Intel® Verified Reference Configuration for 5G FlexCore 2.0 UPF with Red Hat OpenShift Container Platform on Intel® Xeon® Scalable Processors using Intel® Infrastructure Power Manager

Authors

Sarita Maini

Siva Prasad Tirulaka

Gordon Noonan

Kashish Singh

Key Contributors

Eoin Walsh

Damien Power

Chetan Hiremath

John M Morgan

Eric Heaton

Brian Neville

Timothy Miskell

Steve Dinkins

Rory Sexton

1 Introduction

FlexCore 2.0 UPF is a 5G reference design that helps develop the User Plane Function (UPF) solution on servers powered by Intel® Xeon® Scalable processors. This 5G reference design enables the virtualization of the User Plane Function (UPF) and runs on a Network Function Virtualization Infrastructure (NFVI). It facilitates evaluation and benchmarking of a virtualized 5G core mobile network data plane.

Intel® Xeon® Scalable processors are uniquely designed for network telco and cloud workloads and include built-in accelerators to increase throughput and additional features that deliver fine-grained control over frequency and power consumption.

Intel® Infrastructure Power Manager for 5G Core (IPM) reference software further reduces run-time power consumption by dynamically matching CPU performance and energy consumption to real-time traffic levels, delivering power savings while maintaining key network performance metrics, such as throughput and packet loss.

FlexCore 2.0 UPF optimizes the performance of 5G core, using a data plane development kit (DPDK), Vector Packet Processing (VPP), Non-Uniform Memory Access (NUMA) optimizations, huge pages. etc. It leverages the Dynamic Device Personalization (DDP) capability of Intel Network Interface Cards (NICs) to further improve the performance of Intel hardware and software by offloading packet forwarding, packet processing and load balancing to Intel E810-2CQDA2 Ethernet Network Adapters, while minimizing the use of processor cores for these functions.

This document is an Intel Verified Reference Configuration (VRC) detailing the steps for deploying FlexCore 2.0 UPF on Intel® Xeon® Scalable Processors in a Red Hat OpenShift Container Platform (OCP) environment as well as IPM and showcasing the impressive throughput and power savings in this CPU-intensive and I/O intensive 5G core workload. The files and commands will need to be customized depending on the processors and PCIe network adapters used in the Device-Under-Test (DUT). This VRC has been tested on Intel 4th Gen Xeon Platinum 8470N processors with Intel Ethernet 800 Series Network Adapters.

Intel VRCs provide a workload-optimized configuration of select hardware and various Intel® Xeon® processor technologies along with optimized software and BIOS settings. It leverages the hardened hardware, firmware, and software which allows customers to integrate on top of this known platform configuration.

Table of Contents

Figures

Tables

1.1 Terminology

Table 1. Terminology

Abbreviation	Description
BIOS	Basic input and output service
DDP	Device Dynamic Personalization
DIMM	Dual Inline Memory Module
DPDK	Data Plane Development Kit
DRAM	Dynamic Random Access Memory
DUT	Device Under Test
GbE	Gigabit Ethernet
IEO	Intel Ethernet Operator
IOMMU	I/O Memory Management Unit. A DMA Remapping Hardware Unit as defined by Intel® Virtualization Technology for Directed I/O
IPM	Intel® Infrastructure Power Manager
NIC	Network Interface Card/Controller
NUMA	Non-Uniform Memory Access
PCIe	Peripheral Component Interconnect express
RH OCP	Red Hat OpenShift Container Platform
SR-IOV	Single Root Input/Output Virtualization
SSD	Solid State Drive
TRex	An open source, low cost, stateful and stateless traffic generator fueled by DPDK.
MTU	Maximum transmission unit
UPF	User Plane Function
VPP	FD.io's Vector Packet Processor (VPP)
VRC	Verified Reference Configuration
Intel® VT	Intel® Virtualization Technology

1.2 Reference Documentation

Table 2. Reference Documents

Reference	Source
Intel® Ethernet Controller E810 Dynamic Device Personalization (DDP) Technology Guide	617015
Intel® Infrastructure Power Manager v23.12	https://cdrdv2.intel.com/v1/dl/getContent/812070
Intel® Infrastructure Power Manager User Guide	https://cdrdv2.intel.com/v1/dl/getContent/812071
Red Hat* OpenShift Container Platform 4.13	https://access.redhat.com/documentation/en-us/openshift_container_platform/4.13
TRex Manual	https://trex-tgn.cisco.com/trex/doc/trex_manual.html
Sustainability on Cloud Platforms	https://www.youtube.com/watch?v=4Pyo6aFdKus
BIOS Settings for Intel® Wireline, Cable, Wireless and Converged Access Platform Reference Guide	https://www.intel.com/content/www/us/en/secure/design/internal/content-details.html?DocID=747130
Samsung Achieves 305 Gbps on 5G UPF Core Utilizing Intel® Architecture	https://networkbuilders.intel.com/solutionslibrary/samsung-achieves-305-gbps-on-5g-upf-core-utilizing-intel-architecture
Implementation of ZTE's High-Performance 5G Core Network UPF	https://networkbuilders.intel.com/solutionslibrary/implementation-of-ztes-high-performance-5g-core-network-upf
Building the 5G Wireless Core	https://cdrdv2-public.intel.com/755367/q2-nt-building-the-wireless-5g-core-white-paper.pdf

2 Overview

The 5G FlexCore 2.0 5G UPF workload-optimized solution is designed to minimize the challenges of infrastructure deployment and optimization for the best performance with balanced I/O across sockets for core-bound as well as I/O-bound workloads. It runs on Intel® Xeon® Scalable processors, which incorporate unique features designed especially for virtualized network workloads. This solution can be used to compare the performance of various generations of Intel Xeon processors.

This Intel Verified Reference Configuration document includes the following:

- Step-by-step instructions for deploying and benchmarking FlexCore 2.0 5G UPF with Red Hat OpenShift Container Platform (RH OCP).
- Deployment steps for Intel® Infrastructure Power Manager (IPM) pod on OCP worker node running FC 2.0 UPF.
- Impressive performance of FlexCore 2.0 5G UPF with Red Hat OpenShift Container Platform (RH OCP) with an Intel®4th Gen Xeon® Scalable Processor.
- Intel Infrastructure Power Manager (IPM) benefits which will help telco and cloud workloads to achieve power optimizations and energy savings goals with IPM, with best-in-class throughput and performance per watt.
- Up to **948 Gbps** (94.8% of the line rate of 1000 Gbps) can be achieved with an acceptable error rate of 0% packet loss with standalone 5G FlexCore 2.0 UPF running in a RH OCP worker node 2-socket server, powered by Intel®4th Gen Xeon® Scalable Processor 8470N, using a Packet Size of 650 Bytes and maintaining a packet loss of 0%, with and without IPM.
- **Up to a maximum of 36% power savings and average 23% power savings using IPM**, depending on the percentage of traffic load, with no impact on I/O throughput. A real-life telco scenario of a 24-hour traffic profile was used for measuring the power consumption.

Note:

Please contact your Intel representative for access to Intel's FlexCore 2.0 UPF and Intel Infrastructure Power Manager packages for deployment.

Note:

This document assumes that a Red Hat OpenShift Platform Configuration (RH OCP) cluster has already been created and the device-under-test is a worker node in that cluster.

3 Solution Components

This solution consists of select hardware and various Intel® Xeon® processor technologies along with optimized software and firmware configurations.

3.1 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 to intelligently 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 code-named 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

3.2 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

3.2.1 Intel® Network Adapters with Data Plane Development Kit (DPDK)

Intel® Network Products deliver continuous innovation 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 NFVI Forwarding Platform.

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

Dynamic Device Personalization (DDP) usage reconfigures network controllers for different network functions on-demand, without the need for migrating all VMs from the server, and 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.

DDP describes the capability of 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.

3.3 Intel® Xeon® Scalable Platform Technologies

3.3.1 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.

3.3.2 Intel® Turbo Boost Technology

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.

3.3.3 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 VRCs for Network Function Virtualization Infrastructure (NFVI) are required to have all virtualization technology features enabled.

3.4 FlexCore 2.0 User Plane Function (UPF)

FlexCore is a 5G reference design which includes the User Plane Function (UPF) and a simulated Session Management Function (SMF). The UPF is the network function which is part of the data (user) plane. Intel's FlexCore facilitates high-performance packet processing, load distribution and reduced CPU utilization, increased network bandwidth and low latency, jitter and packet loss.

3.5 Intel® Infrastructure Power Manager (IPM)

The Intel Infrastructure Power Manager for 5G Core improves the power efficiency of network functions running in software on Intel Xeon Scalable Processor-based servers by automatically making intelligent use of power management features in the hardware. IPM has a server-wide view but primarily decides on power saving approaches from the application level.

3.6 Red Hat* OpenShift Container Platform (RH OCP)

Red Hat® OpenShift® Container Platform is a foundation for building and scaling containerized applications. It is a cloud-based Kubernetes platform and comes with an automatic install to get Kubernetes up and running as quickly as possible. The OS which gets installed on the nodes when the cluster is created is an immutable OS named "Red Hat Enterprise Linux CoreOS" (RHCOS). A cluster has a web console which includes an operator hub. Operators can be used to enable automated installation and configuration of applications. As RHCOS is immutable, all kernel arguments, system settings, applications, drivers, tools, compilers etc. cannot be changed or installed as they can with a Red Hat Enterprise Linux bare-metal operating system. They all require machine config pools, machine configs, image builds uploaded to and downloaded from a local registry or a public registry, containers, pods, operators etc. This is very different from a Kubernetes cluster which runs on bare-metal Linux and is not a tightly controlled operating system like RHCOS.

4 Design Compliance Requirements

The checklists in this chapter provide guidance for assessing the conformance to the Intel® Verified Reference Configuration.

4.1 Intel VRC Hardware Requirements

For the platform to conform to the desired Intel® Verified Reference Configuration for 5G FlexCore 2.0 UPF, the following hardware requirements must be met.

Table 3. Hardware Configuration

Hardware Configuration			
COMPONENT	**REQUIREMENT**	**REQUIRED/RECOMMENDED**	**QUANTITY PER SERVER**
Processor	Intel® Xeon® Platinum 8470N	Recommended	2
Sockets	2-Socket Server	Required	2
Memory	DRAM only configuration: 512 GB (16 x 32 GB DDR5, 4800 MHz)	Required	16
Network	Intel® Ethernet Network Adapter E810-2CQDA2	Required	4
	Intel® Ethernet Network Adapter E810-CQDA2	Required	2
Storage (Boot Drive)	SSD – 480 GB or higher	Required	1
LAN on Motherboard (LOM)	1 or 10 Gbps port for Management NIC	Required	1

4.2 Intel VRC Software Requirements

For the platform to conform to the desired Intel® Verified Reference Configuration for 5G FlexCore 2.0 UPF, the following software requirements must be met.

Table 4. Software Configuration

Software Configuration		
COMPONENT	REQUIREMENT	REQUIRED/RECOMMENDED
Red Hat OpenShift Container Platform	4.13	Required
Red Hat Enterprise Linux CoreOS release	4.13	Required
Red Hat Enterprise Linux CoreOS Kernel	5.14.0-284.41.1.el9_2.x86_64	Required
Red Hat Enterprise Linux CoreOS release 4.13	4.13	Required
Intel's FlexCore 2.0 UPF	23.07	Required
Intel Infrastructure Power Manager	23.12	Required
Node Feature Discovery Operator	4.13.0	Required
SR-IOV Network Operator	4.13.0	Required
Intel Ethernet Operator	1.0.0	Required
Intel ICE DDP COMMS Package	1.3.45.0	Required
Trex Traffic Generator	3.0	Required

4.3 BIOS Settings

To meet the performance requirements for an Intel VRC for 5G FlexCore 2.0 UPF 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.

5 System Setup

The scope of this document does not include deploying a Red Hat OpenShift Platform Configuration (RH OCP) cluster. It assumes that a cluster has already been created and the device-under-test is a worker node in that cluster.

Figure 1 shows the data network setup for the device-under-test (DUT) which is a worker node server in a multi-node RH OCP cluster. This does not show the master nodes, other worker nodes, the provisioning/PXE network or the management network. Please see Red Hat documentation for details. It shows TRex Traffic Generator servers which are connected back-to-back with the DUT, using 100 GbE Direct Access Cables.

The DUT and TRex servers need to be configured with the required hardware components, software components and BIOS settings defined in Section 4 as shown in this figure.

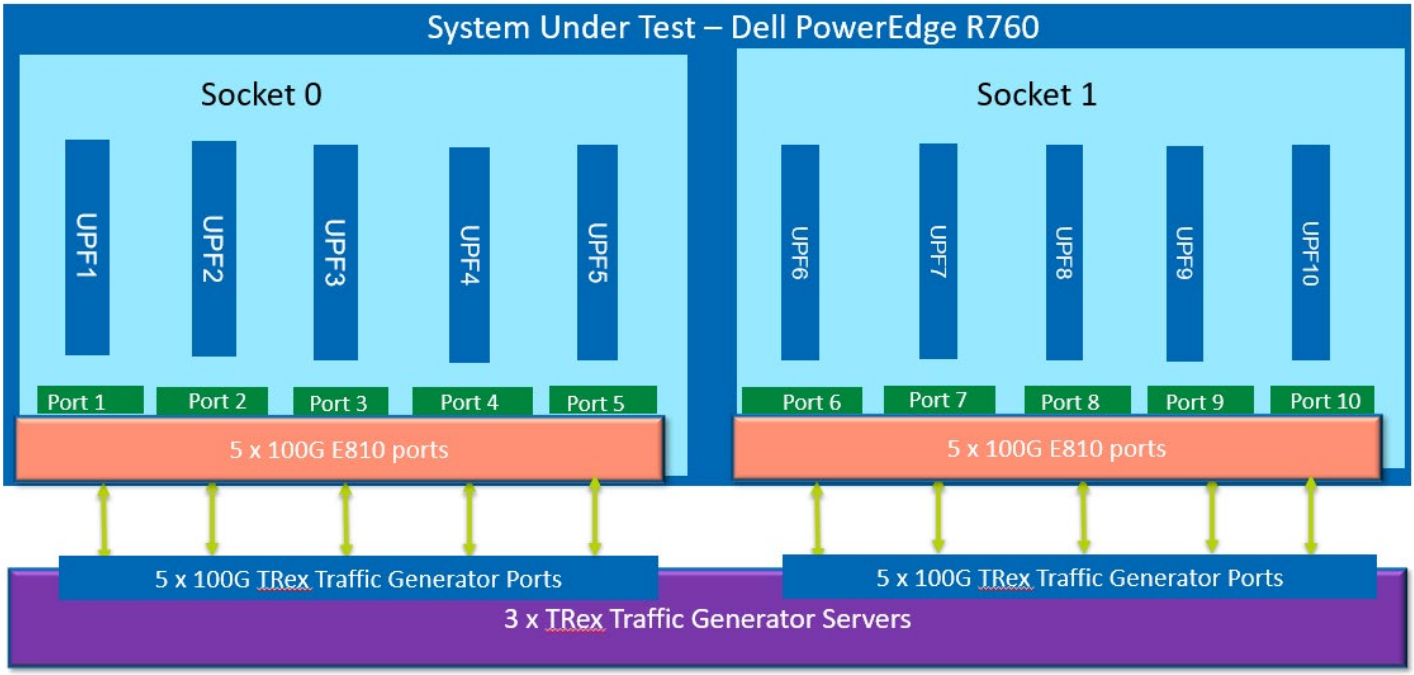

Figure 1. Setup of DUT and Traffic Generator

6 5G FlexCore 2.0 UPF Installation Steps

This section details the step-by-step instructions for deploying the FlexCore 2.0 UPF release v23.07 which is based on VPP 20.09 and DPDK 23.03.

1. Download the release package file fc20-upf-v23.07_vpp2009.dpdk23.03.tgz (See your Intel representative for the link and access permission to the release).

2. Load the base image. This step only needs to be taken once and is persistent across reboots.

```
# podman load -i fc20-upf-v23.07_vpp2009.dpdk23.03.tgz
Getting image source signatures
....
....
....
Copying config 2650e13ffc done
Writing manifest to image destination
Storing signatures
Loaded image: localhost/fc2.0-upf:v23.07_vpp2009.dpdk23.03
```

3. Verify that the image got loaded.

```
# podman image ls
REPOSITORY              TAG                       IMAGE ID      CREATED        SIZE
localhost/fc2.0-upf     v23.07_vpp2009.dpdk23.03  2650e13ffc14  4 months ago   1.64 GB
```

4. Configure a local registry.

```
# oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch
'{"spec":{"managementState":"Managed"}}'
config.imageregistry.operator.openshift.io/cluster patched

# oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch
'{"spec":{"storage":{"emptyDir":{}}}}'
config.imageregistry.operator.openshift.io/cluster patched
```

```
# oc registry info
error: the integrated registry has not been configured

# sleep 120

# oc registry info
image-registry.openshift-image-registry.svc:5000

# HOST=$(oc get route default-route -n openshift-image-registry --template='{{ .spec.host }}')
Error from server (NotFound): routes.route.openshift.io "default-route" not found

# oc patch configs.imageregistry.operator.openshift.io/cluster --type merge -p
'{"spec":{"defaultRoute":true}}'
config.imageregistry.operator.openshift.io/cluster patched

$ HOST=$(oc get route default-route -n openshift-image-registry --template='{{ .spec.host }}')

$ echo $HOST
```

5. Upload the image to the local registry.

```
# oc new-project upf

# oc whoami
kube:admin

# oc policy add-role-to-user registry-viewer $(oc whoami)
# oc policy add-role-to-user registry-editor $(oc whoami)

# oc get secret -n openshift-ingress router-certs-default -o go-template='{{index .data "tls.crt"}}' | base64
-d | sudo tee /etc/pki/ca-trust/source/anchors/${HOST}.crt  > /dev/null

# update-ca-trust enable


# podman login -u kubeadmin -p $(oc whoami -t) $HOST
Login Succeeded!

# podman tag fc2.0-upf:v23.07_vpp2009.dpdk23.03 $HOST/upf/fc2.0-upf:23.07

# docker images | grep "fc2.0-upf"

# podman push $HOST/upf/fc2.0-upf:23.07
Getting image source signatures
Copying blob 8c207c53307f done
Copying blob f334eb4e2ec0 done
Copying blob 0f87d8440b64 done
Copying blob 4d6560c57d85 done
Copying blob 277625e7b9b8 done
Copying blob 20707e4d41cf done
Copying blob e4b2617bacc8 done
Copying blob 9c7cad35fb71 done
Copying blob bfb2752fa526 done
Copying blob 31984ce103dd done
Copying blob 72bf09f9fae7 done
Copying blob 2535b010dfef done
Copying blob 0b096fa9c93e done
Copying blob 3c36edbeafee done
Copying blob 9123f7551ee9 done
Copying blob 0b8cf0aa86f7 done
Copying blob 05381cc395c6 done
Copying blob 9d3ad5bb9800 done
Copying blob 6d85d2bb13a3 done
Copying blob 706f0c289599 done
Copying blob 17fecce9724b done
Copying blob 0499503d879e done
Copying blob 305bfbdf947e done
Copying config 2650e13ffc done
Writing manifest to image destination
Storing signatures
#
# oc get images |grep upf
```

```
sha256:88d33e9ea5751d7c12d8fb6310f7b50fff2ecbfe1385c97262e80fbaced29fc1   image-registry.openshift-image-
registry.svc:5000/upf/fc2.0-upf@sha256:88d33e9ea5751d7c12d8fb6310f7b50fff2ecbfe1385c97262e80fbaced29fc1
```

6. Label the worker node, create a machine config pool.

```
# oc label node worker-5 node-role.kubernetes.io/upf=

# cat upf-mcp.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: upf
  labels:
    cnf-kubelet: '5g-upf'
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,upf]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/upf: ""

# oc create -f upf-mcp.yaml
```

7. Create a performance profile. This will reboot the worker node. Verify that the kernel arguments have been changed in the worker node after the reboot is completed and the node is Ready.

```
# cat performance-profile.yaml
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance-profile
spec:
  additionalKernelArgs:
    - nohz_full=1-51,105-155,53-103,157-207
    - idle=poll
    - rcu_nocb_poll
    - intel_iommu=on
    - iommu=pt
    - nmi_watchdog=0
    - tsc=perfect
    - selinux=0
    - enforcing=0
    - noswap
    - clock=pit
    - audit=0
    - mce=off
    - processor.max_cstate=1
    - intel_idle.max_cstate=0
    - rcutree.kthread_prio=11
    - rcupdate.rcu_normal_after_boot=0
    - softlockup_panic=0
    - nmi_watchdog=0
    - console=ttyS0,115200n8
    - pcie_aspm=off
    - pci=noaer
    - firmware_class.path=/var/lib/firmware
  cpu:
    isolated: '1-51,105-155,53-103,157-207'
    reserved: '0,104,52,156'
  globallyDisableIrqLoadBalancing: true
  hugepages:
    defaultHugepagesSize: 1G
    pages:
      - count: 600
        size: 1G
  nodeSelector:
    node-role.kubernetes.io/upf: ''
```

```
# oc apply -f performance-profile.yaml
```

8. Search the Operator Hub on the cluster web console and install the following operators:
   - Node Feature Discovery Operator
   - Intel Ethernet Operator
   - SR-IOV Network Operator for OpenShift

9. Install the latest ICE COMMS DDP package using the latest release, for each PCIe 100GbE port. This step downloads the ICE COMMS DDP package and updates the DDP profile of the device selected. If nodeSelectors and deviceSelector are left empty, the EthernetClusterConfig will target all compatible NICs on all available nodes.

```
# cat ddp-dev-0a.yaml
apiVersion: ethernet.intel.com/v1
kind: EthernetClusterConfig
metadata:
  name: comms-ddp-0a
  namespace: openshift-sriov-network-operator
spec:
  nodeSelectors:
    kubernetes.io/hostname: <worker node name>
  deviceSelector:
    pciAddress: 0000:0a:00.0
  deviceConfig:
    ddpURL: "https://downloadmirror.intel.com/785846/738693_ice_comms-1.3.45.0.zip"
    ddpChecksum: "322D187C08A8903A89DC4D885A9B4B05B8974C6F"

# oc apply -f ddp-dev-0a.yaml

# oc get EthernetClusterConfig -A
```

10. Unload and reload the irdma and ice drivers on the worker node. Verify that the COMMS DDP package is installed on each PCIe port which will be used for I/O.

```
# oc debug node/<worker node name>
Temporary namespace openshift-debug-5p6dn is created for debugging node...
Starting pod/worker-5-debug ...
To use host binaries, run `chroot /host`
Pod IP: 10.10.10.186
If you don't see a command prompt, try pressing enter.
sh-4.4# chroot /host
sh-4.4# modprobe -r irdma; modprobe -r ice; modprobe ice; modprobe irdma
sh-5.1# devlink dev info
```

11. Workaround for IEO bug: After all ports have the ICE COMMS Package loaded, delete the Intel Ethernet Operator from the web console. If it is not deleted, all UPF and SMF pods will crash in a few hours. This IEO issue will be fixed in future releases.

12. Create SR-IOV network policies for each 100 GbE port and verify that they have been created.

```
# cat enp10s0-sriov-network-node-policy.yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: enp10s0-sriov-policy
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: openshift-sriov-network-operator
  nodeName: worker-5
  nicSelector:
    pfNames:
```

```
      - 'enp10s0#0-2'
    vendor: '8086'
  deviceType: vfio-pci
  spoofChk: 'off'
  trust: 'on'
  nodeSelector:
    node-role.kubernetes.io/upf: ''
  numVfs: 3
  priority: 99
  resourceName: 'enp10s0_rn'

# oc apply -f enp10s0-sriov-network.yaml

# oc get sriovnetworknodepolicies -A
```

13. Create UPFs and SMF pods. Examples of the SMF pod yaml file and one UPF yaml file are shown below. Each UPF pod is allocated 16 worker cores and 1 main core. These are automatically NUMA aligned with the NIC port being used by that UPF pod and all allocated physical cores as well as their sibling cores are NUMA-optimized.

```
# cat smf.yaml
apiVersion: v1
kind: Pod
metadata:
  name: smf
  namespace: openshift-sriov-network-operator
  # annotations:
  #   k8s.v1.cni.cncf.io/networks: enp10s0-sriov-network
spec:
  nodeSelector:
      node-role.kubernetes.io/upf: ''
  containers:
    - name: smf
      image: image-registry.openshift-image-registry.svc:5000/upf/fc2.0-upf:23.07
      securityContext:
        allowPrivilegeEscalation: true
        privileged: true
      command: [ "/bin/sh" , "-c", "tail -f /dev/null" ]
      env:
      - name: MY_POD_IP
        valueFrom:
          fieldRef:
            fieldPath: status.podIP
      - name: MY_NODE_NAME
        valueFrom:
          fieldRef:
            fieldPath: spec.nodeName
      resources:
        requests:
          cpu: 10
          hugepages-1Gi: 30Gi
          memory: 30Gi
        limits:
          cpu: 10
          hugepages-1Gi: 30Gi
          memory: 30Gi
    # mbufs: "65536"
    # heapSize: ""
      volumeMounts:
      - name: hugepages
        mountPath: /dev/hugepages
      - name: dshm
        mountPath: /dev/shm
      - name: dpdk
        mountPath: /var/run/dpdk
      - name: sys
        mountPath: /sys
  volumes:
  - name: hugepages
```

```
    emptyDir:
      medium: HugePages
  - name: dshm
    emptyDir:
      medium: Memory
  - name: dpdk
    hostPath:
      path: /var/run/dpdk
  - name: sys
    hostPath:
      path: /sys

# oc create -f smf.yaml

# cat upf1-enp10s0.yaml
apiVersion: v1
kind: Pod
metadata:
  name: upf1
  namespace: openshift-sriov-network-operator
  #  annotations:
  #  k8s.v1.cni.cncf.io/networks: enp10s0-sriov-network
spec:
  nodeSelector:
      node-role.kubernetes.io/upf: ''
  containers:
    - name: upf1
      image: image-registry.openshift-image-registry.svc:5000/upf/fc2.0-upf:23.07
      securityContext:
        allowPrivilegeEscalation: true
        privileged: true
      command: [ "/bin/sh" , "-c", "tail -f /dev/null" ]
      env:
      - name: MY_POD_IP
        valueFrom:
          fieldRef:
            fieldPath: status.podIP
      - name: MY_NODE_NAME
        valueFrom:
          fieldRef:
            fieldPath: spec.nodeName
      resources:
        requests:
          cpu: 18
          openshift.io/enp10s0_rn: '3'
          hugepages-1Gi: 40Gi
          memory: 30Gi
        limits:
          cpu: 18
          openshift.io/enp10s0_rn: '3'
          hugepages-1Gi: 40Gi
          memory: 30Gi
      volumeMounts:
      - name: hugepages
        mountPath: /dev/hugepages
      - name: dshm
        mountPath: /dev/shm
      - name: dpdk
        mountPath: /var/run/dpdk
      - name: sys
        mountPath: /sys
      - name: vpp
        mountPath: /run/vpp1
  volumes:
  - name: hugepages
    emptyDir:
      medium: HugePages
  - name: dshm
    emptyDir:
      medium: Memory
  - name: dpdk
    hostPath:
      path: /var/run/dpdk
  - name: sys
```

```
    hostPath:
      path: /sys
  - name: vpp
    hostPath:
      path: /run/vpp1

# oc create -f upf1-enp10s0.yaml
```

14. UPF configuration

```
# oc exec -it upf1 -- /bin/bash
# source /opt/intel/scripts/lib.sh 1
# ln -s /opt/VCM/config/intel-upf1-sysconf.16.1024rx.json /opt/VCM/config/intel-upf-sysconf.json
# ln -s /opt/intel/scripts/startup1.v23.07.spr.sh /opt/intel/scripts/startup
# ln -s /etc/vpp/startup.upf1.16.1024rx.conf /etc/vpp/startup.conf
```

Edit /etc/vpp/startup.conf

- Change dev field to match the Bus:Device:Function number of the 3 VFs of the PCIe NIC port being used for this UPF
- Change main core and worker cores to the cores allocated to this pod
- To find the cores allocated to this pod, use the command:
  `cat /proc/self/status |grep -i cpus`

Edit /opt/intel/scripts/startup

- Change the following lines to match the Bus:Device:Function number of the 3 VFs of the PCIe NIC port being used for this UPF.

```
#VF PCI Addresses - fpeth0 / fpeth1 / fpeth2
export PCIDEVICE_INTEL_COM_SRIOV_DPDK_N3=0000:16:01.0
export PCIDEVICE_INTEL_COM_SRIOV_DPDK_N4U=0000:16:01.1
export PCIDEVICE_INTEL_COM_SRIOV_DPDK_N6=0000:16:01.2
```

- Change the following line to match the MAC address of the corresponding Trex port virtual function.

```
export TREX_MAC_PORT_0=3c:fd:fe:a9:e6:04
```

Edit /opt/VCM/config/intel-upf-sysconf.json

- Change the main core and worker cores to the cores allocated to this pod.

Run the script:

```
# /opt/intel/scripts/startup
```

Verify that all the required processes are running and the N3/N6 links are configured with IP addresses and MAC addresses.

Examples:

```
# ps -aef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 02:46 ?        00:00:00 tail -f /dev/null
root          99       0  0 02:51 pts/0    00:00:00 /bin/bash
root         153       1  0 02:55 pts/0    00:00:00 /usr/local/mnvpp-pkg/bin/upAgent
root         154       1  0 02:55 pts/0    00:00:00 tee /tmp/upagent.log
root         159       1  0 02:55 pts/0    00:00:00 /opt/VCM/bin/vcmUpfEif
root         218       1 17 02:55 pts/0    00:00:21 /opt/VCM/bin/vcmDpe
root         260     153 99 02:55 pts/0    00:31:08 /usr/local/mnvpp-pkg/bin/vpp -c /etc/vpp/startup.conf
root         344      99  0 02:57 pts/0    00:00:00 ps -aef
```

The interfaces shown are feth0 (5G core N3 interface), feth2 (5G core N6 interface). These interfaces are used to transmit and receive packets from the traffic generator.

```
[root@upf1 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0@if133: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc noqueue state UP group default
    link/ether 0a:58:0a:83:00:50 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.131.0.80/23 brd 10.131.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::858:aff:fe83:50/64 scope link
       valid_lft forever preferred_lft forever
3: fpeth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 1000
    link/ether 4e:4c:5d:cb:13:be brd ff:ff:ff:ff:ff:ff
    inet 192.182.120.171/24 brd 192.182.120.255 scope global fpeth0
       valid_lft forever preferred_lft forever
    inet6 fe80::18df:e8ff:fe98:32b2/64 scope link
       valid_lft forever preferred_lft forever
4: fpeth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 1000
    link/ether aa:c5:87:4e:49:22 brd ff:ff:ff:ff:ff:ff
    inet 192.182.121.181/24 brd 192.182.121.255 scope global fpeth1
       valid_lft forever preferred_lft forever
    inet6 2405:192:150:133::144/64 scope global nodad
       valid_lft forever preferred_lft forever
    inet6 fe80::a8c5:87ff:fe4e:4922/64 scope link
       valid_lft forever preferred_lft forever
5: fpeth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 1000
    link/ether 52:d4:b3:e5:19:38 brd ff:ff:ff:ff:ff:ff
    inet 192.182.96.191/24 brd 192.182.96.255 scope global fpeth2
       valid_lft forever preferred_lft forever
    inet6 fe80::187d:94ff:fe08:218c/64 scope link
       valid_lft forever preferred_lft forever
```

15. SMF configuration and session establishment.

Session details are as shown below.

Table 5. SMF Simulator Configuration

Abbreviation	Description
Number of UEs	12,500 per UPF x 10 UPFs = 125,000
Flows per UE	10
PDU Sessions per UE	1
Packet Detection Rules (PDRs)	20 (10 Uplink / 10 Downlink)
Forwarding Action Rules (FARs)	2
Quality Enforcement Rules (QERs)	2 per PDR
Usage Reporting Rules (URRs)	3 per PDR
Uplink/Downlink (UL/DL) Ratio	1:3

```
# oc exec -it smf -- /bin/bash
# cd /opt/intel/smf
# ip a

smf_est<n>.json and config<n>.txt files are used for UPF<n>. These are included in the UPF base image.
smfsim is the SMF simulator binary file.
config<n>.txt is the configuration input file for the upf instance n.
smf_est<n>.json is the input json file for upf instance n.
```

Example of SMF session establishment with UPF1:

Edit /opt/intel/smf/config1.txt to add the local IP address of the eth0 port of the SMF pod and the eth0 port of the UPF1 pod.

Establish 12500 sessions with UPF1 using 1 gNodeB VF and a session establishment rate of 200 sessions per second.

```
# ./smfsim -n 12500 -r 200 -g 1 -j smf_est1.json -c config1.txt -d CRIT
```

Verify on the DUT that the UPF is showing the number of sessions established:

```
# oc exec -it upf1 -- /bin/bash
# cd /opt/intel/scripts/
# source lib.sh 1
# ./vppctlx show dpe stats |grep -A10 ”API Level Statistics”
API Level Statistics:
        API        CREATE_SESS  MODIFY_SESS  DEL_SESS
        REQ:       12500      0          0
        RSP-S:     12500      0          0
        RSP-F:     0          0          0
        URR-NOTI:  0
        DDN-NOTI:  0
```

7 Intel® Infrastructure Power Manager (IPM) Installation Steps

This section details the step-by-step instructions for deploying the Intel Infrastructure Power Manager (IPM). Please contact your Intel representative to get access to the Intel Infrastructure Power Manager download links.

1. Download the latest release package IPM 23.12 from this link: https://cdrdv2.intel.com/v1/dl/getContent/812070

2. The Users Guide is available at this link: https://cdrdv2.intel.com/v1/dl/getContent/812071

3. Untar the release package and files within the package, setup a directory with the required binary and config file required for IPM installation. Create a namespace for the IPM agent.

```
# mkdir ipm
# cd ipm

Copy the release to this directory and untar the file
#
# tar xzf IPM-23.12_binary.tar.gz
# ll
total 41668
-rw-r--r--. 1 root     root        10408 Dec 20 12:03 'Intel OBL Internal Use License Agreement
[v2022.12.20].txt'
-rw-rw-r--. 1 dvuser63 dvuser63 21282345 Jan 23 13:11  IPM-23.12_binary.tar.gz
-rw-r--r--. 1 root     root       218737 Dec 20 15:22  ipm_agent_binary_v23.12.tar.gz
-rw-r--r--. 1 root     root        14304 Dec 20 15:23  ipm_collateral_v23.12.tar.gz
-rw-r--r--. 1 root     root     21134487 Dec 20 15:23  ipm_watcher_binary_v23.12.tar.gz

# tar xzf ipm_agent_binary_v23.12.tar.gz
# tar xzf ipm_collateral_v23.12.tar.gz
# ll
total 41668
-rw-r--r--. 1 root     root        10408 Dec 20 12:03 'Intel OBL Internal Use License Agreement
[v2022.12.20].txt'
-rw-rw-r--. 1 dvuser63 dvuser63 21282345 Jan 23 13:11  IPM-23.12_binary.tar.gz
drwxr-xr-x. 4 root     root          177 Dec 14 10:44  ipm_agent_binary_v23.12
-rw-r--r--. 1 root     root       218737 Dec 20 15:22  ipm_agent_binary_v23.12.tar.gz
drwxr-xr-x. 5 root     root           46 Dec 15 05:19  ipm_collateral_v23.12
-rw-r--r--. 1 root     root        14304 Dec 20 15:23  ipm_collateral_v23.12.tar.gz
-rw-r--r--. 1 root     root     21134487 Dec 20 15:23  ipm_watcher_binary_v23.12.tar.gz

# oc new-project ipm-agent

# mkdir ~/IPM/23.12

# cp -r ipm_collateral_v23.12/docker/centos8.2 ~/IPM/v23.12/
# ll ~/IPM/v23.12/
total 0
drwxr-xr-x. 5 root root 77 Jan 23 16:36 centos8.2
# ll ~/IPM/v23.12/centos8.2/
total 4
drwxr-xr-x. 3 root root 121 Jan 23 16:36 dpdk-l3fwd
drwxr-xr-x. 2 root root 109 Jan 23 16:36 ipm-agent
```

```
drwxr-xr-x. 2 root root  92 Jan 23 16:36 ipm-watcher
-rw-r--r--. 1 root root 499 Jan 23 16:36 README.md

# ll ~/IPM/v23.12/centos8.2/ipm-agent/
total 20
-rw-r--r--. 1 root root  498 Jan 23 16:36 config.json
-rw-r--r--. 1 root root  891 Jan 23 16:36 Dockerfile
-rw-r--r--. 1 root root  479 Jan 23 16:36 Makefile
-rw-r--r--. 1 root root 1232 Jan 23 16:36 README.md
-rwxr-xr-x. 1 root root  688 Jan 23 16:36 start-docker-ipm-agent.sh

# cp -r ipm_agent_binary_v23.12/centos8.2/ipm-agent ~/IPM/v23.12/centos8.2/ipm-agent/

# cd ~/IPM/v23.12/centos8.2/ipm-agent/
[root@arch05 ipm-agent]# ll
total 276
-rw-r--r--. 1 root root    498 Jan 23 16:36 config.json
-rw-r--r--. 1 root root    891 Jan 23 16:36 Dockerfile
-rwxr-xr-x. 1 root root 260664 Jan 23 16:38 ipm-agent
-rw-r--r--. 1 root root    479 Jan 23 16:36 Makefile
-rw-r--r--. 1 root root   1232 Jan 23 16:36 README.md
-rwxr-xr-x. 1 root root    688 Jan 23 16:36 start-docker-ipm-agent.sh
```

4. The directory ~/IPM/v23.12/centos8.2/ipm-agent now has the config file, Dockerfile and Makefile required to build, tag and upload the IPM image to the local registry. Please modify the Dockerfile, Makefile and config.json file examples shown here, based on your test setup and project requirements.

```
# cd ~/IPM/v23.12/centos8.2/ipm-agent/
# ll
total 292
-rw-r--r--. 1 root root   3873 Jan 23 16:53 config.json
-rw-r--r--. 1 root root    893 Jan 23 16:44 Dockerfile
-rwxr-xr-x. 1 root root 260664 Jan 23 16:38 ipm-agent
-rw-r--r--. 1 root root   1081 Jan 23 17:56 ipm-agent-pod.yaml
-rw-r--r--. 1 root root    583 Jan 23 17:32 Makefile
-rw-r--r--. 1 root root   1232 Jan 23 16:36 README.md
-rwxr-xr-x. 1 root root    688 Jan 23 16:36 start-docker-ipm-agent.sh

# vim Dockerfile Makefile config.json

# cat Makefile
image_version=23.12
image_name=ipm-agent-centos
registry=<path to local registry>/ipm-agent

all: Dockerfile ipm-agent
        docker build -t ${image_name}:${image_version} -t ${image_name}:${image_version} -t
${registry}/${image_name}:${image_version} .

push:
        docker push ${registry}/${image_name}:${image_version}


clean:
        docker rmi ${image_name}:$(image_version) ${image_name}:${image_version} --force
        rm ipm-agent config.json

# cat Dockerfile

#SPDX-License-Identifier: BSD-3-Clause
#Copyright(c) 2022 Intel Corporation

FROM centos:8

WORKDIR /etc/yum.repos.d/
RUN sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-*
RUN sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g' /etc/yum.repos.d/CentOS-*
RUN yum update -y
```

```
# Install base tools
RUN yum install -y sudo vim wget

# Install IPM-agent required libraries
RUN yum makecache --refresh
RUN yum -y install dnf
RUN dnf --enablerepo=powertools install -y libnghttp2-devel
RUN yum install -y jansson jansson-devel libevent-devel libcurl-devel libvirt-devel

RUN mkdir -p /opt/ipm-agent/config
WORKDIR /opt/ipm-agent/

# Copy compiled binary and configuration to image
COPY ./ipm-agent /opt/ipm-agent/ipm-agent
COPY ./config.json /opt/ipm-agent/config/config.json

# cat /opt/ipm-agent/config/config.json
{
        "global": {
                "loop time": 3000,
                "log time": 10,
                "core": 207,
                "default upper": "2100",
                "default lower": "800",
                "inactive upper": "2100",
                "inactive lower": "800",
                "uncore upper": "1700",
                "uncore lower": "800",
                "inactive loop time": 60,
                "active between": "00:00-23:59",
                "hysteresis": 3,
                "rocketing": true,
                "rocketing threshold": 85
        },
        "upf1": {
                "enabled": true,
                "cores": "1-9,105-113",
                "upper": "default",
                "lower": "default",
                "mode": "dpdk",
                "capacity factor": "30",
                "telemetry path": "/var/run/dpdk/upf1/dpdk_telemetry.v2"
        },
        "upf2": {
                "enabled": true,
                "cores": "10-18,114-122",
                "upper": "default",
                "lower": "default",
                "mode": "dpdk",
                "capacity factor": "30",
                "telemetry path": "/var/run/dpdk/upf2/dpdk_telemetry.v2"
        },
        "upf3": {
                "enabled": true,
                "cores": "19-27,123-131",
                "upper": "default",
                "lower": "default",
                "mode": "dpdk",
                "capacity factor": "30",
                "telemetry path": "/var/run/dpdk/upf3/dpdk_telemetry.v2"
        },
        "upf4": {
                "enabled": true,
                "cores": "28-36,132-140",
                "upper": "default",
                "lower": "default",
                "mode": "dpdk",
                "capacity factor": "30",
                "telemetry path": "/var/run/dpdk/upf4/dpdk_telemetry.v2"
        },
        "upf5": {
                "enabled": true,
                "cores": "37-45,141-149",
                "upper": "default",
```

```
                "lower": "default",
                "mode": "dpdk",
                "capacity factor": "30",
                "telemetry path": "/var/run/dpdk/upf5/dpdk_telemetry.v2"
        },
        "upf6": {
                "enabled": true,
                "cores": "53-61,157-165",
                "upper": "default",
                "lower": "default",
                "mode": "dpdk",
                "capacity factor": "30",
                "telemetry path": "/var/run/dpdk/upf6/dpdk_telemetry.v2"
        },
        "upf7": {
                "enabled": true,
                "cores": "62-70,166-174",
                "upper": "default",
                "lower": "default",
                "mode": "dpdk",
                "capacity factor": "30",
                "telemetry path": "/var/run/dpdk/upf7/dpdk_telemetry.v2"
        },
        "upf8": {
                "enabled": true,
                "cores": "71-79,175-183",
                "upper": "default",
                "lower": "default",
                "mode": "dpdk",
                "capacity factor": "30",
                "telemetry path": "/var/run/dpdk/upf8/dpdk_telemetry.v2"
        },
        "upf9": {
                "enabled": true,
                "cores": "80-88,184-192",
                "upper": "default",
                "lower": "default",
                "mode": "dpdk",
                "capacity factor": "30",
                "telemetry path": "/var/run/dpdk/upf9/dpdk_telemetry.v2"
        },
        "upf10": {
                "enabled": true,
                "cores": "89-97,193-201",
                "upper": "default",
                "lower": "default",
                "mode": "dpdk",
                "capacity factor": "30",
                "telemetry path": "/var/run/dpdk/upf10/dpdk_telemetry.v2"
        }
}
```

5. Build the ipm-agent image and upload it to the local registry.

```
# make
docker build -t ipm-agent-centos:23.12 -t ipm-agent-centos:23.12 -t <path to local registry>/ipm-agent/ipm-
agent-centos:23.12 .
STEP 1/14: FROM centos:8
Resolved "centos" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull quay.io/centos/centos:8...
Getting image source signatures
Copying blob a1d0c7532777 done
Copying config 5d0da3dc97 done
Writing manifest to image destination
STEP 2/14: WORKDIR /etc/yum.repos.d/
--> 043540e7c78d
STEP 3/14: RUN sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-*
--> 6666cdbe47b7
STEP 4/14: RUN sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g'
/etc/yum.repos.d/CentOS-*
--> 710e624d570d
```

```
STEP 5/14: RUN ls -lt /etc/yum.repos.d/
total 48
-rw-r--r--. 1 root root  721 Jan 10 23:46 CentOS-Linux-AppStream.repo
-rw-r--r--. 1 root root  706 Jan 10 23:46 CentOS-Linux-BaseOS.repo
-rw-r--r--. 1 root root 1132 Jan 10 23:46 CentOS-Linux-ContinuousRelease.repo
-rw-r--r--. 1 root root  318 Jan 10 23:46 CentOS-Linux-Debuginfo.repo
-rw-r--r--. 1 root root  734 Jan 10 23:46 CentOS-Linux-Devel.repo
-rw-r--r--. 1 root root  706 Jan 10 23:46 CentOS-Linux-Extras.repo
-rw-r--r--. 1 root root  721 Jan 10 23:46 CentOS-Linux-FastTrack.repo
-rw-r--r--. 1 root root  742 Jan 10 23:46 CentOS-Linux-HighAvailability.repo
-rw-r--r--. 1 root root  693 Jan 10 23:46 CentOS-Linux-Media.repo
-rw-r--r--. 1 root root  708 Jan 10 23:46 CentOS-Linux-Plus.repo
-rw-r--r--. 1 root root  726 Jan 10 23:46 CentOS-Linux-PowerTools.repo
-rw-r--r--. 1 root root  898 Jan 10 23:46 CentOS-Linux-Sources.repo
--> 9ead7d676c01
STEP 6/14: RUN sed -i 's/enabled=0/enabled=1/g' /etc/yum.repos.d/CentOS-Linux-PowerTools.repo
--> fa087de09767
STEP 7/14: RUN yum update -y
CentOS Linux 8 - AppStream                      4.4 MB/s | 8.4 MB     00:01
CentOS Linux 8 - BaseOS                         2.8 MB/s | 4.6 MB     00:01
CentOS Linux 8 - Extras                          10 kB/s |  10 kB     00:01
CentOS Linux 8 - PowerTools                     1.6 MB/s | 2.3 MB     00:01
Dependencies resolved.
================================================================================
 Package                    Arch   Version                     Repo       Size
================================================================================
Upgrading:
 bash                       x86_64 4.4.20-2.el8                baseos    1.5 M
 bind-export-libs           x86_64 32:9.11.26-6.el8            baseos    1.1 M
 binutils                   x86_64 2.30-108.el8_5.1            baseos    5.8 M
 ca-certificates            noarch 2021.2.50-80.0.el8_4        baseos    390 k
 ....
 ....
 ....
  platform-python-pip-9.0.3-20.el8.noarch
  python3-unbound-1.7.3-17.el8.x86_64
  rpm-plugin-systemd-inhibit-4.14.3-19.el8.x86_64
  shared-mime-info-1.9-3.el8.x86_64
  trousers-0.3.15-1.el8.x86_64
  trousers-lib-0.3.15-1.el8.x86_64
  unbound-libs-1.7.3-17.el8.x86_64
  which-2.21-16.el8.x86_64
  xkeyboard-config-2.28-1.el8.noarch

Complete!
--> 2ecc2db3a894
STEP 8/14: RUN yum install -y sudo vim wget
Last metadata expiration check: 0:00:41 ago on Wed Jan 10 23:46:18 2024.
Dependencies resolved.
================================================================================
 Package                  Arch      Version                Repository    Size
================================================================================
Installing:
 sudo                     x86_64    1.8.29-7.el8_4.1       baseos       925 k
 vim-enhanced             x86_64    2:8.0.1763-16.el8      appstream    1.4 M
 wget                     x86_64    1.19.5-10.el8          appstream    734 k
Installing dependencies:
 gpm-libs                 x86_64    1.20.7-17.el8          appstream     39 k
 libpsl                   x86_64    0.20.2-6.el8           baseos        61 k
 publicsuffix-list-dafsa  noarch    20180723-1.el8         baseos        56 k
 vim-common               x86_64    2:8.0.1763-16.el8      appstream    6.3 M
 vim-filesystem           noarch    2:8.0.1763-16.el8      appstream     49 k

Transaction Summary
================================================================================
Install  8 Packages

Total download size: 9.5 M
Installed size: 36 M
Downloading Packages:
(1/8): gpm-libs-1.20.7-17.el8.x86_64.rpm         71 kB/s |  39 kB     00:00
(2/8): vim-filesystem-8.0.1763-16.el8.noarch.rp 217 kB/s |  49 kB     00:00
(3/8): vim-enhanced-8.0.1763-16.el8.x86_64.rpm  1.4 MB/s | 1.4 MB     00:00
(4/8): vim-common-8.0.1763-16.el8.x86_64.rpm    5.6 MB/s | 6.3 MB     00:01
```

```
(5/8): libpsl-0.20.2-6.el8.x86_64.rpm           212 kB/s |  61 kB     00:00
(6/8): wget-1.19.5-10.el8.x86_64.rpm            1.5 MB/s | 734 kB     00:00
(7/8): publicsuffix-list-dafsa-20180723-1.el8.n 294 kB/s |  56 kB     00:00
(8/8): sudo-1.8.29-7.el8_4.1.x86_64.rpm         3.1 MB/s | 925 kB     00:00
--------------------------------------------------------------------------------
Total                                           6.2 MB/s | 9.5 MB     00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : publicsuffix-list-dafsa-20180723-1.el8.noarch          1/8
  Installing       : libpsl-0.20.2-6.el8.x86_64                             2/8
  Installing       : vim-filesystem-2:8.0.1763-16.el8.noarch                3/8
  Installing       : vim-common-2:8.0.1763-16.el8.x86_64                    4/8
....
....
....

Installed:
  gpm-libs-1.20.7-17.el8.x86_64
  libpsl-0.20.2-6.el8.x86_64
  publicsuffix-list-dafsa-20180723-1.el8.noarch
  sudo-1.8.29-7.el8_4.1.x86_64
  vim-common-2:8.0.1763-16.el8.x86_64
  vim-enhanced-2:8.0.1763-16.el8.x86_64
  vim-filesystem-2:8.0.1763-16.el8.noarch
  wget-1.19.5-10.el8.x86_64

Complete!
--> a6b4f22286bd
STEP 9/14: RUN yum install -y jansson jansson-devel libnghttp2-devel libevent-devel libcurl-devel
Last metadata expiration check: 0:00:51 ago on Wed Jan 10 23:46:18 2024.
Dependencies resolved.
================================================================================
 Package                Arch       Version                 Repository      Size
================================================================================
Installing:
 jansson                x86_64     2.11-3.el8              baseos          46 k
 jansson-devel          x86_64     2.11-3.el8              appstream       16 k
 libcurl-devel          x86_64     7.61.1-22.el8           baseos         834 k
 libevent-devel         x86_64     2.1.8-5.el8             appstream      104 k
 libnghttp2-devel       x86_64     1.33.0-3.el8_2.1        powertools      60 k
Installing dependencies:
 libpkgconf             x86_64     1.4.2-1.el8             baseos          35 k
 pkgconf                x86_64     1.4.2-1.el8             baseos          38 k
 pkgconf-m4             noarch     1.4.2-1.el8             baseos          17 k
 pkgconf-pkg-config     x86_64     1.4.2-1.el8             baseos          15 k

Transaction Summary
================================================================================
Install  9 Packages

Total download size: 1.1 M
Installed size: 2.2 M
Downloading Packages:
(1/9): jansson-devel-2.11-3.el8.x86_64.rpm       23 kB/s |  16 kB     00:00
(2/9): jansson-2.11-3.el8.x86_64.rpm             60 kB/s |  46 kB     00:00
(3/9): libevent-devel-2.1.8-5.el8.x86_64.rpm    122 kB/s | 104 kB     00:00
(4/9): libpkgconf-1.4.2-1.el8.x86_64.rpm        148 kB/s |  35 kB     00:00
(5/9): pkgconf-1.4.2-1.el8.x86_64.rpm           171 kB/s |  38 kB     00:00
(6/9): pkgconf-m4-1.4.2-1.el8.noarch.rpm         82 kB/s |  17 kB     00:00
(7/9): pkgconf-pkg-config-1.4.2-1.el8.x86_64.rp  74 kB/s |  15 kB     00:00
(8/9): libcurl-devel-7.61.1-22.el8.x86_64.rpm   1.3 MB/s | 834 kB     00:00
(9/9): libnghttp2-devel-1.33.0-3.el8_2.1.x86_64 228 kB/s |  60 kB     00:00
...
...
...
...
  Verifying        : pkgconf-pkg-config-1.4.2-1.el8.x86_64                 8/9
  Verifying        : libnghttp2-devel-1.33.0-3.el8_2.1.x86_64              9/9

Installed:
```

```
  jansson-2.11-3.el8.x86_64                  jansson-devel-2.11-3.el8.x86_64
  libcurl-devel-7.61.1-22.el8.x86_64         libevent-devel-2.1.8-5.el8.x86_64
  libnghttp2-devel-1.33.0-3.el8_2.1.x86_64   libpkgconf-1.4.2-1.el8.x86_64
  pkgconf-1.4.2-1.el8.x86_64                 pkgconf-m4-1.4.2-1.el8.noarch
  pkgconf-pkg-config-1.4.2-1.el8.x86_64

Complete!
--> ae123dc4ab71
STEP 10/14: RUN mkdir -p /opt/ipm-agent/config
--> 3e223c5d370a
STEP 11/14: WORKDIR /opt/ipm-agent/
--> a488eebc216f
STEP 12/14: COPY ./ipm-agent /opt/ipm-agent/ipm-agent
--> 5a1f9b9aa204
STEP 13/14: COPY ./config.json /opt/ipm-agent/config/config.json
--> 6531015cc0be
STEP 14/14: COPY ./config.json /opt/ipm-agent/config/config.json
COMMIT ipm-agent-image:23.12
--> ec0ecfe55d32
Successfully tagged localhost/ipm-agent-centos:23.12
Successfully tagged <path to local registry>/ipm-agent/ipm-agent-centos:23:12
ec0ecfe55d32f454e14198c75cee72a0ed21bb6fa846065d1dc233f3ecf42153
```

To push to the local registry, "make push" can be used or the following commands:

```
# podman tag localhost/ipm-agent-centos:23.12 <path to local registry>/ipm-agent/ipm-agent-centos

# podman push <path to local registry>/ipm-agent/ipm-agent-centos
Getting image source signatures
Copying blob c13db9c5111c done
Copying blob 413de8f18a52 done
Copying blob 06c9bbd14c7f done
Copying blob 258538215388 done
Copying blob 29e74c6f4feb done
Copying blob 74ddd0ec08fa done
Copying blob c9646d6ba72f done
Copying blob d1ce6c51812a done
Copying blob c6142bf06443 done
Copying blob d58ec62986a4 done
Copying blob 0324a43f85a6 done
Copying blob bd080be5a669 done
Copying config ec0ecfe55d done
Writing manifest to image destination

# podman images |grep centos |grep ipm

# oc get images |grep ipm
```

6. Create the IPM pod with the image built and check the pod status to verify that the state is "Running". The ipm-agent namespace was created in step 3.

```
# cat ipm-agent-pod.yaml
apiVersion: v1

#SPDX-License-Identifier: BSD-3-Clause
#Copyright(c) 2022 Intel Corporation

kind: Pod
metadata:
  name: ipm-agent
  namespace: ipm-agent
  labels:
    ipm-agent: "true"
spec:
  nodeSelector:
    node-role.kubernetes.io/upf: ''
  containers:
  - image: image-registry.openshift-image-registry.svc:5000/ipm-agent/ipm-agent-centos
    name: ipm-agent-centos
    imagePullPolicy: Always
    resources:
```

```
      requests:
        memory: "100Mi"
        cpu: "1"
      limits:
        memory: "200Mi"
        cpu: "1"
    command: [ "/bin/sh" , "-c", "tail -f /dev/null" ]

    volumeMounts:
    - mountPath: /sys/devices
      name: host-devices
    - mountPath: /var/run/dpdk
      name: host-run
    securityContext:
      privileged: true
      allowPrivilegeEscalation: true
      runAsUser: 0
  volumes:
  - name: host-devices
    hostPath:
      path: /sys/devices
      type: Directory
  - name: host-run
    hostPath:
      path: /var/run/dpdk
      type: Directory

# oc apply -f ipm-agent-pod.yaml

# oc get pods -n ipm-agent
NAME        READY   STATUS    RESTARTS   AGE
ipm-agent   1/1     Running   0          6d23h
```

7. Run the IPM command inside the pod to start the ipm-agent and setup frequency control and enabling power control for the cores being monitored, without UPF traffic, to measure "idle" state power usage and with various levels of UPF traffic load for power usage statistics when the CPU utilization varies based on the traffic percentage of the line rate. For details of each core's frequency **downshifting** and **upshifting** based on each core's utilization, a log level of INFO can be used. For power measurements, use a log level of NONE.

   Monitoring and frequency control starts on each upf using /var/run/dpdk/upf<#>/dpdk_telemetry.v2 endpoint based on the core utilization of each core in that UPF as reported by "/eal/lcore/busyness". The config.json file specifies which cores P-states are being controlled and the upper and lower limits.

```
# oc exec -it ipm-agent -- /bin/bash
[root@ipm-agent ipm-agent]# /opt/ipm-agent/ipm-agent -c /opt/ipm-agent/config/config.json -l INFO
```

8. Scaling driver and scaling governor being used in the worker node.

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
performance

# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_driver
acpi-cpufreq
```

8 TRex Traffic Generator

TRex is a Linux traffic generator which works on any COTS x86 server. It benchmarks platforms and workloads using realistic traffic. This section details the step-by-step instructions for deploying the TRex Traffic Generator.
This section details the step-by-step instructions for deploying the TRex Traffic Generator.

1. Hardware recommendations:
   a. Any x86 servers which support Intel® E810-2CQDA2 and E810-CQDA2. The number of servers required depend on the number of PCIe x16 slots and the number of cores in the processors used in each server.

b. For this VRC, three TRex servers and a total of 10 x 100GbE PCIe NICs were used.

2. Software recommendations:
   a. Install Red Hat Enterprise Linux OS on the TRex servers.
   b. Download the latest TRex package.
   c. Please read the TRex manual at this link for details on downloading, installing, and running TRex traffic generators.

8.1 TRex Traffic Generators Configuration

This section details the instructions for deploying the TRex Traffic Generator for 5G core workloads.

Install TRex software on the required number of TRex servers. Examples of a TRex config file and a traffic profile in a python script to emulate a 5G core traffic profile are shown in this section.

Configure the NIC ports to use destination IP addresses and MAC addresses which match those of the UPF pod instances on the DUT.

Allocate one SR-IOV virtual function to each NIC port. Turn trust on and spoof check off. Bind the virtual function interface for each port to the DPDK-compatible driver vfio-pci. Configure the TRex yaml file, the traffic profile script and start the traffic.

Verify that one Virtual Function (VF) is allocated to each 100G NIC port and the VF is bound to the vfio-pci driver.

Here are some useful commands:

```
lshw -class network -businfo |grep E810-C
echo 1 > /sys/bus/pci/devices/<domain>:/<bus>:<device>.<function>/sriov_numvfs
e.g.
echo 1 > /sys/bus/pci/devices/0000\:16\:00.0/sriov_numvfs
cat /sys/bus/pci/devices/0000\:16\:00.0/sriov_numvfs

lshw -class network -businfo  |grep -i virtual
ip link set <port name> vf 0 trust on
ip link set <port name> vf 0 spoofchk off

wget https://raw.githubusercontent.com/DPDK/dpdk/v22.11/usertools/dpdk-devbind.py
chmod +x dpdk-devbind.py
dpdk-devbind.py -b vfio-pci <bus>:01.0
e.g.
dpdk-devbind.py -b vfio-pci 16:01.0

dpdk-devbind.py -s

ip link set dev <port name> vf 0 mac <mac address>

cat /sys/devices/system/cpu/cpu<core_number>/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu<core_number>/cpufreq/scaling_max_freq
cat /sys/devices/system/cpu/cpu<core_number>/cpufreq/scaling_min_freq
```

Example TRex config file:

```
# cd /opt/trex/v3.00/
```

Edit trex_cfg.yaml to add the bus:device:function of the local TRex NIC port virtual functions as the source interfaces and the MAC addresses of the N6 (feth2) port of the UPF pods to which the local ports are connected as port-info.

Example of trex_cfg.yaml:

```
- port_limit      : 8
  version         : 2
    #
#List of interfaces. Change to suit your setup. Use ./dpdk_setup_ports.py -s to see available options
  interfaces    : ['16:01.0','2a:01.0']
  port_info     :
          - dest_mac         : '52:d4:b3:e5:19:38'
```

```
          - dest_mac         : '42:3c:50:f7:bb:3b'
```

In one window on each TRex server, run this command:

```
# ./_t-rex-64 --cfg ./trex_cfg.yaml --no-scapy-server -i -c 11
```

8.2 TRex Traffic Generators Setup – TRex Client

Table 6. TRex Traffic Profile

Abbreviation	Description
Number of UEs per NIC port	12,500
Flows per UE	10
Uplink/Downlink (UL/DL) Ratio	1:3
Packet Size	650 Bytes

Start the TRex client in a different window on each TRex server.

Using trex-console, start the traffic using one python script per UPF to transmit and receive traffic for that UPF. One 100G NIC port on a TRex server is connected back-to-back to a 100G NIC port on the DUT for each UPF.

Example of a python script and command to start traffic for the UPF1 pod:

```
# cd /opt/trex/v3.00/

# ./trex-console
trex>start -f UPF1.py -p 0 -m 18.5mpps --force
```

Modify the speed (mpps) of the generated traffic in this start command to get 0% packet loss on both the TRex side and the DUT side.

The tui command can be used to view the Rx and Tx packets from the TRex side.

```
trex> tui
```

The DUT benchmarks can be viewed using the /opt/intel/scripts/pckts script in each UPF pod.

Note: feth0 is the N3 interface and feth2 is the N6 interface. The Uplink / Downlink ration is 1:3. The Uplink packets are received on N3 and are transmitted on N6. The Downlink packets are received on N6 and transmitted on N3. As a result 3 times more packets are received on N6 compared to N3 and 3 times more packets are transmitted on N3 compared to N6. The TOTAL RATES have the final miss rate and Tx bitrate. These are reported as packet loss and I/O throughput.

Sample benchmarks on the DUT using the "pckts" script, for one UPF with a line rate of 100 Gbps:

```
# oc exec -it upf1 -- /bin/bash
[root@upf1 /]#
[root@upf1 /]# cd /opt/intel/scripts/
[root@upf1 scripts]#
[root@upf1 scripts]# source lib.sh 1

# [root@upf1 scripts]# ./pckts 1 60

fpeth0 [interval 20.02s]

  RX packet rate:         3882684 PPS
  TX packet rate:        11648051 PPS
       loss rate:        -7765367 PPS  (-199.99997%)
       miss rate:               0 PPS  ( 0.00000%)
```

```
     tx err rate:               0 PPS  ( 0.00000%)
       drop rate:               0 PPS
  RX bitrate:           23296.105 Mbps
  TX bitrate:           73988.424 Mbps

fpeth2 [interval 20.02s]

  RX packet rate:        11648044 PPS
  TX packet rate:         3882683 PPS
       loss rate:         7765361 PPS  ( 66.66665%)
       miss rate:               0 PPS  ( 0.00000%)
     tx err rate:               0 PPS  ( 0.00000%)
       drop rate:               0 PPS
  RX bitrate:           69888.267 Mbps
  TX bitrate:           21929.395 Mbps

TOTAL RATES [interval 20.02s]

  RX packet rate:        15530728 PPS
  TX packet rate:        15530734 PPS
       loss rate:              -6 PPS  (-0.00004%)
       miss rate:               0 PPS  ( 0.00000%)
     tx err rate:               0 PPS  ( 0.00000%)
       drop rate:               0 PPS
  RX bitrate:           93184.372 Mbps
  TX bitrate:           95917.819 Mbps

TOTAL COUNTERS measured for 20.02s interval

  RX packets:           310970974
  TX packets:           310971097
        loss:                -123  (-0.00004%)
        miss:                   0  ( 0.00000%)
      tx err:                   0  ( 0.00000%)
        drop:                   4
```

8.3 Benchmarking without IPM

Follow the steps described in this document without using the ipm-agent pod.

8.4 Benchmarking with IPM

Follow the steps described in this document and run the following command in the ipm-agent pod:

```
# oc exec -it ipm-agent -- /bin/bash
[root@ipm-agent ipm-agent]# /opt/ipm-agent/ipm-agent -c /opt/ipm-agent/config/config.json -l NONE

OR

# oc exec -it ipm-agent -- /bin/bash
[root@ipm-agent ipm-agent]# /opt/ipm-agent/ipm-agent -c /opt/ipm-agent/config/config.json -l INFO
```

9 5G FlexCore 2.0 UPF Benchmarks

This section shows the benchmarks of 5G FlexCore 2.0 UPF with Red Hat OpenShift Container Platform on a 2-socket worker node server with Intel® Xeon® Platinum 8470N which has 52 physical cores and 104 logical cores per NUMA node with hyper-threading enabled, 208 logical cores per server, processor base frequency of 1.70 GHz, a maximum turbo frequency of 3.70 GHz, PCIe Gen 5.0, a maximum number of PCIe lanes of 80 and TDP of 300 W per processor resulting in a TDP of 600 W per 2-socket server. The benchmarks using Intel® Infrastructure Power Manager with the IPM agent running are compared with benchmarks without the IPM agent.

9.1 I/O Throughput with and without IPM

The results of the 5G FlexCore 2.0 UPF in a Red Hat OpenShift Container Platform worker node showed an I/O throughput of up to 948 Gbps using IPM, with minimal throughput changes and a packet loss of 0% using a packet size of 650 Bytes.

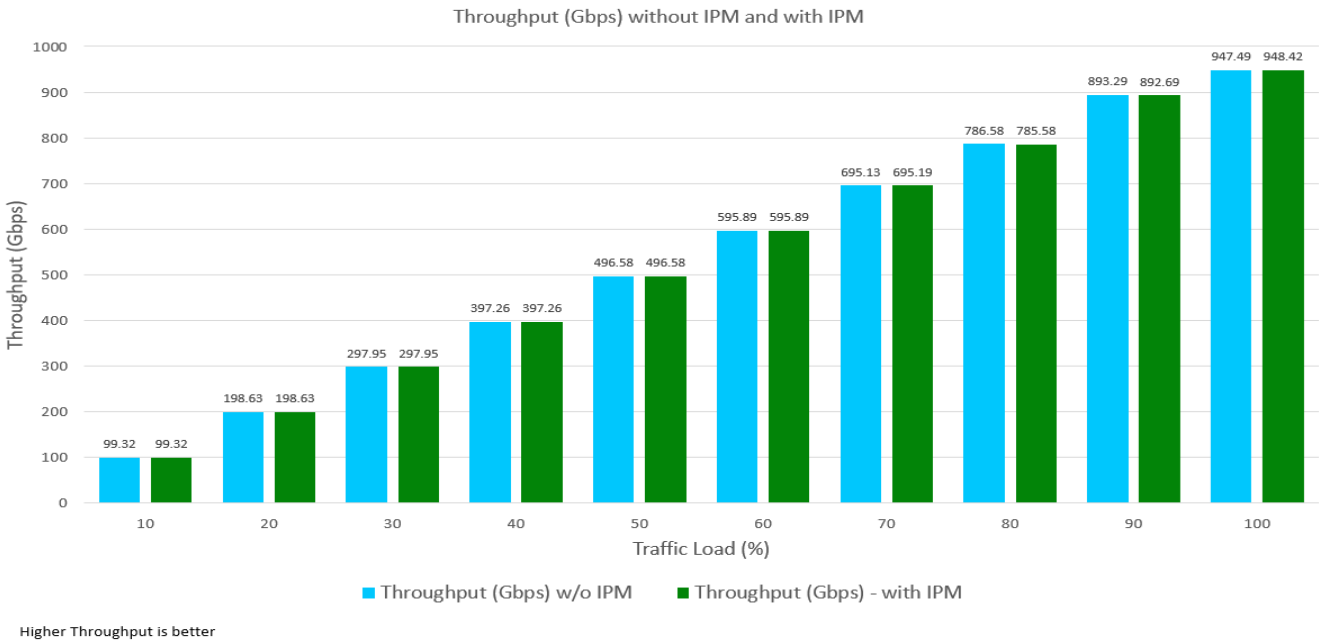

Figure 2. FlexCore 2.0 UPF Throughput (Gbps) with and without IPM [1, 2]

9.2 I/O Throughput with 24-hour Traffic Profile

The results of the 5G FlexCore 2.0 UPF using a realistic 75% traffic profile with a packet loss of 0% using a packet size of 650 Bytes, with and without IPM, demonstrated no impact on throughput and impressive power savings.

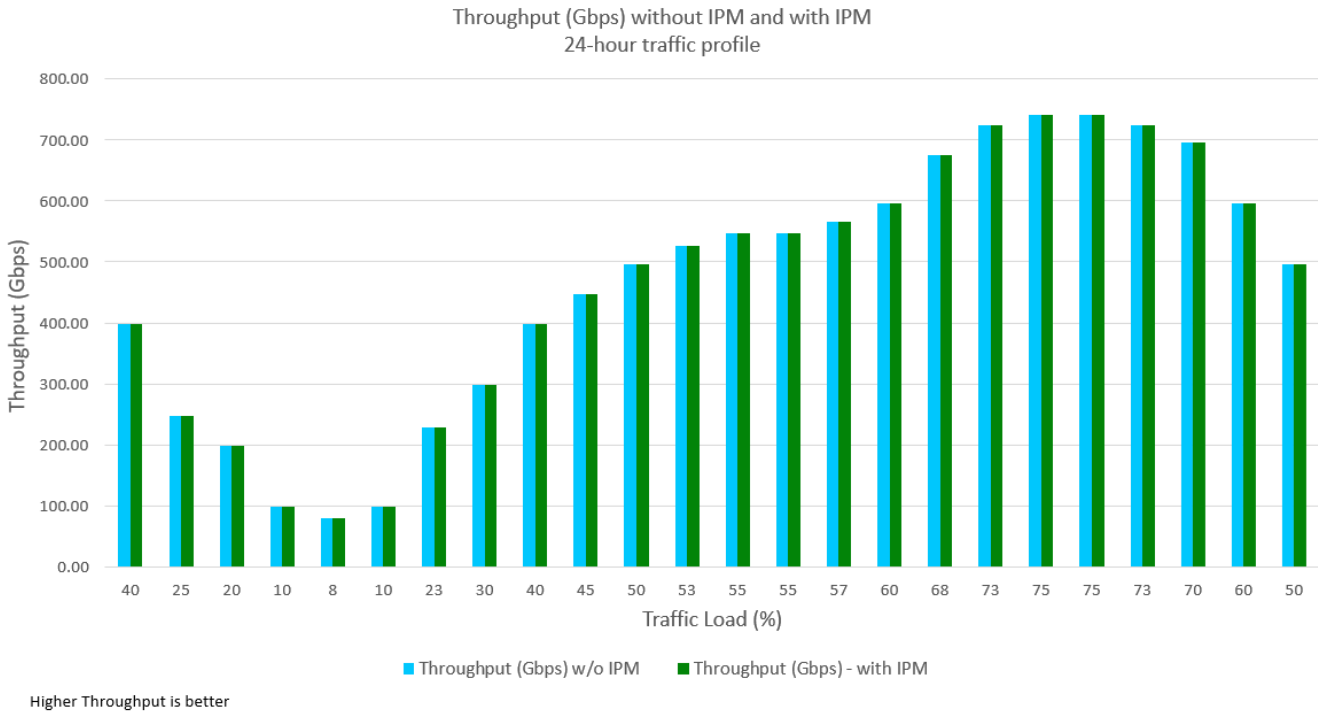

Figure 3. FlexCore 2.0 UPF Throughput (Gbps) with 24-hour traffic profile using IPM [1, 2]

9.3 Power Usage with 24-hour Traffic Profile

The results of the 5G FlexCore 2.0 UPF benchmarks in a RH OCP worker node with and without IPM, using a realistic 75% traffic profile, showed up to a maximum 36% power savings and an average 23% power savings with IPM, with a packet loss of 0% and a packet size of 650 Bytes. Without IPM, the power usage was the same (almost equal to) system TDP even in an idle state with no traffic. Using IPM which leverages the DPDK telemetry logical cores "busyness" to determine actual CPU load for the monitored active UPF worker cores, the CPU frequency was downshifted or upshifted by IPM, depending on the core utilization which changed based on the traffic load percentage, resulting in much lower power consumption and impressive energy efficiency.

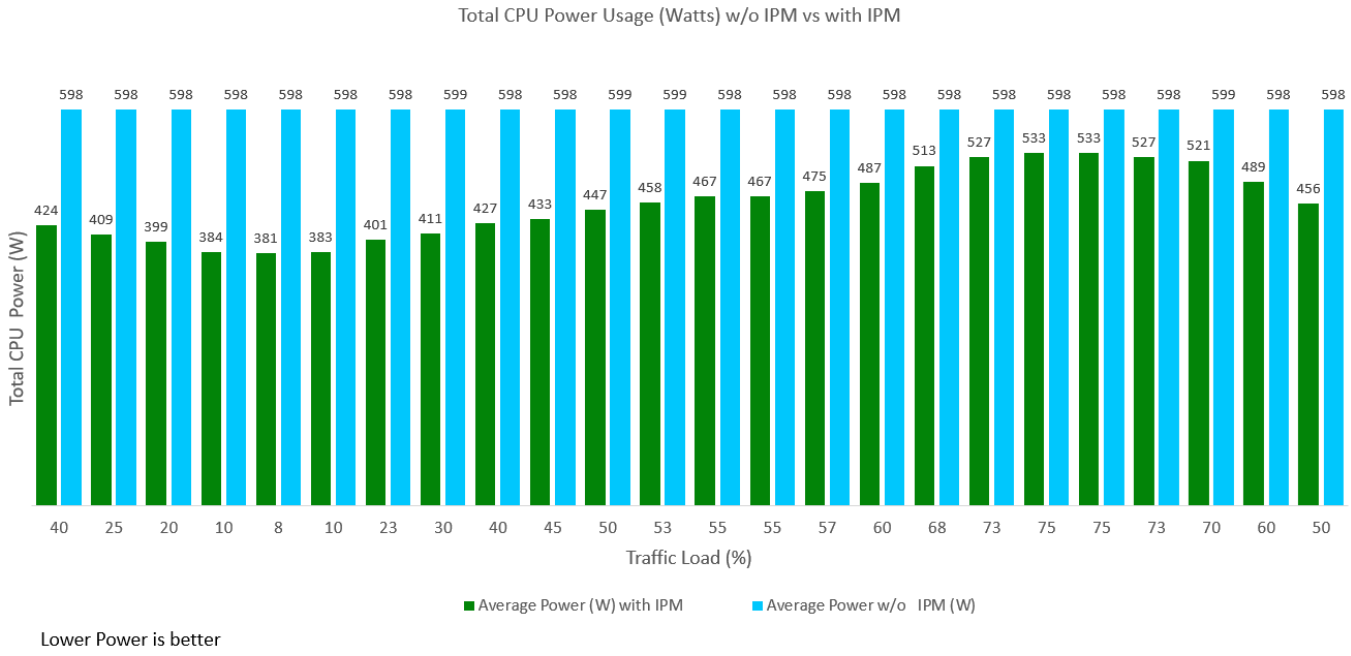

Figure 4. Server Power Savings with 24-hour traffic profile using IPM[1, 2]

10 Summary

This document is a Verified Reference Configuration (VRC) of a 5G Core solution. It defines the hardware and software required, provides guidance on how to configure them in a test setup and includes a step-by-step guide for deploying Intel's FlexCore 2.0 UPF as well as the Intel Infrastructure Power Manager in a Red Hat OpenShift Container Platform infrastructure. It describes how to optimize FlexCore 2.0 UPF for maximum performance and minimum packet loss. In addition, it shows how to leverage the Intel® Infrastructure Power Manager (IPM) to improve energy efficiency by automatically increasing or decreasing CPU frequencies in real-time in response to core utilization which varies based on traffic load, resulting in a significant reduction in overall power consumption.

The outstanding throughput achieved with this 5G core UPF workload running on 4th Gen Intel Xeon Platinum 8470N processors and Intel Network Adapters, was up to **948 Gbps** which was **94.8% of the line rate** of 1000 Gbps, with a packet size of 650 Bytes and **0% packet loss**, with or without IPM.

The usage of the Intel Infrastructure Power Management tool with FlexCore 2.0 UPF running on Intel 4th Gen Xeon Scalable processors demonstrated a **maximum of 36% power savings** and an **average 23% power savings** with IPM using a commercial telco traffic profile. The power consumption was reduced with no impact on performance metrics such as I/O throughput or packet loss. With 5G Core software using DPDK-based applications without power savings technologies, CPU power consumption is always at a maximum. With IPM using DPDK telemetry to compute the actual core "busyness", every monitored core has the optimal power at any time. Therefore, in 5G applications and workloads based on DPDK, usage of the Intel Infrastructure Power Management tool becomes very important for real-time power consumption and cost savings and meeting sustainability goals.

1 Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more on the Performance Index site. Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure. Your costs and results may vary. Intel technologies may require enabled hardware, software or service activation.

2 Configuration

Test by Intel as of 3/17/2024. 1-node, 2x Intel® Xeon® Platinum 8470N, 52 cores, HT On, Turbo On, Total Memory 2048GB (32x64GB DDR5 4800 MT/s [4400 MT/s]), microcode 0x2b0004b1, Red Hat Enterprise Linux CoreOS 413.92.202311151359-0 (Plow), kernel 5.14.0-284.41.1.el9_2.x86_64, DDP ICE COMMS package version 1.3.45.0

0424/SM/PDF Please Recycle 820820-001US