Intel® Verified Reference Configuration for Network Function Virtualization Infrastructure (NFVI) v4 and Secure Access Service Edge (SASE) on Kubernetes Platform

Intel Accelerated Solution

Revision 1.0

June 2023

Authors
Jonathan Tsai, Timothy Miskell, Ai Bee Lim, Vaishnavi Saravanan, Chenxi Wang

Key Contributors
David Lu, Georgii Tkachuk

Document Number: 782901

Contents

Figures

Tables

Revision History

Document Number	Revision Number	Description	Revision Date
782901	1.0	Initial release	July 2023

§

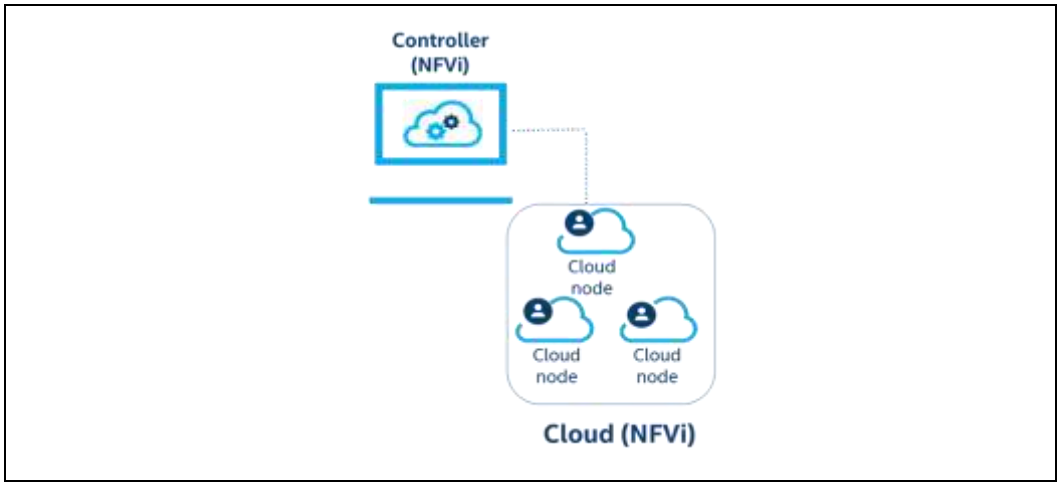
1 Introduction

Intel Accelerated Solutions are a family of workload-optimized, infrastructure solutions, based on the Intel® Xeon® Scalable processor family targeting complex workloads of today. This document describes a reference architecture for the 4th Gen Intel® Xeon® Scalable processor family.

When network operators, service providers, cloud service providers, or enterprise infrastructure companies choose an Intel® Verified Reference Configuration (VRC) for Network Function Virtualization Infrastructure (NFVI) deployment based upon an Intel® Xeon® Processor Scalable Family-based Intel® Verified Reference Configuration, they should be able to deploy various network function virtualized applications more securely and efficiently than ever before. End users spend less time, effort, and expense evaluating hardware and software options. Intel® Verified Reference Configurations help end users simplify design choices by bundling hardware and software pieces together while making the high performance more predictable.

Intel® Verified Reference Configurations for NFVI v4 are based on a multi-node architecture that consists of a controller, a cloud node and storage at a minimum. Thus, it is expected that the Intel® Verified Reference Configuration for NFVI provides all required resources to implement a software-defined infrastructure that resides within each cloud server instance and is controlled by the hypervisor. The Controller Node is intended to be used for control, signaling and management implementing the Virtual/Container Network Function (VNF) Management (VNFM) and Virtualization Infrastructure Management (VIM). Thus it may not require additional local storage and hardware acceleration.

Figure 1. Intel® Verified Reference Configurations for NFVI Environment



All Intel® Verified Reference Configurations feature a workload-optimized stack tuned to take full advantage of an Intel® Architecture (IA) foundation. To be certified as an Intel® Verified Reference Configuration, Original Equipment Manufacturer (OEM) systems must meet a performance threshold that represents a premium customer experience.

There are two configurations for Intel® Verified Reference Configurations for NFVI reference designs for the Cloud Node:

- Intel® Verified Reference Configurations for NFVI Plus configuration for the Cloud Node is defined with at least a 32-core 4th Generation Intel® Xeon® Scalable processor and high-performance network, with storage and integrated platform acceleration products from Intel® for maximum virtual machine density.
- Intel® Verified Reference Configurations for NFVI Base configuration for the Cloud Node is defined with a 24-core or higher 4th Generation Intel® Xeon® Scalable processor and network, with storage and add-in platform acceleration products from Intel® targeting for optimized value and performance-based solutions.

There is also a configuration for Intel® Verified Reference Configurations for NFVI reference designs for the Controller Node:

Intel® Verified Reference Configurations for NFVI configuration for the Controller Node for NFVI is defined with at least a 24-core or higher 4th Generation Intel® Xeon® Scalable processor and network adapters to be able to communicate with all the cloud nodes. Application storage and add-in platform acceleration products may not be required for bare bone controller functionality to manage and communicate with Cloud nodes.

Bill of Materials (BOM) requirement details for the configurations are provided later section of this document.

Intel® Verified Reference Configurations for NFVI are defined in collaboration with Communication Service Provider and ecosystem partners to demonstrate the value of an I/O Balanced Architecture for Network Function Virtualization. The solution leverages the hardened hardware, firmware, and software to allow customers to integrate on top of this known good foundation.

With the steady rise of cloud-based network traffic, edge computing has become increasingly more important to process this larger amount of data efficiently. By processing data at the network edge, latency for applications and services is reduced while potentially heavy traffic loads at the data center level, are alleviated by the distribution of traffic among the edge branches. However, this expanded network radius also corresponds to a wider attack range for a potential security threat. With these challenges in mind, Secure Access Service Edge (SASE) is intended to deliver a comprehensive solution for providing strong computational capability at the edge, securing the larger service area, and easily deploying new branches of the network.

Intel® Verified Reference Configuration for NFVI with SASE provides numerous benefits to ensure end users have excellent performance for their network applications. Some of the key benefits of the Intel® Verified Reference Configuration based on the 4th Generation Intel® Xeon® Scalable Processor Family processor include:

- High core counts and per-core performance
- Compact, power-efficient system-on-chip (Soc) platform
- Streamlined path to cloud-native operations
- Accelerated AI inference
- Accelerated encryption and compression
- Platform-level security enhancements

Figure 2. SASE Use Cases

1.1 Terminology

Table 1. Terminology

Term	Description
AIC	Add-In Card
AMX	Advanced Matrix Extensions
API	Application Program interface
AGF	Access Gateway Function
BIOS	Basic Input/Output System
BOM	Bill of Materials
BtG	Boot Guard Technology
CUPS	Control Plane and User Plane Separation
DC	Data Center
DIMM	Dual Inline Memory Module
DPDK	Data Plane Development Kit
DRAM	Dynamic Random Access Memory
DUT	Device Under Test
ECDH	Elliptic Curve Diffie-Hellman protocol
ECDSA	Elliptic Curve Digital Signature Algorithm
GbE	Gigabit Ethernet
HQoS	Hierarchical Quality of Service
Intel® QAT	Intel® QuickAssist Technology
Intel® TXT	Intel® Trusted Execution Technology
Intel® UPI	Intel® Ultra Path Interconnect
Intel® VT	Intel® Virtualization Technology
MLC	Memory Latency Checker
NAT	Network Address Translation
NFVI	Network Function Virtualization Infrastructure
NIC	Network Interface Controller

Term	Description
NUMA	Non-Uniform Memory Access
NVMe*	Non-Volatile Memory Express*
OAM	Operation, Administration and Management
OCP	Open Compute Project
OEM	Original Equipment Manufacturer
PCIe*	Peripheral Component Interconnect express*
PHY	Physical Layer
PXE	Pre-boot Execution Environment
QinQ	A standard that allows multiple VLAN headers in an Ethernet frame
RAS	Reliability, Availability, and Serviceability
SASE	Secure Access Service Edge
SR-IOV	Single Root Input/Output Virtualization
SSD	Solid State Drive
TCO	Total Cost of Ownership
TPM	Trusted Platform Module
Intel® TXT	Intel® Trusted Execution Technology
TPM	Trusted Platform Module
Intel® UPI	Intel® Ultra Path Interconnect
VIM	Virtualization Infrastructure Management
VMX	Virtual Machine Extension
VNFM	Virtual Network Function Management
VRC	Verified Reference Configuration
Intel® VT	Intel® Virtualization Technology

1.2 Reference Documents and Resources

Table 2. Reference Documents and Resources

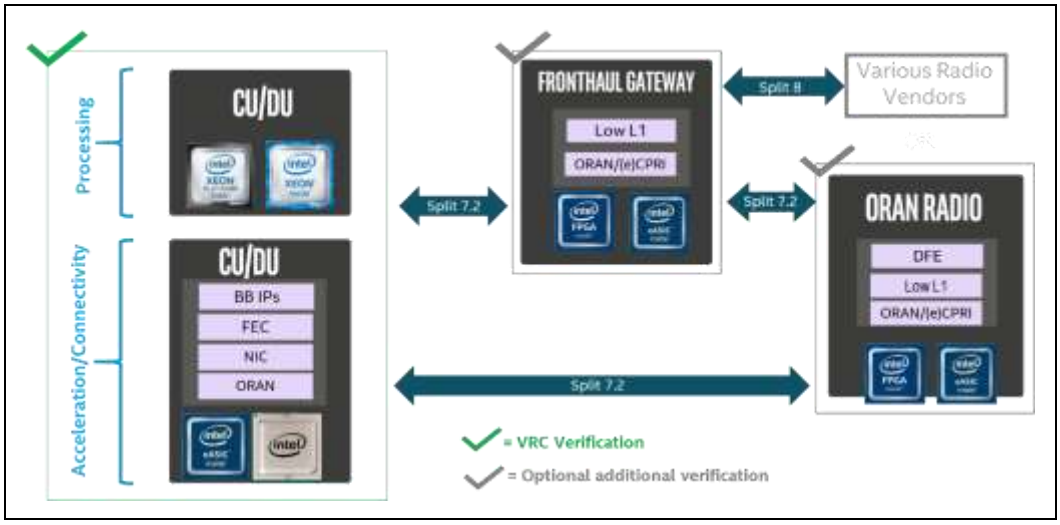
Document	Document Number/Location
Intel® QuickAssist Technology Software for Linux* - Getting Started Guide – HW Version 2.0	https://developer.intel.com/quickassist
Intel® Network Platform Xeon-SP (NPX-SP) VRC for NFVI High Level Design Specification	736017
Kubernetes Documentation Homepage	https://kubernetes.io/docs/home/

Document	Document Number/Location
Intel® Ethernet Connection E810 Application Device Queues (ADQ) Configuration Guide	609008
BIOS Settings for Intel® Wireline, Cable, Wireless and Converged Access Platform	https://cdrdv2.intel.com/v1/dl/getContent/747130
Intel® Data Streaming Accelerator User Guide	353216

2 Solution Components

Intel® Verified Reference Configuration consists of select hardware, various Intel® Xeon® processor technologies along with optimized software and firmware configurations. It consists of the components below.

Figure 2. Solution Overview



2.1 Intel® Xeon® Processor Scalable Performance Family

Intel® Xeon® Scalable processors are designed to accelerate performance across the fastest-growing workloads. These processors have the most built-in accelerators of any CPU on the market to help maximize performance efficiency for emerging workloads, especially those powered by AI.

In addition to delivering outstanding general-purpose performance, Intel® Xeon® drives efficiency with built-in accelerators. Data center operators can leverage built-in AI, telemetry, and power management tools for intelligent control electricity usage.

Intel's innovative workload accelerators enable end users to do more with less reducing TCO by delivering performance, power, resource, and cost efficiency as well as providing advanced security technologies.

The 4th Gen Intel® Xeon® Scalable Processors (formerly codenamed Sapphire Rapids) are the latest processors for Datacenter workloads that offer:

- **Enhanced Per Core Performance** with up to 60 cores in a standard socket
- **Enhanced Memory Performance** with support for up to 4800MT/s DIMMs (2 DPC)
- Increased Memory Capacity with up to 8 channels
- **Breakthrough System Memory & Storage** with Intel® Optane™ persistent memory 200 series

- **Built-in AI Acceleration** with enhanced performance of Intel® Deep Learning Boost
- **Faster UPI** with 3 Intel® Ultra Path Interconnect (Intel® UPI) at 11.2 GT/s
- **More, Faster I/O** with PCI Express 4 and up to 64 lanes (per socket) at 16 GT/s
- **New Hardware-Enhanced Security** delivering security technologies leadership with Intel® SGX, Intel® TME, Intel® PFR etc.
- **Enhanced Intel® Speed Select Technology (Intel® SST)** with three capabilities supported on the majority of Gold CPUs

2.2 Intel® C741 Chipset (codenamed: Emmitsburg) Platform Controller Hub

The Emmitsburg PCH provides extensive I/O support. The functions and capabilities are as follows:

- ACPI Power Management Logic Support, Revision 4.0a
- PCI Express Base Specification, Revision 4.0
- Integrated Serial ATA host controller supports data transfer rates of up to 6 b/s on all ports.
- xHCI USB controller with 10 USB 3.2 Gen 1 and 14 USB2 ports
- Serial Peripheral Interface
- Enhanced Serial Peripheral Interface
- Flexible I/O—Allows some high speed I/O signals to be configured as PCIe root ports, SATA, or USB 3.2 Gen 1
- General Purpose Input Output (GPIO)
- Interrupt controller, and timer functions
- System Management Bus Specification, Version 2.0
- Integrated Clock Controller/Real Time Clock Controller
- Intel® High-Definition Audio (Intel® HD Audio)
- Integrated 10/100/1000 Mbps Ethernet MAC
- Supports Intel® Rapid Storage Technology enterprise (Intel® RSTe)
- Supports Intel® Active Management Technology and Intel® Server Platform Services (Intel® SPS)
- Supports Intel® Virtualization Technology (Intel® VT) for Directed I/O (Intel® VT-d)
- Supports Intel® Trusted Execution Technology (Intel® TXT)
- JTAG Boundary Scan support
- Intel® Trace Hub (Intel® TH) for debug
- ADR Support

2.3 Intel® Ethernet 800 Series

Intel® Ethernet 800 Series offers:

- **Higher Bandwidth** as Intel's first NIC with PCIe* 4.0 and 50Gb PAM4 SerDes
- **Improved Application Efficiency** with Application Device Queues (ADQ), Dynamic Device Personalization (DDP)
- **Versatility** with Flexible speeds: 2x100/50/25/10GbE, 4x25/10GbE, or 8x10GbE
- **RDMA support** for both iWARP and RoCEv2 providing a choice in hyper-converged networks

2.3.1 Intel® Network Adapter with Data Plane Development Kit (DPDK)

Intel® Network Products offer continuously innovative solutions for high throughput and performance for networking infrastructure. The Intel® Network Adapter with Data Plane Development Kit (DPDK) provides highly optimized Network Virtualization and fast data path packet processing. DPDK offers many use cases that are hardened on this Intel® Accelerated Solution for NFVI Forwarding Platform.

2.3.2 Intel® Ethernet 800 Series Dynamic Device Personalization (DDP)

Dynamic Device Personalization (DDP) usage to reconfigure network controllers for different network functions on-demand, without the need for migrating all VMs from the server, avoids unnecessary loss of compute for VMs during server cold restart. It also improves packet processing performance for applications/VMs by adding the capability to process new protocols in the network controller at run-time.

This kind of on-demand reconfiguration is offered in the Intel® Ethernet 800 Series NICs.

Dynamic Device Personalization describes the capability of the Intel® Ethernet 800 Series devices to load an additional firmware profile on top of the device's default firmware image, enabling parsing and classification of additional specified packet types that can be distributed to specific queues on the NIC's host interface using standard filters. Software applies these custom profiles in a non-permanent, transaction-like mode so that the original network controller's configuration is restored after NIC reset or by rolling back profile changes by software. Using APIs provided by drivers, personality profiles can be applied by the DPDK. Support for kernel drivers and integration with higher level management/orchestration tools is in progress.

DDP can be used to optimize packet processing performance for different network functions, native or running in virtual environment. By applying a DDP profile to the network controller, the following use cases could be addressed.

A general purpose, OS-default DDP package is automatically installed with all supported Intel® Ethernet Controller 800 Series drivers on Microsoft* Windows*, ESX*, FreeBSD*, and Linux* operating systems. Additional DDP packages are available to address needs for specific market segments. For example, a

telecommunications (Comms) DDP package is available to support certain market-specific protocols in addition to the protocols in the OS-default package.

- The OS-default DDP package supports the following:
  - MAC, EtherType, VLAN
  - IPv4, IPv6, TCP, ARP, UDP
  - SCTP, ICMP, ICMPv6, CTRL
  - LLDP, VXLAN-GPE, VXLAN (non-GPE), Geneve, GRE, NVGRE, RoCEv2
  - MPLS (up to 5 consecutive MPLS labels in the outermost Layer 2 header group)
- In addition to the previous list, the Comms DDP package also supports the following protocols:
  - GTP
  - PPPOE
  - L2TPv3
  - IPSec
  - PFCP

2.4 Intel® Xeon® Scalable Platform Technologies

2.4.1 Intel® Hyper-Threading Technology (Intel® HT Technology)

Enables multiple threads to run on each core, which ensures that systems use processor resources more efficiently. Intel® HT Technology also increases processor throughput, improving overall performance on threaded software.

2.4.2 Intel® Turbo Boost Technology

Accelerates processor and graphics performance for peak loads, automatically allowing processor cores to run faster than the rated operating frequency if they're operating below power, current, and temperature specification limits.

2.4.3 Intel® Speed Select Technology (Intel® SST)

Intel® Speed Select Technology is a collection of features that improve performance and optimize TCO by providing more control over CPU performance. With Intel® SST, one server can do more.

- **Intel® SST- BF**: Trade off base frequency between higher and lower priority cores
- **Intel® SST- TF**: Capability to configure turbo frequencies beyond traditional max frequency limit
- **Intel® SST-CP**: Capability to allocate Surplus frequency to highest priority cores
- **Intel® SST-PP**: Capability to configure the CPU to run at 3 distinct operating points or profiles

2.4.4 Intel® Virtualization Technology (Intel® VT)

Intel® Virtualization Technology (Intel® VT) provides hardware abstraction to allow multiple workloads to co-exist and share common resources while maintaining full isolation.

The Intel® VT portfolio includes the capability in the CPU to implement Virtual Machine Extension (VMX) instructions. These allow all software in the Virtual Machine to run natively without performance impact from operations such as instruction translation and page table swaps with memory virtualization support.

Intel® VT also includes input/output (I/O) virtualization that supports offloading packet processing to network adapters, directly assigning virtual functions to virtual machines with Single Root I/O Virtualization (SR-IOV) and Intel® Data Direct I/O Technology Enhancement (Intel® DDIO), providing native network performance in the VM.

To obtain all the benefits of Intel® VT, Intel® Verified Reference Configurations for Network Function Virtualization Infrastructure (NFVI) are required to have all virtualization technology features enabled.

2.4.5 Intel® Resource Director Technology (Intel® RDT)

Intel® Resource Director Technology (Intel® RDT) are collection of advanced technologies providing control over shared platform resources

Cache Monitoring Tech (CMT): Per-thread L3 Occupancy Monitoring

Cache Allocation Tech (CAT): Per-thread L3 Occupancy Control, New Code/Data Prioritization (CDP) extension

Memory BW Monitoring (MBM): Per-thread Memory Bandwidth Monitoring

Memory Bandwidth Allocation (MBA): Per-core Bandwidth Control – more precisely control "noisy neighbors"

2.4.6 Intel® On Demand

Intel® On Demand (formerly SDSI) is a new capability offering on demand one-time activation of incremental features during the Intel® Xeon CPU Life Cycle. Once enabled through a backend infrastructure, a hardened SW architecture for license delivery, with automated billing. License provisioning on the platform via out of band and in band communication for the one-time activation of features. CPU Features include: SGX, IAA, QAT, DLB, DSA.

2.4.7 Intel® Accelerator Interfacing Architecture (AiA)

Accelerator Interfacing Architecture (AiA) includes work submission, new instructions that provides low latency user-space work dispatch and synchronization between software and accelerators. AiA improves overall performance of the system for use in conjunction with the integrated accelerators in CPU, as well as discrete accelerators like NIC, GPU, FPGA-instantiated accelerators.

2.4.8 Intel® Deep Learning Boost

Intel® Deep Learning Boost includes extensive hardware inside CPU to provide set of instructions to accelerate Artificial Intelligence (AI) or Deep Learning Inference and training workload. Instructions include intel Deep Learning Boost i.e. VNNI/INT8, BF16 and with 4th Gen Intel® Xeon Scalable Processor the new Intel® AMX/TMUL instructions with INT8 and BF16 support.

2.4.9 Intel® Dynamic Load Balancer (DLB)

Intel® Dynamic Load Balancer previously known as Hardware Queue Manager improves the system performance related to handling network data on multi-core Intel® Xeon Scalable processors. Intel® DLB is a hardware accelerator that provides comprehensive scheduling support for data that need to be sent to CPU Cores using fully virtualized producer to consumer queuing. It enables efficient core-to-core communication via the following methods:

1. Distributed Processing - Intel® DLB enables the efficient distribution of network processing across multiple CPU cores/threads.
2. Dynamic Load Balancing - Intel® DLB dynamically distributes network data across multiple CPU cores for processing as the system load varies.
3. Dynamic Network Processing Reordering- Intel® DLB restores the order of networking data packets processed simultaneously on CPU cores.

2.4.10 Next-Gen Intel® QuickAssist Technology (Intel® QAT)

This product contains Intel® QuickAssist Technology which is integrated into the Processor with the following functions:

- Cryptographic Functions
  - Cipher Operations
  - NULL, AES
  - Snow3G UEA2
  - ZUC/128-EEA3(64B/1024B)
  - SM4
  - AES-GCM: Single Pass
  -
  - CHACHA20-POLYHash Operation
  - NULL, SHA-1
  - SHA-224/256
  - SHA-384/512
  - AES-(X)CBC-MAC
  - Galois Hash 64/128
  - UIA2 (Snow3G)
  - SHA3-224(64B/1024B)
  - SHA3-256(64B/1024B)
  - SHA3-384(64B/1024B)
  - SHA3-384(64B/1024B)
  - SHA3-512(64B/1024B)
  - ZUC/EIA3(64B/1024B)
  - SM3(64B/1024B)
- Authentication Operation
  - SHA1, SHA-256, SHA-512, SHA-224, SHA-384, SHA3-256, SHA3-512,

  - SHA3-224, SHA3-384, SM3, All HMAC variations
  - AES-CBC-MAC, AES-XCBC-MAC, GHASH64 (GMAC), GHASH128 (GMAC)
- Wireless Authentication Operation
  - AES-CBC-MAC, AES F9, SNOW3G UIA2, ZUC (128-EIA3)
- Cipher-Hash Combined Operation
- Key Derivation Operation
- Wireless Cryptography
  - AES, SM4, SNOW 3G*(UEA2), ZUC(128-EEA3)
  - Public Key Functions
- RSA Operation
- Diffie-Helman Operation
- Digital Signature Standard Operation
- Key Derivation Operation
- Elliptic Curve Cryptography: ECDSA* and ECDH*
  - Compression/Decompression Functions
- DEFLATE, LZ4s, LZ4

2.4.11 Intel® Data Streaming Accelerator (DSA)

4th Gen Intel® Xeon® Scalable Processors incorporate Intel® Data Streaming Accelerator Technology version 1.0 through a data accelerator for improving the performance of storage, networking, and various other I/O applications. The DMA engine is optimized for moving data between memory. The Intel® Data Streaming Accelerator replaces Intel® QuickData Technology used in previous Server processor generations. The goal of Intel® DSA is to provide higher overall system performance for data mover and transformation operations, while freeing up CPU cycles for higher level functions. For more details, refer to Intel® Data Streaming Accelerator Specification, https://software.intel.com/content/www/us/en/develop/articles/intel-data-streamingaccelerator-architecture-specification.html

2.4.12 Intel® In-Memory Analytics Accelerator (Intel® IAA)

Intel® IAA **increases query throughput** and **decreases memory footprint** via:

1. Deeper compression than software-only techniques
2. More effective bandwidth, as deeply compressed data consumes less bandwidth
3. Core offload, as IAA performs computationally demanding scan and filter operations in place of cores.

2.4.13 Intel® Volume Management Device (VMD) & Intel® Virtual RAID on CPU (VROC)

Intel® VMD and Intel® VROC together provide RAS features for NVMe storage with maximum system up-time, ease of maintenance, cost-effective RAID solution and better NVMe performance in virtualization applications.

2.4.14 Intel® Seamless Firmware Update Technology

Intel® Seamless Firmware Update Technology is a cutting-edge solution from Intel that updates microcode with no perceived degradation to the services running on the platform. This technology can aggregate Intel firmware updates during run time without interrupting system operation. Also. it maintains server uptime while updating and activating firmware components.

2.4.15 Hardware Enhanced Security Features

4th Gen Intel® Xeon® Scalable Processors offer new hardware enhanced security features:

- **Intel® Software Guard Extensions (Intel® SGX) with additional integrity feature:** Provides fine grain data protection via application isolation in memory. Integrity provides greater resistance against physical attacks
- **Intel® Platform Firmware Resilience (Intel® PFR):** Verification of platform firmware images now extended to peripherals
- **Software Hardening Execution Controls**: Intel Virtualization Technology-Redirect Protection (formerly HLAT) - Enhanced page-table protections for OS kernel. Control flow Enforcement Tech (CET) - Return Oriented Programming (ROP), Jump Oriented Programming (JOP), Call Oriented Programming (COP) attack prevention with Shadow Stack and ENDBRANCH.
- Intel® Total Memory Encryption (**Intel® TME-MK**): Hardware-enabled memory encryption designed for multi-tenant server platforms. Also supports full memory encryption via single key

2.4.16 Trusted Platform Module (TPM)

TPM protects the system start-up process by ensuring it is tamper-free before releasing system control to the operating system. TPM 1.2 also provides secured storage for sensitive data, such as security keys and passwords, and performs encryption and hash functions.

Intel® Trusted Execution Technology (Intel® TXT) utilizes this technology.

2.4.17 Intel® Trusted Execution Technology (Intel® TXT)

Intel® TXT provides the foundation for highly scalable platform security in physical and virtual infrastructures. It helps harden servers at the hardware level against threats of hypervisor, BIOS, or other firmware attacks, malicious rootkit installations, and other types of attacks or misconfiguration to firmware and operating systems.

2.4.18 Intel® Hyper-Threading Technology (Intel® HT Technology)

Intel® HT Technology enables multiple threads to run on each core, which ensures that systems use processor resources more efficiently. Intel® HT Technology also increases processor throughput, improving overall performance on threaded software.

2.4.19 Intel® Boot Guard (Security)

Hardware-based boot integrity protection prevents unauthorized software and malware takeover of boot blocks critical to a system's function, thus providing added level of platform security based on hardware.

3 Design Compliance Requirements

This chapter focuses on the design requirements for Intel® Verified Reference Configuration for NFVI.

3.1 Verified Reference Configuration Hardware Requirements

The checklists in this chapter provide guidance for assessing the conformance to the Intel® Verified Reference Configuration for NFVI hardware platform requirement for the Cloud Node Base Configuration, Cloud Node Plus Configuration, Controller Node. For the platform to conform to the desired Intel® Verified Reference Configuration for NFVI, all requirements in the checklist must be met.

Table 3. Intel® Verified Reference Configuration for NFVI Plus Configuration- the Cloud Node HW Configuration

Ingredient	Requirement	Required/ Recommended	Quantity
Processor	Intel® Xeon® Gold 6438N Processor at 1.8GHz, 32C/64T, 205W or higher number SKU	Required	2
Memory	Option 1: DRAM only configuration: 512 GB (16x 32 GB DDR5, 4800 MHz)	Required	16
	Option 2: DRAM only configuration: 512 GB (32x 16 GB DDR5, 4800 MHz)		32
Network [1]	Intel® Ethernet Network Adapter E810-2CQDA2	Required	4 (2 per NUMA node)
Storage (Boot Drive)	480 GB or equivalent boot drive	Required	1
Storage (Capacity)	3.84 TB or equivalent drive (recommended NUMA aligned)	Recommended	4 (2 per NUMA node)
LAN on Motherboard (LOM)	10 Gbps or 25 Gbps port for Pre-boot Execution Environment (PXE) and Operation, Administration and Management (OAM)	Required	4 (2 per NUMA node)
	1/10 Gbps port for Management NIC	Required	1

Table 4. Intel® Verified Reference Configuration for NFVI Base Configuration- the Cloud Node HW Configuration

Ingredient	Requirement	Required/ Recommended	Quantity
Processor	Intel® Xeon® Gold 5418N processor at 1.8 GHz, 24C/48T, 165W or higher number SKU	Required	2
Memory[1]	DRAM only configuration: 256 GB (16x 16 GB DDR5, 4800 MHz)	Required	16
Network [2]	Option 1 – Intel® Ethernet Network Adapter E810-CQDA2	Required	4 (2 per NUMA node)
	Option 2 – Intel® Ethernet Network Adapter E810-2CQDA2		2 (1 per NUMA node)
Storage (Boot Drive)	480 GB or equivalent boot drive	Required	2
Storage (Capacity)	3.84 TB or equivalent drive (recommended NUMA aligned)	Recommended	2 (1 per NUMA node)
LAN on Motherboard (LOM)	10 Gbps or 25 Gbps port for PXE/OAM	Required	2
	1/10 Gbps port for Management NIC	Required	1

Table 5. Intel® Verified Reference Configuration for NFVI – Controller Node HW Configuration

Ingredient	Requirement	Required/ Recommended	Quantity per Node
Processor[1]	Intel® Xeon® Gold 5418N processor at 1.8 GHz, 24C/48T, 165W (SST-PP config 2) or higher number SKU	Required	2
Memory[2]	DRAM only configuration: 256 GB (16x 16 GB DDR5, 4800 MHz)	Required	16
Network [3]	Option 1 – Intel® Ethernet Network Adapter E810-CQDA2	Required	2
	Option 2 – Intel® Ethernet Network Adapter E810-CQDA2 with Ethernet Port Configuration Tool (EPCT) to break down interface to 4x25G on single port or 8x10G mode (4x10 on single port)		
	Option 3 – Intel® Ethernet Network Adapter E810-XXVDA2		
Storage (Boot Drive)	480 GB or equivalent boot drive	Required	2
Storage (Capacity)	3.84 TB or equivalent drive (recommended NUMA aligned)	Required	2 (1 per NUMA node)
LAN on Motherboard (LOM)	10 Gbps or 25Gbps port for PXE/OAM	Recommended	2
	1/10 Gbps port for Management NIC	Required	1

3.2 Verified Reference Configuration Software Requirements

The table below is a guide for assessing the conformance to Intel Accelerated Solution for NFVI or SASE Intel Accelerated Solution for NFVI or SASE software requirements.

For the platform to conform the desired Intel Accelerated Solution for NFVI or SASE, all requirements listed in the checklist below must be satisfied.

Table 6. Intel Accelerated Solution for NFVI or SASE – SW Configuration

Ingredient	SW Version Details
DPDK	21.11.2
Intel® QAT	QAT 20.L.1.0.10-00005
Intel® Ethernet Network Adapter E810-CQDA2	CVL 4.10 0x800151a9 ice 4.18.0-372.40.1.el8_6.x86_64 iavf 4.18.0-372.40.1.el8_6.x86_64
Intel® Ethernet Network Adapter E810-2CQDA2	CVL 4.10 0x800151a9 ice 4.18.0-372.40.1.el8_6.x86_64 iavf 4.18.0-372.40.1.el8_6.x86_64
Red Hat	OpenShift version 4.12.1
Node Feature Discovery Operator	4.10.6
SR-IOV Network Operator	4.12.0-202303231115
Intel Device Plugins Operator	0.24.1
Kernel	4.18.0-372.40.1.el8_6.x86_64

Note:

1. Intel® recommends checking your system's exposure to the "Spectre" and "Meltdown" exploits.
2. Intel® QAT Software Package available from https://developer.intel.com/quickassist
3. The Intel® QAT driver is required to achieve the performance KPI's needed for certification. In the event of an issue, Red Hat* may request the user reproduce the issue with the Intel® QAT In-Tree driver to rule out interaction with the Intel® QAT driver from https://developer.intel.com/quickassist. Refer to https://access.redhat.com/articles/1067 which explains Red Hat* Support policy for Out Of Tree (OOT) drivers.
4. The customer should be aware of Red Hat's Certified Guest Operating System policy here (tier 3 vs. tier 1): https://access.redhat.com/articles/973163

5. Intel® Ethernet Adapter E810-CQDA2 Non-Volatile Memory (NVM) Update Utility for Intel® Ethernet Network Adapter 810 Series can be found at the following link: https://www.intel.com/content/www/us/en/download/19624/non-volatile-memory-nvm-update-utility-for-intel-ethernet-network-adapter-e810-series.html?wapkw=nvm%20update

3.3 BIOS Settings

To meet the performance requirements for an Intel® Verified Reference Configuration for NFVI platform solution, Intel® recommends using the BIOS settings for enabling processor p-state and c-state with Intel® Turbo Boost Technology ("turbo mode") enabled. Hyper threading is recommended to provide higher thread density. Intel® also recommends using the BIOS settings for on demand Performance with power consideration.

Refer to the document *BIOS Settings for Intel® Wireline, Cable, Wireless and Converged Access Platform (#747130)* ***chapter 3*** for information on the BIOS settings.

Note: BIOS settings differ from vendor to vendor. Please contact your Intel Representative if you do not see the exact setting in your BIOS.

3.4 Platform Technology Requirements

This section lists the requirements for Intel's advanced platform technologies.

NFVI requires Intel® VT and Intel® Scalable I/O Virtualization (Intel® Scalable IOV) to be enabled to reap the benefits of hardware virtualization. Either Intel® Boot Guard or Intel® Trusted Execution Technology establishes the firmware verification, allowing for platform static root of trust.

Table 7. Platform Technology Requirements

Platform Technologies		Enable/Disable	Required/Recommended
Intel® VT	Intel® CPU VMX Support	Enable	Required
	Intel® I/O Virtualization	Enable	Required
Intel® Boot Guard	Intel® Boot Guard	Enable	Required
Intel® TXT	Intel® Trusted Execution Technology	Enable	Recommended

3.5 Platform Security

Intel Accelerated Solutions for vRAN must enable Intel® Boot Guard Technology to verify that the platform firmware is suitable during the boot phase.

In addition to protecting against the known attacks, all Intel Accelerated Solutions recommend installing the Trusted Platform Module (TPM). The TPM enables administrators to secure platforms for a trusted (measured) boot with known trustworthy (measured) firmware and OS. This allows local and remote verification by third parties to advertise known safe conditions for these platforms through implementation of Intel® Trusted Execution Technology (Intel® TXT).

3.6 Side Channel Mitigation

Reference architecture protection is verified with Spectre and Meltdown exposure using the latest Spectre and Meltdown Mitigation Detection Tool, which confirms the effectiveness of firmware and operating system updates against known attacks.

§

4 Platform Tuning for Worker Node

4.1 Boot Parameter Setup

For the workload testing, it is first necessary to setup the host command line with appropriate boot parameters as well as 1GB hugepages. In the "/etc/default/grub" file, update the line "GRUB_CMDLINE_LINUX" to include the following parameters:

"intel_iommu=on iommu=pt default_hugepagesz=1G hugepagesz=1G hugepages=50 sm=on iova_sl"

After modifying in grub file, run "update-grub" and "reboot" to apply the changes and verify the change with "cat /proc/cmdline":

```
cat /proc/cmdline

BOOT_IMAGE=/vmlinuz-5.15.0-71-generic root=UUID=06ea0cd2-8228-454b-bb0b-
f2abe7a32664 ro intel_iommu=on iommu=pt default_hugepagesz=1G hugepagesz=1G
hugepages=50 sm=on iova_sl
```

4.2 Building QAT Driver

Follow the instructions provided below to build the QAT2.0 OOT Driver:

1. Download the QAT2.0 driver (QAT20.L.1.0.20-00008.tar.gz) from https://www.intel.com/content/www/us/en/developer/topic-technology/open/quick-assist-technology/overview.html by accessing the Linux* Hardware v2.0 driver page.

2. Run the following commands to install the driver:

```
    apt-get install -y gcc gcc-c++ systemd-devel pciutils kmod

    mkdir /root/QAT
    mv QAT20* /root/QAT
    cd /root/QAT
    tar -xf *
    ./configure --enable-icp-sriov=host
    make
    make install
```

4.3 Docker Installation

Follow the instructions below to install Docker:

```
    apt-get install -y ca-certificates curl gnupg lsb-release
    curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

    echo "deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/docker-archive-keyring.gpg]
```

```
https://download.docker.com/linux/ubuntu  $(lsb_release -cs) stable" | sudo
tee /etc/apt/sources.list.d/docker.list > /dev/null

    apt-get update -y
    apt-get install -y docker-ce=5:20.10.13~3-0~ubuntu-jammy docker-ce-
cli=5:20.10.13~3-0~ubuntu-jammy containerd.io

    mkdir -p /etc/docker

    cat > /etc/docker/daemon.json <<EOF
    {
    "exec-opts": ["native.cgroupdriver=systemd"],
    "log-driver": "json-file",
    "log-opts": {
    "max-size": "100m"
    },
    "storage-driver": "overlay2",
    "storage-opts": [
    "overlay2.override_kernel_check=true"
    ]
    }
    EOF

    systemctl enable docker
    systemctl daemon-reload
    systemctl restart docker
```

4.4 Kubernetes - kubeadm* Installation

4.4.1 Install Kubernetes

Follow the instructions below to install Kubernetes:

```
    apt-get remove curl
    apt install curl

    curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo
gpg --dearmor -o /etc/apt/keyrings/kubernetes-archive-keyring.gpg

    echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg]
https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list

    apt update -y
    apt install -y kubectl=1.21.2-00 kubeadm=1.21.2-00 kubelet=1.21.2-00 --
allow-downgrades

    systemctl enable --now kubelet
    systemctl start kubelet

    cat <<EOF > /etc/sysctl.d/k8s.conf
    net.bridge.bridge-nf-call-ip6tables = 1
    net.bridge.bridge-nf-call-iptables = 1
    EOF

    sysctl --system
```

4.4.2 Kubernetes Cluster Initialization

Turn swap off if not yet off:

```
    swapoff -a
    # enable swap off permanently, modify /etc/fstab, comment out the last
line
    # /swap.img none swap sw 0 0
```

To initialize the K8s on the controller node, run the following command:

```
    kubeadm init --kubernetes-version=v1.21.2 --pod-network-cidr=10.244.0.0/16
--apiserver-advertise-address=10.10.10.107 --token-ttl 0 --ignore-preflight-
errors=SystemVerification
```

To start using the cluster, you need to run the following as a regular user

```
    cp /etc/kubernetes/admin.conf $HOME/

    chown $(id -u):$(id -g) $HOME/admin.conf

    export KUBECONFIG=$HOME/admin.conf

    echo 'export KUBECONFIG=$HOME/admin.conf' >> $HOME/.bashrc
```

Deploy a pod network to the cluster

```
    #Download the yaml file for calico plugin
    wget https://docs.projectcalico.org/v3.18/manifests/calico.yaml --no-
check-certificate

    #Setup calico network
    kubectl apply -f calico.yaml

    #To see if all system PODs are up and running
    kubectl get pods -A
```

4.4.3 Kubernetes Cluster Reinitialization

Follow the instructions below to reinitialize Kubernetes Cluster:

```
    kubeadm reset
    reboot
```

4.4.4 Kubernetes Initialization – Master/Worker Node (on the same machine)

Follow the instructions below to allow K8 master node to be used both as master and worker node on a single server:

```
    kubectl get nodes
    kubectl describe nodes <nodename> | grep -i taint
    kubectl taint nodes --all node-role.kubernetes.io/master-
```

5 Performance Verification

This chapter aims to verify the performance metrics for the reference configuration for NFVI to ensure that there is no anomaly seen. Refer to information in this chapter to ensure that the performance baseline for the platform is as expected.

The Plus solution was tested on July 24, 2023, with the following hardware and software configurations:

- 2 NUMA nodes
- 2x Intel® Xeon® Gold 6438N processors
- Total Memory: 512 GB, 32 slots/16 GB/4800 MT/s DDR5 RDIMM
- Hyperthreading: Enable
- Turbo: Enable
- C-State: Enable
- Storage: 894.3G INTEL SSDSC2KG96
- Network devices: 4x Dual port Intel® Ethernet Network Adapter E810-2CQDA2
- Network speed: 50 GbE
- BIOS: EGSDCRB1.86B.0037.P02.2305260955
- Microcode: 0xab0004b1
- OS/Software: Ubuntu 22.04.2 LTS (kernel 5.15.0-71-generic)

The Base solution was tested on July 24, 2023, with the following hardware and software configurations:

- 2 NUMA nodes
- 2x Intel® Xeon® Gold 5418N processors
- Total Memory: 256 GB, 16 slots/16 GB/4800 MT/s DDR5 RDIMM
- Hyperthreading: Enable
- Turbo: Enable
- C-State: Enable
- Storage: 447.1G INTEL SSDSC2KB48
- Network devices: 2x Dual port Intel® Ethernet Network Adapter E810-2CQDA2
- Network speed: 50 GbE
- BIOS: American Megatrends International, LLC. 3A15.TEL3P1
- Microcode: 0x2b000461
- OS/Software: Ubuntu 22.04 LTS (kernel 5.15.0-76-generic)

5.1 Memory Latency Checker (MLC)

The first application is the Memory Latency Checker which can be downloaded from https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html

Download the latest version and execute this application, unzip the tarball package and go into Linux* folder and execute `./mlc or ./mlc_avx512`.

Table 8. Memory Latency Checker

Key Performance Metric	Local Socket (Plus)	Local Socket (Base)
Idle Latency	100 ns	110 ns
Memory Bandwidths between nodes within the system (using read-only traffic type) MB/s	200000	190000

Table 9. Peak Injection Memory Bandwidth (1 MB/sec) Using All Threads

Peak Injection Memory Bandwidth (1 MB/sec) using all threads	Plus	Base
All Reads	410000	380000
3:1 Reads-Writes	390000	340000
2:1 Reads-Writes	350000	336000
1:1 Reads-Writes	370000	321000
STREAM-Triad	360000	307000
Loaded Latencies using Read-only traffic type with Delay=0 (ns)	220	200
L2-L2 HIT latency (ns)	62	69
L2-L2 HITM latency (ns)	63	70

Note: If the latency performance and memory bandwidth performance is outside the range, please verify the validity of the Platform components, BIOS settings, kernel power performance profile used and other software components.

5.2 NGINX*

Intel® QAT hardware acceleration helps to offload public key exchange for SSL layer 7 application. Intel® Verified Reference Configuration for NFVI – Plus platform and Base platform should be able to demonstrate approximately 10500 Connection Per Second (CPS) with the full software stack of the NGINX* application with Intel® QAT HW (2 devices).

Intel® Verified Reference Configuration for NFVI - Plus platform and Base platform without Intel® QAT should demonstrate approximately 800 Connections Per Second (CPS) with the out-of-box software stack of the NGINX* application.

Table 10. NGINX* Workload Configuration

Ingredient	Software Version Details
Async mode nginx	v0.5.0
OpenSSL	1.1.1m
QAT Engine	v1.1.0
IPP Crypto	Ippcp_2021.7.1
IPSec MB	1.4

Table 11. NGINX* Performance

Configuration[1]	Intel® QAT HW (2 devices)	Intel® QAT HW (1 device)	Intel® QAT-SW	Out-of-box Software
Intel® Verified Reference Configuration for NFVI - Plus platform	10876 CPS[2,3]	5445 CPS	2558 CPS	835 CPS
Intel® Verified Reference Configuration for NFVI - Base platform	10867 CPS[4,5]	5445 CPS	2271 CPS	804 CPS
	[1] Uses ECDHE-X448-RSA4K TLS 1.3 handshakes and 4C8T for NGINX [2] Intel® QAT HW (2 devices) is showing up to 13.0x performance over Core Software. [3] Intel® QAT HW (2 devices) is showing up to 4.3x performance over QAT-SW. [4] Intel® QAT HW (2 devices) is showing up to 13.5x performance over Core Software. [5] Intel® QAT HW (2 devices) is showing up to 4.8x performance over QAT-SW.			

Figure 3: NGINX Plus Configuration Performance Graph

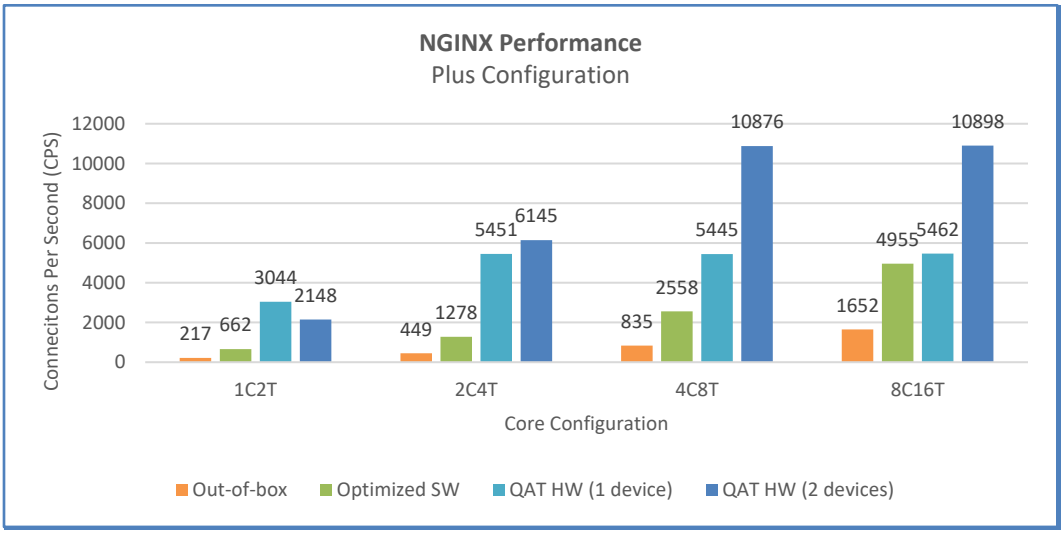
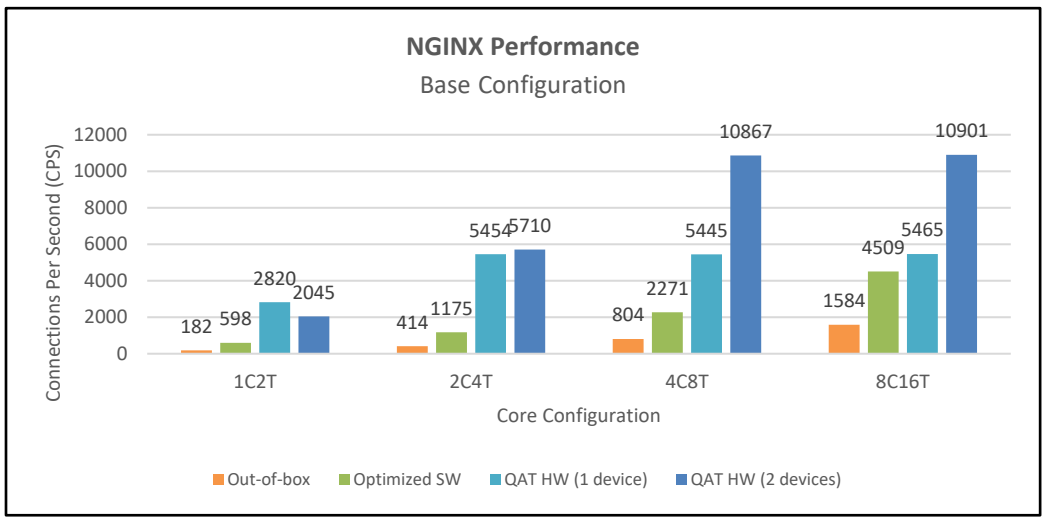
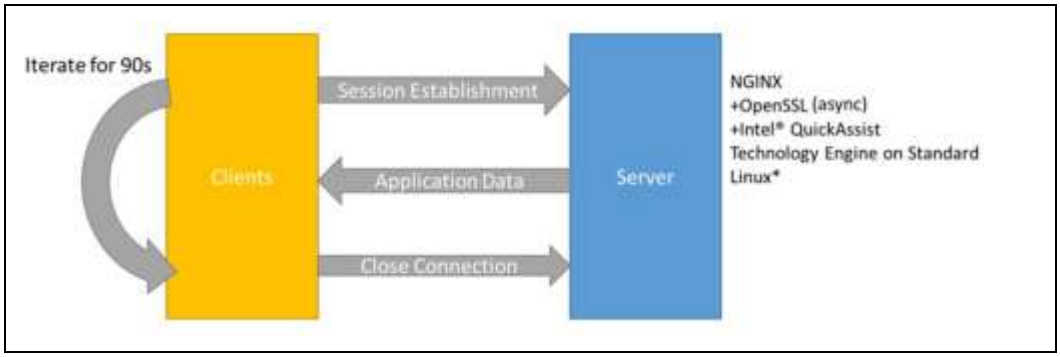


Figure 4: NGINX Base Configuration Performance Graph



5.2.1 NGINX Test Methodology

Figure 5. Test Methodology for SSL with NGINX*



The test methodology implements the following to measure the maximum CPS that the system can sustain:

- NGINX* Server Stack Utilizing Intel® QAT Engine and OpenSSL* 1.1.1m
- OpenSSL* scripts running in a container simulate client traffic sending 0-byte requests to the server (HTTPS workloads).
- The test measures the number of connection requests per second that the server can sustain.

To run the NGINX workload testing follow the steps below:

1. Make sure that the QAT driver has been setup as described in Section 4.2
2. Run the following commands to update the QAT configuration for the NGINX testing:

```
for i in {0..3}
do
        sed -i "s/dc/sym/" /etc/4xxx_dev${i}.conf
        sed -i "s/SSL/SHIM/" /etc/4xxx_dev${i}.conf
        sed -i "s/LimitDevAccess = 0/LimitDevAccess = 1/"
/etc/4xxx_dev${i}.conf
done

sed -i "s/dc/sym/" /etc/4xxxvf_dev0.conf
sed -i "s/SSL/SHIM/" /etc/4xxxvf_dev0.conf
sed -i "s/LimitDevAccess = 0/LimitDevAccess = 1/" /etc/4xxxvf_dev0.conf

for i in {1..63}
do
        cp /etc/4xxxvf_dev0.conf /etc/4xxxvf_dev${i}.conf
done

/etc/init.d/qat_service restart
```

3. Create the following file called "`nginx_pod.yaml`":

```
apiVersion: v1
kind: Pod
metadata:
```

```
  labels:
    run: nginx-pod
  name: nginx-pod
spec:
  containers:
  - args:
    - /bin/bash
    image: ubuntu:22.04
    name: nginx-pod
    stdin: true
    securityContext:
      privileged: true
  restartPolicy: Always
```

4. Create the following file called "openssl_install.sh":

```
#!/bin/bash

apt update -y ; apt install -y git make gcc vim

cd /home
git clone https://github.com/openssl/openssl.git
cd openssl
git checkout OpenSSL_1_1_1m

./config --prefix=/usr/local/ssl -Wl,-rpath,/usr/local/ssl/lib
make
make install
```

5. Create the following file called "async_nginx_setup.sh":

```
#!/bin/bash

apt update -y ; apt install -y make gcc g++ pkg-config libssl-dev zlib1g-dev
wget git nasm autoconf cmake libtool iproute2 libudev-dev

cd /home
export ASYNC_MODE_NGINX="v0.5.0"
git clone --depth 1 -b $ASYNC_MODE_NGINX
https://github.com/intel/asynch_mode_nginx.git

export OPENSSL_LIB=/usr/local/ssl/lib
export ICP_ROOT=/root/QAT

cd asynch_mode_nginx
./configure --prefix=/var/www --conf-path=/usr/share/nginx/conf/nginx.conf --
sbin-path=/usr/bin/nginx --pid-path=/run/nginx.pid --lock-
path=/run/lock/nginx.lock --modules-path=/usr/lib64/nginx --without-
http_rewrite_module --with-http_ssl_module --add-dynamic-
module=modules/nginx_qat_module/ --with-cc-opt="-DNGX_SECURE_MEM -
I$OPENSSL_LIB/include -I$ICP_ROOT/quickassist/include -
I$ICP_ROOT/quickassist/include/dc -Wno-error=deprecated-declarations" --with-
ld-opt="-Wl,-rpath=$OPENSSL_LIB/lib -L$OPENSSL_LIB/lib"
make && make install
```

6. Create the following file called "qat_hw_setup.sh":

```
#!/bin/bash
# Remove previous version of QAT Engine if needed
if [ -d "/home/QAT_Engine" ]
then
        cd /home/QAT_Engine
```

```
        make uninstall
        make clean
        cd ..
        rm -rf QAT_Engine
fi

cd /home
export QAT_ENGINE_VERSION="v1.1.0"
git clone --depth 1 -b $QAT_ENGINE_VERSION https://github.com/intel/QAT_Engine

apt install -y autoconf automake libtool libssl-dev

cd QAT_Engine
./autogen.sh
./configure --with-openssl_install_dir=/usr/local/ssl --enable-qat_hw --with-
qat_hw_dir=/root/QAT
make && make install
```

7. Create the following file called "`qat_sw_setup.sh`":

```
#!/bin/bash

# Remove previous version of QAT Engine if needed
if [ -d "/home/QAT_Engine" ]
then
        cd /home/QAT_Engine
        make uninstall
        make clean
        cd ..
        rm -rf QAT_Engine
fi

cd /home
export QAT_ENGINE_VERSION="v1.1.0"
export IPP_CRYPTO_VERSION="ippcp_2021.7.1"
export IPSEC_MB_VERSION="v1.4"

git clone --depth 1 -b $QAT_ENGINE_VERSION https://github.com/intel/QAT_Engine
git clone --depth 1 -b $IPP_CRYPTO_VERSION https://github.com/intel/ipp-crypto
git clone --depth 1 -b $IPSEC_MB_VERSION https://github.com/intel/intel-ipsec-
mb

apt install -y autoconf automake libtool libssl-dev

cd /home/ipp-crypto/sources/ippcp/crypto_mb
cmake . -B"../build" -DOPENSSL_INCLUDE_DIR=/usr/local/ssl/include/openssl -
DOPENSSL_LIBRARIES=/usr/local/ssl/lib -
DOPENSSL_ROOT_DIR=/usr/local/ssl/bin/openssl
cd ../build
make crypto_mb && make install

cd /home/intel-ipsec-mb
make && make install LIB_INSTALL_DIR=/usr/lib64

cd /home/QAT_Engine
sed -i "s/GCM_IV_DATA_LEN/IMB_GCM_IV_DATA_LEN/g" qat_evp.c
./autogen.sh
./configure --with-openssl_install_dir=/usr/local/ssl/lib --disable-qat_hw --
enable-qat_sw
make && make install
```

```
cp /usr/lib64/libIPSec_MB.so.1 /usr/lib/x86_64-linux-gnu/
cp /usr/local/lib/libcrypto_mb.so.11 /usr/lib/x86_64-linux-gnu/
```

8. Use the following command to create the certificate and key to use for NGINX:

```
openssl req -x509 -newkey rsa:4096 -keyout server.key -out server.crt -
days 365 -nodes
```

9. Create the following file called "`nginx_test_qatengine.conf`":

```
user root;
load_module /usr/lib64/nginx/ngx_ssl_engine_qat_module.so;
worker_processes 32;
#ssl_engine qat;
worker_rlimit_nofile 30000;

events
{
        use epoll;
        worker_connections 200000;
        #multi_accept on;
}
ssl_engine {
        use_engine qatengine;
        default_algorithms RSA,EC,DH,PKEY_CRYPTO;
        qat_engine {
            qat_offload_mode async;
            qat_notify_mode poll;
            #qat_poll_mode heuristic;
            #qat shutting down release on;
        }
}
http
{
          ssl_buffer_size 64k;
          keepalive_timeout  100;
          include       /usr/share/nginx/conf/mime.types;
          default_type  application/octet-stream;
          sendfile on;

        server
        {
                listen       4400 ssl reuseport backlog=200000;
                server_name  localhost;
                access_log off;
                sendfile on;
                #ssl                  on;
                ssl_asynch           on;
                ssl_certificate      /home/server.crt;
                ssl_certificate_key  /home/server.key;
```

```
                ssl_session_timeout  5m;
                ssl_ecdh_curve X448;

                ssl_protocols  TLSv1.3;
                ssl_ciphers ALL;

                ssl_prefer_server_ciphers   on;

                location /
                {
                    root   html;
                    index  index.html index.htm;
            }
        }
}
```

10. Create the following file called "`nginx_test_software.conf`":

```
user root;
worker_processes 32;
#ssl_engine qat;
worker_rlimit_nofile 30000;
#pid  /usr/local/nginx/logs/nginx.pid;

events
{
        use epoll;
        worker_connections 200000;
        #multi_accept on;
}
http
{
          ssl_buffer_size 64k;
          keepalive_timeout  100;
          include       /usr/share/nginx/conf/mime.types;
          default_type  application/octet-stream;
          sendfile on;

        server
        {
                listen       4400 ssl reuseport backlog=200000;
                server_name  localhost;
                access_log off;
                sendfile on;
                #ssl                  on;
                #ssl_asynch          on;
                ssl_certificate      /home/server.crt;
                ssl_certificate_key  /home/server.key;

                ssl_session_timeout  5m;
```

```
                ssl_ecdh_curve X448;

                ssl_protocols  TLSv1.3;
                ssl_ciphers ALL;

                ssl_prefer_server_ciphers   on;

                location /
                {
                    root   html;
                    index  index.html index.htm;
                }
        }
}
```

11. Create the following file called "`openssl_clients.sh`":

```
#!/bin/bash
#
# Copyright 2017 Intel Corporation
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"),
# to deal in the Software without restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense,
# and/or sell copies of the Software, and to permit persons to whom
# the Software is furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
# IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
# DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
# TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
# THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# SPDX-License-Identifier: MIT
#

#######################################
############# USER INPUT #############
#######################################
ip_address=localhost
 time=90
clients=2500
portbase=4400
```

```
cipher=TLS_AES_128_GCM_SHA256;
OPENSSL_DIR=/usr/local/ssl

######################################################################
###LOGIC TO FIND HYPERTHREAD CORE CORRESPONDING TO PHYSICAL CORE ####
######################################################################


# Cores to use for client traffic generation.  When
#   running the CPS tests on one system, use every core that is
#   not being used for nginx threads.
#

_cores_used=""

total_cores=$(lscpu | grep -E '^Thread|^Core|^Socket|^CPU\(' | sed -n '1p' |
awk '{print $2}')
thread_per_core=$( lscpu | grep -E '^Thread|^Core|^Socket|^CPU\(' | sed -n
'2p' |awk '{print $4}' )
cores_per_socket=$(  lscpu | grep -E '^Thread|^Core|^Socket|^CPU\(' | sed -n
'3p' |awk '{print $4}' )
socket_count=$( lscpu | grep -E '^Thread|^Core|^Socket|^CPU\(' | sed -n '4p'
|awk '{print $2}' )
total_physical_cores=$(( $cores_per_socket * $socket_count ))

_cores_used="-c 5-31,69-95"

#####################################
############# USER INPUT #############
#####################################

#Check for OpenSSL Directory
if [ ! -d $OPENSSL_DIR ];
then
    printf "\n$OPENSSL_DIR does not exist.\n\n"
    printf "Please modify the OPENSSL_DIR variable in the User Input
section!\n\n"
    exit 0
fi

helpAndError () {
    printf "\nThis script is to run the CPS testing HTTPS.\n"
    printf "\nTo use this script: ./connection_test_update1.sh \n"
    printf "\nTo do a dry-run, use the emulation flag:\n"
    printf "./connection test update1.sh --emulation\n\n"
    exit 0
}
emulation=0

#Check for h flag or no command line args
if [[ $1 == *"h"* ]]; then
```

```
    helpAndError
    exit 0
fi

#Check for emulation flag
if [[ $@ == **emulation** ]]
then
    emulation=1
fi

#The total commandline will be cmd1 + "192.168.1.1:4400" + cmd2
cmd1="$OPENSSL_DIR/bin/openssl s_time -connect"
cmd2="-new -ciphersuites $cipher -time $_time"

#Print out variables to check
printf "\n Location of OpenSSL:           $OPENSSL_DIR\n"
printf " IP Addresses:                  $ip_address\n"
printf " Time:                          $_time\n"
printf " Clients:                       $clients\n"
printf " Port Base:                     $portbase\n"
printf " Cipher:                        $cipher\n"
printf " Cores Used:                    $_cores_used\n"


#Remove previous .test files
rm -rf ./.test_*

#Get starttime
starttime=$(date +%s)

#Kick off the tests after checking for emulation
if [[ $emulation -eq 1 ]]
then
    for (( i = 0; i < ${clients}; i++ )); do
        printf "$cmd1 $ip_address:$(($portbase)) $cmd2 >
.test_$(($portbase))_$i &\n"
    done
    exit 0
else
    for (( i = 0; i < ${clients}; i++ )); do
        taskset $_cores_used $cmd1 $ip_address:$(($portbase)) $cmd2 >
.test_$(($portbase))_$i &
    done
fi

waitstarttime=$(date +%s)
# wait until all processes complete
while [ $(ps -ef | grep "openssl s_time" | wc -l) != 1 ];
do
    sleep 1
```

```
done

total=$(cat ./.test_$(($portbase))* | awk '(/^[0-9]* connections in [0-9]*
real/){ total += $1/$4 } END {print total}')
echo $total >> .test_sum
sumTotal=$(cat .test_sum | awk '{total += $1 } END { print total }')
printf "Connections per second:      $sumTotal CPS\n"
printf "Finished in %d seconds (%d seconds waiting for procs to start)\n"
$(($(date +%s) - $starttime)) $(($waitstarttime - $starttime))
rm -rf ./.test_*
```

12. Create the following file called "openssl_pod.yaml":

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: openssl-pod
  name: openssl-pod
spec:
  containers:
  - args:
    - /bin/bash
    image: ubuntu:22.04
    name: openssl-pod
    stdin: true
  restartPolicy: Always
```

13. Create and execute the following file called "`setup_pods.sh`":

```
#!/bin/bash

# Create NGINX and OpenSSL pods

kubectl delete pod nginx-pod
kubectl delete pod openssl-pod

kubectl create -f nginx_pod.yaml
kubectl create -f openssl_pod.yaml

sleep 20

# Copy installation scripts and testing files to pods

kubectl cp async_nginx_setup.sh nginx-pod:/home
kubectl cp qat_sw_setup.sh nginx-pod:/home
kubectl cp qat_hw_setup.sh nginx-pod:/home

kubectl cp openssl_install.sh nginx-pod:/home
kubectl cp openssl_install.sh openssl-pod:/home

kubectl cp server.crt nginx-pod:/home
kubectl cp server.key nginx-pod:/home

echo "To setup NGINX with QAT SW, run: kubectl exec nginx-pod --
/home/qat_sw_setup.sh"
echo "To setup NGINX with QAT HW, run: kubectl exec nginx-pod --
/home/qat_hw_setup.sh"
```

14. Create the following file called "`run_nginx_test.sh`":

```
#!/bin/bash

read -p $'Please enter number of cores to test with\n' num_cores
read -p $'Please enter the QAT Path to test with\n' QAT_PATH

# Copy QAT driver directory and NGINX configuration files to nginx-pod

kubectl cp $QAT_PATH nginx-pod:/root/
kubectl cp nginx_test_qatengine.conf nginx-pod:/home
kubectl cp nginx_test_software.conf nginx-pod:/home

# Install Openssl, async_mode_nginx and QAT_Engine with QAT HW in nginx-pod

kubectl exec nginx-pod -- /bin/bash -c "/home/openssl_install.sh"
kubectl exec nginx-pod -- /bin/bash -c "/home/async_nginx_setup.sh"
kubectl exec nginx-pod -- /bin/bash -c "/home/qat_hw_setup.sh"

# Install Openssl in openssl-pod

kubectl exec openssl-pod -- /bin/bash -c "/home/openssl_install.sh"

# Setup IP address for OpenSSL files

kubectl cp openssl_clients.sh openssl-pod:/home

nginx_ip=$(kubectl exec nginx-pod -- ip a | grep -A 3 eth | grep "inet " | awk
'{print $2;}' | awk -F"/" '{print $1;}')
kubectl exec openssl-pod -- sed -i "s/localhost/${nginx_ip}/"
/home/openssl_clients.sh

# Set the cores for OpenSSL
cores_per_socket=$(  lscpu | grep -E '^Thread|^Core|^Socket|^CPU\(' | sed -n
'3p' |awk '{print $4}' )
kubectl exec openssl-pod -- sed -i "s/5-31,69-95/$(( ${num_cores} + 1 ))-
$(( ${cores_per_socket} - 1 )),$(( ${num_cores} + $(( ${cores_per_socket} * 2
+ 1 )) ))- $(( ${cores_per_socket} * 3 - 1 ))/" /home/openssl_clients.sh

# Run the QAT HW test

kubectl exec nginx-pod -- taskset -c "1-${num_cores}, $(( ${cores_per_socket} *
2 + 1 ))-$(( ${num_cores} + $(( ${cores_per_socket} * 2 )) ))" nginx -c
/home/nginx_test_qatengine.conf

kubectl exec openssl-pod -- /home/openssl_clients.sh >
qat_hw_results_nginx.log &
sleep 120

kubectl exec nginx-pod -- pkill nginx
sleep 10

# Install QAT Engine with SW

kubectl exec nginx-pod -- /bin/bash -c "/home/qat_sw_setup.sh"

# Run the QAT SW test

kubectl exec nginx-pod -- taskset -c "1-${num_cores}, $(( ${cores_per_socket} *
2 + 1 ))-$(( ${num_cores} + $(( ${cores_per_socket} * 2 )) ))" nginx -c
/home/nginx_test_qatengine.conf
```

```
kubectl exec openssl-pod -- /home/openssl_clients.sh >
qat_sw_results_nginx.log &
sleep 120

kubectl exec nginx-pod -- pkill nginx
sleep 10

# Run the core SW test

kubectl exec nginx-pod -- taskset -c "1-${num_cores},$(( ${cores_per_socket} *
2 + 1 ))-$(( ${num_cores} + $(( ${cores_per_socket} * 2 )) ))" nginx -c
/home/nginx_test_software.conf

kubectl exec openssl-pod -- /home/openssl_clients.sh > coresw_results.log &
sleep 120

kubectl exec nginx-pod -- pkill nginx
```

15. Run the file "`run_nginx_test.sh`" to perform the testing and generate the results for the out-of-box SW, QAT SW, and QAT HW (2 devices) configurations.
16. To generate the results for the QAT HW (1 device) configuration, change the number of worker processes in the "`nginx_test_qatengine.conf`" from 32 to 16 and rerun the "`run_nginx_test.sh`" file.
    a. Rename the "`qat_hw_results_nginx.log`" files from the previous step first to prevent them from being overwritten
    b. The "`qat_hw_results_nginx.log`" file in this run will correspond to the results of this configuration
    c. After this test is complete, revert the number of worker processes in the "`nginx_test_qatengine.conf`" back to 32

5.3 QATzip

QATzip is a user space library which builds on top of the Intel® QuickAssist Technology user space library, to provide extended accelerated compression and decompression services by offloading the actual compression and decompression request(s). QATzip produces data using the standard gzip* format (RFC1952) with extended headers or lz4 blocks with lz4 frame format. The data can be decompressed with a compliant gzip* or lz4 implementation. QATzip is designed to take full advantage of the performance provided by Intel® QuickAssist Technology.

To compare the value proposition for QATzip, the compression throughput, decompression throughput, and compression ratio can be gathered to review the value proposition of QAT offload.

5.4 VPP IPSec

Vector Packet Processor (VPP) Internet Protocol Security (IPSec) is generally used for firewall or VPN applications and provides secure remote access to onsite servers. For a given platform, the VPP IPSec workload can demonstrate the effectiveness of its crypto processing capabilities.

For the Intel Verified Reference Configuration for NFVI Plus Platform, ensure that the results of the system follow the expected results as shown in Table 133, to baseline the performance of the platform.

Table 12. VPP IPSec Workload Configuration

Ingredient	Software Version Details
VPP	23.02
DPDK	22.07
T-Rex	v3.00
IPSec MB	1.3

Table 13. VPP IPSec Plus Configuration Performance

Packet Size (bytes)	Optimized SW[1] Throughput (Gbps)	Out-of-box Software[2] Throughput (Gbps)
64	24.98	0.50
128	36.48	1.00
256	58.00	2.00
512	82.99	3.49
1024	99.97	7.49
1280	99.94	9.00
1420	99.99	10.00

Note: [1] Uses IPSecMB crypto handler

[2] Uses openssl crypto handler

Figure 6: VPP IPSec Plus Configuration Performance Graph

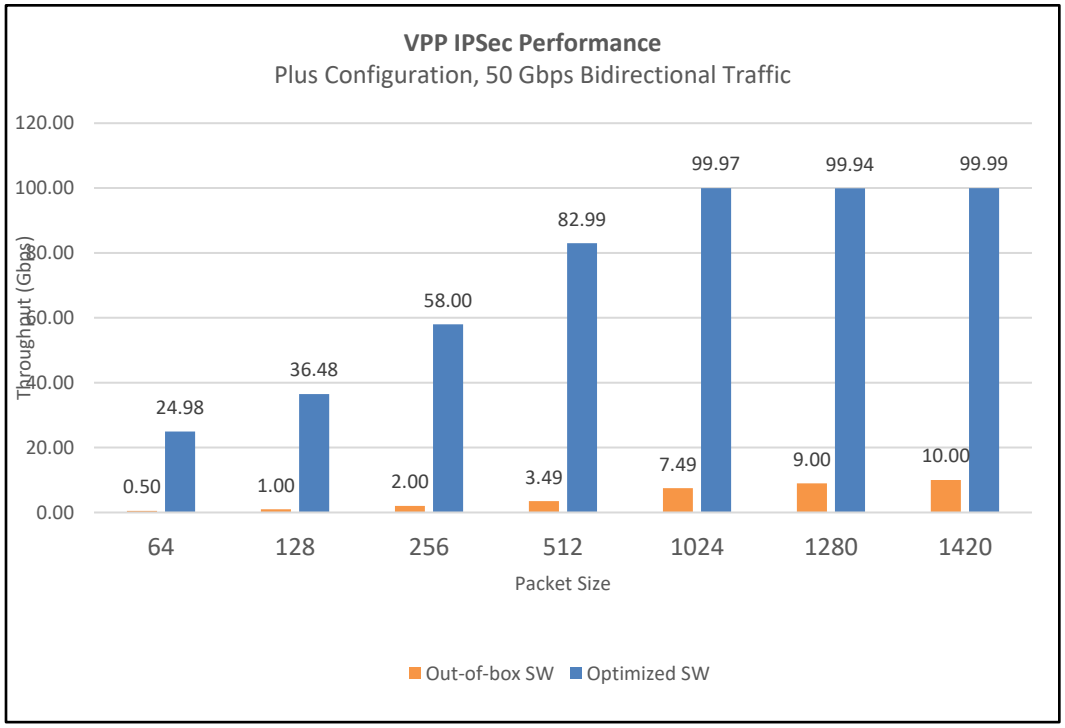
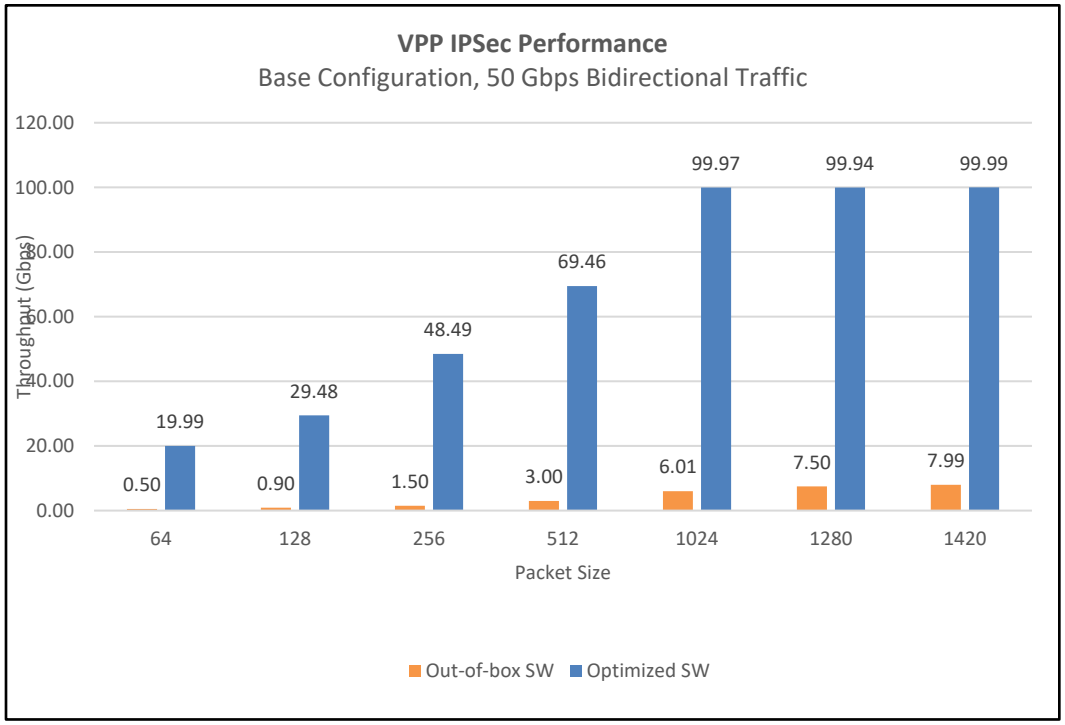


Table 14: VPP IPSec Base Configuration Performance

Packet Size (bytes)	Optimized SW[1] Throughput (Gbps)	Out-of-box Software[2] Throughput (Gbps)
64	19.99	0.50
128	29.48	0.90
1256	48.49	1.50
512	69.46	3.00
1024	99.97	6.01
1280	99.94	7.50
1420	99.99	7.99

Figure 7: VPP IPSec Base Configuration Performance Graph



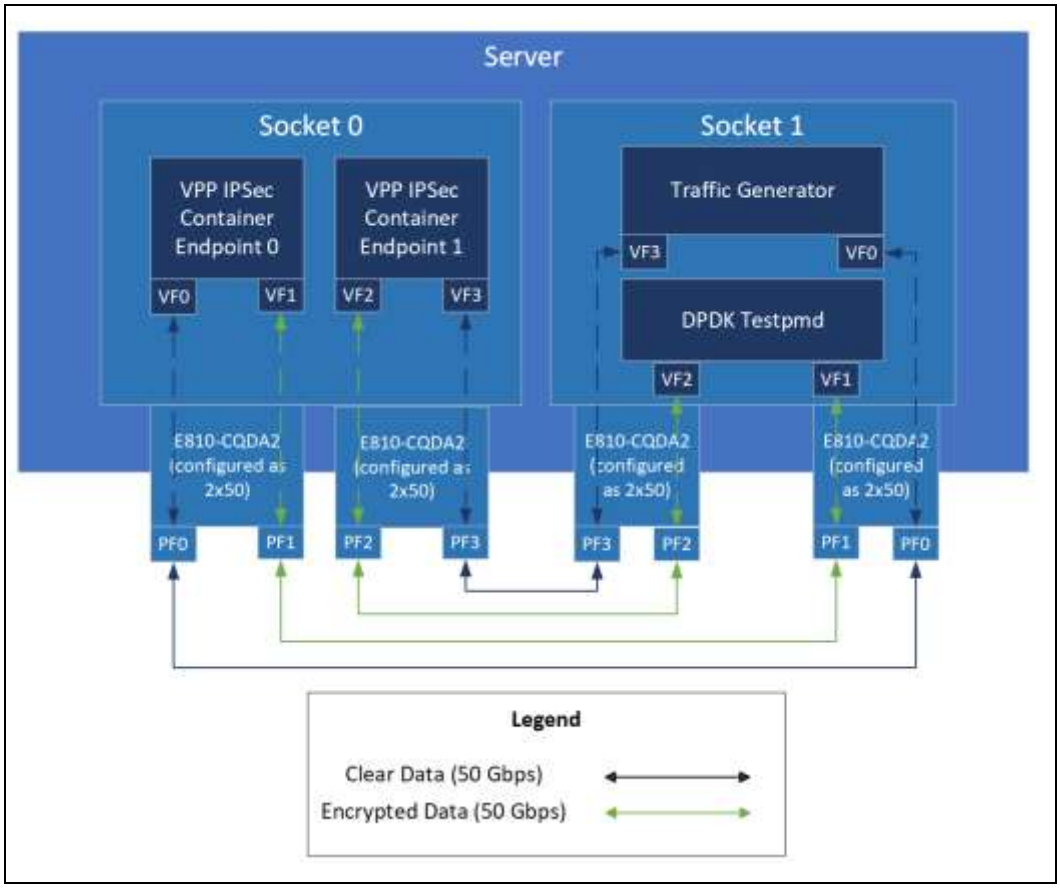
5.4.1 VPP IPSec Test Methodology

5.4.1.1 Setup

To perform the VPP IPSec benchmark testing, the system under test must be setup as shown in Figure 8.

Each socket of the system must contain four 50Gbps NIC ports in the Plus Configuration or four 25 Gbps NIC ports in the Base Configuration to be connected to each other. Socket 0 will host the VPP IPSec container endpoints while socket 1 will host the traffic generator and also use the DPDK testpmd application as a simple switch. The generated traffic will flow bidirectionally throughout the setup. On socket 1, the two ports receiving the clear data should be used for the traffic generator, while the two ports receiving the encrypted data should run the DPDK testpmd application to forward the traffic to the other port.

Figure 8. Test Setup for VPP IPSec



To run the VPP IPSec testing, the first step is to configure the NIC devices to have a total of four ports available for each socket of the worker node server. Download the Ethernet Port Configuration Tool (EPCT) from this link: https://www.intel.com/content/www/us/en/download/19437/ethernet-port-configuration-tool-linux.html?wapkw=ethernet%20port%20configuration%20tool. Follow the EPCT instructions to configure the NIC devices.

Following this, it is necessary to utilize the SR-IOV Network Operator to create VFs of the NIC ports to be used in the VPP IPSec endpoints. From the Red Hat OpenShift web console of the cluster, navigate to the OperatorHub and search for the SR-IOV Network Operator. Install the operator, then navigate to the "Installed Operators" tab.

Click on the "SR-IOV Network Operator" and create an instance of the Sriov Network Node Policy. Under "nicSelector," select "rootDevices" and provide the PCI address of the first NIC port that will be used. Change the "resourceName" to the name of the network device followed by "_vf", and change "numVfs" to 1. Press "Create" and repeat this step for the other 7 NIC ports that will be used. Once completed, the worker node will automatically restart to apply the settings.

5.4.1.2 Testing Procedure

To run the VPP IPSec testing workload follow the steps below

1. Setup the NIC ports' virtual functions to be used in the testing:
    a. For each PCI address of the NIC ports being used, run:
    `echo 1 > /sys/bus/pci/devices/<pci_address>/sriov_numvfs`
2. Create the following file called "`setup_vpp_pods.sh`":

```
#!/bin/bash

# Set spoof-checking off and trust on for NIC VFs
for i in $(ip a | grep "<" | awk '{print $2}' | sed 's/://')
do
    ip link set $i vf 0 spoof off
    ip link set $i vf 0 trust on
done

kubectl delete pod vpp-ep0
kubectl delete pod vpp-ep1
kubectl delete pod dpdk-pod
kubectl delete pod trex-pod

for i in 0 1
do
    kubectl create -f vpp_ep${i}.yaml
    sleep 20
    kubectl cp install_vpp.sh vpp-ep${i}:/home
    kubectl cp ep${i} vpp-ep${i}:/home
    kubectl cp 0001-add-qat-pf-vf-delay.patch vpp-ep${i}:/home
    kubectl exec vpp-ep${i} -- /home/install_vpp.sh
done

./setup_dpdk_pod.sh
./setup_trex_pod.sh
```

3. Create the following file called "`install_vpp.sh`":

```
#!/bin/bash

apt-get update ; apt-get install -y vim git make build-essential gdb sudo pkg-
config wget

cd /home
git clone https://github.com/FDio/vpp
cd vpp
git checkout v23.02

mkdir -p ./build/external/patches/dpdk_22.07/
mv /home/0001-add-qat-pf-vf-delay.patch ./build/external/patches/dpdk_22.07/
sed -i "s/vm.nr_hugepages=1024/#vm.nr_hugepages=1024/" /home/vpp/src/vpp/conf/80-
vpp.conf
```

```
yes | make install-dep
make install-ext-deps
make pkg-deb
dpkg -i /home/vpp/build-root/*.deb
```

4. Create the following file called "`0001-add-qat-pf-vf-delay.patch`":

```
diff --git a/drivers/common/qat/qat_adf/adf_pf2vf_msg.h
b/drivers/common/qat/qat_adf/adf_pf2vf_msg.h
index 4029b1c14a..018edc28e5 100644
--- a/drivers/common/qat/qat_adf/adf_pf2vf_msg.h
+++ b/drivers/common/qat/qat_adf/adf_pf2vf_msg.h
@@ -128,7 +128,7 @@

 /* How long to wait for far side to acknowledge receipt */
 #define ADF_IOV_MSG_ACK_DELAY_US                5
-#define ADF_IOV_MSG_ACK_MAX_RETRY      (100 * 1000 /
ADF_IOV_MSG_ACK_DELAY_US)
+#define ADF_IOV_MSG_ACK_MAX_RETRY      (100 * 100000 /
ADF_IOV_MSG_ACK_DELAY_US)
 /* If CSR is busy, how long to delay before retrying */
 #define ADF_IOV_MSG_RETRY_DELAY                         5
 #define ADF_IOV_MSG_MAX_RETRIES                         3
```

5. Create the following file called "`vpp_ep0.yaml`":

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: vpp-ep0
  name: vpp-ep0
spec:
  containers:
  - args:
    - /bin/bash
    image: ubuntu:22.04
    name: vpp-ep0
    stdin: true
    securityContext:
      privileged: true
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        memory: "50Gi"
        hugepages-1Gi: "15Gi"
      requests:
        memory: "50Gi"
        hugepages-1Gi: "15Gi"
  volumes:
  - name: hugepage
```

```
    emptyDir:
      medium: HugePages
  restartPolicy: Always
```

6. Create the following file called "vpp_ep1.yaml":

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: vpp-ep1
  name: vpp-ep1
spec:
  containers:
  - args:
    - /bin/bash
    image: ubuntu:22.04
    name: vpp-ep1
    stdin: true
    securityContext:
      privileged: true
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        memory: "50Gi"
        hugepages-1Gi: "15Gi"
      requests:
        memory: "50Gi"
        hugepages-1Gi: "15Gi"
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
  restartPolicy: Always
```

7. Create the following file called "dpdk_pod.yaml":

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: dpdk-pod
  name: dpdk-pod
spec:
  containers:
  - args:
    - /bin/bash
    image: ubuntu:22.04
    name: dpdk-pod
    stdin: true
```

```
    securityContext:
      privileged: true
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        memory: "30Gi"
        hugepages-1Gi: "10Gi"
      requests:
        memory: "30Gi"
        hugepages-1Gi: "10Gi"
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
  restartPolicy: Always
```

8. Create the following file called "`setup_dpdk_pod.sh`":

```
#!/bin/bash

kubectl create -f dpdk_pod.yaml
wget http://fast.dpdk.org/rel/dpdk-22.11.1.tar.xz
sleep 120
kubectl cp dpdk-stable-22.11.1.tar.xz dpdk-pod:/home
kubectl cp run_testpmd.sh dpdk-pod:/home
kubectl cp install_dpdk.sh dpdk-pod:/home
kubectl exec dpdk-pod -- /home/install_dpdk.sh
```

9. Create the following file called "`install_dpdk.sh`":

```
#!/bin/bash

apt update -y ; apt install -y kmod pciutils iproute2 meson python3-pyelftools
libnuma-dev

cd /home
tar -xf dpdk-stable-22.11.1.tar.xz
cd dpdk-stable-22.11.1
meson setup -Dexamples=all build
cd build
ninja
ninja install
ldconfig
```

10. Create the following file called "`run_testpmd.sh`":

```
#!/bin/bash

/home/dpdk-stable-22.11.1/build/app/dpdk-testpmd -l 42-47 -n 4 --socket-mem=0,4096
-a <sock1_port_1> -a <sock1_port_2> --main-lcore=42 --in-memory
```

a. Replace "<sock1_port_1>" and "<sock1_port_2>" with the PCI addresses (ex: 81:09.0) of the Ethernet Adaptive Virtual Functions created for the NIC ports corresponding to PF1 and PF2 of socket 1.

11. Create the following file called "trex_pod.yaml":

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: trex-pod
  name: trex-pod
spec:
  containers:
  - args:
    - /bin/bash
    image: python
    name: trex-pod
    stdin: true
    securityContext:
      privileged: true
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        memory: "10Gi"
        hugepages-1Gi: "10Gi"
      requests:
        memory: "10Gi"
        hugepages-1Gi: "10Gi"
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
  restartPolicy: Always
```

12. Create the following file called "setup_trex_pod.sh":

```
#!/bin/bash

kubectl create -f trex_pod.yaml
wget --no-check-certificate https://trex-tgn.cisco.com/trex/release/v3.00.tar.gz
sleep 30
kubectl cp v3.00.tar.gz trex-pod:/home
kubectl cp vpp_packets_p0.py trex-pod:/home
kubectl cp vpp_packets_p1.py trex-pod:/home
kubectl cp install_trex.sh trex-pod:/home
kubectl cp trex_cfg.yaml trex-pod:/etc
kubectl cp /lib/modules trex-pod:/lib

kubectl exec trex-pod -- /home/install_trex.sh
```

13. Create the following file called "`trex_cfg.yaml`":

```
- version       : 2
  interfaces    : ["<sock1_port_0>","<sock1_port_3>"]
  port_limit    : 2

  enable_zmq_pub : true
  zmq_pub_port : 4500
  zmq_rpc_port : 4501
  limit_memory: 8192
  rx_desc : 1024
  tx_desc : 1024
  c : 7
  port_bandwidth_gb : 50
  services_core : 32
  port_info:
    - dst_mac: "<sock0_port0_vf_mac>" # Socket 0 Port 0 MAC
      src_mac : "<sock1_port0_vf_mac>" # Socket 1 Port 0 MAC
    - dst_mac: "<sock0_port3_vf_mac>" # Socket 0 Port 3 MAC
      src_mac : "<sock1_port3_vf_mac>" # Socket 1 Port 3 MAC

  platform :
    master_thread_id : 33
    latency_thread_id : 34
    dual_if :
      - socket : 1
        threads : [35,36,37,38,39,40,41]
```

b. Replace <sock1_port_0> and <sock1_port_3> with the corresponding PCI address of the VFs for PF0 and PF3 of socket 1. Replace <sock0_port0_vf_mac>, <sock1_port0_vf_mac>, <sock0_port3_vf_mac>, and <sock1_port3_vf_mac> with the corresponding MAC address of the VF for each port.

c. If using Base configuration, change "port_bandwidth_gb" from "50" to "25"

14. Create the following file called "`vpp_packets_p0.py`":

```
from trex_stl_lib.api import *
import argparse

class STLS1(object):

    def __init__ (self):
        self.mode = 0
        self.fsize = 64; # the size of the packet
        self.tunnels = 128; # number of VPP IPSec tunnels being used

    def create_pkt_base (self,addr):
        pkt_dst = "10.64.0." + str(addr)
        t=[

Ether(dst="<sock0_port0_vf_mac>")/IP(src="192.168.105.3",dst=pkt_dst)/UDP(dport=10
24,sport=49000),

Ether()/Dot1Q(vlan=12)/IP(src="16.0.0.1",dst="48.0.0.1")/UDP(dport=12,sport=1025),
```

```
Ether()/Dot1Q(vlan=12)/Dot1Q(vlan=12)/IP(src="16.0.0.1",dst="48.0.0.1")/UDP(dport=
12,sport=1025),

Ether()/Dot1Q(vlan=12)/IP(src="16.0.0.1",dst="48.0.0.1")/TCP(dport=12,sport=1025),
                Ether()/Dot1Q(vlan=12)/Ipv6(src="::5")/TCP(dport=12,sport=1025),
                Ether()/IP()/UDP()/Ipv6(src="::5")/TCP(dport=12,sport=1025)
                ];
        return t[self.mode]

    def create_stream (self):

        # Create base packet and pad it to size
        size = self.fsize - 4; # HW will add 4 bytes ethernet FCS
        profile = []
        for i in range(1,self.tunnels + 1):
            base_pkt =  self.create_pkt_base(i)
            pad = max(0, size - len(base_pkt)) * 'x'
            pkt = STLPktBuilder(pkt = base_pkt/pad,
                vm = [])
            profile.append(STLStream(packet = pkt, mode = STLTXCont()))
        return STLProfile(profile).get_streams()

    def get_streams (self, tunables, **kwargs):
        parser = argparse.ArgumentParser(description='Argparser for
{}'.format(os.path.basename(__file__)),

formatter_class=argparse.ArgumentDefaultsHelpFormatter)

        args = parser.parse_args(tunables)
        # create 1 stream
        return self.create_stream()

# dynamic load - used for trex console or simulator
def register():
    return STLS1()
```

d. Replace <sock0_port0_vf_mac> with the corresponding MAC address of the VF for port 0 in socket 0

15. Create the following file called "vpp_packets_p1.py":

```
from trex_stl_lib.api import *
import argparse

class STLS1(object):

    def __init__ (self):
        self.mode = 0
        self.fsize = 64; # the size of the packet
        self.tunnels = 128; # number of VPP IPSec tunnels being used

    def create_pkt_base (self,addr):
```

```
        pkt_dst = "20.64.0." + str(addr)
        t=[

Ether(dst="<sock0_port3_vf_mac>")/IP(src="192.168.115.3",dst=pkt_dst)/UDP(dport=10
24,sport=49000),

Ether()/Dot1Q(vlan=12)/IP(src="16.0.0.1",dst="48.0.0.1")/UDP(dport=12,sport=1025),

Ether()/Dot1Q(vlan=12)/Dot1Q(vlan=12)/IP(src="16.0.0.1",dst="48.0.0.1")/UDP(dport=
12,sport=1025),

Ether()/Dot1Q(vlan=12)/IP(src="16.0.0.1",dst="48.0.0.1")/TCP(dport=12,sport=1025),
                Ether()/Dot1Q(vlan=12)/Ipv6(src="::5")/TCP(dport=12,sport=1025),
                Ether()/IP()/UDP()/Ipv6(src="::5")/TCP(dport=12,sport=1025)
                ];
        return t[self.mode]

    def create_stream (self):

        # Create base packet and pad it to size
        size = self.fsize - 4; # HW will add 4 bytes ethernet FCS
        profile = []
        for i in range(1,self.tunnels + 1):
            base_pkt =  self.create_pkt_base(i)
            pad = max(0, size - len(base_pkt)) * 'x'
            pkt = STLPktBuilder(pkt = base_pkt/pad,
                vm = [])
            profile.append(STLStream(packet = pkt, mode = STLTXCont()))
        return STLProfile(profile).get_streams()

    def get_streams (self, tunables, **kwargs):
        parser = argparse.ArgumentParser(description='Argparser for
{}'.format(os.path.basename(__file__)),

formatter_class=argparse.ArgumentDefaultsHelpFormatter)

        args = parser.parse_args(tunables)
        # create 1 stream
        return self.create_stream()

# dynamic load - used for trex console or simulator
def register():
    return STLS1()
```

e. Replace <sock0_port3_vf_mac> with the corresponding MAC address of the VF for port 3 in socket 0

16. Create the following file called "`install_trex.sh`":

```
#!/bin/bash


apt update -y ; apt install -y vim kmod pciutils file tmux iproute2 meson python3-
pyelftools
```

```
cd /home
tar -xf v3.00.tar.gz

mv vpp_packets_p0.py vpp_packets_p1.py v3.00/stl

cd /usr/lib/x86_64-linux-gnu/
ln -s -f libc.a liblibc.a

sed -i 's/collections.Hashable/collections.abc.Hashable/'
/home/v3.00/external_libs/pyyaml-3.11/python3/yaml/constructor.py
```

17. Create the directories ep0 and ep1:

```
mkdir ep0
mkdir ep1
```

18. Inside directory ep0, create the following file called "`start_vpp_native.sh`"

```
#!/bin/bash
vpp -c /home/ep0/startup_native.conf
exit 0
```

19. In the directory ep0, create the following file called "`start_vpp_qat_sw.sh`":

```
#!/usr/bin/env bash
vpp -c /home/ep0/startup_qat_sw.conf
exit 0
```

20. Inside directory ep0, create the following file called "`startup_native.conf`":

```
unix {
   nodaemon
   log /var/log/vpp/vpp.log
   full-coredump
   cli-listen /run/vpp/cli.sock
   gid vpp
   interactive
   exec /home/ep0/ipsec.cli
}

logging {
  size 4096
  default-log-level debug
  default-syslog-log-level debug
}

api-trace {
  ## This stanza controls binary API tracing. Unless there is a very strong
reason,
  ## please leave this feature enabled.
  On
  ## Additional parameters:
  ##
  ## To set the number of binary API trace records in the circular buffer,
configure nitems
```

```
  ##
  ## nitems <nnn>
  ##
  ## To save the api message table decode tables, configure a filename. Results in
/tmp/<filename>
  ## Very handy for understanding api message changes between versions,
identifying missing
  ## plugins, and so forth.
  ##
  ## save-api-table <filename>
}

api-segment {
  gid vpp
}

socksvr {
  default
}

cpu {
       ## In the VPP there is one main thread and optionally the user can create
worker(s)
       ## The main thread and worker thread(s) can be pinned to CPU core(s)
manually or automatically

       ## Manual pinning of thread(s) to CPU core(s)

       ## Set logical CPU core where main thread runs, if main core is not set
       ## VPP will use core 1 if available
       main-core 0

       ## Set logical CPU core(s) where worker threads are running
       corelist-workers 1-8

       ## Automatic pinning of thread(s) to CPU core(s)

       ## Sets number of CPU core(s) to be skipped (1 … N-1)
       ## Skipped CPU core(s) are not used for pinning main thread and working
thread(s).
       ## The main thread is automatically pinned to the first available CPU core
and worker(s)
       ## are pinned to next free CPU core(s) after core assigned to main thread
       # skip-cores 4

       ## Specify a number of workers to be created
       ## Workers are pinned to N consecutive CPU cores while skipping "skip-
cores" CPU core(s)
       ## and main thread's CPU core
       # workers 4

       ## Set scheduling policy and priority of main and worker threads
```

```
       ## Scheduling policy options are: other (SCHED_OTHER), batch (SCHED_BATCH)
       ## idle (SCHED_IDLE), fifo (SCHED_FIFO), rr (SCHED_RR)
       scheduler-policy fifo

       ## Scheduling priority is used only for "real-time policies (fifo and rr),
       ## and has to be in the range of priorities supported for a particular
policy
       # scheduler-priority 50
}

buffers {
       ## Increase number of buffers allocated, needed only in scenarios with
       ## large number of interfaces and worker threads. Value is per numa node.
       ## Default is 16384 (8192 if running unpriviledged)
       # buffers-per-numa 128000

       ## Size of buffer data area
       ## Default is 2048
       default data-size 2048

       ## Size of the memory pages allocated for buffer data
       ## Default will try 'default-hugepage' then 'default'
       ## you can also pass a size in K/M/G e.g. '8M'
       # page-size default-hugepage
}

dpdk {
       ## Change default settings for all interfaces
       dev default {
          ## Number of receive queues, enables RSS
          ## Default is 1
          num-rx-queues 4

          ## Number of transmit queues, Default is equal
          ## to number of worker threads or 1 if no workers treads
          num-tx-queues 4

          ## Number of descriptors in transmit and receive rings
          ## increasing or reducing number can impact performance
          ## Default is 1024 for both rx and tx
          num-rx-desc 4096
          num-tx-desc 4096

          ## VLAN strip offload mode for interface
          ## Default is off
          #vlan-strip-offload off

          ## TCP Segment Offload
          ## Default is off
```

```
          ## To enable TSO`, 'enable-tcp-udp-checksum' must be set
          tso off

          ## Devargs
          ## device specific init args
          ## Default is NULL
          # devargs safe-mode-support=1,pipeline-mode-support=1

          ## rss-queues
          ## set valid rss steering queues
          rss-queues 0-4
       }

       ## Whitelist specific interface by specifying PCI address and in
       ## addition specify custom parameters for this interface
       dev <sock0_port0> {
            workers 1-4
            name eth1
            num-rx-queues 4
       }
       dev <sock0_port1> {
            workers 5-8
            name eth2
            num-rx-queues 4
       }

       ## Blacklist specific device type by specifying PCI vendor:device
       ## Whitelist entries take precedence
       # blacklist 8086:10fb

       ## Disable multi-segment buffers, improves performance but
       ## disables Jumbo MTU support
       no-multi-seg
       ## Change hugepages allocation per-socket, needed only if there is need for
       ## larger number of mbufs. Default is 256M on each detected CPU socket
       socket-mem 8192
       ## Disables UDP / TCP TX checksum offload. Typically needed for use
       ## faster vector PMDs (together with no-multi-seg)
       no-tx-checksum-offload
       ## Enable UDP / TCP TX checksum offload
       ## This is the reversed option of 'no-tx-checksum-offload'
       # enable-tcp-udp-checksum
       ## Change UIO driver used by VPP, Options are: igb_uio, vfio-pci,
       ## uio_pci_generic or auto (default)
       uio-driver vfio-pci
       log-level debug
}

memory {
## Set the main heap size, default is 1G
```

```
        main-heap-size 1G

        ## Set the main heap page size. Default page size is OS default page
        ## which is in most cases 4K. if different page size is specified VPP
        ## will try to allocate main heap by using specified page size.
        ## special keyword 'default-hugepage' will use system default hugepage
        ## size
        main-heap-page-size 1G
}

## node variant defaults
node {
  ## specify the preferred default variant
  #       default { variant avx512 }

  ## specify the preferred variant, for a given node
  #        ip4-rewrite { variant avx2 }
}


plugins {
        ## Adjusting the plugin path depending on where the VPP plugins are
        #       path /ws/vpp/build-root/install-vpp-native/vpp/lib/vpp_plugins
        path /usr/lib/x86_64-linux-gnu/vpp_plugins

        ## Disable all plugins by default and then selectively enable specific
plugins
        # plugin default { disable }
        plugin dpdk_plugin.so { enable }
        # plugin acl_plugin.so { enable }

        ## For QAT: Comment out all plugins below
        ## For IPSecMB: Uncomment plugin crypto_native_plugin.so
        plugin crypto_ipsecmb_plugin.so { disable }
        # plugin crypto_native_plugin.so { disable }
        # plugin crypto_sw_scheduler_plugin.so { disable }
}

## Statistics Segment
statseg {
    # socket-name <filename>, name of the stats segment socket
    #     defaults to /run/vpp/stats.sock
    # size <nnn>[KMG], size of the stats segment, defaults to 32mb
    size 32M
    # per-node-counters on | off, defaults to none
    per-node-counters off
    # update-interval <f64-seconds>, sets the segment scrape / update interval
}
```

a. Replace <sock0_port0> and <sock0_port1> with the corresponding PCI address of the VFs for PF0 and PF1 of socket 1

21. In directory ep0, create the following file called "`startup_qat_sw.conf`":

```
   nodaemon
   log /var/log/vpp/vpp.log
   full-coredump
   cli-listen /run/vpp/cli.sock
   gid vpp
   interactive
   exec /home/ep0/ipsec.cli
}

logging {
  size 4096
  default-log-level debug
  default-syslog-log-level debug
}

api-trace {
  ## This stanza controls binary API tracing. Unless there is a very strong
reason,
  ## please leave this feature enabled.
  On
  ## Additional parameters:
  ##
  ## To set the number of binary API trace records in the circular buffer,
configure nitems
  ##
  ## nitems <nnn>
  ##
  ## To save the api message table decode tables, configure a filename.
Results in /tmp/<filename>
  ## Very handy for understanding api message changes between versions,
identifying missing
  ## plugins, and so forth.
  ##
  ## save-api-table <filename>
}

api-segment {
  gid vpp
}

socksvr {
  default
}

cpu {
       ## In the VPP there is one main thread and optionally the user can
create worker(s)
       ## The main thread and worker thread(s) can be pinned to CPU core(s)
manually or automatically
```

```
       ## Manual pinning of thread(s) to CPU core(s)

       ## Set logical CPU core where main thread runs, if main core is not set
       ## VPP will use core 1 if available
       main-core 0

       ## Set logical CPU core(s) where worker threads are running
       corelist-workers 1-8

       ## Automatic pinning of thread(s) to CPU core(s)

       ## Sets number of CPU core(s) to be skipped (1 … N-1)
       ## Skipped CPU core(s) are not used for pinning main thread and working
thread(s).
       ## The main thread is automatically pinned to the first available CPU
core and worker(s)
       ## are pinned to next free CPU core(s) after core assigned to main
thread
       # skip-cores 4

       ## Specify a number of workers to be created
       ## Workers are pinned to N consecutive CPU cores while skippi"g "skip-
co"es" CPU core(s)
       ## and main thr'ad's CPU core
       # workers 4

       ## Set scheduling policy and priority of main and worker threads

       ## Scheduling policy options are: other (SCHED_OTHER), batch
(SCHED_BATCH)
       ## idle (SCHED_IDLE), fifo (SCHED_FIFO), rr (SCHED_RR)
       scheduler-policy fifo

       ## Scheduling priority is used only f"r "real-time policies (fifo and
rr),
       ## and has to be in the range of priorities supported for a particular
policy
       # scheduler-priority 50
}

buffers {
       ## Increase number of buffers allocated, needed only in scenarios with
       ## large number of interfaces and worker threads. Value is per numa
node.
       ## Default is 16384 (8192 if running unpriviledged)
       # buffers-per-numa 128000

       ## Size of buffer data area
       ## Default is 2048
       default data-size 2048

       ## Size of the memory pages allocated for buffer data
```

```
       ## Default will t‘y 'default-hugep’ge' th‘n 'defa’lt'
       ## you can also pass a size in K/M/G e.‘. ’8M'
       # page-size default-hugepage
}

dpdk {
       ## Change default settings for all interfaces
       dev default {
          ## Number of receive queues, enables RSS
          ## Default is 1
          num-rx-queues 4

          ## Number of transmit queues, Default is equal
          ## to number of worker threads or 1 if no workers treads
          num-tx-queues 4

          ## Number of descriptors in transmit and receive rings
          ## increasing or reducing number can impact performance
          ## Default is 1024 for both rx and tx
          num-rx-desc 4096
          num-tx-desc 4096

          ## VLAN strip offload mode for interface
          ## Default is off
          #vlan-strip-offload off

          ## TCP Segment Offload
          ## Default is off
          ## To enable TS‘, 'enable-tcp-udp-check’um' must be set
          tso off

          ## Devargs
          ## device specific init args
          ## Default is NULL
          # devargs safe-mode-support=1,pipeline-mode-support=1

          ## rss-queues
          ## set valid rss steering queues
          rss-queues 0-4
       }

       ## Whitelist specific interface by specifying PCI address and in
       ## addition specify custom parameters for this interface
       dev <sock0_port0> {
            workers 1-4
            name eth1
            num-rx-queues 4
       }
       dev <sock0_port1> {
            workers 5-8
```

```
            name eth2
            num-rx-queues 4
       }

       ## Blacklist specific device type by specifying PCI vendor:device
       ## Whitelist entries take precedence
       # blacklist 8086:10fb

       ## Disable multi-segment buffers, improves performance but
       ## disables Jumbo MTU support
       no-multi-seg
       ## Change hugepages allocation per-socket, needed only if there is need
for
       ## larger number of mbufs. Default is 256M on each detected CPU socket
       socket-mem 8192
       ## Disables UDP / TCP TX checksum offload. Typically needed for use
       ## faster vector PMDs (together with no-multi-seg)
       no-tx-checksum-offload
       ## Enable UDP / TCP TX checksum offload
       ## This is the reversed option 'f 'no-tx-checksum-offl'ad'
       # enable-tcp-udp-checksum
       ## Change UIO driver used by VPP, Options are: igb_uio, vfio-pci,
       ## uio_pci_generic or auto (default)
       uio-driver vfio-pci
       log-level debug
}

memory {
## Set the main heap size, default is 1G
        main-heap-size 1G

        ## Set the main heap page size. Default page size is OS default page
        ## which is in most cases 4K. if different page size is specified VPP
        ## will try to allocate main heap by using specified page size.
        ## special keywo'd 'default-hugep'ge' will use system default hugepage
        ## size
        main-heap-page-size 1G
}

## node variant defaults
node {
  ## specify the preferred default variant
  #       default { variant avx512 }

  ## specify the preferred variant, for a given node
  #        ip4-rewrite { variant avx2 }
}


plugins {
```

```
        ## Adjusting the plugin path depending on where the VPP plugins are
        #       path /ws/vpp/build-root/install-vpp-native/vpp/lib/vpp_plugins
        path /usr/lib/x86_64-linux-gnu/vpp_plugins

        ## Disable all plugins by default and then selectively enable specific
plugins
        # plugin default { disable }
        plugin dpdk_plugin.so { enable }
        # plugin acl_plugin.so { enable }

        ## For QAT: Comment out all plugins below
        ## For IPSecMB: Uncomment plugin crypto_native_plugin.so
        # plugin crypto_ipsecmb_plugin.so { disable }
         plugin crypto_native_plugin.so { disable }
        # plugin crypto_sw_scheduler_plugin.so { disable }
}

## Statistics Segment
statseg {
    # socket-name <filename>, name of the stats segment socket
    #     defaults to /run/vpp/stats.sock
    # size <nnn>[KMG], size of the stats segment, defaults to 32mb
    size 32M
    # per-node-counters on | off, defaults to none
    per-node-counters off
    # update-interval <f64-seconds>, sets the segment scrape / update interval
}
```

a. Replace <sock0_port0> and <sock0_port1> with the corresponding PCI address of the VFs for PF0 and PF1 of socket 1

22. Inside the directory ep0, create the following file called "`create_ipsec_cli_ep0.py`":

```
#!/usr/bin/env python3
import sys

def main( argv ):

    crypto_handler       = argv[1]
    crypto_alg           = argv[2]
    integ_alg            = argv[3]
    num_ipsec_tunnels    = argv[4]

    if not crypto_handler in ["ipsecmb", "native"]:
        print( "Unknown crypto handler: %s" % (crypto_handler) )
        return 1

    if not crypto_alg in ["aes-cbc-128", "aes-gcm-128"]:
        print( "Unknown crypto algorithm: %s" % (crypto_alg) )
        return 1
```

```
    if not integ_alg in ["none", "sha1-96"]:
        print( "Unknown integrity algorithm: %s" % (integ_alg) )
        return 1

    try:
        num_ipsec_tunnels = int( argv[4] )
    except:
        print( "Invalid number of IPSec tunnels: %s" % (num_ipsec_tunnels) )
        return 1

    if num_ipsec_tunnels <= 0:
        print( "Invalid number of IPSec tunnels: %d" % (num_ipsec_tunnels) )
        return 1

    if num_ipsec_tunnels >= 256:
        print( "Invalid number of IPSec tunnels: %d" % (num_ipsec_tunnels) )
        return 1

    output_f_name="ipsec.cli"
    num_interfaces       = 2
    mtu                  = 1518
    intf_ip_addrs        = ["192.168.105.2", "172.16.10.2"]
    neigh_ip_addrs       = ["192.168.105.3", "172.16.10.3"]
    neigh_mac_addrs      = ["<sock1_port0_vf_mac>", "<sock1_port1_vf_mac>"]
    neigh_subnets        = ["20.64.0.0/10", "10.192.0.0/10"]
    crypto_key           = "4339314b55523947594d6d3547666b45"
    integ_key            = "4339314b55523947594d6d3547666b45"
    start_sa             = 20
    start_spi            = 1000
    tunnel_src_prefix    = "10.128.0."
    tunnel_dst_prefix    = "10.192.0."
    ext_ip_prefix        = "10.64.0."
    reverse_spi          = False

    lines = []

    # Set MTU on interfaces
    for i in range( 1, num_interfaces + 1 ):
        lines.append( "set interface mtu %d eth%d\n" % (mtu, i) )
    lines.append( "\n" )

    # Set IP addresses on interfaces and enable promiscuous mode
    for i in range( 1, num_interfaces + 1 ):
        next_intf_ip_addr = intf_ip_addrs[i-1]
        lines.append( "set interface ip address eth%d %s/24\n" % (i, next intf i
p_addr) )
        lines.append( "set interface promiscuous on eth%d\n" % (i) )
    lines.append( "\n" )

    # Create IPSec Tunnels]
```

```
    next_sa = start_sa
    next_spi = start_spi
    for i in range( 1, num_ipsec_tunnels + 1 ):
        next_ip_tunnel_name = "ipip%d" % ( i - 1 )
        lines.append( "create ipip tunnel src %s%d dst %s%d\n" % (tunnel_src_pre
fix, i, tunnel_dst_prefix, i) )
        lines.append( "set interface state ipip%d up\n" % (i - 1) )
        if reverse_spi == True:
            lines.append( "ipsec sa add %d spi %d crypto-alg %s crypto-key %s in
teg-alg %s" % (next_sa, next_spi + 1, crypto_alg, crypto_key, integ_alg) )
        else:
            lines.append( "ipsec sa add %d spi %d crypto-alg %s crypto-key %s in
teg-alg %s" % (next_sa, next_spi, crypto_alg, crypto_key, integ_alg) )
        if integ_alg != "none":
            lines[-1] += " integ-key %s" % ( integ_key )
        lines[-1] += "\n"
        next_sa += 1
        next_spi += 1
        if reverse_spi == True:
            lines.append( "ipsec sa add %d spi %d crypto-alg %s crypto-key %s in
teg-alg %s" % (next_sa, next_spi - 1, crypto_alg, crypto_key, integ_alg) )
        else:
            lines.append( "ipsec sa add %d spi %d crypto-alg %s crypto-key %s in
teg-alg %s" % (next_sa, next_spi, crypto_alg, crypto_key, integ_alg) )
        if integ_alg != "none":
            lines[-1] += " integ-key %s" % ( integ_key )
        lines[-1] += "\n"
        lines.append( "ipsec tunnel protect %s sa-in %d sa-out %d\n" % (next_ip_
tunnel_name, next_sa-1, next_sa) )
        lines.append( "set interface ip address %s %s%d/32\n" % (next_ip_tunnel_
name, tunnel_src_prefix, i) )
        lines.append( "ip route add %s%d/32 via %s\n" % (ext_ip_prefix, i, next_
ip_tunnel_name) )
        lines.append( "\n" )
        next_sa += 1
        next_spi += 1

    # Set ARP table entries and IP route entries
    for i in range( 1, num_interfaces + 1 ):
        next_neigh_ip_addr = neigh_ip_addrs[i-1]
        next_neigh_mac_addr = neigh_mac_addrs[i-1]
        next_neigh_subnet = neigh_subnets[i-1]
        lines.append( "set ip neighbor eth%d %s %s\n" % (i, next_neigh_ip_addr,
next_neigh_mac_addr) )
        lines.append( "ip route add %s via %s eth%d\n" % (next neigh subnet, nex
t_neigh_ip_addr, i) )
    lines.append( "\n" )

    # Set interface state up
    for i in range( 1, num_interfaces + 1 ):
        lines.append( "set interface state eth%d up\n" % (i) )
    lines.append( "\n" )
```

```
    lines.append( "set crypto handler all openssl\n")
    lines.append( "set crypto handler all %s\n" % (crypto_handler) )

    with open( output_f_name, 'w' ) as output_file:
        output_file.writelines( lines )

if __name__ == "__main__":
sys.exit( main(sys.argv) )
```

a. Replace <sock1_port0_vf_mac> and <sock1_port1_vf_mac> with the MAC addresses of the VFs of the corresponding ports.

23. In the directory ep0, run the following command to generate the "`ipsec.cli`" file to use for VPP:

```
python3 create_ipsec_cli_ep0.py ipsecmb aes-cbc-128 sha1-96 128
```

24. In the directory ep1, create the following file called "`start_vpp_native.sh`":

```
#!/usr/bin/env bash
vpp -c /home/ep1/startup_native.conf
exit 0
```

25. In the directory ep1, create the following file called "`start_vpp_qat_sw.sh`":

```
#!/usr/bin/env bash
vpp -c /home/ep1/startup_qat_sw.conf
exit 0
```

26. In the directory ep1, create the following file called "`startup_native.conf`":

```
unix {
   nodaemon
   log /var/log/vpp/vpp.log
   full-coredump
   cli-listen /run/vpp/cli.sock
   gid vpp
   interactive
   exec /home/ep1/ipsec.cli
}

logging {
  size 4096
  default-log-level debug
  default-syslog-log-level debug
}

api-trace {
  ## This stanza controls binary API tracing. Unless there is a very strong
reason,
  ## please leave this feature enabled.
  on
  ## Additional parameters:
  ##
```

```
  ## To set the number of binary API trace records in the circular buffer,
configure nitems
  ##
  ## nitems <nnn>
  ##
  ## To save the api message table decode tables, configure a filename. Results in
/tmp/<filename>
  ## Very handy for understanding api message changes between versions,
identifying missing
  ## plugins, and so forth.
  ##
  ## save-api-table <filename>
}

api-segment {
  gid vpp
}

socksvr {
  default
}

cpu {
       ## In the VPP there is one main thread and optionally the user can create
worker(s)
       ## The main thread and worker thread(s) can be pinned to CPU core(s)
manually or automatically

       ## Manual pinning of thread(s) to CPU core(s)

       ## Set logical CPU core where main thread runs, if main core is not set
       ## VPP will use core 1 if available
       main-core 64

       ## Set logical CPU core(s) where worker threads are running
       corelist-workers 65-72

       ## Automatic pinning of thread(s) to CPU core(s)

       ## Sets number of CPU core(s) to be skipped (1 ... N-1)
       ## Skipped CPU core(s) are not used for pinning main thread and working
thread(s).
       ## The main thread is automatically pinned to the first available CPU core
and worker(s)
       ## are pinned to next free CPU core(s) after core assigned to main thread
       # skip-cores 4

       ## Specify a number of workers to be created
       ## Workers are pinned to N consecutive CPU cores while skipping "skip-
cores" CPU core(s)
       ## and main thread's CPU core
       # workers 4
```

```
       ## Set scheduling policy and priority of main and worker threads

       ## Scheduling policy options are: other (SCHED_OTHER), batch (SCHED_BATCH)
       ## idle (SCHED_IDLE), fifo (SCHED_FIFO), rr (SCHED_RR)
       scheduler-policy fifo

       ## Scheduling priority is used only for "real-time policies (fifo and rr),
       ## and has to be in the range of priorities supported for a particular
policy
       # scheduler-priority 50
}

buffers {
       ## Increase number of buffers allocated, needed only in scenarios with
       ## large number of interfaces and worker threads. Value is per numa node.
       ## Default is 16384 (8192 if running unpriviledged)
       # buffers-per-numa 128000

       ## Size of buffer data area
       ## Default is 2048
       default data-size 2048

       ## Size of the memory pages allocated for buffer data
       ## Default will try 'default-hugepage' then 'default'
       ## you can also pass a size in K/M/G e.g. '8M'
       # page-size default-hugepage
}

dpdk {
       ## Change default settings for all interfaces
       dev default {
          ## Number of receive queues, enables RSS
          ## Default is 1
          num-rx-queues 4

          ## Number of transmit queues, Default is equal
          ## to number of worker threads or 1 if no workers treads
          num-tx-queues 4

          ## Number of descriptors in transmit and receive rings
          ## increasing or reducing number can impact performance
          ## Default is 1024 for both rx and tx
          num-rx-desc 4096
          num-tx-desc 4096

          ## VLAN strip offload mode for interface
          ## Default is off
          #vlan-strip-offload off
```

```
         ## TCP Segment Offload
         ## Default is off
         ## To enable TSO, 'enable-tcp-udp-checksum' must be set
         tso off

         ## Devargs
         ## device specific init args
         ## Default is NULL
         # devargs safe-mode-support=1,pipeline-mode-support=1

         ## rss-queues
         ## set valid rss steering queues
         rss-queues 0-4
      }

      ## Whitelist specific interface by specifying PCI address and in
      ## addition specify custom parameters for this interface
      dev <sock0_port2> {
           workers 65-68
           name eth1
           num-rx-queues 4
      }
      dev <sock0_port3> {
           workers 69-72
           name eth2
           num-rx-queues 4
      }

      ## Blacklist specific device type by specifying PCI vendor:device
      ## Whitelist entries take precedence
      # blacklist 8086:10fb

      ## Disable multi-segment buffers, improves performance but
      ## disables Jumbo MTU support
      no-multi-seg
      ## Change hugepages allocation per-socket, needed only if there is need for
      ## larger number of mbufs. Default is 256M on each detected CPU socket
      socket-mem 8192
      ## Disables UDP / TCP TX checksum offload. Typically needed for use
      ## faster vector PMDs (together with no-multi-seg)
      no-tx-checksum-offload
      ## Enable UDP / TCP TX checksum offload
      ## This is the reversed option of 'no-tx-checksum-offload'
      # enable-tcp-udp-checksum
      ## Change UIO driver used by VPP, Options are: igb uio, vfio-pci,
      ## uio_pci_generic or auto (default)
      uio-driver vfio-pci
      log-level debug
}
```

```
memory {
## Set the main heap size, default is 1G
        main-heap-size 1G

        ## Set the main heap page size. Default page size is OS default page
        ## which is in most cases 4K. if different page size is specified VPP
        ## will try to allocate main heap by using specified page size.
        ## special keyword 'default-hugepage' will use system default hugepage
        ## size
        main-heap-page-size 1G
}

## node variant defaults
node {
  ## specify the preferred default variant
  #       default { variant avx512 }

  ## specify the preferred variant, for a given node
  #        ip4-rewrite { variant avx2 }
}


plugins {
        ## Adjusting the plugin path depending on where the VPP plugins are
        #       path /ws/vpp/build-root/install-vpp-native/vpp/lib/vpp_plugins
        path /usr/lib/x86_64-linux-gnu/vpp_plugins

        ## Disable all plugins by default and then selectively enable specific
plugins
        # plugin default { disable }
        plugin dpdk_plugin.so { enable }
        # plugin acl_plugin.so { enable }

        ## For QAT: Comment out all plugins below
        ## For IPSecMB: Uncomment plugin crypto_native_plugin.so
        plugin crypto_ipsecmb_plugin.so { disable }
        # plugin crypto_native_plugin.so { disable }
        # plugin crypto_sw_scheduler_plugin.so { disable }
}

## Statistics Segment
statseg {
    # socket-name <filename>, name of the stats segment socket
    #     defaults to /run/vpp/stats.sock
    # size <nnn>[KMG], size of the stats segment, defaults to 32mb
    size 32M
    # per-node-counters on | off, defaults to none
    per-node-counters off
    # update-interval <f64-seconds>, sets the segment scrape / update interval
}
```

a. Replace <sock0_port2> and <sock0_port3> with the corresponding PCI address of the VFs for PF0 and PF1 of socket 1

27. In the directory ep1, create the following file called "`startup_qat_sw.conf`":

```
unix {
   nodaemon
   log /var/log/vpp/vpp.log
   full-coredump
   cli-listen /run/vpp/cli.sock
   gid vpp
   interactive
   exec /home/ep1/ipsec.cli
}

logging {
  size 4096
  default-log-level debug
  default-syslog-log-level debug
}

api-trace {
  ## This stanza controls binary API tracing. Unless there is a very strong
reason,
  ## please leave this feature enabled.
  on
  ## Additional parameters:
  ##
  ## To set the number of binary API trace records in the circular buffer,
configure nitems
  ##
  ## nitems <nnn>
  ##
  ## To save the api message table decode tables, configure a filename. Results in
/tmp/<filename>
  ## Very handy for understanding api message changes between versions,
identifying missing
  ## plugins, and so forth.
  ##
  ## save-api-table <filename>
}

api-segment {
  gid vpp
}

socksvr {
  default
}

cpu {
```

```
        ## In the VPP there is one main thread and optionally the user can create
worker(s)
        ## The main thread and worker thread(s) can be pinned to CPU core(s)
manually or automatically

        ## Manual pinning of thread(s) to CPU core(s)

        ## Set logical CPU core where main thread runs, if main core is not set
        ## VPP will use core 1 if available
        main-core 64

        ## Set logical CPU core(s) where worker threads are running
        corelist-workers 65-72

        ## Automatic pinning of thread(s) to CPU core(s)

        ## Sets number of CPU core(s) to be skipped (1 ... N-1)
        ## Skipped CPU core(s) are not used for pinning main thread and working
thread(s).
        ## The main thread is automatically pinned to the first available CPU core
and worker(s)
        ## are pinned to next free CPU core(s) after core assigned to main thread
        # skip-cores 4

        ## Specify a number of workers to be created
        ## Workers are pinned to N consecutive CPU cores while skipping "skip-
cores" CPU core(s)
        ## and main thread's CPU core
        # workers 4

        ## Set scheduling policy and priority of main and worker threads

        ## Scheduling policy options are: other (SCHED_OTHER), batch (SCHED_BATCH)
        ## idle (SCHED_IDLE), fifo (SCHED_FIFO), rr (SCHED_RR)
        scheduler-policy fifo

        ## Scheduling priority is used only for "real-time policies (fifo and rr),
        ## and has to be in the range of priorities supported for a particular
policy
        # scheduler-priority 50
}

buffers {
        ## Increase number of buffers allocated, needed only in scenarios with
        ## large number of interfaces and worker threads. Value is per numa node.
        ## Default is 16384 (8192 if running unpriviledged)
        # buffers-per-numa 128000

        ## Size of buffer data area
        ## Default is 2048
        default data-size 2048
```

```
       ## Size of the memory pages allocated for buffer data
       ## Default will try 'default-hugepage' then 'default'
       ## you can also pass a size in K/M/G e.g. '8M'
       # page-size default-hugepage
}

dpdk {
       ## Change default settings for all interfaces
       dev default {
          ## Number of receive queues, enables RSS
          ## Default is 1
          num-rx-queues 4

          ## Number of transmit queues, Default is equal
          ## to number of worker threads or 1 if no workers treads
          num-tx-queues 4

          ## Number of descriptors in transmit and receive rings
          ## increasing or reducing number can impact performance
          ## Default is 1024 for both rx and tx
          num-rx-desc 4096
          num-tx-desc 4096

          ## VLAN strip offload mode for interface
          ## Default is off
          #vlan-strip-offload off

          ## TCP Segment Offload
          ## Default is off
          ## To enable TSO, 'enable-tcp-udp-checksum' must be set
          tso off

          ## Devargs
          ## device specific init args
          ## Default is NULL
          # devargs safe-mode-support=1,pipeline-mode-support=1

          ## rss-queues
          ## set valid rss steering queues
          rss-queues 0-4
       }

       ## Whitelist specific interface by specifying PCI address and in
       ## addition specify custom parameters for this interface
       dev <sock0_port2> {
            workers 65-68
            name eth1
            num-rx-queues 4
       }
```

```
       dev <sock0_port3> {
            workers 69-72
            name eth2
            num-rx-queues 4
       }

       ## Blacklist specific device type by specifying PCI vendor:device
       ## Whitelist entries take precedence
       # blacklist 8086:10fb

       ## Disable multi-segment buffers, improves performance but
       ## disables Jumbo MTU support
       no-multi-seg
       ## Change hugepages allocation per-socket, needed only if there is need for
       ## larger number of mbufs. Default is 256M on each detected CPU socket
       socket-mem 8192
       ## Disables UDP / TCP TX checksum offload. Typically needed for use
       ## faster vector PMDs (together with no-multi-seg)
       no-tx-checksum-offload
       ## Enable UDP / TCP TX checksum offload
       ## This is the reversed option of 'no-tx-checksum-offload'
       # enable-tcp-udp-checksum
       ## Change UIO driver used by VPP, Options are: igb_uio, vfio-pci,
       ## uio_pci_generic or auto (default)
       uio-driver vfio-pci
       log-level debug
}

memory {
## Set the main heap size, default is 1G
        main-heap-size 1G

        ## Set the main heap page size. Default page size is OS default page
        ## which is in most cases 4K. if different page size is specified VPP
        ## will try to allocate main heap by using specified page size.
        ## special keyword 'default-hugepage' will use system default hugepage
        ## size
        main-heap-page-size 1G
}

## node variant defaults
node {
  ## specify the preferred default variant
  #       default { variant avx512 }

  ## specify the preferred variant, for a given node
  #        ip4-rewrite { variant avx2 }
}
```

```
plugins {
        ## Adjusting the plugin path depending on where the VPP plugins are
        #       path /ws/vpp/build-root/install-vpp-native/vpp/lib/vpp_plugins
        path /usr/lib/x86_64-linux-gnu/vpp_plugins

        ## Disable all plugins by default and then selectively enable specific
plugins
        # plugin default { disable }
        plugin dpdk_plugin.so { enable }
        # plugin acl_plugin.so { enable }

        ## For QAT: Comment out all plugins below
        ## For IPSecMB: Uncomment plugin crypto_native_plugin.so
        # plugin crypto_ipsecmb_plugin.so { disable }
         plugin crypto_native_plugin.so { disable }
        # plugin crypto_sw_scheduler_plugin.so { disable }
}

## Statistics Segment
statseg {
    # socket-name <filename>, name of the stats segment socket
    #     defaults to /run/vpp/stats.sock
    # size <nnn>[KMG], size of the stats segment, defaults to 32mb
    size 32M
    # per-node-counters on | off, defaults to none
    per-node-counters off
    # update-interval <f64-seconds>, sets the segment scrape / update interval
}
```

a. Replace <sock0_port0> and <sock0_port1> with the corresponding PCI address of the VFs for PF0 and PF1 of socket 1

28. In the directory ep1, create the following file called "`create_ipsec_cli_ep1.py`":

```
#!/usr/bin/env python3
import sys

def main( argv ):

    crypto_handler       = argv[1]
    crypto_alg           = argv[2]
    integ_alg            = argv[3]
    num_ipsec_tunnels    = argv[4]

    if not crypto_handler in ["ipsecmb", "native"]:
        print( "Unknown crypto handler: %s" % (crypto_handler) )
        return 1

    if not crypto_alg in ["aes-cbc-128", "aes-gcm-128"]:
        print( "Unknown crypto algorithm: %s" % (crypto_alg) )
        return 1
```

```
    if not integ_alg in ["none", "sha1-96"]:
        print( "Unknown integrity algorithm: %s" % (integ_alg) )
        return 1

    try:
        num_ipsec_tunnels = int( argv[4] )
    except:
        print( "Invalid number of IPSec tunnels: %s" % (num_ipsec_tunnels) )
        return 1

    if num_ipsec_tunnels <= 0:
        print( "Invalid number of IPSec tunnels: %d" % (num_ipsec_tunnels) )
        return 1

    if num_ipsec_tunnels >= 256:
        print( "Invalid number of IPSec tunnels: %d" % (num_ipsec_tunnels) )
        return 1

    output_f_name="ipsec.cli"
    num_interfaces       = 2
    mtu                  = 1518
    intf_ip_addrs        = ["172.16.10.3", "192.168.115.2"]
    neigh_ip_addrs       = ["172.16.10.2", "192.168.115.3"]
    neigh_mac_addrs      = ["<sock1_port2_vf_mac>", "<sock1_port3_vf_mac>"]
    neigh_subnets        = ["10.128.0.0/10", "10.64.0.0/10"]
    crypto_key           = "4339314b55523947594d6d3547666b45"
    integ_key            = "4339314b55523947594d6d3547666b45"
    start_sa             = 20
    start_spi            = 1000
    tunnel_src_prefix    = "10.192.0."
    tunnel_dst_prefix    = "10.128.0."
    ext_ip_prefix        = "20.64.0."
    reverse_spi          = True

    lines = []

    # Set MTU on interfaces
    for i in range( 1, num_interfaces + 1 ):
        lines.append( "set interface mtu %d eth%d\n" % (mtu, i) )
    lines.append( "\n" )

    # Set IP addresses on interfaces and enable promiscuous mode
    for i in range( 1, num_interfaces + 1 ):
        next_intf_ip_addr = intf_ip_addrs[i-1]
        lines.append( "set interface ip address eth%d %s/24\n" % (i,
next_intf_ip_addr) )
        lines.append( "set interface promiscuous on eth%d\n" % (i) )
    lines.append( "\n" )

    # Create IPSec Tunnels]
```

```
    next_sa = start_sa
    next_spi = start_spi
    for i in range( 1, num_ipsec_tunnels + 1 ):
        next_ip_tunnel_name = "ipip%d" % ( i - 1 )
        lines.append( "create ipip tunnel src %s%d dst %s%d\n" %
(tunnel_src_prefix, i, tunnel_dst_prefix, i) )
        lines.append( "set interface state ipip%d up\n" % (i - 1) )
        if reverse_spi == True:
            lines.append( "ipsec sa add %d spi %d crypto-alg %s crypto-key %s
integ-alg %s" % (next_sa, next_spi + 1, crypto_alg, crypto_key, integ_alg) )
        else:
            lines.append( "ipsec sa add %d spi %d crypto-alg %s crypto-key %s
integ-alg %s" % (next_sa, next_spi, crypto_alg, crypto_key, integ_alg) )
        if integ_alg != "none":
            lines[-1] += " integ-key %s" % ( integ_key )
        lines[-1] += "\n"
        next_sa += 1
        next_spi += 1
        if reverse_spi == True:
            lines.append( "ipsec sa add %d spi %d crypto-alg %s crypto-key %s
integ-alg %s" % (next_sa, next_spi - 1, crypto_alg, crypto_key, integ_alg) )
        else:
            lines.append( "ipsec sa add %d spi %d crypto-alg %s crypto-key %s
integ-alg %s" % (next_sa, next_spi, crypto_alg, crypto_key, integ_alg) )
        if integ_alg != "none":
            lines[-1] += " integ-key %s" % ( integ_key )
        lines[-1] += "\n"
        lines.append( "ipsec tunnel protect %s sa-in %d sa-out %d\n" %
(next_ip_tunnel_name, next_sa-1, next_sa) )
        lines.append( "set interface ip address %s %s%d/32\n" %
(next_ip_tunnel_name, tunnel_src_prefix, i) )
        lines.append( "ip route add %s%d/32 via %s\n" % (ext_ip_prefix, i,
next_ip_tunnel_name) )
        lines.append( "\n" )
        next_sa += 1
        next_spi += 1

    # Set ARP table entries and IP route entries
    for i in range( 1, num_interfaces + 1 ):
        next_neigh_ip_addr = neigh_ip_addrs[i-1]
        next_neigh_mac_addr = neigh_mac_addrs[i-1]
        next_neigh_subnet = neigh_subnets[i-1]
        lines.append( "set ip neighbor eth%d %s %s\n" % (i, next_neigh_ip_addr,
next_neigh_mac_addr) )
        lines.append( "ip route add %s via %s eth%d\n" % (next neigh subnet,
next_neigh_ip_addr, i) )
    lines.append( "\n" )

    # Set interface state up
    for i in range( 1, num_interfaces + 1 ):
        lines.append( "set interface state eth%d up\n" % (i) )
    lines.append( "\n" )
```

```
    lines.append( "set crypto handler all openssl\n")
    lines.append( "set crypto handler all %s\n" % (crypto_handler) )

    with open( output_f_name, 'w' ) as output_file:
        output_file.writelines( lines )

if __name__ == "__main__":
    sys.exit( main(sys.argv) )
```

a. Replace <sock1_port2_vf_mac> and <sock1_port3_vf_mac> with the MAC addresses of the VFs of the corresponding NIC ports.

29. In the directory ep1, run the following command to generate the "`ipsec.cli`" file to use for VPP:
    ```
    python3 create_ipsec_cli_ep1.py ipsecmb aes-cbc-128 sha1-96 12
    ```
30. Run the "`setup_vpp_pods.sh`" script.
31. In one console window, run the following commands to start the VPP endpoints:
    ```
    kubectl exec vpp-ep0 -- /home/ep0/start_vpp_native.sh &
    kubectl exec vpp-ep1 -- /home/ep1/start_vpp_native.sh &
    ```
32. In a separate console window, run the following commands to start the DPDK testpmd application:
    ```
    kubectl exec -it dpdk-pod /bin/bash
    ./run_testpmd.sh
    ```
33. In a separate console window, run the following commands to start the T-Rex traffic generator:
    ```
    kubectl exec -it trex-pod /bin/bash
    cd /home/v3.00
    tmux new-session
    ./t-rex-64 -i
    <detach from the tmux-session>
    ./trex-console
    ```
34. In the T-Rex console interface, use the following commands:
    ```
    tui
    start -f stl/vpp_packets_p0.py -p 0 -m 5%
    start -f stl/vpp_packets_p1.py -p 1 -m 5%
    ```
35. Repeat the previous step using "stop" and changing the line rate parameter (-m 5%) to find the highest rate with zero packet loss at steady state. Observe the throughput results and record the total Tx pps amount in Mpps.
36. To stop the traffic, use the command "stop" and exit from the T-Rex console interface with the "quit" command twice.
37. Repeat steps 34-36 after modifying the "`stl/vpp_packets_p0.py`" and "`stl_vpp_packets_p1.py`" files to use different packet sizes according to the baseline results provided in Section 5.4.
    a. Use packet sizes 64, 128, 256, 512, 1024, 1280, and 1420
38. Exit from the T-Rex Pod, then stop the VPP endpoints:
    ```
    kubectl exec vpp-ep0 -- pkill vpp
    kubectl exec vpp-ep1 -- pkill vpp
    ```

39. Repeat the testing from steps 33-37 after using the following commands to start the second VPP configuration:

```
kubectl exec vpp-ep0 -- /home/ep0/start_vpp_qat_sw.sh &
kubectl exec vpp-ep1 -- /home/ep1/start_vpp_qat_sw.sh &
```

40. Use the following equation to calculate the traffic throughput in Gbps using the recorded Tx pps value:
    Throughput = ((packet size) + 20) * 0.008 * (Tx pps value)

5.5 Malconv AI

AI inference is used in network/security to help prevent advanced cyber-attacks. In order to improve the latency associated with this application, the Intel® Xeon® Scalable Processor contains technologies to accelerate AI inference such as AVX-512, AMX, and Vector Neural Network Instructions. The Malconv AI workload utilizes the TensorFlow deep-learning framework, Intel® oneAPI Deep Neural Network Library (oneDNN), Advanced Matrix Extensions (AMX), and Intel® Neural Compressor to improve the performance of the AI inference model.

The starting model for the Malconv AI workload is an open-source deep-learning model called Malconv which is given as a pre-trained Keras H5 format file. This model is used to detect malware by reading the raw execution bytes of files. An Intel optimized version of this h5 model is used for this workload. The performance of the model can be improved by various procedures including conversion to a floating point frozen model and using the Intel® Neural Compressor for post-training quantization to acquire BF16, INT8, and ONNX INT8 precision models.

Ensure that the results of the tests follow the expected results as shown in the following tables to baseline the performance of the platform.

Table 15. Malconv AI Plus Configuration Performance

Model Type	Inference Time (ms)
H5 w/o oneDNN	68.0
H5 with oneDNN	68.1
FP32 w/o oneDNN	25.0
FP32 with oneDNN	25.1
BF16 with AMX	22.9
BF16 with BF16	42.4
INT8 with AMX	16.3
INT8 with VNNI	23.0
ONNX INT8	42.4

Figure 9: Malconv AI Plus Configuration Performance Graph

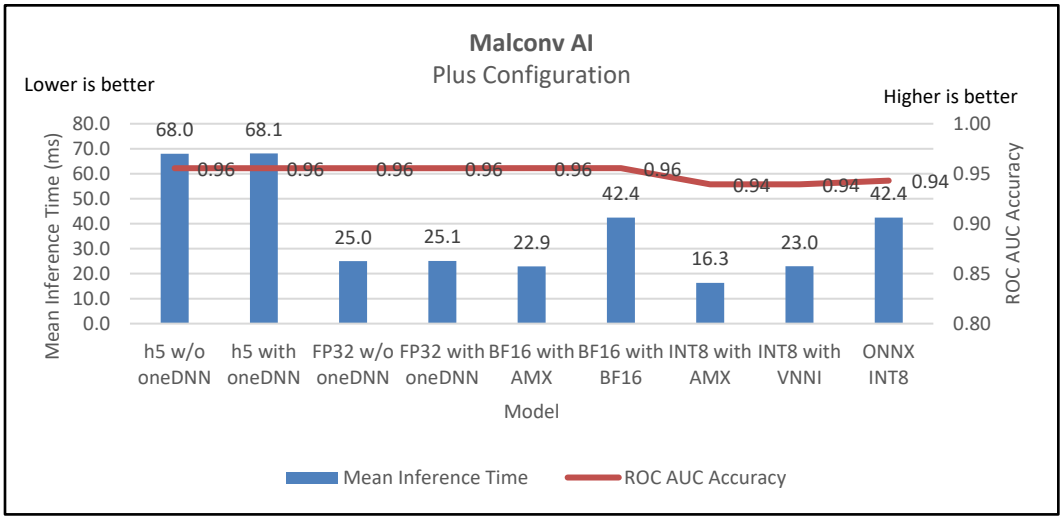
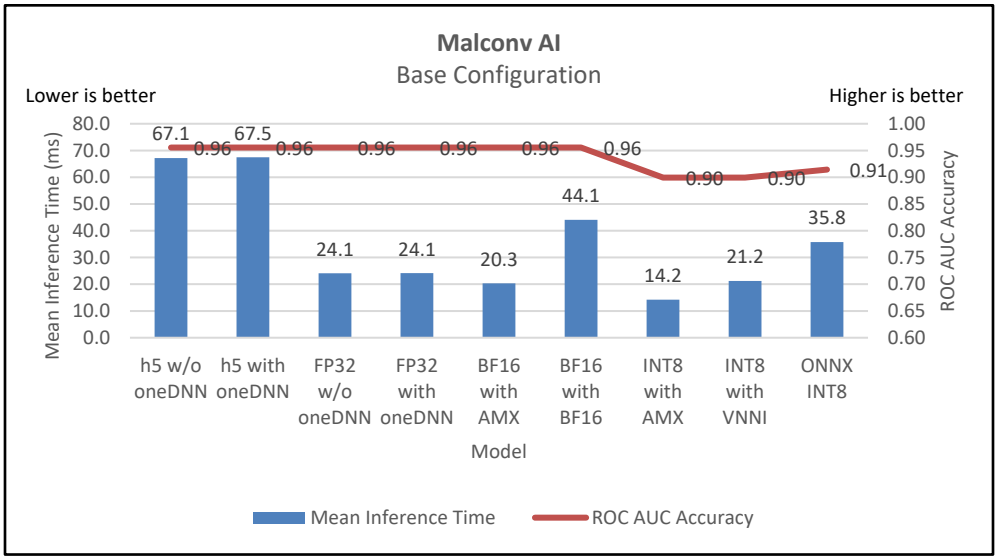


Table 16: Malconv AI Base Configuration Performance

Model Type	Inference Time (ms)
H5 w/o oneDNN	67.1
H5 with oneDNN	67.5
FP32 w/o oneDNN	24.1
FP32 with oneDNN	24.1
BF16 with AMX	20.3
BF16 with BF16	44.1
INT8 with AMX	14.2
INT8 with VNNI	21.2
ONNX INT8	35.8

Figure 10: Malconv AI Base Configuration Performance Graph



5.5.1 Malconv AI Test Methodology

Follow the instructions below to run the Malconv AI inference models:

1. Pull the intel/malconv-model-base Docker image:
   ```
   docker pull intel/malconv-model-base
   ```
2. Run the following commands:
   ```
   docker create –name=”tmp_$$” intel/malconv-model-base:latest
   docker export tmp_$$ > image-fs.tar
   docker rm tmp_$$
   tar -xf image-fs.tar
   ```
3. The “`IntelOptMalconv.h5`” file is needed while the other files are not necessary
4. Create the following file called “`malconv_pod.yaml`”

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: malconv-ai
  name: malconv-ai
spec:
  containers:
  - args:
    - /bin/bash
    image: quay.io/centos/centos:stream8
    name: malconv-ai
    stdin: true
    securityContext:
      privileged: true
  restartPolicy: Always
```

5. Create the following file called “`setup_malconv_ai.sh`”:

```
#!/bin/bash

kubectl delete pod malconv-ai

kubectl create -f malconv_pod.yaml
sleep 20

kubectl cp install_malconv_ai.sh malconv-ai:/home
kubectl cp h5_to_saved.py malconv-ai:/home
kubectl cp analyze_scores.py malconv-ai:/home
kubectl cp freeze_graph.py malconv-ai:/home
kubectl cp infer_test.py malconv-ai:/home
kubectl cp malconvBF16.yaml malconv-ai:/home
kubectl cp malconvINT8.yaml malconv-ai:/home
kubectl cp onnx_quantize.py malconv-ai:/home
kubectl cp onnx.yaml malconv-ai:/home
kubectl cp quantize.py malconv-ai:/home
kubectl cp datasets malconv-ai:/home
kubectl cp test_malconv_ai.sh malconv-ai:/home
kubectl cp IntelOptMalconv.h5 malconv-ai:/home
kubectl exec malconv-ai -- /home/install_malconv_ai.sh
```

6. Create the following file called "`h5_to_saved.py`" to first convert the Keras h5 format to the SavedModel format:

```
import tensorflow as tf
import argparse
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('-m', '--input_model', type=str, dest='input_model',
    help='path of H5 model', required=True)
    parser.add_argument('-o', '--output_path', type=str, dest='output_path',
    help='path of output file', required=True)
    args = parser.parse_args()
    return args

if __name__ == '__main__':
    args = parse_args()
    model = tf.keras.models.load_model(args.input_model)
   model.save(args.output_path)
```

7. Create the following file called "`freeze_graph.py`" to convert the SavedModel to the FP32 frozen graph model:

```
import tensorflow as tf
from tensorflow.python.saved_model import signature_constants
from tensorflow.python.training import saver
from tensorflow.python.framework import convert_to_constants
from tensorflow.core.protobuf import config_pb2
from tensorflow.python.grappler import tf_optimizer
from tensorflow.core.protobuf import meta_graph_pb2
from tensorflow.python.platform import gfile
from tensorflow.python.eager import context
import sys
assert context.executing_eagerly()
if len(sys.argv) != 3:
    print('Usage:')
    print(f'\tpython3 {sys.argv[0]} model_path output_pbfile')
    sys.exit(1)

model_path=sys.argv[1]
output_pbfile=sys.argv[2]
```

```
model = tf.keras.models.load_model(model_path)
model.summary()

func = model.signatures[signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY]
frozen_func = convert_to_constants.convert_variables_to_constants_v2(func)

grappler_meta_graph_def = saver.export_meta_graph(
    graph_def=frozen_func.graph.as_graph_def(), graph=frozen_func.graph)

# Add a collection 'train_op' so that Grappler knows the outputs.
fetch_collection = meta_graph_pb2.CollectionDef()
for array in frozen_func.inputs + frozen_func.outputs:
    fetch_collection.node_list.value.append(array.name)

grappler_meta_graph_def.collection_def["train_op"].CopyFrom(fetch_collection)

grappler_session_config = config_pb2.ConfigProto()
rewrite_options = grappler_session_config.graph_options.rewrite_options
rewrite_options.min_graph_nodes = -1
opt = tf_optimizer.OptimizeGraph(grappler_session_config,
grappler_meta_graph_def, graph_id=b"tf_graph")

f = gfile.GFile(output_pbfile, 'wb')
f.write(opt.SerializeToString())
```

8. Create the following YAML config file called "`malconvBF16.yaml`":

```
version: 1.0
model:
  name: malconv
  framework: tensorflow
  inputs: input_1
  outputs: Identity

quantization:
  model_wise:
    activation:
      granularity: per_tensor
      dtype: bf16
      algorithm: minmax
    weight:
      granularity: per_channel
      dtype: bf16
      algorithm: minmax

tuning:
  accuracy_criterion:
    relative: 0.01
  exit_policy:
    timeout: 0
    max_trials: 100
  random_seed: 9527
```

9. Create the following YAML config file called "`malconvINT8.yaml`":

```
version: 1.0
model:
  name: malconv
  framework: tensorflow
  inputs: input_1
  outputs: Identity

quantization:
```

```
  model_wise:
    activation:
      granularity: per_tensor
      dtype: int8
      algorithm: minmax
    weight:
      granularity: per_channel
      dtype: int8
      algorithm: minmax

tuning:
  accuracy_criterion:
    relative: 0.01
  exit_policy:
    timeout: 0
    max_trials: 100
  random_seed: 9527
```

10. Create the following file called "`quantize.py`" to quantize the FP32 frozen model with Intel® Neural Compressor:

```
import os
import argparse
import numpy as np
from neural_compressor.experimental import Quantization, common
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('-m', '--input_model', type=str, dest='input_model',
help='path of frozen fp32 model', required=True)
    parser.add_argument('-c', '--input_config', type=str, dest='input_config',
help='path of yaml config file', required=True)
    parser.add_argument('-i', '--input_path', type=str, dest='input_path',
help='path of input dataset', required=True)
    parser.add_argument('-o', '--output_file', type=str, dest='output_file',
help='path of output file', required=True)
    args = parser.parse_args()
    return args

def load_dataset(input_path):
    result = []
    mal_path = os.path.join(input_path, 'MALICIOUS')
    if os.path.exists(mal_path):
        mal_files = [(1, os.path.join(mal_path, fp)) for fp in
os.listdir(mal_path)]
        result.extend(mal_files)

    clean_path = os.path.join(input_path, 'KNOWN')
    if os.path.exists(clean_path):
        clean_files = [(0, os.path.join(clean_path, fp)) for fp in
os.listdir(clean_path)]
        result.extend(clean_files)
    return result

def read_file(filepath: str, expect_size: int):
    if filepath[-4:] == '.npy':
        data = np.load(filepath, allow_pickle=True)
    else:
        data = np.fromfile(filepath, np.ubyte)
    if data.size < expect_size:
        data = np.pad(data, (0, expect_size - data.size), 'constant',
constant_values=(0, 0))
    else:
        data = data[:expect_size]
```

```
    return np.array([data])

class Dataset:
    def __init__(self, input_path):
        self.batch_size = 32
 self.dataset = load_dataset(input_path)

    def __iter__(self):
        for label, filepath in self.dataset:
            data = read_file(filepath, expect_size=1048576)
            yield data, label

    def __len__(self):
        return len(self.dataset)

if __name__ == '__main__':
    os.environ['TF_ENABLE_ONEDNN_OPTS'] = '1'
    os.environ['ONEDNN_MAX_CPU_ISA'] = 'AVX512_CORE_AMX'
    os.environ['DNNL_ENABLE_MAX_CPU_ISA'] = '1'
    args = parse_args()
    quantizer = Quantization(args.input_config)
    quantizer.model = common.Model(args.input_model)
    quantizer.calib_dataloader = Dataset(args.input_path)
    q_model = quantizer.fit()
    q_model.save(args.output_file)
```

11. Create the following file called "`onnx_quantize.py`":

```
import os
import argparse
import numpy as np
import onnx
from neural_compressor.experimental import Quantization, common
from neural_compressor import options

def load_dataset(input_path):
    result = []
    mal_path = os.path.join(input_path, 'MALICIOUS')
    if os.path.exists(mal_path):
        mal_files = [(1, os.path.join(mal_path, fp)) for fp in
os.listdir(mal_path)]
        result.extend(mal_files)

    clean_path = os.path.join(input_path, 'KNOWN')
    if os.path.exists(clean_path):
        clean_files = [(0, os.path.join(clean_path, fp)) for fp in
os.listdir(clean_path)]
        result.extend(clean_files)

    return result

def read_file(filepath: str, expect_size: int):
    if filepath[-4:] == '.npy':
        data = np.load(filepath, allow_pickle=True)
    else:
        data = np.fromfile(filepath, np.ubyte)

    if data.size < expect_size:
        data = np.pad(data, (0, expect_size - data.size), 'constant',
constant_values=(0, 0))
```

```
    else:
        data = data[:expect_size]
    data = np.array([data])
    return data.astype(np.float32)

class Dataset:
    def __init__(self, input_path):
        self.batch_size = 32
        self.dataset = load_dataset(input_path)

    def __iter__(self):
        for label, filepath in self.dataset:
            data = read_file(filepath, expect_size=1048576)
            yield data, label

    def __len__(self):
        return len(self.dataset)

def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('-m', '--input_model', type=str,
dest='input_model', help='path of frozen fp32 model', required=True)
    parser.add_argument('-c', '--input_config', type=str,
dest='input_config', help='path of yaml config file', required=True)
    parser.add_argument('-i', '--input_path', type=str,
dest='input_path', help='path of input dataset', required=True)
    parser.add_argument('-o', '--output_file', type=str,
dest='output_file', help='path of output file', required=True)
    args = parser.parse_args()
    return args

class Quantize:
    def __init__(self):
        self.args = parse_args()
        self._create_quantizer()

    def run(self):
        q_model = self.quantizer()
        q_model.save(self.args.output_file)

    def _create_quantizer(self):
        args = self.args
        model = onnx.load(args.input_model)
        options.onnxrt.graph_optimization.level = 'ENABLE_BASIC'
        self.quantizer = Quantization(args.input_config)
        self.quantizer.model = common.Model(model)
        self.quantizer.calib_dataloader = Dataset(args.input_path)

def main():
    Quantize().run()

if __name__ == '__main__':
    main()
```

12. Create the following file called "`onnx.yaml`":

```
version: 1.0
```

```
model:
  name: malconv
  framework: onnxrt_qlinearops
  inputs: input_1
  outputs: Identity

quantization:
  approach: post_training_static_quant

tuning:
  accuracy_criterion:
    relative: 0.01
  exit_policy:
    timeout: 0
    max_trials: 1
  random_seed: 9527
```

13. Create the following file called "`infer_test.py`":

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="-1"
import time
import socket
import argparse
import numpy as np
from progress.bar import Bar
from analyze_scores import analyze_scores

def read_file(filepath, expect_size=1048576):
    if filepath[-4:] == '.npy':
        data = np.load(filepath, allow_pickle=True)
    else:
        data = np.fromfile(filepath, np.ubyte)
    if data.size < expect_size:
        data = np.pad(data, (0, expect_size - data.size), 'constant',
constant_values=(0, 0))
    else:
        data = data[:expect_size]

    return np.array([data])

class H5Model:
    def __init__(self, h5_path):
        import tensorflow as tf
        self.model = tf.keras.models.load_model(h5_path)

    def predict(self, input_data):
        start = time.time()
        result = self.model.predict(input_data)
        finish = time.time()
        return result[0], 1000 * (finish - start)

class SavedModel:
    def __init__(self, save_model_dir):
        import tensorflow as tf
        self.session = tf.compat.v1.Session()
```

```
        meta_graph_def =
tf.compat.v1.saved_model.loader.load(self.session, ['serve'],
save_model_dir)
        signature = meta_graph_def.signature_def
        serving_default = signature['serving_default']
        self.X = serving_default.inputs['input_1'].name
        self.y = serving_default.outputs['dense_2'].name

    def predict(self, input_data):
        start = time.time()
        result = self.session.run(self.y, feed_dict={self.X:
input_data})
        finish = time.time()
        return result[0][0], 1000 * (finish - start)

class ONNXModel:
    def __init__(self, onnx_file, num_cores=1):
        import onnxruntime as ort
        self.onnx_file = onnx_file
        if num_cores > 0:
            sess_options = ort.SessionOptions()
            sess_options.intra_op_num_threads = num_cores
            sess_options.execution_mode =
ort.ExecutionMode.ORT_SEQUENTIAL
            sess_options.graph_optimization_level =
ort.GraphOptimizationLevel.ORT_ENABLE_ALL
            self.session = ort.InferenceSession(self.onnx_file,
sess_options=sess_options, providers=['CPUExecutionProvider'])
        else:
            print('ONNXModel: not specify session options')
            self.session = ort.InferenceSession(self.onnx_file,
providers=['CPUExecutionProvider'])
        self.input_name = self.session.get_inputs()[0].name
        self.output_name = self.session.get_outputs()[0].name

    def predict(self, input_data):
        float32_data = input_data.astype(np.float32)
        start = time.time()
        result = self.session.run([self.output_name],
{self.input_name: float32_data})
        finish = time.time()
        return result[0][0][0], 1000 * (finish - start)

class FrozenModel:
    def __init__(self, pb_filepath, config=None):
        import tensorflow as tf
        from tensorflow.python.client import timeline
        import json

        self.timeline = timeline
        self.json = json
        graph = tf.Graph()
        with graph.as_default():
            graph_def = tf.compat.v1.GraphDef()
            with open(pb_filepath, "rb") as f:
                self.model_path = pb_filepath
                graph_def.ParseFromString(f.read())
                _ = tf.import_graph_def(graph_def, name='')
```

```
                self.session = tf.compat.v1.Session(config=config,
graph=graph)
                self.input_t1 = graph.get_tensor_by_name("input_1:0")
                self.output_data =
graph.get_tensor_by_name("Identity:0")
                self.run_options =
tf.compat.v1.RunOptions(trace_level=tf.compat.v1.RunOptions.FULL_TRAC
E)
                self.run_metadata = tf.compat.v1.RunMetadata()

    def predict(self, input_data):
        result = self.session.run(self.output_data,
feed_dict={self.input_t1: input_data}, options=self.run_options,
run_metadata=self.run_metadata)
        # calculation infer time from timeline
        # much more accurate than time.time() method.
        tl = self.timeline.Timeline(self.run_metadata.step_stats)
        ctf = tl.generate_chrome_trace_format()
        tmp = self.json.loads(ctf)
        try:
            # latency = last_op_start_time + last_op_duration -
first_op_start times
            latency = tmp['traceEvents'][-1]['ts'] +
tmp['traceEvents'][-1]['dur'] - tmp['traceEvents'][3]['ts']
        except:
            # may not have last_op_duration during warm up stage.
            # thus use this to catch the exception. will not be
calculated into the final results.
            latency = tmp['traceEvents'][-1]['ts'] -
tmp['traceEvents'][3]['ts']
        return result[0][0], latency/1000

class TFLiteModel:
    def __init__(self, tflite_file):
        import tensorflow as tf
        self.interpreter = tf.lite.Interpreter(tflite_file)
        self.interpreter.allocate_tensors()

    def predict(self, input_data):
        start = time.time()
        input_data = input_data.astype(np.float32)
        input_details = self.interpreter.get_input_details()
        output_details = self.interpreter.get_output_details()
        self.interpreter.set_tensor(input_details[0]['index'],
input_data)
        self.interpreter.invoke()
        result =
self.interpreter.get_tensor(output_details[0]['index'])
        finish = time.time()
        return result[0][0], 1000 * (finish - start)

class TestOnDataset:
    def __init__(self, model, input_path, num_files):
        self.model = model
        self.threshold = 0.99
        self.avg_infer_time = None
        self.standard_deviation = None

        self.all_files = []
```

```
        mal_path = os.path.join(input_path, 'MALICIOUS')
        mal_files = [(1, os.path.join(mal_path, fp)) for fp in
os.listdir(mal_path)]
        mal_files = mal_files[0: int(num_files/2)] if num_files>0
else mal_files[0:]
        self.all_files.extend(mal_files)

        clean_path = os.path.join(input_path, 'KNOWN')
        clean_files = [(0, os.path.join(clean_path, fp)) for fp in
os.listdir(clean_path)]
        clean_files = clean_files[0:int(num_files/2)] if num_files>0
else clean_files[0:]
        self.all_files.extend(clean_files)

    def run(self):
        input_tensor = read_file(self.all_files[0][1])
        for _ in range(30):
            self.model.predict(input_tensor)

        all_infer_time = []
        files, scores, pred, all_y = [], [], [], []
        bar = Bar('Progress... ', max=len(self.all_files))
        for label, filepath in self.all_files:
            int8_data = read_file(filepath)
            score, infer_time = self.model.predict(int8_data)
            all_infer_time.append(infer_time)
            files.append(filepath)
            scores.append(score)
            pred.append(int(score >= self.threshold))
            all_y.append(label)
            bar.next()
        bar.finish()

        self.avg_infer_time = np.mean(all_infer_time)
        self.standard_deviation = np.std(all_infer_time)

        print(f'average inference time: {self.avg_infer_time} ms')
        print(f'standard deviation: {self.standard_deviation} ms')
        print(f'filecount: {len(self.all_files)}')

        analyze_scores(
            all_data=[{'Filename': files, 'Score': scores, 'Predict':
pred, 'Actual': all_y}],
            labels=[socket.gethostname()],
            ref_fprs=[0.05, 0.01],
        )

def load_model(model_path, num_cores):
    if model_path[-2:] == 'h5':
        return H5Model(model_path)
    if model_path[-4:] == 'onnx':
        return ONNXModel(model_path, num_cores)
    if model_path[-6:] == 'tflite':
        return TFLiteModel(model_path)
    if os.path.isdir(model_path):
        return SavedModel(model_path)
    return FrozenModel(model_path)

def main():
```

```
    parser = argparse.ArgumentParser()
    parser.add_argument(
        '-m', '--model_path', type=str, dest='model_path',
help='model path', required=True)
    parser.add_argument(
        '-i', '--input_path', type=str, dest='input_path',
help='input dataset path', required=True)
    parser.add_argument(
        '-c', '--num_cores', type=int, dest='num_cores', help='number
of cores for ONNX runtime', default=1)
    parser.add_argument(
        '-n', '--number of files', type=int, dest='num_files',
help='number of files to be tested', default=100)

    args = parser.parse_args()
    model = load_model(args.model_path, args.num_cores)
    TestOnDataset(model, args.input_path, args.num_files).run()

if __name__ == '__main__':
    main()
```

14. Create the following file called "`analyze_scores.py`":

```
import os
import csv
import argparse
import numpy as np
from collections import defaultdict
from sklearn.metrics import f1_score, roc_curve, roc_auc_score,
confusion_matrix

def float2string(inp):
    return ('%.15f' % inp).rstrip('0').rstrip('.')

def format_predict_data(fields, prediction):
    result = defaultdict(list)

    for row in prediction:
        for field, value in zip(fields, row):
            result[field].append(value)

    result['Score'] = [float(x) for x in result['Score']]
    result['Predict'] = [int(x) for x in result['Predict']]
    result['Actual'] = [int(x) for x in result['Actual']]

    return result

def read_predict_file(path: str) -> dict:
    with open(path, 'r') as csv_file:
        csv_reader = csv.reader(csv_file)
        _ = next(csv_reader)
        fields = next(csv_reader)
        return format_predict_data(fields, csv_reader)

def apply_threshold(scores: list, threshold: float) -> list:
    return [int(score >= threshold) for score in scores]

def recall_specificity_at_thresh(y_scores, y_test, threshold,
adjusted_ben=None, adjusted_mal=None):
    predict_discrete = apply_threshold(y_scores, threshold)
    cm = confusion_matrix(y_test, predict_discrete, labels=[0, 1])

    if adjusted_ben:
```

```
        assert adjusted_ben >= (cm[0][0] + cm[0][1])
        cm[0][0] += (adjusted_ben - (cm[0][0] + cm[0][1])) # Count them as TNs

    if adjusted_mal:
        assert adjusted_mal >= (cm[1][0] + cm[1][1]))
        cm[1][0] += (adjusted_mal - (cm[1][0] + cm[1][1])) # Count them as FNs

    tn, fp, fn, tp = cm.ravel()

    recall = (tp * 100.0) / float(fn + tp) if float(fn + tp) != 0 else 100.0
    specificity = (tn * 100.0) / float(fp + tn) if float(fp + tn) != 0 else
100.0

    return recall, specificity

def parse_args():
    parser = argparse.ArgumentParser(
        description='Do analysis on computed malicious class scores.')
    parser.add_argument(
        '--pred_fps',
        nargs='+',
        help=('All filepaths leading to prediction files you wish compare. '
            'Separate each with a space.'))
    parser.add_argument(
        '--labels',
        help=('Labels for prediction files you wish to see in the plot '
            'legend. Supply one for each prediction file. Separate each '
            'with a comma.'))
    parser.add_argument(
        '--cust_threshs',
        default='',
        required=False,
        metavar='thresh1,thresh2,...',
        help=('All custom threhsolds you would like to test. Separate each '
            'with a comma.'))
    parser.add_argument(
        '--ref_fprs',
        default='',
        required=False,
        metavar='fpr1,fpr2,...',
        help=('All FPRs which you want to discover corresponding recall. '
            'Separate each with a comma.'))
    return parser.parse_args()

class Analyzer:
    def __init__(self, **kwargs):
        self.roc_data = []
        self.labels = kwargs['labels']
        self.ref_fprs = kwargs['ref_fprs']
        self.custom_thresholds = kwargs['custom_thresholds'] if
'custom_thresholds' in kwargs else None

        if 'pred_fps' in kwargs:
            self.all_data = self._read_predict_files(kwargs['pred_fps'])
        else:
            self.all_data = kwargs['all_data']

    def run(self):
        self._print_header()
        self._compute_custom_threshold_stats()
        self._compute_roc_curves()
        self._compute_tprs()
        self._print_tail()

    @staticmethod
    def _print_header():
```

```
        print('-' * 80)

    @staticmethod
    def _print_tail():
        print('\nExiting...')
        print('-' * 80)

    def _read_predict_files(self, pred_fps):
        print('Reading in predictions files...')
        result = []
        for label, filepath in zip(self.labels, pred_fps):
            result.append(read_predict_file(filepath))
            print(f'\tRead in {label} predictions: {filepath}!')
        return result

    def _compute_custom_threshold_stats(self) -> None:
        if not self.custom_thresholds:
            return

        print('\nComputing custom threshold stats...')
        for label, data in enumerate(self.labels, self.all_data):
            print(f'\n--> Using predictions with label \'{label}\':')
            for threshold in self.custom_thresholds:
                print(f'----> Stats using custom threshold
{float2string(threshold)}...')
                r, s = recall_specificity_at_thresh(data['Score'],
data['Actual'], threshold)
                predict = apply_threshold(data['Score'], thresh)
                f1 = f1_score(data['Actual'], predict)
                print(f'------> Recall: {r:%.6f}, Specificity: {s:%.6f}, F1:
{f1:%.6f}')

    def _compute_roc_curves(self):
        print('\nComputing ROC curves...')
        for label, data in zip(self.labels, self.all_data):
            print(f'\n--> Using predictions with label \'{label}\':')
            fpr, tpr, thresholds = roc_curve(data['Actual'], data['Score'])
            self.roc_data.append((fpr, tpr, thresholds))
            auc = roc_auc_score(data['Actual'], data['Score'])
            print(f'----> ROC AUC: {float2string(auc)}')

    def _compute_tprs(self):
        if not self.ref_fprs:
            return

        print('\nComputing TPRs at reference FPRs...')
        for label, data, (fpr, tpr, thresholds) in zip(self.labels,
self.all_data, self.roc_data):
            print(f'\n--> Using predictions with label \'{label}\':')
            for ref_fpr in self.ref_fprs:
                print(f'----> Stats using reference FPR <=
{float2string(ref_fpr)}...')
                idx = np.sum(fpr <= ref_fpr) - 1
                print(f'------> Threshold: {thresholds[idx]:.10f}, FPR:
{fpr[idx]:.10f}, TPR {tpr[idx]:.10f}')

def analyze_scores(**kwargs):
    if 'pred_fps' in kwargs or 'all_data' in kwargs:
        Analyzer(**kwargs).run()
    else:
        raise Exception('No prediction file or data found!')

def main():
    args = parse_args()

    for fp in args.pred_fps:
```

```
        assert (os.path.isfile(fp))

    analyze_scores(
        pred_fps=args.pred_fps,
        labels=args.labels.strip().split(','),
        ref_fprs=[float(i) for i in args.ref_fprs.strip().split(',') if i !=
''],
        custom_thresholds=[float(i) for i in
args.cust_threshs.strip().split(',') if i != ''],
    )

if __name__ == '__main__':
    main()
```

15. Create the following file called "`test_malconv_ai.sh`":

```
#!/bin/bash

cd /home

set -e

num_loop=$(seq 3)
dataset=/home/datasets

export TF_CPP_MIN_LOG_LEVEL=3

function random_sleep {
    echo ""
    SEC=$[($RANDOM % 5) + 10 ]
    echo "[$(date +"%Y-%m-%d %T")] Sleep $SEC seconds..."
    eval "sleep $SEC"
}

function exec_cmd {
    echo "[$(date +"%Y-%m-%d %T")] Running CMD: $*"
    eval $*
    STATUS=$?
}

cores_per_socket=$(  lscpu | grep -E '^Thread|^Core|^Socket|^CPU\(' | sed -n
'3p' |awk '{print $4}' )

# Models without oneDNN acceleration
declare -a tf_models=("malconv.h5" "malconv.fp32.pb")

for model in "${tf_models[@]}"
do
    # Test without oneDNN
    random_sleep
    # Run one instance on each core of the socket
    for (( core=0; core<=$(( $cores_per_socket - 1 )); core++ ))
    do
        exec_cmd "TF_ENABLE_ONEDNN_OPTS=0 numactl -C $core -m 0 python3
infer_test.py -i $dataset -m $model -n 1000 >
logs/${model}_no_dnn_core${core}.log" &
    done

    # wait for all the commands to finish
    while [ $(ps -ef | grep "python3 infer_test.py" | wc -l) != 1 ];
    do
        sleep 1
    done

    # Test with oneDNN
```

```
    random_sleep
    # Run one instance on each core of the socket
    for (( core=0; core<=$(( $cores_per_socket - 1 )); core++ ))
    do
        exec_cmd "TF_ENABLE_ONEDNN_OPTS=1 numactl -C $core -m 0 python3
infer_test.py -i $dataset -m $model -n 1000 > logs/${model}_core${core}.log" &
    done

    # wait for all the commands to finish
    while [ $(ps -ef | grep "python3 infer_test.py" | wc -l) != 1 ];
    do
        sleep 1
    done
done

# Enable oneDNN and ISA AVX512_core_AMX
declare -a tf_models_amx=("malconv.bf16.pb" "malconv.int8.pb")

for model in "${tf_models_amx[@]}"
do
    random_sleep
    # Run one instance on each core of the socket
    for (( core=0; core<=$(( $cores_per_socket - 1 )); core++ ))
    do
        exec_cmd "TF_ENABLE_ONEDNN_OPTS=1 ONEDNN_MAX_CPU_ISA=AVX512_CORE_AMX
numactl -C $core -m 0 python3 infer_test.py -i $dataset -m $model -n 1000 >
logs/${model}_amx_core${core}.log" &
    done

    # wait for all the commands to finish
    while [ $(ps -ef | grep "python3 infer_test.py" | wc -l) != 1 ];
    do
        sleep 1
    done
done

# Test BF16 model with AVX512_CORE_BF16
random_sleep
for (( core=0; core<=$(( $cores_per_socket - 1 )); core++ ))
do
        exec_cmd "TF_ENABLE_ONEDNN_OPTS=1 ONEDNN_MAX_CPU_ISA=AVX512_CORE_BF16
numactl -C $core -m 0 python3 infer_test.py -i $dataset -m malconv.bf16.pb -n
1000 > logs/malconv.bf16.pb_bf16_core${core}.log" &
done

# wait for all the commands to finish
while [ $(ps -ef | grep "python3 infer_test.py" | wc -l) != 1 ];
do
    sleep 1
done


# Test INT8 model with AVX512_CORE_VNNI
random_sleep
for (( core=0; core<=$(( $cores_per_socket - 1 )); core++ ))
do
        exec_cmd "TF_ENABLE_ONEDNN_OPTS=1 ONEDNN_MAX_CPU_ISA=AVX512_CORE_VNNI
numactl -C $core -m 0 python3 infer_test.py -i $dataset -m malconv.int8.pb -n
1000 > logs/malconv.int8.pb_vnni_core${core}.log" &
done

# wait for all the commands to finish
while [ $(ps -ef | grep "python3 infer_test.py" | wc -l) != 1 ];
do
    sleep 1
done
```

```
# Test ONNX
random_sleep
for (( core=0; core<=$(( $cores_per_socket - 1 )); core++ ))
do
        exec_cmd "TF_ENABLE_ONEDNN_OPTS=0 numactl -C $core -m 0 python3
infer_test.py -i $dataset -m malconv.int8.onnx -n 1000 >
logs/malconv.int8.onnx_core${core}.log" &
done

# wait for all the commands to finish
while [ $(ps -ef | grep "python3 infer_test.py" | wc -l) != 1 ];
do
    sleep 1
done

cd logs
cat *h5*no_dnn*.log | grep "average inference time" | awk -v N=$4 '{print $4}'
>> h5_no_dnn_results.log
cat *h5_core*.log | grep "average inference time" | awk -v N=$4 '{print $4}'
>> h5_results.log
cat *fp32*no_dnn*.log | grep "average inference time" | awk -v N=$4 '{print
$4}' >> fp32_no_dnn_results.log
cat *fp32*pb_core*.log | grep "average inference time" | awk -v N=$4 '{print
$4}' >> fp32_results.log
cat *bf16*pb_amx*.log | grep "average inference time" | awk -v N=$4 '{print
$4}' >> bf16_amx_results.log
cat *bf16*pb_bf16*.log | grep "average inference time" | awk -v N=$4 '{print
$4}' >> bf16_results.log
cat *int8*pb_amx*.log | grep "average inference time" | awk -v N=$4 '{print
$4}' >> int8_amx_results.log
cat *int8*vnni*.log | grep "average inference time" | awk -v N=$4 '{print $4}'
>> int8_results.log
cat *int8*onnx*.log | grep "average inference time" | awk -v N=$4 '{print $4}'
>> onnx_int8_results.log

cat *h5*no_dnn*.log | grep "ROC AUC" | awk -v N=$4 '{print $4}' >>
h5_no_dnn_accuracy.log
cat *h5_core*.log | grep "ROC AUC" | awk -v N=$4 '{print $4}' >>
h5_accuracy.log
cat *fp32*no_dnn*.log | grep "ROC AUC" | awk -v N=$4 '{print $4}' >>
fp32_no_dnn_accuracy.log
cat *fp32*pb_core*.log | grep "ROC AUC" | awk -v N=$4 '{print $4}' >>
fp32_accuracy.log
cat *bf16*pb amx*.log | grep "ROC AUC" | awk -v N=$4 '{print $4}' >>
bf16_amx_accuracy.log
cat *bf16*pb_bf16*.log | grep "ROC AUC" | awk -v N=$4 '{print $4}' >>
bf16_accuracy.log
cat *int8*pb_amx*.log | grep "ROC AUC" | awk -v N=$4 '{print $4}' >>
int8_amx_accuracy.log
cat *int8*vnni*.log | grep "ROC AUC" | awk -v N=$4 '{print $4}' >>
int8_accuracy.log
cat *int8*onnx*.log | grep "ROC AUC" | awk -v N=$4 '{print $4}' >>
onnx_int8_accuracy.log

rm *core*.log
```

16. Create the following file called "`install_malconv_ai.sh`":

```
#!/bin/bash

# Setup base environment

cd /home
mkdir logs
```

```
dnf update -y
dnf remove -y python36
dnf install -y git python39-pip curl wget numactl python39 python39-devel gcc
dnf --enablerepo=powertools install -y opencv # needed by intel neural
compressor
pip3 install --upgrade pip


# TensorFlow setup

# Install the TensorFlow pip package dependencies
pip3 install -U --user pip numpy wheel packaging requests opt_einsum
pip3 install -U --user keras_preprocessing --no-deps

# Install bazel build dependencies
yum install -y gcc-c++ unzip zip
yum install -y java-11-openjdk
yum install -y perl perl-Data-Dumper cmake

# Clone Intel-TensorFlow repo
git clone https://github.com/tensorflow/tensorflow.git
cd tensorflow
git checkout v2.10.0
cd ..

# Install bazel. To know which bazel version is required to build the
TensorFlow
# please refer to the .bazelversion file in the tensorflow folder
# For this version, bazel 5.1.1 is required.
wget https://github.com/bazelbuild/bazel/releases/download/5.1.1/bazel-5.1.1-
installer-linux-x86_64.sh
chmod +x bazel-5.1.1-installer-linux-x86_64.sh
./bazel-5.1.1-installer-linux-x86_64.sh

# Clone oneDNN repo and checkout to rls-v2.5. The latest 2.5.4 version will be
here.
git clone https://github.com/oneapi-src/oneDNN.git
cd oneDNN
git checkout rls-v2.5
cd ..

read -p “Please modify the /home/tensorflow/tensorflow/workspace2.bzl with the
changes documented in the comment of this script before continuing”
# Change the oneDNN version in the file
/home/tensorflow/tensorflow/workspace2.bzl
# Search for "mkl_dnn_v1" and replace the whole block "tf_http_archive{...}"
with the following codes.
# The local path of oneDNN archive needs to be specified.
#native.new_local_repository{
#name = "mkl_dnn_v1",
#build_file = "//third_party/mkl_dnn:mkldnn_v1.BUILD",
#path = "/home/oneDNN"
#}

# Build the whl file that can be pip3 installed. This step might take a while
to compile.
cd /home/tensorflow

# Configure using the default options.
./configure
# bazel build
bazel build --config=mkl //tensorflow/tools/pip_package:build_pip_package

# Generate the whl file to your specified location
./bazel-bin/tensorflow/tools/pip_package/build_pip_package
/home/tensorflow_pkg
```

```
# Installation.
pip3 install /home/tensorflow_pkg/tensorflow*.whl

# Install other dependencies
pip3 install wheel neural-compressor==1.10.1 progress onnx tf2onnx onnxruntime
sklearn setuptools

# Need to downgrade numpy and protobuf
pip3 install numpy==1.23.1 protobuf==3.20.*

cd /home
mv IntelOptMalconv.h5 malconv.h5

# Setup the models

python3 h5_to_saved.py -m malconv.h5 -o malconv_saved_model
python3 freeze_graph.py malconv_saved_model malconv.fp32.pb

python3 quantize.py -m malconv.fp32.pb -c malconvBF16.yaml -i /home/datasets -
o malconv.bf16.pb
python3 quantize.py -m malconv.fp32.pb -c malconvINT8.yaml -i /home/datasets -
o malconv.int8.pb

# Convert FP32 to ONNX INT8
python3 -m tf2onnx.convert --opset 13 --input malconv.fp32.pb --inputs
input_1:0 --outputs Identity:0 --output malconv.fp32.onnx
python3 onnx_quantize.py -i /home/datasets -m malconv.fp32.onnx -c onnx.yaml -
o malconv.int8.onnx
```

17. Create the following file called "`run_malconv_ai.sh`":

```
#!/bin/bash

kubectl exec malconv-ai -- /home/test_malconv_ai.sh

kubectl cp malconv-ai:/home/logs logs/
echo "Security AI Mean Inference Times" > malconv_ai.log
awk '{ sum += $1 } END { print "Keras h5 without oneDNN:",sum/NR,"ms"
}' logs/h5_no_dnn_results.log >> malconv_ai.log
awk '{ sum += $1 } END { print "Keras h5 with oneDNN:",sum/NR,"ms" }'
logs/h5_results.log >> malconv_ai.log
awk '{ sum += $1 } END { print "FP32 without oneDNN:",sum/NR,"ms" }'
logs/fp32_no_dnn_results.log >> malconv_ai.log
awk '{ sum += $1 } END { print "FP32 with oneDNN:",sum/NR,"ms" }'
logs/fp32_results.log >> malconv_ai.log
awk '{ sum += $1 } END { print "BF16 with AMX:",sum/NR,"ms" }'
logs/bf16_amx_results.log >> malconv_ai.log
awk '{ sum += $1 } END { print "BF16 with BF16:",sum/NR,"ms" }'
logs/bf16_results.log >> malconv_ai.log
awk '{ sum += $1 } END { print "INT8 with AMX:",sum/NR,"ms" }'
logs/int8_amx_results.log >> malconv_ai.log
awk '{ sum += $1 } END { print "INT8 with VNNI:",sum/NR,"ms" }'
logs/int8_results.log >> malconv_ai.log
awk '{ sum += $1 } END { print "ONNX INT8:",sum/NR,"ms" }'
logs/onnx_int8_results.log >> malconv_ai.log
```

18. You will need to provide your own testing dataset to use. Create the following directories:

```
mkdir -p datasets/KNOWN
mkdir datasets/MALICIOUS
```

19. Place the benign files into the "`datasets/KNOWN`" directory, and place the malicious files in the "`datasets/MALICIOUS`" directory
20. Run the "`setup_malconv_ai.sh`" script and then the "`run_malconv_ai.sh`" script. The generated "`malconv_ai.log`" file will contain the mean inference time results of the four models tested.

    Note: The testing was completed using an Intel-optimized version of Tensorflow that is not currently publicly available. These scripts use the default Tensorflow version which may result in different performance results.

§

5.6 Data Streaming Accelerator (DSA)

The Data Streaming Accelerator (DSA) is used for improving the performance of storage, networking, and various other I/O applications. The DMA engine is optimized for moving data between memory. The goal of Intel® DSA is to provide higher overall system performance for data mover and transformation operations, while freeing up CPU cycles for higher level functions. Please refer to document ID 353216 for the Data Streaming Accelerator User Guide for configuration information and the idxd-config GitHub page (https://github.com/intel/idxd-config) for the accel-config library and a dsa-test file for verifying the configuration.

6 Summary

The Intel NFVI v4 and SASE Verified Reference Configuration, is an Intel Accelerated Solution defined on 4th Gen Intel® Xeon® Scalable processors. This solution, combined with architectural improvements, feature enhancements, integrated Accelerators with high memory and IO bandwidth, provides a significant performance and scalability advantage in today's NFVI environments. These processors are optimized for network, cloud native, wireline, and wireless core-intensive workloads, and are especially suited for NFVI and SASE workload coupled using Intel® Ethernet E810-Network Controllers and Data Plane Development Kit.