White Paper

Power Saving Edge Computing on SKT 5G MEC

Authors

SK Telecom

Moonyoung, Aaron Chung
Mincheol, Daniel Park
Jian, Li
Keunhyun, Kim

Intel Corporation

Wooram, Alex Kim
Jayden, Lee
Kannan Babu, Ramia
Bhavik, Dhandhalya
Palaniappan, Ramanathan
S, Deepak
Sankar, Chokkalingam
Pinkesh, Shah

1. Introduction

As mobile operators are launching 5G services, they are uncovering a few key challenges, including:

a. Accelerating 5G service deployments with better customer experience leading to increasing revenue, growth and market share.

b. Increased carbon emissions and energy consumption not just due to more bandwidth consumption but also the resulting additional costs for carbon offsets as well as regulation from governments and policy makers globally.

Thus, operators are considering the adoption of power saving 5G network infrastructure from access to core that has been designed to require less energy, in addition to deploying edge computing infrastructure that can reduce the total amount of data traversing the network by running applications at the edge. In edge networks, data can be processed and stored nearer to the end user and devices, rather than relying on data centers that can be hundreds of miles away. This could lead to a significant reduction in energy consumption related to network transport, while also benefiting from low latency that edge provides. So, to resolve the operator's challenge, this paper describes tests of power saving edge computing capabilities on SKT's 5G MEC servers by applying Intel reference software for dynamic power control and Intel reference software for edge energy savings, a feature that is available on 3rd Generation Intel® Xeon® Scalable processors and Intel® Ethernet Adaptor E810.

Object of Test

This document describes the results of testing the SKT 5G MEC software running on Intel® architecture server boards, including benchmarking data and instructions on how to replicate the tests. Telecom equipment manufacturers (TEMs) and independent software vendors (ISVs) can use the implementation guidelines from this work to optimize and further develop power saving solutions for energy wise high-performance edge computing services.

Test objectives were:

- Demonstrate microservices based application (VM, PoD) power measurement using the telemetry plugin for Intel reference software for dynamic power control and Intel reference software for edge energy savings to obtain data and using AI inference for power prediction.
- Show a dynamic reduction in microservices applications (VM, PoD) and Kubernetes node level power saving through CPU P-state control over a 24-hour time frame and during specific off-peak time frames.
- Validate inter-operability between Intel reference software for dynamic power control and Intel reference software for edge energy savings by testing multiple application containers on SKT 5G MEC.

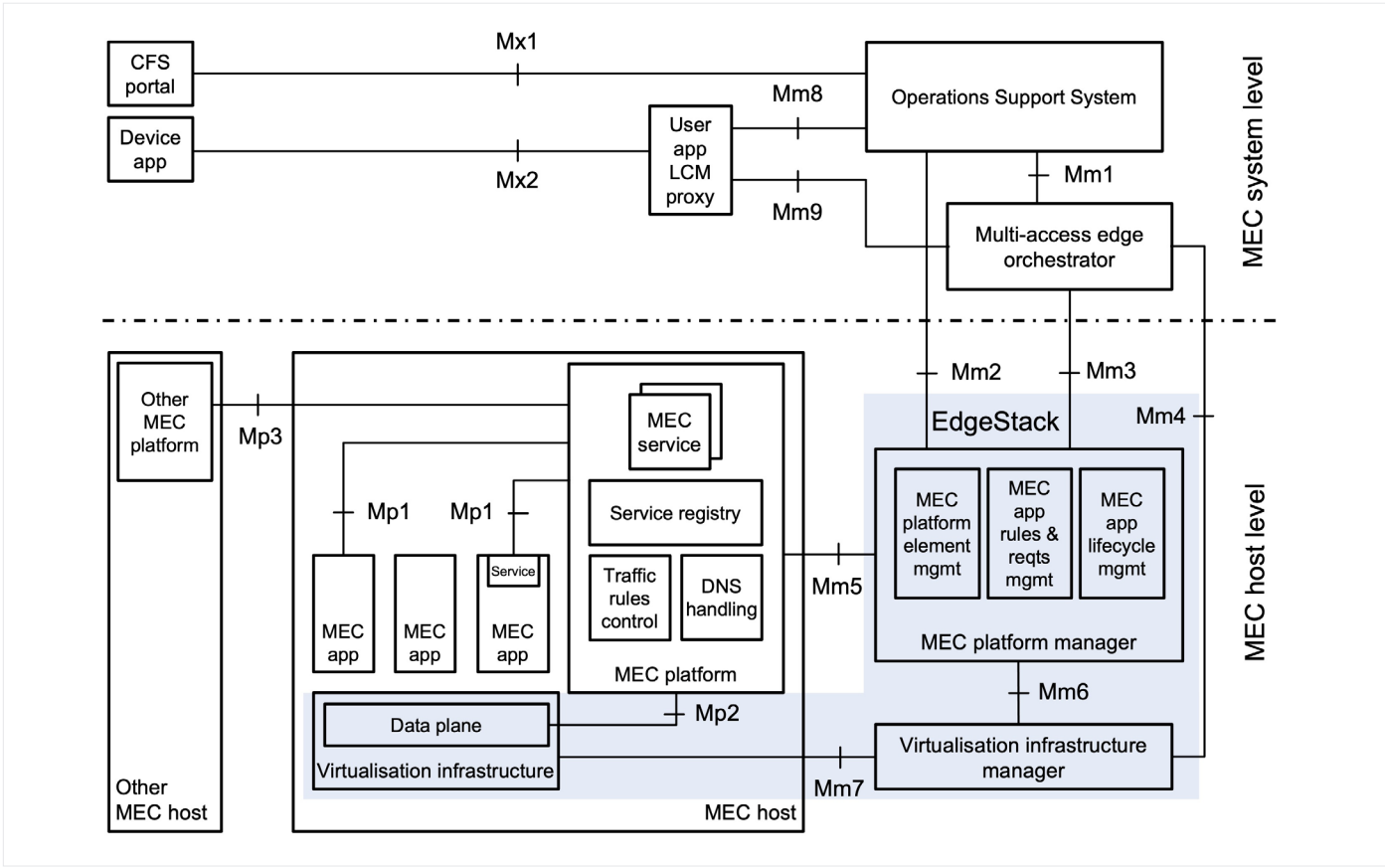

Figure 1. ETSI MEC Reference Architecture and SKT EdgeStack

1.1 Overview of SKT 5G MEC 2.0

SKT 5G MEC2.0 is a cloud software stack for edge environment that includes SKT EdgeStack. SKT 5G MEC 2.0 features services that are distributed from multiple sites that are on a smaller scale than a central data center. SKT EdgeStack is designed with a lightweight cloud architecture, which provides compact control nodes. However, SKT EdgeStack supports large scale deployment and carrier grade high availability for enterprise level cloud services. SKT EdgeStack provides virtual machines (VM) and managed Kubernetes to support both NFV and container workloads. They are managed by a single Kubernetes technology that utilizes KubeVirt and cluster-API technologies, which require less maintenance. VM and container workloads are deployed in separate nodes called VM zone and container zone. The data plane of each virtual resource is processed in a different way. SKT EdgeStack provides ETSI MEC MM3 Interface as a north bound interface. SKT EdgeStack is following the ETSI MEC Reference Architecture. Figure 1 shows this architecture with SKT EdgeStack capabilities highlighted in the blue area. These include the MEC platform manager, virtualization infrastructure manager, and data plane in virtualization infrastructure.

SKT EdgeStack uses various open source software, with Kubernetes used as a base system framework. All the components are containerized and running as pods. VMs are provisioned as pods, to provide them with more flexibility in a lightweight way that is in accordance with how SKT uses KubeVirt technology. To provide managed Kubernetes, the Cluster API is used, which is a Cloud Native Computing Foundation (CNCF) standard API that creates and manages Kubernetes clusters. Cluster API is used for other third-party cloud solutions such as OpenShift and VMware, and they are integrated with SKT EdgeStack without additional development using the Cluster API. As the network data plane OVS is used, and SKT proprietary SDN controller based on ONOS framework is used to compute and push flow rules to control the VM and container network flows. Prometheus and Elastic Search are used together as the monitoring system, with a SKT proprietary dashboard system that leverages Grafana and Kibana (see Figures 1, 2).

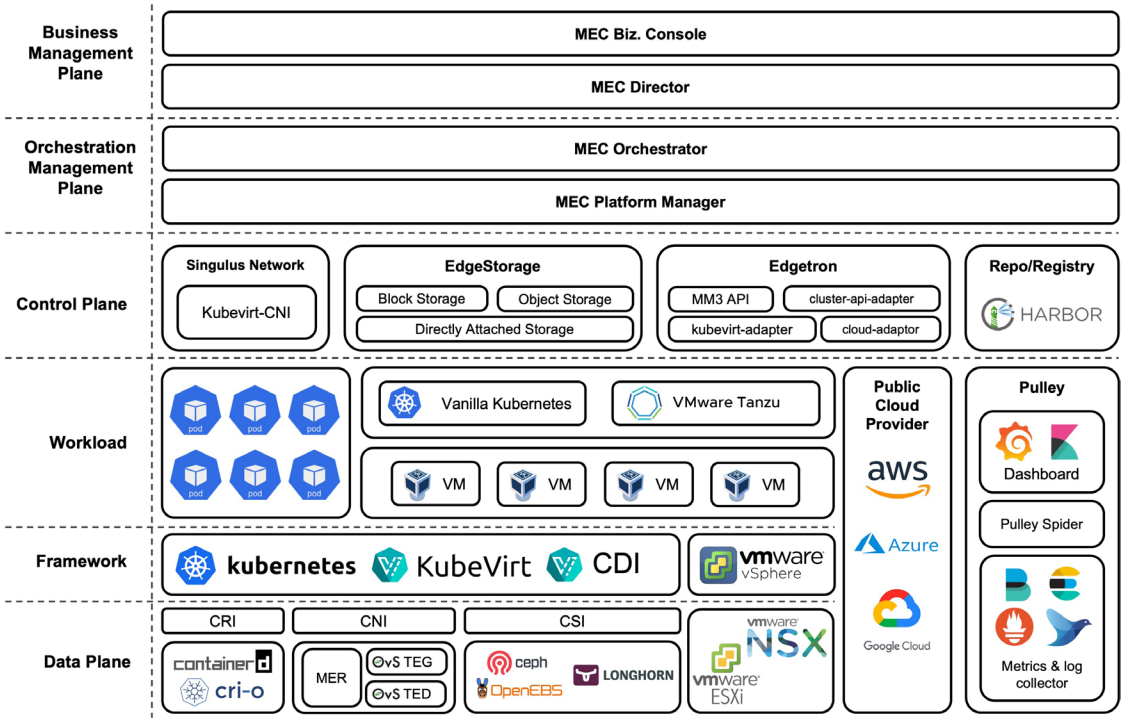

Figure 2. SKT 5G MEC 2.0 Architecture

SKT EdgeStack provides the following main features (See Table 1).

No	Feature	Description
1	Large scale VM deployment and life cycle management	Using KubeVirt technology, VMs are created and managed by pods much more elastically.
2	Large scale virtual resource monitoring	Up to 500 hosts and all of VMs and containers inside the hosts are monitored, and all of the metrics are provided through ElasticSearch DB.
3	High performance virtual overlay network	Nearly line rate speed is delivered even when overlay network using SDN technology is present.
4	High performance underlay network	Provides line-rate network speed using Single Root IO Virtualization (SR-IOV) technology.
5	Shared block storage	All the VM and container data is stored securely in three node clustered storage service.
6	Various guest OS support including Windows OS	Support various Linux distribution including Ubuntu, CentOS, Fedora and Rocky. Also support to provision Windows OS as a guest OS.
7	Integrated system log management	All systems logs are collected and stored in a central place.
8	MEC application HA/load balance as a service	Load balance as a service is provided and can be used for MEC application high availability.
9	Distributed firewall (security group)	VMs are protected by security group service.
10	Scalable lightweight edge gateway	MEC service is accessed via edge gateway from external network, even without deploying additional nodes.
11	Direct attached storage	Disk volume can be created and managed separately from VMs.
12	Public cloud integration	AWS EC2 can be created and managed from SKT EdgeStack.
13	Managed Kubernetes	Kubernetes cluster is easily provided on VM and managed by SKT EdgeStack.
14	High performance container network	Container network performance has been tested to be similar to Kubernetes on bare metal.
15	VMware TKG integration	VMware TKG cluster can be created and managed from SKT EdgeStack.

Table 1. SKT 5G MEC 2.0 Features

1.2 Key Technology

1.2.1 Intel Reference Software for Dynamic Power Control

This software provides the ability to minimize application power using CPU core frequency (P-state) management with the help of an AI model for applications that are running on edge clusters. It controls in near real-time the frequency of all the cores in a cluster worker node on which the applications of interest are running. The objective is to use the least amount of energy to process packets under given loss rate condition. The software includes:

- Telemetry plugin: Delivers system and application metrics (traffic rate, packet drop, core frequency)
- Frequency predictor: Dynamically predicts core frequency based on traffic volume while keeping zero packet drop with AI model (reinforcement learning)
- P-state controller sets target core frequency

1.2.2 Intel Reference Software Components for Edge Energy Savings

This software is a set of cloud native components that can be installed in a running MEC stack based on Kubernetes resulting in power savings at both the application level and the cluster level. The current list of components are as follows:

- Intel AI-Estimator and Exporter – Used for measuring application/container level power measurement.
- Intel AI-P-state Controller, Node Frequency Manager – Used in conjunction with a given application for operating in optimal power.
- Intel Node Auto Scalar, Edge Provider – Used for cluster level power savings.

Another way edge network deployments are different from cloud or data center deployment is that remote edge locations can have limits on available power. Total available power and availability of green energy are important considerations. Optimizing the available power is important because it impacts the battery life of green power sources or reduces energy expenses. The rest results listed in this paper will show that Intel's reference software components for edge energy savings, when deployed in an edge cluster, are engineered to provide 20%-30% daily power savings for the operation of the cluster (details are in sections 3.2 and 3.3).

2. Test Setup

2.1 Device Under Test Configuration

Figures 3 and 4 show the structure of the system architecture used in the tests as well as the data plane traffic flow. The overall hardware and software composition of the DUT and test application configuration is shown in Table 2, Listing 1. The system was configured to use application power measurement and CPU P-states control in conjunction with the SKT EdgeStack. To measure microservice application power measurement and node level power saving, Intel reference software for edge energy savings is installed on each controller and worker node as below (see Figure 3) and each component includes:

- AI-Estimator: Provides real time application power metering.
- AI-P-state Controller: Provides core frequency change per application.
- Edge Provider: Node insertion and deletion from K8S cluster.
- Node Auto-Scaler: Monitors and controls the node for longer C6 (low power state) residence.

This configuration can then provide:

- Power measurement of microservice based VM or PoD.
- AI backed P-state control per application.
- Scheduling enhancement for K8S node level to take advantage of C-state benefits.
- Closed loop management based on power objectives.

To see more details about the test application (VM) under test system configuration see Addendum 1.

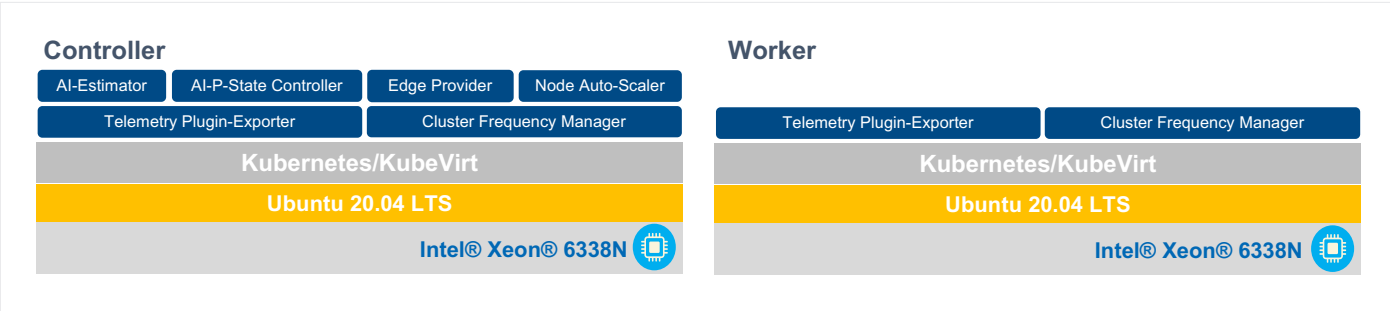

Figure 3. Test Systems Architectures

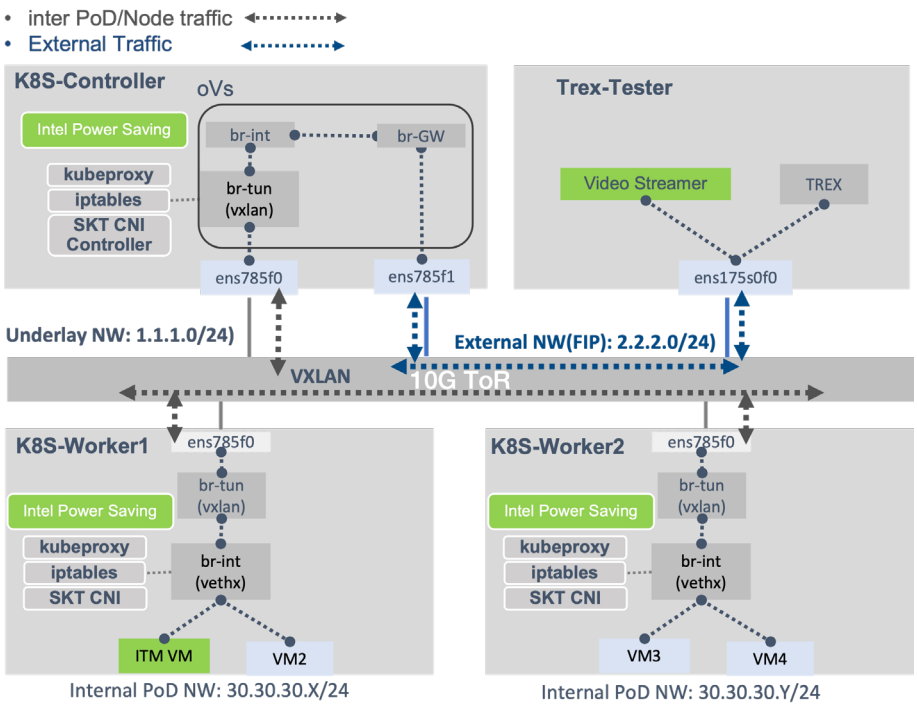

Figure 4. Data Plane Traffic Flow Under Test System

Category		Description
Processor	Product	Intel® Xeon® Gold 6338N processors
	Frequency	2.2 to 3.5GHz
	Cores per processor	32
Memory	DIMM slots per processor	8 channels per processor
	Capacity	512 GB DRAM (32 GB x16 DIMM)
	Memory speed	2666 MHz (MT/s), DDR4
	SGX EPC	64 GB per socket
Network	Intel NIC	1x Intel® Ethernet Controller 10G X550T
Storage	Intel	1x Intel S4610 960 GB SSD INTEL_SSDSC2KG96
2RU Server	Vendor	Intel M50CYP WHITLEY
Host OS	Vendor/version	Ubuntu 20.04.2 LTS 5.4.0-126-generic
BIOS	Vendor/version	Intel Corporation Version: SE5C6200.86B.0022.D64.2105220049 Release Date: 05/22/2021
microcode	Vendor/version	Intel Corporation microcode: 0xd000363
Kubernetes	Vendor/version	Opensource, v1.23.7 (client and server)
kubevirt	Vendor/version	Opensource, v0.54.0
SKT EdgeStack	Vendor/version	V2.6
Other SW	Open Source	Intelligent Traffic Management (ITM), vflow 0.9.0, wrk
Tested By		Intel
Test Date		October 2022

Table 2. Device Under Test Configuration

2.1.1 Application Power Measurement

Figure 5 shows logical structure of application power measurement inside a typical cloud native Kubernetes cluster; it has two important components:

- **Exporter:** A telemetry plugin on worker node: running as a daemon-set, responsible to collect application and system level metrics (such as cycles, uncore/core freq etc.) then, creates Prometheus registry and pushes all the metrics there. Exporter metric is cgroup_usage.
- **AI-Estimator:** Power Prediction Application with AI Inference, runs on the controller node, pulls data from Prometheus. Estimator uses collected system and application-level metrics and feeds those data to deep learning model which runs inside the estimator. The model predicts power, and the estimator creates its own Prometheus registry to push the predicted power back to Prometheus. Estimator metric is predicted_power.

Apart from the Estimator and Exporter components, there is one more offline component called Analyzer, which is a part of the AI training phase. The Analyzer component runs in servers powered by either Intel® Xeon® Scalable processors / Intel® Xeon® D processors. In these tests, the aim of using Analyzer is to collect data for each SKU from a CPU generation and train for the specific SKU for accurate power prediction. There were three main phases:

- Data preparation: Different types of applications were considered for model training with different workload scenarios.
- Model training: Requires two inputs: input labeled data and expected output for training.
- One data source is data that was prepared through the analyzer and training test scenarios. The SKT EdgeStack test scenarios also provide per application power as expected output for use in model training. To derive per application power, system power consumption levels were calculated when no applications were running. Next, one instance of an application was deployed, and system power was measured. The difference provides per application power that is used for training. System and application-level metrics were collected for both scenario (before and after application deployment) to find co-relation between metrics and power.
- Model Evaluation: On unseen data, model has achieved error < 5 watts mean absolute error.

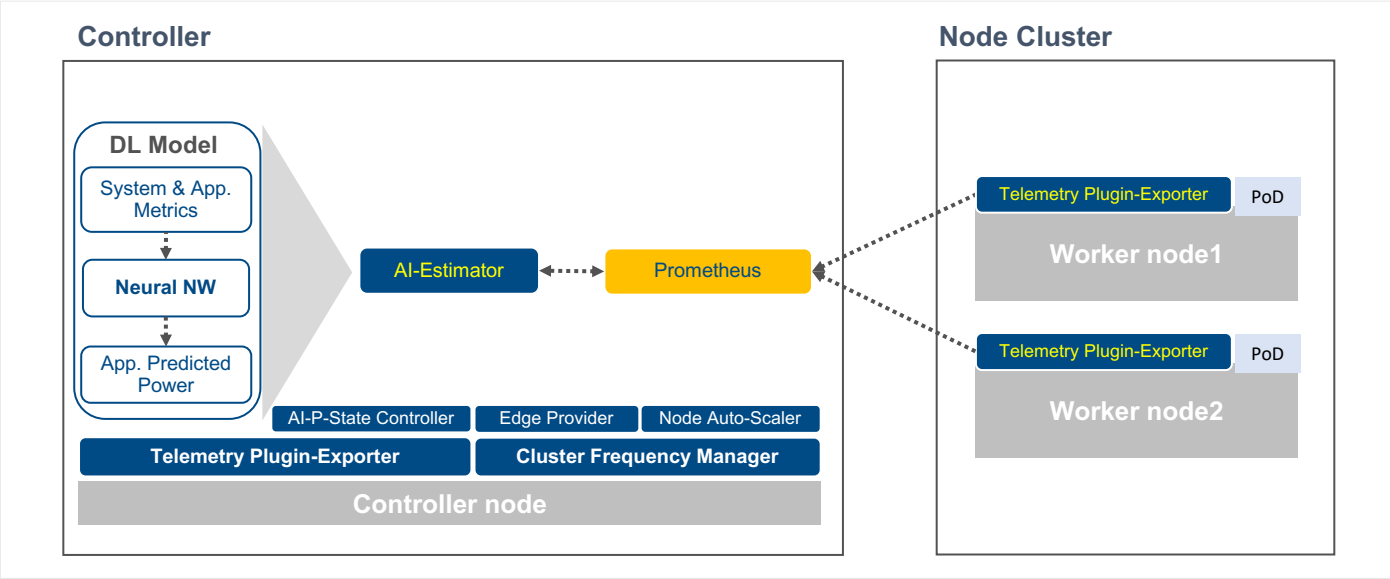

Figure 5. Application Power Measurement High-Level Architecture

2.1.2 Application Power Saving Through P-state Control

Application power minimization by core frequency (P-state) management using an AI model for applications that are running on Kubernetes clusters. The AI model controls in near real-time, frequency of all the cores on which the applications of interest are running in a cluster worker node. The objective is to use the least amount of energy to process packets under given loss rate condition. Figure 6 shows the application power saving through P-state control setup. The application frequency control has four important components:

- **Exporter** (Telegraf Daemon-set) – Exporter is a telemetry plugin that is responsible for collecting application and system metrics (such as application traffic rate, dropped packets, uncore/core freq etc.). Exporter creates a Prometheus registry and pushes all the metrics into Prometheus, which stores the data in a database. The exporter metric is application traffic, packet drop, core frequency, socket power, and platform power.

- **P-state Controller** (nfm_Server) – This component is on a worker node to control CPU P-states. It uses the user space governor for the core frequency control. It listens for requests from the frequency predictor to change core frequency. Once the request is received to set a particular frequency, it uses a Linux file system-based interface to set desired core frequency.
- **Frequency Predictor** (inference by ReInforce Model) – The Frequency Predictor uses data from Prometheus, which runs on the control-plane node and uses collected system and application-level metrics which are processed by the deep learning model running inside the predictor. The model predicts the lowest core frequency for which zero loss traffic rate can be maintained. It sends information about required frequency to P-state control component on the worker node.
- **Grafana Dashboard** – The application core frequency is displayed on this dashboard panel using Prometheus query language (promQL).

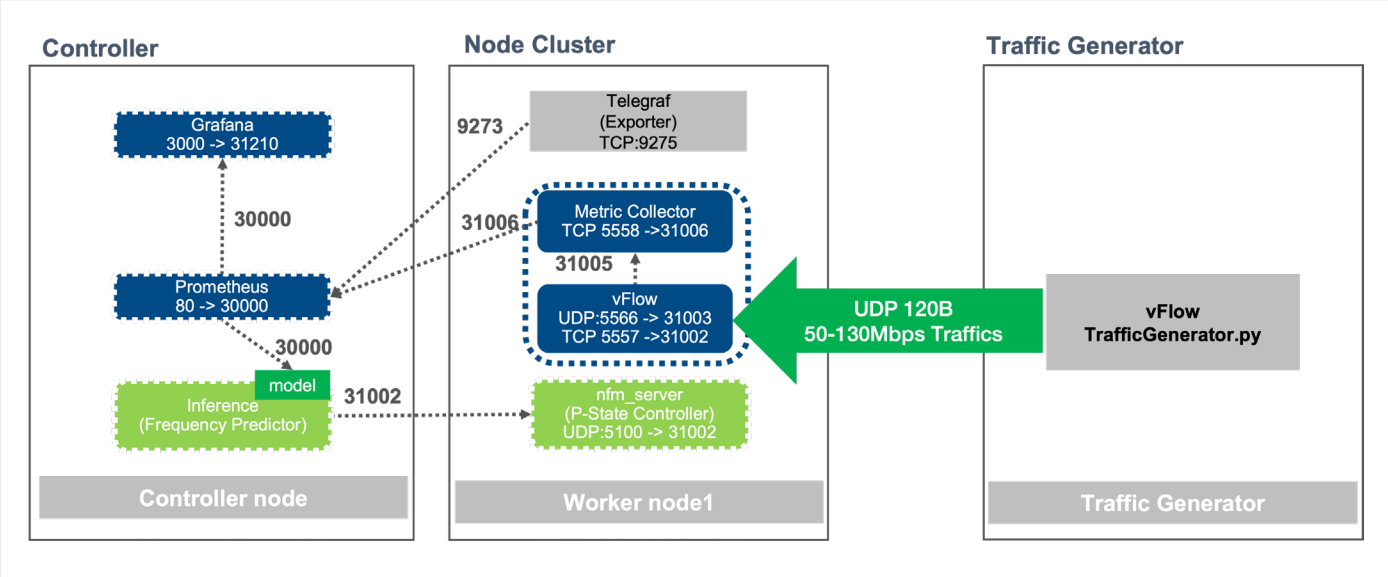

Figure 6. Application Power Saving Through P-state Control

Apart from the component discussed above, there is one more offline component called AI analytics training framework that is needed for the training phase. The training component shown in Figure 7 can run on any server or control-plane node. It also requires the worker node to collect telemetry data required for training. The training phase has three main phases.

- **Data Preparation** - In this phase, the required training data is prepared. Core frequency, application traffic rate, and packet drop rate are required. These data are received from the worker node and stored in the database for the neural network to use for training.
- **Model Training** - Model training requires generating traffic with different rates, changing frequency, and recording packet loss. We generate traffic and send it to worker node where system and application metrics are captured and sent to the database. Traffic generation starts at between 36Kpps to 91Kpps and the cycle repeats. The model adjusts core frequencies and learns with traffic rate, packet drop rate, and frequency value. The cycle continues for as many iterations as defined in the configuration file.
- **Model Evaluation** - At the end of the training cycles specified in the configuration file, the recorded training data is available for examination. Initially there will be packets dropped as the model is learning, but toward the end of the learning cycle there should not be any packet drops for all possible traffic rates. If there is no packet drop for the whole cycle, the model is considered trained and ready for deployment.

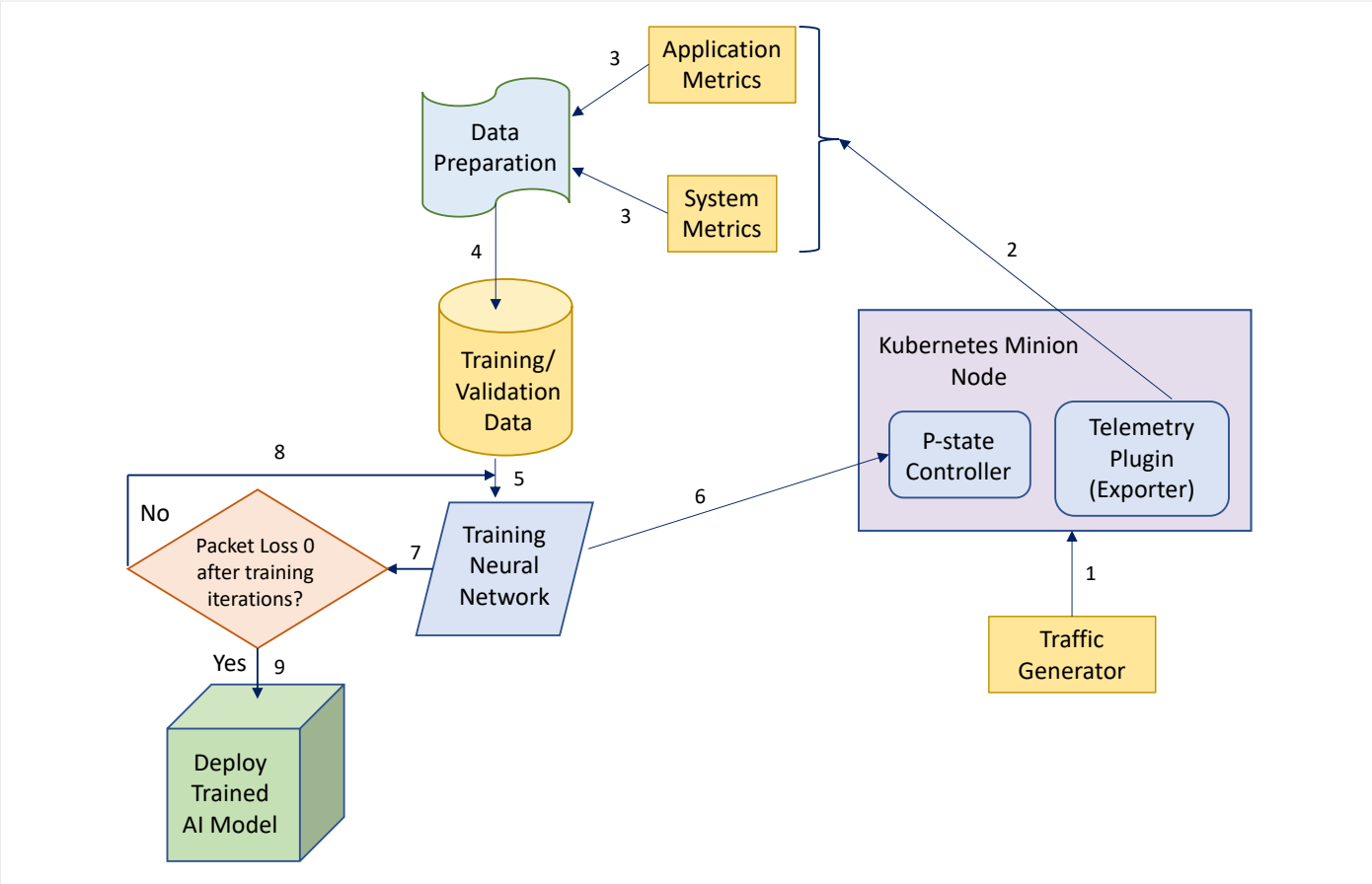

Figure 7. Model Training Flow, from Data Preparation, Training, and Deployment Ready Model

2.1.3 Cluster Power Savings Through Adaptive Node Scaler

One set of tests measured cluster-level node power savings using an adaptive node scaler approach in combination with edge provider green edge building blocks. Adaptive node scaler saves power by using appropriate policies that enable the Kubernetes scheduler to keep cores in a node in C6 states for a longer time by de-scheduling and not-scheduling new pods (see Figure 8).

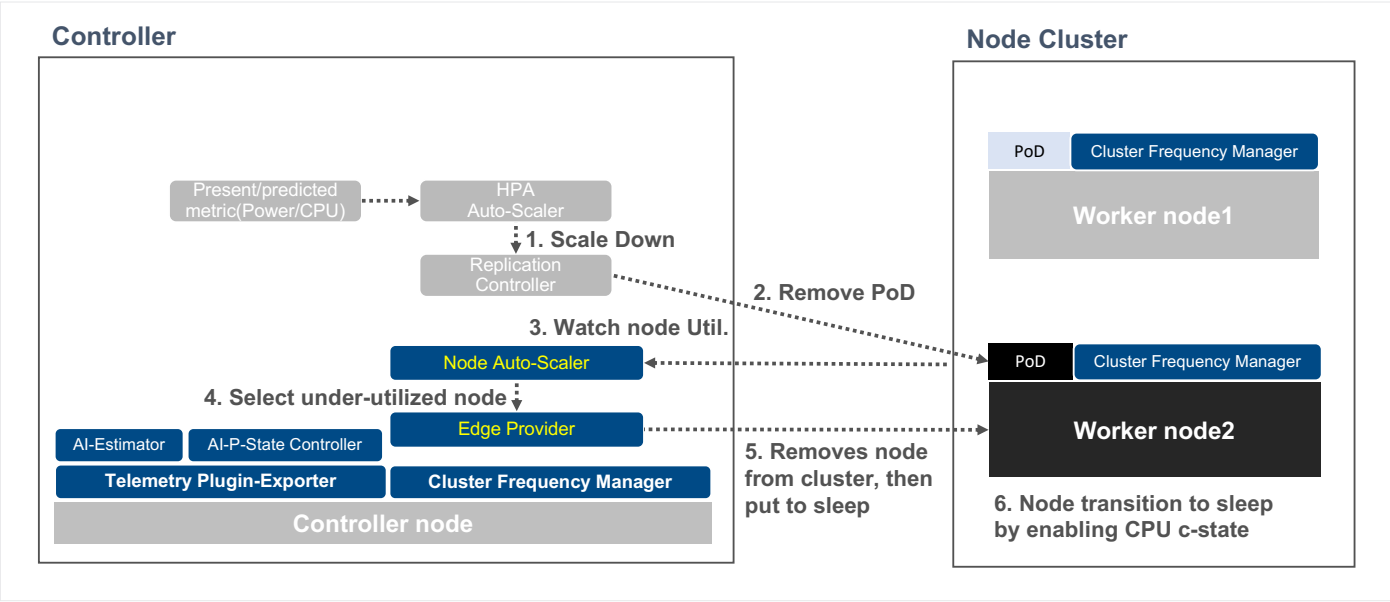

Figure 8. Node Auto-Scaler High-Level Architecture

The components of the auto-scaler tests include:

- **Edge provider:** Mainly responsible for bringing in and taking out a node from cluster as per direction from node auto-scaler.
- **Node auto-scaler:** Monitors the metrics and upon matching a policy it will try to free up a node or instructs the edge provider to add a node back into the cluster. It can migrate the workloads, if possible, to remove the least loaded nodes from the cluster. This enabled the workloads to migrate across nodes and help in an effective scaling out of the nodes from the cluster. The following declarative intents can prevent the pods from migrating:
  - Local storage (hostpath etc.).
  - Node selector.
  - Explicit pod specification: Pods that are not backed by a controller object.
  - Kube-system pods that are not run on the node by default.
  - Pods with restrictive PodDisruptionBudget.
- **Visualization through Grafana Dashboard:** Telegraf can be run as a system daemon to collect platform power telemetry from all nodes including those that are scaled out by the auto-scaler. Application power is added to the Grafana dashboard panel using Prometheus query language (promQL).
- **Kubernetes scheduler configuration:** The kube-scheduler configuration has been modified to use the "MostAllocated" scoring strategy. This enables the pods to be packed in the available nodes and thereby other nodes can be kept in low power state as much as possible (see Addendum 2).
- **K8S deployment:** The node auto-scaler is deployed using helm at local directory. The Edge provider can be run as a privileged pod or as a daemon. In the test setup it is run as a daemon on the control plane node.

To see the auto-scalar and Kubernetes configuration, please see Addendum 2.

2.2 Test Procedure

2.2.1 Application Power Measurement

To demonstrate the application power consumption measurement, the Intel Intelligent Traffic Management (ITM) sample application was deployed. This application takes live stream coming from traffic cameras and predicts the total number of vehicles passing the camera as well as how many of those vehicles would become involved in a collision. ITM was converted to a docker compose-based deployment inside a VM running on the worker node of the cluster.

Traffic was initiated from the video streamer capability in traffic generation server (see Figure 4) which is outside the MEC cluster using the Real Time Streaming Protocol (RTSP) to stream video that was then fed to the Analytics component running inside ITM. The data then went to the Measurement function and was visualized through Grafana dashboard. The application power consumption was displayed in Grafana dashboard panel using promQL. To access the stream feed, we used url 'rtsp://<trex_ip>:8554', "stream fps ~ 32 fps (resolution 1920x1080)." The ITM Analytic component performed per frame car detection, collision detection, pedestrian detection and social distancing between pedestrians. To see how to program the power measurement see the code listed in Addendum 3.

2.2.2 Application Power Saving Through P-state Control

To demonstrate the power reduction capability of P-state control, Intel set up a test that deployed the open source vFlow application on the worker node as shown in Figure 6. This application decodes NetFlow data from a stream of UDP packets that are generated by the vFlow traffic generator. The tests included running four scenarios of frequency management to compare the efficiency of the solution. Following that, measurements were taken of power consumption in a variety of scenarios: fixed frequency, power under OS-based frequency management with turbo frequency enabled and disabled, and lastly frequency controlled by AI predictor.

- **OS-based Application Power Management**
  Figure 6 shows test setup that was used when application frequency was controlled by OS or kept fixed. There are three systems used for the measurement, except inference and nfm_server that is only used for AI-based application power control tests. On the right is a traffic generator that generates 120Byte UDP packets at a 50 Mbps to 130 Mbps transmission rate. The packets are sent to the vFlow to consume. The second system is a vFlow pod running the application that computes NetFlow stats as well as a metrics collector that collects and sends application metrics to Prometheus database on the worker node. System metrics are also sent to the Prometheus database as part of edge building block software from the Telegraf PoD. The third system is a controller (control-plane) node that hosts the Prometheus database and Grafana dashboard for visualization.
- **AI-based Application Power Management**
  Compared to above OS-based Application Power Management, the AI-based frequency predictor (Inference) was used on the controller node to predict frequency based on application and system metrics. An additional pod called nfm_server which is a user-space P-state controller also ran on the worker node. The nfm_server pod receives instructions from inference pod on what core frequency to run at, and then it sets that frequency.

For each case, power was measured with respective configuration to control P-state and Turbo frequency.

a. Fixed P1 & Turbo-Off for base line:

```
echo userspace > /sys/devices/system/cpu/cpu<4,5,68,69>/cpufreq/scaling_governor
echo 2200000 > /sys/devices/system/cpu/cpu<4,5,68,69>/cpufreq/scaling_setspeed
```

b. OS based P-state enabled/Turbo-Off:

```
echo ondemand > /sys/devices/system/cpu/cpu<4,5,68,69>/cpufreq/scaling_governor
echo 2200000 > /sys/devices/system/cpu/cpu<4/5/68/69>/cpufreq/scaling_max_freq
```

c. OS based P-state enabled/Turbo-On:

```
echo ondemand > /sys/devices/system/cpu/cpu<4,5,68,69>/cpufreq/scaling_governor
echo 2201000 /sys/devices/system/cpu/cpu<4/5/68/69>/cpufreq/scaling_max_freq
```

d. AI based P-state enabled/Turbo-Off:

```
echo userspace > /sys/devices/system/cpu/cpu<4,5,68,69>/cpufreq/scaling_governor
echo 2200000 > /sys/devices/system/cpu/cpu<4/5/68/69>/cpufreq/scaling_max_freq
```

2.2.3 Cluster Power Savings Through Adaptive Node Scaler

Node scaling has been tested with edge provider using both kubeadm and kubespray. Kubespray takes a longer time to add or delete a node compared to the kubeadm. When removing the node, in addition to remove the node from the cluster, it also deletes the downloaded images, removes the related system software and configuration. The kubespray can be further improved with additional flags so that the node addition and removal time can be reduced, particularly for use by the cluster node auto-scaler (with warm start features).

We deployed the modified sample application to test the native K8S horizontal pod autoscale (HPA). https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

- Native K8S CPU manager was enabled by manual at each worker node.
- php test pod, deployment service was instantiated, the configuration included
- Patch the nodes with provider ID, as the node auto-scaler handles nodes with this ID set,
- Start the edge-provider as below,
- ./edge_server &> ./server_log &
- Used a local docker registry running on the control plane node to store the node auto-scaler image, and helm to deploy the node auto-scaler (see Addendum 3)
- The telegraf (or any node exporter) does read the RAPL (/sys/devices/virtual/powercap/intel-rapl/*/energy_uj and converts to watts).

The Kubernetes scheduler and pod configuration can be found in Addendum 4. To increase the node CPU load, use the wrk tool, by sending the query request over multiple threads and connections to the exposed php service. When the load reduces, the HPA scales down the number of workload instances (replicas). Then wrk can be run as below, to generate requests from multiple threads to php-service IP: port wrk -t 12 -d 1200 -c 20 http:// 10.233.43.4:80

3. Test Results

3.1 Application Power Measurement

As per Figure 9, before the RTSP stream, the VM was consuming ~8 watts of power. but after RTSP stream feed, because of Analytics component, it started consuming ~17watts of power as per Figure 10.

The inference accuracy can be verified by deploying the same application to fresh system and measuring power consumption before and after deployment using RAPL library. RAPL library reports total package power, so one can subtract after and before to calculate application power.

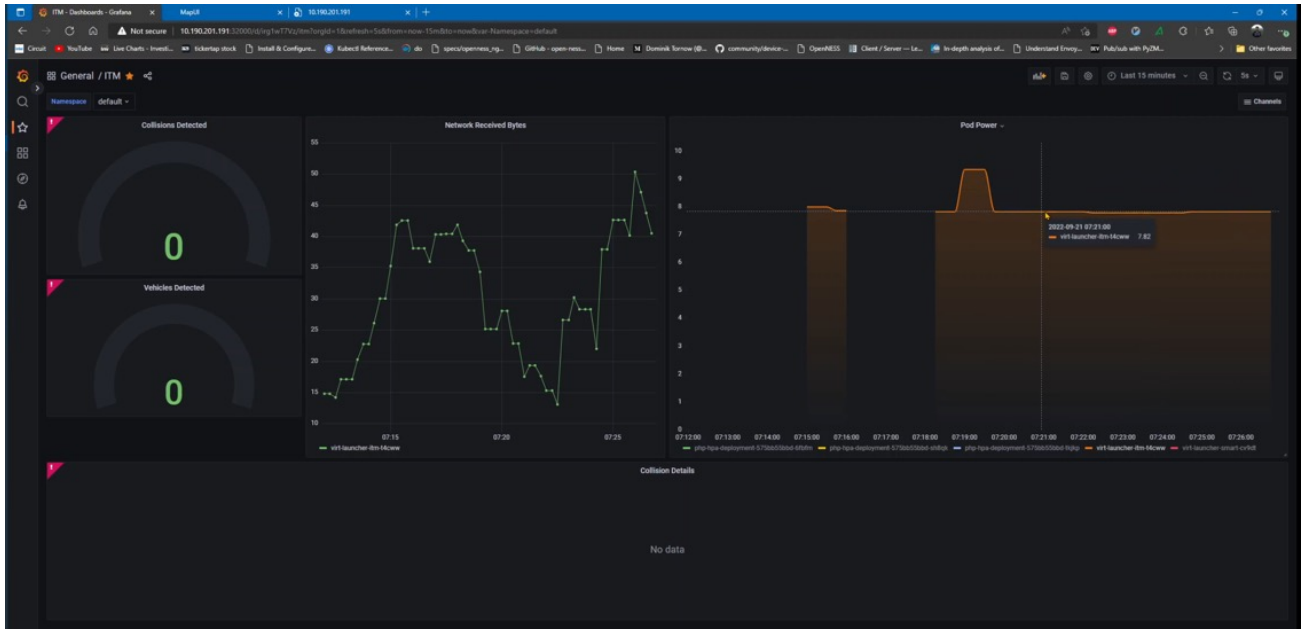

Figure 9. Power Consumption of ITM VM with No Traffic

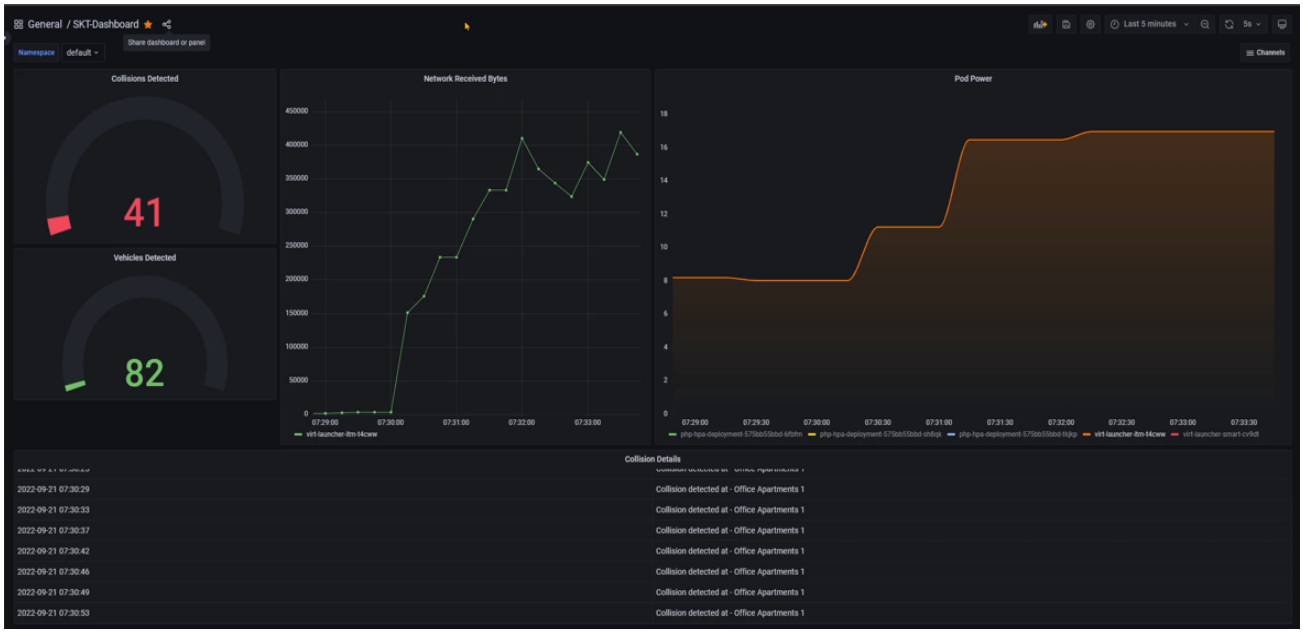

Figure 10. Power Consumption of ITM VM After RTSP Stream Coming from External Traffic Generator

Figure 11 shows external camera stream that is created by ITM Analytics component running inside cluster for vehicle and collision detection.

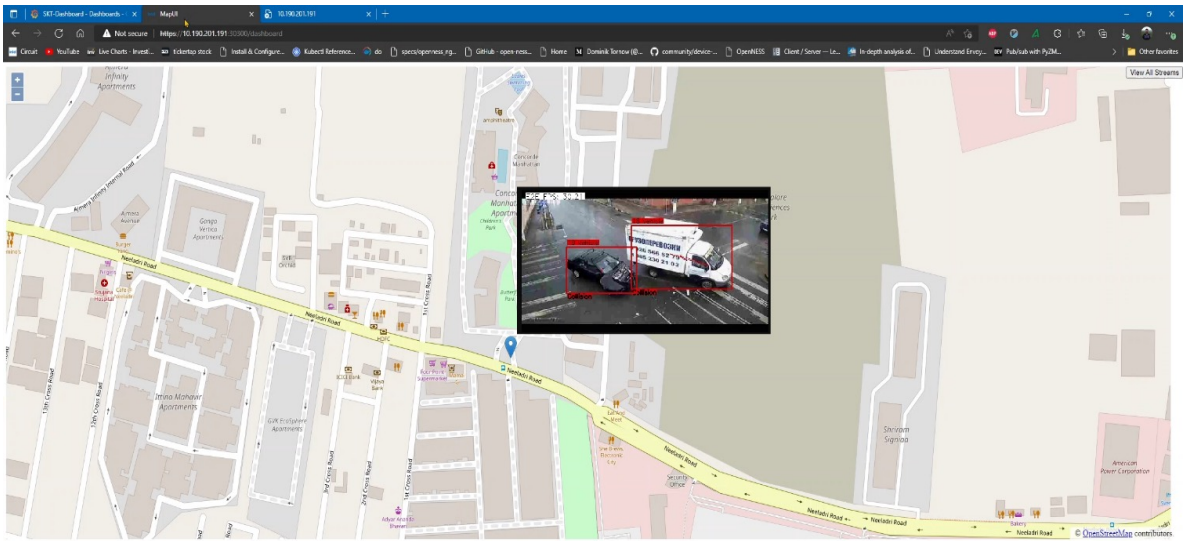

Figure 11. Map UI of Live Camera (Blue Point in Map)[1]

3.2 Application Power Saving Through P-state Control

Figure 12 shows results when application power was controlled by OS-based P-state enabled/Turbo disabled cases. Top right chart shows packet dropped which is zero and bottom right chart shows CPU power. We can see that core frequency and package power follows traffic pattern.

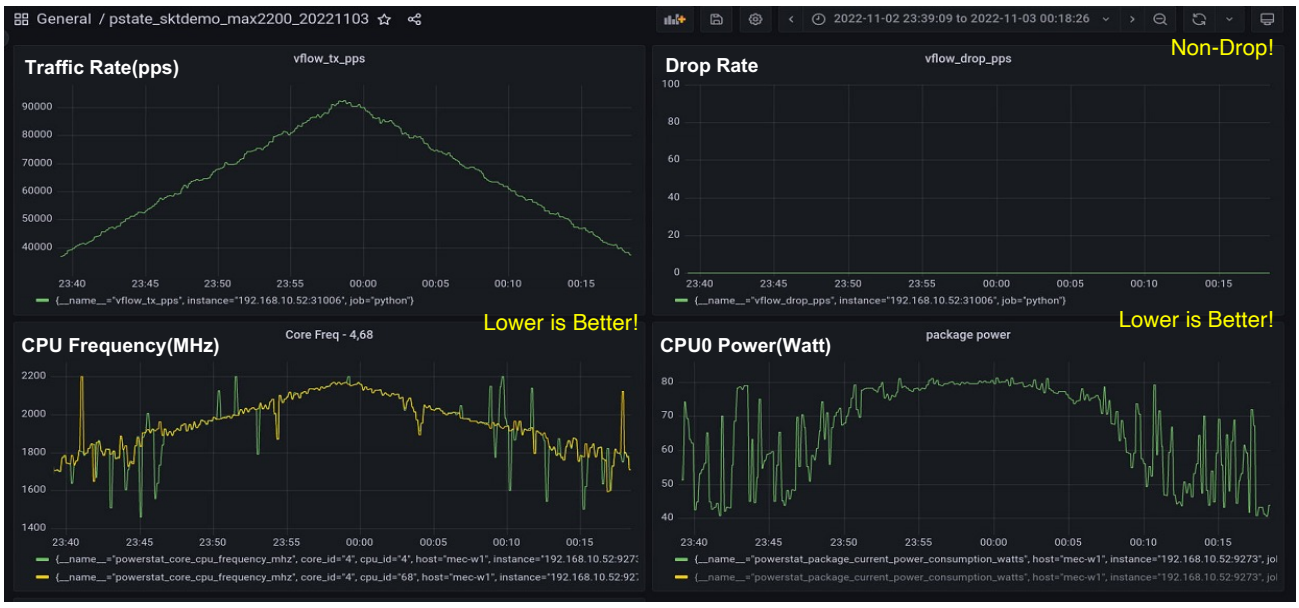

Figure 12. OS based Application Power Management Results (P-state enabled/Turbo-Off)

Figure 13 shows output when AI-based application power management was run. See that core frequency and package power follows traffic pattern as well as OS-based power management cases and see that with AI-based frequency prediction, when traffic is slow, core frequency is lower than OS selected frequency also see that frequency change is more gradual in AI-based frequency management. On the bottom right chart, which shows package power, we see more time spent at lower package power compared to OS control-based package power.

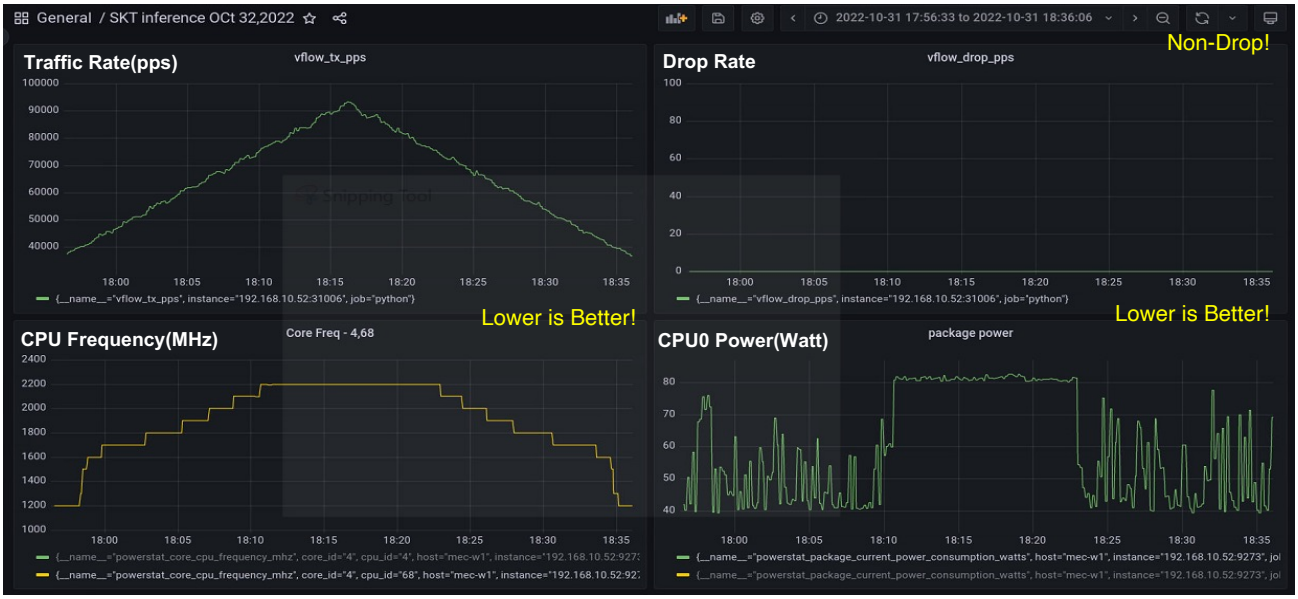

Figure 13. AI based Application Power Management Results

Table 3 shows average core frequency and CPU power for four different configurations to compare them and can see in column "AI based P-state" that it has the lowest average core frequency and CPU power compared to all other options. It saves approximately 25% of the power by using AI-based application power management compared to base line configuration (fixed P1 frequency/turbo-off) and 7% power compared to OS-based power management with turbo disabled. With the AI-trained model, inference picks the minimal frequency required for the given Tx Rate with 0 packet drop. On the other hand, OS-based P-state management not always the minimal required one.

Parameter	AI based P-state /turbo-Off	OS based P-state /turbo-On	OS based P-state /turbo-Off	Fixed P1 Base /turbo-Off
Avg. core Freq- core 4 (MHz)	1877.13	2208.28	1939.22	2194.90
Avg. core Freq- core 68 (MHz)	1888.52	2237.28	1943.97	2195.11
Avg. core Freq- core 5 (MHz)	1877.51	2199.36	1938.53	2195.22
Avg. core Freq- core 69 (MHz)	1876.93	2221.37	1940.31	2195.17
Avg. CPU Power-CPU0 (W)	59.72	63.81	64.33	79.86

Table 3. Application Power Management Results

3.3 Cluster/Node Level Power Savings Through Adaptive Auto Scaler

The node auto-scaler requests the edge-provider to remove the mec-w2 from the cluster. With mec-w2 out of the cluster, as workloads are not running in the mec-w2 node, it can go into the C6 power saving mode. Figure 14 shows the power consumption output for the three nodes in the cluster. We see a savings of ~110W, when the mec-w2 is removed from the cluster (see Addendum 5).

Figure 14. Power Consumption of the Node (mec-w2) when it is scaled-in and scaled-out

4. Summary

Mobile networks have periods of low traffic which provide an opportunity for power savings and energy efficiency improvement. With more advancement in technology for sustainability and green energy, operators can reduce energy consumption by using infrastructure that is designed to require less energy by matching power saving techniques to different workload requirements. These test results present a reference architecture of a microservice-based power saving edge to validate the functionality of Intel reference software for edge energy savings on SKT EdgeStack for commercial services, leveraging Intel 3rd Generation Xeon Scalable processor's enhanced power saving features with network AI prediction. The tests described in this paper, based on the SUT configuration in Table 2, include:

- Demonstrated microservice-based application power monitoring and full functionality of Intel reference software for edge energy savings on the SKT 5G EdgeStack.
- Proven AI-based P-state control, up to 25% average saving for PoD under busy-hour with zero packet loss in comparing with default base configuration (fixed P1 with turbo-off). The inference picks the minimal frequency for given traffic levels, so it can be an effective power control solution especially for user plane micro services application running 24 hours every day on MEC.
- Proven cluster level power savings features by enabling CPU power management (C6 power saving mode) at node, whenever an appropriate utilization (CPU/memory/IO, etc.) policy is met.

The above results indicate that TEMs and ISVs can utilize power saving edge functionalities to help and meet the requirements for less carbon emissions, and energy consumption on microservice-based workloads on open edge computing services.

Mobile operators can leverage this power saving functionality to also meet government regulation and customer demand of new energy saving requirement with growth of managed B2B/B2B2C sales revenues.

5. Resources

3rd Generation Intel® Xeon® Scalable processors

K8S HPA

Wireless Network-Ready Intelligent Traffic Management Reference Implementation

Linux Kernel 5.4 Power Capping Framework

VFlow: PFIX, sFlow and Netflow collector

Intel® Xeon® P-states

6. Glossary of Terms

Term	Description
K8S	Kubernetes
NFV	Network Function Virtualization
NFVi	Network Function Virtualization Infrastructure
MEC	Mobile Edge Computing
TEM	Telecom Equipment Manufacturer
ISV	Independent Software Vendor
ITM	Intelligent Traffic Management
HPA	Horizontal Pod Autoscaler
VM	Virtual Machine
PoD	A grouping of one or more K8S containers

Addendum 1 - Test Application (VM) Under Test System

```
intel@mec-m:~$ edgectl list-networks
+----------+-------+----------+-------------+------+-----------+---------+----------------------+
|   NAME   | TYPE | SEGMENTID |     CIDR    | MTU  | GATEWAY IP | EXTERNAL |         AGE         |
+----------+-------+----------+-------------+------+-----------+---------+----------------------+
| external | FLAT |    N/A    |  2.2.2.0/24  | 1500 |  2.2.2.1  |   True   | 1 month and 26 days ago |
+----------+-------+----------+-------------+------+-----------+---------+----------------------+
| internet1 | VXLAN |   1010   | 30.30.30.0/24 | 1500 | 30.30.30.1 |  False   | 1 month and 26 days ago |
+----------+-------+----------+-------------+------+-----------+---------+----------------------+
intel@mec-m:~$ edgectl list-routers
+--------+-------+-----------------+----------+----------+-----------+---------------+----------------------+
| NAME  | SNAT |       MAC       | INTERNAL | EXTERNAL | VROUTER IP | PEER ROUTER IP |         AGE         |
+--------+-------+-----------------+----------+----------+-----------+---------------+----------------------+
| router | False | 52:54:00:02:32:5c | internet1 | external |  2.2.2.2  |    2.2.2.1    | 1 month and 26 days ago |
+--------+-------+-----------------+----------+----------+-----------+---------------+----------------------+
intel@mec-m:~$ edgectl list-sgs
+--------------------------------------+-------------+--------------------------+----------------------+
|                  ID                  |    NAME    |       DESCRIPTION       |         AGE         |
+--------------------------------------+-------------+--------------------------+----------------------+
| c5071a6b-d606-4fde-86c1-6bcc3160bb28 |   default   |  Allow all egress traffic | 1 month and 26 days ago |
+--------------------------------------+-------------+--------------------------+----------------------+
| eb99fdd0-560f-4f85-b3a4-3780a22f3a00 | ingress-all | Allow all ingress traffic | 1 month and 26 days ago |
+--------------------------------------+-------------+--------------------------+----------------------+
intel@mec-m:~$ edgectl list-vms
+----------+------------------+--------------------+-------+-----------+----------+--------------------------------+-------------+
|   NAME   |      IMAGE       |       STATE        | READY |  FLAVOR  | KEY PAIR |              IPS               |     AGE     |
+----------+------------------+--------------------+-------+-----------+----------+--------------------------------+-------------+
| powertel | ubuntu-2004-image | ErrorUnschedulable | False |  m1.large |   N/A    |     internet1: 30.30.30.198    |  6 days ago |
+----------+------------------+--------------------+-------+-----------+----------+--------------------------------+-------------+
|   nfm    | ubuntu-2004-image |      Running       |  True |  m1.large |   N/A    | k8s-pod-network: 10.233.103.196 | 13 days ago |
|          |                  |                    |       |           |          |     internet1: 30.30.30.25     |             |
|          |                  |                    |       |           |          |      floating: 2.2.2.169       |             |
+----------+------------------+--------------------+-------+-----------+----------+--------------------------------+-------------+
|   itm    | ubuntu-2004-image |      Running       |  True | m1.xlarge |   N/A    | k8s-pod-network: 10.233.103.176 | 15 days ago |
|          |                  |                    |       |           |          |     internet1: 30.30.30.221     |             |
|          |                  |                    |       |           |          |      floating: 2.2.2.116       |             |
+----------+------------------+--------------------+-------+-----------+----------+--------------------------------+-------------+
intel@mec-m:~$ k get vmi
NAME      AGE    PHASE       IP             NODENAME   READY
itm       7d23h  Running     10.233.103.176  mec-w1     True
nfm       5d17h  Running     10.233.103.196  mec-w1     True
powertel  5d17h  Scheduling                             False
```

Addendum 2 - Kubernetes Scheduler, Node Auto-Scaler Configuration

```
root@mec-m:~# cat /etc/kubernetes/cluster-autoscaler.yaml
apiVersion: kubescheduler.config.k8s.io/v1beta2
kind: KubeSchedulerConfiguration
leaderElection:
 leaderElect: true
clientConnection:
 kubeconfig: /etc/kubernetes/scheduler.conf
profiles:
- schedulerName: default-scheduler
 pluginConfig:
 - name: NodeResourcesFit
  args:
   scoringStrategy:
    resources:
    - name: cpu
     weight: 1
    - name: memory
     weight: 1
    type: "MostAllocated"

intel@mec-m:~$ ps ax | grep edge_server
1542459 pts/0   S+    0:00 grep --color=auto edge_server
3025827 ?     Sl   43:38 ./edge_server

intel@mec-m:~$ k get pod edge-ca-release-edge-cluster-autoscaler-ddbbf4fbf-8p9nr -n kube-system
NAME                                READY  STATUS   RESTARTS    AGE
edge-ca-release-edge-cluster-autoscaler-ddbbf4fbf-8p9nr  1/1    Running  2 (28d ago)  36d
```

Addendum 3 - Application Power Measurement Component

```
root@mec-m:~/aipower/helmFiles# gedge
NAME                    READY  STATUS        RESTARTS       AGE
pod/estimator-f66bdbb6-p76vw        1/1    Running       1 (28m ago)     3h14m
pod/exporter-6fq2k              1/1    Running       0           3h15m
pod/exporter-tcm7b              1/1    Running       0           3h15m
pod/grafana-5f66798fd6-2znb5        1/1    Running       0           12d
pod/prometheus-node-exporter-6v6mb     0/1    CrashLoopBackOff  1704 (5m4s ago)  6d1h
pod/prometheus-node-exporter-j7b6j     1/1    Running       0           6d1h
pod/prometheus-server-644dd7756d-258gw  2/2    Running       0           6d1h
pod/telegraf-5f28w              1/1    Running       0           26d
pod/telegraf-rhrnq              1/1    Running       2 (6d3h ago)    26d

NAME                  TYPE     CLUSTER-IP    EXTERNAL-IP  PORT(S)      AGE
service/estimator-service       ClusterIP  None        <none>     8000/TCP     3h14m
service/grafana             NodePort   10.233.19.110  <none>     3000:31210/TCP  12d
service/prometheus-node-exporter  ClusterIP  10.233.10.249  <none>     9100/TCP     6d1h
service/prometheus-server       NodePort   10.233.29.163  <none>     80:30418/TCP    6d1h

NAME                    DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE SELECTOR  AGE
daemonset.apps/exporter           2      2      2    2       2       <none>      3h15m
daemonset.apps/prometheus-node-exporter  2      2      1    2       1       <none>      6d1h
daemonset.apps/telegraf           2      2      2    2       2       <none>      26d

NAME                  READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/estimator       1/1    1       1       3h14m
deployment.apps/grafana         1/1    1       1       12d
deployment.apps/prometheus-server  1/1    1       1       6d1h

NAME                        DESIRED  CURRENT  READY  AGE
replicaset.apps/estimator-f66bdbb6        1      1      1    3h14m
replicaset.apps/grafana-5f66798fd6        1      1      1    12d
replicaset.apps/prometheus-server-644dd7756d  1      1      1    6d1h
```

Addendum 4 - Kubernetes Scheduler and PoD Configuration

```
root@mec-w1:~# cat /etc/kubernetes/kubelet.env
KUBE_LOGTOSTDERR="--logtostderr=true"
KUBE_LOG_LEVEL="--v=2"
KUBELET_ADDRESS="--node-ip=192.168.10.52"
KUBELET_HOSTNAME="--hostname-override=mec-w1"
KUBELET_ARGS="--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf \
--config=/etc/kubernetes/kubelet-config.yaml \
--kubeconfig=/etc/kubernetes/kubelet.conf \
--container-runtime=remote \
--container-runtime-endpoint=unix:///var/run/containerd/containerd.sock \
--runtime-cgroups=/systemd/system.slice \
--cpu-manager-policy=static \
  --housekeeping-interval=30s "
KUBELET_NETWORK_PLUGIN="--network-plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin-dir=/opt/cni/bin"
KUBELET_CLOUDPROVIDER=""
PATH=/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

intel@mec-m:~$ k get pod -o=wide
NAME                    READY  STATUS   RESTARTS  AGE   IP          NODE    NOMINATED NODE  READINESS GATES
php-hpa-deployment-575bb55bbd-qvbpd  1/1   Running  0      4d2h  10.233.103.185  mec-w1  <none>       <none>
php-hpa-deployment-575bb55bbd-sljnf  1/1   Running  0      4d6h  10.233.103.170  mec-w1  <none>       <none>
php-hpa-deployment-575bb55bbd-vbsnz  1/1   Running  0      4d2h  10.233.103.186  mec-w1  <none>       <none>native K8S
cpu manager was enabled by manual

intel@mec-m:~$ k get svc,pod,ep
NAME                TYPE     CLUSTER-IP    EXTERNAL-IP  PORT(S)              AGE
service/php-service       NodePort   10.233.43.4   <none>     80:30008/TCP            17d

NAME                    READY  STATUS   RESTARTS  AGE
pod/php-hpa-deployment-575bb55bbd-qvbpd  1/1   Running  0      4d2h
pod/php-hpa-deployment-575bb55bbd-sljnf  1/1   Running  0      4d6h
pod/php-hpa-deployment-575bb55bbd-vbsnz  1/1   Running  0      4d2h

NAME                ENDPOINTS                          AGE
endpoints/php-service       10.233.103.170:80,10.233.103.185:80,10.233.103.186:80  17d

intel@mec-m:~$ kubectl patch node <host-name> -p '{"spec":{"providerID":"edge://<host-name>" } }'
```

Addendum 5 - Kubernetes Clusters Node Status

```
intel@mec-m:~$ k get node -o=wide
NAME   STATUS  ROLES          AGE  VERSION  INTERNAL-IP   EXTERNAL-IP  OS-IMAGE        KERNEL-VERSION
CONTAINER-RUNTIME
mec-m  Ready   control-plane,master  55d  v1.23.7  192.168.10.51  <none>     Ubuntu 20.04.2 LTS  5.4.0-122-generic
containerd://1.6.4
mec-w1 Ready   worker         55d  v1.23.7  192.168.10.52  <none>     Ubuntu 20.04.2 LTS  5.4.0-126-generic
containerd://1.6.4
```

Table of Contents

0323/LV/H09/PDF ♻Please Recycle 354847-001US