TLS Performance Characterization on modern x86 CPUs

Pawel Szymanski, Manasi Deval

Netdev 0x14, August 2020

Ethernet Product Group

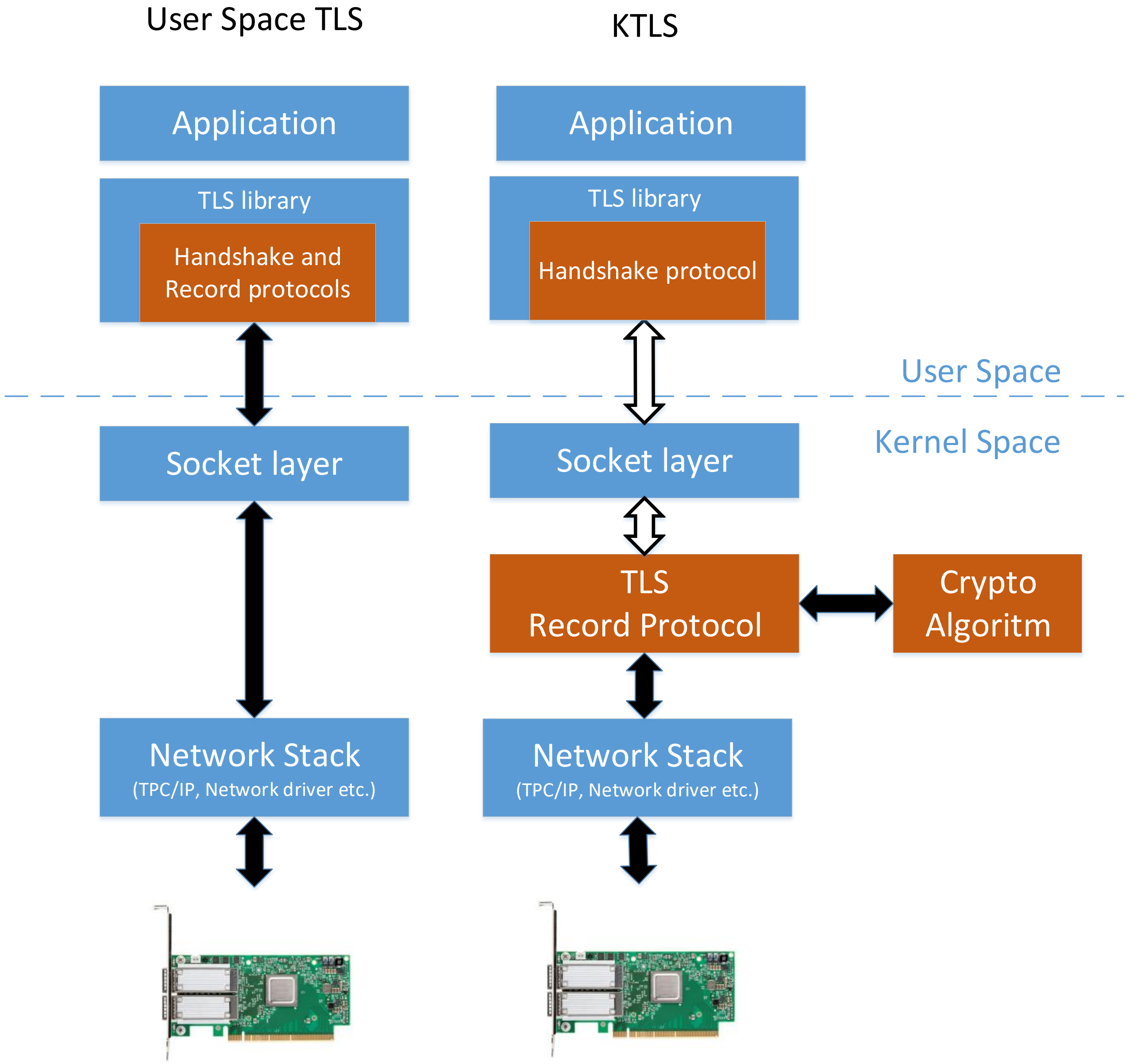
Agenda

- Background
- Test Setup
- Performance Characterization Results
- Summary

Background – What is TLS?

- Network protocol providing privacy and data integrity
- Runs in L4 layer - for example TCP and UDP
- Commonly used in many Internet applications such as web browsing, VoD etc.
- Consists of 2 subprotocols:
  - **Handshake Protocol** – used to negotiate the security parameters of the connection (for example crypto algorithm)
  - **Record Protocol** – fragments application data in records, protects the records and transmits them over a transport protocol
- Supports multiple crypto algorithms – for example AES GCM

Background – TLS implementation options in Linux



User Space TLS

Application

TLS library

Handshake and Record protocols

Socket layer

Network Stack (TPC/IP, Network driver etc.)

KTLS

Application

TLS library

Handshake protocol

Socket layer

TLS Record Protocol

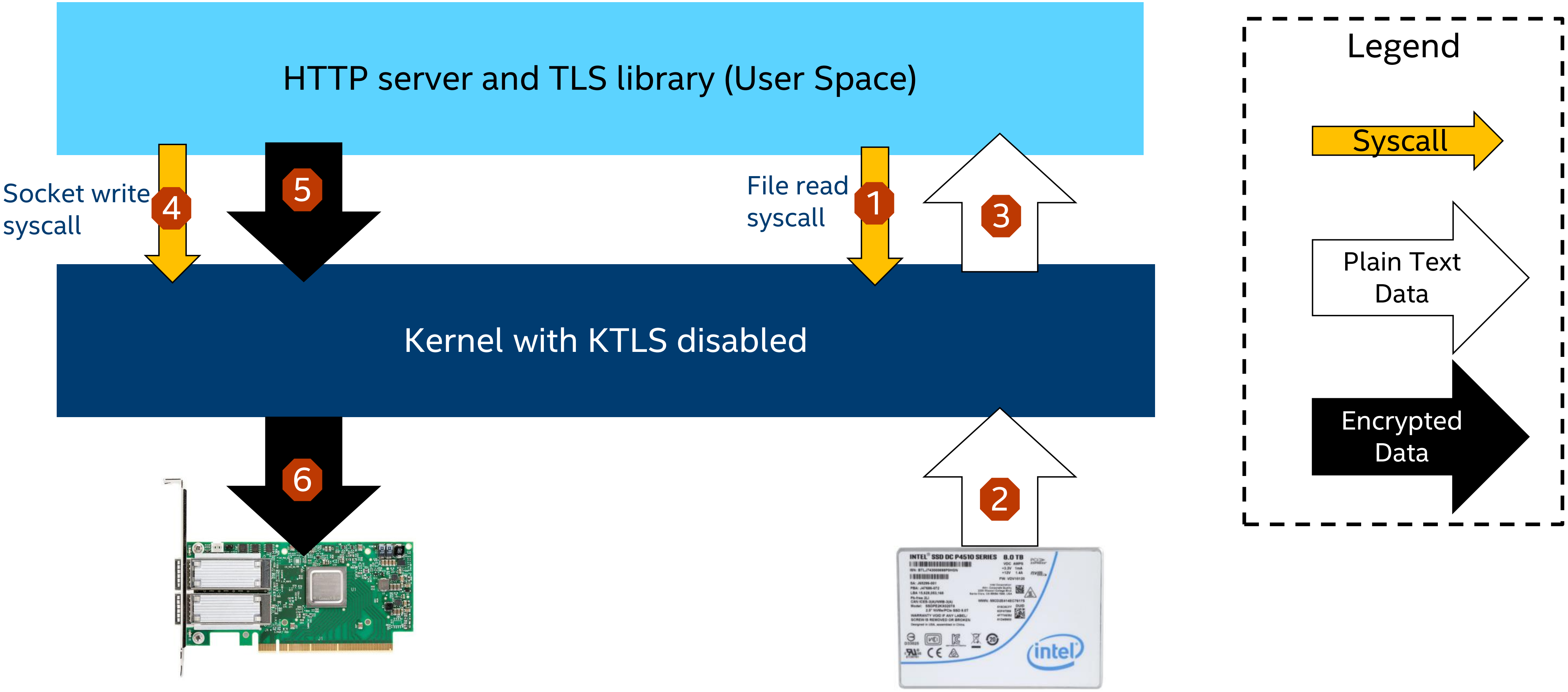
Crypto Algoritm

Network Stack (TPC/IP, Network driver etc.)

User Space

Kernel Space

HTTP server - User Space TLS Data Flow



HTTP server and TLS library (User Space)

Socket write
syscall

4

5

File read
syscall

1

3

Kernel with KTLS disabled

6

2

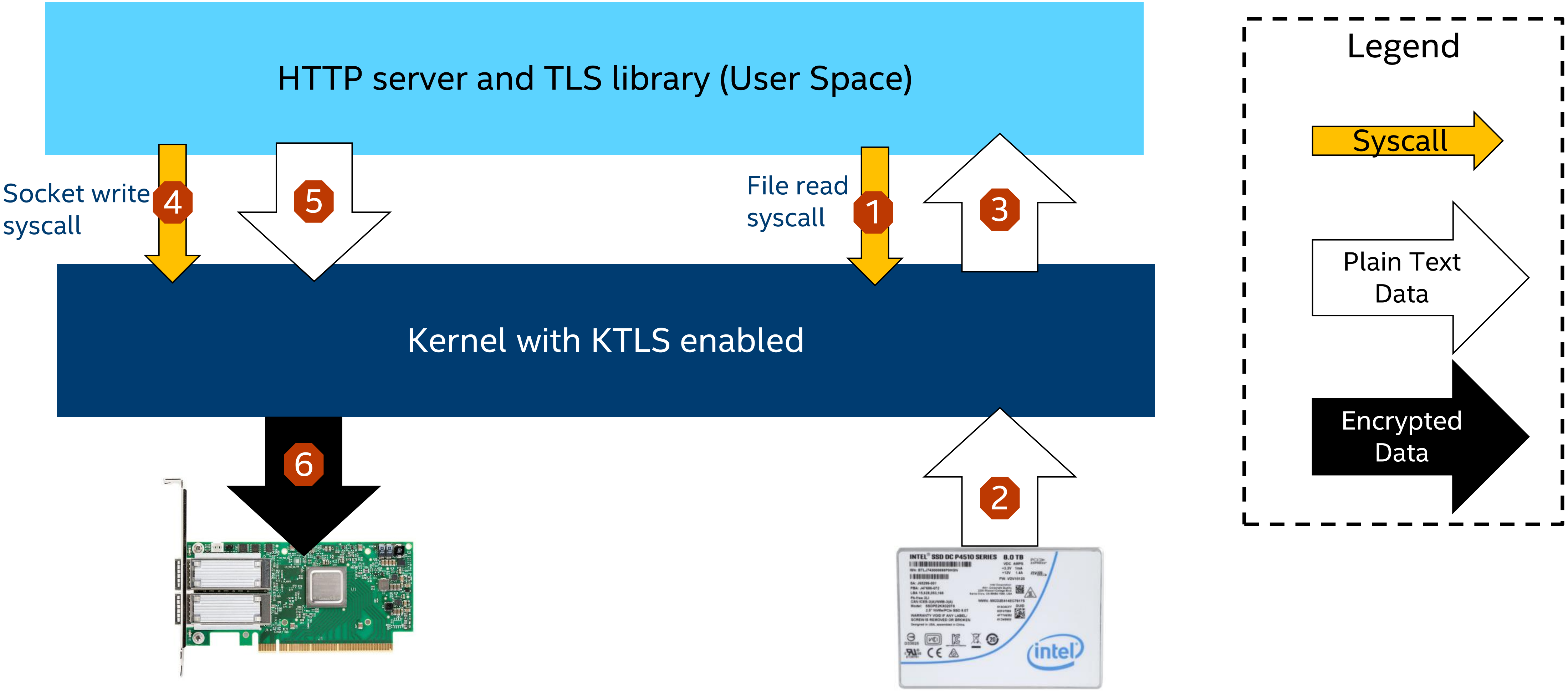
Legend

Syscall

Plain Text
Data

Encrypted
Data

HTTP server - Kernel TLS Write Data Flow



HTTP server and TLS library (User Space)

Socket write syscall

4

5

File read syscall

1

3

Kernel with KTLS enabled

6

2

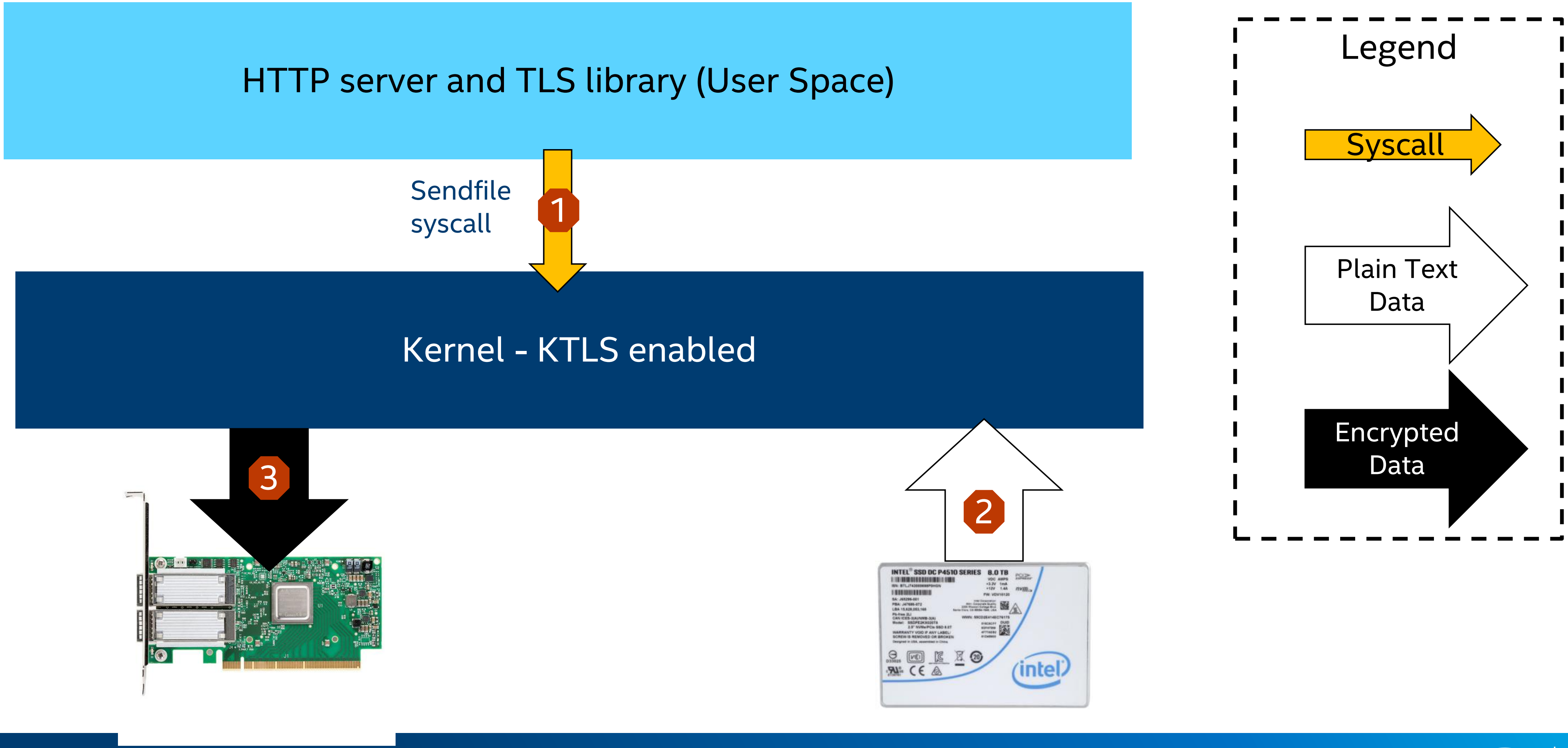
Legend

Syscall

Plain Text Data

Encrypted Data

HTTP server - Kernel TLS Sendfile Data Flow



HTTP server and TLS library (User Space)

Sendfile syscall

1

Kernel - KTLS enabled

2

3

Legend

Syscall

Plain Text Data

Encrypted Data

TLS Performance Characterization Goal

Compare TLS Record protocol throughput for User Space TLS, KTLS Write and KTLS Sendfile in the following scenarios:

- Simple Web Server
  - File size: 1KB - 10MB
  - TCP/TLS connection number: 100
  - Each connection sends HTTP Get requests back-to-back
- Media streaming (e.g. MPEG DASH)
  - File size: 1MB
  - TCP/TLS connection number: 10K
  - Each connection sends HTTP Get request with 1-5s space in between

Hardware Setup

HTTP Server

2x Intel Xeon Gold 6142M CPU 32C @ 2.60GHz (Skylake)

384GB DDR4 (192 GB per socket, 32GB per channel) @ 2666 MT/s

1x Intel M.2 SATA SSD DC S3110 520GB (OS)

1x Intel NVMe SSD DC P4600 1TB

1x Intel 800 series NIC - 100GbE

CPU0

UPI

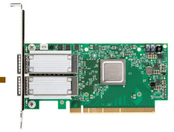
CPU1

HTTP Client

2x Intel Xeon Gold 6142M CPU 32C @ 2.60GHz (Skylake)

384GB DDR4 @ 2666 MT/s

1x Intel M.2 SATA SSD DC S3110 520GB (OS)
1x Intel 800 series NIC - 100GbE



100GbE

BIOS Configuration

Hyper-threading	Disabled
C-states	Disabled
P-states (EIST)	Disabled
Turbo	Disabled
CPU Power & Performance Policy	Performance
Enable CPU HWPM	Native Mode

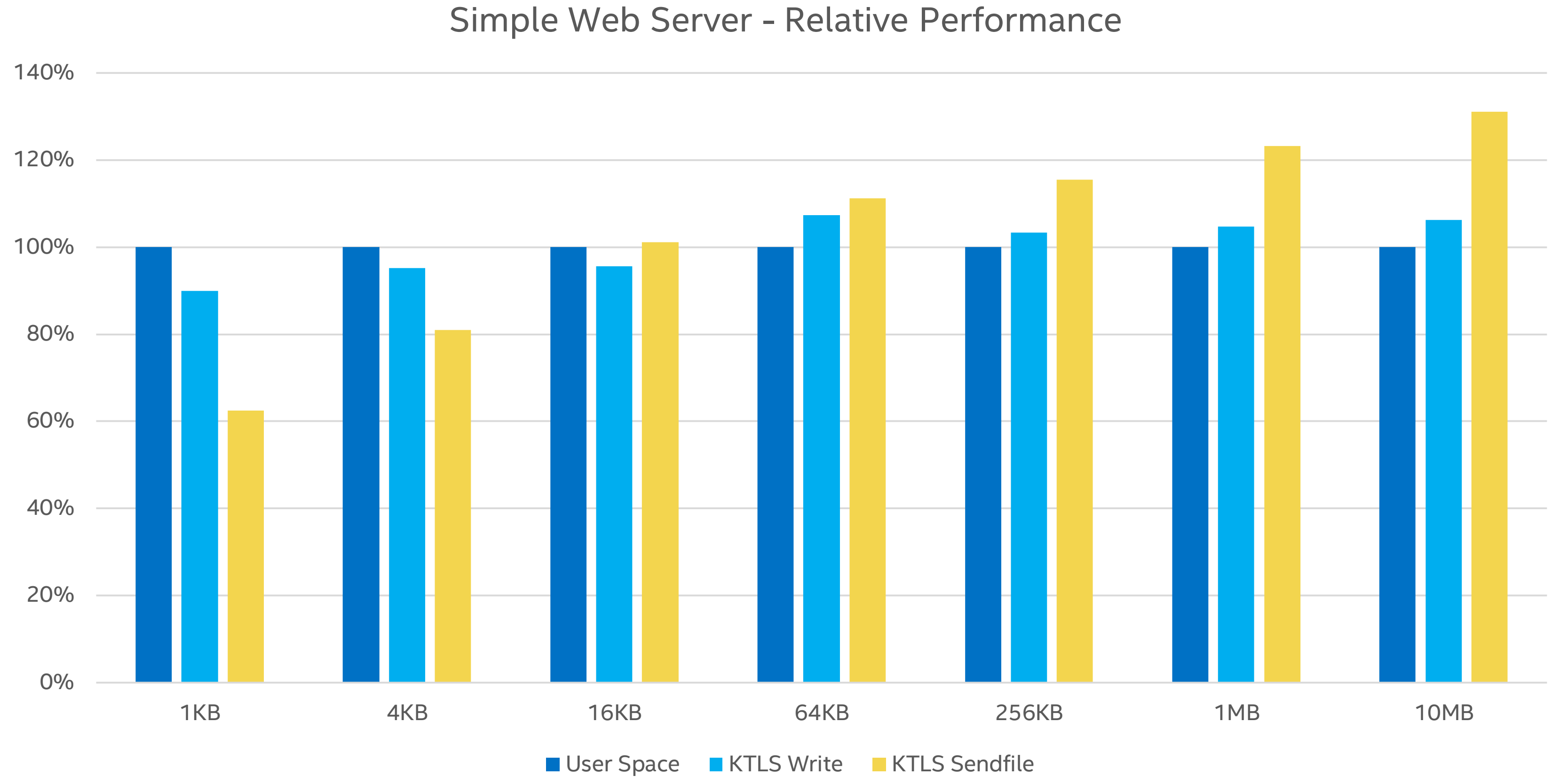
Software Configuration

OS	Ubuntu 18.04.2
Linux Kernel	5.1.0 with KTLS enabled and AESNI driver
OpenSSL	OpenSSL 3.0.0-dev with AESNI support enabled
NGINX (server)	1.5.11 with KTLS Sendfile patch
WRK (client)	4.1.0
TLS configuration	TLS 1.2 Max Record Size – 16KB Crypto Algorithm - AES128-GCM-SHA256
HTTP	HTTP 1.1 Persistent connections – Enabled HTTP GET Requests

Simple Web Server – Throughput Comparison

Test parameters:

- 16 NGINX process
- 100 HTTPS connections
- HTTP GET Requests



KTLS Sendfile is more efficient for file size 64KB and above

Why KTLS Sendfile is less efficient for smaller files?

1. Sending HTTP response needs 2 syscalls
   - Write() syscall to send HTTP Response Header from user space buffer
   - Sendfile() syscall to send HTTP Response payload from file system

2. Precomputed hash key exponents not reused for subsequent TLS records
   - AESNI Crypto driver precomputes hash key exponents to parallelize encryption process (so called Karatsuba algorithm)
   - No mechanism to reuse pre-computed hash keys between subsequent encrypt requests passed from TLS to Crypto driver

Media Streaming Test Scenario

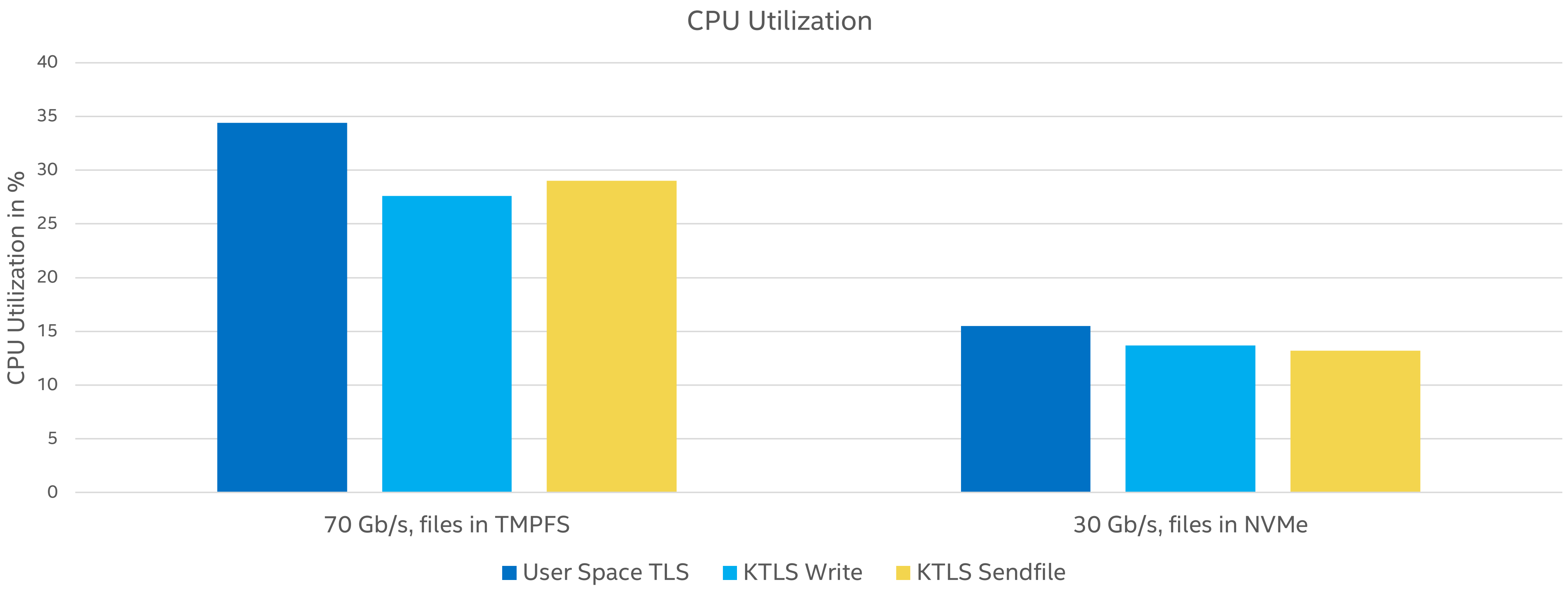
HTTPS traffic parameters:

- Iso Throughput:
  - Files in TMPFS - 70 Gb/s
  - Files in NVMe – 30 Gb/s
- File size: 1MB
- # of connections: 10K

Metrics taken :

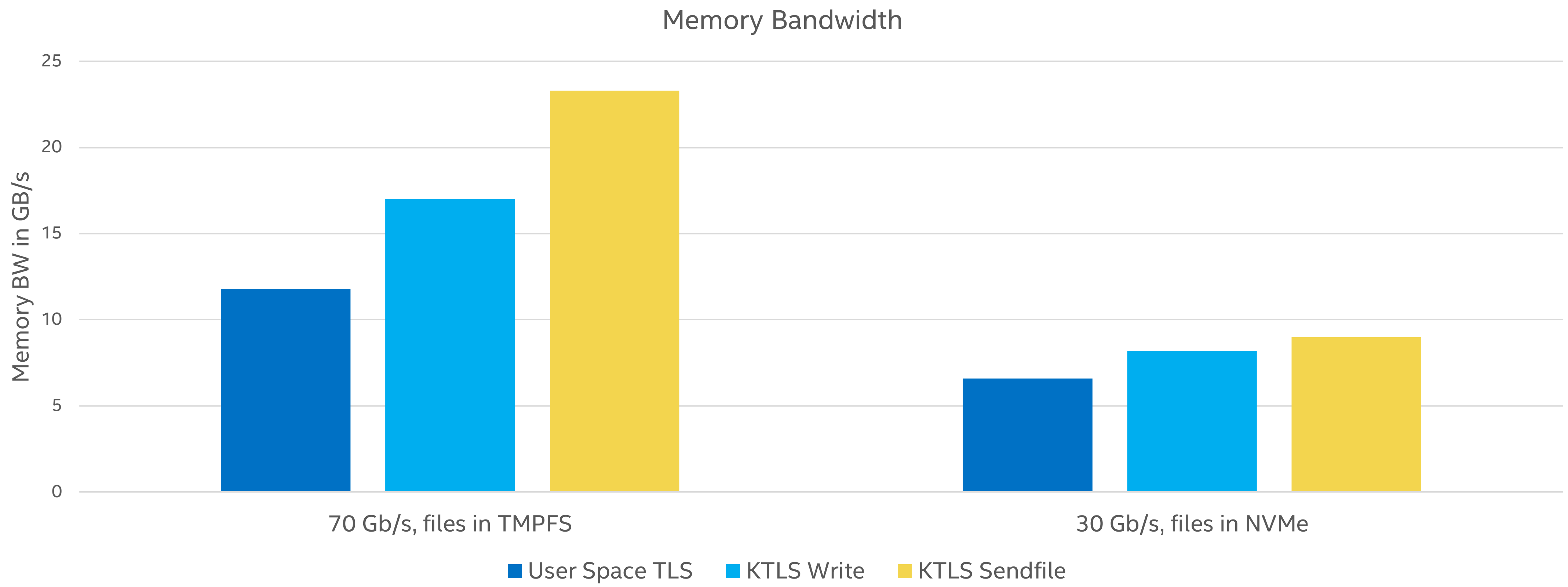
- CPU Utilization
- Memory Bandwidth Utilization

Media Streaming Test – CPU utilization



- KTLS Write and KTLS Sendfile efficiency are close
- User Space efficiency is 16-20% lower

Media Streaming Test – Memory Bandwidth



KTLS Sendfile and KTLS Write consume much more memory BW than User Space TLS

Key Take-aways

- Main options to implement TLS
  - User Space TLS
  - KTLS: Write and Sendfile
- In Simple Web Server scenario, KTLS Sendfile provides highest performance for files 64KB and above
- In Multimedia Streaming scenario, KTLS Sendfile and KTLS Write provide lower CPU utilization, but higher memory bandwidth utilization

Thank You