Computer Networks 188 (2021) 107829

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

5G network slices resource orchestration using Machine Learning techniques☆

Nazih Salhab [a,b,*], Rami Langar [a], Rana Rahim [b,c]

[a] LIGM, CNRS-UMR 8049, University Gustave Eiffel, Champs-sur-Marne 77420, France
[b] Doctoral School of Sciences and Technologies (DSST), Lebanese University, Tripoli, Lebanon
[c] Faculty of Science, Lebanese University, Tripoli, Lebanon

ARTICLE INFO

Keywords:
Network slicing
Machine-Learning
Resource orchestration
5G and beyond
OpenAirInterface OAI

ABSTRACT

To efficiently serve heterogeneous demands in terms of data rate, reliability, latency and mobility, network operators must optimize the utilization of their infrastructure resources. In this context, we propose a framework to orchestrate resources for 5G networks by leveraging Machine Learning (ML) techniques. We start by classifying the demands for resources into groups in order to adequately serve them by dedicated logical virtual networks or Network Slices (NSs). To optimally implement these heterogeneous NSs that share the same infrastructure, we develop a new dynamic slicing approach of Physical Resource Blocks (PRBs). On first hand, we propose a predictive approach to achieve optimal slicing decisions of the PRBs from a limited resource pool. On second hand, we design an admission controller and a slice scheduler and formalize them as Knapsack problems. Finally, we design an adaptive resource manager by leveraging Deep Reinforcement Learning (DRL). Using our 5G experimental prototype based on OpenAirInterface (OAI), we generate a realistic dataset for evaluating ML based approaches as well as two baselines solutions (i.e. static slicing and uninformed random slicing-decisions). Simulation results show that using regression trees for both classification and prediction, coupled with the DRL-based adaptive resource manager, outperform alternative approaches in terms of prediction accuracy, resource smoothing, system utilization and network throughput.

1. Introduction

Mobile networks are anticipated to provide three classes of services known as enhanced Mobile Broadband (eMBB), massive Machine Type Communication (mMTC) and ultra-Reliable Low-Latency Communication (uRLLC) [1]. According to a feasibility study technical report from Third Generation Partnership Project (3GPP) [2], each class (eMBB, mMTC and URLLC) has its requirements in terms of throughput, mobility, reliability, latency, energy efficiency, in addition to different connectivity and traffic densities. In order to keep costs affordable, mobile network operators have to optimize their resources to serve all of these heterogeneous demands. This is particularly interesting due to natural scarcity of resources, whether external (spectrum and power) or infrastructural (compute, networking, and storage). Leveraging two enabling technologies, namely Software-Defined Networking (SDN) and Network Function Virtualization (NFV), a solution consists of provisioning dedicated logical networks, also known as Network Slices (NSs) [3], to assure required Quality of Service (QoS). The first step consists of classifying the requests. In particular, we consider QoS Class Identifiers (QCIs), in order to have each type of traffic associated to a class of service supported by a tailor-made NS. NSs are fine-tuned slices that are optimized to maximize service level objectives while meeting underlying system's constraints. Note that we denote by slicing ratios the amount of Physical Resource Blocks (PRBs), in the Radio Access Network (RAN), that are divided among the instantiated slices. Aiming to provide an End-To-End (E2E) QoS network design, we implement four building blocks, including: (i) demands classification, (ii) optimum slicing ratios prediction, (iii) admission control coupled with scheduling, and (iv) adaptive resource management. In this context, we aim to address resource orchestration for 5G network slices. Recall that resource orchestration is about effectively consuming available scarce resources, streamlining them into capabilities and leveraging the capabilities to create added-value through an optimized performance. Our proposed resource orchestration process acts as follows. After demands classification, we address predicting the optimum slicing ratios for several NSs sharing resources from a limited resource pool bounded by underlying systems capacities. To do so, we first propose employing

☆ A preliminary version of this paper appeared in the proceedings of the 2019 IEEE Global Communication Conference (GLOBECOM 2019).

* Corresponding author at: LIGM, CNRS-UMR 8049, University Gustave Eiffel, Champs-sur-Marne 77420, France.
E-mail addresses: nazih.salhab@univ-eiffel.fr (N. Salhab), rami.langar@univ-eiffel.fr (R. Langar), rana.rahim@ul.edu.lb (R. Rahim).

https://doi.org/10.1016/j.comnet.2021.107829
Received 14 August 2020; Received in revised form 27 December 2020; Accepted 7 January 2021
Available online 13 January 2021

Machine Learning (ML) techniques and particularly, Regression Trees (RTs) for demands classification and slicing ratios forecasting. Recall that an RT is a graph that uses branching method to illustrate every possible outcome of a decision. Then, we propose a Deep Reinforcement Learning (DRL) implementation for an adaptive resource manager. We validate our proposal using an experimental 5G prototype [4], which is implemented as a micro-service oriented architecture based on OpenAirInterface™ (OAI) [5] and Docker [6] containers. Specifically, we configure multiple slices, generate a dataset and benchmark the performance of several ML models as well as two baseline solutions, namely static slicing and un-informed random slicing-decisions. Accordingly, we compare the ground-truth with our predicted value of slicing ratios, and we analyze the effect of resource orchestration using four different metrics: Number of Uplink (UL) PRBs, Buffer Status Report (BSR), system utilization and network throughput.

The main contributions of our paper can be summarized as follows:

- First, we comprehensively review the state of the art on resource orchestration for 5G network slices.
- Second, we design a framework consisting of four building blocks for resource orchestration.
- Third, we provide formulation, models and algorithms to implement these building blocks. Specifically, we propose (i) using ML techniques for classification, (ii) predicting slicing ratios based on RTs, (iii) modeling admission control and scheduling as Knapsack optimization problems and (iv) leveraging DRL for adaptive resource management.
- Finally, we show the effectiveness of our proposal using our 5G experimental prototype based on OAI including ground-truth measurements and other metrics, namely, UL PRBs, BSR, system Central Processing Unit (CPU) utilization and network throughput.

The remainder of this paper is organized as follows. In Section 2, we present an overview of the related works. Section 3 describes our system design and details our proposed models for implementing the building blocks of our framework. Section 4 presents the performance evaluation including a description of our prototype, used dataset, and the discussion of the results. We conclude this paper in Section 5.

2. Related work

Resource orchestration has triggered interest among researchers in the past few years. In what follows, we discuss a selection of relevant papers grouped by research areas.

2.1. Classification and slicing approaches

Authors in [7] proposed a framework integrating various ML algorithms, SDN and NFV. They used a traffic classification module and network slicing for self-organizing networks. In addition, they implemented such traffic classification and network slicing for eMBB, but they did not consider admission control, and scheduling processes in their proposed framework.

On another hand, Authors in [8] investigated a management and orchestration architecture incorporating SDN and NFV for instantiating and managing the federated network slices. They elaborated on their proposed architecture, but, they did not address the validation of such architecture in a 3GPP compliant testbed.

Authors in [9] used a DRL-based approach to allow network entities to learn about the network, aiming to make optimal decisions related to network slicing in 5G. They concluded that a DRL outperforms Q-learning as well as greedy and random approaches in terms of average utility per service request, but they did not provide implementation details.

A multi-access edge computing broker to answer heterogeneous tenant demands and related privileges in a network slicing environment was proposed by authors in [10]. They devised an orchestration mechanism able to fulfill tenant requests while avoiding Service Level Agreement (SLA) violations, but they did not consider the elasticity of cloud resources.

2.2. Traffic and resource prediction approaches

Authors in [11] proposed a supervised ML model based on Decision Trees (DTs) for root-cause-analysis of QoS degradation. They observed that DTs can predict future QoS anomalies with confidence in order to proactively exploit these findings. They shed some lights on a real use case where inputs about path, devices, boards, ports and links faults are processed to detect anomalies, but, no validation through system implementation was included. Conversely, in this paper, we elaborate on the implementation of 5G experimental prototype used to validate our proposal in Section 4.

Authors in [12] used feature-selection based prioritization to predict mobile traffic leveraging data from an open dataset used in a big-data challenge. They proposed to reduce the volume of traffic log data sent from base stations to the server while maintaining high prediction accuracy, but they did not consider adaptive resource management of the server.

Authors in [13] used a stochastic model to represent time series data using hierarchical hidden Markov model, which includes two nested hidden Markov chains and one observable process. Knowing that MCs are memory-less by design, they did not consider the elasticity provided by cloud resources.

A multi-objective genetic algorithm to optimize resource allocation while minimizing CPU and memory utilization and the energy consumption was proposed by authors in [14]. Their approach consisted of forecasting the resource requirement according to historical time slots in addition to Virtual Machines (VMs) placement, whereas, in our work, our formulation applies not only to VMs but also to containers that are suitable for a cloud-native deployment [15]. We used Docker containers [6] for implementing our platform used for evaluating our proposals. Moreover, in our previous work [4], we demonstrated a micro-service based deployment of 4G EPC core using OpenAirInterface (OAI). Finally, multiple authors used machine learning-based approaches for time series predictions [16]. This includes Trees [17], K-Nearest Neighbor (KNN) [18], Discriminant-based [19], Random Forests [20], Support Vector Machine (SVM) [21] and Gaussian Process Regression (GPR) [22]. We use these techniques as baselines, when benchmarking our results.

2.3. Admission control and scheduling optimization approaches

Authors in [23] presented a testbed called OVNES (OVerbooking NEtwork Slices) in charge of collecting network statistics, predicting traffic behaviors by leveraging ML and applying admission control policies to select requests increasing network efficiency and scheduling. The paper did not provide details about the implementation of these modules. Same authors in [24] designed a hierarchical control plane to manage the E2E orchestration of slices. They formulated the orchestration problem as a stochastic yield management problem and proposed optimal and heuristic approaches. However, they assumed a static maximum capacity of resources, which is not the case in cloud-oriented deployment, as there is elasticity of cloud resources through auto-scalability.

An intelligent resource scheduling strategy for 5G RAN by exploiting a collaborative learning framework leveraging deep-learning and re-inforcement learning was proposed in [25].

Authors in [26] designed a network slice admission control algorithm leveraging ML that learns the best acceptance policy while satisfying service guarantees to tenants. They provided an analytical

model for slice admissibility, analyzed the system using a Semi-Markov decision processes and optimized the benefits using a practical ML approach.

Authors in [27] proposed an admission control algorithm using a multi-unit combinatorial auction model to determine fast winner when reserving resources with performance guarantees. They developed a reinforcement learning-based utility-maximizing strategy to distribute resources across tenants.

Authors in [28] proposed a DRL based approach for optimizing network latency in an SDN context. They collected optimal paths form the DRL agent and aimed at predicting future demands using deep neural networks. Moreover, they formulated the flow rules placement as an integer linear program to minimize the total network delay.

Inspired by these works, we formalize four building blocks along with their models and implement the whole system in a 5G experimental prototype.

2.4. Adaptive resource management approaches

A framework for the configuration of radio resource management in a sliced RAN using static slicing ratios was proposed by authors in [29]. They evaluated the blocking rate and the throughput per data radio bearer of different types of slices. However, to achieve efficient resource allocation within a changing environment, dynamic slicing based on traffic load is necessary.

Authors in [30] designed three key building blocks for network slicing, namely a forecasting module, an admission control agent and a scheduler. They used Holt–Winters method for traffic prediction. They tolerated some violation of SLAs for an increase in resource utilization. Details of the used dataset and its characteristics were not provided. In contrast, we include an adaptive resource manager allowing to avoid SLA violations and guarantee isolation between slices. Also, we elaborate the details of our dataset generation.

On the other hand, authors in [31] proposed using Seasonal Auto-Regressive Moving-Average (SARIMA) for predictions. Although, SARIMA is not complex as it does not rely on predictor variables, it fails when there is an unusual growth or slowdown in the time series trends. Conversely, predictor-based approaches capture such change through its predictors.

We propose a framework for resource orchestration in 5G networks consisting of multiple stages leveraging ML techniques in contrast to the aforementioned approaches using Holt–Winters method or SARIMA. Note that ML based approaches are expected to be competitive in accuracy thanks to data used for training the ML models [32].

As a conclusion, differently from these works, which focused on a single QoS aspect, in our work, we design a comprehensive QoS provisioning framework including classification, forecasting, admission control, scheduling and resource management to fill the gaps that we identified in the state of the art.

3. System design

The design of our network slices orchestrator, depicted in Fig. 1, is inline with the 3GPP technical specification detailing the concept and requirements for network sharing and management architecture [33]. After classifying tenants demands, accomplished by the Gatekeeper building block, the tenants traffic profile along with classified requirements are used to predict the adequate slicing ratios for each of the classified demand. This process is followed by admission control, resource management and scheduling processes. The starting point is to get NSs requirements grouped in terms of network characteristics such as spectral efficiency, latency, reliability, and energy efficiency, for a service instance [34]. We use supervised ML for implementing the classification [35]. The Decision Maker building block is composed of two sub-modules: a Forecast Aware Slicer using ML based regression and an Admission Controller that either grants or denies resource requests according to current and predicted loads. Two outcomes are anticipated from the admission controller. The granted requests are forwarded to the Slice Scheduler to be served in the nearest time window. The observations about denied requests, in case of no admission, are sent to the adaptive Resource Manager to train its resource management techniques. We use DRL by implementing an automatic flow control system to efficiently handle high resource utilization and maximize the throughput. The Slice Scheduler provides a feedback to the Decision Maker to close the optimization loop and serve the postponed requests. In what follows, we present in details the aforementioned building blocks by formulating each sub-problem and proposing corresponding solutions.

Table 1
Standardized QCI characteristics [36].

l	QCI	Bearer type	Delay budget	Loss rate	Priority
0	1	GBR	100 ms	10^{-2}	2
1	3	GBR	50 ms	10^{-3}	3
2	6	non-GBR	300 ms	10^{-6}	6
3	65	GBR	10 ms	10^{-2}	0.7
4	66	GBR	100 ms	10^{-2}	2

3.1. Gatekeeper model and problem formulation

Based on slice blueprints, used as an input to our orchestrator, in Fig. 1, we implement an initial phase of classification of the demands. It is impractical to let every use case dictate a tailor-made network slice to meet its requirements. Instead, a simple approach consists of aggregating the traffic per slice type. Table 1 reports our view of some QoS classes of traffic according to blueprints and the SLAs [36]. For instance, when traffic class $l = 3$, QCI = 65 with Guaranteed Bit Rate (GBR) bearer type, delay budget=10 ms, packet loss tolerance = 10^{-2}, and a priority of 0.7, we can say that the Mission Critical Push To Talk (MCPTT) service can be fit [36].

Denoting by $r_h^{(l)}(t)$ a request of a tenant h for a traffic class (l) over time (t), every instance (or event) of a point process ξ can be represented by $\xi_h^{(l)} = \sum_{t=0}^{T} \delta_t r_h^{(l)}(t)$ to constitute a feature vector, where δ_t denotes the Dirac function.

Using classification formulation elaborated in [37], let us denote by k a possible category, and by α_k the transpose of the corresponding weights vector. Recall that a weight vector is a set of parameters that are calculated during the training phase to correctly classify the training set and maximize the utility function. Our classification problem consists of assigning a score to each possible category k through the multiplication, using a dot product, of the feature vector of an instance by its related weights vector. The selected category would be the one with the highest utility resulting from assigning instance h to category k. Accordingly, our utility function can be formulated as follows.

$$\text{utility}(\xi_h^{(l)}, k) = \alpha_k \cdot \xi_h^{(l)} \tag{1}$$

This formulation applies to multiple classification techniques including regression trees. Recall that a regression tree is built using a binary recursive partitioning, which is an iterative process that splits the data into partitions or branches. Then, it continues splitting each partition into smaller groups as the method moves up through each branch [17].

3.2. Decision maker model and problem formulation

3.2.1. Forecast aware slicer

A traffic profile is a graph of network traffic based on data collected over a profiling time window. This serves as an input to the orchestrator for an enriched decision making process. Based on these consolidated traffic profiles, used as an input in Fig. 1, the forecast aware slicer predicts the optimum slicing ratios for the different slices in order to have a good starting point for the slicing process. Accordingly,

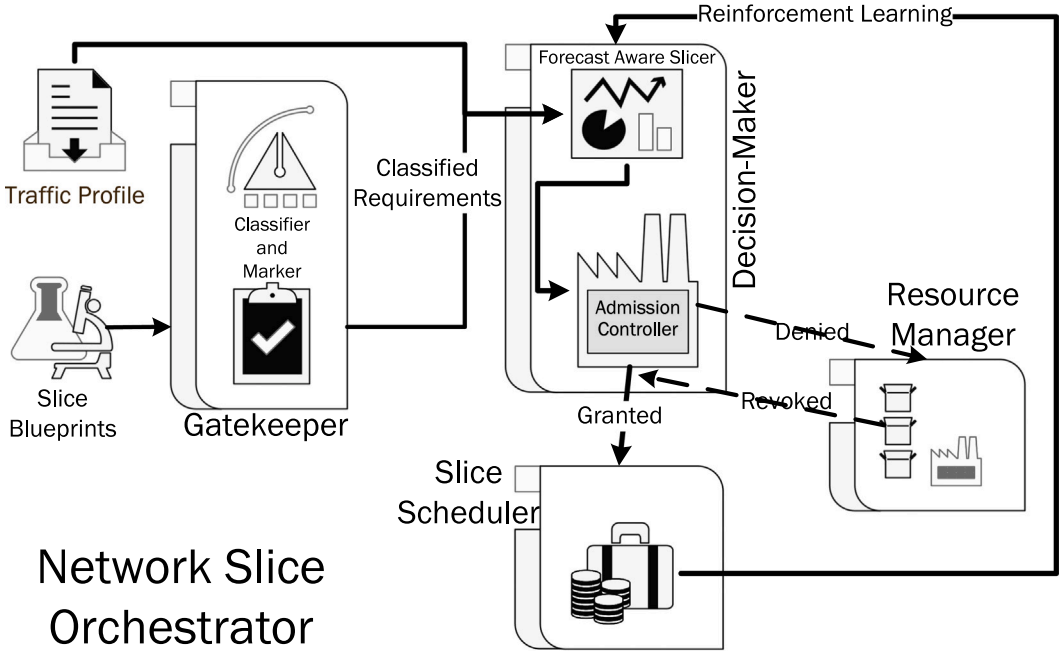


Fig. 1. Block diagram of 5G network slice orchestrator.

in this module, we use multiple predictor variables extracted from the enriched historical traffic profile, namely: timestamp, day-of-the-week, planned-event existence, and cloudy conditions environmental factor, to predict optimal values for the slicing ratio. Note that cloudy conditions and other environmental factors affect the slicing forecast in two ways. From technical point of view: Radio propagation is affected in general by moisture content in the air, as water tends to absorb electro-magnetic waves reducing the range and bandwidth of wireless systems. From ecological point of view: cloudy conditions are usually coupled with a decrease in term of mobility. Accordingly, users tend to use more frequently their data connections. Accordingly, the demands for radio resources is typically increased.

To choose the best performance forecasting technique, we evaluated several methods including the RTs. Denoting by $X = (x_1, x_2, \dots, x_n)' \in \mathbb{R}^n$, a vector of n predictor variables, and by $y \in \mathbb{R}$, a scalar output denoting the response variable; we formulate our regression model as follows.

$$y = f(X, \beta) + e \tag{2}$$

where e is an independent random noise involved in the statistical relationship between response variable y and predictor variables x_i allowing a non perfect deterministic relation. Parameter $\beta = (\beta_1, \beta_2, \dots, \beta_n)'$ is a vector of n unknowns that are evaluated during the training based on the chosen regression model by minimizing the sum of squared errors (S), that is considered as our cost function. In particular, our proposed objective function for growing an RT consists of minimizing the sum of squared errors over all the leaves c of our tree $\mathbb{T}$ as follows:

$$S^* = \text{minimize} \sum_{c \in leaves(\mathbb{T})} \sum_{g \in c} (y_g - m_c)^2 \tag{3}$$

where $m_c = \frac{1}{n_c} \sum_{g \in c} y_g$ is the prediction for leaf c having n_c points in it.

Please note that we will focus on the RTs as they provide the best performance compared to other methods, as we will see in the evaluation section.

3.2.2. Proposed algorithm for growing regression trees

We propose a simple yet efficient algorithm (Algorithm 1) for growing regression trees. Starting initially with a current node containing all the (M) points, we calculate the prediction for leaf (c) and the cost function (S^*). Note that the object node is an internal variable denoting the current node in which the decomposition is taking place. Let us denote by (q) a quality indicator that is a function of the minimum leaf size for the sought tree and by parameter (ε_0) the threshold for the largest variation of $\Delta(S)$. The search for optimum S^* is repeated until either the largest decrease of S would be less than ε_0 or one of the resulting nodes would contain less than q points. The result of this algorithm is a grown tree and its depth that allows us to predict the adequate slicing ratios. These are used as a starting point for the split of the PRBs. Afterwards, the admission control takes place.

Algorithm 1: ML-based Regression Tree Growing

Data: M points of k predictors and their responses
Result: Grown Tree $\mathbb{T}$ for responses and its depth δ

```
1  do
2      Initialize a single node containing all M points;
3      Calculate m_c and S;
4      if ∀ points in current node, predictors are same then
5          break;
6      else
7          search over all binary splits of all variables for the one
           which reduces S
8      end
9      if (Max(Δ(S)) < ε_0 or ∃ cardinal(node) < q) then
10         break;
11     else
12         take that split and create 2 new nodes
13     end
14 while no more new nodes;
15 S* ← S;
16 return grown tree and its depth δ
```

3.2.3. Complexity analysis of regression tree growing

Based on Algorithm 1, our regression tree calculates a quality condition that is used as a stopping criterion (line 9) before proceeding with the split of the data. It does this, for each predictor in every node

that is not a leaf node. The process repeats as long as there are some levels (affecting the depth) to be treated. Denoting by M the number of points used for training, in the best case of a balanced tree, the depth δ is $\mathcal{O}(\log_2(M))$ because of the split into two nodes (line 12). However, in the worst case of depth, δ is $\mathcal{O}(M)$ because each split decomposes the data in 1 and $(M' - 1)$ examples, where M' is the number of points of the current node. Denoting by k the number of predictors, the time complexity for the regression tree growing is $\mathcal{O}(k.M.\delta)$, that corresponds to $\mathcal{O}(k.M^2)$ in the worst case or $\mathcal{O}(k.M.\log_2(M))$ in the best case.

3.3. Admission controller

An Admission Controller receives the requests that need to be scheduled. Based on the current system load (l) and supported by the data generated by the Forecast Aware Slicer, it decides whether to grant or deny each individual request according to its priority class, as reported in Table 1.

Afterwards, granted requests are sent to the Slice Scheduler. In addition, it forwards the observations to the Resource Manager so that a reinforcement learning takes place. Accordingly, for high priority resource requests, a pool re-dimensioning could take place in order to decrease the chances of service denial, taking into consideration the capabilities of underlying system and the availability of additional infrastructure resources. We will address the priority concept in Section 3.5.

3.3.1. Problem formulation and resolution

At an instant (t) , we denote by x_{ij} a binary decision variable indicating whether a request j is served by slice i and thus admitted into the system or not. Index variables n and m denote the number of requests and the number of slices, respectively. Each admitted request j is valued as v_j to reflect its individual revenue v_i corresponding to the amount of consumed resources. In this context, we assume that a slice tenant pays a monetary amount corresponding to the consumed resources. For simplicity, we will not get into a particular pricing of a multi-tenancy environment. Several models are available online by major Cloud service providers, such as Google Cloud Provider (GCP) [38] or Amazon Web Services (AWS) [39]. We formalize the Admission Controller problem as D-dimensional Multiple-Choice Knapsack problem that is bounded by D constraints imposed by the hosting system capacities. The admission controller problem is formalized as follows.

$$\max_x a = \sum_{i=1}^{m}\sum_{j=1}^{n} v_j x_{ij}^{(t)} \tag{4a}$$

s.t.

$$\sum_{j=1}^{n} w_j^{(d)} x_{ij}^{(t)} \leq C_i^{(d)(t)},\ d \in \{1,\dots,D\},\ i \in \{1,\dots,m\} \tag{4b}$$

$$\sum_{i=1}^{m} x_{ij}^{(t)} \leq 1, \qquad j \in \{1,\dots,n\} \tag{4c}$$

$$x_{ij}^{(t)} \in \{0,1\}, \qquad i \in \{1,\dots,m\},\ j \in \{1,\dots,n\} \tag{4d}$$

The objective function in (4a) aims to maximize the value resulting from the resource utilization, while serving network slice requests having weights $w_j^{(d)}$ expressed in terms of the D system capacities. The set of constraints (4b) specifies the constraints on the infrastructure computing resources in regard to the required demands in terms of the D capacities. These resources cannot bypass upper bounds imposed by underlying system capabilities denoted by $C_i^{(d)}$. Assuming that a single task is assigned to exactly one system, constraint (4c) enforces such exclusivity. Constraint (4d) ensures atomic mapping with binary values. In the evaluation Section 4, we will consider that the number of capacities D is 2, representing the number of virtual Central Processing Units (vCPUs) and the amount of memory [14].

Problem (4) is NP-hard [40], but can be solved using a polynomial time algorithm, listed in Algorithm 2 [41], and described as follows. The process starts by sorting the constraints on computing resources in increasing order to identify the bottleneck constraint. Note that we mean by bottleneck constraint, the tightest dimension from the D-dimensions of the Multiple Choice Knapsack Problem (D-MCKP) that is first consumed when solving the mapping problem. Accordingly, the index of such bottleneck is denoted d^*. We reject any network slice requests for which requirements cannot be satisfied. Then, we calculate the efficiency of each request defined as the ratio between the network resource utilization value v of each request and the bottleneck constraint value $C^{(d^*)}$. These values are then sorted in decreasing order starting by the requests with the highest efficiency. This process is reiterated in order to include additional requests that satisfy the constraints on resources. The algorithm stops once no more requests can be satisfied. In this case, the final network resource utilization as well as the selected set of slice requests to be served are obtained.

Algorithm 2: Heuristic for D-dimension Multiple Choice Knapsack Problem

Data: Requests r_j with their values v_j and weights w_j, systems s_i with constraints $C_i^{(d)}$;
Result: Max resource utilization mapping

```
1  Sort the constraints on s_i in increasing order;
2  Prioritize the bottlenecks (to get d*);
3  for r_j = 1 to n do
4  |  for constraint d = 1 to D do
5  |  |  if (w_j^(d) > C_i^(d)) then
6  |  |  |  disregard this r_j;
7  |  |  |  break;
8  |  |  end
9  |  end
10 end
11 Get shortlist of eligible requests (n')
12 for r_j = 1 to n' do
13 |  efficiency(r_j) ← v_j/C_i^(d*);
14 end
15 Sort requests by decreasing efficiency per bottleneck;
16 for j = 1 to n' do
17 |  if (r_j fits in s_i) then consider it ;
18 end
19 if (r_j has non-integer variables) then disregard it ;
20 Return decision variables that maximize the resource utilization;
```

3.3.2. Complexity analysis of the proposed D-MCKP

The algorithm used to solve our D-dimension Multiple Choice Knapsack Problem (D-MCKP) problem is based on the well-known Quicksort algorithm, which employs divide and conquer strategy to do the sorting [42]. It is known that the time complexity of Quicksort of n items is $\mathcal{O}(n.log_2(n))$ in both best and average cases, and $\mathcal{O}(n^2)$ in the worst case [42]. As Quicksort has the best performance in the average case for most inputs, it is generally considered the "fastest" sorting algorithm among known sorting algorithms [42]. Thus, the sort instruction (line 1) has a time complexity of $\mathcal{O}(D.\log_2(D))$ as D is the number of constraints. The for-loop (lines 3–10) in the algorithm (2) consists of $(D.n)$ iterations in the worst case that is when no request violates the constraints and the for-loop is not prematurely ended. Thus, its complexity is $\mathcal{O}(D.n)$. The sorting process is the time dominant task in the second part of the algorithm (lines 12–24). The for-loops in the last part of the algorithm consist of (n') iterations. The second sort instruction of the algorithm (line 15) consists of a sort operation of (n') numbers that should be less complex than the first sort instruction knowing that $(n' < n)$ by design. All in all, provided that D and the

number of slices m are far less than n, the time complexity of the proposed algorithm is $\mathcal{O}(n.\log_2(n))$, where n is the number of requests.

3.4. Slice scheduler model and problem formulation

Once the Admission Controller has mapped the demands to network slices, the mapping is addressed to the Slice Scheduler to properly serve corresponding demands with minimal time duration. We denote by p_j the processing time of a transmission request j, out of n requests, such that its time-span is $c_{j,t}$. Note that, as we are considering micro-service based architectural deployment, the processing time is independent from the processing capabilities of the processing node as each micro-service is atomic and is similar to any of its replicas [43]. Our problem consists in finding a schedule minimizing the total time duration. We define a binary decision variable z_{jt} to indicate if a request j is scheduled in time window (τ). Our slice scheduler can be formalized, once again, as an optimization problem as follows.

$$\min_x \dot{s} = \sum_{j=1}^{n}\sum_{t=1}^{\tau} c_{jt} z_{jt} \tag{5a}$$

s.t.

$$\sum_{j=1}^{n}\sum_{\sigma=max\{0;t-p_j+1\}}^{\tau} z_{j\sigma} \leq N,\ t \in \{1,\dots,\tau\} \tag{5b}$$

$$\sum_{t=1}^{\tau} z_{jt} \leq 1, \qquad j \in \{1,\dots,n\} \tag{5c}$$

$$z_{jt} \in \{0,1\},\quad j \in \{1,\dots,n\},\quad t \in \{1,\dots,\tau\} \tag{5d}$$

Provided that the host implementing a slice has finite capacity and can handle a maximum of up to N requests concurrently, constraint (5b) stipulates that during time window (τ), up to N requests can be executed. Constraint (5c) ensures that each request has to be scheduled only once. Finally, constraint (5d) stipulates that each request should be either served in current time window τ or deferred to following time window.

Note that τ is chosen according to the lowest granularity reported in the delay budget of the requested quality of services profiles, reported in Table 1. In our case, we considered τ to be 100 ms. Such a value is reported in [44] as the delay budget for conversational voice, Non-mission critical user plane Push to Talk voice, Mission critical video user plane. It is worth noting that the formulated problem in (5) is NP-hard [40] and corresponds to a Knapsack problem, which is a particular case of a D-MCKP. The same Algorithm 2 proposed above to solve the D-MCKP problem can be thus used to solve our particular scheduling problem in a polynomial time. In particular, we consider that D in this case to be equal to one.

3.5. Resource manager model and problem formulation

As stated earlier, denied requests from the Admission Controller module are sent to the adaptive Resource Manager to train its resource management techniques and decrease the chances of service denial in the future.

Let us denote by l_d and l, the demanded load level and the actual system load, respectively. These loads are expressed in bit and imply certain infrastructure requirements in terms of vCPUs and memory as elaborated in [45]. As an initial setting, we can either be based on readings from a similar application running on another VM/container or by allocating the minimum of available vCPUs, that is usually a unit.

Accordingly, our goal here is to let the resource manager function at an optimum flow rate (f_{set}), expressed in bit/s, while maintaining a safety load margin (l_g) to account for the shortfall between the demanded and actual system loads. We define the flow rate as the number of requests passing through the resource manager point in a given time period usually expressed in a per-second basis. We can achieve this optimal behavior, by controlling the variation of the resource manager processing rate over time through elasticity. Such elasticity consists of scaling-in and/or scaling-out the operations capacities in terms of resources. Using a micro-service based architecture, and particularly Docker containers, allows us to seamlessly do this auto-scaling as we demonstrated in one of our previous works [4]. Note that leveraging the prediction achieved by the forecast aware slicer typically favorites less scaling commands. In addition, with Docker containers, such elasticity guarantees that the session of the application are not interrupted as reported by [46] and verified by ourselves in our previous work [4].

As it is not efficient to keep scaling-in/out on constant basis, two questions need to be thoughtfully analyzed. First, provided that we use a particular policy (Π), how to automatically determine an optimum flow rate (f_{set}), to maximize the performance of the resource manager? Second, how to determine such ideal particular policy? Thus, the rationale behind the design of our resource manager is to act as an adaptive flow control system by leveraging DRL. Indeed, using DRL allows to capture all the intricate details of the acquired knowledge and thus relieves from explicitly doing the feature engineering process as elaborated in the following.

3.5.1. System model

Our resource manager is supposed to adapt its available resources according to the received binary priority ρ_t from the Admission Controller. Two main actions are possible: (i) increase/decrease when $\rho_t = 1$ or (ii) maintain the current processing flow rate, when $\rho_t = 0$. Accordingly, the resource manager adapts its flow rate by considering the dynamics of the resource demands. For this end, we will consider the following three variables: (i) the flow variation over time (γ) representing the acceleration/deceleration, (ii) the flow rate per time-unit (f) and (iii) the current system load (l). Note that γ has a direct impact on the cost in a cloud computing deployment [47]. Ideally, γ should follow the demands flow (γ_d) change in terms of increase/decrease. To bound this flow variation, we define two parameters $\gamma_{\min}$ and $\gamma_{\max}$ representing the minimum and maximum permissible flow variation over time, respectively.

Prior formalizing our problem, we give an illustrative example, in the context of automotive industry. Our resource manager is similar to the cruise control system that is used to automatically set the vehicle's speed. In this case, our resource manager can decide to accelerate, maintain or decelerate according to the changes in the environmental factors, while keeping a safe distance from the demands.

In this context, inspired by vehicle dynamics, that maps the safe distance between two vehicles to the velocity and the time gap in between, we formulate the load difference between the demand and the system load as a linear function of the flow rate per time unit and the time gap. Accordingly, let us formalize the safety load margin (l_g) that determines the reference flow rate of our system (f_{ref}). We consider a simple model to let the flow rate be the distance between the demanded and current systems load over the time in between. We formulate l_g as a linear function of f, as follows.

$$l_g = t_{\text{gap}}.f + l_{g0} \tag{6}$$

where lg_0 is the initial load margin and t_{gap} is the time gap to transit from current to desired system state.

Let us denote by the relative load margin (l_{rel}) the difference between the demanded load (l_d) and the system load (l) as follows.

$$l_{\text{rel}} = l_d - l \tag{7}$$

The system maintains some safety margin to account for the demands variability as follows.

$$f = \begin{cases} \min(f_d, f_{set}) & \text{if } l_{rel} < l_g \\ f_{set} & \text{otherwise} \end{cases} \tag{8}$$

Three observations are collected from the environment: (i) the flow error (e_t), defined as the difference between the reference flow rate f_{ref}

and f for each time step t, (ii) its integral $\int e_t \, dt$ allowing to eliminate the steady state error, and (iii) the current flow (f) providing a boost effect.

The mechanism is similar to a Proportional–Integral–Derivative (PID) controller, where the integral term seeks to eliminate a residual error by adding some control effect onto the historic cumulative value of the error. Accordingly, when the error is eliminated, the integral term will cease to grow. It will result in diminishing the proportional effect when the error decreases, or compensating such error by the growing integral effect.

Finally, initial conditions of load and flow for the demands and the resource manager are denoted by: (l_{d0}, f_{d0}) and (l_0, f_0), respectively.

3.5.2. Problem formulation

Keeping in mind that our objective is to maximize the resource utilization, our adaptive resource manager problem can be formalized as follows.

$$\begin{aligned} &\max_{t} \quad \sum_{t\geq 0} \rho_t . l \\ &\text{s.t.} \quad \text{(6), (7), (8)} \end{aligned} \tag{9}$$

Problem (9) is NP-hard as it has non-linear and conditional constraints. Accordingly, we propose to decompose it, as depicted in Fig. 2, and solve it using DRL, as explained here-after.

First, recall that reinforcement learning consists of dynamically learning through a trial and error method to maximize an outcome. By following a policy Π, the system follows sample paths of state $s \in \mathbb{S}$, action $a \in \mathbb{A}$, and reward $r \in \mathbb{R}$ (e.g., s_0, a_0, r_0, s_1, a_1, r_1, etc.). To enable our system to learn the best set of actions autonomously, we also define a reward r_t, which is a function of the control input (u_t), the flow error (e_t) and a binary bias (b_t) as follows:

$$r_t = -(0.1e_t^2 + u_{t-1}^2) + b_t \tag{10}$$

Note that e_t is the flow error between desired flow rate and the flow rate from previous iteration, while u_{t-1} represents the control input from the previous time step. We consider b_t as a binary variable reflecting the minimized magnitude of the flow error, such that b_t =1 if $e_t^2 \leq \varepsilon$, or b_t = 0 otherwise, with ε being a small preset threshold. Note that as the error e_t and previous control input u_{t-1} are both preceded by a negation coefficient, therefore, the reward function, expressed in (10), is large whenever e_t and u_{t-1} have small magnitudes, especially that they are also squared. Accordingly, we can observe that the reward value is big when the magnitudes of the error and the previous control input are small and vice versa. In addition, it is worthy to note that the we decreased the impact of the error further by using a 10% coefficient in front of the squared error term. This multiplier is arbitrary chosen, less than one, to make sure that only a small proportion of the error affects the reward value.

We denote by p the transition probability from a state s to another, and by $\mathbb{E}$ the expectation. On first hand, we need to find the optimal policy Π^* that maximizes the total reward as follows.

$$\begin{aligned} \Pi^* = \arg\max_{\Pi} \mathbb{E}[\sum_{t\geq 0} \delta^t r_t | \Pi] \\ \text{s.t.} \quad s_0 \sim p(s_0) \\ a_t \sim \Pi(\cdot|s_t) \\ s_{t+1} \sim p(\cdot|s_t, a_t) \end{aligned} \tag{11}$$

where t is a time step and δ is a discount factor that is ≤ 1. From another side, we define the value function $V^{\Pi}(s)$ as the expected cumulative reward from following a policy Π, starting from state s, as follows.

$$V^{\Pi}(s) = \mathbb{E}[\sum_{t\geq 0} \delta^t r_t | s_0 = s, \Pi] \tag{12}$$

We define the Quality (Q) value $Q^{\Pi}(s,a)$ as the expected cumulative reward from taking that action a in state s and following the policy Π, as follows.

$$Q^{\Pi}(s,a) = \mathbb{E}[\sum_{t\geq 0} \delta^t r_t | s_0 = s, a_0 = a, \Pi] \tag{13}$$

On second hand, based on the defined actions in the system model, as well as the Q-value in (13) and our designed instant reward, we can write the DRL problem for a particular policy Π as a maximization of the expected Q-Value as follows.

$$Q^*(s,a) = \max_{\Pi} \mathbb{E}[\sum_{t\geq 0} \delta^t r_t | s_0 = s, a_0 = a, \Pi] \tag{14a}$$

s.t.

$$a \in \mathbb{A}, s \in \mathbb{S} \tag{14b}$$

Based on the Bellman equation [48], problem (14a) could be rewritten in a recursive form as follows.

$$Q^*(s,a) = \mathbb{E}_{s'}[r + \delta \max Q^*(s', a') | s, a] \tag{15}$$

where $Q^*(s', a')$ is the next time-step Q-value. To solve this problem to optimality, we can use value iteration as follows.

$$Q_{i+1}(s,a) = \mathbb{E}[r + \delta \max Q_i(s', a') | s, a] \tag{16}$$

We can say that Q_i, expressed in (16), will converge to Q^* when $i \to \infty$. However, this problem is not scalable. Thus, we propose to use a function approximator to estimate $Q(s,a)$ by exploiting deep neural networks. On first hand, we propose using a Deep Q-Learning (DQL) function approximator of the Q-Value, as shown in Section 3.5.3.

In order to avoid an overly complicated Q-function and in order to enable future-proofness for a possible continuous action space, we also consider learning an optimum policy Π^* for all possible actions [9]. Accordingly, on second hand, we also propose using deep neural networks to learn both of Q-value and the optimum policy, using Q-learning and Policy Gradients methods respectively, by training both of an actor (the policy) and a critic (the Q-value), as shown in Section 3.5.4. In this case, we use a Deep Deterministic Policy Gradient (DDPG) agent to learn both of Π and Q and finds an optimal policy (Π^*) that maximizes the long-term reward. We will investigate the performance both of these approaches next in our evaluation section.

Algorithm 3: Listing of Deep Q-Learning Algorithm

Data: Replay Buffer R, with capacity N
Result: Optimal Q ($\phi(s_t), a, \theta$)
Initialize replay buffer R to capacity N
Initialize Q with random weights
while *(simulation-condition)* **do**
  while *(episode-iteration)* **do**
    Initialize sequence $s_1 = \{x_1\}$ and old sequence $\phi_1 = \phi(s_1)$
    for t=1 ... T **do**
      According to probability ϵ
      case: exploration, select random action a_t
      case: exploitation, select $a_t = \max_a Q^*(\phi(s_t), a, \theta)$
      Execute a_t and observe r_t and state x_{t+1}
      Set $s_{t+1} = s_t, a_t, x_{t+1}$
      Preprocess $\phi_{t+1} = \phi(s_{t+1})$
      Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in R
      Get random minibatch of transitions $(\phi_i, a_i, r_i, \phi_{i+1})$ from R
      if *isterminal*(ϕ_{i+1}) **then**
        Set $y_i = r_i$
      else
        Set $y_i = r_i + \delta \max_{a'} Q(\phi_{i+1}, a', \theta)$
      end
      Perform a gradient descent step on $(y_i - Q(\phi_i, a_i, \theta))^2$
    end
  end
end

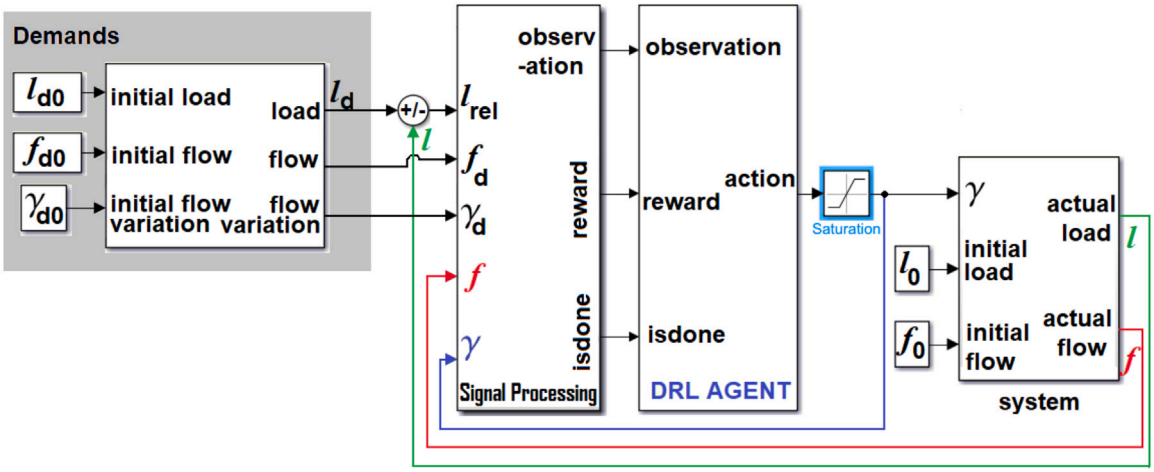


Fig. 2. Block diagram of the resource manager with adaptive flow control.

3.5.3. Algorithm for Deep Q-Learning with experience replay

The proposed algorithm for Deep Q-Learning with experience replay is listed in Algorithm 3 and works as follows. First, we initialize the replay buffer R and the Q-network with random weights. The simulation is terminated when the (*simulation-condition*) is no more true (Line 3), that is when $f < 0$ or $l_{\text{rel}} < 0$. During the training of the agent, we run at max N_{ep} training episodes, with each episode lasting for up to T time steps. We stop the training process, when the agent receives an episode reward $> V$ that is a value that we set to be relatively high. We denote by N_{ep} the maximum number of episodes and by V the stop training value. The condition (*episode-iteration*) is fulfilled as long as both of the conditions on N_{ep} and V are satisfied. Accordingly, the latter specifies when the agent stops upon reception of an episode reward greater than V. We initialize the state of start at the beginning of each episode. For each time step, with a small probability ϵ, select an action of exploration (try new action) or exploitation (select a greedy action from current policy), and observe the reward r_t and next state s_{t+1} (Lines 6–10). We store the transition in the replay buffer. Finally, we sample a random minibatch of transitions form R and perform a gradient descent step (Lines 14–20).

3.5.4. Algorithm for deep deterministic policy gradient

The proposed algorithm for training our DDPG agent is listed in Algorithm 4. First, we initialize all of the critic $Q(s, a|\phi)$, the actor network $\Pi(s|\theta)$ with random values of ϕ and θ respectively, to set a target network Q'. Then, we start our iterative training process (Line 7). We also initialize a replay buffer R, which we will populate during the iteration of each time step (Lines 5 and 15). Within each time step (t), we select an action according to the policy π with a certain random noise ($\mathcal{N}$) and we execute it as an exploration (Line 13). We store the transition in the replay buffer R, accordingly. Note that in the forward pass, we compute a loss function (Line 18) to update the critic by minimizing such loss over the chosen random transition i chosen from the replay buffer R. We also, update the policy Π using policy gradient (Line 21), through the soft-update of the policy and critic parameters (θ) and (ϕ), respectively (Lines 23–24).

Algorithm 4: Listing of DDPG Algorithm

Data: Policy Π parameters (θ), critic Q parameters (ϕ)
1 and discount factor (δ)
Result: Optimal policy Π^* with maximum reward
2 Initialize critic network $Q(s, a|\phi)$ with random ϕ
3 Initialize actor network $\Pi(s|\theta)$ with random θ
4 Update target network Q' and Π' with $\phi' \leftarrow \phi$, $\theta' \leftarrow \theta$
5 Initialize the replay buffer R
6 Let episode $\leftarrow 0$
7 **while** *(simulation-condition)* **do**
8 **while** *(episode-iteration)* **do**
9 episode $\leftarrow$ episode + 1
10 Initialize a random process $\mathcal{N}$ as noise
11 Collect initial observation state s_1
12 **for** t=1 ... T **do**
13 Execute action $a_t = \Pi(s_t|\theta) + \mathcal{N}$
14 Observe reward r_t and new state s_{t+1}
15 Store transition (s_t, a_t, r_t, s_{t+1}) in R
16 Get random $(i = 1..N)$ transitions from R
17 Set $y_i = r_i + \delta Q'(s_{i+1}, \Pi'(s_{i+1}|\theta')|\phi')$
18 Loss $L = \frac{1}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i|\phi))^2$
19 Update critic by minimizing L across all i
20 Update Π using policy gradient:
21 $\nabla_\theta J = \frac{1}{N}\sum_i \nabla_a Q(s, a|\phi)|_{s=s_i, a=\Pi(s_i)} \nabla_\theta \Pi(s|\theta)|_{s_i}$
22 Perform soft-update with $\mu \ll 1$ as follows:
23 $\phi' \leftarrow \phi + (1-\mu)\phi'$
24 $\theta' \leftarrow \theta + (1-\mu)\theta'$
25 **end**
26 **end**
27 **end**

4. Performance evaluation

In this section, we start by presenting the implementation of the resource orchestrator in our 5G prototype. We have used such prototype to generate realistic datasets to train the different modules, prior exporting them. Before presenting the overall network slice orchestrator performance used in our prototype, we evaluate the performance of each building block from a standalone viewpoint.

4.1. 5G experimental prototype overview

Fig. 3 depicts our 5G Non-Standalone (NSA) experimental prototype based on OAI implementing RAN and Core Network (CN). Northbound interface (NBI) for Configuration Management (Configuration Manager in orange) interacts with our proposed orchestrator. In its turn, the configuration manager commands the Software-Defined RAN Controller, namely, FlexRAN [5] through the NBI to manage underlying RAN nodes implemented using OAI.

We implement an Operation Support Subsystem/Business Support Subsystem using open-source projects, by setting up a "TICK Stack" (Telegraf, InfluxDB, Chronograph and Kapacitor) [49]. Telegraf is a

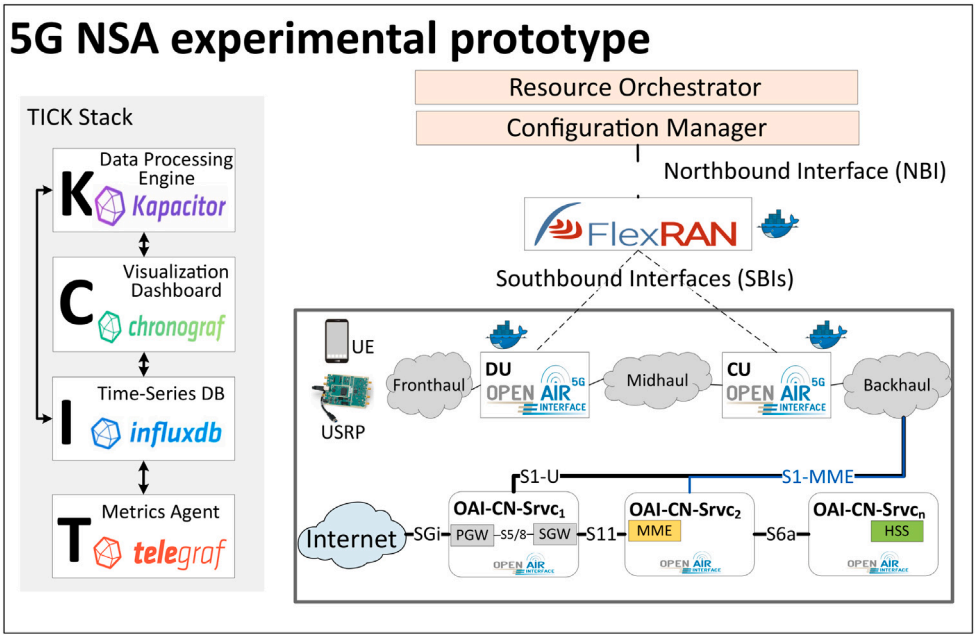


Fig. 3. Our 5G experimental prototype block diagram. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 2
Gatekeeper and decision maker simulation parameters.

Parameter	Explanation	Value
$C^{(1)}$	vCPUs constraint	2
$C^{(2)}$	RAM constraint (GB)	1
D	# of Knapsack constraints	2
slices	number of slices	3
UEs	number of UEs	3
ϵ_0	threshold for tree growing	0.05
S (Complex)	# of splits in the tree	100
S (Medium)	# of splits in the tree	20
S (Simple)	# of splits in the tree	4
q (Complex)	minimum leaf size	4
q (Medium)	minimum leaf size	12
q (Simple)	minimum leaf size	36
N	# of PRBs	50/100
Frame Type	duplexing type	FDD
EUTRA band	LTE band (F=2600 MHz)	7
K (Fine KNN)	# of nearest neighbor	1
K (Coarse KNN)	# of nearest neighbor	100
K (Other KNN)	# of nearest neighbor	10
τ	Time window for closed-loop (ms)	100

plugin-driven server agent for collecting and sending metrics and events from databases, systems, IoT sensors, and HTTP APIs. InfluxDB is a time-series database optimized for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics. Chronograph allows to rapidly build dashboards with real-time visualizations of the accumulated data. Finally, Kapacitor is a native data processing engine. It can process both stream and batch data from InfluxDB, acting in real-time via its programming language called TICK-script. The User Equipments (UEs) are conventional Commercial Off-The-Shelf (COTS) smartphones. We used OAI-5G for RAN to implement both of the Digital Unit (DU) and Central Unit (CU) of a gNB as Docker containers. OAI-CN, including Home Subscriber Server (HSS), Mobility Management Entity (MME) and Serving/Packet Gateways (S/P-GWs) are implemented as services. This implementation is inline with 5G deployment option 3 for NSA [34].

We used B210 USRP that is connected to a USB3 port on a laptop (Quad Core i7, 16 GB of RAM with Low Latency Kernel), implementing the DU and the CU. Accordingly, the distributed RAN is backhauled using a Gigabit Ethernet cabled network to a second laptop (Quad Core i7, 16 GB of RAM). Both laptops run Ubuntu 16.04. We argue that our prototype is 5G-based. On first hand, we are using a functional split in the ex-4G-baseband Unit (BBU) into a central unit (CU) and digital unit (DU). In addition, on the second hand, we are employing a software-defined RAN controller (FlexRAN) to manage the RAN slicing. Both of RAN functional split and SDN controller employment are 5G concepts. We also argue that for the sake of simplicity, a limited number of three UEs can be sufficient, as we used two UEs as users of an eMBB slice, and the third UE as a IoT Gateway behind which, the traffic of multiple IoT sensors/devices is aggregated to implement an mMTC slice. Note that multi-base station validation is not needed in our case since we are not evaluating Handover or cell-reselection based metrics or related Key Performance Indicators (KPIs). Instead, we are focusing on how to orchestrate network slices capacities sharing a BS within one mobile network, therefore a single USRP is sufficient.

4.2. Dataset generation and simulation environment

A dataset is generated using our 5G experimental prototype by running, for 24 h, a background script interfacing with FlexRAN to collect the configuration and performance management data of our COTS UEs. The collected statistics, encoded as a JSON file (JavaScript Object Notation), include provisioned slicing ratios, priority, QCI, power measurements, among other configuration and performance management metrics [5]. With background processes requiring Internet connection, UEs are moved from time to time in our laboratory space to change their radio conditions. ML models are implemented using MATLAB® [50] on Dual Intel Core i7, 2.4 GHz, 4-Cores 7th Gen. with 16 GB of RAM. Afterwards, RTs models are compiled as standalone applications for Linux so that they are implemented in our 5G experimental prototype. Several ML based models are evaluated in MATLAB to classify different NSs (QCI, resource type, loss rate, and priority level) corresponding to three conventional types of 5G slices: eMBB, URLLC and mMTC, as summarized in Table 1. Simulation parameters for the gatekeeper and the decision maker are listed in Table 2. Note that we used both of 10 MHz and 20 MHz channel bandwidth for our Universal Software Radio Peripheral (USRP) [51], which corresponds to 50 Physical Resource Blocks (PRBs) or 100 PRBs, respectively.

Table 3
Classification accuracy, speed, and training time.

Class	Model	FA (%)	FS (obs/s)	TT (s)
Trees [17]	Complex tree	94.7	6000	1.148
	Medium Tree	94.7	5800	0.782
	Simple Tree	95.3	9200	0.646
KNN [18]	Fine KNN	94.7	2200	1.618
	Medium KNN	94.7	2000	1.530
	Coarse KNN	64.7	3600	1.444
	Cosine KNN	84.7	2700	1.704
	Cubic KNN	94	3500	1.623
	Weighted KNN	95.3	4400	1.892
SVM [21]	Linear	96.7	1700	3.234
	Quadratic	96	1900	2.941
	Cubic	94.7	2800	3.879
	Fine Gaussian	92	2800	3.792
	Medium Gaussian	96.7	2900	3.691
	Coarse Gaussian	95.3	2500	3.545
Ensemble [20]	Boosted Trees	33.3	3100	2.186
	Bagged Trees	94	480	4.762
	Subspace Discrim.	95.3	410	6.859
	Subspace KNN	93.3	320	7.471
	RUSBoosted Tree	33.3	8200	6.495
Discriminant [19]	Linear	98	6200	1.294
	Quadratic	96.7	3700	1.922

Table 4
Forecasting methods benchmarking.

Class	Regression	RMSE	R^2	MAE
Linear [16]	Basic	28.77	0.16	25.61
	Interactions Linear	29.34	0.12	25.52
	Robust Linear	29.1	0.14	24
	Stepwise Linear	29.41	0.12	25.82
Trees [17]	Complex Tree	5	0.97	3.07
	Medium Tree	5.17	0.97	3.15
	Simple Tree	5.76	0.97	3.77
SVM [21]	Linear	31.48	0.01	21.74
	Quadratic	18.21	0.66	14.25
	Cubic	16.82	0.71	13.4
	Fine Gaussian	23.69	0.43	19.51
	Medium Gaussian	17.48	0.69	14.12
	Coarse Gaussian	29.06	0.14	20.32
Ensemble [20]	Boosted Trees	5.41	0.97	3.24
	Bagged Trees	12.52	0.84	10.13
GPR [22]	Square Exponential	16.87	0.71	13.62
	Matern 5/2	17.05	0.7	13.73
	Exponential	17.91	0.67	13.6
	Rational Quadratic	16.87	0.71	13.62

4.3. Simulation and implementation results

We start by evaluating the different ML-based approaches used for classification, forecasting and reinforcement learning for the resource manager. Then, we benchmark ML-based Regression Trees, which outperformed other evaluated techniques, among different strategies: Optimum, Static, and Random-slicing. We finally compare to the system performance in terms of number of PRBs, BSR, system utilization and network throughput.

4.3.1. Performance of ML-based classification models

Several ML-based classification techniques are benchmarked when proceeding with classification of 150 different requirements (QCI, Resource Type, Loss Rate, Priority Level) to conventional three slices as anticipated for 5G (eMBB, URLLC, mMTC). Table 3 reports the Forecasting Accuracy (FA), Forecasting Speed (PS) and Training Time (TT) of different classification models of the simulated 150 tenant requests with corresponding requirements, considering one requirement sheet per tenant. We can notice that the majority of model types provide an accuracy of more than 90% except for coarse KNN, cosine KNN, boosted trees and Random UnderSampling Boosted (RUSBoosted) trees. Linear Discriminant provides the highest accuracy.

Table 5
Resource manager parameters and values.

Parameter	Explanation	Value
$\gamma_{\max}$	flow variation upper bound	1
$\gamma_{\min}$	flow variation lower bound	−1
ε	threshold for reward bias	0.25
f_{set}	desired processing flow	30
f_{d0}	demands initial flow	25
f_0	system initial flow	20
L	critic DNN neurons	48
l_{d0}	demands initial load	80
l_{g0}	initial load margin	20
l_0	system initial load	10
N_{ep}	number of Episodes	5000
T	number of time steps	600
t_{gap}	initial time gap	1.4
V	stop training value	260
λ_c	learning rate for critic	10^{-3}
λ_a	learning rate for actor	10^{-4}

Table 4 reports different performance metrics including the Root Mean Square Error (RMSE), coefficient of determination (R^2) and Mean Absolute Error (MAE) [17] for the simulated forecasting models. We can see that trees (complex, medium, simple and even boosted trees), perform the best. However, this comes at the expense of an increased training time, as shown in Fig. 4. Indeed, from that figure, we can clearly see that, although RTs provide the lowest RMSE and highest forecasting speed, but their training time is not the least among the evaluated methods. However, this can be acceptable since the training need is not as frequent as the calls for forecasting. Accordingly, we chose to implement the simple trees for subsequent evaluations due to its outstanding performance and lower complexity compared to other trees.

4.3.2. ML-based forecasting using regression trees

Ground-truth metric:

To measure the ground-truth, we consider a simple two-slices' scenario (eMBB and mMTC) using the 100 PRBs setup (i.e., the USRP card is configured with 20 MHz channel bandwidth). Fig. 5(a) depicts the predicted values and Ground-truth. From this figure, we can see that the predicted values are close to empirical ground-truth. Indeed, the predicted values are spread around the straight line displaying a perfect match ($Y = X$) as shown in Fig. 5(b).

Slicing ratio metric:

To show the benefit of the ML-based RTs for the Forecast Aware Slicer, we compare in Fig. 6, the predicted values of slicing ratios with three alternative schemes: Optimum, Static and Random-slicing approaches. Note that Optimum values are calculated using a bottom-up estimation by aggregating demands of each slice and deducing the ratios. For the static approach, we assume a ratio of 50% for the eMBB slice and 50% for the mMTC slice. We can see that the random approach performs the worst. On the other hand, ML-based RTs outperform both static and random approaches with an average gap of 5% only to the optimal approach. This is due to the highest forecasting accuracy among the evaluated schemes.

4.3.3. Resource manager training and validation

Simulation parameters of the resource manager are reported in Table 5. Fig. 7 shows the reward for the DDPG (black) along with the reward of DQL (red). As our agent is based on actor–critic method, we plot on the same figure the critic's estimate of the discounted long-term reward at the start of each episode (Q0), based on the initial observation of the environment. As training progresses through episodes, we can see that Q0 approaches the true discounted long-term

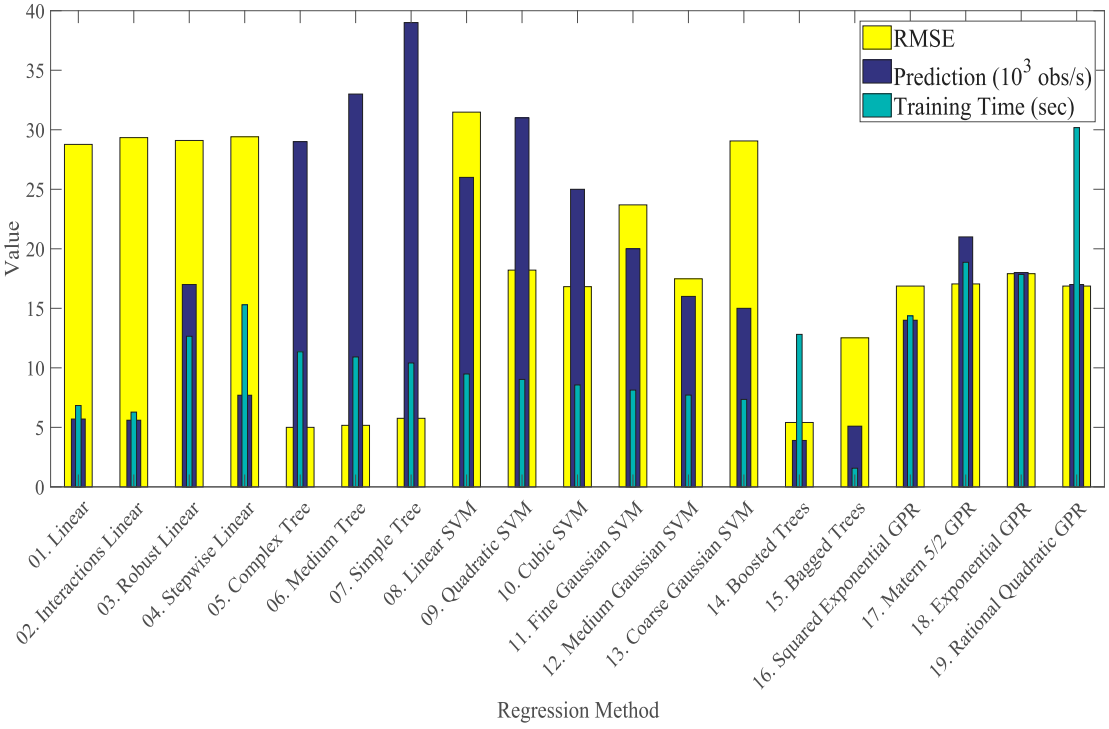


Fig. 4. Performance comparison of regression methods.

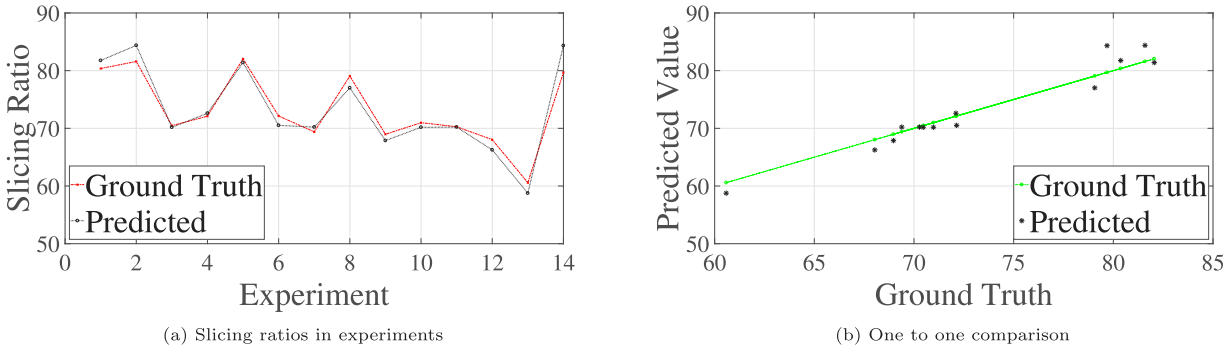


Fig. 5. Comparison of predicted vs. ground truth.

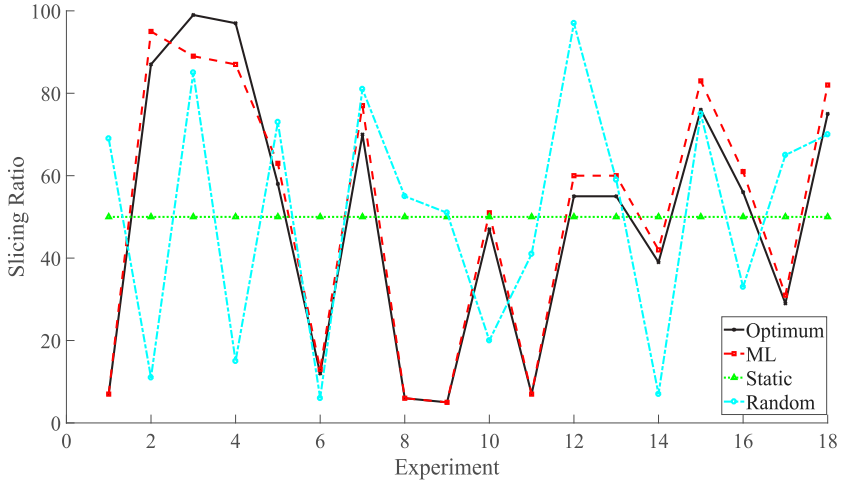


Fig. 6. Ratio of optimum, ML, static, random.

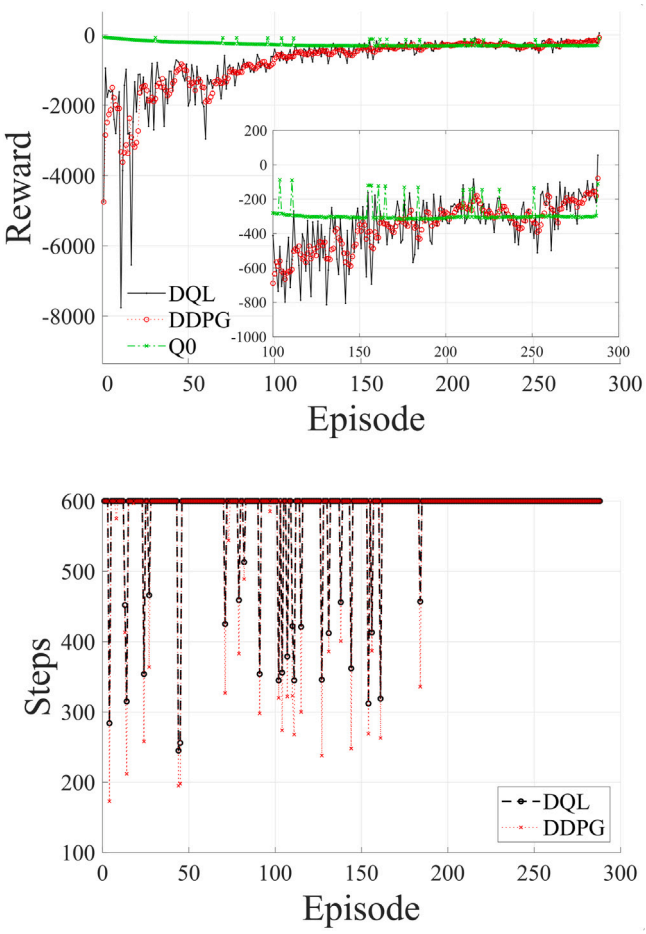


Fig. 7. Reward evolution in training stage. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

reward, which confirms the suitability of the design of the DDPG critic. At the bottom of Fig. 7, we display the number of steps of each episode. We can observe that the number of steps is at most 600, as stipulated in our algorithm. We note that the DDPG takes less steps than DQL.

4.3.4. Overall system performance:

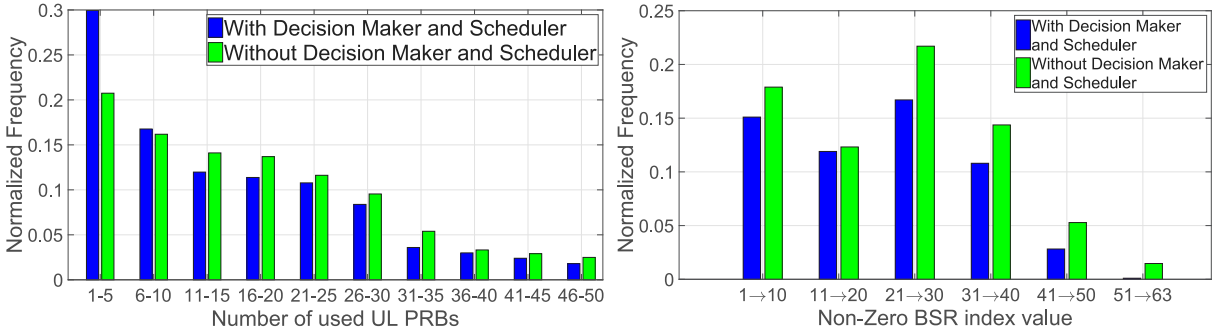
To further show the effectiveness of our system leveraging RTs based Forecast Aware Slicer, we compare it with a simple baseline where the modules are disabled. We create two slices having identifiers (ID) "0" and "1", where we map our devices, successively, to one of these slices and we trigger Youtube video upload. Slice 0 is not configured to do dynamic slicing ratio with forecasting capability, whereas Slice 1 exploits the RTs based forecasting. In order to assess the performance of this scenario, we consider four metrics, namely the UL PRBs, the BSR, the system utilization and the network throughput. It is worth noting that the BSR provides information on how much data is accumulated in the UEs' buffer and is waiting to be uploaded from the UE to the Evolved Node-B/new Generation Node-B (eNB/gNB), as sent in MAC Control Element (CE) [52]. According to the BSR, the eNB/gNB allocates the minimum amount of UL Grant that are resources for the Physical Uplink Shared Channel (PUSCH) when such resource is available. Using BSR, the network optimizes UL resources based on two folds. First, the network allocates UL resources (UL Grant) only when the UE has data to transmit. Second, BSR allows avoiding allocating too much UL resources (more than what is needed by the UEs) to avoid the waste of such resources. As FlexRAN reports the buffer size value in bytes, we map these values to BSR indexes based on a table in [52]. Note that BSR index values range between 0 and 63. The BSR index is a unit-less simplified metric such that "0" value refers to a "UE with no data to transmit" and when the index gets larger, it means that the UE has additional data in the buffer, waiting for transmission.

Number of UL PRBs:

In this experiment, we collected 23586 records of measurements including the total number of used UL PRBs using 10 MHz bandwidth (50 PRBs) USRP configuration. We plot in Fig. 8(a) the two normalized histograms of the used PRBs in both cases: "With" and "Without" the decision maker and the scheduler. Since the sample sizes are different, we have normalized the two histograms in order to have the sum of all bar heights equal to 1 and be able to plot them on the same figure for benchmarking. We use a uniform bin width of 5 PRBs for clarity. The X-axis refers to the number of used UL PRBs at each observation. The Y-Axis refers to the normalized distribution of used UL PRBs. Two main observations can be made here. First, the number of occurrences decreases with growing UL PRB usage in both approaches. We can explain this behavior as a result of the completion of the video upload process. Second, by enabling the decision maker and the scheduler, we significantly favor the use of low number of UL PRBs compared to the case where they are disabled. This is particularly shown for the two first bins (i.e., 1–5 and 6–10 used PRBs cases), where the frequency of used PRBs is higher when enabling these modules than that when they are disabled. This behavior is reversed for high UL PRB usage. From a resource management standpoint, it is a positive behavior as our system allocates sufficient resources to UEs without waste and tries to smooth the usage of the PRBs to avoid sudden spikes in demands. This finding will be confirmed when analyzing the BSR index in the following Fig. 8(b) and Table 6.

Number of BSR, system utilization and network throughput:

To illustrate the number of occurrences where the UEs have data to transmit but no UL resources are available, we plot in Fig. 8(b) the



(a) Normalized PRBs histograms (b) Normalized Non-Zero BSR index

Fig. 8. Decision maker and scheduler effect on PRBs and non-zero BSRs.

Table 6
Decision maker and scheduler effect on non-zero BSRs for UL.

UL ratio	Without decision maker and scheduler	With decision maker and scheduler
10%	29	27
20%	5	0
30%	3	0
50%	111	4
80%	107	42
90%	104	0
100%	101	0

normalized histograms of Non-zero BSR index values, with bin size of 10. Interestingly, we can observe that, by enabling the Decision Maker and the Scheduler modules, the bytes left in the buffers are decreased compared to the case when they are disabled, and that is for all BSR index values. In particular, there are no occurrences when activating these modules in bin 51, unlike the case when these modules are disabled. Moreover, we report in Table 6 the number of events where the BSR index is not zero. From this table, we can see that, when the Decision Maker and scheduler are not activated, there are more than 400 non-zero BSR indexes. These observations are seen even when the maximum possible slices ratios (90% and 100%, respectively) are considered. However, no reported non-zeros BSR index values are observed when these modules are enabled. This means that no bytes are left in the buffer in this case and the transmission of all the data is timely done. As for the observation of the Non-Zeros BSRs gradually increasing from null value that were observed at UL slicing ratios of 20% and 30%, we can interpret it as follows. When the UL ratio increased, additional PRBs were allowed to be used, and thus more bytes were uploaded, but the bottleneck in this case is geared towards the resource manager, and thus cause the increase seen in values 4 then 42. In this case, the resource manager has autonomously triggered its scale-out, which explains why such count decreased again for the 90 and 100% ratios. This confirms our previous results and shows the effectiveness of the orchestration process in processing data in a timely manner.

Finally, Fig. 9 depict the normalized throughput for the eMBB Slice and the system CPU utilization during two hours of simulation, respectively. We can observe, in Fig. 9(a) that, in overall, when the decision maker is enabled, the normalized throughput is higher compared to the case where it is deactivated. But, this comes at a cost of an increased system utilization as seen in Fig. 9(b). Interestingly, we note that the same normalized throughput was maintained although we queued a new video for upload. This is clearly seen in the system utilization, which increased starting from 50 min and onwards, reflecting such additional load.

5. Conclusion

In this paper, we have presented a novel framework based on Machine-Learning (ML) for network slices orchestration in 5G networks. Specifically, we have designed and implemented four building blocks, namely, Gatekeeper for classification, Decision Maker with Forecast Aware Slicer and Admission Controller sub-modules, Slice Scheduler, and adaptive deep reinforcement learning based Resource Manager. We evaluated the system performance using a 5G-ready experimental prototype based on OpenAirInterface and FlexRAN. From our experiments, we have observed that the Regression Trees (RTs) outperform other ML models in terms of classification and prediction accuracy. In particular, compared to linear based regressions, Root Mean Square Error (RMSE) is divided by six, prediction speed almost quadrupled but training time has slightly increased. We also found that the average gap between RTs and the optimal approach is only 5% and its trend is very close to the ground-truth slicing ratio. In addition, after implementing the RTs, the whole system, allowed to reduce the number of wasted Physical Resource Blocks and increase the network throughput compared to the case where system modules are disabled. But, this resulted in an acceptable increase of the system resources utilization. This effect is usually welcomed as it implies higher revenues in a multi-tenant environment, which is a highly desirable behavior for both public and private cloud deployments.

CRediT authorship contribution statement

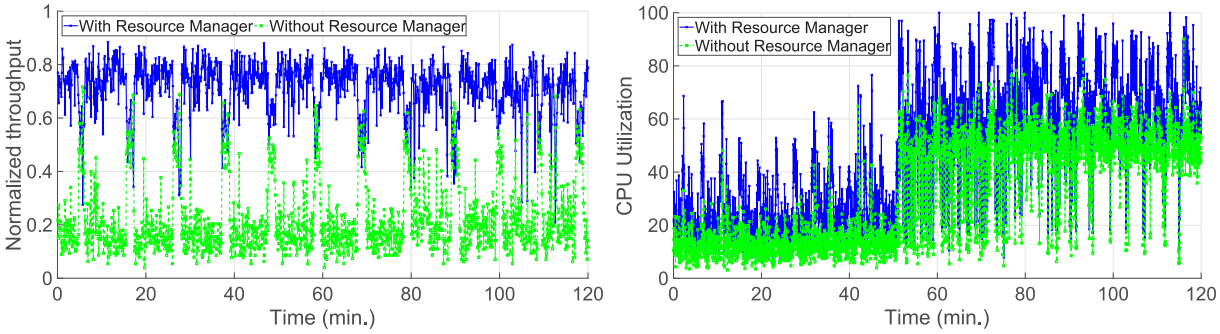
Nazih Salhab: Conceptualization, Methodology, Software development and integration, Data curation, Writing - original draft, Visualization, Investigation. **Rami Langar:** Supervision, Writing, Reviewing and editing. **Rana Rahim:** Supervision, Writing, Reviewing and editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was partially supported by the FUI SCORPION project, France (Grant no. 17/00464), the CNRS PRESS project, France (Grant no. 07771), “Azm & Saade” Association, and the Lebanese University.



(a) eMBB Slice throughput (b) CPU Utilization

Fig. 9. Resource manager effect on network throughput and system utilization.

References

[1] N. Salhab, R. Rahim, R. Langar, R. Boutaba, Machine learning based resource orchestration for 5G network slices, in: 2019 IEEE Global Communications Conference, GLOBECOM, 2019, pp. 1–6.

[2] 3GPP TR 22.864: Feasibility Study on New Services and Markets Technology Enablers - Network Operation; Stage 1 (R.15), 2016.

[3] K. Samdanis, S. Wright, A. Banchs, A. Capone, M. Ulema, K. Obana, 5G Network slicing – Part 2: Algorithms and practice, IEEE Commun. Mag. 55 (8) (2017) 110–111, http://dx.doi.org/10.1109/MCOM.2017.8004164.

[4] N. Salhab, R. Rahim, R. Langar, NFV orchestration platform for 5G over on-the-fly provisioned infrastructure, in: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS, 2019, pp. 971–972.

[5] OpenAirInterface (OAI), 2019, https://www.openairinterface.org/.

[6] DOCKER: Platform for high-velocity innovation, 2019, https://www.docker.com/.

[7] L. Le, B.P. Lin, L. Tung, D. Sinh, SDN/NFV, machine learning, and big data driven network slicing for 5G, in: 2018 IEEE 5G World Forum, 5GWF, 2018, pp. 20–25, http://dx.doi.org/10.1109/5GWF.2018.8516953.

[8] T. Taleb, I. Afolabi, K. Samdanis, F.Z. Yousaf, On multi-domain network slicing orchestration architecture and federated resource control, IEEE Netw. (2019) 1–11, http://dx.doi.org/10.1109/MNET.2018.1800267.

[9] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, L. Wang, Deep reinforcement learning for mobile 5G: Fundamentals, applications, and challenges, IEEE Veh. Technol. Mag. 14 (2019) http://dx.doi.org/10.1109/MVT.2019.2903655.

[10] L. Zanzi, F. Giust, V. Sciancalepore, M2EC: A multi-tenant resource orchestration in multi-access edge computing, in: IEEE Wireless Communications and Networking Conference, WCNC, 2018, pp. 1–6, http://dx.doi.org/10.1109/WCNC.2018.8377292.

[11] G. Zhu, J. Zan, Y. Yang, X. Qi, A supervised learning based QoS assurance architecture for 5G networks, IEEE Access (2019) http://dx.doi.org/10.1109/ACCESS.2019.2907142.

[12] Y. Yamada, R. Shinkuma, T. Sato, E. Oki, Feature-Selection based data prioritization in traffic prediction using machine learning, in: IEEE Global Communications Conference, GLOBECOM, 2018, pp. 1–6.

[13] Y. Xie, J. Hu, Y. Xiang, S. Yu, S. Tang, Y. Wang, Modeling oscillation behavior of network traffic, IEEE Trans. Parallel Distrib. Syst. 24 (9) (2013).

[14] F. Tseng, X. Wang, L. Chou, H. Chao, V.C.M. Leung, Dynamic resource prediction and allocation for cloud data center using the multiobjective genetic algorithm, IEEE Syst. J. 12 (2018).

[15] N. Salhab, R. Rahim, R. Langar, Optimization of virtualization cost, processing power and network load of 5G software-defined data centers, IEEE Trans. Netw. Serv. Manag. (2020) 1–12.

[16] D.J. MacKay, Bayesian interpolation, Neural Comput. 4 (3) (1992) 415–447.

[17] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Chapman and Hall by CRC, 1984, p. 368.

[18] S. Zhang, X. Li, M. Zong, X. Zhu, R. Wang, Efficient kNN classification with different numbers of nearest neighbors, IEEE Trans. Neural Netw. Learn. Syst. 29 (5) (2018) 1774–1785.

[19] R. Gribonval, From projection pursuit and CART to adaptive discriminant analysis? IEEE Trans. Neural Netw. 16 (3) (2005) 522–532.

[20] Y. Wang, S. Xia, Q. Tang, J. Wu, X. Zhu, A novel consistent random forest framework, IEEE Trans. Neural Netw. Learn. Syst. 29 (8) (2018) http://dx.doi.org/10.1109/TNNLS.2017.2729778.

[21] A.Y. Nikravesh, S.A. Ajila, C. Lung, W. Ding, Mobile network traffic prediction using MLP, MLPWD, and SVM, in: 2016 IEEE International Congress on Big Data, BigData Congress, 2016, pp. 402–409.

[22] Y. Xu, F. Yin, W. Xu, J. Lin, S. Cui, Wireless traffic prediction with scalable Gaussian process, IEEE J. Sel. Areas Commun. (2019).

[23] L. Zanzi, V. Sciancalepore, A. Garcia-Saavedra, X. Costa-Perez, OVNES: Demonstrating 5G network slicing overbooking on real deployments, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS, 2018, pp. 1–2, http://dx.doi.org/10.1109/INFCOMW.2018.8406867.

[24] J.X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, X. Costa-Perez, Overbooking network slices through yield-driven end-to-end orchestration, in: ACM CoNEXT '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 353–365, http://dx.doi.org/10.1145/3281411.3281435, https://doi.org/10.1145/3281411.3281435.

[25] M. Yan, G. Feng, J. Zhou, Y. Sun, Y. Liang, Intelligent resource scheduling for 5G radio access network slicing, IEEE Trans. Veh. Technol. 68 (8) (2019) 7691–7703, http://dx.doi.org/10.1109/TVT.2019.2922668.

[26] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, X. Costa-Perez, A machine learning approach to 5G infrastructure market optimization, IEEE Trans. Mob. Comput. (2019) 1, http://dx.doi.org/10.1109/TMC.2019.2896950.

[27] M. Harishankar, S. Pilaka, P. Sharma, N. Srinivasan, C. Joe-Wong, P. Tague, Procuring spontaneous session-level resource guarantees for real-time applications: An auction approach, IEEE J. Sel. Areas Commun. 37 (7) (2019) 1534–1548, http://dx.doi.org/10.1109/JSAC.2019.2916487.

[28] E.H. Bouzidi, A. Outtagarts, R. Langar, Deep reinforcement learning application for network latency management in software defined networks, in: 2019 IEEE Global Communications Conference, GLOBECOM, 2019, pp. 1–6.

[29] J. Pérez-Romero, O. Sallent, R. Ferrús, R. Agustí, On the configuration of radio resource management in sliced RAN, in: NOMS 2018 - IEEE/IFIP Network Operations and Management Symposium, 2018, pp. 1–6.

[30] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, A. Banchs, Mobile traffic forecasting for maximizing 5G network slicing resource utilization, in: IEEE Conference on Computer Communications, INFOCOM, 2017, pp. 1–9.

[31] Y. Yu, J. Wang, M. Song, J. Song, Network traffic prediction and result analysis based on seasonal ARIMA and correlation coefficient, in: 2010 International Conference on Intelligent System Design and Engineering Application, 2010, pp. 980–983.

[32] MIT, Google, The New Proving Ground for Competitive Advantage, Tech. Rep., MIT Technology Review, 2017.

[33] 3GPP TS 32.130: Network sharing: Concepts (rel. 14), 2016.

[34] 3GPP TR 28.801: Study on management and orchestration of network slicing for next generation network (Release 15), 2018.

[35] E. Brynjolfsson, T. Mitchell, What can machine learning do? Workforce implications, Science 358 (6370) (2017).

[36] 3GPP TS 23.203: Policy and charging architecture (Rel. 7), 2016.

[37] S.M. Wong, Y. Yao, Linear structure in information retrieval, in: Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1988, pp. 219–232.

[38] Google, 2019, https://cloud.google.com/compute/all-pricing.

[39] Amazon, 2019, https://aws.amazon.com/ec2/pricing/on-demand/.

[40] R.M. Nauss, The 0–1 knapsack problem with multiple choice constraints, European J. Oper. Res. 2 (2) (1978) 125–131.

[41] N. Salhab, R. Rahim, R. Langar, Throughput-Aware RRHs clustering in cloud radio access networks, in: 2018 Global Information Infrastructure and Networking Symposium, GIIS, 2018, pp. 1–5.

[42] N. Wirth, Algorithms and Data Structures, CUMINCAD, 1986.

[43] A.R. Sampaio, J. Rubin, I. Beschastnikh, N.S. Rosa, Improving microservice-based applications with runtime placement adaptation, J. Internet Serv. Appl. 10 (1) (2019) 1–30.

[44] 3GPP TS 23.501: System architecture for the 5G System (5GS), V16.6.0 (Rel. 16), 2020.
[45] D. Davis, A. Rosemblat, vCPU Sizing Consideration, White Paper, Dell Software, 2019, https://virtualizationreview.com.
[46] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, P. Merle, Autonomic vertical elasticity of docker containers with elasticdocker, in: 2017 IEEE 10th International Conference on Cloud Computing, CLOUD, IEEE, 2017, pp. 472–479.
[47] H. Xu, B. Li, Dynamic cloud pricing for revenue maximization, IEEE Trans. Cloud Comput. 1 (2) (2013) 158–171.
[48] R. Bellman, Dynamic Programming, Princeton University Press, USA, 2010.
[49] Time series database, 2019, https://www.influxdata.com.
[50] Matrix laboratory: MATLAB and Simulink, 2019, https://mathworks.com/.
[51] Universal software radio peripheral (USRP), 2019, http://www.ettus.com.
[52] 3GPP TS 36.321: Medium access control (MAC) protocol specification (Rel. 12), 2015.

Nazih Salhab received his Computer Science and Telecommunication Engineer degree and his Master II Research from the Lebanese University, Engineering Faculty I. After that, he received his double Ph.D. degrees from University Paris-Est in France and the Lebanese University in Lebanon. He is a senior advisor with more than 15 years of international experience in mobile networks operation management, project management and business analysis for major mobile network operators around the world. He also joined several international labs as visiting scientist, including EPFL-Switzerland and D.R. Cheriton, University of Waterloo, ON, Canada. His research interest include: Network Function Virtualization (NFV), Software-Defined Networking (SDN), Cloud Radio Access Network (C-RAN), Orchestration, Artificial Intelligence/Machine Learning (AI/ML) and network resource management.

Rami Langar is currently a Full Professor at University Gustave Eiffel (UGE), France. Before joining UGE, he was an Associate Professor at LIP6, University Pierre and Marie Curie (now Sorbonne University) between 2008 and 2016, and a Post-Doctoral Research Fellow at the School of Computer Science, University of Waterloo, Waterloo, ON, Canada between 2006 and 2008. He received the M.Sc. degree in network and computer science from UPMC in 2002; and the Ph.D degree in network and computer science from Telecom ParisTech, Paris, France, in 2006. Prof. Langar is involved in many European and National French research projects, such as MobileCloud (FP7), GOLDFISH (FP7), ANR ABCD, ANR 5G-INSIGHT, FUI PODIUM, FUI ELASTIC, FUI SCORPION. He was chair of IEEE ComSoc Technical Committee on Information Infrastructure and Networking (TCIIN) for the term Jan. 2018-Dec. 2019, and co-recipient of the IEEE/IFIP International Conference on Network and Service Management 2014 (IEEE/IFIP CNSM 2014) best paper award. His research interests include resource management in future wireless systems, Cloud-RAN, network slicing in 5G/5G+/6G, software-defined wireless networks, smart cities, and mobile Cloud offloading.

Rana Rahim received her Computer science and Telecommunication Engineer degree from the Lebanese University in 2002. She then obtained her Master degree (DEA) in 2003 from the USJ University (Lebanon) and the Lebanese University, and her Ph.D. degree in January 2008 from the University of Technology of Troyes (UTT) - France. She was a postdoctoral researcher at the UTT from October 2008 to October 2009. She obtained her HDR (Habilitation à Diriger des Recherches) in 2016. She is currently Associate professor at the Lebanese University. Her research interests include System management, Quality of Service, IoT Networks, Smart Grids, cloud radio access networks, slicing in 5G and software-defined networks.