Disk|Crypt|Net

High-performance video streaming

Ilias Marinos, Robert Watson (Cambridge),

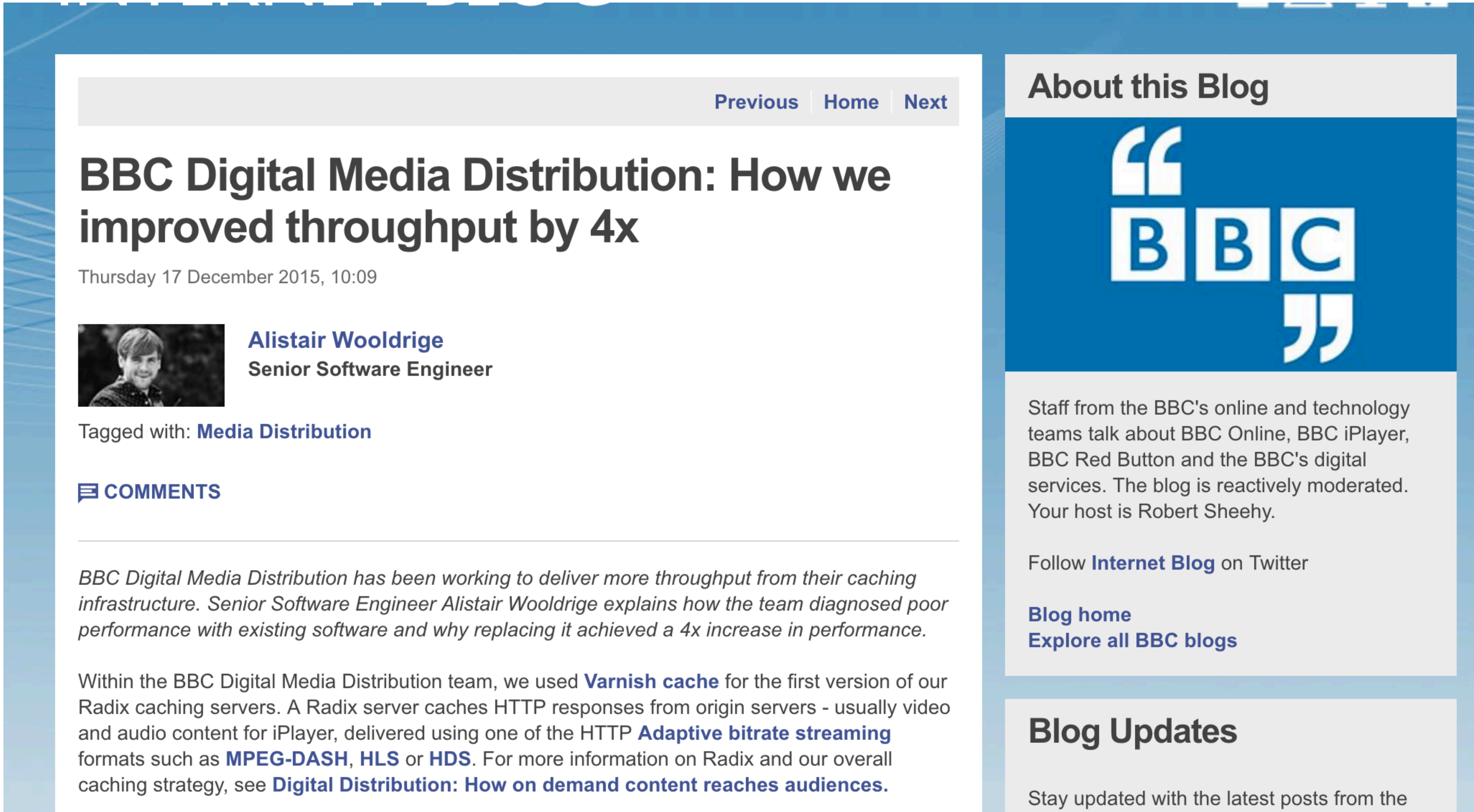
Mark Handley (UCL),

Randall Stewart (Netflix)

Modern Video Streaming

- Just lots of HTTP requests for video chunks.
- Client picks chunks to adapt rate.
- Server is pretty dumb – just has to go fast.
- HTTP/1.1 persistent connections.
- TLS becoming important (95% of Youtube traffic).

- More than 50% of US Internet traffic.
- Important to make good use of expensive hardware. How fast can you go?



Previous | Home | Next

BBC Digital Media Distribution: How we improved throughput by 4x

Thursday 17 December 2015, 10:09

Alistair Wooldrige
Senior Software Engineer

Tagged with: **Media Distribution**

COMMENTS

BBC Digital Media Distribution has been working to deliver more throughput from their caching infrastructure. Senior Software Engineer Alistair Wooldrige explains how the team diagnosed poor performance with existing software and why replacing it achieved a 4x increase in performance.

Within the BBC Digital Media Distribution team, we used **Varnish cache** for the first version of our Radix caching servers. A Radix server caches HTTP responses from origin servers - usually video and audio content for iPlayer, delivered using one of the HTTP **Adaptive bitrate streaming** formats such as **MPEG-DASH**, **HLS** or **HDS**. For more information on Radix and our overall caching strategy, see **Digital Distribution: How on demand content reaches audiences.**

About this Blog

Staff from the BBC's online and technology teams talk about BBC Online, BBC iPlayer, BBC Red Button and the BBC's digital services. The blog is reactively moderated. Your host is Robert Sheehy.

Follow **Internet Blog** on Twitter

Blog home
Explore all BBC blogs

Blog Updates

Stay updated with the latest posts from the

New iPlayer setup, Dec 2015:

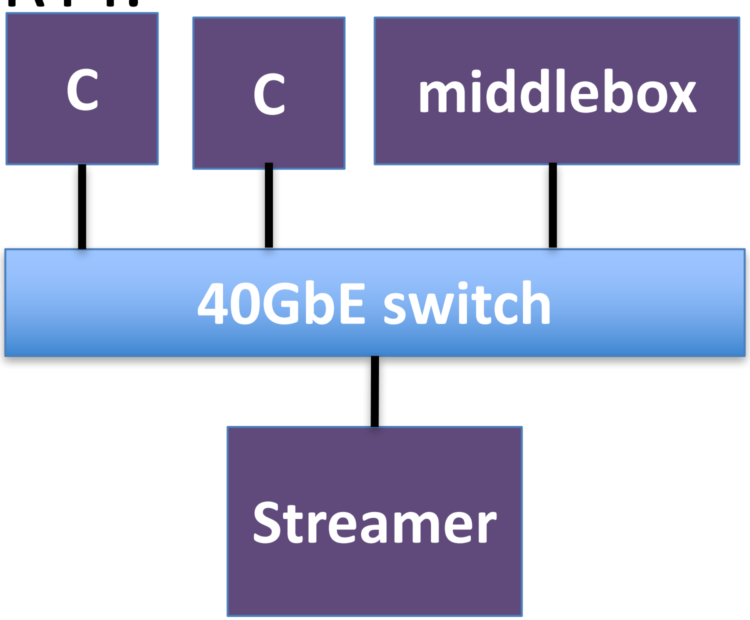
- nginx on Linux, 24 cores on two Intel Xeon E5-2680v3 processors, 512 GB DDR4 RAM, 8.6TB RAID array of SSDs.
- 20Gb/s per server. ←Can we improve performance?

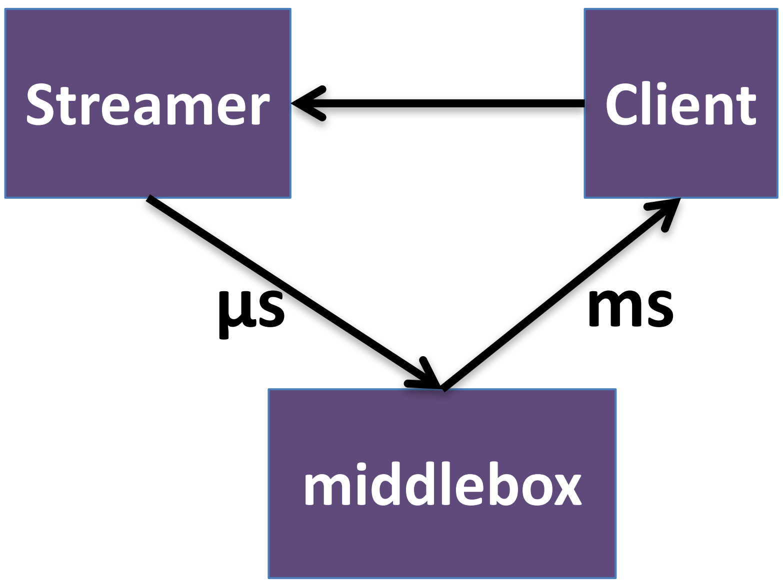
Case study: Netflix

- FreeBSD, but tweaked.
  - Asynchronous `sendfile()`
    - Non-blocking zero copy from disk buffer cache to Net.
  - VM scaling
    - Fake NUMA domains to avoid lock contention.
    - Proactive cleanup of disk buffer cache.
  - RSS-assisted LRO.
    - Sort incoming packets to buckets based on 5-tuple hash to optimize LRO engine efficacy.

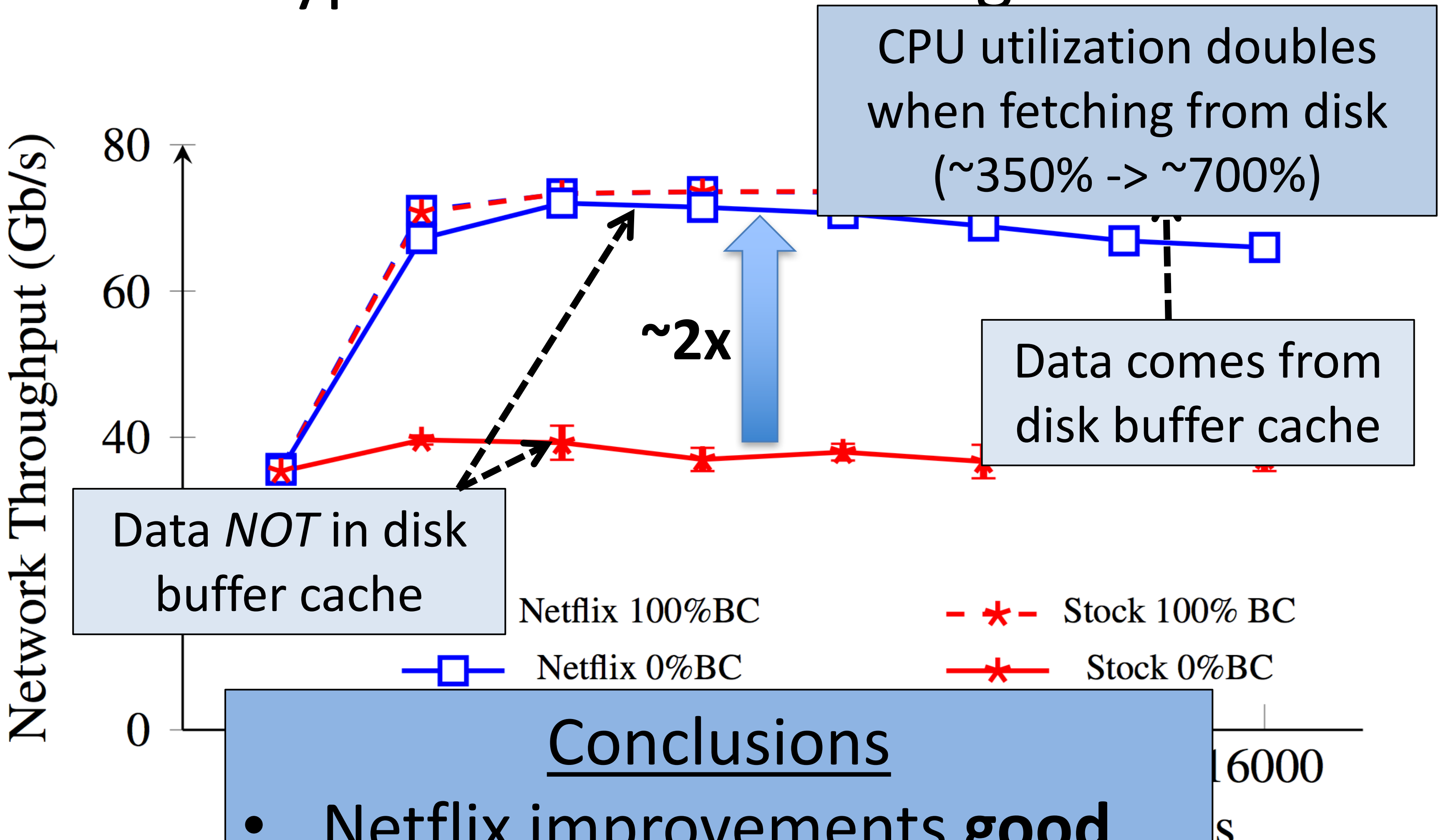
Lets Do Some Experiments

- 8-core Haswell server, 2x40GbE NICs, 128GB RAM, 4x Intel P3700 NVMe disks
- Linux Clients.
- Synthetic workload, middlebox for realistic RTT.





Unencrypted video streaming workload



Conclusions

- Netflix improvements **good**
- CPU utilization is a **problem**

Encryption

Problem:

Sendfile:

- Zero copy from disk buffer cache.

TLS:

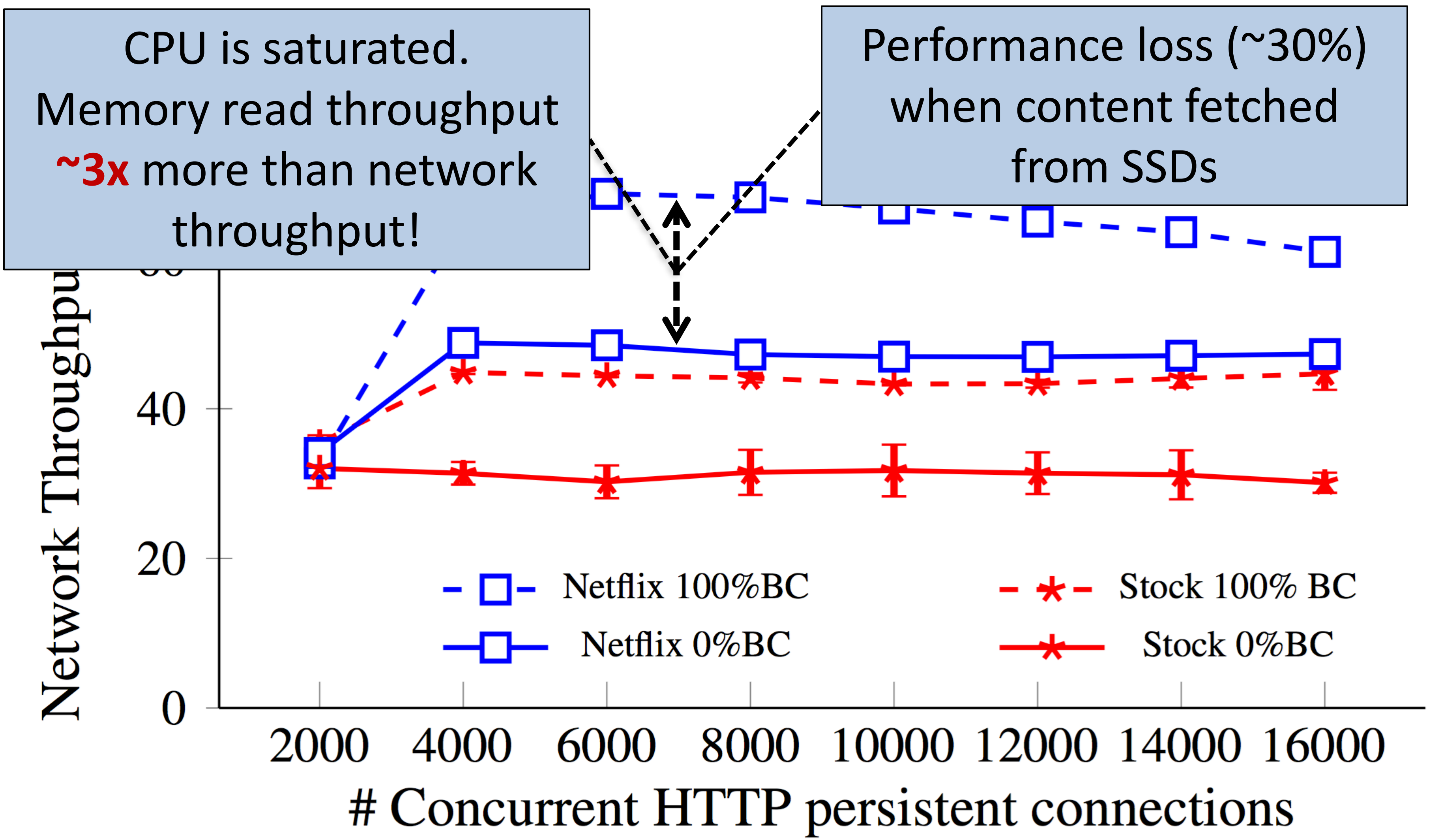
- Different encrypted stream per user.
- Kernel is unaware of TLS.

Sendfile and TLS are fundamentally incompatible!

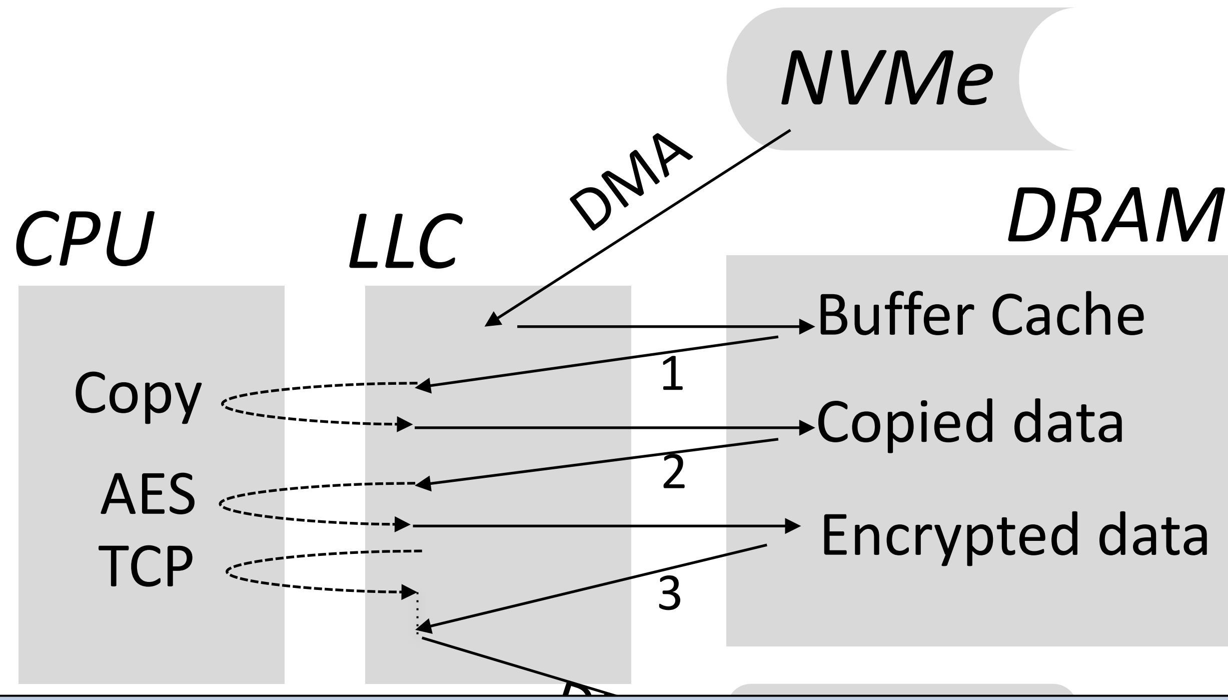
- Conventional TLS stack gave Netflix 20 -> 8.5Gb/s
- Netflix implemented in-kernel TLS support for sendfile!

`sendfile()` **NOT** zerocopy anymore!

Encrypted video streaming workload



What's happening?



The stack is too asynchronous.
Data keeps getting flushed from the LLC, and re-loaded.
System is bottlenecked on memory.

Production Netflix Workload

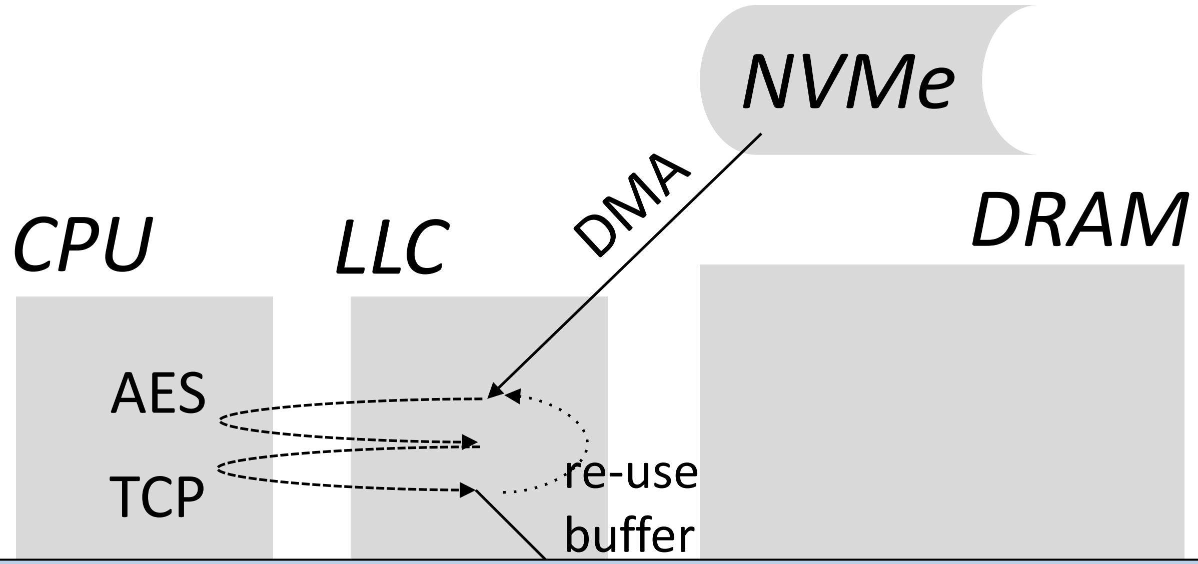
- 192GB for buffer cache, but only 10% hit ratio.
- Streamers bottlenecked in memory bandwidth.

✓ Modern NVMe SSDs have low latency & high throughput.

✓ Modern Intel CPUs DMA directly to L3 cache.

Can we eliminate the disk buffer cache completely, and fetch everything from the SSDs **on-demand**?

Ideal Stack



To achieve this, we must:

- Fetch on demand from the SSD when TCP needs data.
- As soon as the SSD returns data, process it to completion and DMA it to the NIC.

Solution Outline

1. A TCP ACK arrives, freeing up congestion window.
2. Trigger stack to request more data from SSDs to fill that congestion window.

Conventional OS stack NOT suitable:

- Highly asynchronous; storage and network stack are loosely coupled -- relies on VFS & Buffer Cache.
- Introduces overheads related to abstraction layers (VFS, POSIX etc), redundant memory copies and domain transitions (user<->kernel).

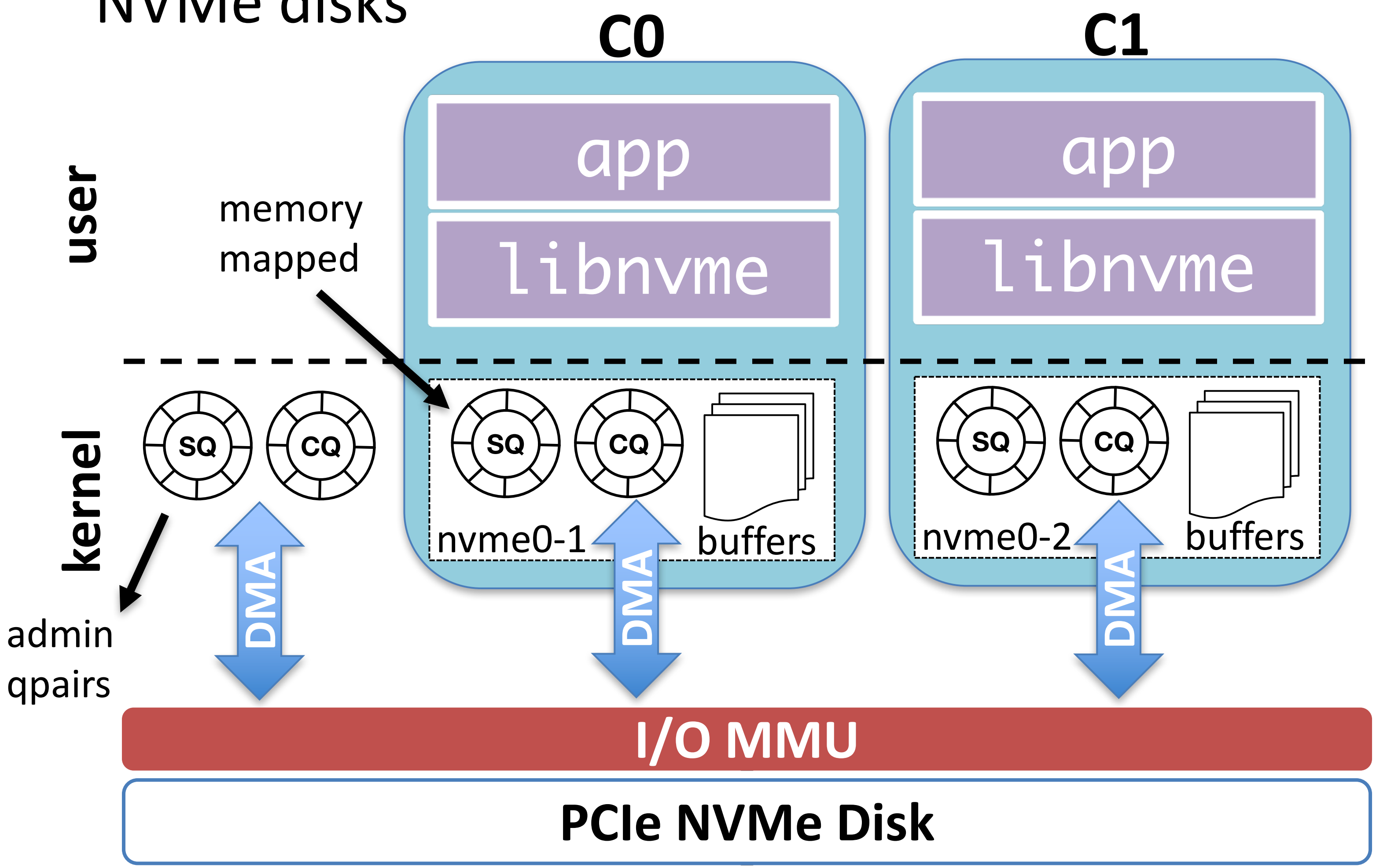
The Atlas Streaming Stack

***Atlas*: a complete user-space stack**

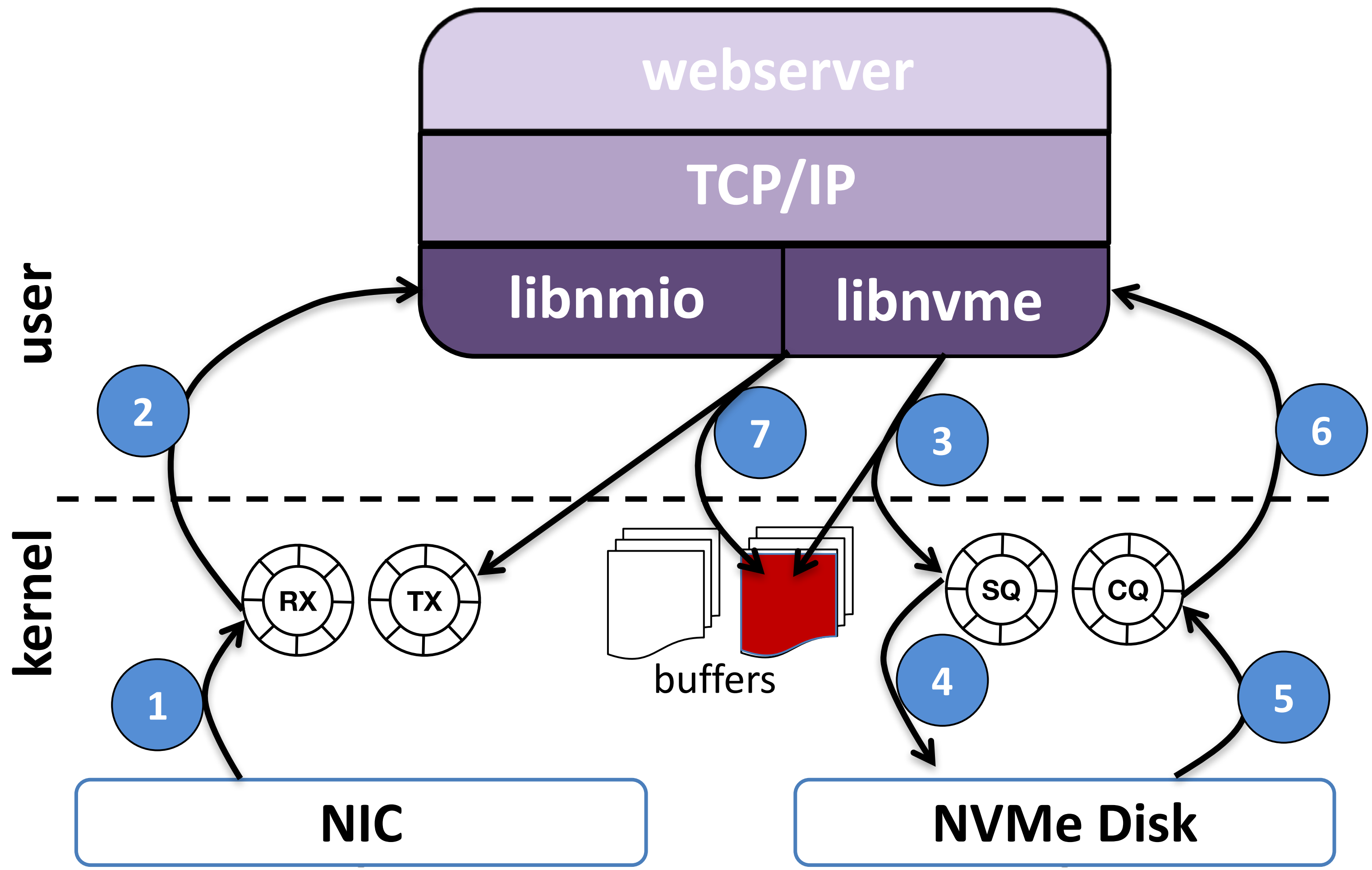
- TCP/IP stack based on modified version of ***Sandstorm*** *(SIGCOMM'14)* and ***netmap*** *(ATC'12)*.
- Storage handled using ***diskmap*** *(*no buffer cache, no sophisticated FS*)*.
- *Lockless,* full *zero-copy* stack from disk<->NIC.
- Tight pipeline to reduce asynchrony, and ideally save memory bandwidth (w/ DDIO).

Diskmap Architecture

Diskmap: a kernel-bypass I/O framework for NVMe disks



The Atlas Execution Pipeline

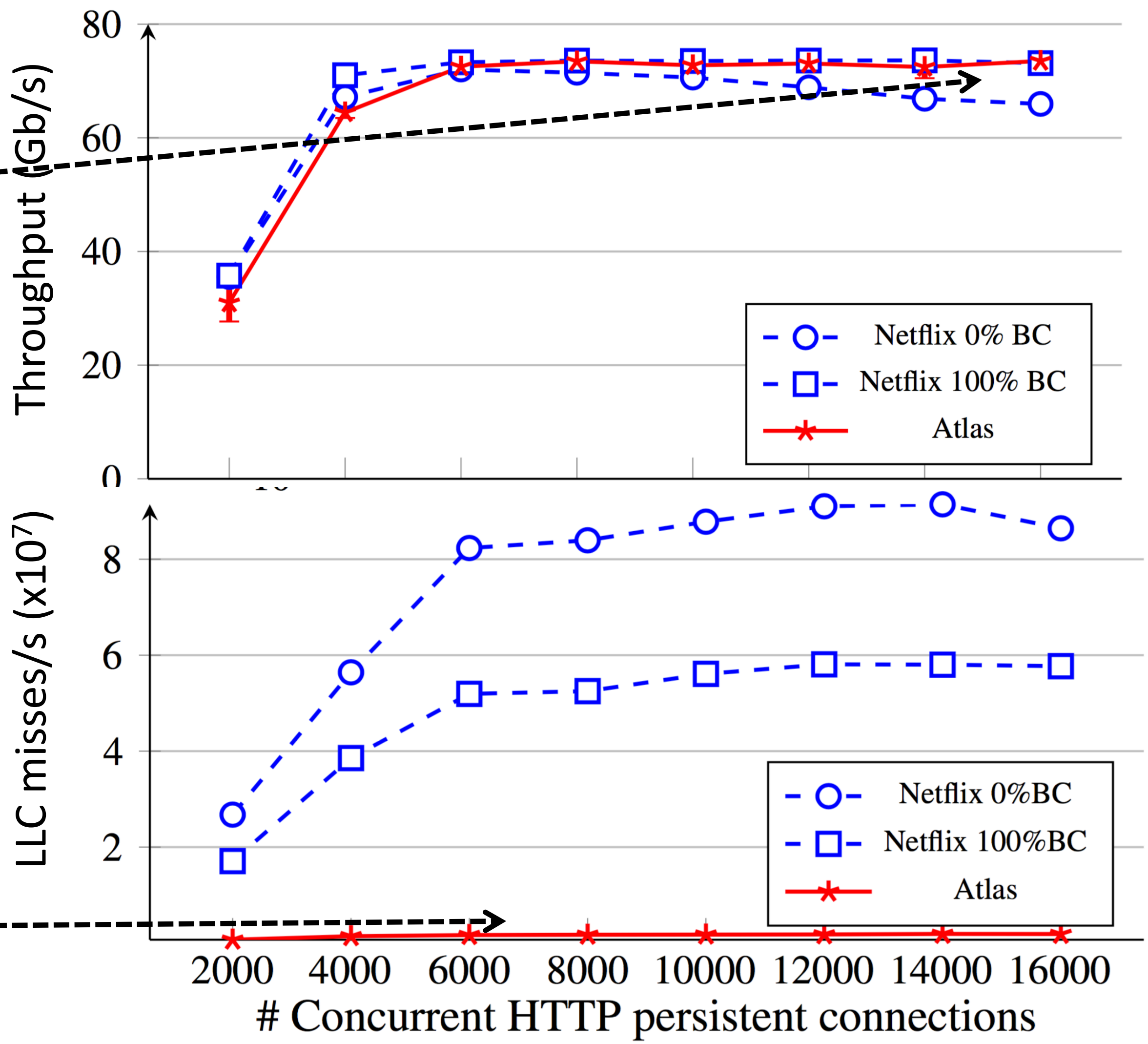


Atlas vs. Netflix, Unencrypted Content

15% better throughput than Netflix when cache hit ratio is low.

Netflix needs 8 cores, Atlas only needs 4

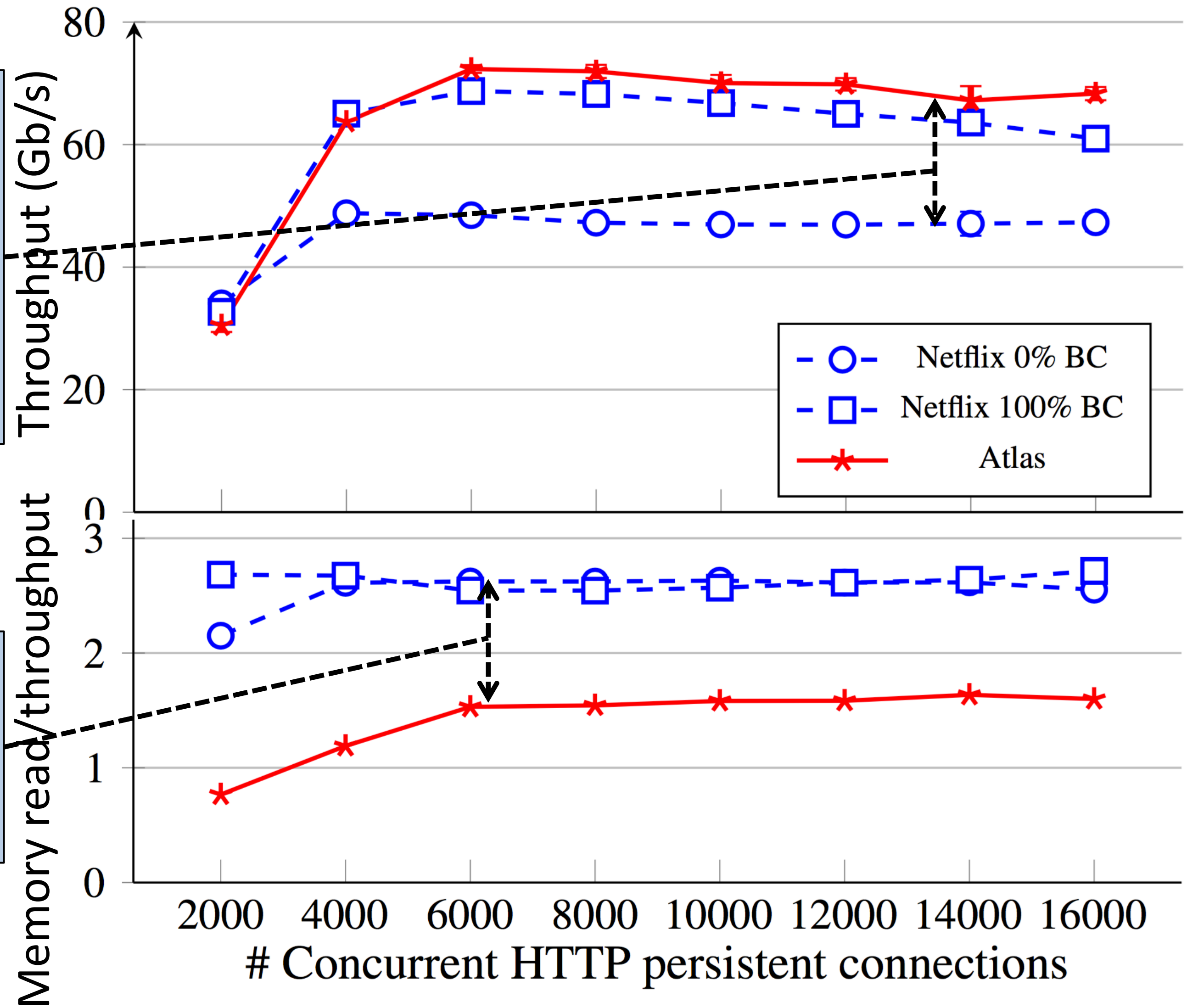
Almost no CPU stalls: data in LLC when we want it.



Atlas vs. Netflix, Encrypted Content

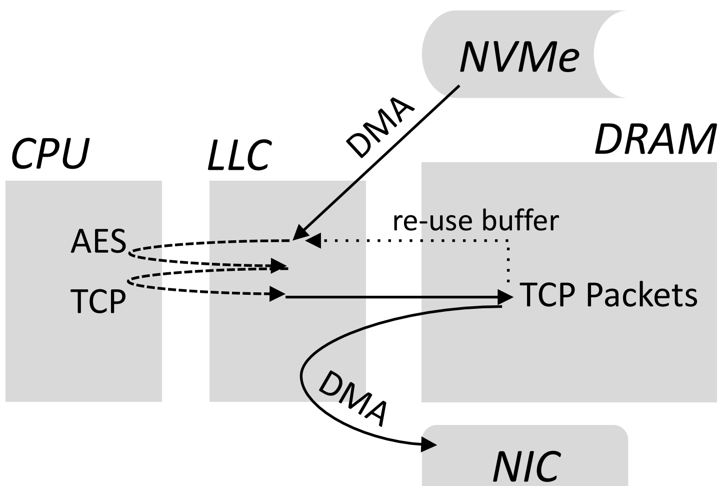
When cache hit ratio is low, 50% more throughput using half the cores.

Almost half the memory reads for each packet sent.

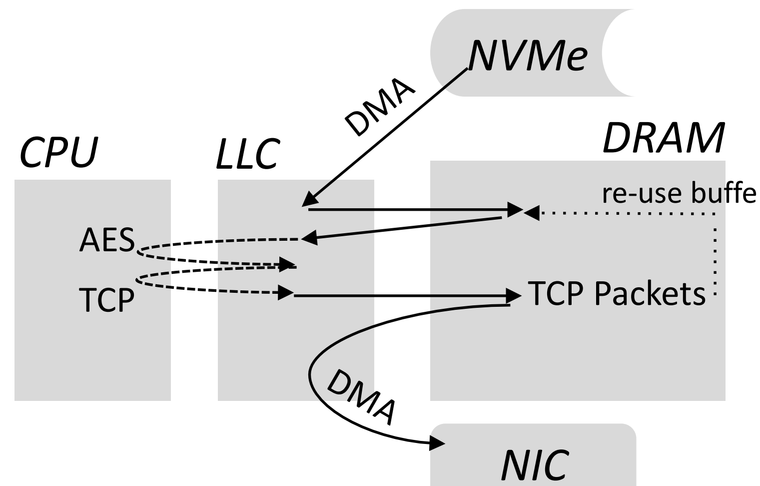


Atlas memory usage

When LLC/CPU is
<u>NOT saturated</u>:



When LLC/CPU is
<u>saturated</u>:



Netmap doesn’t provide a low-delay fine-grained way to communicate DMA completions.
Can’t reuse buffers fast enough (no LIFO stack), and this contributes to some extra cache pressure.

Summary

- Netflix addressed all the low-hanging fruit
  - Very fast, but now **bottlenecked on memory**
- Atlas is a specialized stack
  - Puts SSD directly in TCP control loop
  - Immediately processes disk reads to completion and transmits.
  - 50% throughput improvement with encrypted content, close to 50% reduction in memory reads
- Netflix inspired by Atlas
  - Now experimenting with how to directly trigger encryption off of disk DMA completions in their FreeBSD stack.