PoC #14: Intent-based Cloud Management
Progress Update

PoC host contact/Rapporteur:

NTT: Chao Wu

PoC member contacts/Co-Rapporteurs:

Intel: Emma Collins, Haining Wang , Chris Cavigioli

Intracom Telecom: Nikos Anastopoulos

NTT-AT: Takaaki Tanaka

Acronyms

AMF	Access and Mobility Management Function
LLC	Last Level Cache
NFV	Network Function Virtualisation
NUMA	Non-Uniform Memory Access
SLO	Service Level Objective
SLA	Service Level Agreement
SMF	Session Management Function
UPF	User Plane Function
VDI	Virtual Desk Infrastructure

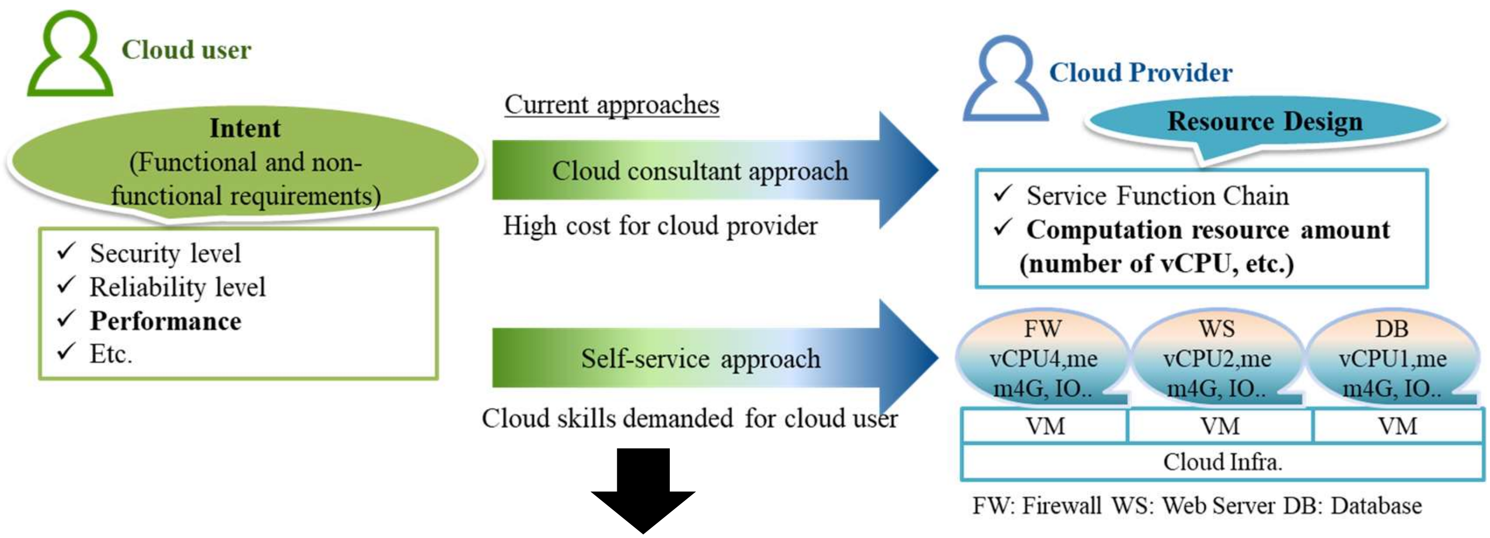
PoC milestones

PoC Milestone	Stages/Milestone description	Target Date
P.S	PoC Project Start	June 2021, ENI #18
P.U	PoC user story	September 2021, ENI #19
	NTT R&D Forum 2021	November 2021
P.D1	PoC Demo	December 2021, ENI#20
P.C	PoC Contribution	March 2022
P.R	PoC Report	March 2022
P.E	PoC Project End	June 2022

Current status

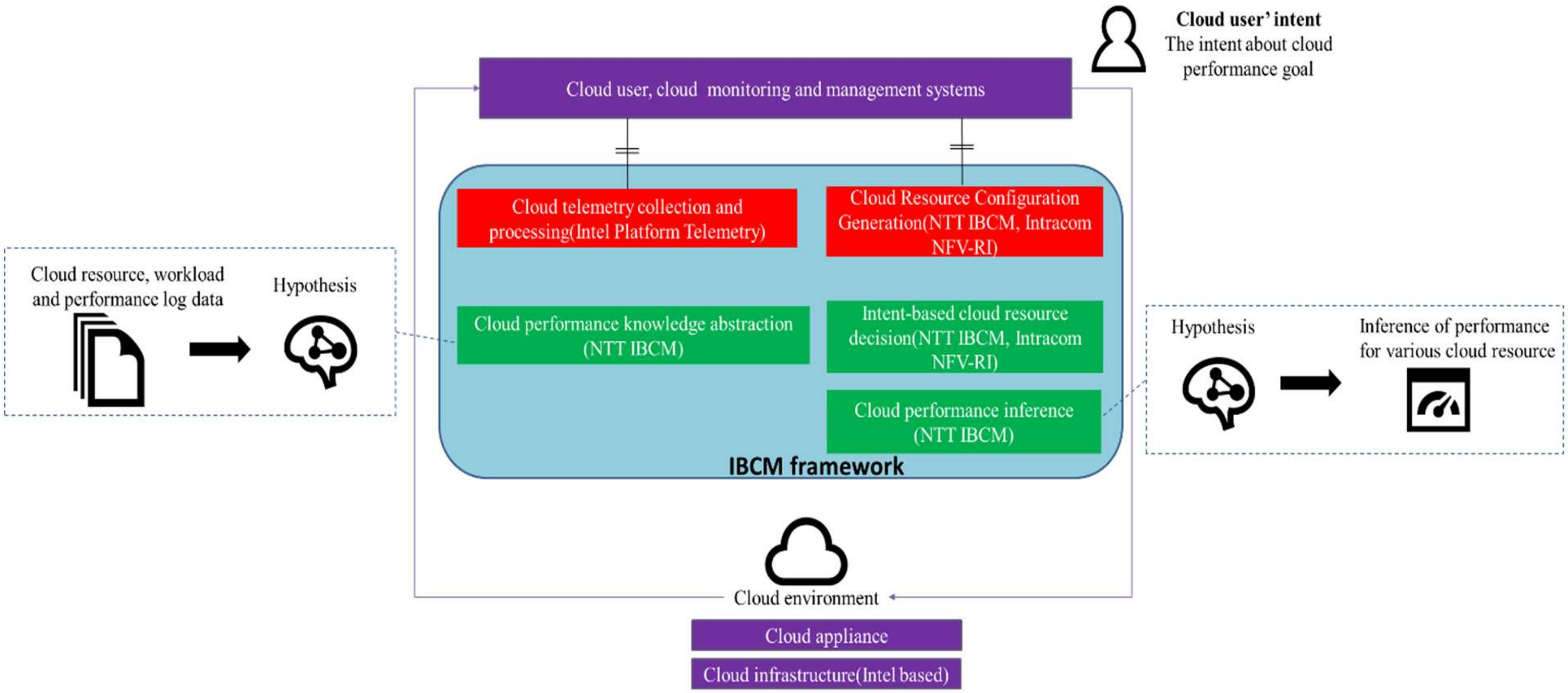
PoC goal

PoC Project Name: Intent-based Cloud Management (IBCM)
Short Description: This PoC will provide an Intent-Based Cloud Management (IBCM) solution that **assists the cloud provider with decision making about cloud computing resources, to meet the cloud performance goal**, i.e. the intent.
In the PoC, we will demonstrate abstracting knowledge(building AI models) from cloud telemetry data, and making decisions of necessary cloud computing resources that meets the cloud performance goal using the knowledge (the models). Consequently, **reduction of OPEX including the human resource cost, time cost, cloud resource cost, energy cost can be expected by using IBCM.**



IBCM: autonomize the cloud resource decision

PoC Architecture



PoC user story

UC #1 Intent-based Cloud Management for VDI service

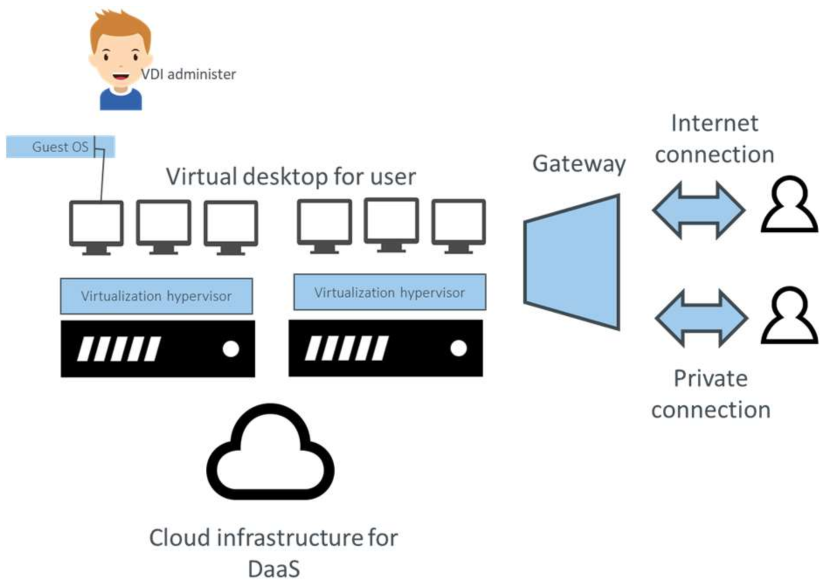
UC #2 Intent-based Cloud Management for NFV workloads

VDI: Virtual Desktop Infrastructure

NFV: Network Function Virtualization

UC #1 Intent-based Cloud Management for VDI service

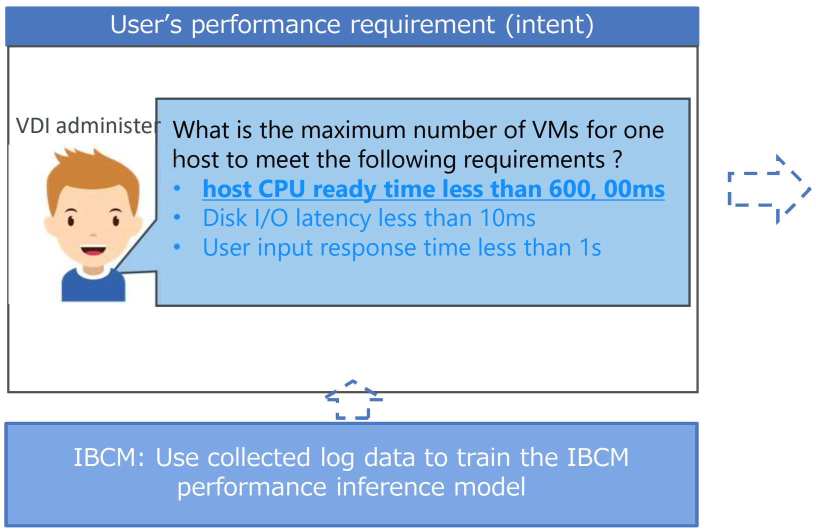
Preconditions: In a VDI service, virtual desktop environments are implemented as VM instances on public/private cloud hosts. VDI users conduct their daily work in the virtual desktop instances.
Problem with conventional approach: In order to maintain users' QoE , VDI administrators need to determine and adjust the number of VMs to be placed on the each host appropriately. However, this decision requires a high level of skill and experience. Improper decision can lead to poor user experience or low resource efficiency.
Objective: IBCM automatically calculates the optimum number of VMs that does not deteriorate the user experience. Thus realizes reduction of human cost and resource cost.

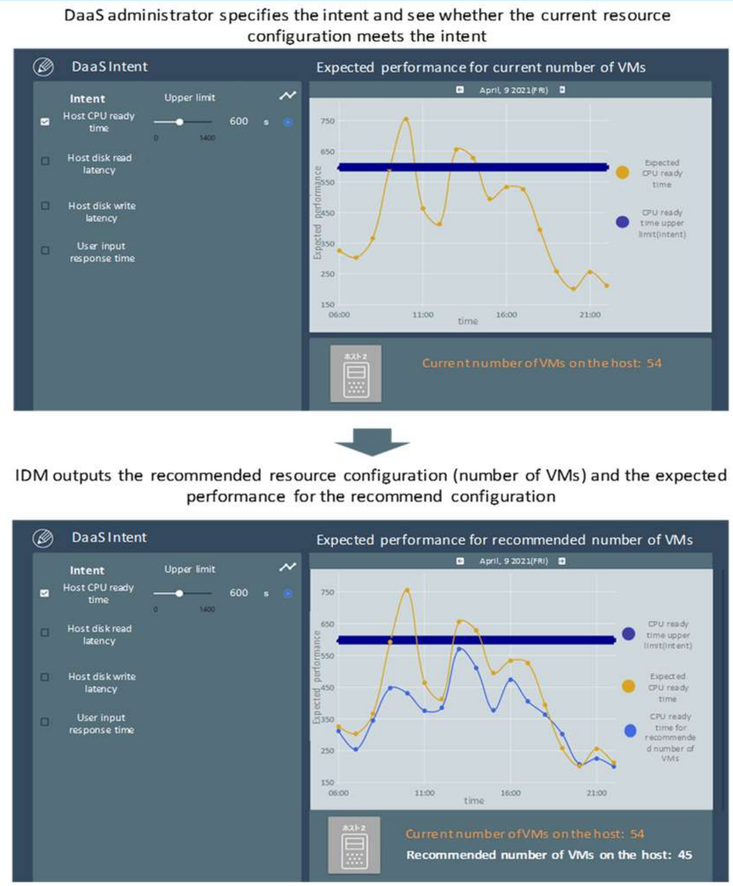


UC #1 Intent-based Cloud Management for VDI service

Step1

Step0





Step2

Step3

Step0: Collect the VDI performance log data using platform telemetry and build the IBCM model.
Step1: The VDI operator specifies the intent through the GUI
Step2: IBCM checks if the current resource configuration meets the intent
Step3: If not, IBCM calculates the number of instances to be allocated to the host that meet the intent as well as the expected performance. The result is fed back to the operator for confirmation.
The decision is transformed into machine-readable resource orchestration template and handed to VDI resource management system for implementation

UC #2 Intent-based Cloud Management for NFV workloads

As a mobile core network operator
I need to find ways to easily & safely colocate NFV workloads on NFVi servers at the core or at the edge,
so that server density gets increased, energy consumption gets minimized, while the workloads' SLOs (provided as "intents") are always maintained despite any dynamic change.

To do this, I need to leverage modern server technologies for efficient hardware resource partitioning (LLC, memory B/W, power) and isolation, combined with AI/ML techniques to accurately apply resource allocations at the right amount and time (no more than needed, and before SLA violations occur)

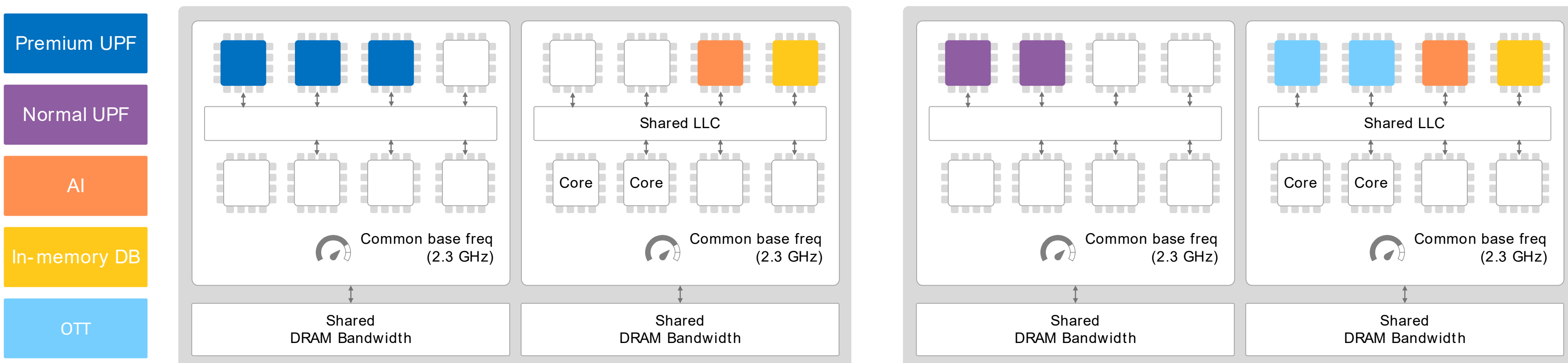
I know that I am successful when

- the declared **SLOs** for high-priority workloads are always met, even when dynamic changes occur (e.g. traffic variation, workload arrival/departure)
- the overall **power consumption** gets reduced, as compared to the best possible deployment scheme that would not use any resource partitioning feature (either because servers can be entirely evacuated, or because they transition from symmetric to asymmetric power configurations)
- the overall **time** needed to discover optimal resource decisions gets reduced, as compared to the best possible manual/semi-manual approach for the same purpose

UC #2 Intent-based Cloud Management for NFV workloads

Differentiated 5G online gaming services

- The operator offers two low-latency gaming services to its customers backed by differentiated 5G slices: a *premium* slice (ultra-low latency), and a *normal* slice (low latency). This essentially translates to two sets of UPFs with differentiated intents: e.g. premium UPF < 0.06 msec, normal UPF < 0.3 msec.
- “AS IS” state:
  - without resource partitioning technologies in place, the operator would isolate each UPF instance to its own NUMA node, reserving upfront any cores left idle in order to avoid contention that would put SLAs under risk. At large scale, a larger number of NFVi servers would be needed to host many UPF instances.
  - with resource partitioning techniques, the operator would need much time and expertise to discover colocated placements that would reduce the total number of servers needed. Even in that case, however, he should experiment assuming the worst-case scenario for each VNF (i.e. max expected traffic), thus missing opportunities for even denser consolidation in quiet periods



UC #2 Intent-based Cloud Management for NFV workloads

Differentiated 5G online gaming services

“TO BE” state

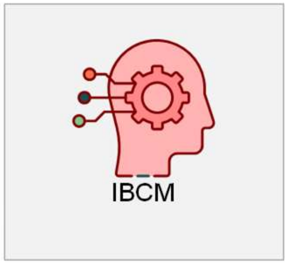
Step 1: user input

Performance-level intents (SLOs)

- premium UPF latency < 0.06 msec
- premium UPF drop rate < 0.0003
- normal UPF latency < 0.3 msec
- normal UPF drop rate < 0.0005

Training data

diurnal load/traffic patterns, as expected in production



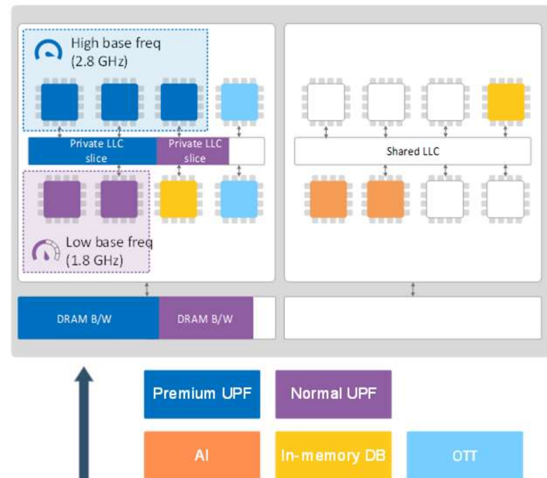
Step 2: train IBCM on intents, in a staging environment

IBCM tests many different resource configs for all colocated workloads, and for different traffic levels, in order to discover the configs that deliver the intended SLOs with minimal HW resource footprint

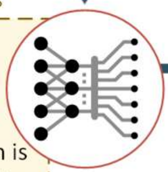
actions:

- allocate LLC partition for a workload (Intel RDT)
- set memory B/W for a workload (Intel RDT)
- set base frequency for a workload (Intel SST)

On every tested config, IBCM gets feedback relevant to the intents using telemetry from various levels (platform, workloads)



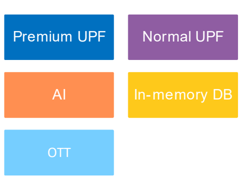
Step 3: use trained IBCM model in the production environment for local & real-time decision making

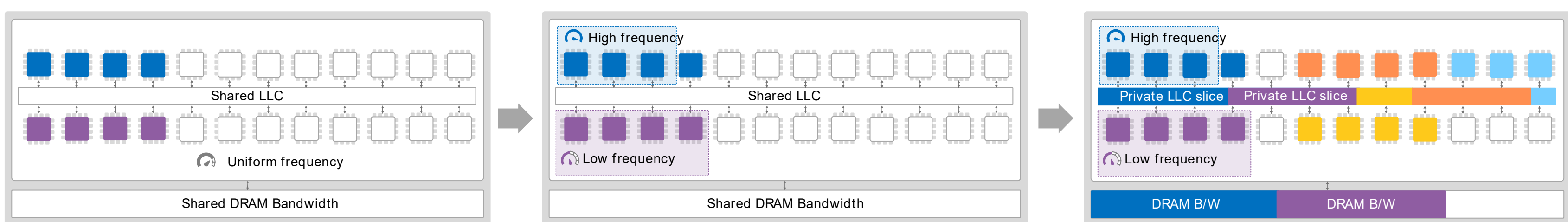


- The production environment needs to be identical to the staging environment, for best reproducibility
- Intents are encoded in the IBCM model, which is trained for the specific application(s) and for the specific traffic bounds provided by the user
- Different applications, intents, traffic bounds will typically require retraining

UC #2 Demo

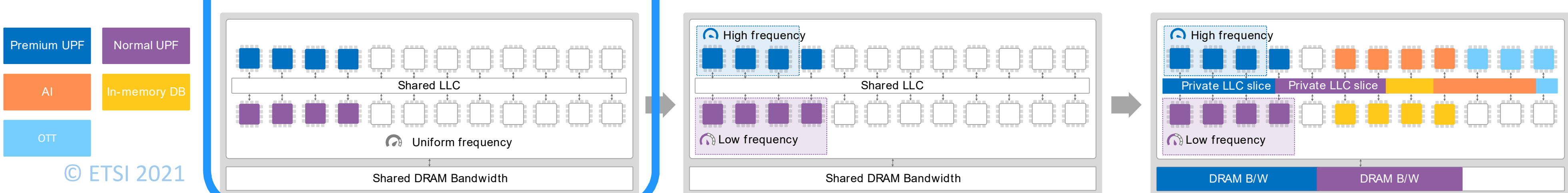
	Baseline scenario	Scenario 2	Scenario 3
Short desc.	1 premium UPF (pUPF) + 1 normal UPF (nUPF), with differentiated latency constraints, each accepting its own traffic	Dynamically adjust CPU frequency for UPFs in line with their incoming traffic	Dense colocation of additional workloads
Problem with existing approach	With a **uniform frequency** configuration for all workloads, nUPF might end up with much larger frequency headroom needed to meet its constraints, as opposed to pUPF	UPFs are using the **highest frequency** that is sufficient to meet their latency SLOs at their busiest times	UPFs are running fine, but there are **CPUs left idle**. When additional workloads kick in to utilize them, they introduce **contention** and SLO violations on UPFs
IBCM's proposition	Statically allocate the **ideal CPU frequencies** to pUPF & nUPF to match their highest load	**Dynamically scale down** UPFs frequencies according to their actual load, without incurring SLO violations	Leverage LLC and DRAM B/W as additional resources to control dynamically, in order to **protect** the latency-sensitive UPFs from contention
Expected benefit	Less power consumed due to asymmetric frequency configuration (e.g. nUPF might get significantly less frequency than its default to meet its constraints)	Additional energy gains due to throttling CPU frequencies when there is no actual need	Energy gains at scale due to colocation. SLO violations back to zero.





UC #2 Demo

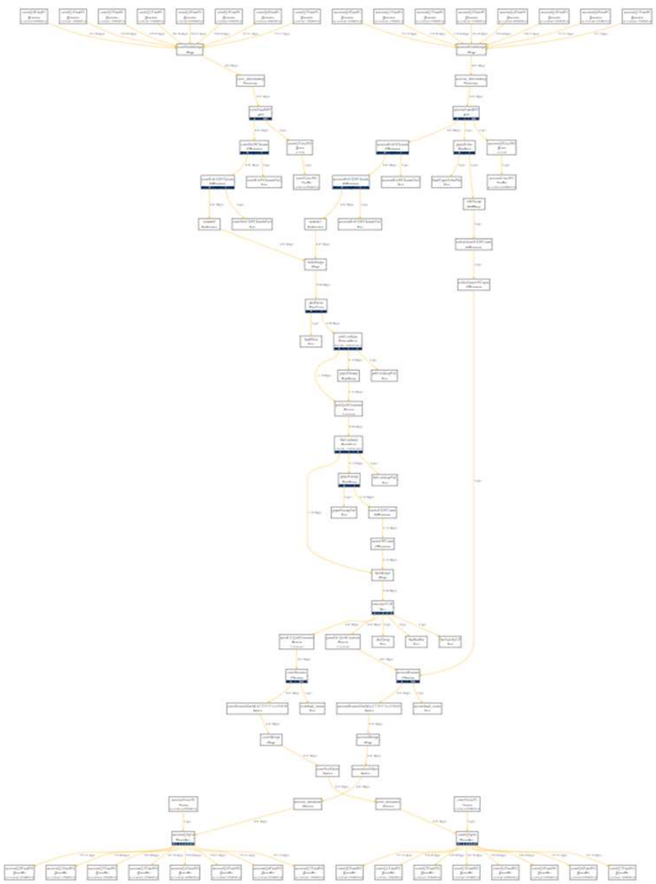
	Baseline scenario	Scenario 2	Scenario 3
Short desc.	1 premium UPF (pUPF) + 1 normal UPF (nUPF), with differentiated latency constraints, each accepting its own traffic	Dynamically adjust CPU frequency for UPFs in line with their incoming traffic	Dense colocation of additional workloads
Problem with existing approach	With a **uniform frequency** configuration for all workloads, nUPF might end up with much larger frequency headroom needed to meet its constraints, as opposed to pUPF	UPFs are using the **highest frequency** that is sufficient to meet their latency SLOs at their busiest times	UPFs are running fine, but there are **CPUs left idle**. When additional workloads kick in to utilize them, they introduce **contention** and SLO violations on UPFs
IBCM's proposition	Statically allocate the **ideal CPU frequencies** to pUPF & nUPF to match their highest load	**Dynamically scale down** UPFs frequencies according to their actual load, without incurring SLO violations	Leverage LLC and DRAM B/W as additional resources to control dynamically, in order to **protect** the latency-sensitive UPFs from contention
Expected benefit	Less power consumed due to asymmetric frequency configuration (e.g. nUPF might get significantly less frequency than its default to meet its constraints)	Additional energy gains due to throttling CPU frequencies when there is no actual need	Energy gains at scale due to colocation. SLO violations back to zero.



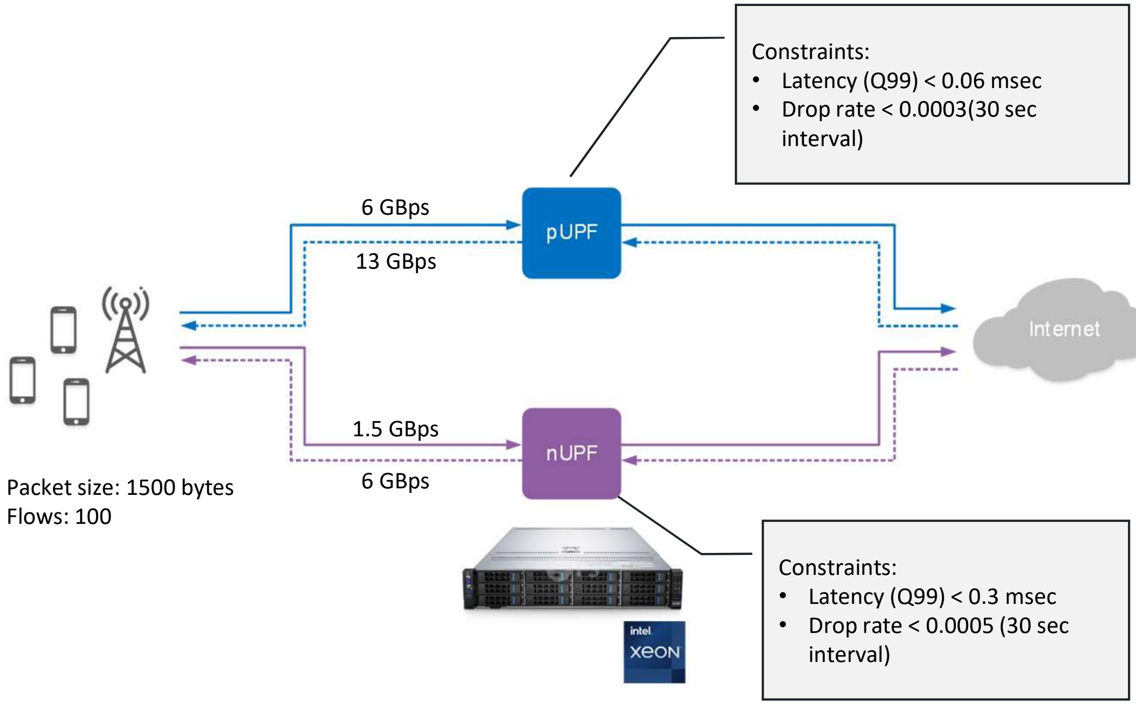
Baseline Scenario - Setup

UPFs based on OMEC's UPF-EPC

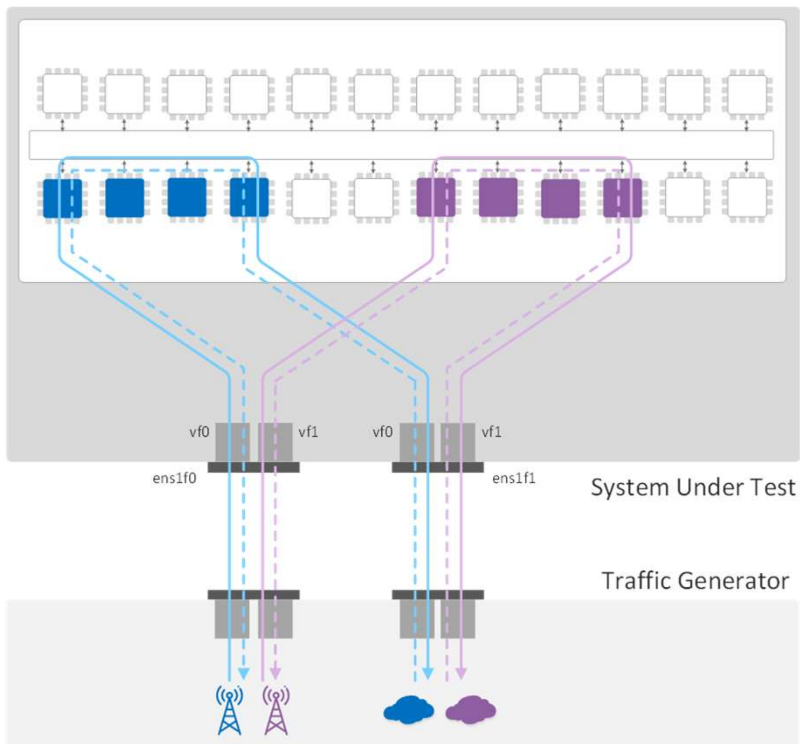
- conforms to Control User Plane Separated (CUPS) architecture
- based on the 3GPP TS23501 specifications of EPC and functions as a co-located Service and Packet Gateway (SPGW-U)
- dataplane built on top of the BESS framework, where each submodule in the SPGW-U pipeline is represented as a node in a processing pipeline graph
- DPDK 20.11.3 under the hood



Baseline Scenario - Setup



Intel® Xeon® Gold 6252N @ 2.30GHz (microcode: 0x5003102)
2 CPUs (NUMA) x 24 cores x 2 hyper-threads
Intel® Hyper-Threading Technology: on
Intel® Turbo Boost Technology: on
Intel® Ethernet Controller XXV710 25GbE
BIOS version: 2.7.7
Ubuntu 20.04.2 LTS / kernel 5.4.0-81-generic



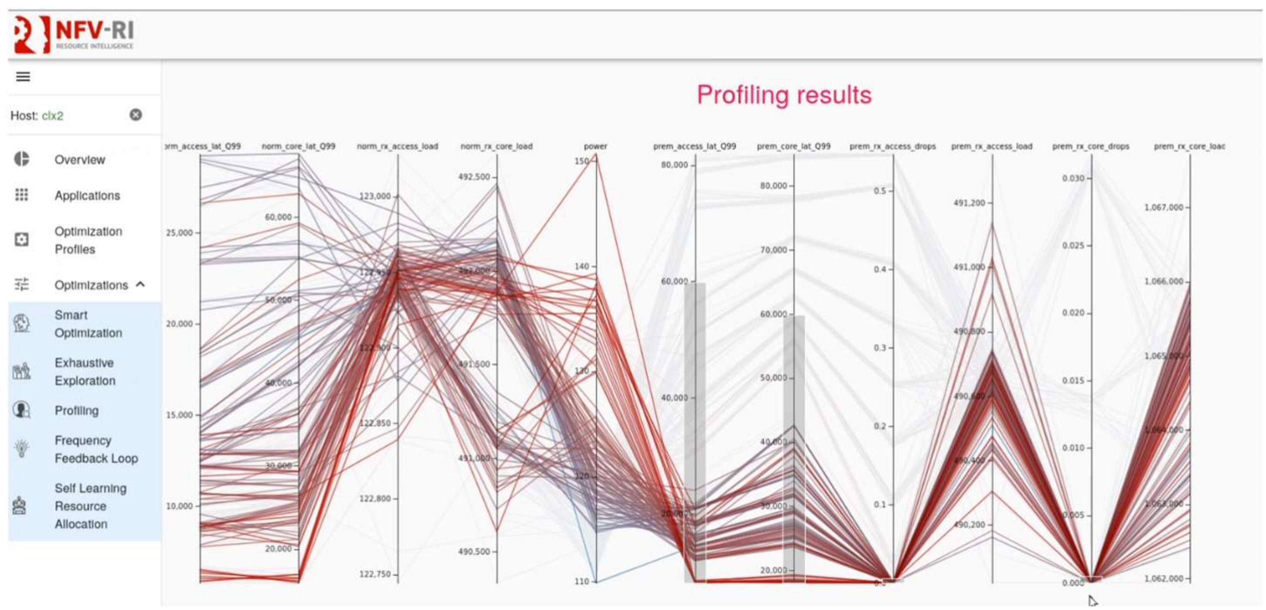
Baseline Scenario - Results

Default:

- pUPF’s default CPU freq: ~3.2 GHZ
- nUPF’s default CPU freq: ~3.2 GHz
- wall power: 264 Watts
- no constraints violations

Exhaustive search of optimal resource combinations:

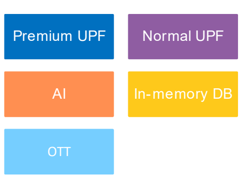
- pUPF’s CPU freq: 2.1 GHz
- nUPF’s CPU freq: 1.4 GHz
- wall power: 220 Wats (**↓16.67%**)
- **no constraints violations**

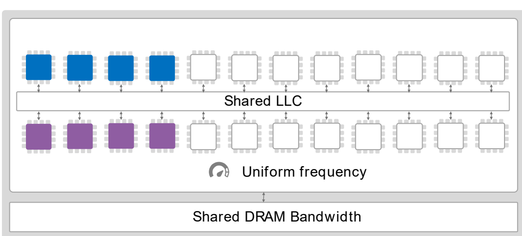


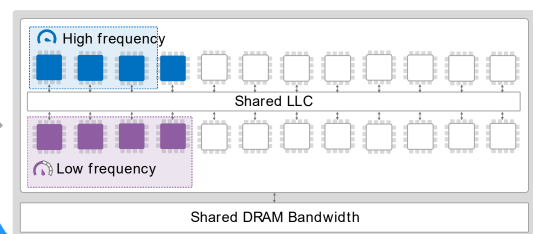
[Scenario demo video]

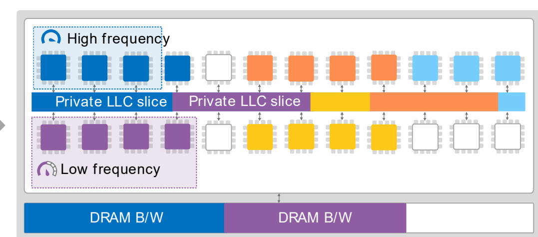
UC #2 Demo

	Baseline scenario	Scenario 2	Scenario 3
Short desc.	1 premium UPF (pUPF) + 1 normal UPF (nUPF), with differentiated latency constraints, each accepting its own traffic	Dynamically adjust CPU frequency for UPFs in line with their incoming traffic	Dense colocation of additional workloads
Problem with existing approach	With a **uniform frequency** configuration for all workloads, nUPF might end up with much larger frequency headroom needed to meet its constraints, as opposed to pUPF	UPFs are using the **highest frequency** that is sufficient to meet their latency SLOs at their busiest times	UPFs are running fine, but there are **CPUs left idle**. When additional workloads kick in to utilize them, they introduce **contention** and SLO violations on UPFs
IBCM's proposition	Statically allocate the **ideal CPU frequencies** to pUPF & nUPF to match their highest load	**Dynamically scale down** UPFs frequencies according to their actual load, without incurring SLO violations	Leverage LLC and DRAM B/W as additional resources to control dynamically, in order to **protect** the latency-sensitive UPFs from contention
Expected benefit	Less power consumed due to asymmetric frequency configuration (e.g. nUPF might get significantly less frequency than its default to meet its constraints)	Additional energy gains due to throttling CPU frequencies when there is no actual need	Energy gains at scale due to colocation. SLO violations back to zero.

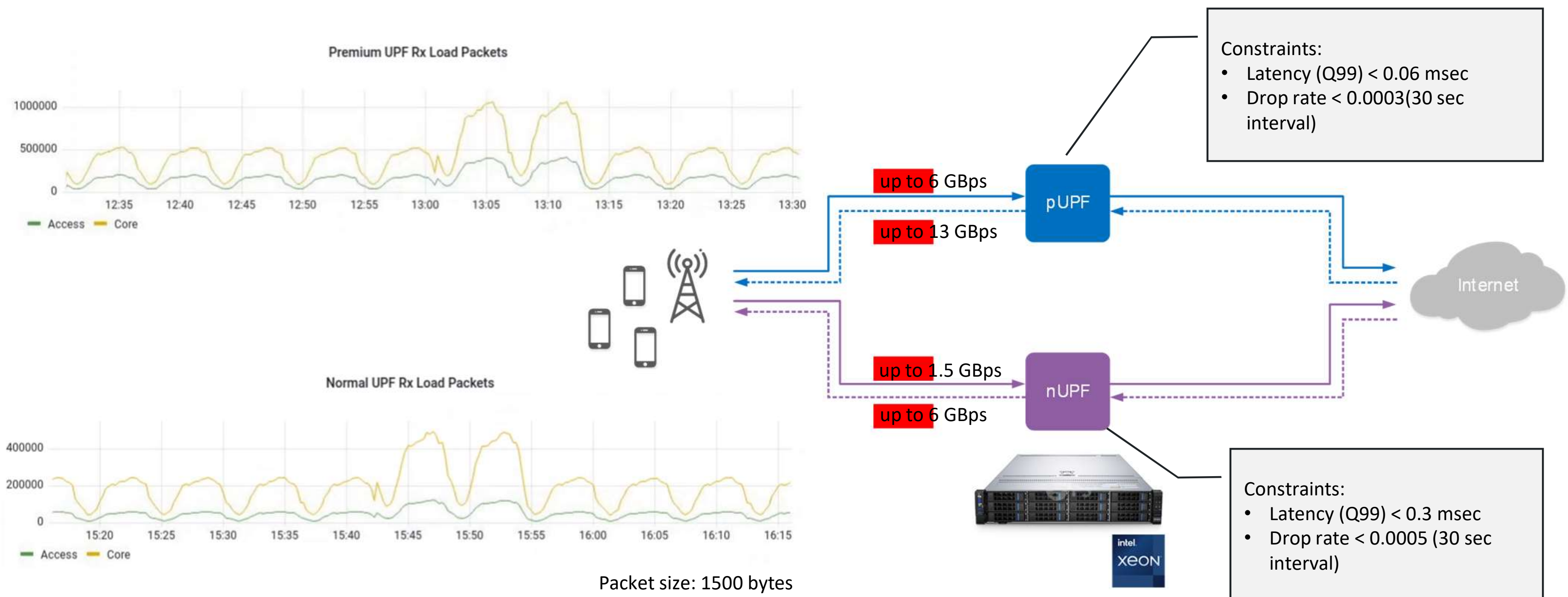








Scenario 2 - Setup



Packet size: 1500 bytes
Flows: 100

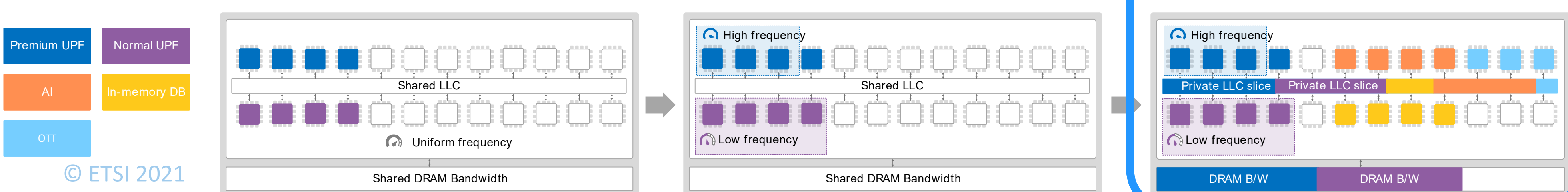
Scenario 2 - Results

Deep RL used to dynamically change CPU frequency in line with traffic:

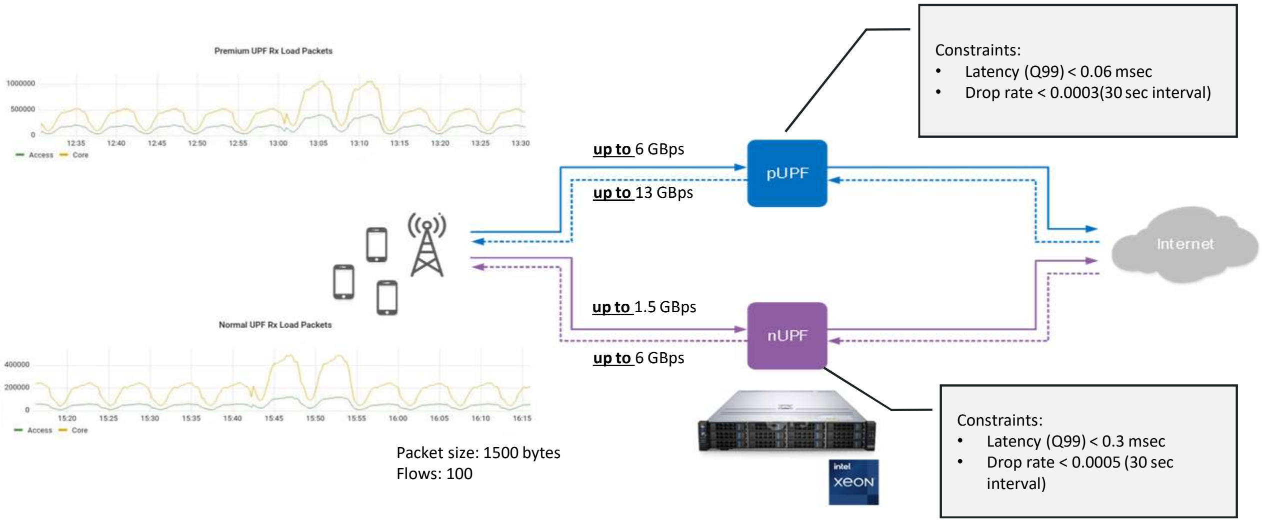
- RL Agent trained on each UPF's patterns to minimize power while **always** meeting the specified latency & packet drops constraints
- after training, RL Agent is deployed in production
- dynamic frequency throttling during non-busy periods further reduces server power **without any SLO violation**
- average power reduced by **21.5%** w.r.t. baseline (264W → 207W), or 5.9% w.r.t. the static frequency provisioning

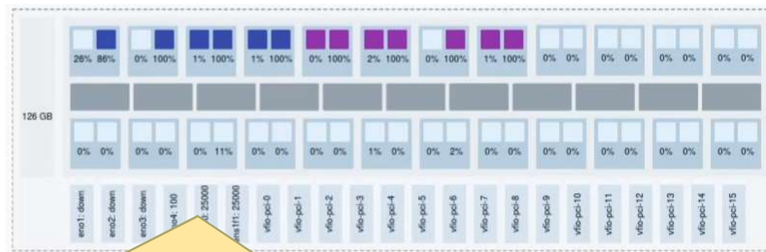
UC #2 Demo

	Baseline scenario	Scenario 2	Scenario 3
Short desc.	1 premium UPF (pUPF) + 1 normal UPF (nUPF), with differentiated latency constraints, each accepting its own traffic	Dynamically adjust CPU frequency for UPFs in line with their incoming traffic	Dense colocation of additional workloads
Problem with existing approach	With a **uniform frequency** configuration for all workloads, nUPF might end up with much larger frequency headroom needed to meet its constraints, as opposed to pUPF	UPFs are using the **highest frequency** that is sufficient to meet their latency SLOs at their busiest times	UPFs are running fine, but there are **CPUs left idle**. When additional workloads kick in to utilize them, they introduce **contention** and SLO violations on UPFs
IBCM's proposition	Statically allocate the **ideal CPU frequencies** to pUPF & nUPF to match their highest load	**Dynamically scale down** UPFs frequencies according to their actual load, without incurring SLO violations	Leverage LLC and DRAM B/W as additional resources to control dynamically, in order to **protect** the latency-sensitive UPFs from contention
Expected benefit	Less power consumed due to asymmetric frequency configuration (e.g. nUPF might get significantly less frequency than its default to meet its constraints)	Additional energy gains due to throttling CPU frequencies when there is no actual need	Energy gains at scale due to colocation. SLO violations back to zero.



Scenario 3 - Setup

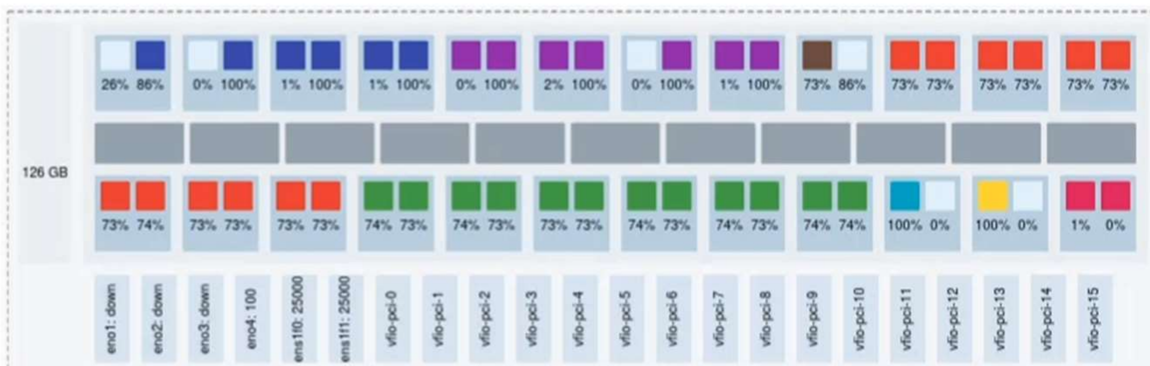




- 8/24 cores of the CPU utilized at 100%
- 16/24 cores almost idle

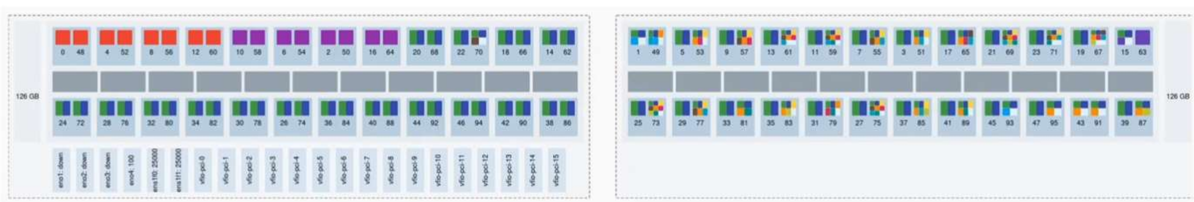
Best-Effort workloads

2 AI inference apps (image classification)
+ 2 OTT streaming apps
+ 1 in-memory DB



Scenario 3 – Colocation impact on UPFs' constraints

Constraint	Premium UPF constraint satisfaction	Normal UPF constraint satisfaction
Access latency	97% † - 100% ‡	100%
Core latency	23% † - 100% ‡	100%
Access drop rate	**88% † - 97% ‡**	100%
Core drop rate	100%	100%



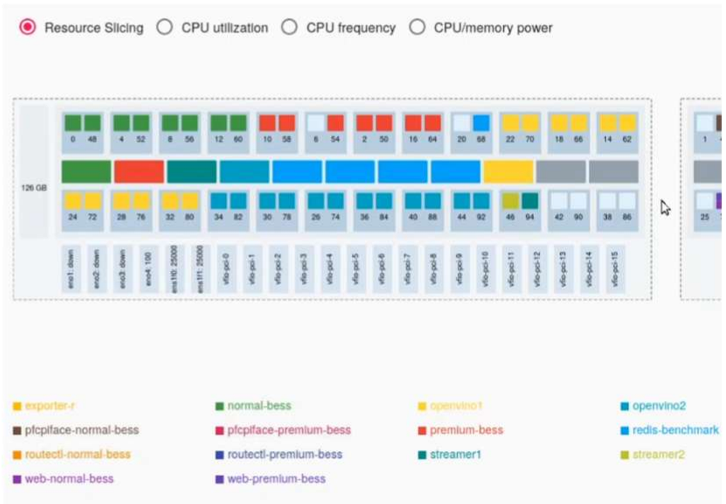
† best effort workloads allowed to bounce across cores of the 2 CPUs (without interfering with the UPFs' cores)

‡ best effort workloads strictly consolidated in the same CPU socket as the UPFs (without interfering with the UPFs' cores)

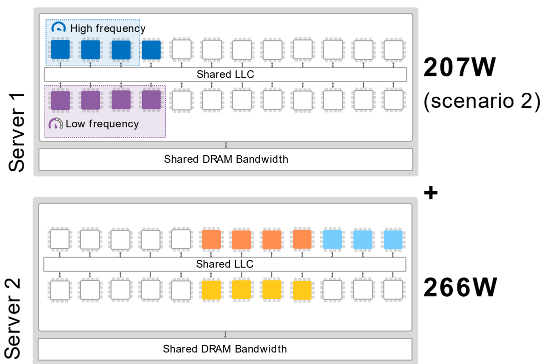
Scenario 3 - Results

RL Agent trained to handle Intel's RDT resources (Last Level Cache + Memory B/W) in addition to CPU frequency:

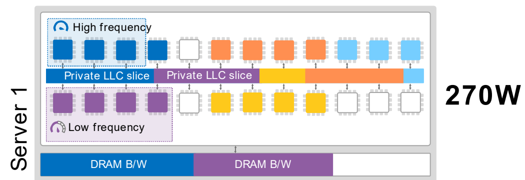
- as in scenario 2, agent trained on each UPF's patterns to minimize power while **always** meeting the latency & packet drops constraints
- protecting UPFs' LLC & memory B/W **restores** premium UPF's constraints satisfaction to **100%**



Dedicated servers for UPFs & BE workloads



Colocated execution (scenario 3)



- **43%** reduction in average power
- BE workloads slowdowns:
  - up to 2.4x for in-memory DB
  - up to 1.2x for AI
  - up to 3x for OTT

Scenario demo video

Q: How much data/time is required? (UC#1)

Log data of at least one typical workday is necessary. (About 500 records of log data, 30 kinds of workload, resource and performance data)

It takes less than 10 minutes to train the model with one day's log data. And the average inference time is less than 5 seconds.

More ideally, weekly collected log data is preferred to guarantee better precision.

Q: How much data/time is required? (UC #2)

Depends on the application complexity and number of actions allowed

Initially staging of the application is required (for now)

- Samples are gathered for different load/traffic levels for all available actions
- For simple applications at least 10 traffic levels are recommended

Such a process for one action dimension (e.g. LLC slicing) requires approx. 30 minutes and the input data are minimal (<10kB). The resulting model is approx. 10MB in size

List of contributors

Chao Wu (NTT)

Emma Collins (Intel)

Haining Wang(Intel)

Cavigioli (Intel)

Anastopoulos Nikos (Intracom Telecom)

Shingo Horiuchi (NTT)

Kenji Murase (NTT)

Hiroaki Kikushima(NTT)

Kenichi Tayama (NTT)

Browne John (Intel)

Macnamara, Chris (Intel)

Evangelos Angelou (Intracom Telecom)

Georgia Panoutsakopoulou (Intracom Telecom)

Loukas Gravias (Intracom Telecom)

Victor Timofei (Intracom Telecom)

Georgios Hortatos (Intracom Telecom)