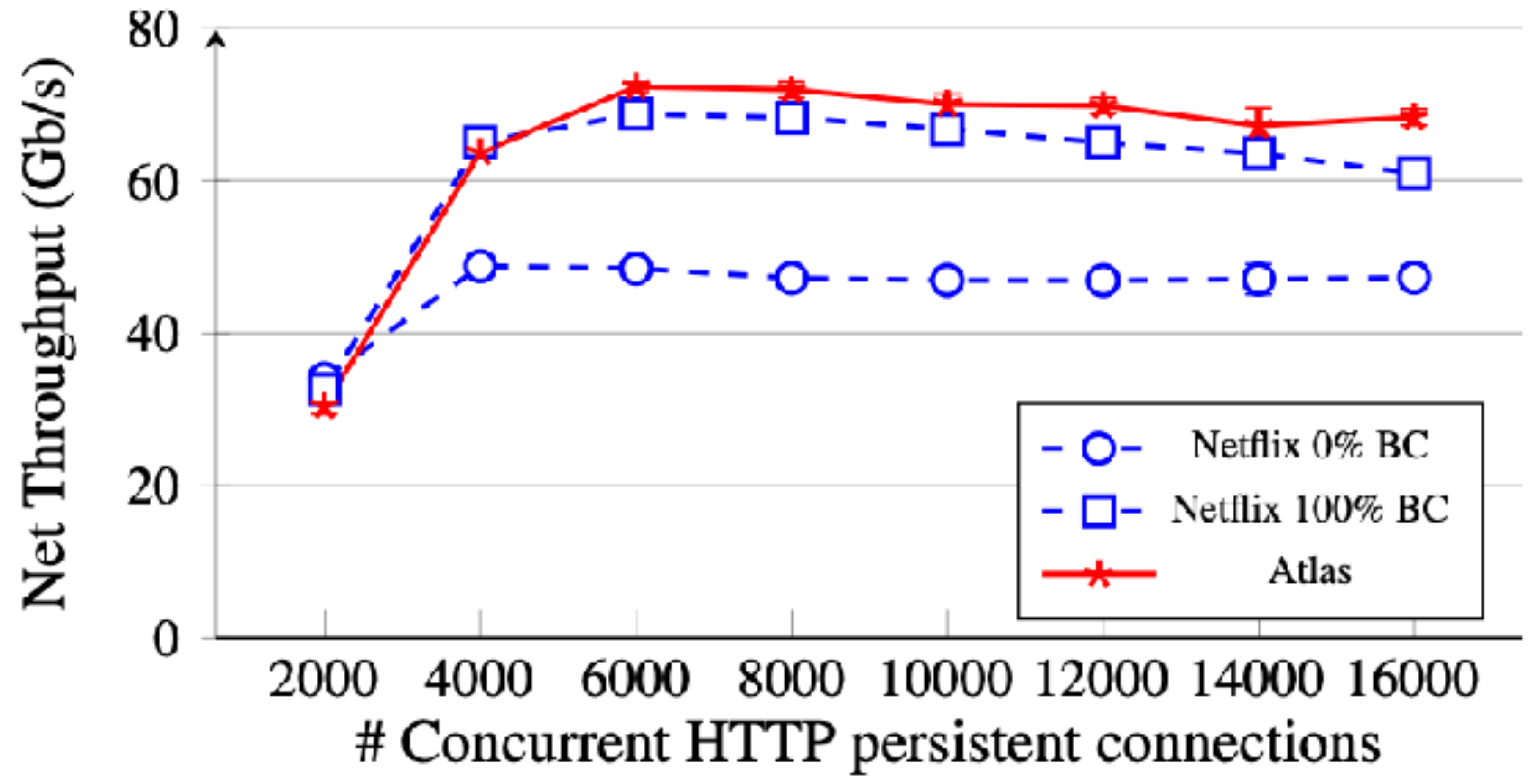
(a) **Delayed notifications incur extra memory writes.**



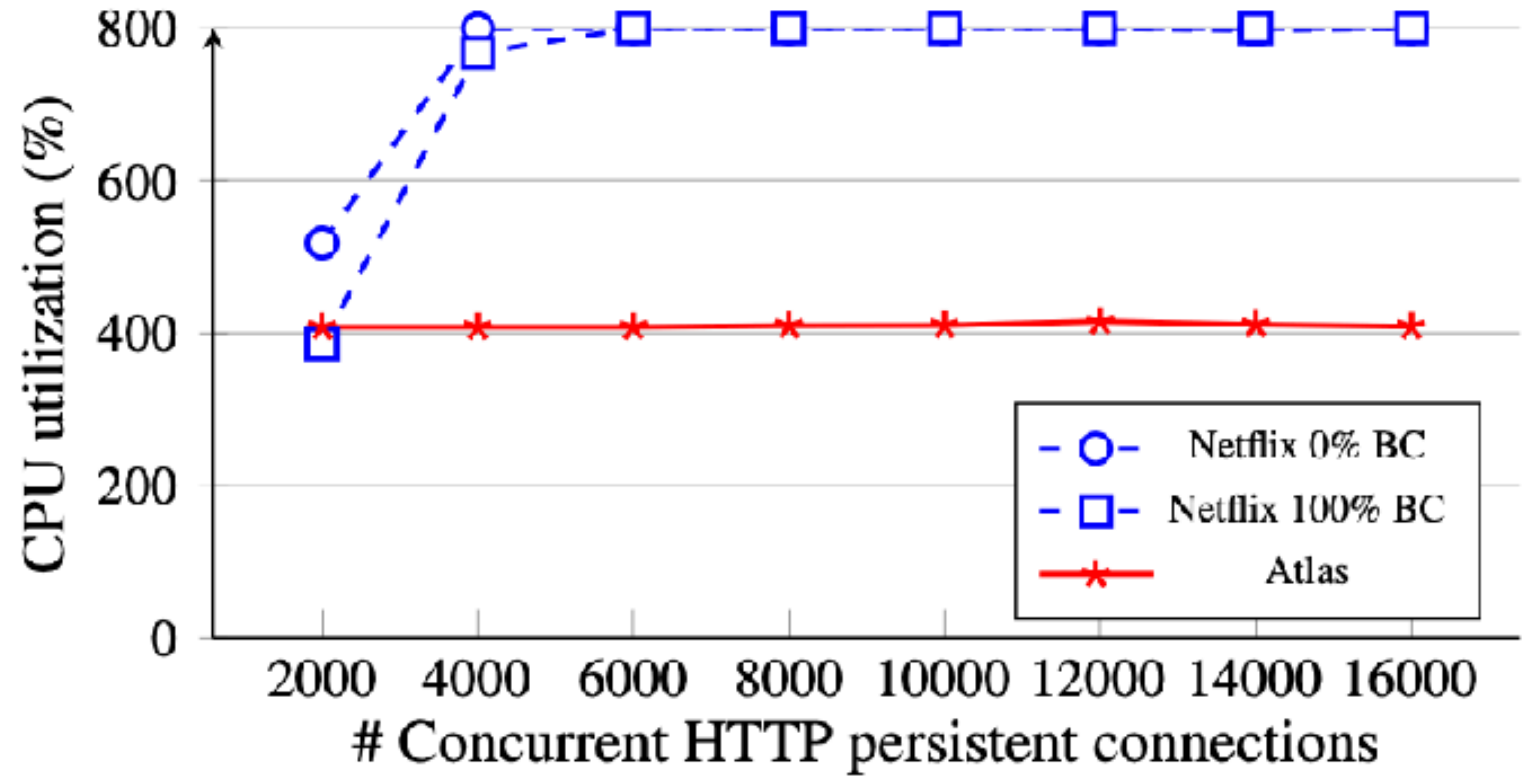
(b) **Heavy load and netmap latency result in LLC evictions.**

Atlas Performance under Encrypted Text

- Higher throughput with fewer CPU cores



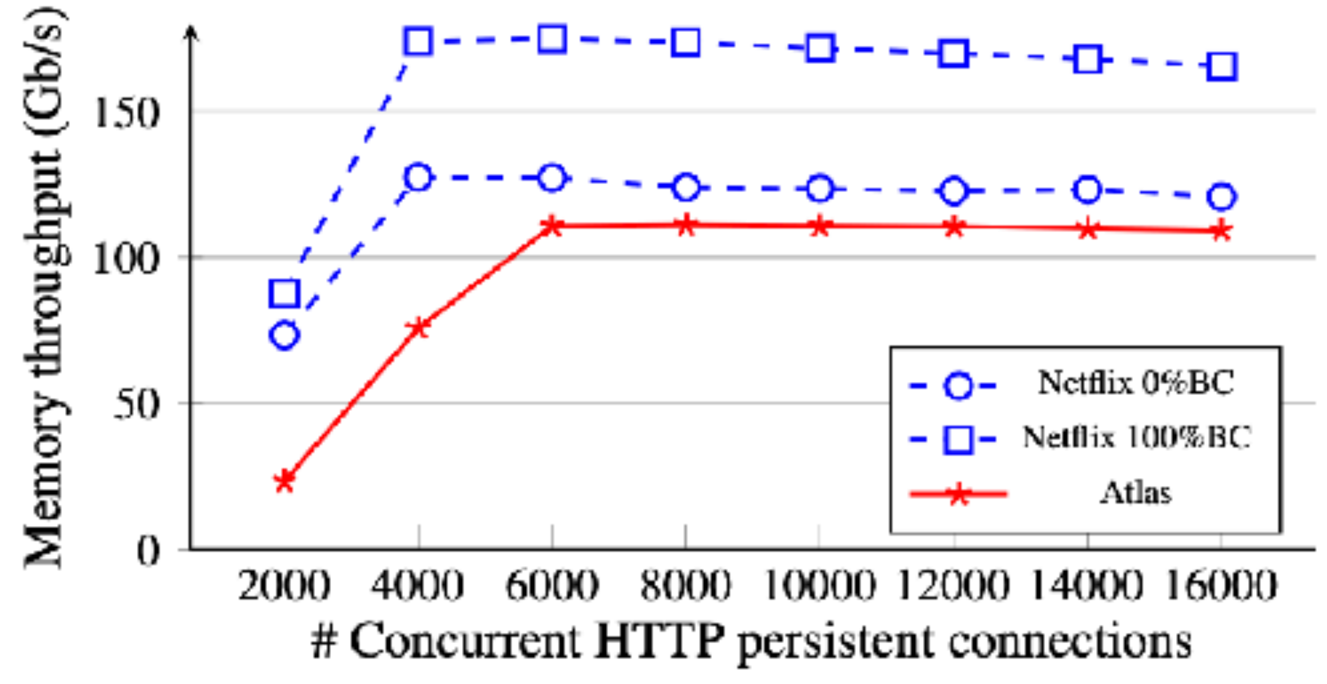
(a) Network throughput (Error bars indicate the 95% CI)



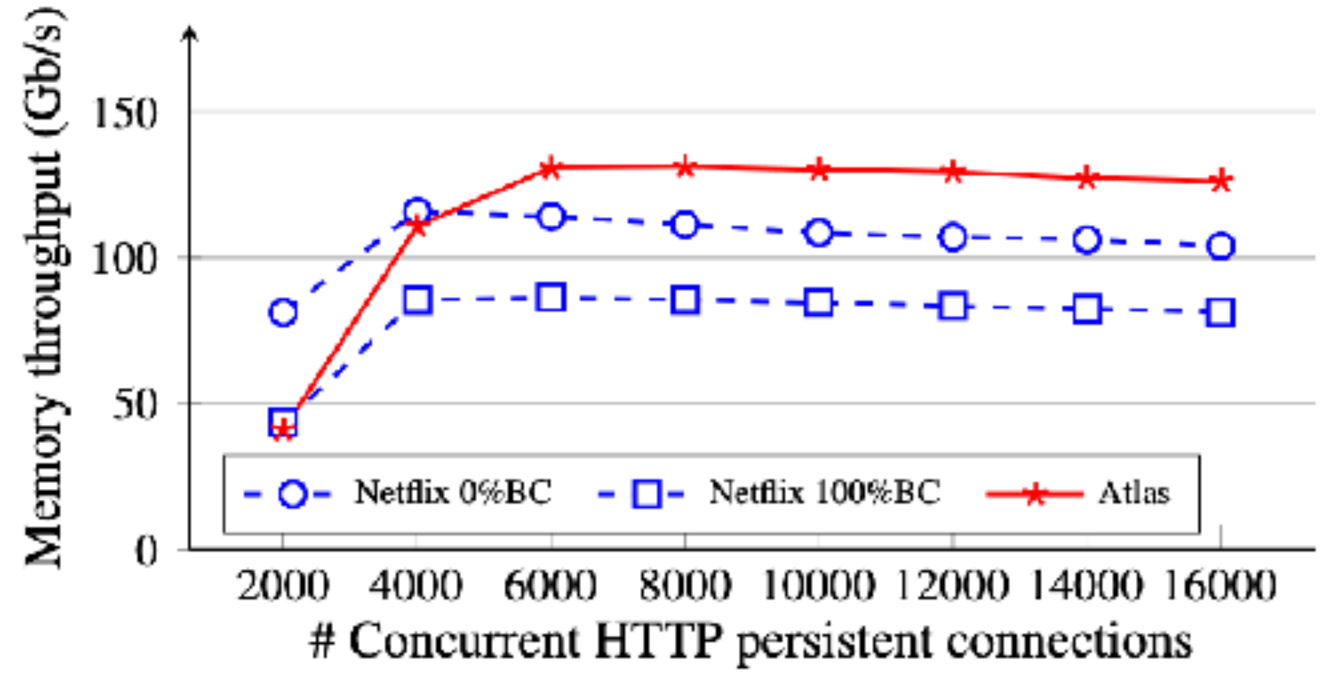
(b) CPU utilization (Average)

Atlas Performance under Encrypted Text (Cont'd)

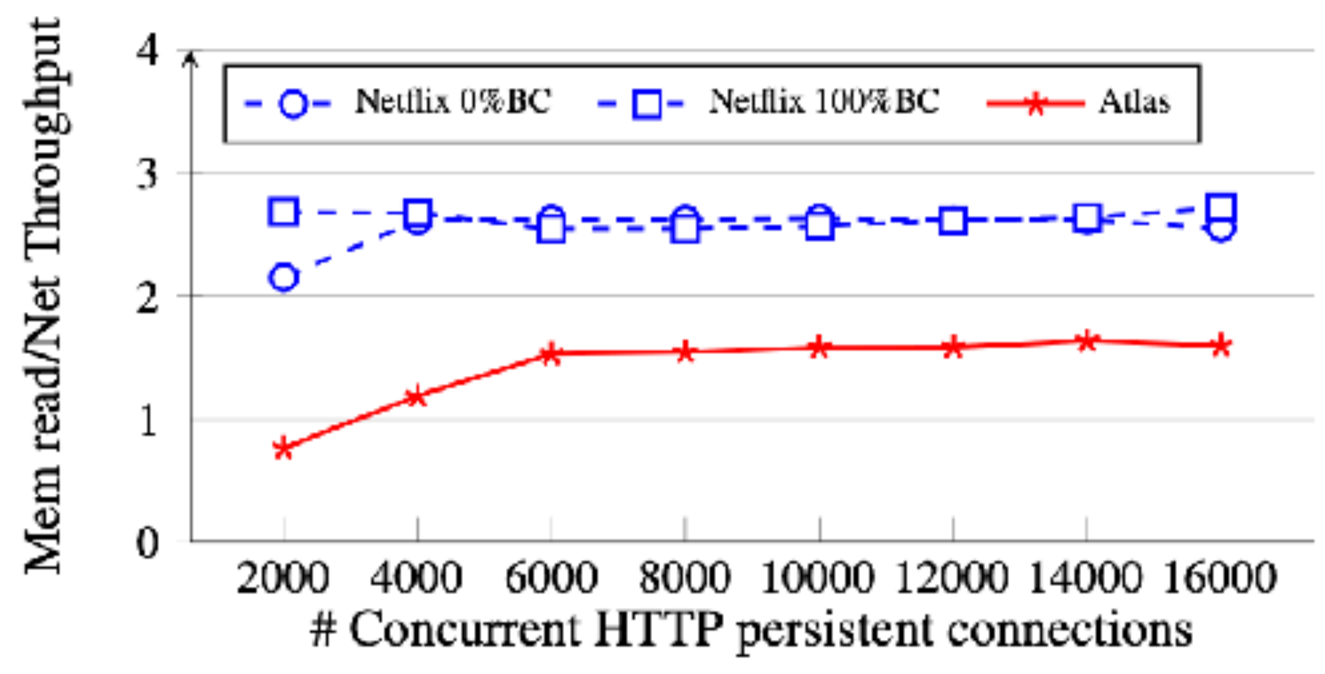
- Fewer memory accesses



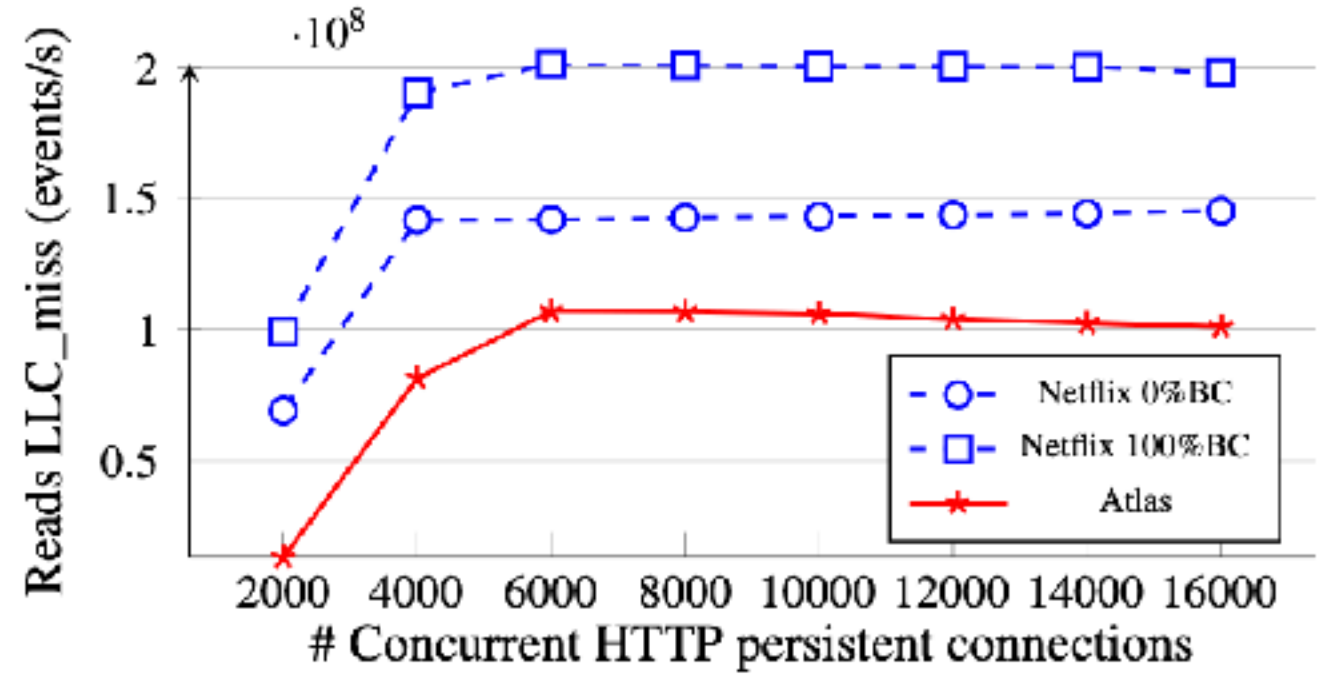
(c) Memory READ



(d) Memory WRITE



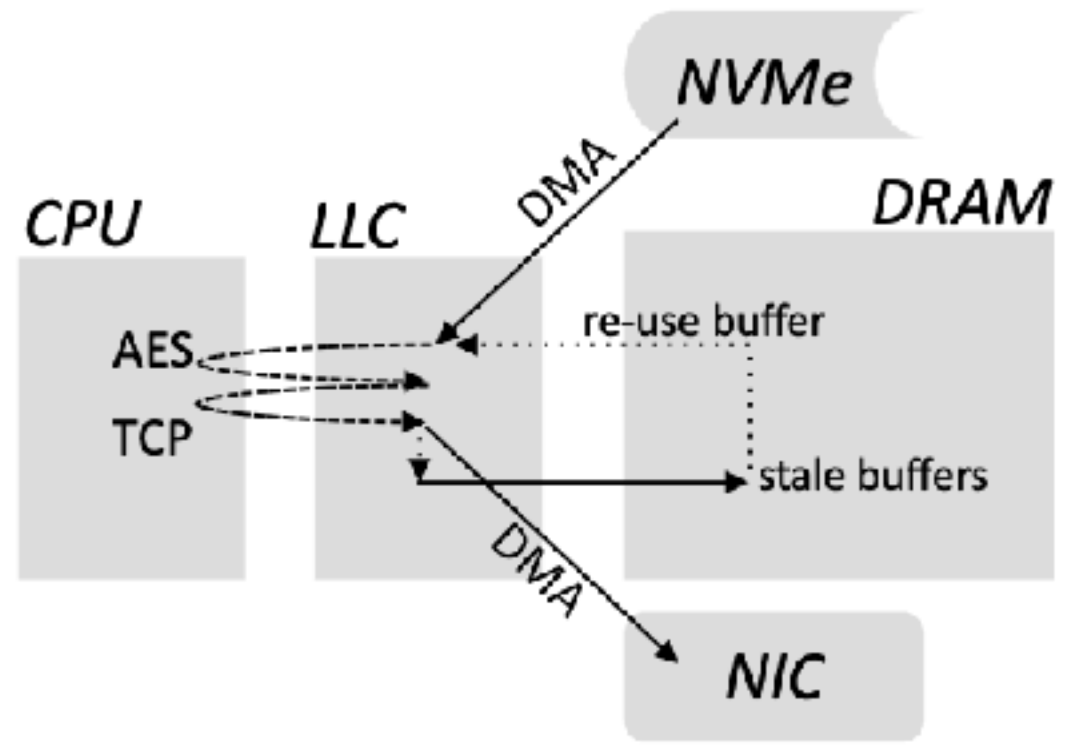
(e) Memory READ-Network Throughput Ratio



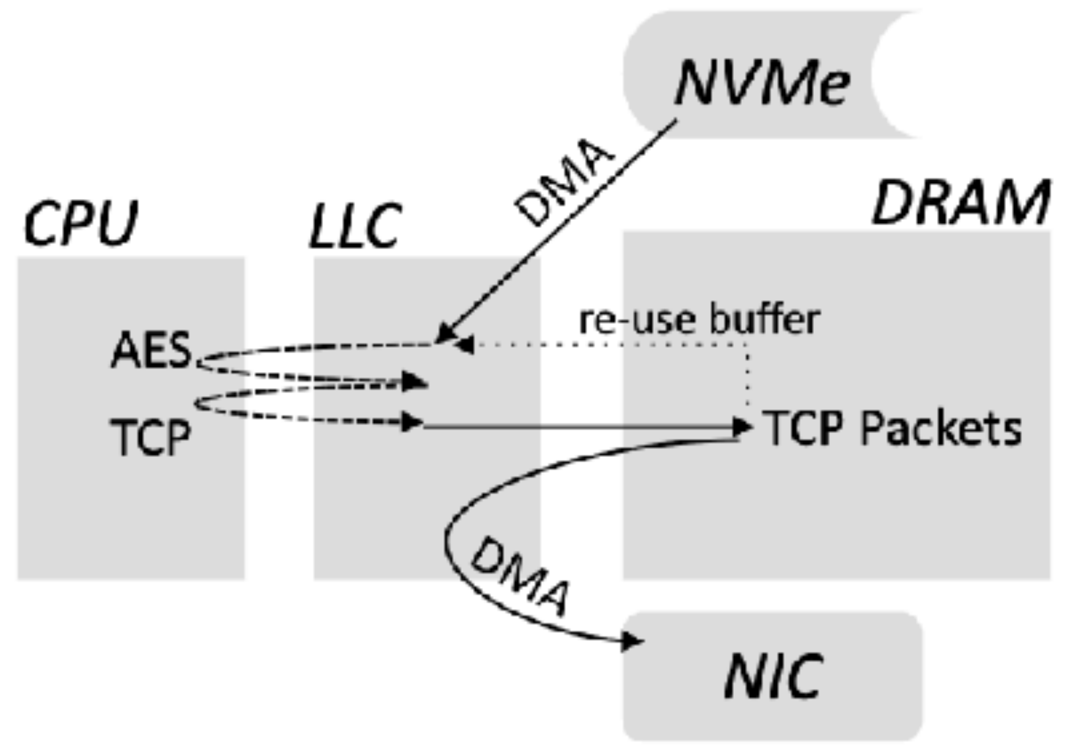
(f) CPU reads served from DRAM due to LLC misses

But, anomalies still happen

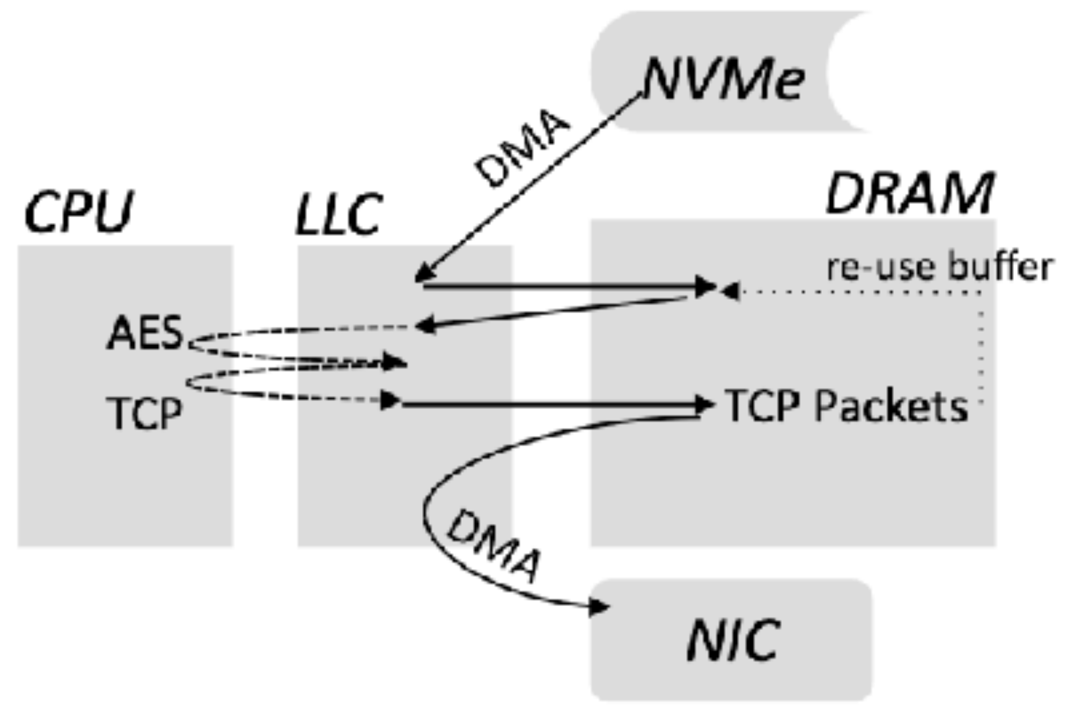
- Delayed notifications incur extra memory writes
- More contentions happen



(a) Delayed notifications incur extra memory writes. No memory reads, but extra writes.



(b) Heavy load and netmap batching result in LLC eviction. One extra memory read and write.



(c) Contention for DDIO portion of LLC evicts DMA'ed data. Two extra reads and writes.

Rethinking Video Streaming

- Hardware trends
  - NVMe SSDs are closer to CPU, let alone future load-store interface
  - DDIO saves memory bandwidth and reduces access latency
  - LLC per processor is nearly stagnated but I/O bandwidths are not

Rethinking Video Streaming

- Hardware trends
    - NVMe SSDs are closer to CPU, let alone future load-store interface
    - DDIO saves memory bandwidth and reduces access latency
    - LLC per processor is nearly stagnated but I/O bandwidths are not
- Software design principles
    - Optimizing LLC for I/Os becomes essential
    - Tight control loops help reduce I/O processing latency
    - Kernel-bypass I/Os are not a panacea, especially considering tail
    - Zero-copy needs to be considered in an end-to-end way

Summary

- Today
  - Atlas (SIGCOMM’17)
- Next
  - QUIC (SIGCOMM’17)