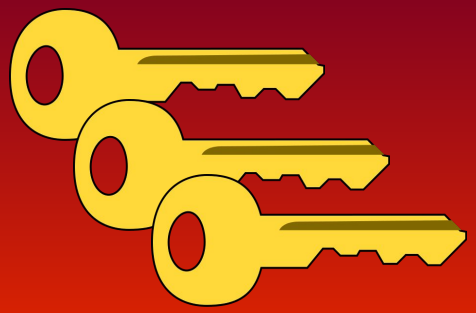




Kernel TLS and hardware TLS offload in FreeBSD 13

by

Mellanox, Chelsio and Netflix

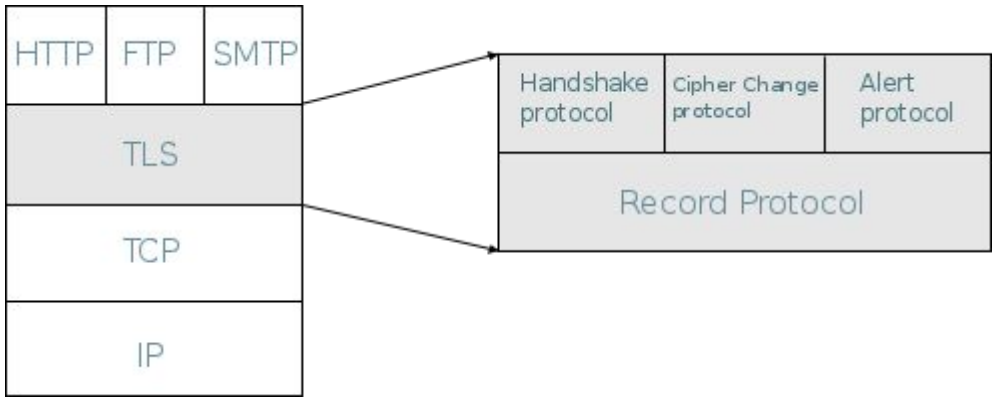
Why crypto?

- Bob and Alice and the secret message
- Mathematical dependance on a relatively small pre-shared key
- When used right:
  - Prevents eavesdropping
  - Prevents data tampering
- When used wrong:
  - Makes denial of service easier

What is TLS ?

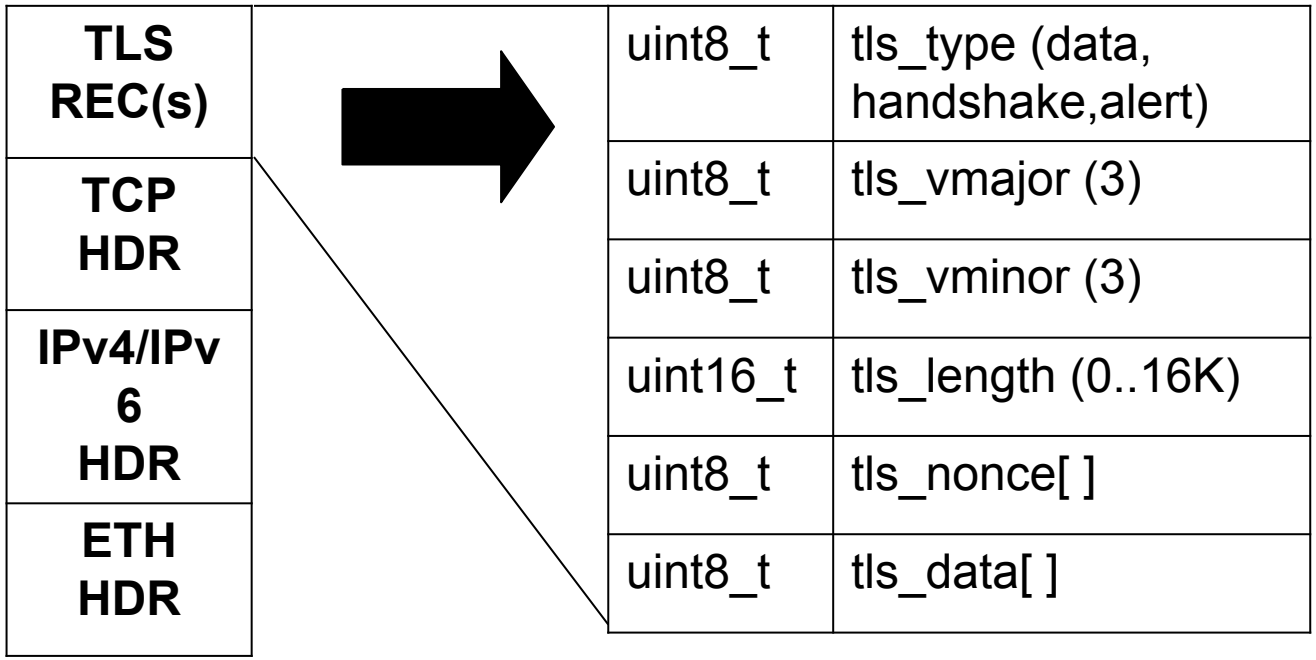
- Transport Layer Security, TLS
- Used behind https:// (TCP port 443)
- Supports multiple crypto codecs among others
  - AES 128B / 256B
- Supports multiple key exchange protocols
  - DiffieHellman, DH
  - Ron Rivest, Adi Shamir, Leonard Adleman, RSA
- Most recent version is v1.3

What is TLS ?


HTTP
FTP
SMTP
TLS
TCP
IP
Handshake protocol
Cipher Change protocol
Alert protocol
Record Protocol

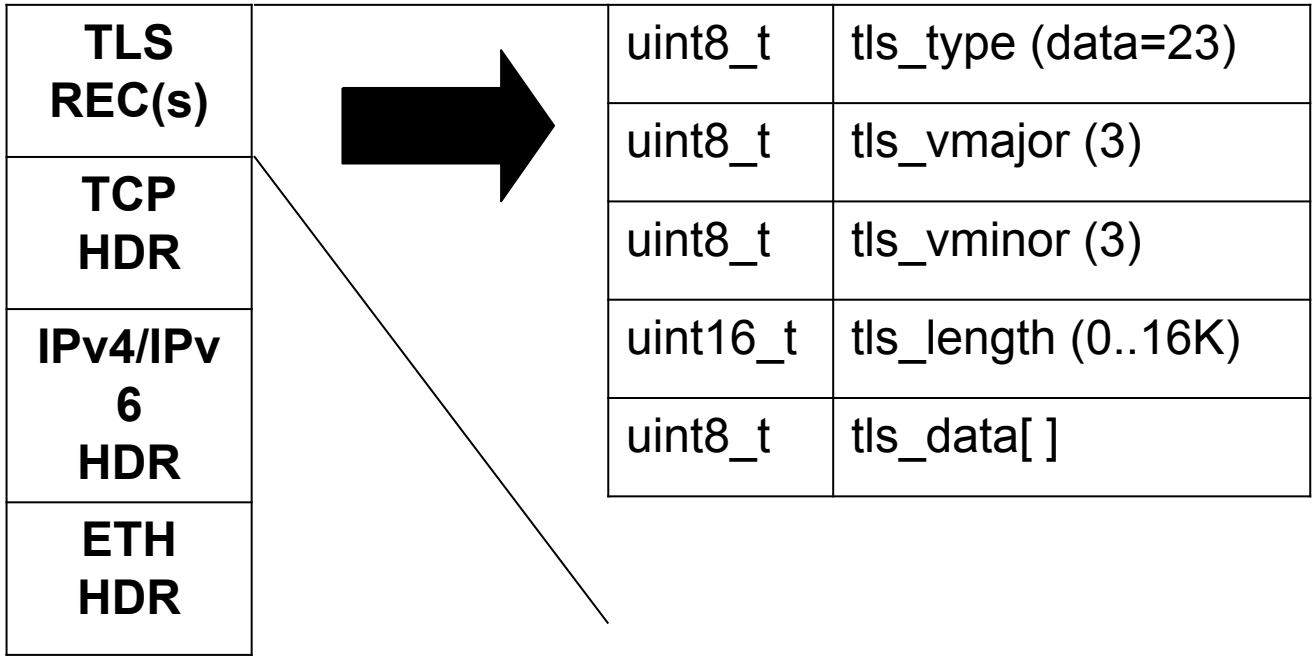
TLS v1.2

- Layout of a TLS record
- More detailed information at: https://tls.ulfheim.net/



TLS v1.3

- Layout of a TLS record
- More detailed information at: https://tls.ulfheim.net/

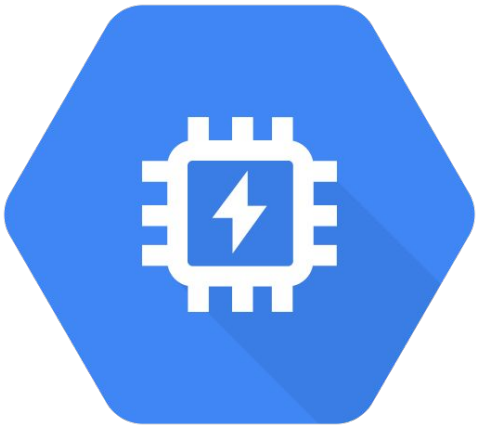


AES 128B / 256B

- Advanced Encryption Standard, AES
  - See: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
- A 16-byte block cipher
- The stream version can stop and resume encryption at any arbitrary point in the TLS record
  - Supports the concept of a crypto cursor
- FreeBSD also supports CBC

TLS implementations

- Current FreeBSD alternatives (OpenSSL based)
  - Generic user-space, AES-NI
  - SW kernel TLS, AES-NI
  - Open Crypto Framework kernel backend
  - TCP Offload Engine for TLS
  - NIC kernel TLS



... vs ...

A look inside OpenSSL

- Datapath is oriented around:
  - typedef struct bio_st BIO;
  - BIO_read()
  - BIO_write()
- All data must have a pointer in user-space in order to be encrypted
- Based on the source and sink methodology
- Refer to the bio(3) manual page

OpenSSL and kTLS

- 16 patches have been submitted by:
  Boris Pismenny <borisp@mellanox.com>
- FreeBSD userspace APIs:
  - #include <sys/ktls.h>
  - setsockopt(TCP_TXTLS_ENABLE)
  - setsockopt(TCP_TXTLS_MODE)
- FreeBSD kernel support added in r351522:
  - https://svnweb.freebsd.org/changeset/base/351522

Netflix kTLS

- Kernel TLS Motivation
  - Handle 100Gb/s of TLS with nginx
  - Retain performance advantages of async sendfile(9) (fewer context switches, no nginx thread pool, no extra memory copy)
  - Eliminate any possible inefficiency

New mbuf technologies

- Not ready flag
- Unmapped mbufs
- Send Tags

not ready mbuf flag

- mbuf flag M_NOTREADY tell socket buffers if mbufs are ready for transmission or not.
- Added to support async sendfile in r275329
- Sendfile(9) adds mbuf to socket buffer marked M_NOTREADY
  - Until M_NOTREADY is cleared, tcp cannot send it
- disk reads are issued into those mbufs
- M_NOTREADY cleared and tcp_usr_ready() routine called after disk read is complete
- Allows a simple mbuf filter routine, like TLS encryption, to process the mbufs before they are submitted to the network driver via the TCP stack.

Netflix “unmapped” mbufs

- Called “unmapped” because they carry an array of pointers to unmapped physical addresses.
- Initially envisioned for sendfile, not TLS
- Dramatically reduces the length of socket buffer mbuf chains, thus reducing cache misses. For a 16K TLS record, it compresses chains by about 6:1 (TLS hdr, trailer and 4 buffers). For unencrypted sendfile, it can compress mbuf chains up to 19:1
  - 5-20% CPU reduction in Netflix unencrypted workloads
- Describes a TLS record entirely, including TLS header, trailer, message data, and pointers to kernel TLS session state in a single mbuf
- A single reference counted entity per TLS record is key for NIC TLS offload to be able to easily handle TCP retransmissions.

Netflix Software kTLS

Software Kernel TLS Implementation, TLS 1.0 -> TLS 1.3

- Plaintext data passed to kernel via sendfile() or sosend().
- The kernel frames TLS records into M_NOMAP mbufs at sendfile() or sosend() time and places them into socket buffers.
- Mbuf chains are marked with M_NOTREADY
- Framed records are queued for encryption when they would previously be marked “ready”
- Encryption is done by a pool of kernel threads (1 per core)
- Once encrypted, mbufs are marked “ready” & sent to TCP

mbuf send tags

- A property of mbufs which tell the underlying network interface about dedicated packet processing and queues.
- A quick and efficient way to demultiplex data traffic.
- Allows for traversal through VLAN and LAGG (Link Aggregation).
- Safe against route changes.

mbuf send tag APIs

- Control path methods:
  - struct mbuf_snd_tag *mst;
  - struct ifnet *ifp;

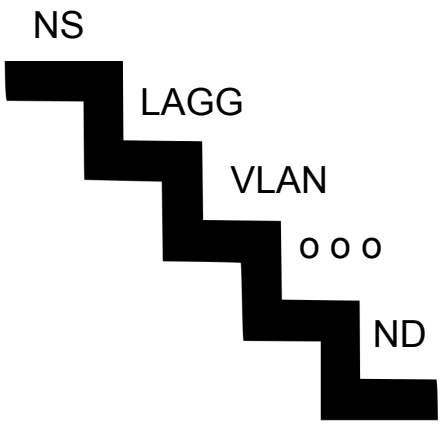
  - Allocate(ifp, &mst)
  - Modify(mst, arg)
  - Query(mst, arg)
  - Free(ifp, mst)

mbuf send tags

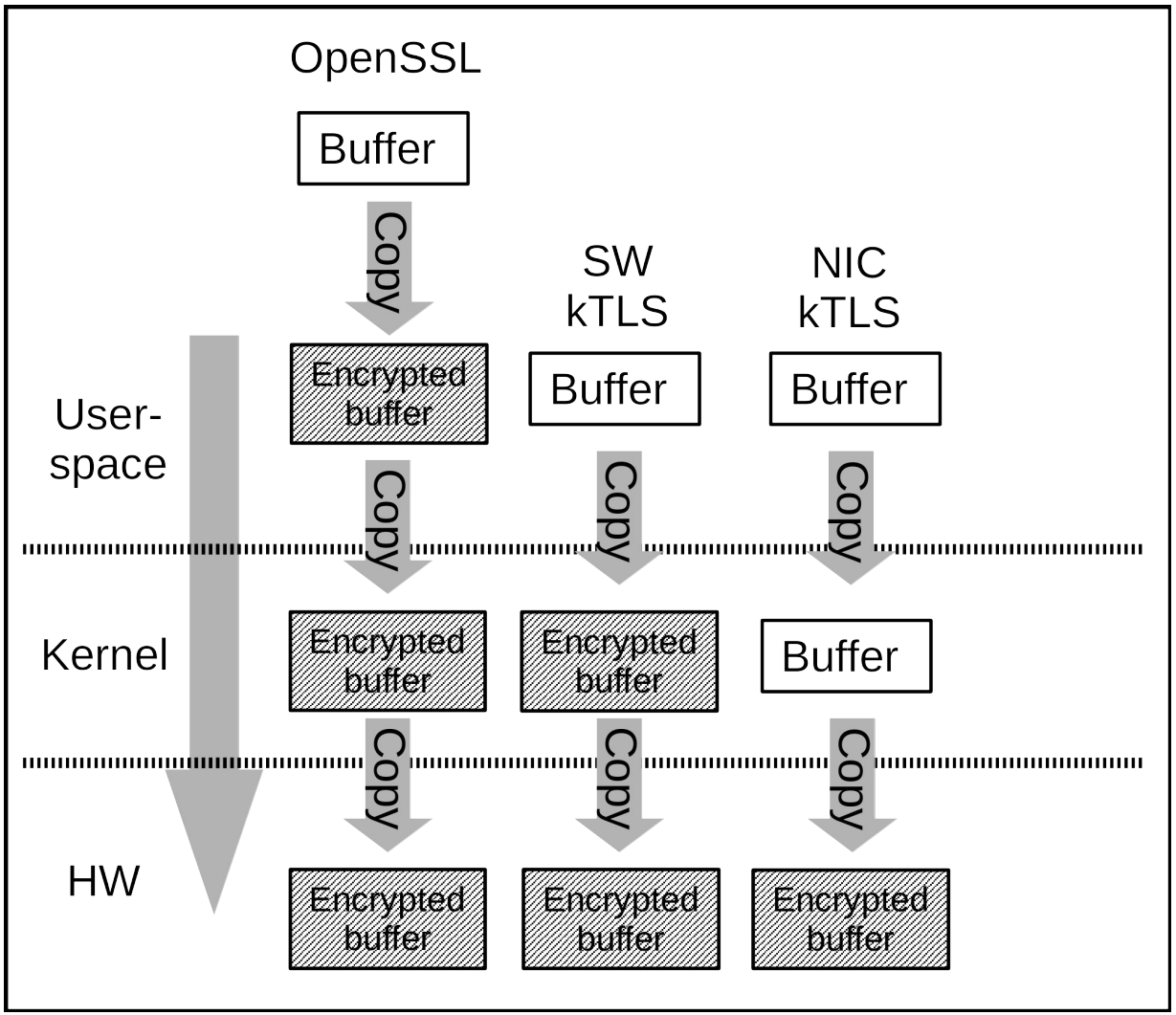
- From Network Stack, NS, perspective:
  - struct mbuf *mb;
  - struct ifnet *ifp;
  - m_pkthdr.snd_tag = mst;
  - m_pkthdr.csum_flag |= CSUM_SND_TAG;
  - ifp->if_output(mb);

mbuf send tags

- From Network Driver, ND, perspective:
  - struct mbuf *mb;
  - struct xxx_send_tag *st;
  - st = container_of(m_pkthdr.snd_tag, …)
  - select queue by st->queue;



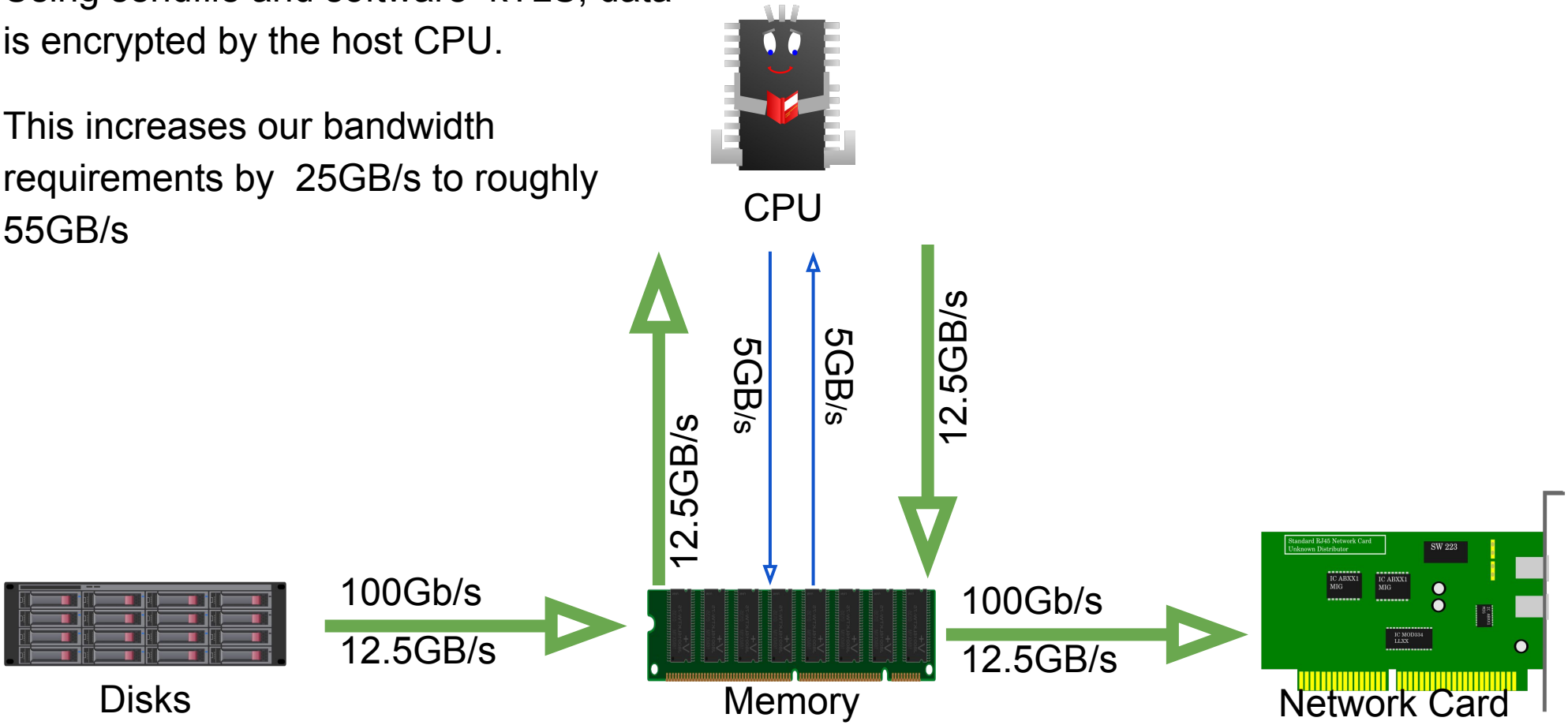
Dataflow overview



Sendfile dataflow overview

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

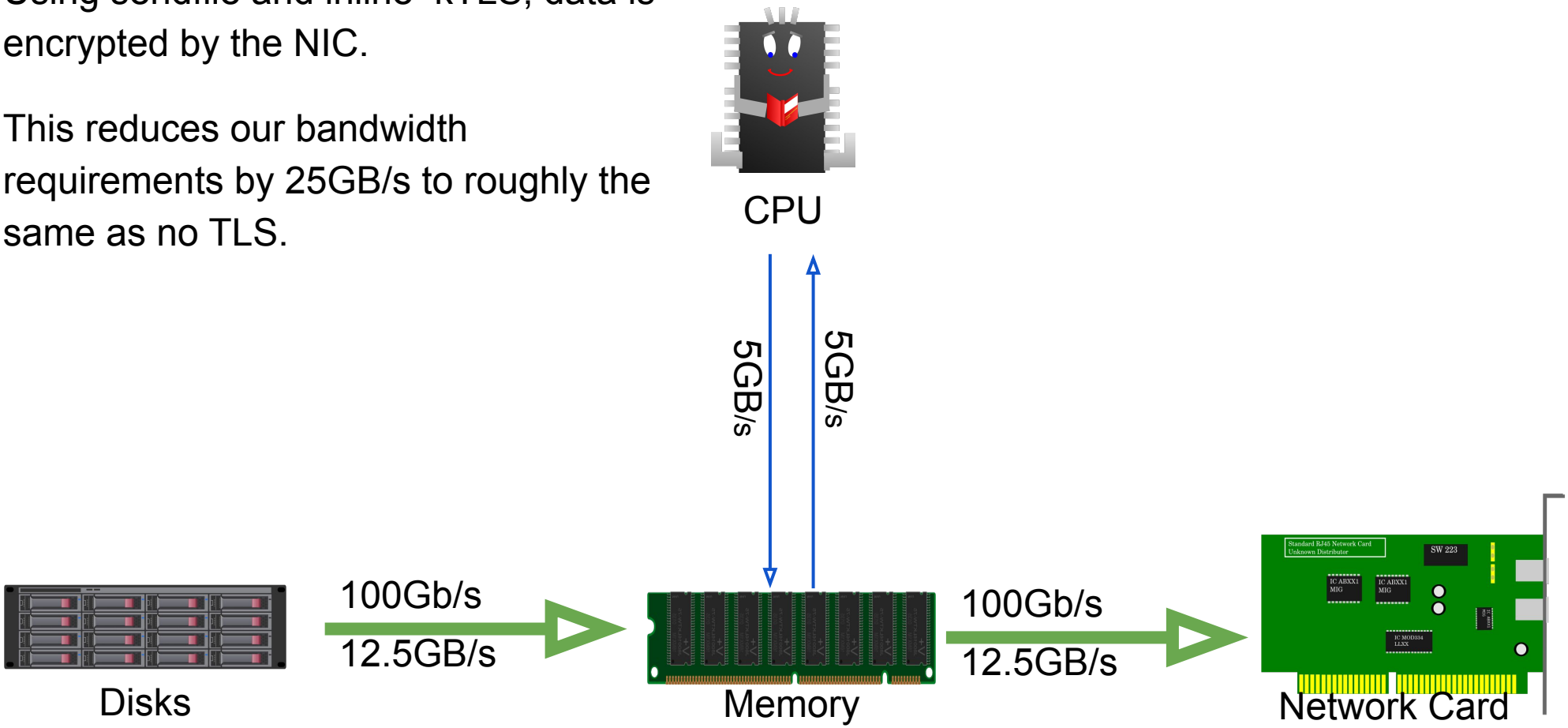
This increases our bandwidth requirements by 25GB/s to roughly 55GB/s



Sendfile dataflow overview

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

This reduces our bandwidth requirements by 25GB/s to roughly the same as no TLS.



TLS before and after

iperf_tls_server.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/> Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
7	0.108610	11.215.5.66	11.215.5.65	TLSv1.2	339	Client Key Exchange, Change Cipher
8	0.215088	11.215.5.65	11.215.5.66	TCP	66	5001 → 24791 [ACK] Seq=947 Ack=409
9	0.636250	11.215.5.66	11.215.5.65	TLSv1.2	111	Encrypted Handshake Message
10	0.636405	11.215.5.65	11.215.5.66	TLSv1.2	247	New Session Ticket, Change Cipher
11	0.736667	11.215.5.66	11.215.5.65	TCP	66	24791 → 5001 [ACK] Seq=454 Ack=112
12	1.079358	11.215.5.65	11.215.5.66	TLSv1.2	111	Encrypted Handshake Message
13	1.081282	11.215.5.66	11.215.5.65	TCP	1514	24791 → 5001 [ACK] Seq=454 Ack=117
14	1.184810	11.215.5.65	11.215.5.66	TCP	66	5001 → 24791 [ACK] Seq=1173 Ack=19
15	1.186756	11.215.5.66	11.215.5.65	TCP	2962	24791 → 5001 [ACK] Seq=1902 Ack=11
16	1.186772	11.215.5.65	11.215.5.66	TCP	66	5001 → 24791 [ACK] Seq=1173 Ack=47
17	1.188830	11.215.5.66	11.215.5.65	TCP	2962	24791 → 5001 [ACK] Seq=4798 Ack=11

▸ Frame 13: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
▸ Ethernet II, Src: RealtekU_55:15:75 (52:54:00:55:15:75), Dst: RealtekU_f3:64:3f (52:54:00:f3:64:3f)
▸ Internet Protocol Version 4, Src: 11.215.5.66, Dst: 11.215.5.65
▸ Transmission Control Protocol, Src Port: 24791, Dst Port: 5001, Seq: 454, Ack: 1173, Len: 1448

```
0000  52 54 00 f3 64 3f 52 54  00 55 15 75 08 00 45 00   RT··d?RT ·U·u··E·
0010  05 dc 00 00 40 00 40 06  12 ec 0b d7 05 42 0b d7   ····@·@· ·····B··
0020  05 41 60 d7 13 89 50 26  6d 48 33 db dd 59 80 10   ·A`···P& mH3··Y··
0030  04 02 7c 0b 00 00 01 01  08 0a 10 12 c2 5f 27 96   ··|····· ·····_'·
0040  6e 25 17 03 03 40 18 00  00 00 00 00 00 00 01 0a   n%···@·· ········
0050  a2 32 df ac 0a 77 87 17  d4 c0 d4 61 35 b3 13 86   ·2···w·· ···a5···
0060  76 a8 36 93 56 88 32 38  fa 47 d2 05 82 d0 84 b9   v·6·V·28 ·G······
0070  38 5e 2c 1c 22 26 cb 63  dc 40 b2 9f 6c 6d bf cc   8^,·"&·c ·@··lm··
0080  ec df 0f a1 93 49 09 e5  40 0b a3 f2 2d 07 45 61   ·····I·· @···-·Ea
0090  2e 52 af 8b ad 86 e0 78  01 b3 16 8e 99 bd c5 df   .R·····x ········
00a0  78 7c c9 32 eb a9 26 b3  b1 60 9f c7 a0 57 9a 1d   x|·2··&· ·`···W··
00b0  39 4f e6 ef 4a 52 70 bb  8e c4 3a da b2 e5 a5 30   9O··JRp· ··:····0
00c0  30 2c c1 0e 0e 84 56 cc  9d 08 d0 f1 16 4a 01 0a   0,····V· ·····J··
00d0  7d 1d bc 6b 34 94 fe 6f  47 f4 d1 39 3e e8 09 fb   }··k4··o G··9>···
00e0  09 dc 44 5b 92 1f 1b ed  fc 2b 2d 9f da a4 74 e7   ··D[···· ·+-···t·
00f0  16 87 7b 9c c7 48 b5 fd  7e 92 2a 00 d5 0a b6 a1   ··{··H·· ~·*·····
0100  dc da 08 7d 1b ae 50 40  43 fe ff e0 e6 3c 6f 33   ···}··P@ C····<o3
0110  9e ff 0d 05 c4 69 1b 84  11 38 69 85 2d 30 9b f6   ·····i·· ·8i·-0··
```

iperf_tls_server.pcap Packets: 100 · Displayed: 100 (100.0%) Profile: Default

iperf_tls_client.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/> Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
8	0.108653	11.215.5.66	11.215.5.65	TLSv1.2	339	Client Key Exchange, Change Cipher
9	0.216265	11.215.5.65	11.215.5.66	TCP	66	5001 → 24791 [ACK] Seq=947 Ack=409
10	0.636028	11.215.5.66	11.215.5.65	TLSv1.2	111	Encrypted Handshake Message
11	0.637301	11.215.5.65	11.215.5.66	TLSv1.2	247	New Session Ticket, Change Cipher
12	0.736716	11.215.5.66	11.215.5.65	TCP	66	24791 → 5001 [ACK] Seq=454 Ack=112
13	1.080408	11.215.5.65	11.215.5.66	TLSv1.2	111	Encrypted Handshake Message
14	1.081042	11.215.5.66	11.215.5.65	TCP	1514	24791 → 5001 [ACK] Seq=454 Ack=117
15	1.185996	11.215.5.65	11.215.5.66	TCP	66	5001 → 24791 [ACK] Seq=1173 Ack=19
16	1.186043	11.215.5.66	11.215.5.65	TCP	2962	24791 → 5001 [ACK] Seq=1902 Ack=11
17	1.187430	11.215.5.65	11.215.5.66	TCP	66	5001 → 24791 [ACK] Seq=1173 Ack=47
18	1.187444	11.215.5.66	11.215.5.65	TCP	2962	24791 → 5001 [ACK] Seq=4798 Ack=11

▸ Frame 14: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
▸ Ethernet II, Src: RealtekU_55:15:75 (52:54:00:55:15:75), Dst: RealtekU_f3:64:3f (52:54:00:f3:64:3f)
▸ Internet Protocol Version 4, Src: 11.215.5.66, Dst: 11.215.5.65
▸ Transmission Control Protocol, Src Port: 24791, Dst Port: 5001, Seq: 454, Ack: 1173, Len: 1448

```
0000  52 54 00 f3 64 3f 52 54  00 55 15 75 08 00 45 00   RT··d?RT ·U·u··E·
0010  05 dc 00 00 40 00 40 06  00 00 0b d7 05 42 0b d7   ····@·@· ·····B··
0020  05 41 60 d7 13 89 50 26  6d 48 33 db dd 59 80 10   ·A`···P& mH3··Y··
0030  04 02 27 ff 00 00 01 01  08 0a 10 12 c2 5f 27 96   ··'····· ·····_'·
0040  6e 25 17 03 03 40 18 00  00 00 00 00 00 00 01 04   n%···@·· ········
0050  00 00 00 00 00 00 01 00  00 13 89 00 00 00 00 00   ········ ········
0060  00 00 00 ff ff ff 9c 00  00 00 00 38 39 30 31 32   ········ ···89012
0070  33 34 35 36 37 38 39 30  31 32 33 34 35 36 37 38   34567890 12345678
0080  39 30 31 32 33 34 35 36  37 38 39 30 31 32 33 34   90123456 78901234
0090  35 36 37 38 39 30 31 32  33 34 35 36 37 38 39 30   56789012 34567890
00a0  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35 36   12345678 90123456
00b0  37 38 39 30 31 32 33 34  35 36 37 38 39 30 31 32   78901234 56789012
00c0  33 34 35 36 37 38 39 30  31 32 33 34 35 36 37 38   34567890 12345678
00d0  39 30 31 32 33 34 35 36  37 38 39 30 31 32 33 34   90123456 78901234
00e0  35 36 37 38 39 30 31 32  33 34 35 36 37 38 39 30   56789012 34567890
00f0  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35 36   12345678 90123456
0100  37 38 39 30 31 32 33 34  35 36 37 38 39 30 31 32   78901234 56789012
0110  33 34 35 36 37 38 39 30  31 32 33 34 35 36 37 38   34567890 12345678
```

iperf_tls_client.pcap Packets: 100 · Displayed: 100 (100.0%) Profile: Default

NIC kTLS offload challenges

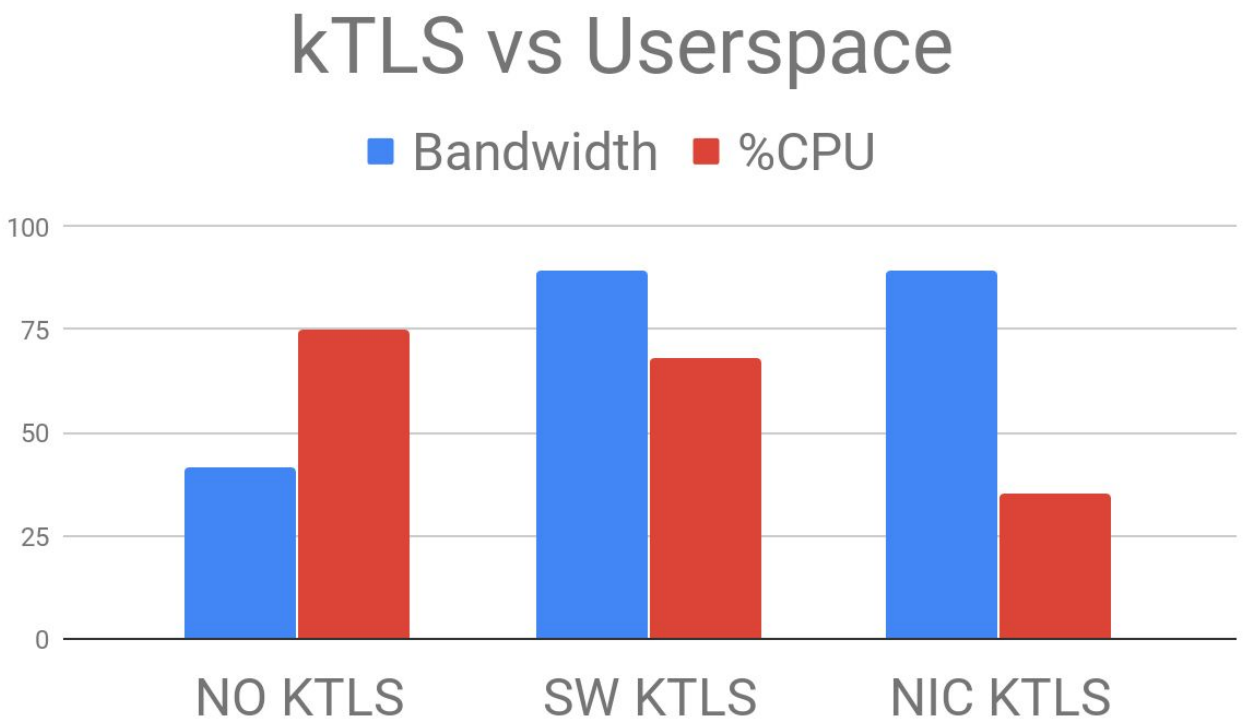
- Minor OSI model violation.
- Packets are sent containing full headers, except for un-encrypted payload.
- Prior to retransmission, crypto cursor needs update by re-transmitting off-the-wire parts of the TLS record, if any.

Benchmarks

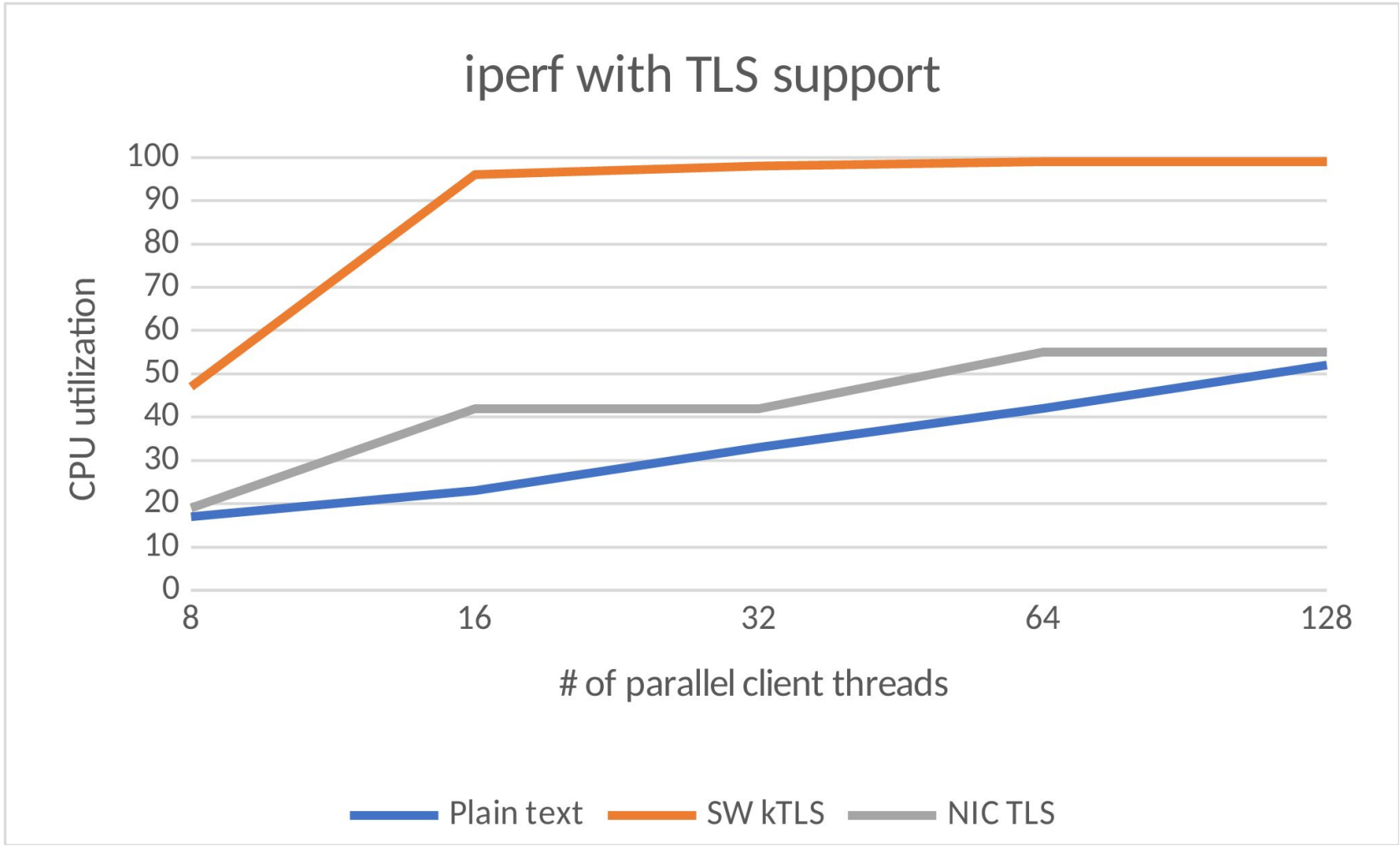
Netflix Video Serving with TLS

Kernel TLS Performance: 90Gb/s, 68% CPU (SW), 35% CPU (T6 NIC kTLS)

- Original (~2016) Netflix 100G NVME flash appliance
  - E5-2697A v4 @ 2.60GHz (16 core / 32 HTT), 128GB DDR4 2400MT/s, 1x100GbE, 4xNVME



Mellanox NIC TLS



Mellanox NIC TLS support

- ConnectX-6 DX (coming October 2019)
  - http://www.mellanox.com/page/ethernet_cards_overview
  - 16 000 000 simultaneous TLS connections (25, 50, 100 and 200 Gbit/s)

Chelsio HW TLS support

- T6 NIC TLS supports TLS v1.1 and v1.2 using both AES-CBC and AES-GCM.
- TOE TLS support for kTLS is in progress.
- ccr(4) can be used for AES-GCM via the OCF backend.

Questions and Answers

Q/A