Serving Netflix Video at 400Gb/s on FreeBSD

Drew Gallatin
EuroBSDCon 2021
Outline:

● Motivation
● Description of production platform
● Description of workload
● To NUMA or not to NUMA?
● Inline Hardware (NIC) kTLS
● Alternate platforms
Motivation:

- Since 2020, Netflix has been able to serve 200Gb/s of TLS encrypted video traffic from a single server.
- How can we serve ~400Gb/s of video from the same servers?
Netflix Video Serving Workload

- FreeBSD-current
- NGINX web server
- Video served via sendfile(2) and encrypted using software kTLS
Netflix Video Serving Hardware

- **AMD EPYC 7502P (“Rome”)**
  - 32 cores @ 2.5GHz
  - 256GB DDR4-3200
    - 8 channels
    - ~150GB/s mem bw
  - Or ~1.2Tb/s in networking units
- 128 lanes PCIe Gen4
  - ~250GB/s of IO bandwidth
  - Or ~2Tb/s in networking units
Netflix Video Serving Hardware

- 2x Mellanox ConnectX-6 Dx
  - Gen4 x16, 2 full speed 100GbE ports per NIC
  - 4 x 100GbE in total
  - Support for NIC kTLS offload
- 18x WD SN720 NVME
  - 2TB
  - PCIe Gen3 x4
Performance Results:

- 240Gb/s
- Limited by memory BW
  - Determined empirically by using AMDuProfPCM
Netflix 400Gb/s Video Serving Data Flow

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

400Gb/s == 50GB/s

~200GB/sec of memory bandwidth and ~64 PCIe Gen 4 lanes are needed to serve 400Gb/s
Can NUMA get us to 400Gb/s

- Use STREAM benchmark bandwidth as a proxy
  - Single Node: 150GB/s
  - Four Nodes: 175GB/s
What is NUMA?

Non Uniform Memory Architecture

That means memory and/or devices can be “closer” to some CPU cores
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”
4 Node configurations are common on AMD EPYC
Cross-Domain costs

Latency Penalties:

● 12-28ns
Cross-Domain costs

Bandwidth Limit:

- AMD Infinity Fabric
  - ~47GB/s per link
  - ~280GB/s total
Strategy: Keep as much of our 200GB/sec of bulk data off the NUMA fabric is possible

- Bulk data congests NUMA fabric and leads to CPU stalls when competing with normal memory accesses.
4 Nodes, worst case

Steps to send data:
4 Nodes, worst case

Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
4 Nodes, worst case

Steps to send data:

- DMA data from disk to memory
  - First NUMA bus crossing
- CPU reads data for encryption
  - Second NUMA crossing
4 Nodes, worst case

Steps to send data:

- DMA data from disk to memory
  - First NUMA bus crossing
- CPU reads data for encryption
  - Second NUMA crossing
- CPU writes data for encryption
  - Third NUMA crossing
4 Nodes, worst case

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

- 4 NUMA crossings
- 200GB/s of data on the NUMA fabric
  - Fabric saturates, cannot handle the load.
  - CPU Stalls, saturates early
Best Case Summary:

● 0 NUMA crossings
● 0GB/s of data on the NUMA fabric
How can we get as close as possible to the best case?

- Constrained to use 1 IP address per host
- Must use lagg(4) LACP network bonding
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
Network centric siloing

- Associate network connections with NUMA nodes
- Allocate local memory to back media files when they are DMA’ed from disk
- Allocate local memory for TLS crypto destination buffers & do SW crypto locally
- Run kTLS workers, RACK / BBR TCP pacers with domain affinity
- Choose local lagg(4) egress port

All of this is upstream!
4 Nodes, worst case with siloing

Steps to send data:
4 Nodes, worst case with siloing

Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
4 Nodes, worst case with siloing

Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
- CPU reads data for encryption
4 Nodes, worst case with siloing

Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
- CPU reads data for encryption
- CPU writes data for encryption
4 Nodes, worst case with siloing

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

- 1 NUMA crossing on average
  - 100% of disk reads across NUMA
- 50GB/s of data on each NUMA fabric link
  - Much less than the 280GB/sec of Inifinity fabric bandwidth
Real Life is Messy

- NICs on only 2 of the 4 NUMA nodes
- Differing number of NVME on each node
- Hacks to “pretend” we have NICs in all 4 domains
- Impacts worst and average cases
4 Nodes, worst case with siloing: messy

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

- 2 NUMA crossings on average
  - 100% of disk reads across NUMA
  - 100% of network writes across NUMA
- 100GB/s of data on the NUMA fabric
  - Less than the 280GB/s of Infiniity fabric bandwidth
Average Case Summary:

- 1.25 NUMA crossings on average
  - 75% of disk reads across NUMA
  - 50% of NIC transmits across NUMA due to unbalanced setup
- 62.5 GB/sec of data on NUMA fabric
Performance: 1 vs 4 nodes

Flat vs 4 Nodes per Socket

Network BW (Gb/s)

Single Node

4 Nodes
Would NIC based kTLS offload help for 400Gb/s?
Netflix 400Gb/s Video Serving Data Flow

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

400Gb/s == 50GB/s

~200GB/sec of memory bandwidth and ~64 PCIe Gen 4 lanes are needed to serve 400Gb/s
Using sendfile and software kTLS, data is encrypted by the host CPU.

400Gb/s == 50GB/s

~200GB/sec of memory bandwidth and ~64 PCIe Gen 4 lanes are needed to serve 400Gb/s
Netflix 400Gb/s Video Serving Data Flow

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

400Gb/s == 50GB/s

~100GB/sec of memory bandwidth and ~64 PCIe Gen 4 lanes are needed to serve 400Gb/s
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 requirements in half!
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>Segment Size</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15928</td>
</tr>
<tr>
<td>0</td>
<td>14480</td>
</tr>
<tr>
<td>0</td>
<td>13032</td>
</tr>
<tr>
<td>0</td>
<td>11584</td>
</tr>
<tr>
<td>0</td>
<td>10136</td>
</tr>
<tr>
<td>0</td>
<td>8688</td>
</tr>
<tr>
<td>0</td>
<td>7240</td>
</tr>
<tr>
<td>0</td>
<td>5792</td>
</tr>
<tr>
<td>0</td>
<td>4344</td>
</tr>
<tr>
<td>0</td>
<td>2896</td>
</tr>
<tr>
<td>0</td>
<td>1448</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

TCP segments of Plaintext TLS Record

---

**NETFLIX**

**PCIe Bus**

**NIC**

**100GbE Network**
<table>
<thead>
<tr>
<th>Size (Bytes)</th>
<th>TCP segments of Plaintext TLS Record</th>
<th>TCP segments of Encrypted TLS Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>15928</td>
<td></td>
<td>2048</td>
</tr>
<tr>
<td>14480</td>
<td></td>
<td>1448</td>
</tr>
<tr>
<td>13032</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>11584</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10136</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8688</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7240</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5792</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4344</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2896</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1448</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP segments of Plaintext TLS Record</td>
<td>TCP segments of Encrypted TLS Record</td>
<td></td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Host Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15928</td>
<td>7240</td>
<td></td>
</tr>
<tr>
<td>14480</td>
<td>5792</td>
<td></td>
</tr>
<tr>
<td>13032</td>
<td>4344</td>
<td></td>
</tr>
<tr>
<td>11584</td>
<td>2896</td>
<td></td>
</tr>
<tr>
<td>10136</td>
<td>1448</td>
<td></td>
</tr>
<tr>
<td>8688</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100GbE Network</td>
<td></td>
</tr>
<tr>
<td>PCIe Bus</td>
<td>NIC</td>
<td></td>
</tr>
<tr>
<td>194x298</td>
<td>13032</td>
<td></td>
</tr>
<tr>
<td>154x298</td>
<td>11584</td>
<td></td>
</tr>
<tr>
<td>183x298</td>
<td>10136</td>
<td></td>
</tr>
<tr>
<td>234x298</td>
<td>8688</td>
<td></td>
</tr>
<tr>
<td>273x298</td>
<td>7240</td>
<td></td>
</tr>
<tr>
<td>312x298</td>
<td>5792</td>
<td></td>
</tr>
<tr>
<td>355x298</td>
<td>4344</td>
<td></td>
</tr>
<tr>
<td>394x298</td>
<td>2896</td>
<td></td>
</tr>
<tr>
<td>434x298</td>
<td>1448</td>
<td></td>
</tr>
<tr>
<td>473x298</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
CX6-DX: TCP Retransmit
<table>
<thead>
<tr>
<th>NIC</th>
<th>PCIe Bus</th>
<th>100GbE Network</th>
</tr>
</thead>
</table>

TCP segments of Plaintext TLS Record

<table>
<thead>
<tr>
<th>Segment Size (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15528</td>
</tr>
<tr>
<td>14480</td>
</tr>
<tr>
<td>13332</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

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

TCP segments of Encrypted TLS Record

NIC

PCIe Bus

100GbE Network
CX6-DX: Initial Results

Peak: 125Gb/s per NIC, (~250Gb/s total)
Sustained: 75Gb/s per NIC, (~150Gb/s total)

- Pre-release Firmware
CX6-DX: Initial performance

- NIC stores TLS state per-session
- We have a lot of sessions active
  - (~400k sessions for 400Gb/s)
  - Performance gets worse the more sessions we add
- Limited memory on-board NIC
  - NIC pages in and out to buffers in host RAM
  - Buffers managed by NIC
PCIe Relaxed Ordering

- Allows PCIe transactions to pass each other
  - Should eliminate pipeline bubbles due to “slow” reads delaying fast ones.
  - May help with “paging in” TLS connection state
- Enabled Relaxed Ordering
  - Didn’t help
  - Turns out CX6-DX pre-release firmware hardcoded Relaxed Ordering to disabled
CX6-DX: Results from next firmware

- Firmware update enabled Relaxed Ordering on NIC
- Peak results improved: 160Gb/s per NIC (~320Gb/s total)
- Note that peak and sustained were effectively identical from this fw update forward.
- This is a new record!
- Nearly as fast as SW TLS (per NIC): 160Gb/s vs 190Gb/s, much faster overall
CX6-DX: Results from production fw

- Firmware update added “TLS_OPTIMIZE” setting
- Peak & sustained results improved:

190Gb/s per NIC (~380Gb/s total)!
CX6-DX: What’s needed to use of NIC TLS in production at Netflix?

● QoE testing
  ○ Measure various factors, such as rebuffer rate, play delay, time to quality, etc.
  ○ Initial results are great
  ○ Larger, more complete study scheduled soon.
CX6-DX: What’s needed to use of NIC TLS in production at Netflix?

- Track retransmits & move sessions to software
  - Monitor bytes retransmitted for lossy networks
  - Monitor segments retransmitted to protect against attacks
CX6-DX: Mixed HW/SW session perf?

- Moving a non-trivial percentage of conns to SW has unanticipated BW cost.
- Setting SW switch threshold to 1% bytes retransmitted moves \( \frac{1}{3} \) of conns to SW
- Max stable BW moves from 380Gb/s to 350Gb/s with roughly \( \frac{1}{3} \) of connections in SW
  - Performance impact is more than expected
4 Nodes, worst case with siloing + NIC kTLS

Steps to send data:
4 Nodes, worst case with siloing + NIC kTLS

Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
4 Nodes, worst case with siloing + NIC kTLS

Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
- DMA data from memory to network
4 Nodes, worst case with siloing + NIC kTLS

Steps to send data:
- DMA data from disk to memory
  - First NUMA bus crossing
- DMA data from memory to network
Worst Case Summary:

- 2 NUMA crossing on average
  - 100% of disk reads across NUMA
  - 100% of network writes across NUMA
- 100GB/s of data on the NUMA fabric
  - Less than the 280GB/s of Infinity fabric bandwidth
Average Case Summary:

● 1.25 NUMA crossings on average
  ○ 75% of disk reads across NUMA
  ○ 50% of NIC transmits across NUMA due to unbalanced setup
● 62.5 GB/sec of data on NUMA fabric
Other platforms? Ampere Altra

- “Mt. Snow”
  - Q80-30: 80 3.0GHz Arm Neoverse-N1 cores
  - 8 channels of 256GB DDR4-3200
  - 128 Lanes Gen4 PCIe
  - 16x WD SN720 2TB NVMe
  - 2 Mellanox CX6-DX NICs
Other platforms? Ampere Altra

- Minimal access to system counters
  - No way to see memory BW usage
  - No way to see IO bandwidth or latency
  - Leads to feeling like you’re driving blind
Other platforms? Ampere Altra

- Poor performance with SW kTLS:
  - CPU limited at 180Gb/s

- Poor initial performance with NIC TLS
  - PCIe limited at 240Gb/s
    - Very low CPU utilization
    - NICs saturated, and we see lots of output drops
Ampere: PCIe Extended Tags

- Poor initial performance with NIC TLS: 240Gb/s
- Very low CPU utilization
- NICs saturated, and we see lots of output drops
- Seems like a PCIe problem
Ampere: PCIe Extended Tags

- PCIe is more of a network than a bus
- Number of outstanding DMA reads is limited by the number of PCIe “tags”
- PCIe tag space is 5-bits by default, allowing for 32 DMAs to be in-flight at the same time
- PCIe extended tags increase the tag space to 8 bits, allowing 256 DMA reads in flight at the same time
- Like increasing TCP window size.
Ampere: PCIe Extended Tags

- After enabling extended tags, we see a bandwidth improvement:

240Gb/s -> 320Gb/s
Other platforms? Intel Ice Lake Xeon

- 8352V CPU
  - 36 cores, 2.1GHz
  - 8 channels 256GB DDR4-3200 (running at 2933)
  - 64 Lanes Gen4 PCIe
  - 20x Kioxia 4TB NVMe (PCIe Gen4)
  - 2 Mellanox CX6-DX NICs
Intel Ice Lake Xeon

- 230Gb/s SW kTLS
  - Limited by memory BW
Intel Ice Lake Xeon (WIP)

- 230Gb/s SW kTLS
  - Limited by memory BW
  - 8352V runs memory at 2993, others SKUs run at 3200
  - Would expect the same performance as AMD from that
- BIOS locked out PCIe Relaxed ordering, so no NIC KTLS results yet
But wait, there’s ... not ... more...

- 800Gb prototype sitting on datacenter floor due to shipping exception 😞
- Something to talk about next year?
Many thanks to:

● Warren Harrop & the Netflix Open Connect hardware team for putting together the testbed.
● FreeBSD developers for making such an awesome OS

Slides at: https://people.freebsd.org/~gallatin/talks/euro2021.pdf
Disk centric siloing

- Associate disk controllers with NUMA nodes
- Associate NUMA affinity with files
- Associate network connections with NUMA nodes
- Move connections to be “close” to the disk where the contents file is stored.
- After the connection is moved, there will be 0 NUMA crossings!
Disk centric siloing problems

- No way to tell link partner that we want LACP to direct traffic to a different switch/router port
  - So TCP acks and http requests will come in on the “wrong” port
- Moving connections can lead to TCP re-ordering due to using multiple egress NICs
- Some clients issue http GET requests for different content on the same TCP connection
  - Content may be on different NUMA domains!
Disk centric siloing problems

- Different numbers of NVME drives on each domain
  - Node 3 has 3x the number of NVME drives as Node 0
- Content popularity differences can lead to hot and cold disks
- All of this adds up to uneven use of each Numa Node.
  - Output limited by hottest Numa node
Disk centric siloing problems

- Moving established NIC TLS sessions to a different egress NIC is painful
Disk centric siloing problems

● Moving NIC TLS sessions is expensive
  ○ Session will be established before content location is known
AMD: NUMA w/NIC kTLS Offload

● Disk Siloing
  ○ Allocate host pages to back files on NUMA node close to NVME, not NIC
  ○ Eliminates the 0.75 crossings for 4 domains with NVME
  ○ Still have the 0.5 crossings on average for remapped NICs
AMD: NUMA w/NIC kTLS Offload

● Disk Siloing
  ○ Assumes equal number of NVME on each node
  ○ Actual machine has:
    ■ Node 0: 2 NVME
    ■ Node 1: 6 NVME
    ■ Node 2: 4 NVME
    ■ Node 3: 6 NVME
AMD: NUMA w/NIC kTLS Offload

● Disk Siloing
  ○ Peak of ~300Gb/s
  ○ Traffic unequal due to more NVME on Node 3
  ○ Output drops on mce3 (NIC port on Node 3) at 98Gb/s, while mce0 (NIC port on Node 1) is mostly idle at 40Gb/s
  ○ Tried “remapping” NVME and pretending some drives in different domains
AMD: NUMA w/NIC kTLS Offload

● Disk Siloing
  ○ Pretended some of Node 3’s NVME drives were in Node 1
    ■ Reached a peak of ~350Gb/s
    ■ Output still uneven between domains because of uneven popularity of content on different NVME drives
  ○ Sharding based on network (LACP) far more even