Serving Netflix Video Traffic at 400Gb/s and Beyond

Drew Gallatin
NAB Show, April 2022
Serving Netflix Video Traffic at 800Gb/s and Beyond

Drew Gallatin
NAB Show, April 2022
Netflix Workload:

- Serve only static media files
- Pre-encoded for all codecs/bitrates
  - Video quality is of the utmost importance, so we don’t transcode on the server
- Greatly simplifies server workload
Netflix Video Serving Stack

- FreeBSD-current
- NGINX web server
- Video served via asynchronous sendfile(2) and encrypted using kTLS
Timeline:

- Asynchronous Sendfile (2014)
- Kernel TLS (2016)
- Network-centric NUMA (2019)
- Inline Hardware (NIC) kTLS (2022)
- 800G initial results
Since we are serving static files, we can use sendfile(2)

Sendfile directs the kernel to send data from a file descriptor to a TCP socket

This eliminates the need to copy data into or out of the kernel
Netflix Video Serving Data Flow

Using sendfile, data is sent directly from disk to network and not touched by the host CPU.
Problem: Disk reads can block sendfile

- When an nginx worker is blocked, it cannot service other requests
- Solutions to prevent nginx from blocking like aio or thread pools scale poorly
Solution: Asynchronous sendfile

- sendfile() becomes “fire and forget”
- Empty buffers are appended to the TCP socket buffer. TCP stops when it sees an empty buffer.
- When disk read completes, disk interrupt handler informs TCP it is ready to send
Asynchronous sendfile

Socket Buffer

Disks

Internet
Asynchronous sendfile

Disks → Socket Buffer → Internet
Asynchronous sendfile
Asynchronous sendfile

Disks → Socket Buffer → Internet
Asynchronous sendfile

Disks -> Socket Buffer -> Internet
Asynchronous sendfile

Socket Buffer

Disks

Internet
Asynchronous sendfile

Socket Buffer

Disks

Internet
Asynchronous Sendfile Performance

- **Intel Xeon E5-2697v2**
  - 12 cores @ 2.7GHz
  - 256GB DDR3-800
  - Chelsio T580 40GbE
- **23Gbs -> 36Gb/s**
Timeline:

- Asynchronous Sendfile (2014)
- **Kernel TLS (2016)**
- Network-centric NUMA (2019)
- Inline Hardware (NIC) kTLS (2022)
- 800G initial results
What’s TLS?

- Transport *Layer* *Security*
- **TLS** encrypts traffic between clients and the OCA
TLS Prevents Sendfile & Triples Memory BW

Data is touched by CPU:
1. Copy from kernel to userspace
2. Read data to encrypt
3. Write encrypted data to memory
4. Copy from userspace to kernel
Solution: Move TLS into the kernel

- Eliminates copies between userspace and kernel
- Restores sendfile dataflow
- TLS handshakes (e.g., session setup, session resumption) handled in userspace.
- TLS state handed to the kernel
- Kernel does bulk crypto as part of sendfile pipeline
Asynchronous sendfile + kTLS

Disks

Socket Buffer

Internet
Asynchronous sendfile
Asynchronous sendfile + kTLS
Asynchronous sendfile + kTLS
Asynchronous sendfile + kTLS
Asynchronous sendfile + kTLS

Socket Buffer

Disks

Internet

NetfliX

NGINX
Asynchronous sendfile + kTLS

Socket Buffer

Disks

Internet
Netflix 800Gb/s Video Serving Data Flow

Using sendfile and software kTLS, data is encrypted by the host CPU.

800Gb/s == 100GB/s

~400GB/sec of memory bandwidth is needed to serve 800Gb/s
Timeline:

- Asynchronous Sendfile (2014)
- Kernel TLS (2016)
- **NUMA (2019)**
- Inline Hardware (NIC) kTLS (2022)
- 800G initial results
What is NUMA?

Non Uniform Memory Architecture

That means memory and/or devices can be “closer” to some CPU cores
Multi CPU Before NUMA

Memory access was UNIFORM:
Each core had equal and direct access to all memory and IO devices.
Multi Socket system with NUMA:

Memory access can be **NON-UNIFORM**

- Each core has unequal access to memory
- Each core has unequal access to I/O devices
Present day NUMA:

Each locality zone called a “NUMA Domain” or “NUMA Node”
Strategy: Keep as much of our 400GB/sec of bulk data off the NUMA fabric as possible

- Bulk data congests NUMA fabric and leads to CPU stalls.
Dual AMD: Worst Case Data Flow

Steps to send data:
Dual AMD: Worst Case Data Flow

Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
Dual AMD: Worst Case Data Flow

Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
- CPU reads data for encryption
  - Second NUMA crossing
Dual AMD: Worst Case Data Flow

Steps to send data:

- DMA data from disk to memory
  - First NUMA bus crossing
- CPU reads data for encryption
  - Second NUMA crossing
- CPU writes encrypted data
  - Third NUMA crossing
Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
- CPU reads data for encryption
  - Second NUMA crossing
- CPU writes encrypted data
  - Third NUMA crossing
- DMA from memory to network
  - Fourth NUMA crossing
Worst Case Summary:

- 4 NUMA crossings
- 400GB/s of data on the NUMA fabric
  - Fabric saturates, cannot handle the load.
  - CPU Stalls, saturates early
Steps to send data:
Steps to send data:

- DMA data from disk to memory
Steps to send data:
- DMA data from disk to memory
- CPU Reads data for encryption
Dual AMD: Best Case Data Flow

Steps to send data:
- DMA data from disk to memory
- CPU Reads data for encryption
- CPU Writes encrypted data
Dual AMD: Best Case Data Flow

Steps to send data:
- DMA data from disk to memory
- CPU Reads data for encryption
- CPU Writes encrypted data
- DMA from memory to Network

0 NUMA crossings!
Best Case Summary:

- 0 NUMA crossings
- 0GB/s of data on the NUMA fabric
Impose order on the chaos.. somehow:

- Disk centric siloing
  - Try to do everything on the NUMA node where the content is stored
- Network centric siloing
  - Try to do as much as we can on the NUMA node that the LACP partner chose for us
Dual AMD: Worst Case Data Flow With Network Centric NUMA Siloing

Steps to send data:
Dual AMD: Worst Case Data Flow With Network Centric NUMA Siloing

Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
Dual AMD: Worst Case Data Flow With Network Centric NUMA Siloing

Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
- CPU Reads data for encryption
Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
- CPU Reads data for encryption
- CPU Writes encrypted data
Dual AMD: Worst Case Data Flow With Network Centric NUMA Siloing

Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
- CPU Reads data for encryption
- CPU Writes encrypted data
- DMA from memory to Network
Worst Case Summary:

- 1 NUMA crossing on average
  - 100% of disk reads across NUMA
- 100GB/s of data on the NUMA fabric
  - Still less than fabric bandwidth
Dual AMD: Worst Case Data Flow With Disk Centric NUMA Siloing

Steps to send data:
Dual AMD: Worst Case Data Flow With Disk Centric NUMA Siloing

Steps to send data:
- DMA data from disk to memory
Dual AMD: Worst Case Data Flow With Disk Centric NUMA Siloing

Steps to send data:
- DMA data from disk to memory
- CPU Reads data for encryption
Dual AMD: Worst Case Data Flow With Disk Centric NUMA Siloing

Steps to send data:
- DMA data from disk to memory
- CPU Reads data for encryption
- CPU Writes encrypted data
Steps to send data:

- DMA data from disk to memory
- CPU Reads data for encryption
- CPU Writes encrypted data
- NIC Reads data for transmit
  - First NUMA bus crossing
Worst Case Summary:

- 1 NUMA crossing on average
  - 100% of disk reads across NUMA
- 100GB/s of data on the NUMA fabric
  - Less than theoretical fabric bandwidth
Dual AMD: Worst Case Data Flow With Strict Disk Centric NUMA Siloing

Steps to send data:
Steps to send data:
- DMA data from disk to memory
Steps to send data:
- DMA data from disk to memory
- CPU Reads data for encryption
Dual AMD: Worst Case Data Flow With Strict Disk Centric NUMA Siloing

Steps to send data:
- DMA data from disk to memory
- CPU Reads data for encryption
- CPU Writes encrypted data
Steps to send data:
- DMA data from disk to memory
- CPU Reads data for encryption
- CPU Writes encrypted data
- NIC Reads data for transmit
Worst Case Summary:

- 0 NUMA crossing on average
  - 0% of disk reads across NUMA
- 0GB/s of data on the NUMA fabric
Timeline:

- Asynchronous Sendfile (2014)
- Kernel TLS (2016)
- NUMA (2019)
- **Inline Hardware (NIC) kTLS (2022)**
- 800G initial results
Why offload kTLS?

- kTLS uses almost half of our CPU cycles
Netflix 800Gb/s Video Serving Data Flow

Using sendfile and software kTLS, data is encrypted by the host CPU.

800Gb/s == 100GB/s

~400GB/sec of memory bandwidth is needed to serve 800Gb/s
Mellanox (NVIDIA) NIC kTLS

- Discussed with Mellanox starting in 2016
- First prototypes of CX6-DX in early 2020
- Iterated for 2+ years to make it production ready
- kTLS offload enabled in production last quarter
What is NIC kTLS?:

- Hardware Inline TLS
- TLS session is established in userspace.
- When crypto is moved to the kernel, the kernel passes crypto keys to the NIC
- TLS records are encrypted by NIC as the data flows through it on transmit
  - No more detour through the CPU for crypto
  - This cuts memory BW & CPU use in half!
Netflix 800Gb/s Video Serving Data Flow

Using sendfile and software kTLS, data is encrypted by the host CPU.

800Gb/s == 100GB/s

~400GB/sec of memory bandwidth is needed to serve 800Gb/s
Netflix 800Gb/s Video Serving Data Flow

Using sendfile and software kTLS, data is encrypted by the host CPU.

800Gb/s == 100GB/s

~400GB/sec of memory bandwidth is needed to serve 800Gb/s
Netflix 800Gb/s Video Serving Data Flow

Using sendfile and NIC kTLS, data is encrypted by the NIC.

800Gb/s == 100GB/s

~200GB/sec of memory bandwidth is needed to serve 800Gb/s
Mellanox ConnectX-6 Dx

- Offloads TLS 1.2 and 1.3 for AES GCM cipher
- Retains crypto state within a TLS record
  - Means that the TCP stack can send partial TLS records without performance loss
- If a packet is sent out of order (e.g., a TCP retransmit), it must re-DMA the record containing the out of order packet
CX6-DX: In-order Transmit
### Host Memory

<table>
<thead>
<tr>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>15928</td>
</tr>
<tr>
<td>14480</td>
</tr>
<tr>
<td>13032</td>
</tr>
<tr>
<td>11584</td>
</tr>
<tr>
<td>10136</td>
</tr>
<tr>
<td>8688</td>
</tr>
<tr>
<td>7240</td>
</tr>
<tr>
<td>5792</td>
</tr>
<tr>
<td>4344</td>
</tr>
<tr>
<td>2896</td>
</tr>
<tr>
<td>1448</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

### TCP segments of Plaintext TLS Record

- NIC
- PCIe Bus
- 100GbE Network
### TCP segments of Plaintext TLS Record

<table>
<thead>
<tr>
<th>Segment Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>15928</td>
</tr>
<tr>
<td>14480</td>
</tr>
<tr>
<td>13032</td>
</tr>
<tr>
<td>11584</td>
</tr>
<tr>
<td>10136</td>
</tr>
<tr>
<td>8688</td>
</tr>
<tr>
<td>7240</td>
</tr>
<tr>
<td>5792</td>
</tr>
<tr>
<td>4344</td>
</tr>
<tr>
<td>2896</td>
</tr>
<tr>
<td>1448</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

---

**NIC**

**PCle Bus**

**Host Memory**

**100GbE Network**
TCP segments of Plaintext TLS Record

Host Memory

| 15928 | 14480 | 13032 | 11584 | 10136 | 8688 | 7240 | 5792 | 4344 | 2896 | 1448 | 0 |

TCP segments of Encrypted TLS Record

| 7240 | 5792 | 4344 | 2896 | 1448 | 0 |
TCP segments of Plaintext TLS Record

TCP segments of Encrypted TLS Record
CX6-DX: TCP Retransmit
### TCP segments of Plaintext TLS Record

<table>
<thead>
<tr>
<th>Segment</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15528</td>
</tr>
<tr>
<td></td>
<td>14480</td>
</tr>
<tr>
<td></td>
<td>1332</td>
</tr>
<tr>
<td></td>
<td>11584</td>
</tr>
<tr>
<td></td>
<td>10136</td>
</tr>
<tr>
<td></td>
<td>8688</td>
</tr>
<tr>
<td></td>
<td>7240</td>
</tr>
<tr>
<td></td>
<td>5792</td>
</tr>
<tr>
<td></td>
<td>4344</td>
</tr>
<tr>
<td></td>
<td>2896</td>
</tr>
<tr>
<td></td>
<td>1448</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

### Host Memory

<table>
<thead>
<tr>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>15528</td>
</tr>
<tr>
<td>14480</td>
</tr>
<tr>
<td>1332</td>
</tr>
<tr>
<td>11584</td>
</tr>
<tr>
<td>10136</td>
</tr>
<tr>
<td>8688</td>
</tr>
<tr>
<td>7240</td>
</tr>
<tr>
<td>5792</td>
</tr>
<tr>
<td>4344</td>
</tr>
<tr>
<td>2896</td>
</tr>
<tr>
<td>1448</td>
</tr>
</tbody>
</table>

### NIC

100GbE Network

### PCIe Bus
TCP segments of Plaintext TLS Record

Host Memory

15928 14480 13032 11584 10136 8688 7240 5792 4344 2896 1448 0

TCP segments of Encrypted TLS Record

100GbE Network
Host Memory

| 15928 | 14480 | 13032 | 11584 | 10136 |  8688 |  7240 |  5792 |  4344 |  2896 |  1448 |   0 |

TCP segments of Plaintext TLS Record

TCP segments of Encrypted TLS Record

NIC

PCle Bus
Timeline:

- Asynchronous Sendfile (2014)
- Kernel TLS (2016)
- NUMA (2019)
- Inline Hardware (NIC) kTLS (2022)
- 800G initial results
800G Prototype Details

- Dell R7525
- 2x AMD EPYC 7713 64c / 128t (128c / 256t total)
- 3x xGMI links between sockets
- 512 GB RAM
- 4x Mellanox ConnectX-6 Dx (8x 100GbE ports)
- 16x Intel Gen4 x4 14TB NVME
Initial Results: 420Gb/s

- Ran in 1NPS mode
- Network Siloing mode
- CPUs mostly idle
  - AMD guessed that xGMI was down-linking to x2
  - Set xGMI speed to 18GT/s and forced link width to x16, and disabled dynamic link width management
Results with DLWM forced: 500Gb/s

- Ran in 1NPS mode
- Network Siloing mode
  - NVME data DMA’ed to NIC’s NUMA Node
- xGMI link usage very uneven:
  - 15GB/s, 4GB/s and 2GB/s
  - Turns out that NVME is not evenly distributed by IO Quadrants
  - Even hashing of cross-socket to xGMI depends on evenly distributed IO
How to Improve xGMI Hashing

● Hashing based on device doing DMA
  ○ NVME is very uneven
  ○ NICs are much less uneven
  ○ “Network Siloing” normally does DMA from NVME to remote node, local to NIC

● Flip things, and do DMA from NVME to local buffers
  ○ “Disk centric siloing”

● The NICs are now doing DMA across xGMI
Results with Disk Centric Siloing: 670Gb/s

- Much more even xGMI hashing:
  - 10/10/7 GB/s
- Problematic because:
  - Daemon that “locks” content into memory is not NUMA aware & can lead to page daemon thrashing.
  - Still pressure on xGMI links
Strict Disk Centric Siloing

- Move Egress NIC to be local to NUMA node with disk
  - No bulk data crosses NUMA Bus
- Incoming TCP traffic still uses original NIC
  - Metadata crosses NUMA bus
“Strict Disk Centric Siloing” Results: 720Gb/s

- Much less xGMI traffic
- Limited by NIC output drops, not CPU.
- Cause of drops is now largely due to:
  - Page daemon interfering with nginx on popular node
  - Uneven loading on NICs due to content popularity differences. (NICs on popular node doing 94Gb/s, others doing 84Gb/s)
Credits

● Async Sendfile
  ○ Gleb Smirnoff, Konstantin Belousov, Igor Sysoev, Jeff Roberson, Scott Long

● kTLS
  ○ Scott Long, Randall Stewart, Drew Gallatin, John Mark Gurney, John Baldwin

● NUMA
  ○ Drew Gallatin, Jeff Roberson, Mark Johnston

● Inline Hardware (NIC) kTLS
  ○ John Baldwin, Drew Gallatin, Hans Petter Seleaski, Boris Pismenny, Navdeep Parhar
Credits

- **Experimental 800GbE Host**
  - Warren Harrop and the Netflix hardware team
  - MBX (integrator)
  - AMD (EPYC 7713 CPUs)
  - Dell (PowerEdge R7525)
  - Mellanox/NVIDIA (ConnectX-6 Dx NICS)
  - Intel (P5316 NVME)
Thank you!