# SEMANTIC

*end-to-end Slicing and data-drivEn autoMAtion of Next generation cellular neTworks with mobIle edge Clouds*

*Marie Skłodowska-Curie Actions (MSCA)*
*Innovative Training Networks (ITN)*
*H2020-MSCA-ITN-2019*
*861165 - SEMANTIC*

## WP2 – MEC/RAN integration and MEC-empowered enhancements

## D2.3: Performance evaluation of MEC-empowered enhancements

| | |
|---|---|
| Contractual Date of Delivery: | M48 |
| Actual Date of Delivery: | 31/01/2024 |
| Responsible Beneficiary: | POLITO |
| Contributing beneficiaries: | POLITO, CTTC, UOA, FOG |
| Security: | Public |
| Nature: | Report |
| Version: | V0.2 |

## Document Information

Version Date: 29/01/2024
Total Number of Pages: 60

## Authors

| Name | Organization | Email |
|---|---|---|
| Dr. Nikos Passas | UOA | passas@di.uoa.gr |
| Vasilis Gretsistas | UOA | vagretsi@di.uoa.gr |
| Madhura Adeppady | POLITO | madhura.adeppady@polito.it |
| Vaishnavi Kasuluru | CTTC | vkasuluru@ctts.es |
| Klearchos Palias | FOG | klearchos@fogus.gr |
| Prof. Carla Fabiana Chiasserini | POLITO | carla.chiasserini@polito.it |
| Dr. Paolo Giaccone | POLITO | paolo.giaccone@polito.it |
| Dr. Dionysis Xenakis | FOG | dionysis@fogus.gr |
| Dr. Luis Blanco | CTTC | luis.blanco@cttc.es |

## Document History

| Revision | Date | Modification | Contact Person |
|---|---|---|---|
| V0.1 | 16/11/2023 | Defining a preliminary table of contents | Madhura Adeppady |
| V0.2 | 29/01/2024 | Merging the contributions received by ESRs | Madhura Adeppady |
| | | | |
| | | | |

## Table of Contents

# List of Figures

# List of Tables

# 1 Executive summary

This report includes the SEMANTIC ESR contributions towards the objectives of WP2 (MEC/RAN integration and MEC-empowered enhancements). More specifically, the report briefly summarizes the proposed solutions to enhance the architectural and functional upgrades for MEC/RAN integration followed by an extensive performance evaluation of the proposed solutions. The document is structured with one section per ESR contributing to WP2, that includes their contribution in the area. A last section at the end concludes the document and sets the target for the future work.

# 2 Microservice Deployment and Management at the Edge (POLITO)

## 2.1 Introduction

Edge computing has emerged as a solution to serve a large number of real-time computational tasks while reducing bandwidth usage and end-to-end latency [1]. Despite enjoying many benefits, the widespread deployment of edge computing is still challenging [2]. From the system perspective, provisioning computing resources at the granularity of virtual machines, as done in traditional cloud computing, brings in long provisional delays and resource wastage, which is unacceptable for resource-constrained edge servers and time-critical services. Also, application developers still bear the heavy burden of explicitly managing the resources, load balancing, and scalability. Serverless computing, with its Function-as-a-Service (FaaS) offering, redefines the way of deploying services [3], as it enables the decomposition of their logic into stateless microservices (MS). Such MSs are run on demand in an event-driven manner in lightweight containers with no need for resource pre-allocation. Using serverless edge computing allows services to use underlying resources on demand without the burden of load balancing, scalability, and runtime environments, thus improving resource utilization [4], [5].

Importantly, in serverless edge computing, MSs run inside the containers only when requested. Thus, serving a request involves creating a new container with appropriate runtime, which may involve downloading the necessary image from the remote repository, fetching and loading essential libraries and dependencies before executing the actual function. This process is known as cold start and the long delay involved in the initialization setup is known as start-up latency, which is one of the main performance issues faced by serverless computing platforms [6]-[8]. A warm container keeps instead the MS instance alive in the memory, with a negligible start-up latency when the warm container is reused for serving a later request for the same MS. However, due to limited memory at the edge nodes, serving all the requests with warm containers is practically impossible [4]. Further, keeping warm containers in memory reduces resource utilization and violates the resource elasticity promises of serverless computing, in which resources are occupied when required.

Recently, many research efforts have been devoted to reducing the cold start frequencies MSs by proposing various strategies for managing the keep-alive time of the warm containers [6], [7]. However, these approaches allocate the resources to the container by largely overlooking MS-specific QoS requirements, e.g., target delay. Unlike prior work, in this paper, we face the above issue with the additional twofold aim to (i) ensure the level of QoS required by the MSs offered to the mobile users, and (ii) reduce the data center’s energy footprint. Indeed, it is well known that edge data centers consume a significant amount of energy, which depends on their CPU loa [9]. Towards these goals, we provide the following contributions:

(1) Through a detailed, yet tractable, model of the system (Sec. 2.2), we formulate an optimization problem that, looking at a finite time horizon, minimizes the servers energy consumption by leveraging cold, warm, and running containers (Sec. 2.3). In particular, we note that, for the MSs with stringent delay constraints, serving the requests using a cold container requires a high CPU speed allocation. In contrast, using a warm container is more energy efficient because it requires low CPU speed allocation due to negligible start-up latency. However, it may be impossible to serve all requests with warm containers due to the limited memory of edge servers.

(2) Since, in spite of the limited lookahead perspective of the proposed formulation, the problem turns out to be NP-hard, we investigate a simple threshold-based queueing solution (Sec. 2.4), which, surprisingly, closely matches the optimum (Sec. 2.5). Specifically, we define a threshold on the number of requests waiting to be served and start new containers from cold/warm state only when this number exceeds the threshold. By adopting a FIFO policy for scheduling the requests and restricting the number of waiting requests, we can reduce the number of running containers and cold starts, hence decreasing the overall energy consumption of the active servers.

## 2.2 System Model

Let us focus on a single data center and let $\mathcal{S}$ be the set of servers available therein, with $s \in \mathcal{S}$ having $\widehat{\tau_s}$ bytes of memory and $\widehat{\mu_s}$ as CPU capacity (in cycles/s). A service orchestrator receives requests for any MS $\mathrm{k} \in \mathcal{K}$ and serves them using the serverless computing paradigm. Any request for MS $k$ demands certain amount of memory and workload (in CPU cycles), denoted by $\tau_k$ and $w_k$, respectively, and has a target maximum delay $D_k$ in terms of time lapse from when the request arrives till the service execution is completed.

A container can be in any of the three states: running ($R$), warm ($W$), cold ($C$) or in any of these two transition states: transiting from $C$ to $R$ ($T_{C\to R}$) or transiting from $R$ to $C$ ($T_{R\to C}$). The transition to/from $W$ involves negligible start-up latency, hence we consider only $T_{C\to R}$ and $T_{R\to C}$. For a container implementing service $k$, let $\delta_{k,C}$ denote the start-up latency of cold start. For simplicity, we assume that the transition time from state $R$ to state $C$ is same as $\delta_{k,C}$.

Let $c_{k,i,R}(t)$ be the container in state $R$ serving the $i$-th request for MS $k$, $r_{k,i}$ at time $t$. When starting to serve request $r_{k,i}$, the orchestrator assigns a CPU speed in cycles/s, denoted by $\mu_{c_{k,i,R}}$, to container $c_{k,i,R}$ such that i) overall CPU capacity at the server is not exceeded and ii) $r_{k,i}$is served within $D_k$. Additionally, the memory assigned to any container implementing MS $k$ in state $R$ is equal to the memory demand of an instance of MS $k$. We stress that a container in warm state at time $t$ that implements MS $k$, $c_{k,W}(t)$, consumes only memory $\tau_{k,W}$. When in state $C$, a container consumes neither CPU nor memory. Further, let $c_{k,i,T_{C\to R}}(t)$ (or, $c_{k,i,T_{R\to C}}(t)$) be the container in transition from $C$ to $R$ (or, from $R$ to $C$) to serve request $r_{k,i}$ at $t$. A container of MS $k$ in any of the above mentioned transition states consumes both memory and CPU cycles/s, denoted by $\mu_{k,T}$ and $\tau_{k,T}$, respectively. For MS $k$, the memory and CPU consumption of all the container transition states remain the same. Also, the orchestrator acts upon each queue according to a FIFO policy, by serving the requests on appropriate containers. Note however that the head-of-the-line (HoL) request and the first request to handle in the queue may not always be the same. If the HoL request has already been scheduled to run on a container that is currently transitioning, then the second request in the queue becomes the first request to handle. Instead, if the HoL request in the queue is not scheduled to run on any container, then it remains as the first request to handle in the queue.

For serving the first request to handle in the queue, we have two cases. In the first one, we consider that no warm container is available. Hence, the orchestrator creates a new container and, once the container reaches state $R$, the first request in the queue is served on it. In the second case, we consider that warm containers are available. In such a situation, the orchestrator can serve the request in a warm or cold container. In both cases, after being assigned to a warm or cold container, the request still remains in the queue and will be removed when the container enters state $R$. During this time, the subsequent request in the queue becomes the first request to be handled by the orchestrator. Let $t$ be the time at which any of the previously mentioned events (arrival of a new request, container finishes serving a request, or container completes the transition) occurs. At time $t$, we denote by $\Omega^s_{k,W}(t)$, $\Omega^s_{k,R}(t)$, and $\Omega^s_{k,T}(t)$ the set of all containers of MS $k$ on server $s$ in state $W$, $R$, and $T_{C\to R}$ or $T_{R\to C}$ respectively. Let $\Omega_{k,R}(t) = \cup_{s\in\mathcal{S}} \Omega^s_{k,R}(t)$, and $\Omega_{k,T}(t) = \cup_{s\in\mathcal{S}} \Omega^s_{k,T}(t)$ be the set of running and transiting containers across all the servers, respectively.

Processing time $T_{k,i}$ of request $r_{k,i}$ running on container $c_{k,i,R}$ is computed by $T_{k,i} = w_k/\mu_{c_{k,i,R}}$ where we assume that $c_{k,i,R}$ runs at rate $\mu_{c_{k,i,R}}$ to process $r_{k,i}$, and thus the expression holds independently from the concurrent requests. The total service time of $r_{k,i}$ is the sum of the queueing delay before being served and the processing time. Note that the queueing delay experienced by a request also includes the start-up latency of the container on which the request is scheduled to run once the container enters state $R$.

The power consumption of a server is composed of (i) a constant $P_{\text{idle}}$ representing the idle power consumption when a server is active, and (ii) a function $P$ of the server's actual CPU load. Thus, the power consumption of server $s$ with CPU load $\lambda_s$ is given by $P_s = P_{\text{idle}} + P(\lambda_s)$.

## 2.3 Problem Formulation

We now describe the event-driven problem formulation whose objective is to reduce the data center's power consumption by minimizing the CPU load on the servers while ensuring that requests are served within the desired target delay.

Below, if the queue $Q_k$ does not contain unhandled requests, then we denote this case by setting $\epsilon_k = 1$. Otherwise, $\epsilon_k = 0$. At $t$, the CPU load $\lambda_{s,t}$ of a server $s$ is the sum of the CPU load by all the running and transiting containers on $s$, i.e.,

$$\lambda_{s,t} = \sum_{k \in \mathcal{K}} \left( \sum_{c_{k,i,R} \in \Omega^{s}_{k,R}(t)} \mu_{c_{k,i,R}} + \sum_{c_{k,T} \in \Omega^{s}_{k,T}(t)} \mu_{k,T} \right)$$

Similarly, at $t$, the memory consumption $\nu_{s,t}$ of server $s$ is the sum of memory consumed by all the running, transiting, and warm containers on $s$, i.e.,

$$\nu_{s,t} = \sum_{k \in \mathcal{K}} \left( \sum_{c_{k,i,R} \in \Omega^{s}_{k,R}(t)} \tau_k + \sum_{c_{k,W} \in \Omega^{s}_{k,W}(t)} \tau_{k,W} + \sum_{c_{k,T} \in \Omega^{s}_{k,T}(t)} \tau_{k,T} \right)$$

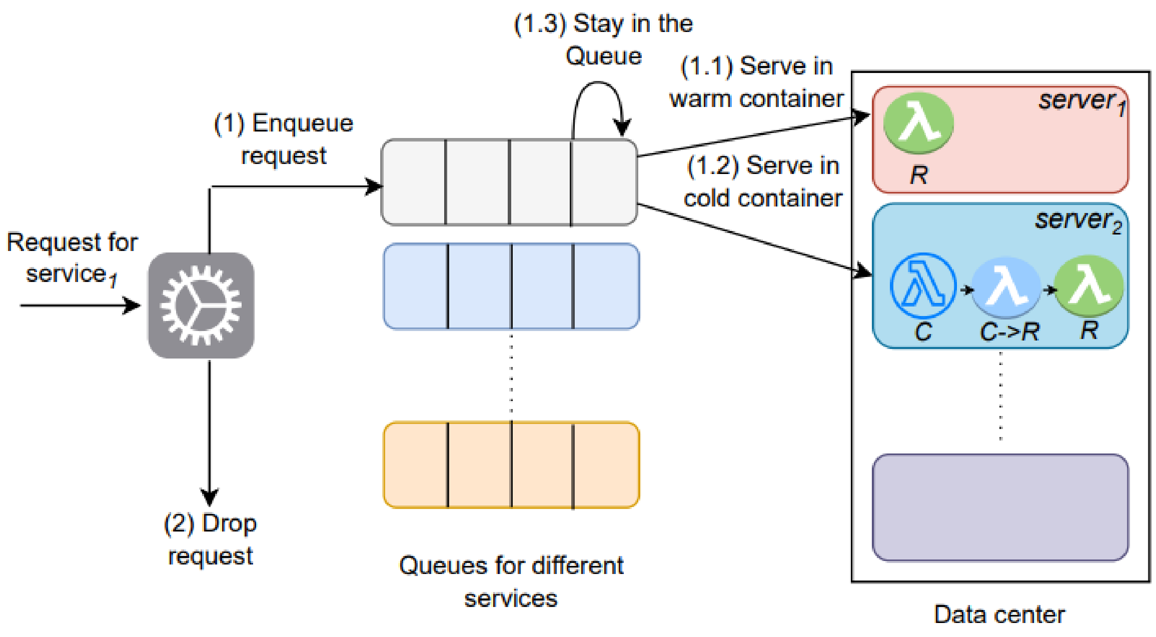


*Figure 2-1: Various decisions the orchestrator can take upon a new request arrival.*

### 2.3.1 Problem formulation of new request arrival event

At $t$, upon the arrival of a new request, $r_{k,i}$, the orchestrator makes one of the decisions on MS $k$ as listed below

(1) The orchestrator decides to queue $r_{k,i}$ into the queue $Q_k$. Let $t_{r_{k,i}}$ be the queue admission time of $r_{k,i}$ which is set to $t$. The decision of queueing $r_{k,i}$ is expressed by setting $q_{r_{k,i}} = 1$. While

queueing $r_{k,i}$, the orchestrator decides $\mu_{r_{k,i}}$, the CPU speed to allocate to it. After queueing $r_{k,i}$, the orchestrator has three options:

a. To schedule the first request in the queue $r_{k,j}$ on an existing warm container $c_{k,W}(t)$ in server $s \in \mathcal{S}$. We denote this decision by setting $x_{r_{k,j},c_{k,W}} = 1$ (Case 1.1 of Figure 2-1). The decision variable $y_{r_{k,j},s} = 1$ indicates that the request is served in server $s$. The container $c_{k,W}(t)$ moves to state $R$ (denoted by $c_{k,j,R}(t)$) with negligible start-up latency and serves $r_{k,j}$. The CPU and the memory allocated to $c_{k,j,R}$ are set equal to, respectively, the CPU allocated to $r_{k,j}$ and to MS $k$'s memory requirements. Also, the orchestrator revises the CPU speed allocated to all the requests in the queue. For the $n$-th request $r_{k,j+n}$ in $Q_k$, with $j$ corresponding to the first request, let $\mu_{r_{k,j+n}}$ be the revised CPU speed allocated to it, where $n \in \{0,1,\dots,|Q_k|-1\}$.

b. To schedule the first request to handle in the queue $r_{k,j}$ on a cold container in server $s$ ($y_{r_{k,j},s} = 1$), denoted by $z_{r_{k,j},c_{k,C}} = 1$ (Case 1.2 of Figure 2-1). That is, at $t$, a new container is created and starts the transition to $R$ ($c_{k,j,T_{C\to R}}(t)$). After the start-up latency of $\delta_{k,C}$, $r_{k,j}$ is run on $\mathrm{c_{k,j,R}}\left(\mathrm{t}+\delta_{k,C}\right)$. At $t$, the orchestrator sets CPU speed of $c_{k,j,R}\left(t+\delta_{k,C}\right)$ to $\mu_{r_{k,j}}$, while the memory allocated to $c_{k,j,R}\left(t+\delta_{k,C}\right)$ is same as the memory requirement of MS $k$. As before, the orchestrator revises the CPU speed allocated to all the queued requests.

c. Not to start a cold or warm container to serve the first request in the queue, denoted by setting $z_{r_{k,j},c_{k,C}} = 0$ and $x_{r_{k,j},c_{k,W}} = 0$, i.e., the first request in $Q_k$, $r_{k,j}$, remains in the queue for the time being (Case 1.3 of Figure 2-1), and the CPU speed assigned to queued requests is not revised.

(2) The orchestrator drops the new request $r_{k,i}$ (Case 2 of Figure 2-1). This decision is denoted by setting $q_{r_{k,i}} = 0$.

Let $\mathrm{m_k(t)} \in \mathcal{M}_k(\mathrm{t})$ denote the time at which currently running, transiting, and warm containers are available to serve the first request to handle in the queue, relative to current time $t$. For the first request to handle, $r_{k,j}$, let $\mu_{r_{k,j}}$ and $\Delta t_j$ be its revised CPU speed allocation and the time at which an existing container will start the execution of $r_{k,j}$ relative to $t$. The orchestrator will run the first request $r_{k,j}$ waiting in the queue on the container that can start serving the request $r_{k,j}$ at the earliest, i.e., at $\Delta t_j = \min\left(\mathcal{M}_k(t)\right)$. The queueing delay experienced by $r_{k,j}$ so far is $t - t_{r_{k,j}}$, where $t_{r_{k,j}}$ is the queue admission time of the request $r_{k,j}$. Additionally, $r_{k,j}$ will stay for $\Delta t_j$ amount of time in the queue before being served. To get served within the target delay, the revised CPU speed of $r_{k,j}$ is calculated by the following equation, $\mu_{r_{k,j}} = w_k / \left[D_k - \left(t - t_{r_{k,j}} + \Delta t_j\right)\right]$. When $r_{k,j}$ runs on this container, the container's new residual time to finish the execution will be equal to $r_{k,j}$'s processing time. Let $\mathcal{M}_k\left(\Delta t_j\right)$ be the set of updated residual times of the running and transiting containers at $\Delta t_j$ to finish their assigned tasks, which is

$$\mathcal{M}_k(\Delta t_1) = \{m_k(t) - \Delta t_1, \forall m_k(t) \in \mathcal{M}_k(t) \setminus \min\left(\mathcal{M}_k(\mathrm{t})\right)\} \cup \{\frac{w_k}{\mu_{r_{k,1}}}\}$$

where $m_k(t)$ is the generic element of set $\mathcal{M}_k(t)$.

Similarly, with respect to the current time $t$, the second request to handle $r_{k,j+1}$ will be served at $\Delta t_{j+1} = \Delta t_j + \min\left(\mathcal{M}_k\left(\Delta t_j\right)\right)$, The revised CPU speed of $r_{k,j+1}$ is given by: $\mu_{r_{k,j+1}} = w_k / \left[D_k - \left(t - t_{r_{k,j+1}} + \Delta t_{j+1}\right)\right]$. Set $\mathcal{M}_k\left(\Delta t_{j+1}\right)$ containing the time relative to $\Delta t_{j+1}$ at which the running and transiting containers can handle the third request in $Q_k$ is,

$$\mathcal{M}_k\left(\Delta t_{j+1}\right) = \{m_k\left(\Delta t_j\right) - \left(\Delta t_{j+1} - \Delta t_j\right), \forall m_k\left(\Delta t_j\right) \in \mathcal{M}_k\left(\Delta t_j\right) \setminus \min\left(\mathcal{M}_k\left(\Delta t_j\right)\right)\} \cup \{\frac{w_k}{\mu_{r_{k,j+1}}}\}.$$

Generalizing the above equation, the $n$-th request to handle in the queue $r_{k,j+n}$ will be served at $\Delta t_{j+n} = \Delta t_{j+n-1} + \min\left(\mathcal{M}_k(\Delta t_{j+n-1})\right)$. By revising the CPU speeds, the time at which the running and transiting containers can handle request $n$ in $Q_k$ relative to $\Delta t_{j+n}$ is given by,

$$\mathcal{M}_k(\Delta t_{j+n}) = \{m_k(\Delta t_{j+n-1}) - (\Delta t_{j+n} - \Delta t_{j+n-1}), \forall m_k(\Delta t_{j+n-1}) \in \mathcal{M}_k(\Delta t_{j+n-1}) \setminus \min\left(\mathcal{M}_k(\Delta t_{j+n-1})\right)\} \cup \{\frac{w_k}{\mu_{r_{k,j+n}}}\}.$$

Thus, the residual queueing delay of the $n$-th queued request $r_{k,j+n}$ is given by $\Delta t_{j+n}$, where $n \in \{0,1,\dots,|Q_k|-1\}$.

The queueing delay of the newly arrived request $r_{k,i}$ will be the sum of the time at which the last request waiting in the queue will start executing, and the updated remaining time of the running and transiting containers at $\Delta t_{j+|Q_k|-1}$ is computed by,

$$d_{r_{k,i}}(\mathcal{M}_k(t)) = \Delta t_{j+|Q_k|-1} + \min\left(\mathcal{M}_k(\Delta t_{j+|Q_k|-1})\right).$$

We use the above estimated worst-case queueing delays in Sec. 2.3.3 for making decisions on starting a new container from cold/warm state, and in Secs. 2.3.3 and 2.3.4 to revise the allocated CPU speeds.

### 2.3.2 Pre-computation of power consumption

Upon the arrival of a new request $r_{k,i}$, the orchestrator needs to make a decision that minimizes the power consumption of the servers in the data center. The optimal policy for this problem is non-trivial. Keeping the requests longer in the queue may reduce the total number of running and transiting containers in the system. But this policy might result in a higher CPU speed allocation to these running containers and hence higher overall power consumption. On the other hand, if we keep the requests for a shorter duration in the queue, we may have a higher number of transiting and running containers with lower CPU speed allocated, again resulting in higher power consumption. Moreover, this policy will reduce the reusability of the same container to serve other requests. Thus, in the objective function, considering just the CPU cycles assigned to currently running and transiting containers is not enough; we also need to account for the impact of queued requests on the power consumption.

To account for the impact of queued requests on the power consumption, we pre-compute the possible power consumptions based on the various decisions the orchestrator can make for the first request in the queue and the newly arrived request. At $t_1$, the orchestrator knows the request execution end and transition complete events for the existing transiting and running containers. Let $\mathcal{L} = \{t_1, t_2, \dots\}$ be the set of the time instants at which these known events will occur, including the current arrival at $t_1$. Based on the decisions made for the new request and the first request in the queue, the orchestrator updates $\mathcal{L}$ to include the time of occurrences of events of request execution end for the queued requests as well.

Let $\mathcal{L}_q$ be the auxilliary set containing the time at which each of the known events will occur if the orchestrator decides to queue the newly arrived request and not to serve the first request in the queue (i.e., $q_{r_{k,i}} \backslash mathord = 1$). Then the power consumption $P_q$ across all the servers in the data center is $P_q = \sum_{\hat{s}\in\mathcal{S}} \sum_{i=1}^{|\mathcal{L}_q|-1} \int_{t_i}^{t_{i+1}} \left(P_{\text{idle}} + P(\lambda_{\hat{s},t_i,q})\right) dt$ where $\lambda_{\hat{s},t_i,q}$ is the load of server $\hat{s}$ at $t_i$ if the decision is to queue the newly arrived request and not to serve the first request in the queue for the time being.

Similarly, the orchestrator pre-computes the power consumption for the other decisions. If the orchestrator decides to queue the new request and start a warm container on server $s$ to serve the first request in the queue, the power consumption $P_{x,s}$ across all the servers in the data center is $P_{x,s} = \sum_{\hat{s}\in\mathcal{S}} \sum_{i=1}^{|\mathcal{L}_{x,s}|-1} \int_{t_i}^{t_{i+1}} \left(P_{\text{idle}} + P(\lambda_{\hat{s},t_i,x,s})\right) dt$, where $\lambda_{\hat{s},t_i,x,s}$ is the load of server $\hat{s}$ at $t_i$ and $\mathcal{L}_{x,s}$ is the auxilliary set containing the time of occurrence of known events. If the decision is to use a cold container on server $s$ to serve the first request in the queue, then the pre-computed power consumption is represented by

$$P_{z,s} = \sum_{\hat{s}\in\mathcal{S}} \sum_{i=1}^{|\mathcal{L}_{z,s}|-1} \int_{t_i}^{t_{i+1}} \left(P_{\text{idle}} + P\left(\lambda_{\hat{s},t_i,z,s}\right)\right) dt \ .$$

Finally, the pre-computed power consumption if the new request is not queued is represented by

$$\mathrm{P} = \sum_{\hat{s}\in\mathcal{S}} \sum_{i=1}^{|\mathcal{L}|-1} \int_{t_i}^{t_{i+1}} \left(P_{\text{idle}} + P\left(\lambda_{\hat{s},t_i}\right)\right) dt \,.$$

### 2.3.3 Objective function for arrival event

The objective is to minimize the power consumption across all servers as well as the expected power consumption in the future, based on the currently known events (i.e., request end and transition complete) while minimizing the number of dropped requests. Notice that the number of requests served can be maximized by adding a high penalty $F$, if the orchestrator decides to drop the newly arrived request. Thus, upon the arrival of a new request, $r_{k,i}$, the orchestrator should solve the following problem:

$$\min_{\{y,x,q,z,\{\mu\}\}} \Bigg[ q_{r_{k,i}} \cdot \left(1 - x_{r_{k,j},c_{k,W}}\right) \cdot \left(1 - z_{r_{k,j},c_{k,C}}\right) \cdot P_q + q_{r_{k,i}} \cdot x_{r_{k,j},c_{k,W}} \cdot \left(1 - z_{r_{k,j},c_{k,C}}\right) \cdot \sum_{s\in\mathcal{S}} y_{r_{k,j},s} \cdot P_{x,s}$$
$$+ q_{r_{k,i}} \cdot \left(1 - x_{r_{k,j},c_{k,W}}\right) \cdot z_{r_{k,j},c_{k,C}} \cdot \sum_{s\in\mathcal{S}} y_{r_{k,j},s} \cdot P_{z,s} + \left(1 - q_{r_{k,i}}\right) \cdot P \Bigg] + \left(1 - q_{r_{k,i}}\right) \cdot F$$

subject to the constraints below (formal expressions can be found in [10], and are omitted for the sake of readability):

(1) At any $t$, the CPU cycles used by the pre-existing running and transiting containers as well as by the newly scheduled container do not exceed the server capability.

(2) The memory allocated to pre-existing containers in warm, running, and transiting states and newly scheduled container cannot exceed the server capability.

(3) If the decision is to run the first request in the queue $r_{k,j}$ on a warm container, the total delay experienced by $r_{k,j}$ cannot exceed the target delay of MS $k$.

(4) If the decision is to run the first request in the queue $r_{k,j}$ on a cold container, the total delay experienced by $r_{k,j}$ cannot exceed the target delay of MS $k$.

(5) If the decision is to queue the new request, the total delay the request is going to experience in the worst case cannot exceed $D_k$. Note that the queueing delay the new request is going to experience depends on the decisions made on the first request in the queue.

(6) The CPU speed allocated to all queued requests must be revised depending on the decisions made on the first request in the queue and the MS target delay.

(7) If $x_{r_{k,j},c_{k,W}} = 1$ or $z_{r_{k,j},c_{k,C}} = 1$, we must specify on which server the warm or cold container has started its transition. Further, when $x_{r_{k,j},c_{k,W}} = 1$, $y_{r_{k,j},s}$ can be set only for that server which has a warm container.

(8) $r_{k,j}$ is either scheduled in a warm container or in a cold container or later.

### 2.3.4 Problem formulation for request execution ends

At time $t$, when the running container $c_{k,i,R}$ finishes its current execution on server $\hat{s} \in \mathcal{S}$, the orchestrator makes one of the following decisions on MS $k$,

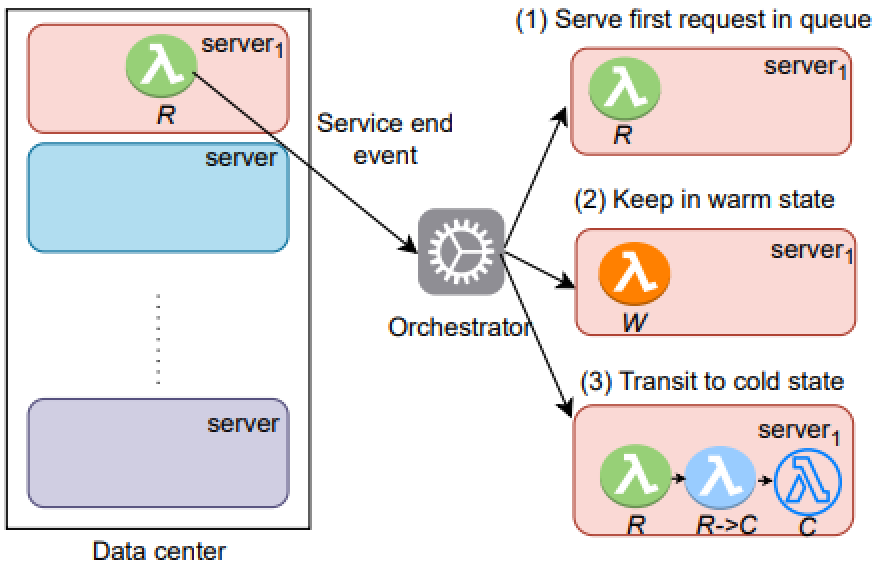


*Figure 2-2: Decisions the orchestrator can make upon request execution ends.*

(1) To schedule the first request to handle in the queue $r_{k,j}$ on the container $c_{k,i,R}(t)$, as shown in case 1 of Figure 2-2. Note that this decision is possible since we assume that the transition time of a container from the state $R$ to the state $W$ is negligible. This decision is denoted by setting $x_{r_{k,j},c_{k,i,R}} = 1$ and $e_{c_{k,i,R}} = 2$. At this stage, $c_{k,i,R}(t)$ is referred to as $c_{k,j,R}(t)$, and CPU speed allocated to $c_{k,j,R}(t)$ is updated to $\mu_{r_{k,j}}$.

(2) Not to schedule the first request to handle in the queue $r_{k,j}$ on the container $c_{k,i,R}(t)$ denoted by setting $x_{r_{k,j},c_{k,i,R}}$ =0. Further, to set $x_{r_{k,j},c_{k,i,R}} = 0$, the orchestrator needs to revise the CPU speed allocated to the queued requests such that they are served within the target delay of the service $k$. For $n$-th queued request, let $\mu_{r_{k,j+n}}$ be its revised CPU speed, where $n \in \{0,1,\dots,|Q_k|-1\}$. Additionally, the orchestrator makes one of the following decisions regarding $c_{k,i,R}(t)$:

   a. To keep $c_{k,i,R}(t)$ in state $W$, denoted by $e_{c_{k,i,R}} = 1$ (case 2 in Figure 2-2). This is useful to reduce the start-up latency of future requests and serve them with low CPU speed.

   b. To delete the container $c_{k,i,R}(t)$, denoted by $e_{c_{k,i,R}} = 0$, and the container $c_{k,i,R}(t)$ enters state $T_{R\to C}$, and it will be removed from the system once it reaches state $C$. This scenario is represented in case 3 of Figure 2-2. Without knowing the future arrival pattern, deciding whether to keep the container in the state $W$ or $C$ is challenging. To make this decision, we considered probability $p_0$ of no arrivals within $t$ and $t + \delta_{k,C}$, and added this as a penalty in the objective function.

The orchestrator decides to set $\hat{x}_{r_{k,j},c_{k,i,R}} = 0$, only if we can meet the target delay of the queued requests with existing running and transiting containers of MS $k$ excluding $c_{k,i,R}(t)$.

For the request execution end event, let $P_{\hat{x}}$ be the pre-computed power consumption if the decision is to set $\hat{x}_{r_{k,j},c_{k,i,R}} = 1$. Let $P_{e_1}$ and $P_{e_0}$ be the pre-computed power consumptions if the orchestrator decides to keep $c_{k,i,R}(t)$ in warm and cold states, respectively. We follow the same procedure described in 2.3.2 to pre-compute these power consumptions. Then the objective function is given by

$$\min_{\{x,e,\{\mu\}\}} \hat{x}_{r_{k,j},c_{k,i,R}} \cdot P_{\hat{x}} + \left(1-\hat{x}_{r_{k,j},c_{k,i,R}}\right)\cdot e_{c_{k,i,R}} \cdot P_{e_1} + \left(1-\hat{x}_{r_{k,j},c_{k,i,R}}\right)\cdot\left(1-e_{c_{k,i,R}}\right)\cdot P_{e_0} - \left(1-\hat{x}_{r_{k,j},c_{k,i,R}}\right)$$
$$\cdot\left[e_{c_{k,i,R}}\cdot(1-p_0) + \left(1-e_{c_{k,i,R}}\right)\cdot p_0\right]$$

subject to the following constraints (a formal expression of the constraints can be found in [10] and is omitted for the sake of readability).

(1) The CPU cycles used by currently running, transiting, and CPU cycles consumed by newly scheduled container cannot exceed the server available CPU capacity.
(2) The memory allocated to pre-existing containers in warm, running, and transiting states and newly scheduled container cannot exceed the server memory.
(3) The deadline of queued requests must be met with revised CPU speeds, if the orchestrator sets $\hat{x}_{r_{k,j},c_{k,i,R}} = 0$.
(4) If $\hat{x}_{r_{k,j},c_{k,i,R}} = 0$, either the container is kept in $W$ state ($e_{c_{k,i,R}} = 1$) or it is destroyed ($e_{c_{k,i,R}} = 0$); if $\hat{x}_{r_{k,j},c_{k,i,R}} = 1$, the container starts serving request $r_{k,j}$ once it enters state $R$, i.e., $e_{c_{k,i,R}} = 2$.

The optimization problem for new request arrival and request execution end events is in the form of Mixed Integer Non-Linear Program (MINLP) and, hence, NP-hard [11].

## 2.4 Threshold-based Algorithm

The key idea is to set a threshold on the number of unhandled requests in the queue and create a running container from cold/warm state only when the number of unhandled requests in the queue exceeds the defined threshold. This idea reduces the number of running and transiting containers in the system, thereby minimizing overall energy consumption.

Upon the arrival of a new request $r_{k,i}$ for MS $k$, the orchestrator decides whether to start a new container based on the number of unhandled requests in the queue $Q_k$. Suppose the number of unhandled requests in $Q_k$ is greater than or equal to the threshold. In that case, the orchestrator first verifies whether the first request to handle in the queue $r_{k,j}$ can be served on an existing warm container. If so, it determines the CPU speed to allocate to the warm container so as to meet the target delay using the expression $\mu_{r_{k,j}} = w_k / \left[D_k - \left(t - t_{r_{k,j}}\right)\right]$. All the eligible servers having the warm containers of MS $k$ and enough computing resources to serve $r_{k,j}$ are sorted in the increasing order of their remaining CPU speeds. The algorithm then selects the first server from the sorted set to serve $r_{k,j}$.

If instead there are no warm containers in any server, the orchestrator determines whether the request can be served in the cold container, as well as the CPU speed to allocate to the container using: $\mu_{r_{k,j}} = w_k / \left[D_k - \left(t - t_{r_{k,j}} + \delta_{k,C}\right)\right]$. All the eligible servers having enough computing and memory resources to serve $r_{k,j}$ in a cold container are sorted in the increasing order of their remaining CPU speeds. The orchestrator selects the first server in the sorted set to serve $r_{k,j}$. The request is dropped if it cannot be served in either warm or cold containers. Finally, after making one of the above decisions, $r_{k,i}$ is enqueued.

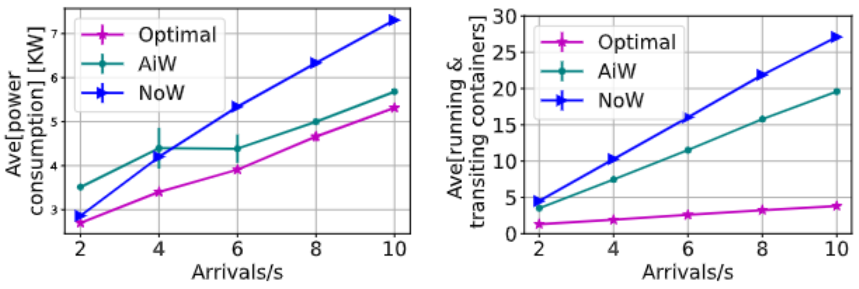
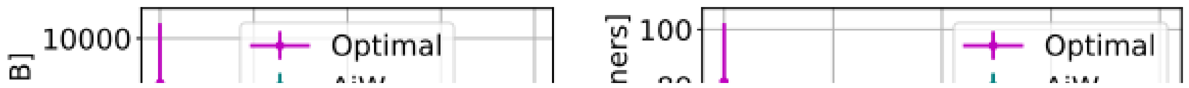


*Figure 2-3: NoW and AiW vs. optimal: average power consumption (left); average number of running and transiting containers (right).*

At time $t$, when an existing container $c_{k,i,R}(t)$ finishes its current execution on server $\hat{s}$, we again follow a threshold-based approach to make decisions on the container and first request to handle in the queue $r_{k,j}$. If the number of unhandled requests waiting in the queue exceeds the threshold, the orchestrator decides whether $c_{k,i,R}(t)$ can be reused to serve the first request in $Q_k$, exploiting $\mu_{r_{k,j}} = w_k / \left[D_k - \left(t - t_{r_{k,j}}\right)\right]$ to determine the CPU speed to allocate to $c_{k,i,R}(t)$. If the server has enough computing and memory resources, then $r_{k,j}$ is served on $c_{k,i,R}(t)$. Otherwise, the first request is dropped, and we have two options: i) *NoW*: after serving a request, if the queue is empty, the container is always transitioned to state $C$, ii) *AiW*: after serving a request, if the queue is empty, the container is always kept in state $W$.

## 2.5 Performance Evaluation

Here, we evaluate the performance of the optimal solution against the simple threshold based queueing solution. Due to the complexity of the optimum, we focus on a small-scale scenario and derive the optimum using Gurobi. In particular, we consider only one type of MS, and that requests arrive according to a Poisson process with varying rate. Further, we consider a simple Python function as MS, and, using OpenWhisk, we measured its start-up latency, which resulted to be $\delta_{k,C} = 1.5$s. The MS workload requirement is fixed to $w_k = 3$G clock cycles. In the simulation setup, we set the number of active servers in the data center to 4. Further, we pre-create three warm containers per server at the beginning of the simulation and the simulation is carried out for 1,000s. We set the target delay for the MS requests to 2s.



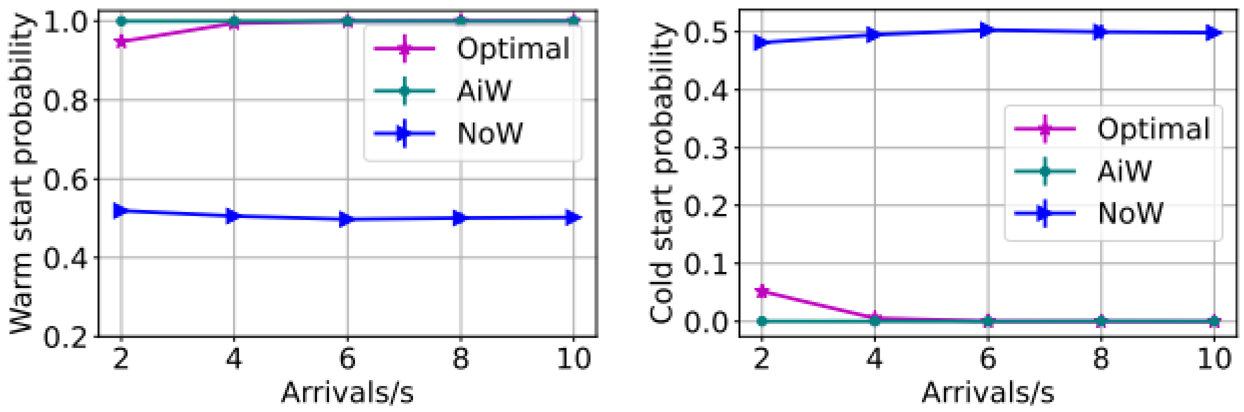


*Figure 2-4: Performance of NoW and AiW vs. optimal, as the arrival rate of service requests varies: average memory consumption (left); average number of warm containers (right).*

*Figure 2-5: Performance of NoW and AiW vs optimal: warm (left) and cold (righ) start probabilities.*

Surprisingly, Figure 2-3(left) shows that the power consumption of AiW is comparable with that of the optimum. Optimum uses significantly fewer containers to serve the requests; however, the containers run at higher CPU speeds due to queueing delay. Conversely, AiW uses a higher number of containers to serve the requests, but, thanks to the threshold on the number of requests in the queue, these containers run at lower CPU speeds. As a result, the overall CPU load at the data center is comparable to the optimum. In the case of NoW, the power consumption is higher than both AiW and the optimum because of the higher number of cold starts. Further, Figure 2-4 presents the measured average memory consumption and the number of warm containers across all the servers in the data center for various arrival rates. Note that warm, running, and transiting containers contribute to the server's memory utilization. For lower arrival rates ($< 6/s$), the memory consumption in the optimum case is higher because the containers are kept in the warm state after serving the requests, but, due to the low arrival rate, they are never used again. Consequently, the memory consumption of NoW is the lowest because it never keeps the container in the warm state.

Finally, Figure 2-5 depicts the cold and warm start probabilities for optimum, AiW, and NoW. The warm start probabilities of AiW and optimum are close to one because the containers are either used again to serve the requests or kept warm to serve future requests. However, the NoW approach has a lower warm start probability because the container is always transitioned to the cold state if the queue is empty.

# 3 Video Bitrate Adaptation in MEC-empowered networks (UOA)

## 3.1 Introduction

In the previous deliverable (d.2.2), we presented the SoTA and our proposed solution for streaming media over MEC-empowered networks. In the current deliverable (d.2.3) we recap our previous work and show the results of a novel scheme in video bitrate adaptation in MEC-empowered networks. Finally, we evaluate the performance of our solution over different approaches.

## 3.2 HTTP Adaptive Video Streaming

### 3.2.1 Concept

Adaptive streaming is a new streaming approach that it is designed to deliver multimedia to the user in the most efficient way possible and in the highest available quality for each specific user. Video content is encoded at different quality levels and each quality level is determined by its corresponding average bitrate. In order to adaptively stream media, the media is split up into chunks, so the content is divided in segments that have a typical duration (most commonly one to ten seconds). Each segment can be decoded independently of other segments. An HTTP Adaptive Streaming (HAS) client initiates a new session by downloading a manifest file. This manifest file is used to hold the information and description of the various streams and the bandwidths they are associated with. Based on the network conditions and the current buffer-filling level, the Rate Determination Algorithm (RDA) in the HAS client determines the quality for the next segment download. The objective of the RDA is to optimize the global QoE determined by the occurrence of video freezes, the average quality level, and the frequency of quality changes. This procedure is shown at Figure 3-1.

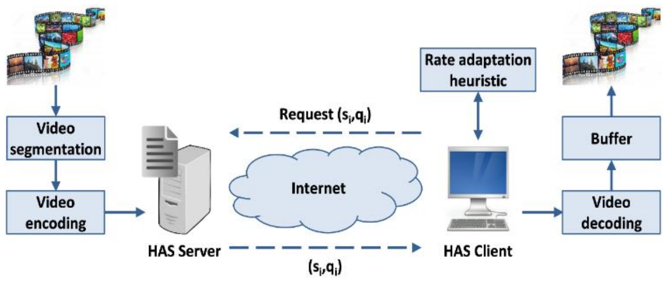


*Figure 3-1: The concept of HTTP Adaptive Streaming*

The main advantage of HAS over traditional download and common real-time streaming is that it is able to adapt any video quality according to the available bandwidth so as to avoid video stallings. Consequently, HAS facilitates video streaming over a best-effort network. In addition, HTTP- based video streams can easily traverse firewalls and reuse the already deployed HTTP infrastructure such as HTTP servers, HTTP proxies, and Content Delivery Network (CDN) nodes. Because of these advantages, major players such as Microsoft, Apple, Adobe and Netflix have massively adopted the adaptive streaming paradigm [13].

Adaptive streaming is preferred over traditional progressive streaming for two main reasons. The first is quality, meaning that a video that is only 1280 x 720 will never play at correct quality levels on a screen that is 1920 x 1080. It will be stretched and pixelated. On the contrary, adaptive streaming allows the video provider to create a different video for each of the resolutions that he or she wishes to target.

The second problem is buffering which is an unpleasant situation for all the users when the video pauses sequentially. Buffering happens commonly when there is an unstable Internet connection. Consequently, the video has to pause several times because it cannot be downloaded quickly enough, in order to wait for more data and then start downloading again. Most videos play at 24 frames per second, so the Internet connection needs to download at least 24 frames every second to avoid buffering. This problem is very common, especially on mobile devices, where the connection can vary greatly depending on the user's location and can cause bad user experience.

Adaptive streaming can resolve this second buffering problem by "adapting" to the speed of the user's Internet connection. A small video can be downloaded faster than a large video, so if a user has a slow Internet connection, an adaptive video stream will switch to a smaller video file size to keep the video playing. From a user's perspective, it is preferable to watch a few minutes of lower quality video in order to avoid buffering, than to sit and watch a spinning icon until the stream catches up [14].

MPEG-DASH (Dynamic Adaptive Streaming Over HTTP) is a standard flexible bitrate streaming technique. MPEG has developed quite a few extensively used multimedia standards, including MPEG-21, MPEG-7, MPEG-4 and MPEG-2. Their newest standard MPEG-DASH is an effort to resolve the intricacies of media delivery to various devices with an integrated common standard. When media content is delivered from conventional HTTP web servers, MPEG-DASH empowers high quality streaming of this media content over the Internet.

### 3.2.2 Implementation

A media streaming scenario between a simple HTTP server and a HAS client take place as follows. The multimedia content is captured and stored on an HTTP server and is delivered using HTTP. The content exists on the server in two parts: Media Presentation Description (MPD), which describes a manifest of the available content, its various alternatives, their URL addresses, and other characteristics; and segments, which contain the actual multimedia bitstreams in the form of chunks, in single or multiple files.

The MPEG-DASH Media Presentation Description (MPD) is basically an XML document containing information about media segments, their relationships and information necessary to choose among them, and other metadata that may be needed by clients. MPD's structure is as follows [15]:

- Periods, contained in the top-level MPD element, describe a part of the content with a start time and duration. Multiple Periods can be used for scenes or chapters, or to separate ads from program content.
- Adaptation sets, which contain a media stream or a set of media streams.
- Representations allow an Adaptation Set to contain the same content encoded in different ways. In most cases, Representations will be provided in multiple screen sizes and bandwidths in order to allow clients to request the highest quality content that they can play without waiting to buffer or wasting bandwidth.
- Sub representations contain information that only applies to one media stream in a Representation. They also provide information necessary to extract one stream from a multiplexed container, or to extract a lower quality version of a stream.
- Media segments are the actual media files that the DASH client plays, generally by playing them back-to-back as if they were the same file. Media Segment locations can be described using BaseURL for a single-segment Representation, a list of segments (SegmentList) or a template (SegmentTemplate).
- Index Segments come in two types: one Representation Index Segment for the entire Representation which is always a separate file, or a Single Index Segment per Media Segment which can be a byte range in the same file as the Media Segment.

In order to play the content, the DASH client first requests the MPD file from the server. The MPD can be delivered in various ways such as via HTTP or email. By parsing the MPD, the DASH client knows all the information about the program timing, media-content availability, media types, resolutions, minimum and maximum bandwidths, and the existence of various encoded alternatives of multimedia components, accessibility features and required digital rights management (DRM), media-component locations on the network, and other content characteristics. The client capitalizes the information and selects the appropriate encoded alternative in order to start streaming the content by fetching the segments using HTTP GET requests.

After appropriate buffering to allow for network throughput variations, the client continues fetching the subsequent segments but keeps on monitoring the network bandwidth fluctuations. Depending on its measurements, the client decides how to adapt to the available bandwidth by fetching segments of different bitrates to avoid buffering.

The MPEG-DASH specification only defines the MPD and the segment formats. The delivery of the MPD and the media encoding formats containing the segments, as well as the client behavior for fetching, adaptation heuristics, and playing content, are outside of MPEG-DASH's scope [16].

## 3.3 Experiments' Setup

### 3.3.1 NETWORK TOPOLOGY – SCENARIO

The main actors in our implementation are three, as shown at Figure 3-1:

1) The client who wants to see a video at the best available quality.

2) The proxy or cache server, which has stored some of the video segments according to a caching algorithm, limited by its buffer size, as well as the manifest file.

3) The main server, which is equipped with all the video resolution chunks.

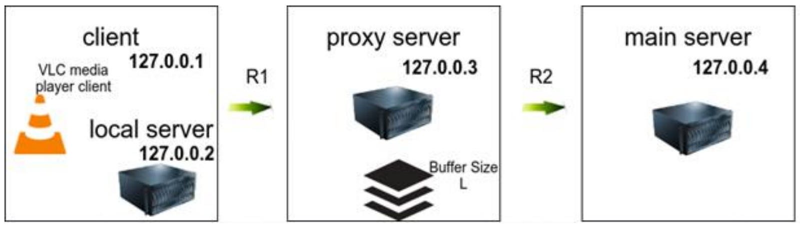


*Figure 3-2: Implementation with an intermediate cache server*

The cache server is designed to be a simple repository for our video content, providing local clients with accelerated access to cached files. The main server is intended to provide a content management service which runs entirely behind the user interface of our custom application, creating the backhaul connection with the cache server, and containing all the given resolutions and the coding rates.

Specifically, the cache server will contain a few segments from some of the video resolutions, meaning that it will have some caching capabilities. So, the basic connection will be between the client and the cache server. The backhaul connection, between the cache and the main server, will be utilized so the cache server can be able to receive chunks with a fixed rate from the main server, that contains all the segments. The number of the chunks that will be stored in the cache server depends on its available buffer size ***L*** and the implemented caching algorithm. The link rate between client and cache will be referred as ***R1*** and the link rate between cache and main server as ***R2***.

The roles of the actors of our problem statement are the following:

- Client: The client requests a video. The video has to be displayed in the best resolution, based on the available bandwidth. Practically, the client requests the manifest file from the cache server which contains all the available video details. For this reason, the cache must have available information about the manifest.

- Cache Server: The cache server has some chunks of the video in specific resolutions according to its caching policy. It interacts both with the client at one end and with the main server at the other end. The cache server needs to set up a session with the main server that has all the video chunks in order to cache as many chunks as it can. As long as the segments are stored in the cache, the client requests the video and a new fronthaul connection with the cache server is set up. Each time a request is received, the cache checks the cache buffer and if there is a cache hit, the cache server sends it directly to the client and the channel cache-main is not utilized at all. Otherwise, in the case of a cache failure, the cache asks from the main, the wanted chunk in order to serve the client's request.
- Main Server: The main server has all the possible resolutions of the video (360p, 480p, 720p, 1080p, 1440p, 2160p) that are defined by the manifest file. This server interacts only with the cache server providing specific video chunks. In our setup, its role is only to give the manifest file to the cache (only in cases that the cache isn't already equipped with it) and serve the client when there is a cache miss. In case that the cache has all the available chunks of the video, meaning that it can fully serve the client, the backhaul link is not used at all.

The flow of information between these nodes is depicted at Figure 3-3.

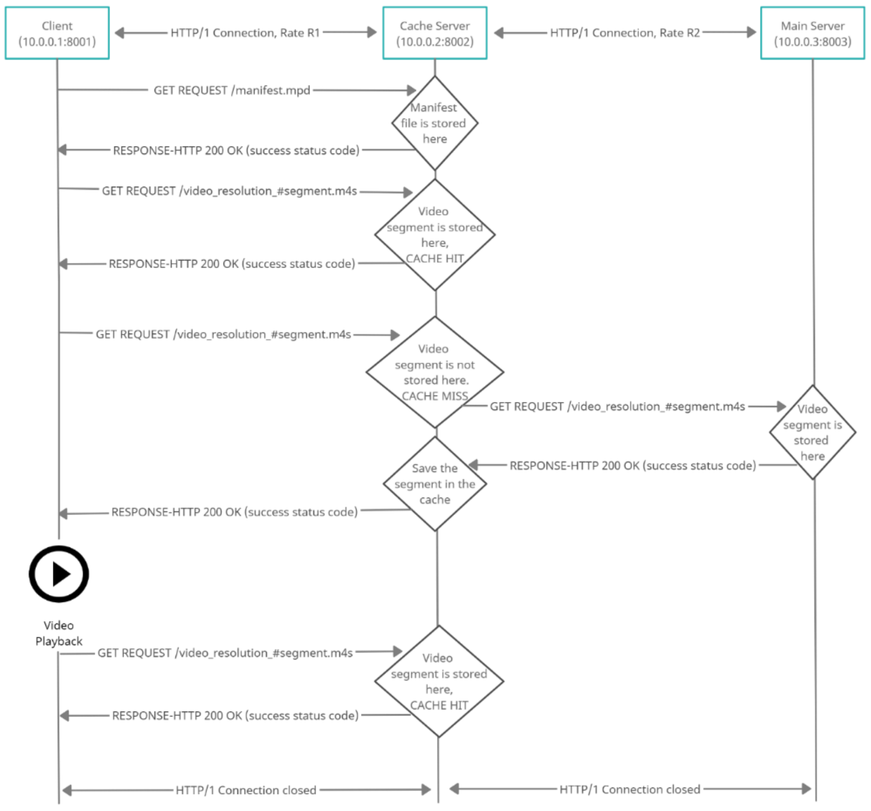

Client (10.0.0.1:8001)
HTTP/1 Connection, Rate R1
Cache Server (10.0.0.2:8002)
HTTP/1 Connection, Rate R2
Main Server (10.0.0.3:8003)
GET REQUEST /manifest.mpd
Manifest file is stored here
RESPONSE-HTTP 200 OK (success status code)
GET REQUEST /video_resolution_#segment.m4s
Video segment is stored here, CACHE HIT
RESPONSE-HTTP 200 OK (success status code)
GET REQUEST /video_resolution_#segment.m4s
Video segment is not stored here. CACHE MISS
GET REQUEST /video_resolution_#segment.m4s
Video segment is stored here
RESPONSE-HTTP 200 OK (success status code)
Save the segment in the cache
RESPONSE-HTTP 200 OK (success status code)
Video Playback
GET REQUEST /video_resolution_#segment.m4s
Video segment is stored here, CACHE HIT
RESPONSE-HTTP 200 OK (success status code)
HTTP/1 Connection closed
HTTP/1 Connection closed

*Figure 3-3: Flow of information*

We have also added an extra server to the topology, called Local Server (see Figure 3-2), where its objective is, in case of a smart caching policy aware of the contents in the cache, to download proactively more segments, directly from the Main Server, which are more likely to be selected by the media player. However, in our case, the existence of the Local Server is not of this use and acts only as a regular server that if the request can be served then it responds, and not in the opposite case.

The client part of our topology is implemented as both a VLC client and a Local Server and this is why in Figure 3-2 they exist in the same node. Moving on, the Proxy Server is the cache to our topology with the available buffer sizes 0, 100, 1000, 3000 MBs and the Main Server is the database that contains all the segments of all the different resolutions, 3000 MBs in total.

### 3.3.2 CONTENT PREPARATION

For our experiments, we need a video with a resolution of 3840x2160 (4K-UHD), which we recorded with a mobile phone. The video size is 1.8 GB. We create a bash.sh script, which converts every .mp4 file in the current folder and subfolder, to a multi-bitrate video in MP4-DASH. This script creates six different quality versions of the video, an audio file and an .mpd manifest file with all the appropriate information about the given resolutions. Then, we save the name of every file without the file's extension, and we start the procedure of converting each file to a multi-bitrate video in MPEG-DASH.

Our first step is to convert the audio file with the help of the ffmpeg commands about the audio and video codec [17],[18]. Next, we convert the video file to six different quality versions. The video codec that is used is H.264. We know that in this codec, the frames are organized in I,P, and B frames and I frames appear every 30 frames. A series of grouping frames is called GOP (Group of Pictures). The first frame in a GOP is an intra-frame called an I frame which contains most of the vital information for the following sequence of P frames which are forward predictive and B frames which are both forward and backward (bi-directional) predictive. A B frame estimates changes to the frame based on the previous and following frames. I frames contain more data than P and B frames.

Finally, we have all the required video and audio files, and using the MP4Box tool we can produce the following files:

- a manifest-mpd file, which will store all the needed information about the video. This file will contain all the available video resolutions, their bitrates and their size. In our implementation, a client, who wants to see a specific video, will request the manifest file.
- all the video segments are of 5 seconds for all the available resolutions and for the audio. For this reason, because our video is 5 minutes long, for each resolution the segments that are created are 61 (5 minutes = 300 seconds and 300 seconds / 5 seconds = 60 segments, and 1 more segment for initialization).
- an initialization file in order to start loading the segments (manifest_set1_init.mp4).

Table 3-1 below presents the video bitrates and respective sizes after the encoding and segmentation process.

*Table 3-1*

VIDEO BITRATES AND SIZES AFTER THE SEGMENTATION

| Resolution | VIDEO BITRATE (MBPS) | Size (MBs) |
|---|---|---|
| 2160p | 45 | 1700 |
| 1440p | 16 | 597.6 |
| 1080p | 8 | 299.1 |

| | | |
|---|---|---|
| 720p | 5 | 187.3 |
| 480p | 2.5 | 94.1 |
| 360p | 1 | 37.7 |

### 3.3.3 CACHING ALGORITHM

An important factor in our video streaming environment is the selected caching policy. For our work, we have a simple random caching algorithm. It downloads segments and saves them in the cache. The buffer size can be changed depending on the capacity of the cache.

The algorithm generates two random numbers, one for the resolution and one for the segment. So, the range to choose for the resolution is 360, 480, 720, 1080, 1440 or 2160, and the range to choose for the segment is [1,61], because each resolution has 61 segments. After, randomly choosing the file, it checks if it fits into the buffer. If it fits, it copies in from the main server's public folder. If it doesn't it moves to the next one only if the available buffer size is less than the size of the smallest available non-cached segment.

## 3.4 Numerical Results

### 3.4.1 SET OF EXPERIMENTS

For the purposes of emulation and analysis, we have conducted two different sets of experiments. The first set, given the name "Get Before Send", or GBS for short, is an approach where the data from the server is fetched serially, meaning that the second segment is sent to the client only after the first segment has been fetched. In the GBS case, if a cache hit takes place, the proxy server forwards the video segment (locally cached) directly. If a cache failure takes place, the proxy downloads the full video segment before forwarding it to the client. This approach is expected to show that the traffic R1 that we applied in the fronthaul channel and the traffic R2 that we applied in the backhaul channel were under-utilized.

The second set of experiments named "Send While Get", or SWG, as the name suggests, is more efficient than the first and the segments are fetched from the server simultaneously and are sent to the client without the need of any queue. In this approach, given the fact that it is smarter than the first, it is expected to show that the channels are utilized in a better way, leading us to better results in all the QoE metrics that we will export later after conducting a variety of experiments.

Therefore, in the following set of experiments, we use the SWG technique, which means that the Proxy Server does not wait for a single segment to be delivered to the Local Server, but while a segment is sent, the next one is already beginning the process of being requested and delivered. If a cache hit takes place, the proxy server forwards the video segment (locally cached) directly. If a cache failure takes place, we have configured the proxy server to read the proxy-to-main socket packet per packet and forward directly the packets upon reception.

In the SWG case there are no stallings observed, which means that the QoE is increased for the user. Also, the bitrate oscillations are fewer as compared to the GBS case, resulting in the client exploiting better the cache hits. However, the most significant observation is that both the fronthaul and the backhaul channel, are utilized to a better extent, due to the fact that no channel stays idle for a long period of time.

Below we present first the QoE metrics related to HAS as a function of the fronthaul channel R1 and as a function of the cache buffer size L.

### 3.4.2 QOE METRICS VS FRONTHAUL R1 CHANNEL

In the 3 next diagrams we can see how average resolution improves with the increase of the buffer size L, for different values of the fronthaul channel R1.

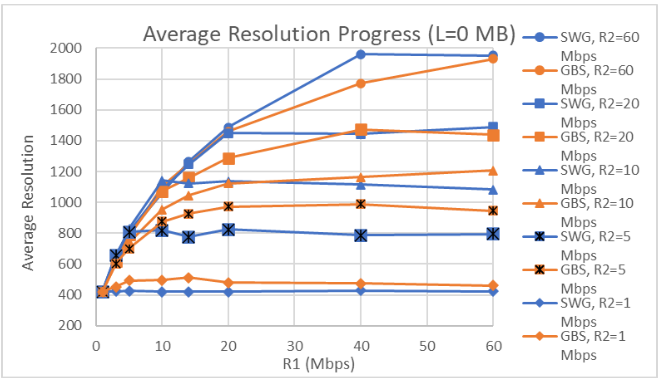


*Figure 3-4: Average Resolution vs R1 (L=0)*

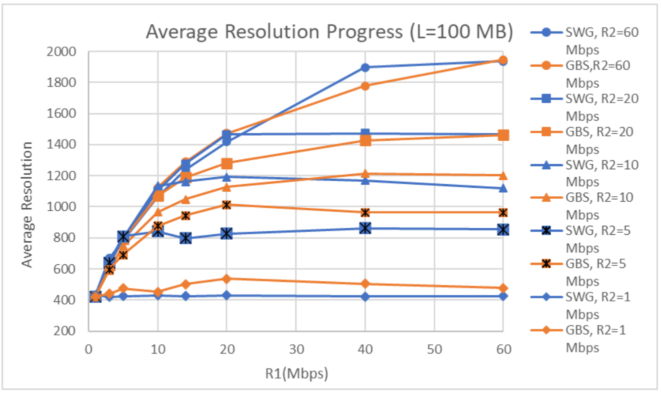


*Figure 3-5: Average Resolution vs R1 (L=1000 MB)*

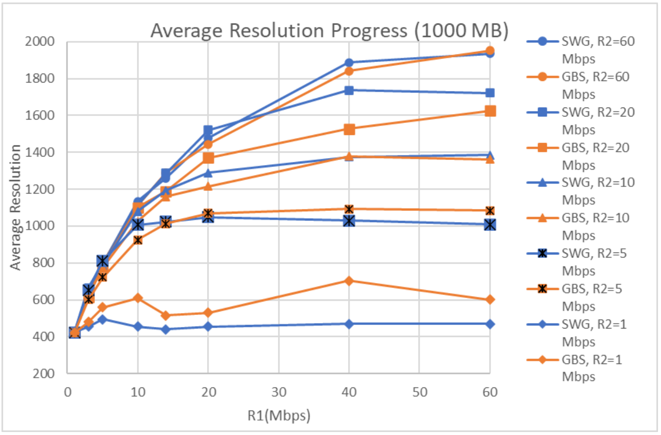


*Figure 3-6: Average Resolution vs R1 (L=1000 MB)*

After plotting the graphs, we infer the following conclusions:

- as it is reasonable, the higher the backhaul R2 (for the same R1), the higher the average resolution playing for the client.
- We notice a linear increase of the average resolution at every curve of the figure, while R2<R1. When R2>=R1 is valid, the value of the quality of the video stabilizes and cannot be further improved.
- For R2 = 1, 5 or 10 Mbps, a false image is given that the GBS approach gives better results. The fact in this case is that the HAS algorithm is over-optimistic, making the media player increase the quality. However, this increase is not justified in terms of bandwidth, and this is the reason of the continuous occurrences of bitrate oscillations and as a result, of stallings. On the other hand, the SWG approach has less stallings than the GBS one, so it is the one that gives better results in terms of QoE.
- The right approach that shows the dominance of the SWG technique is mostly shown at the curves of the figures for R2 = 20 and 60 Mbps, regardless of buffer size.

- While the buffer size L is increased, we can notice an increase of the average resolution for the same R1 and R2 values. The positive impact of caching is clearly noticeable.
- To make the example more understandable, in case of an R1 = 20 Mbps and R2 = 10 Mbps, if the buffer size L = 0, then the average resolution ranges between 1100 and 1200, but if the L = 1000, then the average resolution is almost increased at about 100 units, reaching a value close to 1300.

On the next three figures, we can see the average altitude for 2 distinct values of R1, in the 3 different buffer sizes that we have available in our experiment.

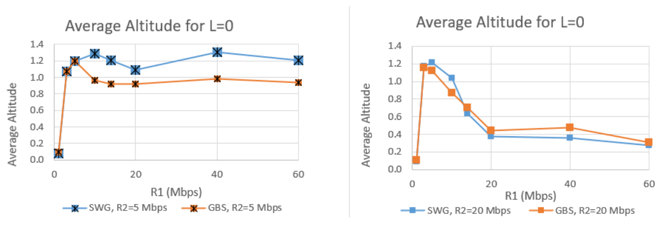


*Figure 3-7: Average Altitude VS R1 for R2=5,20 Mbps (L=0 MB)*

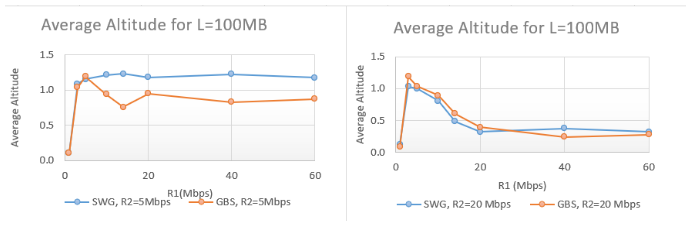


*Figure 3-8: Average Altitude VS R1 for R2=5,20 Mbps (L=100 MB)*

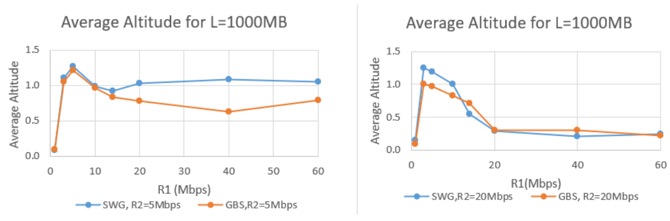


*Figure 3-9: Average Altitude VS R1 for R2=5,20 Mbps (L=1000 MB)*

- The peak of the curve in the figure for R1 = 5 Mbps, is a good example for someone to notice the over-optimism of the HAS algorithm, which causes many bitrate oscillations, leading to the increase of the altitude value.
- If the channel is of a higher rate, this impact is eliminated, and the average altitude reaches to very low values.
  - In order to fully extinguish the impact and cause it to reach lower values, a higher R2 is needed, as shown by comparing the figures above.
  - The higher the R2, the SWG approach remains the best option.
  - The increase of the buffer size does not demonstrate any big difference in the behavior of the curve in the figure. A quite general conclusion is that the higher the buffer size L, the lower the average altitude.

Concerning the mean network throughput R1, we are able to observe the following in terms of buffer sizes 0, 100 and 1000MB:

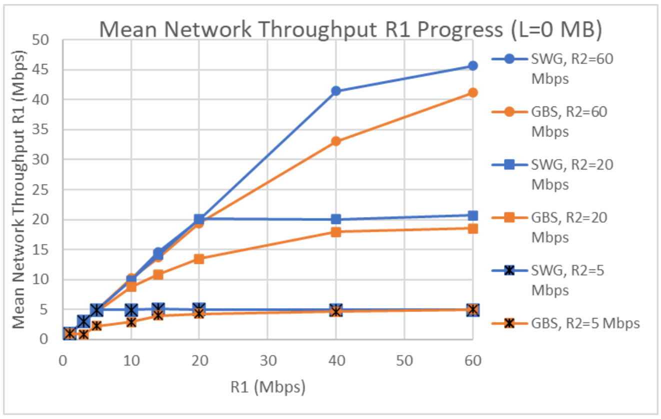


*Figure 3-10: Mean Network Throughput R1 vs R1 for L= 0MB*

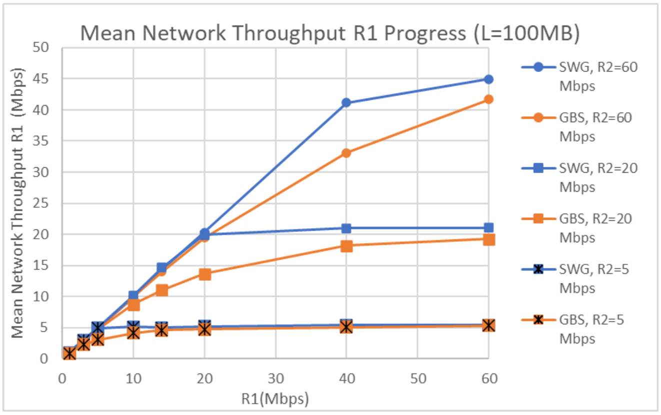


*Figure 3-11: Mean Network Throughput R1 vs R1 for L= 100MB*

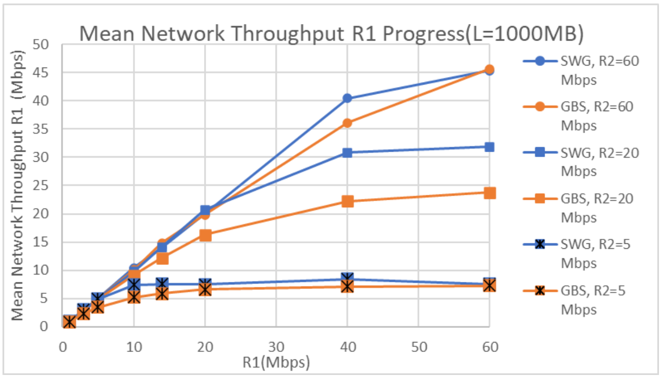


*Figure 3-12: Mean Network Throughput R1 vs R1 for L= 1000MB*

- The results are quite expected, as the throughput is increased linearly until the point where R1 = R2, where it reaches the highest value.
- For R2 = 60 Mbps, the throughput is not 60 as expected, but it is lower because the highest available quality (2160p) has an average bitrate of almost 45Mbps.
- The higher the buffer size, meaning that the fronthaul channel R1 is used more often than the backhaul, the higher the mean throughput R1.
- Better utilization of the channels in the SWG approach, especially when then buffer size is higher.
- In the GBS approach, the network resources remain unutilized, something prohibited for any vendor.
- The gap shown in the diagram between the two approaches for the same R2 value reaches higher values of R2. The R2 increase is responsible for the change of place of the gap in the right direction.

In the next two figures, we validate the behavior of the backhaul R2 for buffer sizes 100 MB and 1000 MB.

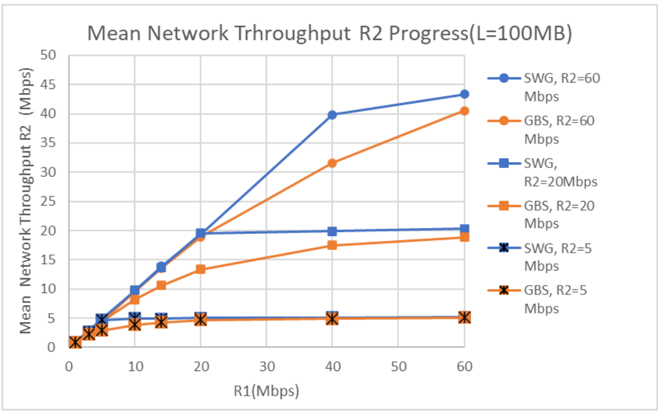


*Figure 3-13: Mean Network Throughput R2 vs R1 for L=100 MB*

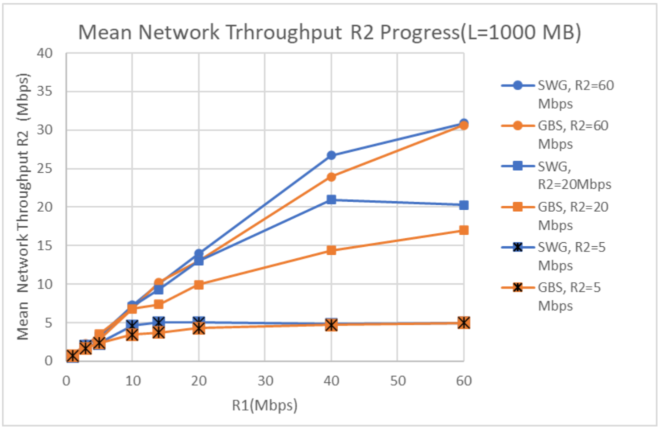


*Figure 3-14: Mean Network Throughput R2 vs R1 for L=1000 MB*

We can easily notice that:

- The curve is linear until it reaches an upper limit.
- Again, better utilization of the channel is shown in the SWG approach, compared to the GBS approach that leaves resources even more unutilized.
- However, the use of a non-smart algorithm and the non-existence of a multithread approach makes the backhaul channel idle for a long time, in case of a cache hit.
- Ideally, in case of a cache hit where the fronthaul channel would serve the client, the backhaul channel should download new segments to the cache, in order to benefit later, in terms of QoE and network resources.
- The SWG approach is generally more beneficial, but the biggest benefit comes in the points from R1 = 0 to 10 Mbps, where there is a range of available resolutions. The higher the R1 than those values, there is no more benefit, due to the absence of another higher resolution.

The Mean Opinion Score, that a client may extract if he watches a video, is depicted below for 3 different kind of cache servers:

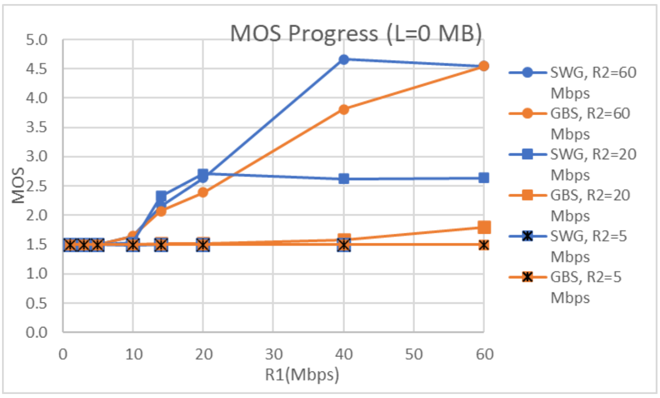


*Figure 3-15: MOS vs R1 for L=0 MB*

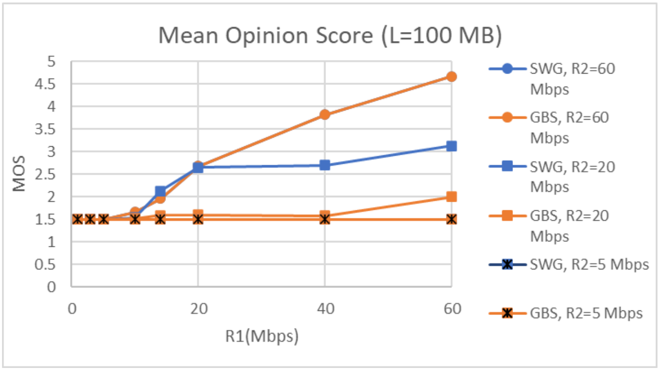


*Figure 3-16: MOS vs R1 for L=100 MB*

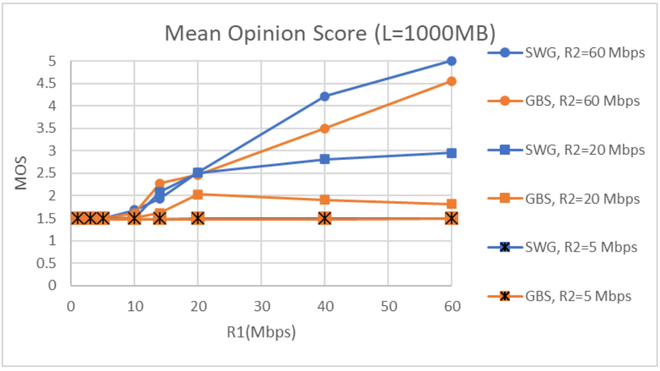


*Figure 3-17: MOS vs R1 for L=1000 MB*

We can conclude that:

- The MOS metric demonstrates the client's preference towards the SWG approach.
- The HAS algorithm, due to the fact that acts as over-optimistic sometimes for the available bandwidth, leads to the video freezing many times (stalling), and as a result the client is not satisfied.
- The bottom line is that the caching algorithms and the HAS algorithms are always interdependent because if the HAS algorithm is not cache-aware then caching does not help very much. In other words, we are wasting resources for storage, without being beneficial.
- The higher the R2 value, the less the bottleneck, so there is a benefit in the MOS value.
- For R2 = 5, the stallings that occur are so many that they lead to the worst value of the MOS. As a result, both approaches do not differ that much in terms of what the client feels while watching the video.

Last, as regards the number of stallings of our experiments, we conclude to the following results:

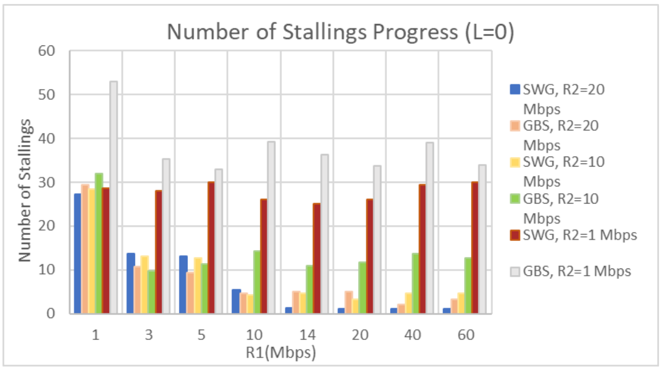


*Figure 3-18: Number of stallings vs R1 for L=0 MB*

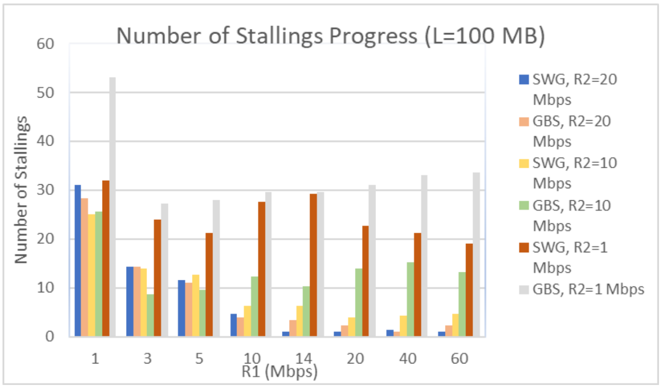


*Figure 3-19: Number of stallings vs R1 for L=100 MB*

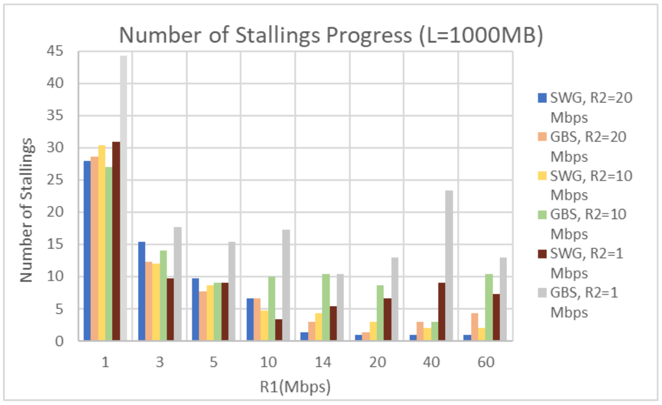


*Figure 3-20: Number of stallings vs R1 for L=1000 MB*

- To highlight a clear conclusion, it is possible that more samples were needed.
- Generally, there is a tendency for stallings to decrease while the R1 increases and while the buffer increases, since more segments are found cached and the media player uses them in his favor to avoid freezing.
- For a stable value of R1, there are more stallings for lower values of R2.
- The GBS approach compared to the SWG presents more stallings in the majority of cases. If we had more samples, the GBS approach would always have more stallings.
- The existence of a larger buffer benefits in terms of stallings, mostly in cases of higher R1 values. In cases of lower R1 values, there is also an improve in terms of stallings, but it is a smaller improve compared to higher R1 values.

### 3.4.3 QOE METRICS VS BUFFER SIZE L

The average resolution progress, as the buffer size is increased is shown in the next figure.

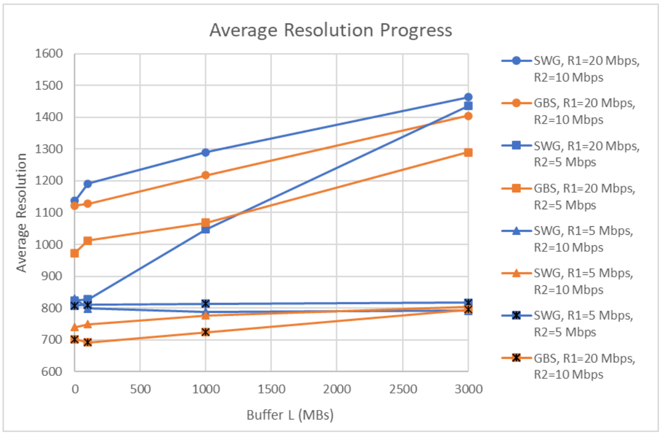


*Figure 3-21: Average resolution vs buffer size L*

We can observe that:

- The increase of the buffer size creates a satisfying profit over resolution, in the case of larger R1. If R1 is low, there is a slightly smaller increase, or no increase at all, meaning that caching is not always beneficial, but should be accompanied with high-speed channels.
- The SWG approach is much better than the GBS.

As for the backhaul throughput progress over buffer size:

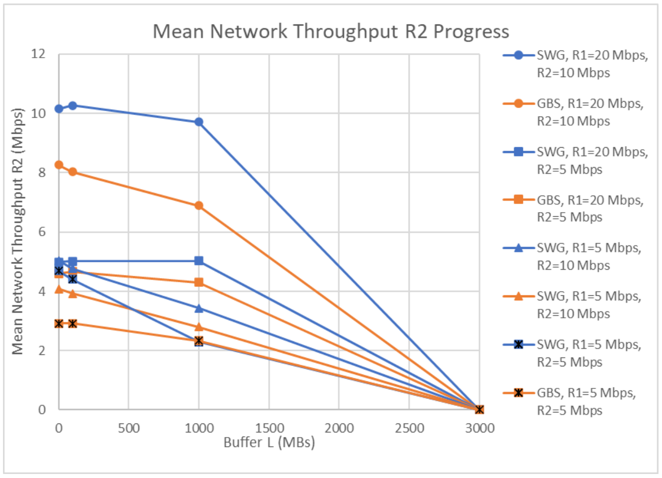
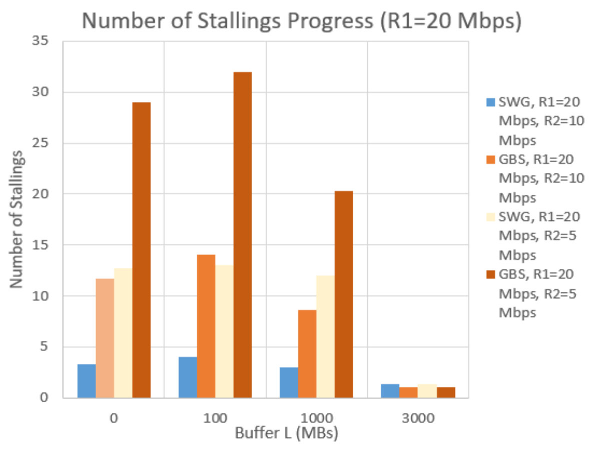


*Figure 3-22: Mean network throughput progress vs buffer size L*

- From this diagram, we can conclude on the utilization of the channels, especially in the GBS approach.
- This is demonstrated more clearly when in the case of the buffer size being 3000 MB, where all the segments are cached, the R2 throughput is zero. And besides this being acceptable for lower R2 values, it is a big waste of resources for larger R2 values.
- If we dealt with a multithread approach or a cache-aware HAS algorithm, then the curve would not fall progressively to zero, but the available R2 would be used to download even more segments proactively, optimizing the QoE of the client.

Now, if we compare the number of stallings that have occurred for two different fronthaul rate values, we can say that:



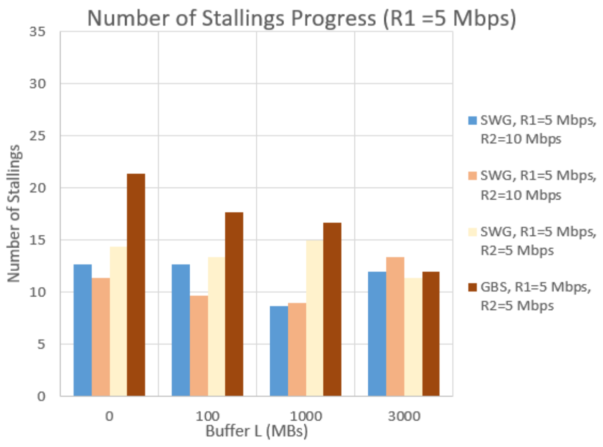


*Figure 3-23: Number of stallings vs buffer size for R1=5 Mbps and R1= 20Mbps*

- In both cases, the GBS approach has more stallings than the SWG approach. This is clear for R1 = 20.
- The higher R1 values can benefit to a larger extent from the increase of the buffer size and decrease the stallings in those cases.
- If the value of the R1 is low, there is no reason for any vendor to up the available resources and provide extra buffer size, since, at the end of the day, the QoE perceived by the client will not show many differences on the stallings perspective. Stallings will still occur, and they will annoy the client.

## 3.5 Conclusions

We examined how to deliver DASH-based content with the help of an intermediate server, apart from the main server, in order to demonstrate possible caching benefits for different sizes of intermediate storage servers. The goal was to exploit and take full advantage of the available bandwidth, along with the advantages of the caching approach. For this reason, we created a simple client-cache-main topology to implement the HTTP Adaptive logic, where if the cache server contained some segments, it served them immediately. Otherwise, the main server was delivering the missing segments. We compared the basic GBS logic of exchanging requests, with a slightly smarter SWG approach, so that it can serve some requests at the same time of requesting the next segment. Our aim was to increase the performance of our experiment to some extent and, consequently, the QoE of the client while watching the video.

The two main findings of our research are that the SWG logic enhances the performance of almost all QoE-wised metrics, especially when accompanied by a satisfactory caching policy, and reduces the under-utilization that we observed in our initial approach. We also noticed that the chosen HAS algorithm plays an important role in our experiments due to the dangers hidden by its over-optimism of which segment (in terms of resolution) will be the next to be played. So, if the media player is not aware of what is available on the intermediate server, this may lead to many stallings happening in the experiment, resulting in the loss of any benefit we had due to caching.

Finally, the caching algorithm that we used in all the implementations, responsible for filling the buffer of the intermediate server, is random and does not contain any particular intelligence in how to fill the cache with segments. An implementation of a smarter algorithm for future work is proposed, not based on randomness, but on the possibility of playing certain segments instead of others. For example, if the channels had a traffic of 5Mbps, then there is no reason to cache the segments of the 4K resolution, but certainly smaller than that. This would point out the assistance provided by the caching policy to all video streaming services.

# 4 AR-Assisted, GPS-Free Indoor Positioning (FOGUS)

The section here, presents our latest work, with respect to the Resource Orchestration techniques at the Network Edge. Our proposed solution is an Indoor Positioning System implementation, utilizing the well-known Fingerprint technique along with Visual-Based capabilities. The work in this document has been divided into the following Sections: Sec. 4.1 - Introduction, Sec. 4.2 - System Overview, Sec. 4.3 - System Model, Sec. 4.4 - Experimental Set-Up and Sec. 4.5 describing some of our most recent Results with respect to the proposed solution and implementation.

Deliverable 2.2-Brief Recap

In the previous work presented in *Deliverable 2.2,* we made a strong research and analysis on the Indoor Positioning Systems domain, giving emphasis mainly on the functionality of such systems and on how we could achieve the highest level of accuracy in our proposed work/solution. Our initial goal was to study and understand some previous works made in the field *-a brief description is made in the "Introduction Section"-* consequently trying to conclude on a solution that matches our needs and goals. To this extent, after long research we decided to move on with a "Hybrid" Indoor Positioning System solution, combining two different types of technologies, a combination of the Fingerprint technique for both Wifi and Cellular radio signals with Augmented Reality capabilities. Moreover, the most crucial part regarding our approach, was the right choice of the Algorithms and Tools to be exploited in order to achieve the best possible Positioning accuracy. After the research part, we started implementing the final version of our solution for testing in real life case scenarios. Initially, our implementation was tested in small-scale indoor environments giving us the opportunity to understand any issues or limitations with respect to the approach *(i.e Paragraph: Drawbacks in 4.1)*. After that we concluded on a big-scale Indoor space in a Lab Room at the University Of Athens and began with our experiments there (i.e., Sec. 4.4). Our findings (i.e., Sec. 4.5) showed that our Hybrid IPS succeeded with a great level of accuracy Indoors. Nevertheless, in some cases due to human or obstacle presence around the area a small deviation is observed in our results, yet at an acceptable rate as expected *(i.e., less than 1m)*. More details about that matter can be found on the final Sec. 4.5.

## 4.1 Introduction

As implied by the name, Indoor Positioning System (IPS) is the technology used to find a user's location with the highest accuracy possible in indoor spaces. An IPS is operating continuously, meaning that it receives data of someone's location in real time, unless of course the system is disabled by him. In many cases though, the absolute position of the user is not necessary and an approximate position would also cover our needs (i.e user A is inside a building and wants to know only the room in which user B probably is/was). In this "case example" there is no need for high accuracy in the IPS. On the other hand in many other applications the level of accuracy can be crucial and mandatory, of course always depending on someone's needs. To this extent, Indoor Positioning Systems have a wide range of applications, each tailored to address specific needs in various industries, ultimately improving efficiency, safety, and user experiences in indoor environments.

<u>IPS Functionality:</u> When using an IPS, the user's position is shown over a map (i.e two or three dimensional), which is a simulation of the space where the user is in. In order to make this correlation of real-time location and projection to the map, the creation of an Indoor Map of the environment is required to be made beforehand. This procedure is called Indoor Mapping and is a separate research study. Moreover, when the Map of an indoor space exists and a user wants to get directions of how to go to another point on the map, an Indoor Path planning procedure is required. Navigation of a user in an indoor space is done in terms of avoiding obstacles like walls and finally reaching his destination in the minimum possible time. The path on the map is created using various mathematical models for calculating the shortest path. Therefore, creating an IPS may consider the sub problems of (1) Indoor

mapping (2) Indoor Positioning and finally the (3) Indoor Path Planning. Last but not least, the positioning accuracy of an IPS is highly dependent on the technologies and techniques utilized by the system. Wifi and Visual-Based are proven to be the best applied approaches so far, offering accuracy on average up to 6m & 1.5m respectively. The next paragraph briefly describes the different types of technologies that can be used to achieve accurate positioning in an indoor environment.

<u>IPS Methods:</u> The problem of finding someone's positioning in an Indoor Space, has been approached so far using various technologies with the aim of replacing the GPS that cannot be used with high accuracy. Some of these technologies are wireless (i.e Radio Frequency) technologies (i.e WiFi, Bluetooth, Ultra-Wide band), Inertial technologies, Audible and non-Audible Sound technologies, Light and Vision-Based technologies. Among them, the WiFi technology is the most commonly used technology for Indoor Positioning Systems. Its main advantage is its universality, meaning that someone does not need to install new machines or infrastructure to use it. To this extent, next we present some of the most common Location Positioning technologies used today: Wi-Fi-Based Systems: Utilize Wi-Fi signals from access points to triangulate positions. The most common approach using the WiFi and Mobile Radio technologies is the Fingerprinting technique, which creates an a priori map of points with their Signal Strength measurements, and then in real time, the signal received by the user is compared with the stored measurements, thus making it possible to deduce his actual position in the specified area. Moreover, Time of Arrival of the signal can be used to find the location of a user. In ToA, the transmitter includes the time when the signal is transmitted in the radio packet, and the receiver calculates the time spent on the transmission. With this information, the receiver can calculate the distance between two Access Points, with the prerequisite that the nodes are synchronized. Moreover, it was observed that the existence of APs with stronger Signal Strength is more important than the number of Access Points as the weakened signal shows significant fluctuations in its values, as the utilization of such signals affects the accuracy of the collected measurements. The main disadvantage of using the WiFi technology refers to the signal propagation model (i.e due to the existence of obstacles around the area) and the existing infrastructure (i.e number and location of APs). Bluetooth-Based Systems: Use Bluetooth beacons or Low Energy (BLE) signals for proximity-based positioning. RFID (i.e Radio-Frequency Identification): Tags and readers communicate via radio waves for precise tracking. Infrared and Ultrasound: Employ sensors that transmit and receive signals for accurate positioning. Geomagnetic Positioning: Relies on the Earth's magnetic field for orientation and positioning. Finally, we have the Visual Based approach in which by using the camera of a mobile device, the surrounding area can be captured and then find a target location, by matching these images with the current image of a camera. Also, by using specific points (i.e Access Points) as landmarks, a target location can be calculated in relation to the distance from these points. Another

visual technology that has received a lot of attention in recent years is Augmented Reality (i.e AR). AR, is an interactive experience that combines real world and computer-generated content. AR can be defined as a system that incorporates three basic features: (1) a combination of real and virtual worlds (2) real-time interaction and (3) accurate 3D registration of virtual and real objects. Augmented reality has been used a lot in recent years in navigation applications. For better accuracy of the location of "augmented" objects in space, special emphasis was placed on algorithms that utilize vision sensors, i.e. the camera of mobile devices, to locate and store various points of interest in space and map the environment. This helps to compare and recognize points in space and not whole images with the camera. The main algorithm used in AR systems is Tracking and Registration which achieves the recognition and registration of an environment based on discrete point recognition. To this extent, an AR tracker is a specific pattern or image that an AR app can recognize. Once the app finds the pattern, it constantly 'tracks' the position of the pattern in real world space, so the application can finally accurately place a virtual object onto the tracker. Another approach that uses the camera of the user's mobile device is by scanning QR codes. First, it is required to create a map of an indoor space or a building and use some spots as points of interest, where a QR code is assigned to each point. Then navigation is achieved by scanning these QR codes that represent the current position of the user over the map. So, with the knowledge of the current position, navigation to other destinations can be achieved by calculating the shortest route to a target point and directions are given based on the mobile's inertial sensors.

Previous Work: Throughout the latest years many attempts have been made on the Indoor Positioning domain, regarding among others the achievement of the highest possible accuracy. Here we will briefly describe some of those. A study made in [23] proposes an AP placement method for Wifi Fingerprinting that considers existing APs where the parameters of the radio propagation model are first obtained based on data collected from the existing APs and then, based on the self-learning parameters, a genetic algorithm (GA)-based optimization method is performed for the AP placement. The authors in [24] made an analysis on the main drawback for large-scale applications of WiFi-based localization which is the varying characteristics of received signal strength (RSS), as a result degenerating the localization performance seriously. To overcome this issue this work presents a hybrid fingerprint quality evaluation model (HFQuM) for accurate WiFi localisation which resulted in great results in comparison with other existing similar fingerprint methods. Furthermore, a work made in [25], proposes a novel WiFi and vision-integrated fingerprint (Wi-Vi fingerprint) for accurate and robust indoor localization. As in our case their experiments took place in a Big-Scale Indoor area of 12000m2 and a mega-mall of 7200 m2 with different types of smartphones. In the localization step, a multiscale localisation strategy was proposed, including coarse localization from weighted access points (WAPs)-based WiFi matching, the Gaussian weighted KNN (GW-KNN) based image level localization from holistic visual features, and finally, the metric localization for refinement. Their experimental results demonstrated that the proposed method can achieve 95% and 98% site recognition rates from image-level localization. Indoor localization is a challenging task as the signal propagation in indoor environments does not adhere to the classical path loss or other simple models. A framework proposed in [26], puts forth a dictionary learning framework for fingerprinting indoor locations using GSM, WiFi, and other sensor measurements. Different from the existing works, the proposed methods are model-free, can handle missing measurements, and fingerprints can be flexibly updated in an online fashion. The extracted results in this work demonstrated the efficacy and usefulness of the proposed algorithm, in comparison to the state-of-the-art classification-based and model-based indoor localization algorithms. Finally, [27] proposes a fingerprint augment framework based on super-resolution (i.e., FASR), which achieves the fusion of fingerprint augment and super-resolution based on mutual conversion between fingerprint data and fingerprint image, in order to improve the accuracy of WiFi fingerprint-based indoor localization with limited fingerprints while reducing the site survey effort. The results confirmed the potentiality of the super-resolution algorithm on RSS estimation as well as high positioning accuracy.

Drawbacks: The accuracy of an IPS can be affected by various factors: *Signal Interference:* Obstacles such as walls, furniture, or equipment can interfere with wireless signals used by IPS technologies like Wi-

Fi, Bluetooth, or RFID. These obstructions can weaken or distort the signals, leading to inaccuracies in determining the device's location (1) *Multipath Effects:* When signals bounce off surfaces, they create multiple signal paths that can interfere with the original signal. This phenomenon can cause inaccuracies in calculating the position of a device (2) *Mapping and Calibration:* If obstacles were not accounted for during the initial mapping or calibration phase of the IPS, inaccuracies may arise. Changes in the environment or the addition of new obstacles can also impact the accuracy if the system isn't updated accordingly (3) *Sensor Limitations:* Some IPS technologies rely on sensors or beacons placed strategically around an area. Obstacles blocking the line of sight between the sensors and the device can affect the accuracy of location tracking. Some proposed solutions to mitigate these issues and maintain accuracy despite obstacles could be: (1) Signal Optimization: Adjustment of the signal strength, frequencies, or placement of beacons/sensors to minimize interference from obstacles (2) Algorithms and Software Improvements: Enhanced algorithms to account for multipath effects and software utilization that can better interpret signal distortions caused by obstacles.(3) Dynamic Mapping: Implementation of systems that can dynamically update maps based on changes in the environment to ensure accuracy despite obstacles.

In conclusion, our proposed solution is a Hybrid combination of the Fingerprint technique *(i.e., WiFi & Mobile Radio signal measurements)* and Augmented Reality *(i.e., AR Cloud Anchors)*. Next, *Sec. 4.2* presents our *System's Architecture* as well as some of the *Tools* used for the implementation part.

## 4.2 System Overview

In this work, we have implemented an end-to-end system capable of providing *cm-grade* accuracy in 3D indoor setups. The system is based on the idea of fingerprinting, which comes down to the idea that a single point on a 3D grid has a specific fingerprint in terms of received signals from wireless networks *(i.e., Wi-Fi, cellular, Bluetooth, etc.)* as well as visual signals *(i.e., Augmented Reality)*. Key novelty of our proposed system is the combination of wireless network signals with 3D digital objects used to refer to the relative position of a camera-enabled device *(i.e., in our case an Android device)* with respect to the different objects placed as 3D Anchors in the setup. Using ML algorithms we manage to achieve cm-grade accuracy in 3D space, managing the accuracy vs processing trade-off.  Next is described the architecture of the proposed IPS.

System Architecture: Our system is based on a "two-phase" positioning approach: *(1) Exploration (i.e., training/mapping)* and *(2) Exploitation (i.e., online service use)*. During the first phase, we decide on the granularity of the space grid, collect measurements on radio signals and distances from 3D digital objects placed strategically in the setup, creating the fingerprint per grid-point *(i.e., RP)*. A location server collects measurements upon request *(i.e., during the exploitation phase)*. Moreover, during the second phase, the location client switches on its camera and performs radio measurement collection. The combination of those measurements is then sent to a server and state-of-the-art Machine-Learning algorithms provide an accurate estimation on the position of the camera-/communication-enabled device on the 3D Indoor space. To this extent, (Figure 4-1) illustrates the Architecture of our proposed IPS, followed by a detailed analysis of the roles and technologies involved in every step of the Mapping and Positioning procedures.

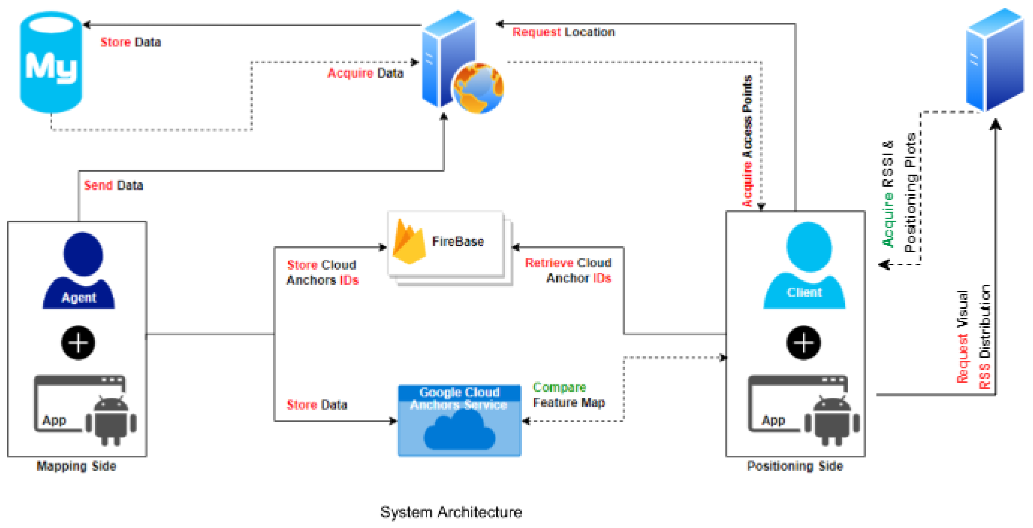
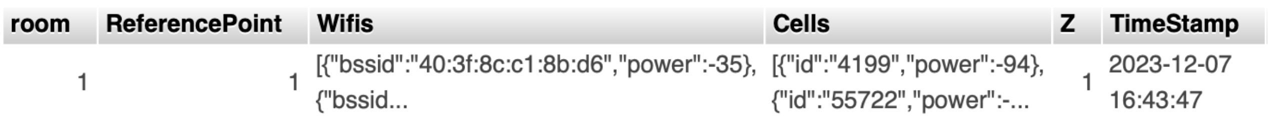


*Figure 4-1: System Architecture*

<u>System Specifications: </u>Our custom Android Mobile Application (i.e., MA) embodies two separate functionalities. The first one is the Mapping Function *(i.e., Offline Phase)* and the second one refers to the user’s Positioning (i.e., Offline Phase). In the work presented in this document, a user that uses the MA to Map an Indoor space is referred to as the “Agent” and any other user that uses the application to locate his/her location in the Indoor space will be referred to as the “Client”. Thereafter, the MA was designed with the following basic functionalities: (1) After its successful execution and installation on the mobile device, a user is the free to act either as an “Agent”, or as a “Client” 2) The Mapping Procedure allows an Agent to place an Augmented Reality 3D object in space marking his location. After defining/registering the Room in which he is located, then for 30 seconds the surrounding space is mapped by turning the device camera in all possible angles around that exact point. In parallel time, the device takes 4 measurements from the available mobile Radio and WiFi signals and holds the average of those values. After 30 seconds, this acquired information is then uploaded to our Database. At this point let's clarify that for each CA a unique identifier is generated as reference within Google’s Databases and normally is quite large in size. For this reason, a short form of this identifier is stored in our Local Database while the full form of the corresponding ID is stored in the Firebase 3) The Location function allows a Client to find his positioning indoors, after the area has been properly mapped. This process works in 2 distinct phases: 3.1) In the first phase, the mobile device uses the information relative to the available mobile Radio signals and WiFi networks, to locate the closest RP to the user. These RPs after being fetched by the Server, the best are returned sorted to the user, with the choice based on the signal measurements taken in the previous step 3.2) After matching at least one point, the MA derives the Client’s location based on the distance (D) calculation from that point and the available signals. In the case where no CA is found then the Client’s location can be extracted only from the available WiFi and Radio signals. On the Server side there is a Database of Mysql type, that stores the RPs that have been used during Mapping procedure and stores all the correlative data about them. These data include, the Room in which the measurements took place and the signal information gathered from both the available WiFi and Cellular APs *(i.e RSSS, BSSID, SSID from the former and Cell_ID and RSSI from the latter)*. Moreover, in the Server, 2 Java Servlets operate, regarding the request operations, used for storing new RPs or restoring location information. The first Servlet responds to the requests made about adding a new measurement in the Database, while the second one implements the Data Matching Algorithm and returns the RPs that are found to be closer to Client. Furthermore, the Database created in Firebase, stores the full-form IDs of the CAs, so when it is required to find the exact location of a user, the MA calls the CA Service to compare them with the view acquired from the device.

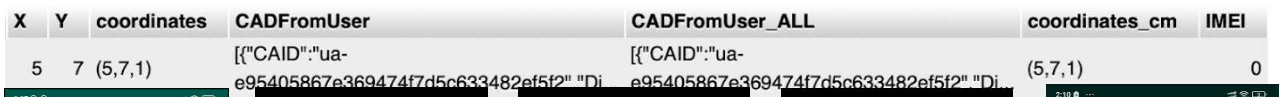
System Procedures: Here are described the steps to be followed when using our custom MA for the Indoor Mapping and Positioning operations. As mentioned earlier, the MA consists of 2 phases with the first one being the Mapping and the other one being the Positioning procedures. *1) Mapping:* once the application is installed and running on our mobile device it's camera automatically gets activated and the user has at his disposal the "Place Anchor" action button that allows him to place a CA at the precise location of the mobile device in the AR environment *(i.e., Indoor space)*. In our case the 3D object used as a CA is a round "sphere". After choosing to place the CA a dialog box appears in which the user has to fill in with his information *(i.e., Room Number, Number of RP)* about the exact RP that is going to be placed on. After that, the process of recording the RPs around the placed CA initiates. This process takes around 30 seconds to complete, ideally giving enough time to successfully record the environment around the 3D object. At the same time 4 measurements are also collected from the WiFi and Mobile Radio signals, through the MA. The time interval for getting those measurements is set to be 5 seconds and by the end of the process the average value is stored per BSSID or Cell ID, both detected by the MA.



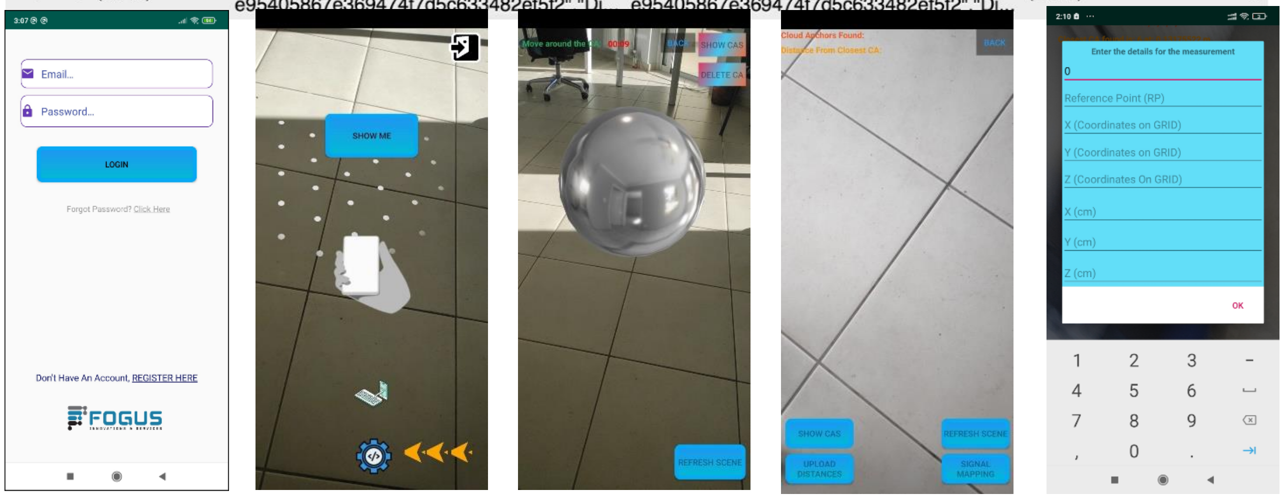
| room | ReferencePoint | Wifis | Cells | Z | TimeStamp |
|---|---|---|---|---|---|
| 1 | 1 | [{"bssid":"40:3f:8c:c1:8b:d6","power":-35}, {"bssid... | [{"id":"4199","power":-94}, {"id":"55722","power":-... | 1 | 2023-12-07 16:43:47 |

*Figure 4-2: Fingerprint database.*

After the 30 seconds have passed, the Agent can upload the FPs on the Google Server. Moreover the CA_ID is stored in the Firebase and the received signals are also sent via the Server to our Database (i.e., Figure 4-2), though the specified Servlet used for this purpose. Finally, the mapping process can be initiated again to establish a new CA in the Indoor Environment. The process will take place in the same



| X | Y | coordinates | CADFromUser | CADFromUser_ALL | coordinates_cm | IMEI |
|---|---|---|---|---|---|---|
| 5 | 7 | (5,7,1) | [{"CAID":"ua-e95405867e369474f7d5c633482ef5f2","Di... | [{"CAID":"ua-e95405867e369474f7d5c633482ef5f2","Di... | (5,7,1) | 0 |



Room until all CAs in it have successfully been mapped. Figure 4-3 below presents the steps to be followed when the Agent initiates the Mapping process on his mobile device.

*Figure 4-3: Mobile application UI.*

*2) Positioning:* Similar to the previous phase, here a user can operate now as a Client. The camera of the device is activated again through the MA and transports the user to the AR environment. Then the positioning phase immediately begins. The process consists of 2 parts, with the first one being the collection of the available measured WiFi and Mobile Radio signals. The Server then uses the equivalent Servlet to compare these measurements with the RPs stored in the Database by executing specific Machine Learning (i.e., ML) algorithms. The unique IDs of these points are stored in a short form *(i.e., integer number)*, so Firebase retrieves their full form IDs. The MA will use the IDs of these points to match the camera view with the stored points. The process just described *(i.e., Hosting/Resolving Cloud Anchors)*, executes on the

Google Servers automatically. In the end, if a successful match is found, then the CA that was placed during the Mapping procedure and an Augmented label on top of the placed 3D object shows the exact distance from the Client's mobile device. Last but not least, on the Client's screen is also displayed information that has been used during the measurement collection process. This information refers to the distance from the nearest RP *(i.e., closest CA)* as also the points detected by the CA Service *(i.e CAs found)*. By selecting the corresponding action button on the application, the position of the Client in relation to the CA detected can be shown. To this extent, a list containing all the CAs detected in the Indoor space along with their distances from the user are also displayed to the user on the corresponding dialog box.

Implementation Tools: This Subsection summarizes the tools (Figure 4-4) used for the implementation of our proposed system. The MA (i.e., Frontend part) is created mostly in Kotlin *(https://kotlinlang.org/)* language in the Android Studio IDE *(https://developer.android.com/studio)* which contains all the necessary classes and functions for the Mapping and Positioning procedures. All HTTP requests to the Server were created with the help of the Volley library and for the AR part the ARCore *(https://developers.google.com/ar)* library and the Sceneform plug-in were utilized. Additionally, to display the 3D "sphere" objects on the indoor space and to find the APs through AR, the Appoly library was used. As far as the Backend part is concerned *(i.e., HTTP Servlets)*, we use the Intellij IDE *(https://www.jetbrains.com/idea/)* and all Servlets are written in Java language. Moreover, the Main Server runs locally on XAMPP *(https://www.apachefriends.org/index.html)*, while Apache Server, MySQL Database and Tomcat *(https://tomcat.apache.org/)* on which our Servlets are executed, are also being enabled.

*Figure 4-4: Implementation tools.*

## 4.3 System Model

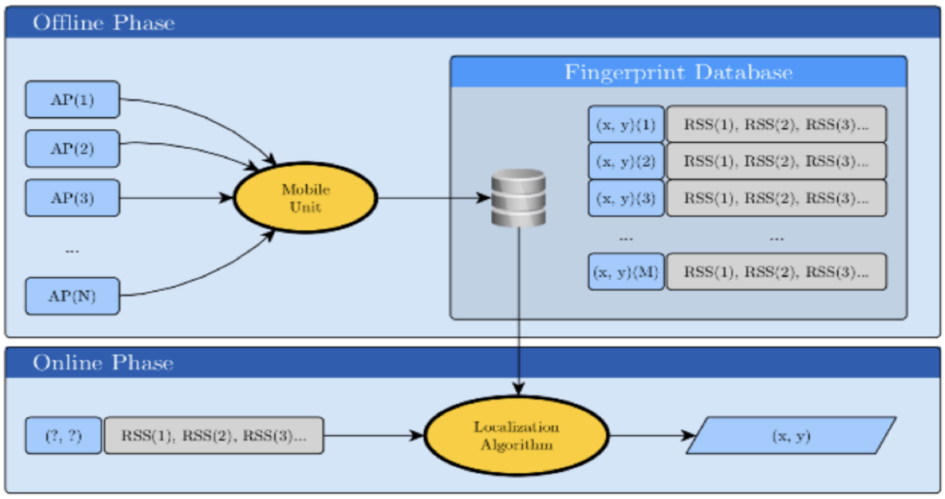
The system model of the proposed IPS solution, takes into consideration different types of inputs, following next. The steps for implementing a typical Indoor Positioning System nowadays vary from one approach to another, yet with the most applied one being the *Fingerprint technique* Figure 4-5**.** The indoor space is a Mapped Grid {G} (Figure 4-6) of dimensions {MxN} and Reference Points {RP} where RPi ∈ RP. The

Fingerprint technique offers the ability to map an area by collecting a variety of Received Signal Strength measurements*({APw:ssid,bssid,power}, {APc:cid,power})*, more precisely in this work gathered from 2 types {Tk} of Access Points {Nk} where k ∈ K , consequently creating a mapped area based on those received data. In our case the RSS measurements are collected from all the available WiFi {APw} and Cellular base stations {APc} being available to the user, with the help of our custom smartphone android mobile application. As also mentioned earlier we define the user who maps the Indoor space as the Agent {Ua} and the one looking for his positioning on it, as the Client {Uc}. Moreover, as {L} is defined as the set of {RPs} where all measurements are collected and {li = (x, y, z : *coordinates are also defined in centimeters*)}, in which li ∈ L. Moreover, before the measurement collection process begins, the initial phase demands the installation of the Cloud Anchors {CAi ∈ CAset} around the space and their host on the Google Servers where each one contains a unique ID number. As far as the second phase of our implementation is concerned, after the Mapping process is been completed, {Uc} can utilize our mobile application and after "resolving" all the available {CAi..CAn} he begins collecting RSS measurements from all {Tk} of {Nk} around him. As a result our application returns the closest RP found on the GRID with respect to the data acquired during the 2nd phase.

*Figure 4-5: Fingerprinting technique*

## 4.4 Experimental Setup

Our proposed Hybrid IPS tested in different types of real life scenarios, each time by switching not only the type of the environment (i.e small-scale/big-scale spaces) but also the number of Access Points (i.e Raspberry Pis) and the 3D Objects (i.e Cloud Anchors) placed around the area. Small-scale areas initially



used for testing purposes in order to validate that our implementation and algorithms work as expected, as the main Indoor Space used in our work is a big Indoor Room of 820 in total mapped Reference Points. Moreover, the aforementioned area is not an empty space, meaning that during the Mapping process the IPS can experience variations in the Signal collection process when the density level of obstacles or humans is high within the area. Figure 4-6 and Figure 4-7(left) present our GRID *(i.e 820 RPs placed in Lab206 at the University Of Athens)* and the tripod device used to collect our measurements, which has attached the Android device on it. Finally, Figure 4-7(center) and Figure 4-7(right) showing the Lab area and one of our personal APs placed in the corner of the room respectively.

As can be observed from the above plot, this is a top-view of our Mapped area. As *"blue dots"* we define the Cloud Anchors installed on the GRID and as *"red dots"* the WiFi Access Points around it. As far as the latter is concerned we placed 4 in total personal APs with 3 of them being Raspberry Pi devices acting like routers and 1 WiFi Router already existing and operating in the Lab. The reason behind this choice was that, as previously mentioned, accuracy is highly dependent on the number and quality of the collected signal measurements, so it would be safe to at least always consider our personal Access Points, consequently having a basic rate of accuracy always.

<u>Algorithms:</u> Our work follows a variation of approaches and scenarios to prove its efficiency as a proposed system for Indoor Mapping and Positioning procedures. To this extent, we chose 3 implementations/approaches with respect to the Mapping procedure. The main difference between these implementations lies in how we create the map of the indoor space and then how we use the measured data we have stored in order to select a user's position. The first approach is the *FullGrid_RoomDriven (i.e FGRD)*. For each RP on the Grid *(i.e Indoor space divided into segments of the same size)*, a CA is placed. Initially the Fingerprinting technique is applied to collect the signals measured from all the APs in the indoor space *(i.e available Wifi & Cellular)* and then find the user's position with respect to the CAs located in the room. The user's position will be considered the RP, from which he has the closest distance from those that have been detected during this process. Let us note that the distance from each CA is computed through

the MA. Second approach is the *FullGrid_BestMatch (i.e FGBM)*. Again similar to the previous case, a Map is created with one CA per each RP, but the selection of the user's location is made among the 20 best RPs selected by the KNN algorithm. More precisely, we use the Euclidean distance of each record from the user's measurement to calculate the 20 Best Records corresponding to the 20 RPs closest to the user based on the WiFi and mobile antenna signals. The third way is the *Partial_Grid (i.e PG)* approach. In this implementation the idea is to place as few CAs as possible. Thereafter, the CAs are placed again along the Map but this time not on specific RPs but instead on central locations on the Grid. After placing the CAs at the spots of our preference, then for the rest of the RPs during the signal measurements process, we also look for nearby CAs. In this way it is ensured that each RP "sees" at least one CA.

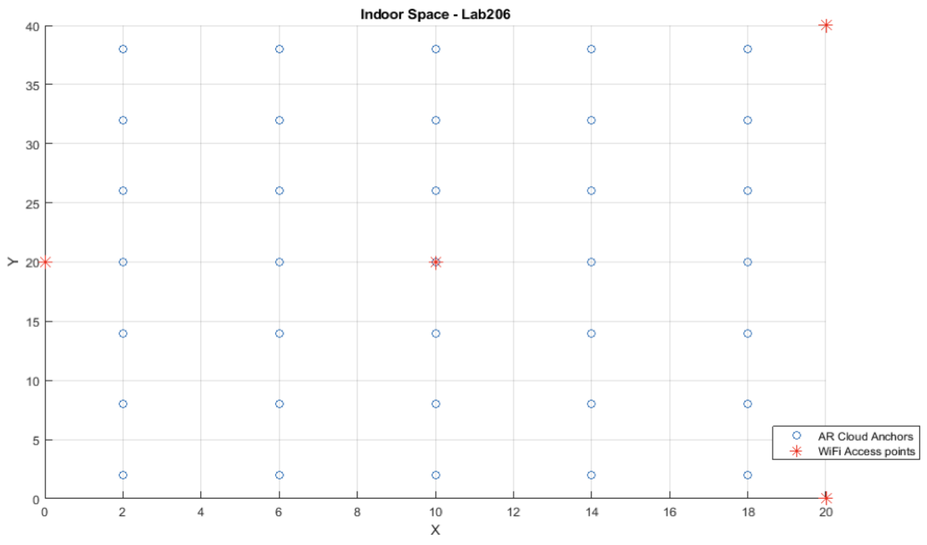


Figure 4-6: Indoor GRID.

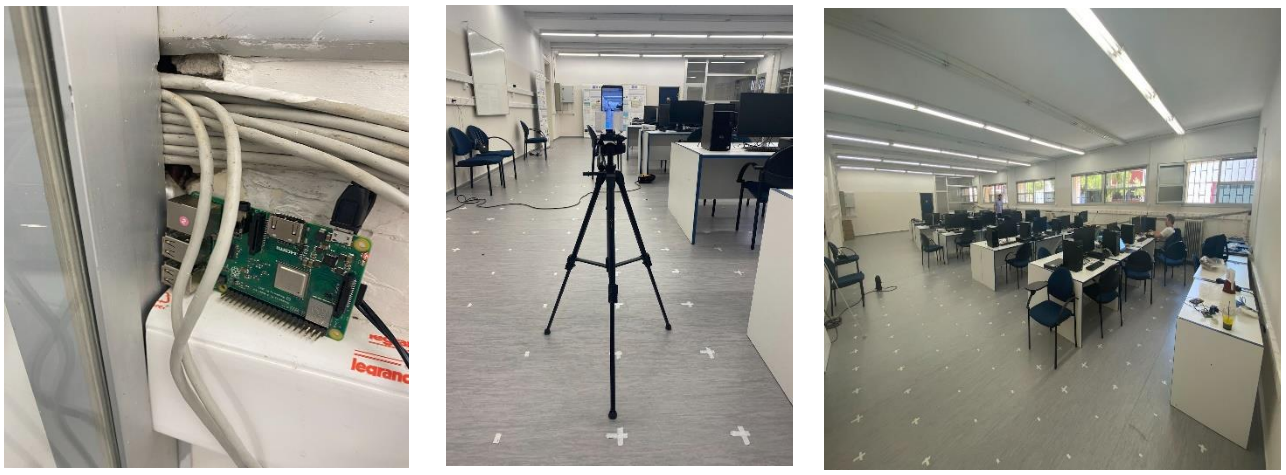


*Figure 4-7: Testbed setup: Raspberry Pi AP (left); tripod (center); indoor area/lab 206 (right).*

<u>Scenarios:</u> Our approach exploits an algorithm that combines the signal measurements gathered from the Wireless Networks *(i.e WiFi & Cellular APs)* and from Visual Data, resulting from "resolving" CAs in the indoor space. Thereafter, finding the Client's location can result from the combination of these collected information. In such a manner, the comparative algorithms discussed next, are different approaches using the same data collected in the Mapping phase: (1) Best-Server: compares the received RSSI measurement

of the most powerful WiFi *(i.e in terms of Received Signal Strength)* with the measurements taken and stored in the Database for this exact AP and subsequently returns the RP with the smallest difference in value *(2) WiFi:* Collects all the available WiFi signals and returns the point with the closest Euclidean distance after applying the *K-Nearest Neighbor (KNN)* algorithm *(3) Cellular:* an algorithm that receives all the signals from the available mobile antennas and returns the point with the closest Euclidean distance again by applying the *KNN algorithm (4) Hybrid:* an algorithm combining the previous 2 algorithms. In practice, it takes into consideration all the available signal measurements from both WiFi and Cellular APs,

$$f_X(x) = P(X = x),\ x \in A$$

for the Euclidean distance computation *(5) Visual N*: An algorithm that considers only the visual measurements gathered from each CA found in space (6) *WiFi-Cellular-Visual:* the final algorithm, as the name suggests, is a combination of all the aforementioned, meaning that the algorithm will take in consideration all the acquired data.

Algorithm Variations: (1) No_Penalties KNN (np_KNN): If values in the database are also present in our measurement (e.g. a WiFi present in our measurement is not present in the record examined by the database then we will not take this value into account. Likewise for the values which are present in the records but do not appear in our measurement) (2) With_Penalties KNN(wp_KNN): puts a penalty on the value that is absent from the count/record. This penalty is defined as a maximum possible value (e.g. if in a measurement WiFi X is detected and when we compare it with the records in the database we notice that WiFi X is absent in some records, then we will enter the value -120 (maximum possible value according to the standard) for the records that do not have WiFi X and then we will apply the KNN algorithm. Note that the maximum values used in this implementation are -120 (in dbm) for the WiF is, -140 (in dbm) for the cell antennas and 1000 (in centimeters) for the undetected cloud anchors.

Metrics: Several metrics were used to examine the evaluation of our proposed algorithms. (1) The first one (i.e., Hit Ratio) is used to evaluate the number of RPs which are correctly located in the mapped area. All measurements are made by going to a RP and evaluating whether the point returned by the algorithm matches with the user's position, or not. (2) The second group of metrics used in order to define a Distance Tolerance with respect to the RPs located from the correct points, are the *Probability Density (i.e., PDF)* (Figure 4-8(left)) and the *Cumulative Distribution (i.e., CDF)* (Figure 4-8(right)) functions. With Distance Tolerance is meant the degree of distance between the detected RP and the correct point. In

such a case if RP is categorized as correct and it is next to a detected point and as a Distance Tolerance is chosen to be only 1 neighbor, then the measurement will be considered as a correct one. In the same manner, the process can be done for all possible neighbors. (3) The third metric is the Average Distance returned by each algorithm, in contrast to the number of the correct points found in space. Thereafter, in the case where a point is located by our algorithm but the one next to it is the correct one, then the distance would be 1-2m. In the opposite case the distance would be set to be equal to 0.

*Figure 4-8: CDF (left); PDF (right)*

$$F_X(x) = P(\omega \in \Omega : X(\omega) \leq x)$$

## 4.5 Results

The final Section of the work in this document, presents and describes some of the results acquired after testing our implementation in the Big-Scale Room, Lab206 (Figure 4-6). As mentioned previously, in our testing experiment we used 4 personal WiFi APs, placed around the area. Below Figure 4-9 showing the Signal Strength distribution of all personal APs in our Indoor testing environment after the Mapping process is completed.

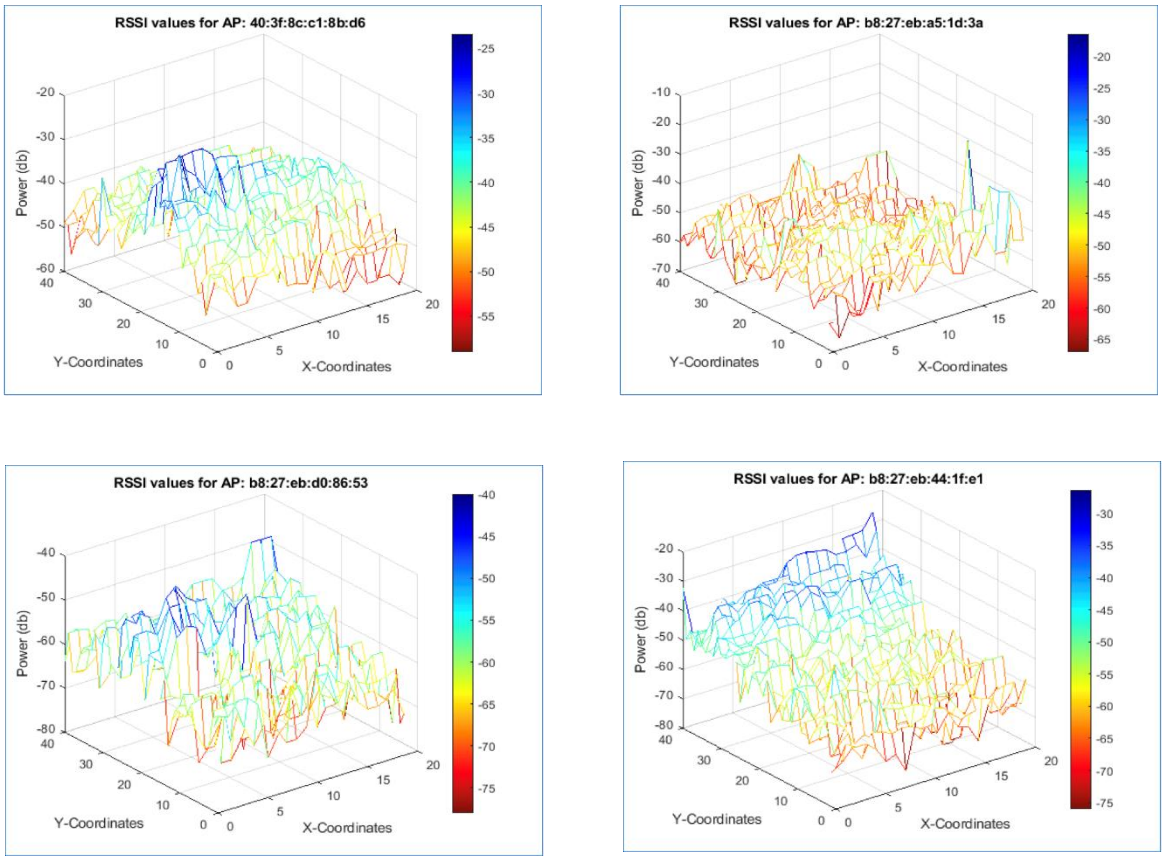


*Figure 4-9: Router AP (Top left); AP1 (top right); AP2 (bottom left); AP3 (bottom right)*

As can be observed from the above plots, each one refers to a specific personal AP installed in Lab206. The first plot (i.e top left corner) refers to the basic Router used in the Lab as the main router. As can be seen the RSS levels are higher closer to this AP and lower as the device moves away from it. The same outcome was also observed for the other 3 APs *(i.e Raspberry Pis)* as expected. On the other hand, in cases where obstacles exist, a noticeable decrease in the power levels *(i.e db)* appears, proving our assumptions that high interference can have a great negative impact on such systems. Moreover, our proposed solution offered a great impact on accuracy, in most cases deviating from the actual positioning mapped from {Ua} not more than 60 to 70 cm. Last but not least in cases where the interference level was high enough due to known factors mentioned in Sec. 4.1, the positioning accuracy achieved gave us higher deviation from the actual location, in some cases up to more than 1.5m.

# 5 MEC service-provisioning for the beyond 5G RAN (CTTC)

## 5.1 Background Information

### 5.1.1 MEC Integration with 5G and beyond 5G

The four main functions of 5th Generation (5G) are communication, computation, control and content delivery. 5G is not only an advancement in mobile broadband (eMBB) services but it also extends to support vast diversity of massive Machine Type Communication (mMTC) and Ultra-Reliable Low Latency Communication (URLLC) based application services along with support for different verticals. It is foreseen that each application in 5G has different requirements in terms of latency, data rates, reliability/availability, mobility and transmit powers, inducing signification surge in demand for computation and storage resources closer to the users along with enhanced communication resources. 5G alone is not enough for achieving such advancements. The edge computing power of MEC and its integration with 5G brings applications, 3rd party services and content closer to the user in an efficient way and is one of the main enablers for turning telecommunication networks into versatile service platforms.

#### 5.1.1.1 Key enablers for MEX integration with 5G

1) 5G Service-Based Architecture (SBA): In 5G core network, the 4th Generation (4G) monolithic Evolved Packet Core (EPC) [28] is disaggregated to implement each function in such a way that it could run independently on a Common Off-The Shelf (COTS) server hardware. It allows 5G core function to be flexible and decentralized.
2) The ability of an Application Function to influence User Plane Function (UPF) (re)selection and traffic routing directly via the Policy Control Function (PCF) or indirectly via the Network Exposure Function (NEF), depending on the operator's policies [28].
3) Local Routing and Traffic Steering: The 5G Core Network selects a UPF close to the UE and executes the traffic steering from the UPF to the local Data Network via a N6 interface. This may be based on the UE's subscription data, UE location, the information from Application Function (AF) [28].
4) The Session and Service Continuity (SSC) modes for different UE and application mobility scenarios
5) Support of Local Area Data Network (LADN)

#### 5.1.1.2 MEC integration with 5G

The 5G system architecture specified by 3GPP has been designed with significant changes in RAN and core part to handle wide range of use cases from a massive amount of simple IoT devices to the other extreme of high bit rate, high reliability mission critical services. The user plane and control plane network functions are worth introducing before diving in to MEC integration with 5G.

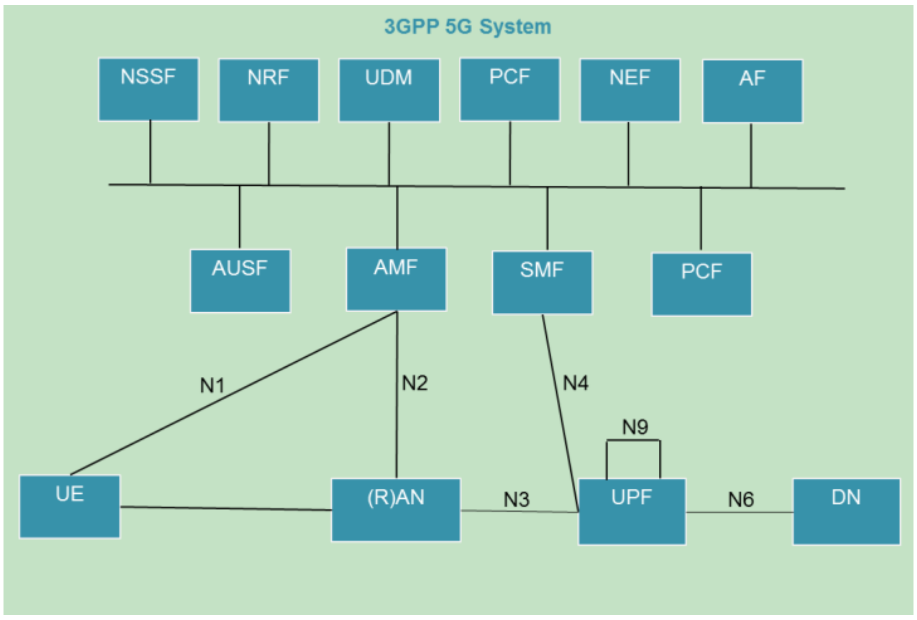
1) Network Resource Function (NRF): Contains the list of network functions and the services they produce.
2) Network Exposure Function (NEF): Responsible for service exposure and authorization of service access requests.
3) Application Function (AF): Application influence on traffic routing, accessing NEF and interaction with PCF.
4) Authentication Server Function (AUSF): Authentication server.
5) Network Slice Selection Function (NSSF): Responsible for network slice instances selection and allocating necessary AMF for users.
6) Access and Mobility Management Function (AMF) and Session Management Function (SMF): Handles mobility related procedures, authentication and authorization for the access layer, security anchor functionality and responsible for the termination of RAN control plane and Non-Access Stratum (NAS) procedures, protecting the integrity of signaling, management of registrations, connections and reachability.
7) Policy Control Function (PCF): Handle's policies and rules of 5G system. It can be accessed directly or via NEF depending on whether the AF is considered trusted or not.
8) Unified Data Management (UDM): Generates Authentication and Key Agreement (AKA). credentials and responsible for user identification handling, access authorization and subscription management.

9) User Plane Function (UPF): Distributed and configurable data plane that supports packet routing & forwarding, packet inspection, Quality of Support (QoS) handling, acts as external Protocol Data Unit (PDU) session point of interconnect to Data Network (DN), and is an anchor point for intra- & inter-RAT mobility.

*Figure 5-1: Figure 5.1 1: 5G Service-Based Architecture [28].*

As shown in figure 5.1-2, UPF has a major role in MEC integration as it is a distributed and configurable data plane and, in some deployment, locally it can be a part of MEC implementation. MEC orchestrator acts as an AF and interacts with network functions either directly or via NEF. NEF can be deployed centrally together with similar network functions, or an instance of NEF can also be deployed in the edge to allow low latency, high throughput service access from a MEC host. MEC hosts are most often deployed in edge or central data network. UPF steers the user plane traffic towards targeted MEC [28].

The distributed MEC host accommodates MEC apps, a message broker as a MEC platform service, and another MEC platform service to steer traffic to local accelerators. The choice to run a service as a MEC app or as a platform service is likely to be an implementation choice.



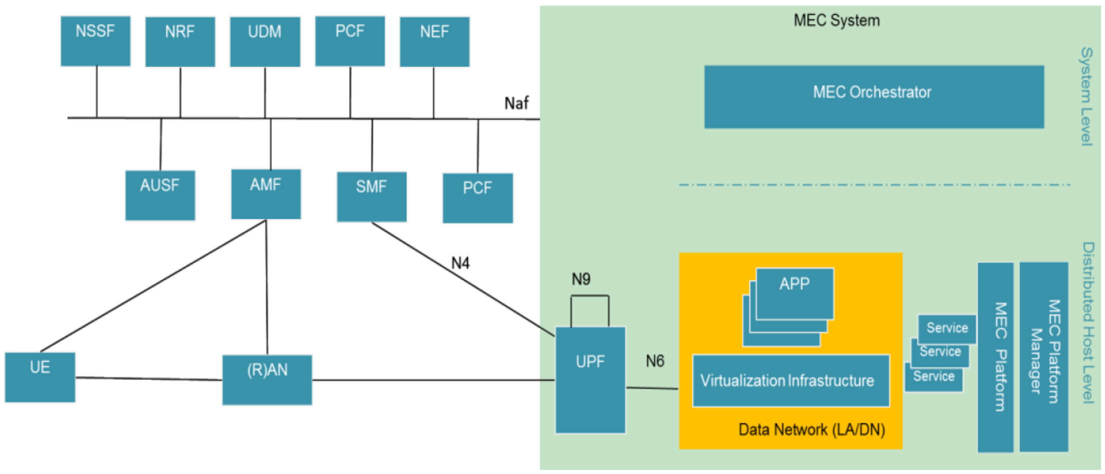


*Figure 5-2 : Integrated MEC deployment in 5G network [28].*

#### 5.1.1.3 MEC deployment

MEC can be flexibly deployed in different locations depending on requirements. It could be near the Base Station or near the central Data Network. The 4 standard examples are shown in figure 5.1-3.

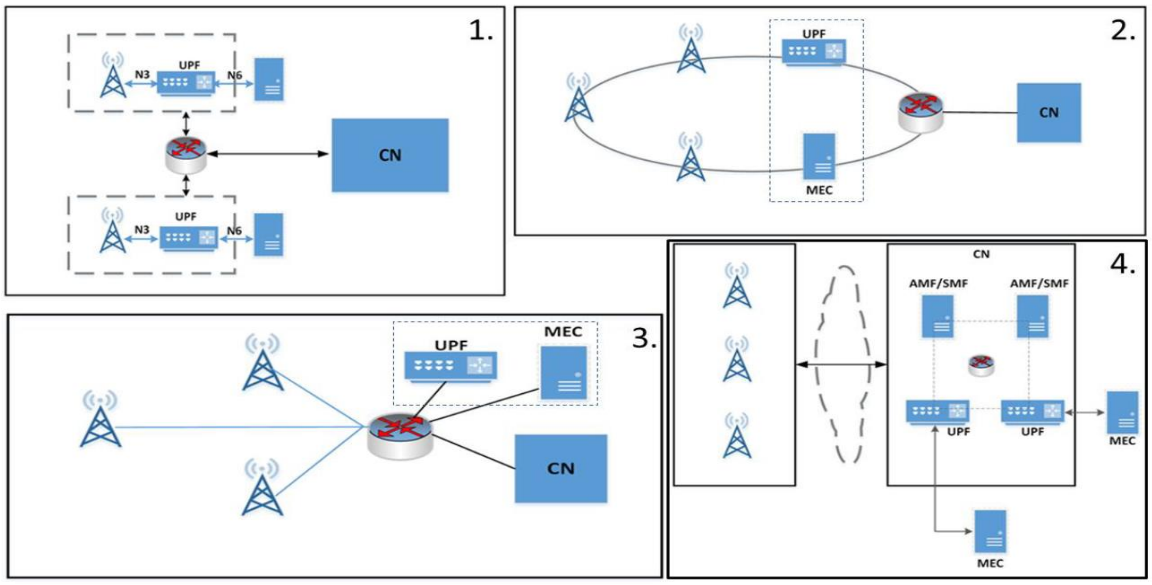


*Figure 5-3: Examples of the physical deployment of MEC [28]: (1) MEC and the local UPF collocated with the base station; (2) MEC collocated with a transition node, possibly with a local UPF; (3) MEC and the local UPF collocated with a network aggregation point; (4) MEC collocated with the core network functions.*

### 5.1.2 Open-RAN

#### 5.1.2.1 Motivation

The traditional wireless network infrastructure makes it difficult for data centers and service providers to tackle massive connections with different services requirements and provide reliable and efficient service to the users. The 5G network helps in solving the above problem with the help of Virtualization, MEC and Network Slicing.

The dependency of RAN part more on hardware and the vendor lock in is causing huge CAPEX and OPEX cost and restricting in building an intelligent cooperative reliable network. So, it is important to build 2G, 3G, 4G and 5G RAN solutions based on a general-purpose, vendor-neutral hardware and software-defined technology. Virtualization and RAN disaggregation in 5G helped in building such technology called Open-RAN (O-RAN) whose main principles are openness and intelligence.

#### 5.1.2.2 Architecture

The main components of ORAN are [29],

1) O-RU, O-DU and O-CU whose functionalities are similar to that in 5G disaggregated RAN except with added support of O-RAN based specifications and interface.
2) Near-Real Time RAN Intelligence Controller (RIC) for control/optimization of RAN elements and resources based on fine grained data using online AI/ML. It is suitable for application with latency needs of between 10ms and 1s.

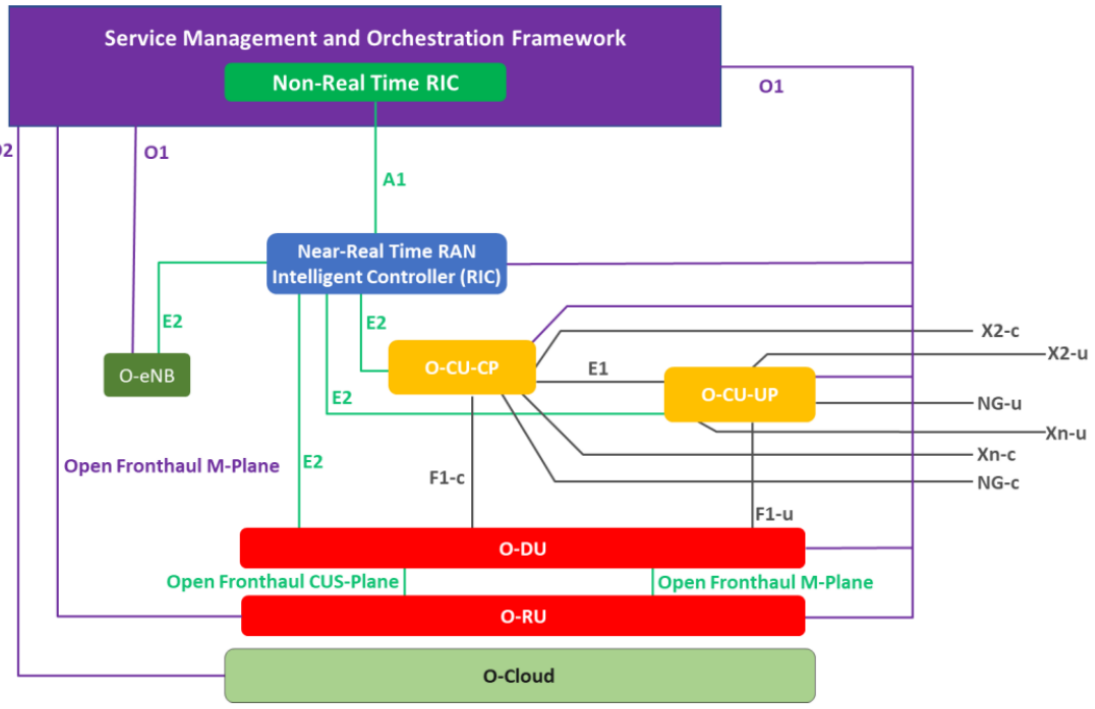
3) Non-Real Time RIC for Control/optimization of RAN elements and resources based on coarse grained data using online AI/ML. It is suitable for applications with latency requirements greater than 1s. It also provides policy based guidance to near-Real Time RIC. RIC components can be placed either in edge or core network depending on the usage scenario.

*Figure 5-4: O-RAN Architecture [29]*

### 5.1.3 Cell-free mMIMO

#### 5.1.3.1 Motivation

Conventional mobile networks (a.k.a. cellular wireless networks) are based on cellular topologies where a land area is divided into cells and each cell is served by one base station. Future wireless networks



are expected to manage to serve billions of devices simultaneously with high throughput demands for supporting many applications like voice, real-time video, high quality movies, etc. Cellular networks fail to handle such huge connections efficiently since user terminals at the cell boundary suffer from very high interference, and hence, perform badly. Furthermore, conventional cellular systems are designed mainly for human users. In future wireless networks, machine-type communications such as the Internet of Things, Internet of Everything, Smart X and many more are expected to play an important role. The main challenge of machine-type communications is scalable and efficient connectivity for billions of devices. Centralized technology with cellular topologies does not seem to be working for such scenarios since each cell can cover only a limited number of user terminals [32].

The combination between cell-free structure and massive MIMO (mMIMO) technology yields to the new concept: Cell-free mMIMO. Cell-free mMIMO network infrastructure is a beneficial epitome of the general distributed massive MIMO concept. By relying on time-division duplex (TDD) operation, a large number of geographically distributed antennas jointly serve a lower number of UEs with the aid of a fronthaul network and a central processing unit (CPU) operating with same time-frequency resource. The cellular or cell boundary concepts disappear in cell-free mMIMO. The basic premise behind cell-free systems is to reap all the benefits of network MIMO solutions by providing many more antennas than the number of users which allows us to invoke transmit pre-coding (TPC) for eliminating the interference at the UEs. The key properties of massive MIMO can be beneficially exploited for supporting scalable implementations. Then, the noise, fading and inter-user interference can be averaged out by relying on the law of large numbers. As a benefit, cell-free systems are capable of providing better service than conventional uncoordinated small cells [30].

#### 5.1.3.2 Architecture

Cell-free mMIMO consist of many geographically distributed Access Points (APs) with single/multiple antennas jointly serving a lower number of User Equipment's (UEs) with the aid of a fronthaul links and a Central Processing Unit (CPU) operating at the same time-frequency resource. The APs do not exchange any Channel State Information´s (CSI) and the coordination and power control between APs are handled by the CPU. Each AP of CF massive MIMO systems only requires local CSI to perform both transmit preprocessing and signal detection. It is beneficial for low latency machine critical applications.

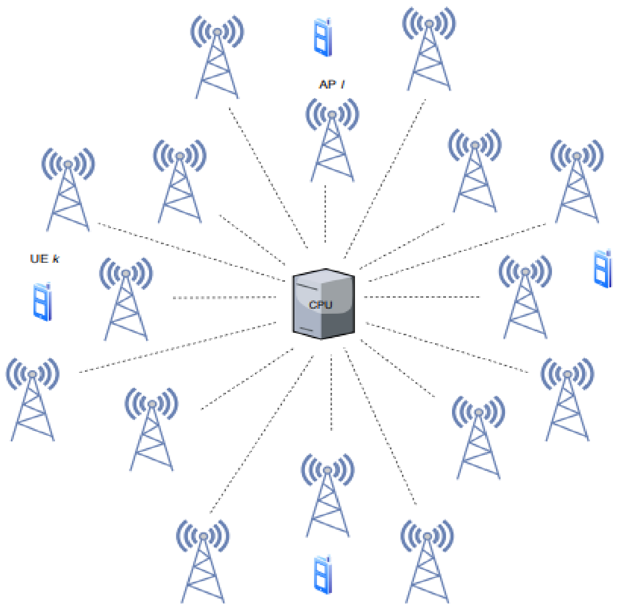


*Figure 5-5: Cell-free mMIMO system [31].*

## 5.2 ANN aided joint APs selection & beamforming computation for cell-free mMIMO O-RAN system

### 5.2.1 Cell-free mMIMO O-RAN system

The hardware and software supporting the O-RAN disaggregation concept can support the needs of cell-free mMIMO with fine granularity of the processing options (processing at AP vs CPU, Inter-CPU coordination). The CPU, which is one of the key components of cell-free mMIMO, is dis-aggregated into O-CU and O-DU. The APs are considered to be O-RU. Among many split options, distribution of RAN functions between the above parts of network is by function split 8 or split 7.2x, as per 3rd Generation Partnership Project (3GPP) specification. In Split 8, O-RUs are responsible only for converting signals while even the beamforming is shifted to the O-DU. This split requires very high fronthaul capacity. On the other hand, it can be attractive for a network where a vast number of (low price) O-RUs have to be deployed. In Split 7.2x, O-DUs are responsible for signal encoding and decoding, while O-RUs perform some light-weight processing. They can be used for centralized CF mMIMO networks while having lower fronthaul requirements. These split options make CPU processing easier and help in maintaining balance between fronthaul capacity and O-RU complexity depending on local processing to be done [32]. The RIC helps in intelligent service management and integration of 3rd party services. The AI/ML based dynamic control mechanism and any optimization algorithms can be installed in RIC. This opens up multiple problem areas related to power control, efficiency improvement, flexible localization, resource allocations, AP selection, clustering, reducing latency and load balancing.

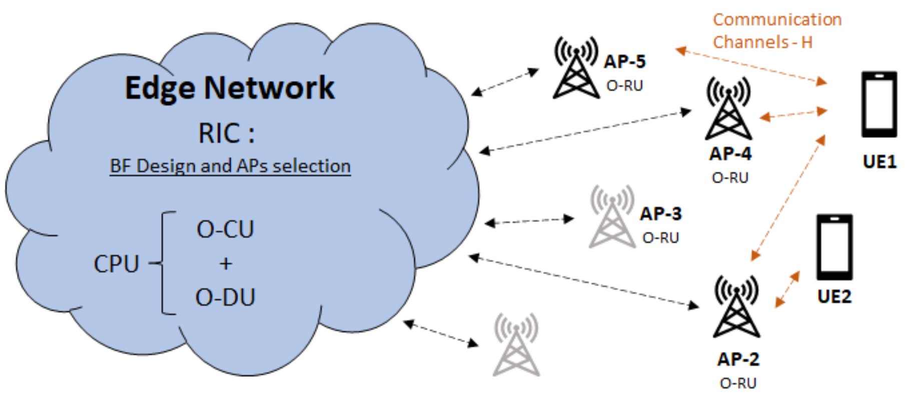


*Figure 5-6: Cell-free mMIMO O-RAN system*

### 5.2.2 Motivation

High spectral efficiency in cell-free is mainly due to the joint service offered by large number of distributed APs. Aligned with the Green Deal initiative promoted by the European Union and in line with current 5G/Beyond 5G(B5G) standards, energy efficiency is the fundamental metric in the forthcoming wireless generation. While power consumption in mMIMO mostly addressed issues on radiofrequency and baseband processing units, cell-free mMIMO must consider new energy-related aspects such as the presence of a dedicated fronthaul connection linking the APs to the CPU and the fixed power expenditure each individual AP entails. These fixed power terms are one of the majorly contributing metrics to the total energy expenditure making the consideration of activation of only few APs a wise option.

Large number of densely deployed APs causes interference and fading. Rapid changes in fading and high computation complexity causes difficulty in optimization of non-convex power control methods as they are np-hard. Load traffic conditions vary over space and time and some APs may be underutilized at certain time periods and only contribute to the increased overall energy consumption of the network. If an AP has negligible contribution to the system performance, it is beneficial to turn off that AP to save energy. Most research on cell-free focuses on problems like just beamforming design, energy efficiency or AP selection and clustering separately, addressing issues with simple setup like single CPU and few APs. Different AP switch on/off strategies were presented in [37], where the focus was on suboptimal heuristic algorithms to reduce the complexity when solving the NP-hard AP selection problem. The previous works still relied either on heuristics or conventional optimization methods [34]. This motivates to investigate the joint design of intelligent AP activation and beamforming design to further improve the energy usage in CF-MIMO.

### 5.2.3 Objective

The main aspects of proposed framework are summarized as follows,

1) Joint beamforming design and APs selection
2) Provide satisfied QoS to users
3) Intelligent and scalable optimization algorithm
4) Minimize power consumption
5) Reduce data overhead in the network

### 5.2.4 Methodology

A cell-free mMIMO communication system is considered in which N APs coherently serve K users/UEs that are randomly distributed on the coverage area. Each AP is equipped with M antennas, while the users are assumed to have a single antenna. A CPU connects with the APs via a fronthaul network. We aim to minimize the power consumption, while guaranteeing certain minimum QoS represented as $\gamma_o$ (in terms of SINR) for each user via joint beamforming design and AP subset ($b_n$) selection from total available APs N. The APs energy consumption includes the energy of electromagnetic radiation, the energy consumed in hardware components and the energy consumed in the backhaul. If there are large number of densely deployed APs, the AP that has negligible contribution to the system performance can be avoided to serve the UE by selecting a subset of APs that contributes significantly to user performance, thereby enabling the cell-free network to provide good service while achieving power savings and reduced fronthaul load.

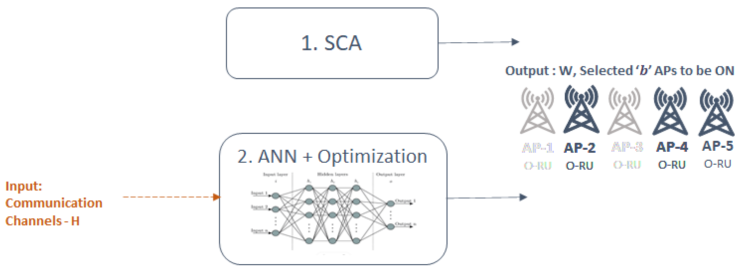

1. SCA
2. ANN + Optimization
Input: Communication Channels - H
Output : W, Selected '$b$' APs to be ON
AP-1
O-RU
AP-2
O-RU
AP-3
O-RU
AP-4
O-RU
AP-5
O-RU

*Figure 5-7: Joint beamforming design and APs selection.*

The mathematical formulation of the problem is Mixed Integer Nonlinear Programming (MINLP) as shown below, which is NP-hard. It can be reformulated as a non-convex Quadratically Constrained Quadratic Problem (QCQP). The minimization problem can be efficiently solved by Successive Convex Approximation (SCA), Alternate Direction Method of Multipliers (ADMM) and other statistical methods that provide fast computation, distributed optimization and parallel processing.

$$\min_{\{w_k\}_{k=1}^{K},\{b_n\}_{n=1}^{N}} \frac{1}{\eta}\sum_{k=1}^{K}||w_k||^2 + P_{FIX}\sum_{n=1}^{N} b_n$$

$$\begin{aligned} &subject\ to, \\ &\sum_{k=1}^{K}||w_k^{[n]}||^2 < P_n && n = 1,2,\dots N \\ &SINR_k \geq \gamma_o && k = 1,2,\dots K \\ &b_n \in \{0,1\} && n = 1,2,\dots N \end{aligned}$$

Here, $P_{FIX}$ is the signaling power of AP plus power consumed by the circuit and $P_n$ is the maximum power available at each AP. As of now, the optimization problem is solved in 2 ways. First, with the help of SCA optimization algorithm and second, using Artificial Neural Network (ANN) + Optimization. Second method is a 2-stage procedure performing AI based AP subset $b_n$ selection where ANN is trained to give best set of APs to be ON based on channel and reduced problem optimization for computation of beamforming weights $w_k$.

### 5.2.5 Results

The effectiveness of the proposed joint design is shown via numerical results in terms of total transmit power and computation time changing with respect to number of APs ON. For computation 6 APs each with 4 antennas are considered to be available in the environment along with 4 UEs.

Figure 5-8 compares the total transmit power achieved by proposed design (orange bar) where only an appropriate APs are selected to be active with the fact when all the APs are considered to be active (blue bar). The result for proposed designs SCA and ANN + optimization is same. It is evident that the total transmit power required is very high when all the APs are ON. When the number of APs are increased, the transmit power is also seen to be increasing incase when all APs are active. In case of proposed design increase in number of APs has no much effect on transmit power.

Figure 5-9 compares the computation complexity of both proposed algorithms i.e. SCA and ANN + optimization. It can be seen that, the SCA takes more time to jointly select APs and compute BF parameters and also with the increase in number of APs the computation complexity also increases. In case of ANN + optimization, the computation complexity remains the same for any number of APs and also faster than SCA.

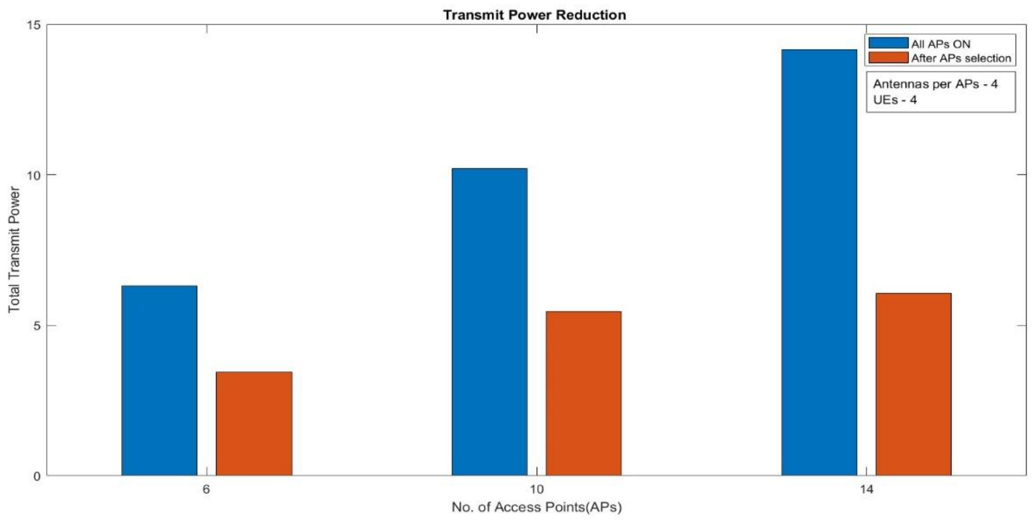


*Figure 5-8: Total transmit power vs Number of APs.*

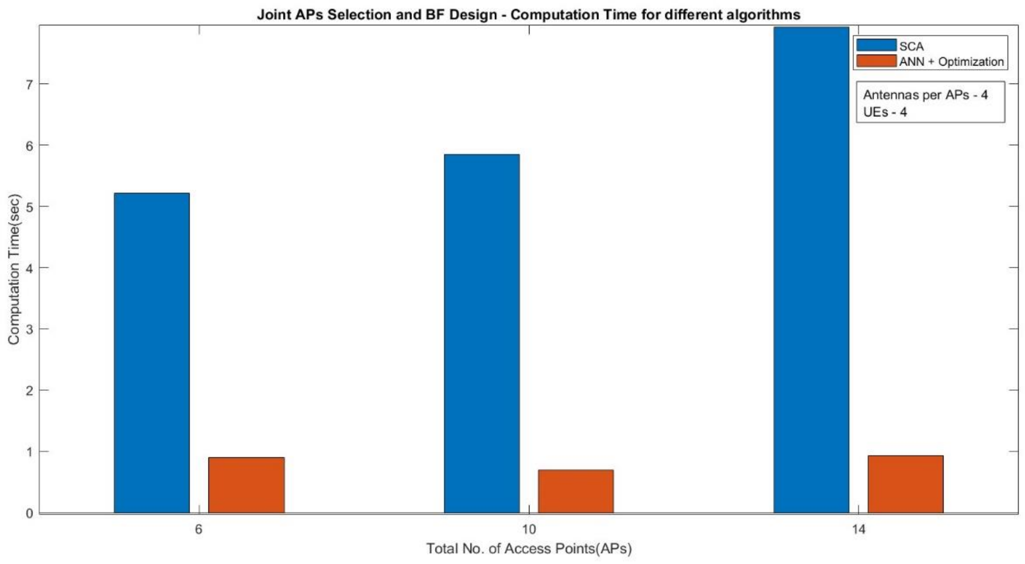


*Figure 5-9: Computation time vs Number of APs.*

### 5.2.6 Conclusions

Power efficiency is achieved by selecting proper subset of APs with the help of 2 proposed designs SCA and ANN + Optimization. AI-enhanced subset selection provides reduced computation time, but more fine-tuning of ANN is to be done. As a future work, Other mathematical tools like ADMM, deep reinforcement learning, branch and bound, etc., will be considered for comparison.

## 5.3 On the use of Probabilistic Forecasting for Network Analysis in Open-RAN

### 5.3.1 Motivation

In the realm of artificial intelligence (AI)-based prediction techniques, the shift from traditional single-point methods to advanced probabilistic forecasting techniques has significantly enhanced decision-making processes. While techniques like Long-Short Term Memory (LSTM) have been prevalent, probabilistic forecasting, exemplified by models like DeepAR and Transformer, introduces a novel dimension by providing decision makers with a spectrum of potential outcomes and associated probabilities. Concurrently, the Open Radio Access Network (Open RAN) architecture has emerged as a revolutionary framework for mobile networks, emphasizing openness, interoperability, and innovation. This work aims to synergize these advancements by proposing the integration of probabilistic forecasting techniques as a radio application (rApp) within the Open RAN architecture.

Focusing on the estimation of utilization and resource demands of Physical Resource Blocks (PRBs) in cellular base stations, we conduct a comprehensive comparison with traditional single-point forecasting methods. Through rigorous evaluations, we demonstrate the clear numerical advantages of probabilistic forecasting techniques, particularly highlighting the superiority of DeepAR over established methods like LSTM and Seasonal-Naive (SN), as well as other probabilistic techniques such as Simple-Feed-Forward (SFF) and Transformer neural networks. This research not only contributes to the evolving landscape of mobile networks but also underscores the potential for enhanced decision-making in Open RAN environments through the adoption of advanced probabilistic forecasting methodologies.

### 5.3.2 Objective

(1) Evaluate the suitability of a probabilistic forecasting approach when implemented as a radio application (rApp) within the O-RAN architecture.
(2) Investigate and compare various probabilistic methods and algorithms to estimate the expected utilization or resource demands of Physical Resource Blocks (PRBs) for different network functions and services within the O-RAN framework.
(3) Demonstrate how leveraging probabilistic forecasting techniques can provide valuable insights for mobile operators, aiding them in making informed resource allocation decisions, particularly in the context of PRB sharing.
(4) Conduct evaluations to showcase the numerical advantages of using probabilistic forecasting techniques over traditional methods, highlighting their capability to offer more accurate and reliable estimates of PRB utilization.
(5) Specifically, compare the performance metrics of DeepAR, including the Mean Square Error (MSE), Mean Absolute Scaled Error (MASE), and Mean Absolute Percentage Error (MAPE), against other probabilistic techniques such as Simple-Feed-Forward (SFF), Transformer, LSTM, and Seasonal-Naive (SN) baselines.
(6) Highlight DeepAR's superior performance with the achievement of the lowest MSE (0.065), lowest MASE (0.175), and a significantly lower MAPE (0.016) compared to the aforementioned probabilistic techniques.
(7) Emphasize the novelty of the research by stating that it is the first work to investigate the applicability of probabilistic forecasting as an rApp within the Open RAN architecture.

### 5.3.3 Methodology

(1) Data Collection: This work uses allocation data from PRB, which can be obtained through the O1 interface from O-CU in an O-RAN architecture. The dataset was created by simulating traffic and mobility patterns for various end users.
(2) Probabilistic Forecasting Techniques Selection: Identify and select a range of probabilistic forecasting techniques, including DeepAR, Simple-Feed-Forward (SFF), Transformer, LSTM, and Seasonal-Naive (SN) baselines. Understand the theoretical underpinnings and implementation details of each technique.
(3) Data Preprocessing: Cleanse and preprocess the collected data to handle missing values and outliers and ensure uniformity across different datasets. Normalize or scale the data as needed for effective model training and evaluation.
(4) Model Training: Implement the selected probabilistic forecasting techniques and train individual models using historical data. Split the datasets into training and validation sets to assess model performance during training. Fine-tune hyperparameters to optimize each model's performance.
(5) Evaluation Metrics: Define evaluation metrics such as Mean Square Error (MSE), Mean Absolute Scaled Error (MASE), and Mean Absolute Percentage Error (MAPE) to assess the performance of each model quantitatively. Establish a baseline for comparison using traditional single-point forecasting methods.

(6) Comparison and Analysis: Conduct a comprehensive comparison of the probabilistic forecasting models, emphasizing DeepAR's performance against other techniques and baselines. Analyze the numerical advantages of probabilistic forecasting in terms of accuracy and reliability in estimating PRB utilization.

### 5.3.4 Results

We compared our different models in Table 5-1. The models that use probabilities performed better than the baseline models (LSTM and SN) when looking at MSE, MASE, and MAPE. This is because baseline models are affected more by unusual data, uncertainty, and changes in the data pattern, which can make their predictions less accurate. Among the models we tested, DeepAR was the most accurate across all metrics. For example, it had the lowest MSE of 0.065, the lowest MASE of 0.175, and a much lower MAPE of 0.016. DeepAR also did well in terms of ND, Coverage, and QL, except for Coverage with a 10% prediction interval. This is because DeepAR's method considers both the average and the variability of the predicted outcomes, which helps it make accurate predictions while acknowledging the right level of uncertainty.

*Table 5-1: Accuracy Comparison*

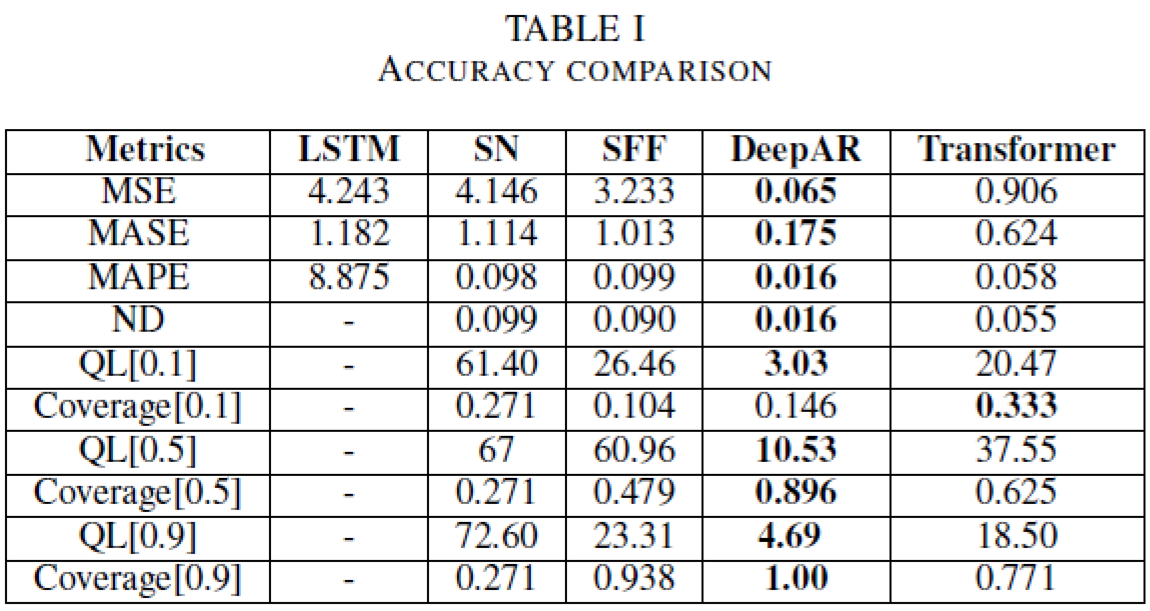


TABLE I
ACCURACY COMPARISON

| Metrics | LSTM | SN | SFF | DeepAR | Transformer |
|---|---|---|---|---|---|
| MSE | 4.243 | 4.146 | 3.233 | **0.065** | 0.906 |
| MASE | 1.182 | 1.114 | 1.013 | **0.175** | 0.624 |
| MAPE | 8.875 | 0.098 | 0.099 | **0.016** | 0.058 |
| ND | - | 0.099 | 0.090 | **0.016** | 0.055 |
| QL[0.1] | - | 61.40 | 26.46 | **3.03** | 20.47 |
| Coverage[0.1] | - | 0.271 | 0.104 | 0.146 | **0.333** |
| QL[0.5] | - | 67 | 60.96 | **10.53** | 37.55 |
| Coverage[0.5] | - | 0.271 | 0.479 | **0.896** | 0.625 |
| QL[0.9] | - | 72.60 | 23.31 | **4.69** | 18.50 |
| Coverage[0.9] | - | 0.271 | 0.938 | **1.00** | 0.771 |

Figure 5-10,Figure 5-11, and Figure 5-12 show the comparison between forecasting methods for future PRB values. The black line represents the true values, the blue line shows LSTM predictions, the red line represents SN predictions, and the olive green line shows probabilistic estimators. The figures reveal that LSTM and SN models produce less accurate predictions. In contrast, SFF and DeepAR predictions are better than those of the Transformer model and deterministic models. DeepAR stands out due to its ability to capture uncertainty, consider temporal dependencies, and use similar time series for training. It is a preferred choice over the Transformer and other models for time series forecasting. However, the suitability of any model may vary depending on the specific dataset and problem.

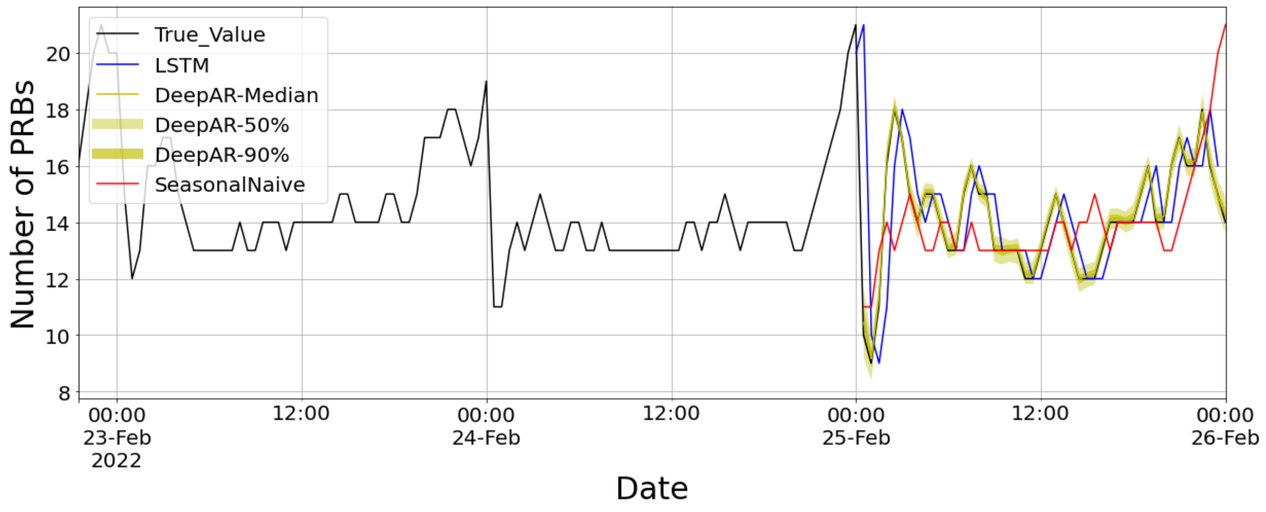


*Figure 5-10: Comparison of prediction of DeepAR estimator with single-point forecast models*

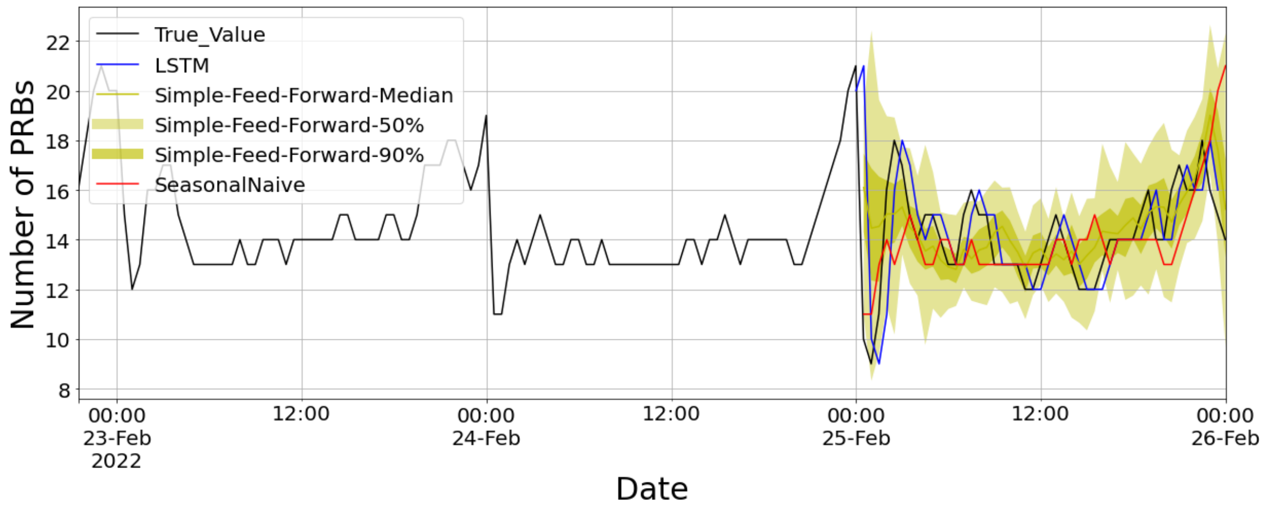


*Figure 5-11: Comparison of prediction of Simple-Feed-Forward estimator with single-point forecast models*

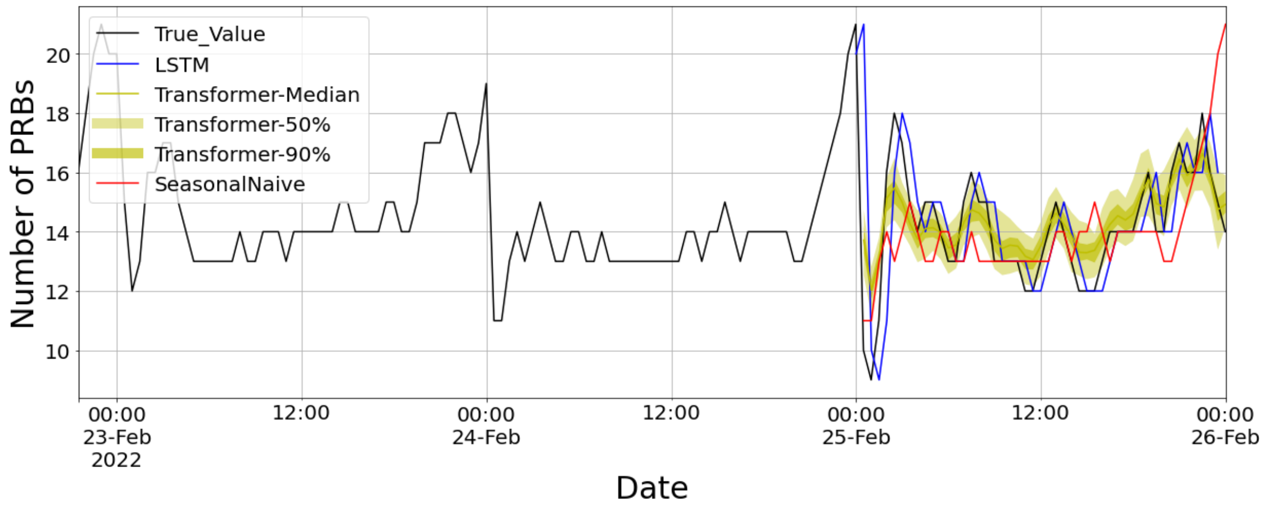


*Figure 5-12: Comparison of prediction of Transformer estimator with single-point forecast models*

In Figure 5-13, the histogram displays both baseline and probabilistic estimators. It is evident that the predictions from the DeepAR and Transformer models are closely centered around the true value. However, the SFF model's predictions show a broader dispersion, indicating increased uncertainty in the forecasting process, as previously observed in Figure 5-10. Notably, Figure 5-13 highlights that predictions from the baseline models deviate significantly from the true value.

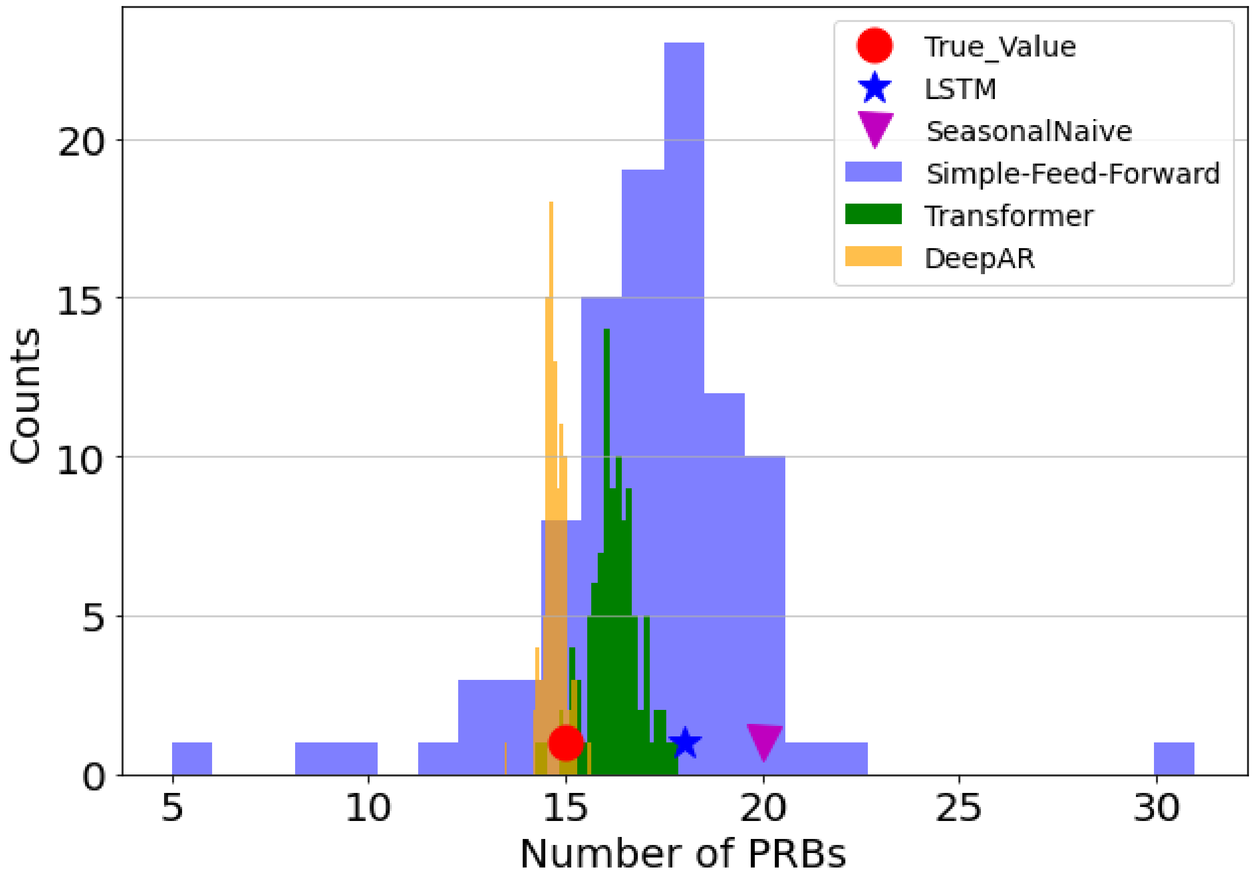


*Figure 5-13: Hist plot of particular timestamp*

## 5.3.5 Conclusion

This section compares probabilistic forecasting techniques, specifically DeepAR and Transformer, with single-point AI-based methods like LSTM, focusing on their applicability within the O-RAN architecture for PRB forecasting in cellular base stations. The results favor probabilistic forecasting, with DeepAR demonstrating lower MSE, MASE, MAPE, QL, ND, and higher Coverage values, indicating its superior accuracy and reliability in estimating PRB utilization. The paper also addresses important considerations when using probabilistic forecast methods in deep learning models, including limitations with small datasets, training time, complexity, and suitability for non-stationary datasets.

# 6 Conclusions and future plans

We addressed the problem of reducing the data center's energy footprint in a serverless edge computing scenario. We formulated an optimization problem aimed at minimizing the energy consumption of active servers in the data center by utilizing cold, warm, and running containers over a finite time horizon. As the problem turns out to be NP-hard, we investigated the optimum against a simple threshold based queueing solution through a small scale simulation setup. Our results reveal that the performance of the threshold-based queueing solution closely matches the optimum in terms of overall energy and memory consumption of the data center.

Through extensive numerical results, we have demonstrated that the proposed video streaming over HTTP algorithm outperforms traditional and contemporary DASH solutions under a vast set of performance metrics, enabling better utilization of reserved network/cache resources, full mitigation of video stalls and maximized user QoE. As future work, we aim to develop low-complexity variants of the proposed polynomial-time yet exact algorithm, investigating the trade-off between time complexity and performance. Future work also includes extension of our system model and proposed solution in view of multi-source video streaming scenarios.

As our proposed solution for the Hybrid IPS is fully completed, with respect to the implementation part, our next plan is to initially submit our work for publication in early 2024 with our final results extracted from the experiments executed in Lab206. Moreover, our work is going to be tested more extensively by using multiple Android devices gathering measurements in the same environment at the same time, as well as under different types of scenarios/circumstances. Last but not least, as I have already discussed with my supervisor about my future plans as an ESR, as I will continue my PhD studies at the UoA until completion and continue working on new research ideas and topics with respect to the Resource Orchestration For Massive Connectivity At The Network Edge and the 5,6G domain.

In conclusion, 5G serves critical functions such as communication, computation, control, and content delivery, expanding its role to support diverse applications like massive Machine Type Communication (mMTC) and Ultra-Reliable Low Latency Communication (URLLC). The integration of 5G with Edge Computing (MEC) and the Open RAN (O-RAN) architecture is pivotal for transforming telecommunication networks into versatile platforms, overcoming challenges related to scalability, efficiency, and vendor lock-in. The advent of cell-free mMIMO, facilitated by O-RAN disaggregation, presents a paradigm shift in wireless network design, promoting scalability and efficient connectivity while eliminating traditional cell boundaries. Furthermore, the strategic synergy between advanced probabilistic forecasting techniques, exemplified by models like DeepAR and Transformer, and the Open RAN architecture enhances decision-making processes, especially in estimating and managing resource demands in cellular base stations. This research signifies a notable advancement in the evolution of mobile networks, emphasizing the potential for improved decision-making within the Open RAN framework through the adoption of cutting-edge probabilistic forecasting methodologies.

# 7 References

[1] Y. C. Hu et al., “Mobile edge computing a key technology towards 5G,” tech. rep., ETSI, 2015.

[2] Z. Tao et al., “A survey of virtual machine management in edge computing,” in Proc. of the IEEE, vol. 107, no. 8, pp. 1482–1499, 2019.

[3] E. Jonas et al., “Cloud programming simplified: A Berkeley view on serverless computing,” arXiv:1902.03383, 2019.

[4] L. Pan et al., “Retention-aware container caching for serverless edge computing,” in IEEE INFOCOM, 2022.

[5] R. Xie et al., “Workflow scheduling in serverless edge computing for the industrial internet of things: A learning approach,” in IEEE Trans. on Ind. Informatics, pp. 1–10, 2022.

[6] I. E. Akkus et al., “SAND: Towards High-Performance serverless computing,” in USENIX ATC, 2018.

[7] M. Shahrad et al., “Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider,” in ATC, 2020.

[8] A. Mohan et al., “Agile cold starts for scalable serverless,” in HotCloud, 2019.

[9] P. Ruiu et al., “On the energy-proportionality of data center networks,” in IEEE Trans. on Sustainable Comp., vol. 2, no. 2, pp. 197–210, 2017.

[10] “Appendix.” https://www.dropbox.com/s/6mn1wp813eh3qpr/Appendix. pdf?dl=0.

[11] M. R. Gary and D. S. Johnson, “Computers and intractability: A guide to the theory of NP-completeness,” 1979.

[12] D. Xenakis et al., “Admission control and end-to-end slicing for video streaming in MEC-empowered cellular networks,” in IEEE GLOBECOM, 2022, pp. 3423-3428, Dec 2022

[13] HTTP/2-Based Methods to Improve the Live Experience of Adaptive Streaming

[14] https://bitmovin.com/adaptive-streaming/

[15] https://www.brendanlong.com/the-structure-of-an-mpeg-dash-mpd.html

[16] https://www.encoding.com/mpeg-dash/

[17] https://rybakov.com/blog/mpeg-dash/

[18] https://blog.desgrange.net/post/2017/04/17/encode-videos-dynamic-adaptive-streaming-http.html

[19]G. S. Park et al., “Video Quality-Aware Traffic Offloading System for Video Streaming Services Over 5G Networks With Dual Connectivity”, in IEEE Trans. on Vehic. Techn., vol.68, no.6, pp.5928-5943, Jun 2019.

[20] A. Mehrabi et al., “Edge Computing Assisted Adaptive Mobile Video Streaming”, in IEEE Trans. on Mob. Comput., vol. 18, no. 4, pp. 787-800, 1 April 2019.

[21] X. Huang et al., “Towards 5G: Joint Optimization of Video Segment Caching, Transcoding and Resource Allocation for Adaptive Video Streaming in a Multi-Access Edge Computing Network”, in IEEE Trans. on Vehic. Techn., vol.70, no.10, pp.10909-10924, Oct 2021.

[22]D. Xenakis, "To DASH, or not to DASH? Optimal Video Bitrate Selection and Edge Network Caching in MEC-empowered Slice-Enabled Networks," in IEEE Transactions on Vehicular Technology, May 2023.

[23] B. Zhou et al., "A Novel Access Point Placement Method for WiFi Fingerprinting Considering Existing APs," in IEEE Wireless Communications Letters, vol. 9, no. 11, pp. 1799-1802, Nov. 2020.

[24] L. Li et al., "A Hybrid Fingerprint Quality Evaluation Model for WiFi Localization," in IEEE Internet of Things Journal, vol. 6, no. 6, pp. 9829-9840, Dec. 2019.

[25] G. Huang et al., "WiFi and Vision-Integrated Fingerprint for Smartphone-Based Self-Localization in Public Indoor Scenes," in IEEE Internet of Things Journal, vol. 7, no. 8, pp. 6748-6761, Aug. 2020.

[26] C. Kumar et al., "Dictionary-Based Statistical Fingerprinting for Indoor Localization," in IEEE Transactions on Vehicular Technology, vol. 68, no. 9, pp. 8827-8841, Sept. 2019.

[27] T. Lan, X. Wang, Z. Chen, J. Zhu and S. Zhang, "Fingerprint Augment Based on Super-Resolution for WiFi Fingerprint Based Indoor Localization," in IEEE Sensors Journal, vol. 22, no. 12, pp. 12152-12162, June, 2022.

[28] ETSI White paper No.28., “MEC in 5G networks,” ISBN No. 979-10-92620-22-1, First edition June 2018.

[29] O-RAN Alliance White paper, “O-RAN use cases and deployment scenarios, Towards open and smart RAN,”, February 2020

[30] H. Q. Ngo, A. Ashikhmin, H. Yang, E. G. Larsson and T. L. Marzetta, "Cell-Free Massive MIMO Versus Small Cells," in *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1834-1850, March 2017

[31] W. A. Chamalee Wickrama Arachchi, K. B. Shashika Manosha, N. Rajatheva, M. Latva-aho, "On Max-Min SINR with MMSE Processing for Uplink Cell-Free Massive MIMO," [online] Available*: arxiv.org/pdf/1908.03187v1.pdf*, August 2019

[32] V. Ranjbar, A. Girycki, M. A. Rahman, S. Pollin, M. Moonen and E. Vinogradov, "Cell-Free mMIMO Support in the O-RAN Architecture: A PHY Layer Perspective for 5G and Beyond Networks," in *IEEE Communications Standards Magazine*, March 2022

[33] C. F. Mendoza, S. Schwarz and M. Rupp, "Deep Reinforcement Learning for Dynamic Access Point Activation in Cell-Free MIMO Networks," *WSA 2021; 25th International ITG Workshop on Smart Antennas*, 2021

[34] T. X. Vu, S. Chatzinotas, S. ShahbazPanahi and B. Ottersten, "Joint Power Allocation and Access Point Selection for Cell-free Massive MIMO," *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020

[35] Thomas Rojat, Raphael Puget, David Filliat, Javier Del Ser, Rodolphe Gelin, and Natalia Diaz-Rodriguez, “Explainable Artificial Intelligence(XAI) on Time Series Data: A Survey,” [online] Available*: arxiv.org/pdf/2104.00950v1.pdf,* April 2021

[36] H. D. Trinh, L. Giupponi and P. Dini, "Mobile Traffic Prediction from Raw Data Using LSTM Networks," *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2018

[37] Q. Guo, R. Gu, H. Yu, T. Taleb and Y. Ji, "Probabilistic-Assured Resource Provisioning with Customizable Hybrid Isolation for Vertical Industrial Slicing," in *IEEE Transactions on Network and Service Management*, 2022

[38] GluonTS – Probabilistic Time Series Modeling in Python, [online] Available: ts.gluton.ai/stable

[39] Vaishnavi k, Luis Blanco, Engine Zeydan, "On use of Probabilistic forecasting for network analysis in O-RAN," IEEE MEDITCOM, 2023

## List of Acronyms and Abbreviations

| *Acronym* | *Description* |
|---|---|
| *3GPP* | Third Generation Partnership Project |
| *5G* | 5th generation (5G) mobile network |
| *ABR* | Adaptive Bitrate |
| *AF* | Application Function |
| *AI/ML* | Artificial Intelligence / Machine Learning |
| *AMF* | Access and Mobility Management Function |
| *AP* | Access Point |
| *AUSF* | Authentication Server Function |
| *BBU* | Baseband Unit |
| *BFT* | Byzantine Faulty Tolerant |
| *biRNN* | Bidirectional Recurrent Neural Network |
| *BPP* | Binomial Point Process |
| *BS* | Base Station |
| *CAT* | Cache Allocation Technology |
| *CDF* | Cumulative Distribution Function |
| *CDN* | Content Distribution Network |
| *CFS* | Customer  Facing Service |
| *CGI* | Global Cell Identity |
| *CHR* | Cache Hit Ratio |
| *CNN* | Convolutional Neural Network |
| *COTS* | Common Off-The Shelf |
| *CP* | Content Provider |
| *CRAN* | Cloud Radio Access Network |
| *CRF* | Combined Recency and Frequency |
| *CS* | Content Server |
| *D2D* | Device-to-Device Communication |
| *DASH* | Dynamic adaptive streaming over HTTP |
| *DDIO* | Data Direct I/O |
| *DN* | Data Network |
| *DNS* | Domain Name System |
| *DP* | Dynamic Programming |
| *DPDK* | Data Plane Development Kit |
| *EAS* | Edge Application Server |
| *EEC* | Edge Enabler Client |
| *EES* | Edge Enabler Server |
| *eMBB* | Enhanced Mobile Broadband |
| *EPC* | Evolved Packet Core |
| *ESN* | Echo State Network |
| *ETSI* | European Telecommunications Standards Institute |
| *EWMA* | Exponential Weighted Moving Average |
| *FBS* | Femto Base Station |
| *FIFO* | First In First Out |
| *FL* | Federated Learning |
| *FNN* | Feedforward Neural Network |
| *HAS* | HTTP Adaptive Streaming |
| *HCPP* | Hard Core Point Process |
| *HHM* | Hidden Markov Model |
| *IMEI* | International Mobile Equipment Identity |
| *KPI* | Key Performance Indicator |

| | |
|---|---|
| ***LADN*** | Local Area Data Network |
| ***LFU*** | Least Frequently Used |
| ***LN*** | Lightning Network |
| ***LRFU*** | Least Recently/Frequently Used |
| ***LRU*** | Least Recently Used |
| ***LSTM*** | Long Short-Term Memory |
| ***MBA*** | Memory Bandwidth Allocation |
| ***MBS*** | Macro Base Station |
| ***MDF*** | Media Description File |
| ***MDP*** | Markov Decision Process |
| ***ME*** | Mobile Equipment |
| ***MEC*** | Multi-Access Edge Computing |
| ***MIMO*** | Multiple-Input Multiple-Output |
| ***ML*** | Machine Learning |
| ***MLP*** | Multilayer Perceptron |
| ***mMTC*** | massive Machine Type Communication |
| ***MNO*** | Mobile Network Operator |
| ***MPD*** | Model Predictive Control |
| ***NAS*** | Non-Access Stratum |
| ***NEF*** | Network Exposure Function |
| ***NFV*** | Network Function Virtualization |
| ***NLP*** | Neural Language Processing |
| ***NN*** | Neural Network |
| ***NRF*** | Network Resource Function |
| ***NSSF*** | Network Slice Selection Function |
| ***O-RAN*** | Open-RAN |
| ***OSS*** | Operations Support System |
| ***OTT*** | Over-the-Top |
| ***PC*** | Payment Channel |
| ***PCF*** | Policy Control Function |
| ***PCP*** | Poisson Cluster Process |
| ***PDU*** | Protocol Data Unit |
| ***PoA*** | Proof-of-Authority |
| ***PoS*** | Proof-of-Stake |
| ***PoW*** | Proof-of-Work |
| ***PPP*** | Poisson Point Process |
| ***QoE*** | Quality of Experience |
| ***QOS*** | Quality of Service |
| ***RAN*** | Radio Access Network |
| ***RAT*** | Radio Access Technology |
| ***RDT*** | Resource Director Technology |
| ***ReLU*** | Rectified Linear Unit |
| ***RIC*** | RAN Intelligence Controller |
| ***RL*** | Reinforcement Learning |
| ***RNN*** | Recurrent Neural Network |
| ***RRH*** | Remote Radio Head |
| ***RWP*** | Random Waypoint |
| ***SBA*** | Service-Based Architecture |
| ***SBS*** | Secondary Base Station |
| ***SC*** | Smart Contract |
| ***SCN*** | Small Cell Network |
| ***SINR*** | Signal to Interference Noise Ratio |

| | |
|---|---|
| ***SSC*** | Session and Service Continuity |
| ***TPS*** | Transactions per Second |
| ***UE*** | User Equipment |
| ***UPF*** | User Plane Function |
| ***URLLC*** | Ultra-Reliable Low Latency Communication |
| ***VL*** | Virtual Link |
| ***VNF*** | Virtual Network Function |
| ***VNFFG*** | Virtual Network Function Forwarding Graph |
| ***VoD*** | Video on Demand |