Martin-Perez, J., Antevski, K., Garcia-Saavedra, A., Li, X. & Bernardos, C. J. (2021). DQN Dynamic Pricing and Revenue Driven Service Federation Strategy. *IEEE Transactions on Network and Service Management, 18*(4), 3987–4001.

DOI: 10.1109/tnsm.2021.3117589

DQN Dynamic Pricing and Revenue driven Service Federation Strategy

Jorge Martín-Pérez*, Kiril Antevski*, Andres Garcia-Saavedra†, Xi Li†, Carlos J. Bernardos*
*University Carlos III of Madrid, Spain,
†NEC Laboratories Europe GmbH, Germany

***Abstract*—This paper proposes a dynamic pricing and revenue-driven service federation strategy based on a Deep Q-Network (DQN) to instantly and automatically decide federation across different service provider domains, each introduces dynamic service prices offering to its customers and towards other domains. A dynamic pricing model is considered in this work based on the analysis of real pricing data collected from public cloud provider, and upon this a dynamic arrival process as a result of the price changes is proposed for formulating the service federation problem as a Markov Decision Problem (MDP). In this work, several reinforcement learning algorithms are developed to solve the problem, and the presented results show that the DQN method reached 90% of the optimal revenue and outperformed existing state-of-the-art strategies, and it can learn the federation pricing dynamics to make optimum federation decisions according to price changes.**

***Index Terms*—federation, pricing, reinforcement learning, optimization**

I. Introduction

The profound shift towards Network Function Virtualization and Softwarization has yielded a rapidly increasing number of virtualized network functions and applications, such as NATs [1], firewalls [2], and even radio access functions [3], composing network services deployed over a virtualized infrastructure. These services usually reside on the premises of service providers (e.g., cloud providers, mobile network operators) and their needs for flexible on-demand deployment (i.e., the service can be deployed anytime and anywhere), higher capacity and lower latency operation are ever-growing. In this paper, we take the role of a service provider at a mobile edge (with limited resource capacity) that aims at maximizing its long-term revenue.

On the one hand, in order to satisfy stringent service requirements, service providers need to over-dimension their infrastructure to face system dynamics with high reliability, which results in additional cost and waste of resources. On the other hand, service providers have limited coverage and footprint, and may not have enough resources in certain areas where the service is requested. In this way, when facing a deficit of resources to accommodate new service requests, they need to lease services or resources from another provider according to already-established terms and service level agreements, i.e., *service federation*.

To this end, procedures to automatize such service federation processes have already been developed [4], [5]. Moreover, different algorithms have been proposed to decide optimum resource federation and service composition across multi-domains (see details in §II). However, all the previous work considers static pricing models. To the best of our knowledge, this is the first work that addresses realistic pricing dynamics when making decisions to deploy services locally or rely on federated infrastructure.

Such decisions have important implications on the final price offered to potential customers. However, the price associated with federated resources has complex time dynamics [6]. Specifically, the end-user price depends on several variables, such as the availability of resources over time, the demands of service requests, and other business factors, *which are unknown to the requester*. Price fluctuations may lead to service rejections, even when there are resources available in the federated domain, due to negative financial revenue when infrastructure costs are overly high. To address this problem, in this paper we explore an *opportunistic* strategy: if we were able to learn patterns in price dynamics associated with federated resources, we then could exploit this knowledge opportunistically, that is, (i) use federated resources when prices are low relative to future prices, even if local resources are available, and (ii) use local resources when federated resources are expensive.

The idea of exploiting resources opportunistically is not new as it has been proposed before in the context of radio access and other systems [7]. However, most of these works assume some sort of knowledge concerning the stochastic distribution of the system or some unrealistic conditions such as independence across random data samples. These assumptions however fail to capture real dynamics of prices over time, which is essential to make correct federation decisions. Conversely, we advocate in this paper for model-free reinforcement learning solutions that break us free from making such unpractical assumptions. In this way, only driven by measured data, our solution is able to learn patterns inherent to real pricing dynamics, and take federation decisions that maximize the long term revenue.

In summary, the contributions of this paper are as follows:

- We analyze the price dynamics of a public cloud provider, and take it as reference for the service pricing of a federated domain. By considering the observed pricing fluctuations, we are able to obtain higher revenues in the federation process;
- We derive a dynamic arrival process, which is impacted by the fluctuations in the service prices;

- We characterize a federated multi-domain scenario as an online decision-making problem that aims at maximizing revenue;
- We design and implement two model-free mechanisms based on reinforcement learning: Q-table solutions, and a Deep Q-network (DQN) solution. Both require training phase to derive a policy for revenue maximization; and
- We perform a thorough data-driven performance evaluation of our solution, and compare our approach against state-of-the-art solutions.

Our results show that, compared to an oracle, our DQN approach achieves 90% of the optimal performance with zero knowledge *a-priori* about system dynamics.

The rest of the paper is organized as follows. First, §II presents the related work of existing architectures for federation, federation-related algorithms, and auction-based resource allocation solutions. Then, we introduce the business scenario we consider in §III. In §IV-A, we analyze the price dynamics of our system, and in §IV-B the arrival process of incoming service requests. Later, we formulate an optimization problem in §V to maximize the provider's revenue, and a Markov Decision Problem in §VI that motivates our DQN approach and other benchmark algorithms to solve the problem. Following, in §VII, we present the experimental evaluation of the algorithms. Finally, in §VIII we conclude the work and point towards future directions.

II. Related work

In [8], federation mechanisms are classified as (*i*) open federation, where the connectivity between administrative domains changes dynamically; or as (*ii*) pre-established federation, where connections are fixed using business contracts and service level agreements. Compared to our work, the manuscript in [8] provides insights into the interactions between the administrative domains; however, it lacks an economic analysis.

There are already several platforms that enable a federation. In the 5GEx project [4], the administrative domains use UNIFY [9], which allows them to expose and exchange information about available resources for federation. In the 5G-TRANSFORMER project [10]–[13], a service orchestrator module provides a pre-established service federation to peering administrative domains. A similar approach is adopted by the 5Growth project [5], [14], as the platform enables various federation approaches such as multi-level multi-domain orchestration or open federation using distributed ledger technologies [15]. These works [4], [5], [9]–[15] provide detailed technical and architectural workflows of how federation can be realized in various scenarios for different use cases but do not present algorithmic solutions to actually make decisions. In our view, our work is complementary and it can be adapted as a tool to generate profitable federation decisions in most of the described solutions and platforms.

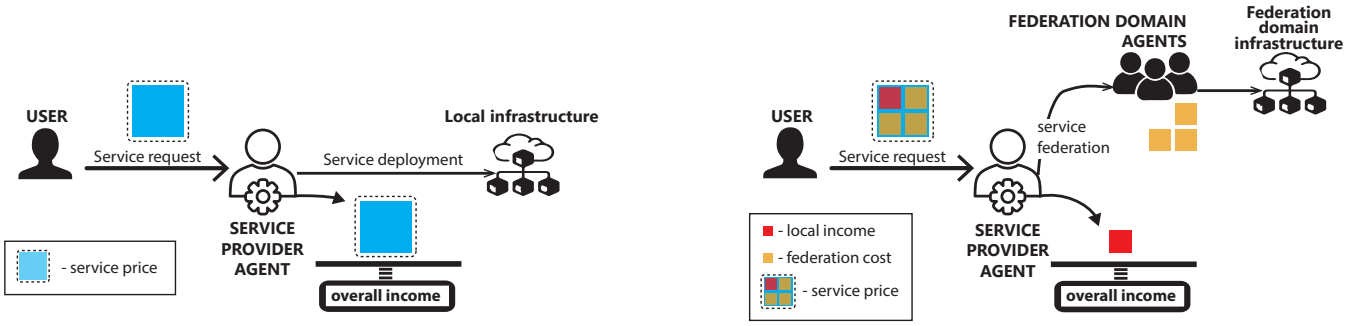
In 2012, the work in [16], a federation is identified as a challenging mechanism to tackle in virtual network embeddings. The work described in [17] proposes an adaptation of an Alternating Direction Method of Multipliers (ADMM) based algorithm, named AD^3 (Alternating Directions Dual Decomposition) [18]. The adapted algorithm solves the Virtual Network Embedding (VNE) problem in a decentralized fashion, and in a multi-domain scenario with each domain offering fixed pricing. [19] formulates the VNE problem in a scenario of non-cooperative domains that bid prices offered to deploy incoming Virtual Network Functions Forwarding Graphs (VNF-FG). The authors of [19] propose a framework based on Actor-Critic [20] agents for domains to decide the bidding prices, and for clients to maximize the number of deployed VNF-FGs. These works [16], [17], [19] study the VNE problem in-depth but focus on generating decisions that are technically efficient rather than economically profitable. Only [19] uses a Cost-based First Fit (CFF) heuristic algorithm to decide for low-price resources; however, compared to our work, they do not consider real price dynamics.

The same applies to [21], which proposes a heuristic to assess the VNE in multi-domain networks. The proposed solution, called consolidation-based, is a greedy approach that gives preference to the deployment of VNFs in master paths before service function chains (SFC) suffer from branching. The heuristic is enhanced with a feedback mechanism that prevents itself from deploying the SFC over links and servers that have recently failed. Similarly, as before, they assume static costs and revenues. Additionally, compared to [21], we focus on learning real scenarios and generating online decisions.

The work in [22] presents a distributed solution to compute a VNE in multi-domain networks. The algorithm is inspired by a large-scale graph processing [23] system that uses message-passing to decentralize the computation of the embedding of incoming VNF-FGs. The proposed algorithm iterates over what authors call "super steps", until each domain has locally deployed a part of the VNF-FG. Finally, a master node collects all feasible solutions proposed by each domain and selects the best. The solution ignores costs in the multi-domain infrastructure and the authors lay focus on scalability.

The authors in [24] focus on the problem of migrating service VNFs among domains that belong to a cooperative federation. Inspired by the flow/state migration problem [25], the paper proposes an algorithm that coordinates each domain orchestration, so as to assess the migration in a finite time, and satisfying non-functional requirements. In contrast to our work, they ignore the price associated with hosting VNFs across different domains.

The work in [26] offers a complete view of a multiple provider federation in 5G networks, and experimental validation of a heuristic approach on top of the described federation model. The work presents an abstraction of the resources that each provider offers to its neighbors within the federation. The abstraction, called the Bis-Bis node, represents a graph with an abstraction of the resources and connections offered to the peering providers. Unlike our work, the authors use a heuristic algorithm that is based on a greedy backtracking approach [27]. The algorithm is evaluated by means of scalability and running time in a multiple provider experimental setup. Unlike our



(a) Service local deployment

(b) Service federation

Fig. 1: Business model

solution, the authors in [26] assume fixed prices.

Regarding dynamic pricing scenarios, works such as [28] and [29] tackle resource allocation in mobile edge computing (MEC) and heterogeneous cloud scenarios in an auction-based manner. [28] proposes TCDA (Truthful Combinatorial Double Auction), a solution to determine both the pricing and resource allocation in a MEC scenario where mobile devices bid to obtain resources in the Edge. TCDA solves the associated optimization problem, and ensures that the pricing and resource allocation satisfies properties as social welfare maximization, locality constraints, and budget balance; among others. [29] solves the offloading of computing tasks in a heterogeneous cloud scenario where also users' mobile phones can execute offloaded tasks. Mobile phone users ask and offer resources to offload and accommodate computing tasks. As in [28], users bid to other mobile users to offload their tasks, and [29] proposes a greedy algorithm to solve both the allocation of such tasks, and derive payments of the auctioning phase. The proposed greedy reverse auctioning algorithm shows near optimal results by means of utility, execution time, and energy consumption. Moreover, it also satisfies economic properties as truthfulness, and individual rationality (as [28]); and it has been implemented as an Android app. Though both works [28], [29] provide good insights about allocating heterogeneous resources when the auction prices are controlled by the platform, their solutions focus on auction-based platforms, not on scenarios when prices are not under the platform control, which is the most common scenario. Moreover, [28], [29] do not use real-world pricing data in their analysis and have to rely on assumptions.

In summary, to the best of our knowledge, the existing literature has not yet considered real price dynamics when deciding the best domain to allocate services. To this end, we rely on data-driven model-free approaches based on Deep Q Networks (DQN), which learns patterns in the data without making any assumptions about the system to maximize long-term revenue.

III. Business Model

We start by analyzing the business model of interest for this work. Inspired by the market of cloud services, we consider a system where a *service provider* offers cloud resources or services at a *service price* rate $p^{(t)}$ that may vary over time depending on the operator's pricing model. In case a user is willing to pay such a price to deploy service σ, it makes the request, which arrives at the system at time $a(\sigma)$, and leaves the system at time $d(\sigma)$. Once a user request arrives at the service provider, the latter decides the best location to deploy the service σ: either on its own infrastructure, or on another domain within the federation it belongs to — see Fig. 1b. Hence, upon each service σ deployment requested by a user, the service provider can take an action $x(\sigma) := \{0, 1, 2\}$ indicating, respectively, whether the service is deployed locally, deployed in the federated domain, or rejected. Note that the user is not aware of the available resources in the service provider infrastructure (nor in the federation). Users are only aware of the price offered for their request $p^{(t)}(\sigma)$. See Table I for a summary of our notation.

Our goal is to maximize the long-term revenue of the service provider. The pricing model does have an impact on the arrival process of the service requests: intuitively, lower prices incentivize a higher user arrival rate. We will discuss the pricing model and the related arrival process model later in §IV-A and §IV-B, respectively. Importantly, however, once there is an agreement between customer and provider, the customer pays the agreed fee $p^{a(\sigma)}$ for every time slot t during which the service is active, i.e., for every $t : a(\sigma) \leq t \leq d(\sigma)$. In contrast,

TABLE I: Notation table

Symbol	Definition
σ	service
$p^{(t)}(\sigma)$	price rate [$/hour] to deploy a service σ at time t
$f^{(t)}(\sigma)$	federation cost [$/hour] at time t for service σ
$a(\sigma)$	arrival time of service σ
$d(\sigma)$	departure time of service σ
$x(\sigma)$	deployment action for service σ
$c(\sigma), m(\sigma), h(\sigma)$	CPU, memory, and disk asked by a service σ
C_l, M_l, H_l	local domain CPU, memory, and disk
C_f, M_f, H_f	federation pool of CPU, memory, and disk

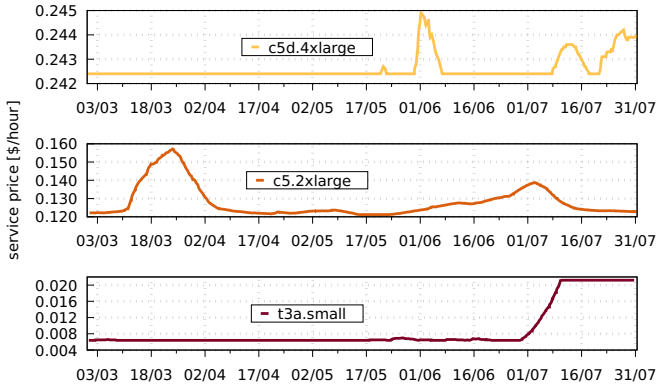


Fig. 2: Service prices of a cloud provider during 2020.

however, should the service σ be deployed in the federated domain, the service provider has to pay the federated domain agents a time-varying fee $f^{(t)}(\sigma), \forall t : a(\sigma) \leq t \leq d(\sigma)$ that depends on the dynamics in the federation. If the service is deployed neither locally nor in the federated domain, then the service will be rejected and, in this case, the customer does not pay the service fee, and thus there is no income for the service provider. This business model allows us to exploit opportunistically (uncertain) price fluctuations, which can provide substantial cost savings, yet provide certainty to the end-users, which is essential for vertical sectors.

As a result, every t we have two concurrent cash flows:

(Fig. 1a) The service provider uses local resources to grant the request, and therefore the agent's income is equal to $p^{a(\sigma)}(\sigma), \forall t : a(\sigma) \leq t \leq d(\sigma)$;

(Fig. 1b) The service provider uses federated resources, and therefore the provider gets $p^{a(\sigma)}(\sigma) - f^{(t)}(\sigma), \forall t : a(\sigma) \leq t \leq d(\sigma)$, where $f^{(t)}(\sigma)$ is the *federation cost*, which fluctuates over time.

In this way, we can denote the agent's income, which represents the instantaneous revenue of the service provider, at time t as follows:

$$r^{(t)}(X_t) := \sum_{\substack{\sigma:\ x(\sigma)=0 \\ a(\sigma)\leq t\leq d(\sigma)}} p^{a(\sigma)}(\sigma) + \sum_{\substack{\sigma:\ x(\sigma)=1 \\ a(\sigma)\leq t\leq d(\sigma)}} \left[p^{a(\sigma)}(\sigma) - f^{(t)}(\sigma)\right] \quad (1)$$

where $X_t := \{x(\sigma)\}_{\sigma:a(\sigma)\leq t}$.

In case the service provider runs out of local resources, its agent can federate the service $x(\sigma) = 1$ at a cost for the service provider $f^{(t)}(\sigma)$ — see the right term in (1). Hence, the availability of resources has an impact on the instantaneous revenue of the service provider. Note that eq. (1) also captures the amount of resources associated with each service σ request since resource-hungry services have a higher price. In the following, §IV details the three different services considered, namely, *t3a.small*, *c5.2xlarge*, and *c5d.4xlarge*; enumerated in ascending amount of resources (see Table II) and the corresponding price (see Fig. 2).

IV. System Dynamics

We face three sources of uncertainty in our system: (i) the pricing model used in federated domains $f^{(t)}$, which may be highly volatile; (ii) the cost associated with local deployments (which ultimately drives the service pricing $p^{(t)}$); and (iii) the process that characterizes the arrival of customers into our system, which is certainly associated with the fees we set $(p^{(t)})$ in a way that is unknown *a priori*. We discuss (i) and (ii) first in §IV-A, and then we discuss (iii) in §IV-B.

A. Pricing

Dynamic pricing mechanisms have become very popular in cloud computing services because they have the ability to maximize the cloud provider's revenue while minimizing the price of the offered service. The literature presents abundant research on the topic, being [30] a remarkable example. However, although the pricing problem has been well studied and, intuitively, prices shall follow the offer-demand trade-off, it is very hard to model the underlying pricing mechanisms applied in practice today. For instance, works such as [6], [31], [32] present spatio-temporal analyses of the pricing method applied by a large cloud provider but have failed to model it. Others, such as [33], [34], have applied predictive methods to this phenomenon, but the proposed solutions pitfall on either predicting price peaks, or the tendency of price over time.

Let us analyze, in the following, the price dynamics of service instances from a major cloud provider[1]. To this end, we collected price data of every service instance offered by the cloud provider between 29/02/2012 and 31/07/2020 for the "Paris, Europe" region. Fig. 2 depicts the price evolution of three service instances over a time window of five months, trivially chosen. Specifically, we have selected *t3a.small*, *c5.2xlarge*, and *c5d.4xlarge* from AWS EC2 Spot instances because they are the closest, in terms of resource requirements, to the services used in a multi-domain case study—a similar scenario as ours–from a well-known network operator in Spain [35].

The figure illustrates the fact that, though prices are reasonably stable over medium-long time periods, there occur a large number of short-time, yet sporadic, fluctuations that may play havoc with standard price prediction mechanisms. These fluctuations may be due to sudden changes in the arrival rate of the users but also to unknown external phenomena, such as energy costs or system failures. We argue in this paper that it is paramount to design a decision-making model that considers such random dynamic events to explore service federation (with unknown pricing model $f^{(t)}$) opportunistically such that we can maximize our agent's revenue.

Motivated by the above, we model the *service price* rates as

$$p^{(t)}(\sigma) = (1 + P)l^{(t)}(\sigma) \quad (2)$$

where $l^{(t)}(\sigma)$ is the local *deployment cost* (which depends on uncertain phenomena, as explained above), and P is the marginal profit over the local deployment cost as *a choice of the operator*.

[1]https://docs.aws.amazon.com/cli/latest/reference/ec2/describe-spot-price-history.html [Accessed 30/11/2020]

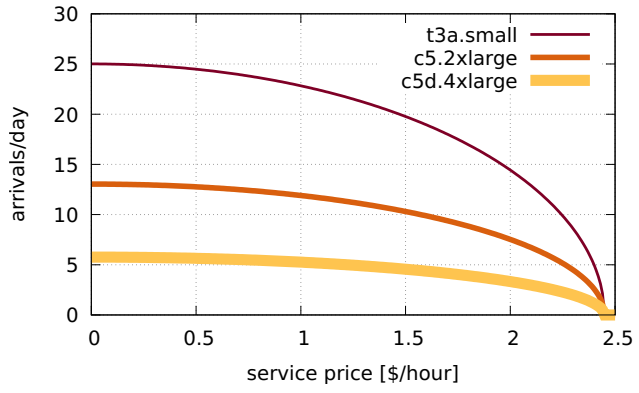


Fig. 3: Impact of the *service price* $p^{(t)}(\sigma)$ on the arrival rate of the different cloud provider service instances. This illustration uses $P = 0.2, a = 2, b = \frac{1}{2}, K = 2$, and k fit to the data provider of a main network operator in Spanish [35].

B. User arrivals

Let $\mathcal{A}$ denote the stochastic process modeling the arrival of users at the system. Intuitively, this shall be a non-homogeneous price-dependant process, i.e., a lower price $p^{(t)}$ incentivize higher arrival rate $\lambda^{(t)}$. In the context of cloud services, this phenomenon has been studied in, for instance, [36], where $\lambda^{(t)} = f(p^{(t)})$ and f satisfying the following assumption.

Assumption 1. *The arrival rate function is a non-negative* $f(\underline{p}^{(t)}) \geq 0$*, decreasing* $f'(\underline{p}^{(t)}) < 0$*, and concave function* $f''(\underline{p}^{(t)}) < 0$*; with no slope when the price is at its minimum* $f'(0) = 0$*, and taking zero values when the price is at its maximum* $f(1) = 0$*. Additionally, the arrival rate function should drastically drop to zero as the price reaches its maximum* $f'(\underline{p}^{(t)}) \xrightarrow[\underline{p}^{(t)} \to 1]{} -\infty$*, with* $\underline{p}^{(t)} \in [0, 1]$ *being the normalized price.*

Then, given Assumption 1,

$$f(\underline{p}^{(t)}) := k \left(1 - (\underline{p}^{(t)})^a\right)^b \tag{3}$$

where k, a, and b are parameters that depend on the system and hence have to be estimated, e.g., by a learning model. Note that the arrival rate function $f(\underline{p}^{(t)})$ is a positive, decreasing, concave function that is 0 when the dynamic price reaches its maximum.

To obtain the proper arrival rate based on $f(p^{(t)}(\sigma))$, the function must be properly defined, and the *service price* $p^{(t)}$ normalized accordingly. Thus, in this work we redefine f as follows:

$$f(p^{(t)}(\sigma)) := \begin{cases} k \left(1 - \left(\frac{p^{(t)}(\sigma)}{K \cdot M}\right)^a\right)^b, & p^{(t)}(\sigma) \leq K \cdot M \\ 0, & p^{(t)}(\sigma) > K \cdot M \end{cases} \tag{4}$$

where $M = \max_{\sigma,t}\{l^{(t)}(\sigma)\}$ is the maximum local deployment cost over time across all services σ (e.g., *t3a.small*), and K is a normalization constant to control the decay of the arrival rate. Note that (4) satisfies Assumption 1.

Using this model and our service pricing model in §IV-A, we plot in Fig. 3 the arrival rate (4) associated with the services that are used in this work. As stated in equation (2), the *service price* (hence the arrival rate) depends on the local *deployment cost* $l^{(t)}(\sigma)$ and margin P. Therefore, the mean arrival rate will decrease with high marginal benefit P or high local deployment costs (see Fig. 4).

V. Optimization Problem

Given the above, hereafter, we formulate an optimization problem with the goal of maximizing the revenue of a service provider in a multi-domain federation scenario. As introduced earlier, in our scenario, a local service provider may use a limited set of resources available locally, or resort to a federated resource provider (at a fee). Specifically, let $(C_l, M_l, H_l) \in \mathbb{N}^3$ denote, respectively, the total number of CPUs, memory, and disk resources available locally. Similarly, $(C_f, M_f, H_f) \in \mathbb{N}^3$ denote the respective resources at federated domain.

A service σ that arrives at time $a(\sigma)$ is characterized by a set of resource requirements $\big(c(\sigma), m(\sigma), h(\sigma)\big) \in \mathbb{N}^3$. Upon each request, the local agent makes a decision $x(\sigma)$, which shall guarantee that the resource capacity is not exhausted at any time. To this end, we have the following constraints:

$$C_l \geq \sum_{\substack{\sigma:\ x(\sigma)=0 \\ a(\sigma)\leq t \\ d(\sigma)>t}} c(\sigma), \quad M_l \geq \sum_{\substack{\sigma:\ x(\sigma)=0 \\ a(\sigma)\leq t \\ d(\sigma)>t}} m(\sigma), \quad H_l \geq \sum_{\substack{\sigma:\ x(\sigma)=0 \\ a(\sigma)\leq t \\ d(\sigma)>t}} h(\sigma), \quad \forall t \tag{5}$$

$$C_f \geq \sum_{\substack{\sigma:\ x(\sigma)=1 \\ a(\sigma)\leq t \\ d(\sigma)>t}} c(\sigma), \quad M_f \geq \sum_{\substack{\sigma:\ x(\sigma)=1 \\ a(\sigma)\leq t \\ d(\sigma)>t}} m(\sigma), \quad H_f \geq \sum_{\substack{\sigma:\ x(\sigma)=1 \\ a(\sigma)\leq t \\ d(\sigma)>t}} h(\sigma), \quad \forall t \tag{6}$$

Constraints (5) refer to the conservation of local domain resources, and constraints (6) to the conservation of the federated pool of resources.

Our objective is to choose the most appropriate action for every service request such that the obtained income (according to our business model in §III) over the long run is maximized. Hence, our optimization problem becomes:

Problem 1 (Federation deployment problem)**.**

$$\begin{aligned} \max_{X_T} \quad & \lim_{T\to\infty} \frac{1}{T} \sum_{t}^{T} r^{(t)}(X_t) \\ \textit{s.t.} \quad & (5), (6) \\ & x(\sigma) \in \{0, 1, 2\}, \quad \forall x(\sigma) \in X_T \end{aligned} \tag{7}$$

with $r^{(t)}(X_t)$ *being the instantaneous reward defined in eq.* (1).

The complexity of Problem 1 is analyzed in Lemma 1.

Lemma 1. *Problem 1 is NP-complete.*

Proof. Problem 1 can be cast into the knapsack problem [37], which is well-known to be NP-complete. To do the mapping, take a problem instance with $T = 1$, no federation resources $C_f = M_f = H_f = 0$, and assume that all services (*i*) do not leave the system, i.e., $d(\sigma) > T,\ \forall\sigma$; (*ii*) only ask for CPU

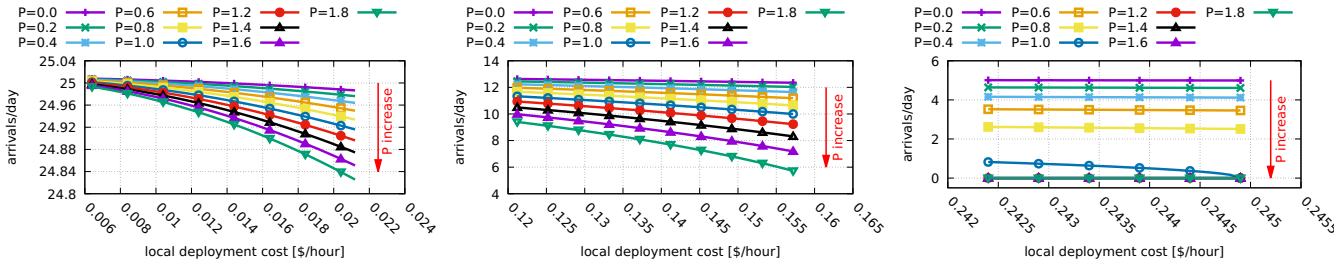


Fig. 4: Impact of local *deployment cost* $l^{(t)}$ and marginal benefit P in (a) t3a.small; (b) c5.2xlarge; and (c) c5d.4xlarge arrival rates. Graphs derived using Fig. 2 prices as $l^{(t)}$ local *deployment cost*.

resources, i.e., $c(\sigma) > 0 \ \wedge \ m(\sigma) = h(\sigma) = 0, \ \forall\sigma$; (*iii*) have a service price equal to the requested CPU $p^{a(\sigma)} = c(\sigma), \ \forall\sigma$; and (*iv*) arrive at $T = 1$, that is, $a(\sigma) = 1, \ \forall\sigma$. The resulting instance of our problem becomes

$$\max_{x(\sigma)\in X_T} \quad \sum_{\sigma:x(\sigma)=0} c(\sigma) \tag{8}$$

$$s.t. \quad C_l \geq \sum_{\sigma:x(\sigma)=0} c(\sigma) \tag{9}$$

which is the knapsack problem with $c(\sigma)$ being the object weights, and C_l the sack capacity. □

Apart from being NP-complete, our problem is not implementable as it has an overly large number of decision variables (more so as we increase the time horizon T towards infinity). It also requires knowledge of future arrivals of services requests and service/federation prices. For this reason, we resort to an online decision.

VI. Markov Decision Problem (MDP)

The optimization problem stated in §V is equivalent to an MDP [38], which we solve using three different algorithms introduced in this section. Any MDP is well defined given the tuple $\left(\mathcal{S}, \mathcal{A}, P_{X_T}, r^{(t)}\right)$, which describes, respectively, state set, action set, transition probabilities (given X_T), and reward function.

The state space $\mathcal{S}$ contains information related to (*i*) the local and *federation cost* of all services $\{\sigma\}$; (*ii*) the resources required for each arriving service σ; and (*iii*) the available resources at both the local domain and federation domains. Specifically, the state has information concerning $\left(\{\delta_{k_l}^{(t)}, \delta_{k_f}^{(t)}\}_{k\in\{C,M,H\}}, \{\sigma : a(\sigma) = t\}\right)$, where $\{\sigma : a(\sigma) = t\}$ contains the service requests at time t, and $\delta_{C_l}^{(t)}$ and $\delta_{C_f}^{(t)}$ are the normalized residual resource capacities, e.g.,

$$\delta_{C_l}^{(t)} = \frac{1}{C_l} \sum_{\substack{\sigma:x(\sigma)=0 \\ a(\sigma)\leq t<d(\sigma)}} c(\sigma), \qquad \delta_{C_f}^{(t)} = \frac{1}{C_f} \sum_{\substack{\sigma:x(\sigma)=1 \\ a(\sigma)\leq t<d(\sigma)}} c(\sigma) \tag{10}$$

for the case of CPU resources. Note that we redefine the state space for each algorithm presented next, for notation convenience.

Conversely, the action space is $\mathcal{A} = \{0, 1, 2\}$ corresponding to the "accept at local domain", "accept at federated domain", and "reject" actions upon incoming service requests. That is, the action variable $x(\sigma)$ of the optimization problem in §V belongs to the action space $\mathcal{A}$ of the MDP.

The transition probabilities are given by the function $P_{x(\sigma_t)}(s^{(t+1)}|s^{(t)}) \in [0, 1]$, that is, how likely it is to end up in state $s^{(t+1)}$ after taking action $x(\sigma_t)$ in state $s^{(t)}$; with σ_t being the service arriving at time t. The transition probabilities function $P_{x(\sigma_t)}$ is known given (*i*) the arrivals of new services; (*ii*) if previous services were deployed locally, federated, or rejected; (*iii*) the lifetime $d(\sigma) - a(\sigma)$ of each running service σ; and (*iv*) their local and *federation cost.*

In the MDP, the rewards $r^{(t)}$ correspond to the instantaneous reward already defined in (1) for the optimization problem in §V. The goal of the MDP is to derive a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that in each state $s^{(t)}$ takes an action $x(\sigma_t)$ that maximizes the long-term reward. However, the agent does not know the state transition probabilities $P_{x(\sigma_t)}(s^{(t+1)}|s^{(t)})$, since the future *federation cost* $f^{(t)}(\sigma)$ is an unknown random variable set by the federated agents. Moreover, the instantaneous reward (1) is also unknown, as it also depends on the *federation cost* $f^{(t)}(\sigma)$ random variable.

The following three algorithms propose different policies π with the goal of maximizing the expected long-term reward

$$\mathbb{E}_{x(\sigma_t)\sim\pi}\left[\sum_t \gamma^t r^{(t)}(\pi)\right]$$

with $\gamma \in [0, 1]$ being the discount factor for future rewards. This long-term reward refers to the income of the operator in the business scenario (§III). Among the presented algorithms, §VI-A proposes a greedy policy, whilst §VI-B and §VI-C derive the agent policy using Q-learning, a form of model-free reinforcement learning. Contrary to traditional dynamic programming methods, Q-learning solves a MDP without knowing the transition probabilities $P_{x(\sigma_t)}$ and instantaneous rewards $r^{(t)}$ (see [39]). Hence, both §VI-B and §VI-C derive policies that solve our MDP by learning the transition probabilities and real-world pricing dynamics. Using reinforcement learning prevents us from making assumptions of the real-world pricing

Algorithm 1: Greedy algorithm

Data: environment, T, $\{\sigma_t\}_{t=0}^{T}$
Result: $\{x(\sigma_t)\}_{t=0}^{T}$
for $t \in [0, T]$ **do**
  if $\frac{c(\sigma_t)}{C_l \delta_{C_l}^{(t)}} \leq 1 \wedge \frac{m(\sigma_t)}{M_l \delta_{M_l}^{(t)}} \leq 1 \wedge \frac{h(\sigma_t)}{H_l \delta_{H_l}^{(t)}} \leq 1$ **then**
    $x(\sigma_t) = 0$;
  else
    if $\frac{c(\sigma_t)}{C_f \delta_{C_f}^{(t)}} \leq 1 \wedge \frac{m(\sigma_t)}{M_f \delta_{M_f}^{(t)}} \leq 1 \wedge \frac{h(\sigma_t)}{H_f \delta_{H_f}^{(t)}} \leq 1$ **then**
      $x(\sigma_t) = 1$;
    else
      $x(\sigma_t) = 2$;
    end
  end
  $r^{(t)}, s^{(t+1)} =$ environment.takeAction($x(\sigma_t)$);
end

traces used in this work (see Fig. 2), but rather learn about real pricing events and how to act upon them.

A. Greedy algorithm

In this section, we introduce a *greedy* approach, which renders a simple strategy suitable for comparison. This approach consists of a simple policy where each service request is locally deployed as long as there is the availability of *local* resources or *federated* resources. If there are no resources in the whole system, the service request is rejected.

In the context of our MDP (described in §VI), the state vector used by the greedy approach is:

$$s^{(t)} = \left(\delta_{C_l}^{(t)}, \delta_{M_l}^{(t)}, \delta_{H_l}^{(t)}, \delta_{C_f}^{(t)}, \delta_{M_f}^{(t)}, \delta_{H_f}^{(t)}, \frac{c(\sigma_t)}{C_l \delta_{C_l}^{(t)}}, \frac{m(\sigma_t)}{M_l \delta_{M_l}^{(t)}}, \frac{h(\sigma_t)}{H_l \delta_{H_l}^{(t)}} \right) \tag{11}$$

where the first six elements are the normalized amount of residual resources (available) (*cpu*, *memory*, *disk*) at time t, and the last three represent the normalized amount of requested resources. In this way, this greedy policy, presented in pseudocode fashion in Algorithm 1, is described as follows:

$$\begin{aligned}
&\pi(0|s^{(t)}) = 1, \quad \frac{c(\sigma_t)}{C_l \delta_{C_l}^{(t)}} \leq 1 \wedge \frac{m(\sigma_t)}{M_l \delta_{M_l}^{(t)}} \leq 1 \wedge \frac{h(\sigma_t)}{H_l \delta_{H_l}^{(t)}} \leq 1 \\
&\pi(1|s^{(t)}) = 1, \quad \frac{c(\sigma_t)}{C_l \delta_{C_l}^{(t)}} > 1 \wedge \frac{m(\sigma_t)}{M_l \delta_{M_l}^{(t)}} > 1 \wedge \frac{h(\sigma_t)}{H_l \delta_{H_l}^{(t)}} > 1 \\
&\qquad\qquad \wedge \frac{c(\sigma_t)}{C_f \delta_{C_f}^{(t)}} \leq 1 \wedge \frac{m(\sigma_t)}{M_f \delta_{M_f}^{(t)}} \leq 1 \wedge \frac{h(\sigma_t)}{H_f \delta_{H_f}^{(t)}} \leq 1 \\
&\pi(2|s^{(t)}) = 1, \quad \text{otherwise}
\end{aligned} \tag{12}$$

B. Q-table algorithm

In this section, we adapt the solution presented in [40], which is a Q-table-based reinforcement-learning solution to the MDP. Q-table is a simple reinforcement learning realization based on a lookup table to search over the $\mathcal{S} \times \mathcal{A}$ space. The rows of the table present the state space $\mathcal{S}$ of the MDP, and the columns present the set of actions that can be taken for each state, i.e., the set $\mathcal{A} = \{0, 1, 2\}$. Each state $s^{(t)}$ is represented by normalized values that represent the average residual resource availability for local and federated resources, the most demanding resource of the arriving service, the instantaneous reward was we to deploy the service locally, $r_{x_0}^{(t)}$, and its *federation cost*:

$$\begin{aligned}
s^{(t)} = \Bigg(&\frac{\delta_{C_l}^{(t)} + \delta_{M_l}^{(t)} + \delta_{H_l}^{(t)}}{3}, \frac{\delta_{C_f}^{(t)} + \delta_{M_f}^{(t)} + \delta_{H_f}^{(t)}}{3}, \\
&\max\left\{ \frac{c(\sigma_t)}{C_l \delta_{C_l}^{(t)}}, \frac{m(\sigma_t)}{M_l \delta_{M_l}^{(t)}}, \frac{h(\sigma_t)}{H_l \delta_{H_l}^{(t)}} \right\}, r_{x_0}^{(t)}, f^{(t)}(\sigma_t) \Bigg)
\end{aligned} \tag{13}$$

As mentioned earlier, the state transitions are not deterministic. As a result, the Q-table requires a training period where its cells are populated to converge towards the expected future rewards. That is, $Q(s^{(t)}, x(\sigma_t))$ will be equal to the expected cumulative reward if action $x(\sigma_t)$ is taken at state $s^{(t)}$. To converge towards the expected future rewards, the Q-table values are filled using the Q-learning recurrence approach during the training stage, which approximates the Bellman equation [38]:

$$\begin{aligned}
Q(s^{(t)}, x(\sigma_t)) = &(1 - \alpha) Q(s^{(t)}, x(\sigma_t)) + \\
&\alpha \left(r^{(t)} + \gamma \max_x Q(s^{(t+1)}, x) \right)
\end{aligned} \tag{14}$$

where $r^{(t)}$ is the instantaneous reward foreseen after taking action $x(\sigma_t)$ at state $s^{(t)}$. Note the instantaneous reward is aggregated with the discounted future reward $\gamma \max_x Q(s^{(t+1)}, x)$. The parameters α—the learning rate, and γ—the discounted factor, are fixed parameters.

In summary, at each time t, a new service request arrives at the operator agent, who takes action $x(\sigma_t)$, and then populates the Q-table for $Q(s^{(t)}, x(\sigma_t))$. However, it must be noted that the instantaneous reward $r^{(t)}$ is not always positive. For example, suppose the action $x(\sigma_t) = 0$ for local domain deployment at $s^{(t)}$ exceeds the local domain capacity. In that case, the local domain rejects the service deployment, and the instantaneous reward is negative. In other words, the operator agent receives a *penalty* for taking the wrong action at time t.

All the Q-table values are initialized to zero at the start of the training ($t = 0$). The training procedure consists of repetitive runs of a generated set of arrivals for a timing interval $[0, T]$. Each training repetition is a single *episode* and the training set of arrivals is consistent for E_p episodes. To train this model so as to *learn* a policy that maximizes long-term reward, we use two different policies in the training stage, namely (*i*) Q-table legacy, and (*ii*) Q-table exploration.

On the one hand, the Q-table legacy strategy chooses the action as:

$$x(\sigma_t) = \max_x \left\{ Q(s^{(t)}, x) + \frac{u}{1 + e} \right\} \tag{15}$$

where $e \in \{1, \ldots, E_p\}$ is the current training episode, and $u \sim \mathcal{U}\{0, 2\}$ is drawn from a discrete uniform distribution. The

Algorithm 2: Q-learning training algorithm

Data: environment, strategy, E_p, T, ε_{min}, ε_{max}, γ
Result: $\left\{Q_T[s^{(t)}, x(\sigma_t)]\right\}_{t=0}^{T}$

$Q_T \leftarrow \mathbf{0}$;
for $e \in E_p$ **do**
  for $t \in [0, T]$ **do**
    if *legacy strategy* **then**
      $u \sim \mathcal{U}\{0, 2\}$;
      $x(\sigma_t) = \max_x \left\{Q(s^{(t)}, x) + \frac{u}{1+e}\right\}$;
    end
    if *exploration strategy* **then**
      $\varepsilon(e) = \frac{e}{E_p}(\varepsilon_{max} - \varepsilon_{min}) + \varepsilon_{min}$;
      $\overline{u} \sim \mathcal{U}\{0, 1\}$;
      $x(\sigma_t) = \begin{cases} \max_x \left\{Q(s^{(t)}, x)\right\}, & \overline{u} \geq \varepsilon(e) \\ \mathcal{U}\{0, 2\}, & \overline{u} < \varepsilon(e) \end{cases}$;
    end
    $r^{(t)}, s^{(t+1)}$ = environment.takeAction($x(\sigma_t)$);
    $Q_T[s^{(t)}, x(\sigma_t)] \leftarrow (1-\alpha) Q_T[s^{(t)}, x(\sigma_t)] + \alpha \left(r^{(t)} + \gamma \max_a Q_T[s^{(t+1)}, a]\right)$;
  end
end

actions taken in the first episodes (e.g., $e = 1$) have a larger random component compared to the last training episodes (e.g., $e \to E_p$). This exploration strategy is used in [40].

On the other hand, the Q-table exploration strategy uses a standard ε-greedy policy [41]:

$$x(\sigma_t) = \begin{cases} \max_x \left\{Q(s^{(t)}, x)\right\}, & \overline{u} \geq \varepsilon(e) \\ u \sim \mathcal{U}\{0, 2\}, & \overline{u} < \varepsilon(e) \end{cases} \tag{16}$$

where e is the training episode, and $\overline{u} \sim \mathcal{U}(0, 1)$ is a random number drawn from a uniform distribution. Moreover, we define $\varepsilon(e) = \frac{e}{E_p}(\varepsilon_{max} - \varepsilon_{min}) + \varepsilon_{min}$, which turns out a linear interpolation between ε_{min} and ε_{max}. In this way $\varepsilon(e)$ (which defines random exploration) drops as episodes pass. Algorithm 2 describes both the training procedure of both the legacy and exploration strategies.

Once the training stage has finished, the followed policy is applied:

$$\pi(x(\sigma_t)|s^{(t)}) = \mathbf{1}_{x(\sigma_t)} \left[\arg\max_x Q(s^{(t)}, x)\right] \tag{17}$$

that is, the action with highest Q-value is selected.

C. Deep Q Network (DQN)

We now present the neural network (NN) approach that this paper uses to solve the MDP. Specifically, the used NN adapts the DQN solution described in [42] to the action $\mathcal{A}$ and state spaces $\mathcal{S}$ of our MDP, as the solution proposed in [42] was not designed to solve the studied service federation problem. As stated in §VI, the goal is to maximize the expected long-term reward $\mathbb{E}_{x(\sigma_t)\sim\pi}\left[\sum_t \gamma^t r^{(t)}(\pi)\right]$. If $Q(s^{(t)}, x(\sigma_t))$ denotes the action-value function, i.e., a function estimating how good action $x(\sigma_t) \in \mathcal{A}$ is, given state $s^{(t)} \in \mathcal{S}$. The authors of [42] presented a NN with weights $\vec{w}$ to approximate the action-value function $Q(s^{(t)}, x(\sigma_t), \vec{w})$. Such a NN is refered as the Q-network (see the NN in Fig. 5).

The output of the Q-network is a layer of 3 neurons that specify the action-value estimation for each action, namely the local deployment $x(\sigma_t) = 0$, federation $x(\sigma_t) = 1$, and rejection $x(\sigma_t) = 2$. The Q-network's input is the state representation $s^{(t)}$, which in our case correspond to a vector containing the normalized residual capacity of local and federated resources, the normalized amount of resources demanded by the arriving instance, and the instantaneous reward if it is locally deployed, as well as its *federation cost*:

$$s^{(t)} = \Bigg(\delta_{C_l}^{(t)}, \delta_{M_l}^{(t)}, \delta_{H_l}^{(t)}, \delta_{C_f}^{(t)}, \delta_{M_f}^{(t)}, \delta_{H_f}^{(t)}, \frac{c(\sigma_t)}{C_l \delta_{C_l}^{(t)}}, \frac{m(\sigma_t)}{M_l \delta_{M_l}^{(t)}}, \frac{h(\sigma_t)}{H_l \delta_{H_l}^{(t)}}, r_{x_0}^{(t)}, f^{(t)}(\sigma_t)\Bigg) \tag{18}$$

It is worth mentioning that the Q-network of [42] differs with ours in the number of neurons at the input and output layers, as both the state (input) and action (output) spaces differ. In particular, our Q-network has $|s^{(t)}|$ neurons[2] at both the input and hidden layer and three neurons in the output layer, with all the layers being densely connected. All neurons in the Q-network use a linear activation, except the hidden neurons, which use a $\tanh(\cdot)$ activation unit.

The Q-network training must update its weights $\vec{w}$ to achieve the best possible estimation of the action-value function. As in the Q-table solution presented in §VI-B, the training procedure is based on the Bellman equation [38] to converge to the optimal action-value function. That is $Q(s, x, \vec{w}_i) \to Q(s, x, \vec{w}^*)$ as $i \to \infty$ with $\vec{w}_i$ denoting the Q-network weights at iteration i in the training process, and $\vec{w}^*$ denoting the weights with which the Q-network estimates the optimal action-value function. Rather than directly using the Bellman equation recurrence, the Q-network updates its action-value estimate by changing the weights $\vec{w}_i$. In more detail, gradient descend on the loss function is used[3]:

$$L_i(\vec{w}) = \left[\left(r^{(i)} + \gamma \max_x Q(s^{(i+1)}, x, \vec{w}_i)\right) - Q(s^{(i)}, x(\sigma_i), \vec{w})\right]^2 \tag{19}$$

This corresponds to the squared difference between the discounted reward with the current weights $\vec{w}_i$, and the action-value estimate given the weights $\vec{w}$.

An initial approach could be to compute the loss function $L_i(\vec{w})$ for every iteration i in the training stage, e.g., perform an update as $\vec{w}_{i+1} = \delta\vec{w} + \vec{w}_i$ with $\delta\vec{w} \propto \nabla L_i(\vec{w})$. However, the states of consequent iterations are very likely to be correlated, as the arrival rate will not drastically change in the next time instant (see Fig. 2). To mitigate this phenomenon in the training

[2] Actually, the Q-network has $k|s^{(t)}|$ neurons at the input and hidden layer, with k being the number of transitions, as explained latter. However, in the performance evaluation we use $k = 1$.

[3] $s^{(i)}, x(\sigma_i)$ denote the state and actions taken at iteration i of the training phase.

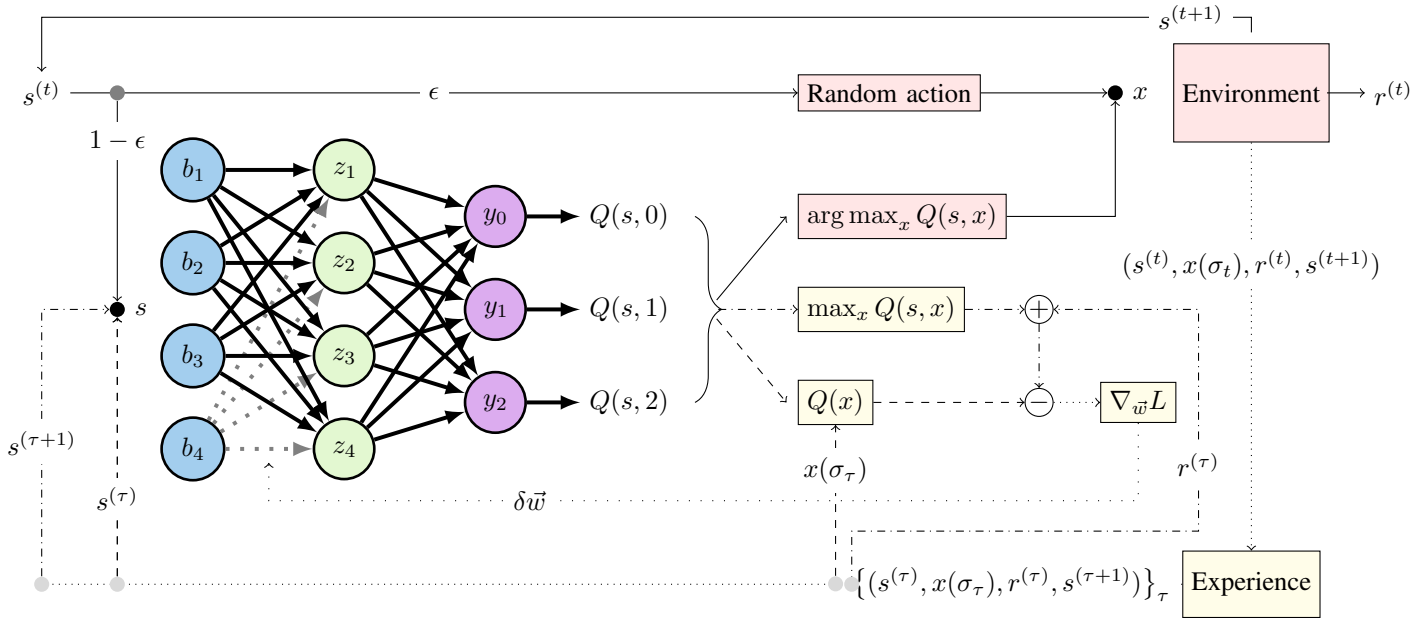


Fig. 5: Deep Q-network with one hidden layer, $k = 1$, and experience replay training. Non-continuous lines illustrate the training interactions, while continuous lines are used to show the execution interactions. In the illustration the state $s^{(t)}$ is a vector of only 4 components.

stage, the Q-network computes the loss gradient over past state-action-reward-state transitions. These transitions are stored into an Experience memory $\mathcal{D}$ with a capacity of up to M past transitions. As Fig. 5 illustrates, the Environment stores in the Experience the current transition, and the Experience will make room for it, if necessary, by removing the oldest transition. At iteration i of the training stage, the Q-network grabs a mini-batch of random past transitions $\left\{(s^{(\tau)}, x(\sigma_\tau), r^{(\tau)}, s^{(\tau+1)})\right\}_\tau$, and computes the loss function (19) gradient by using the current weights $\vec{w}_i$. The random transitions τ are uniformly selected among the M transitions present in the Experience memory $\mathcal{D}$. Note that using past experience does not only reduce the weight updates variability[4], but it also boosts data efficiency as each transition is used in many iterations of the training stage. The Q-network, together with the Experience and the aforementioned training procedure, is referred as DQN. Algorithm 3 details the training steps of the DQN using the RMSprop [43] as gradient descend method.

The state space is represented as in the Q-table algorithm of §VI-B, and the Q-network input corresponds to a concatenation of the last k transitions. During the training stage, the Q-network follows an ε-greedy policy that is later substituted by the following greedy policy during the test stage:

$$\pi(x(\sigma_t)|s^{(t)}) = \mathbf{1}_{x(\sigma_t)}\left[\arg\max_x Q(s^{(t)}, x, \vec{w}_{E_p})\right] \qquad (20)$$

with $\vec{w}_{E_p}$ being the weights after the last training episode E_p.

[4]The weights' updates variability comes as a consequence of the correlation between samples.

Algorithm 3: DQN training algorithm

Data: environment, E_p, T, M, ε, γ
Result: $\vec{w}_{T+1}$

1 **for** $e \in E_p$ **do**
2 environment.reset();
3 initialize $\phi \in \mathbb{R}^{11k}$;
4 **for** $i \in [0, T]$ **do**
5 $s^{(i)}$ = environment.getState();
6 $x(\sigma_i) = \begin{cases} u \sim \mathcal{U}\{0,2\} & ,\overline{u} < \epsilon \\ \arg\max_x Q(s^{(t)}, x, \vec{w}_i) & ,\overline{u} \geq \epsilon \end{cases}$;
7 $r^{(i)}, s^{(i+1)}$ =environment.takeAction($x(\sigma_t)$);
8 $\mathcal{D}$.addExperience($(s^{(i)}, x(\sigma_i), r^{(i)}, s^{(i+1)})$);
9 $\{(s^{(\tau)}, x(\sigma_\tau), r^{(\tau)}, s^{(\tau+1)})\}_\tau^M = \mathcal{D}$.sample($M$);
10 $G_0 = 0$;
11 **for** $\tau \in [0, M)$ **do**
12 $G_\tau = \beta G_{\tau-1} + (1-\beta)L_{\tau-1}^2(\vec{w})$;
13 $\vec{w}_{\tau+1} = \vec{w}_\tau - \frac{\alpha}{\sqrt{G_\tau}} L_{\tau-1}(\vec{w})$;
14 **end**
15 $\vec{w}_{i+1} = \vec{w}_{M_l}$;
16 **end**
17 **end**

VII. Performance Evaluation

This section presents the experimental evaluation of the algorithms introduced in §VI, Specifically, we compare the performance of: (i) the state-of-the-art Q-table solution of [40]; (ii) a variation of [40] that uses an exploration strategy and Algorithm 2 line 8), which we refer as QtEx (see §VI-B; and

TABLE II: Service requirements (from [35])

	t3a.small	c5.2xlarge	c5d.4xlarge
$f\left(\overline{p^{(t)}(\sigma)}\right)$	$5\frac{inst.}{day}$	$12.5\frac{inst.}{day}$	$25\frac{inst.}{day}$
CPUs	2	8	16
Memory	2 GB	16 GB	32 GB
Storage	100 GB	400 GB	800 GB
Life-time	$\frac{1}{192}L$	$\frac{1}{8}L$	$L = [96\ h, 240\ h]$
Marginal benefit	$P = 0.2$		

TABLE III: Data centers' resource capacities (from [35])

data center	CPU	memory	disk
local	80	2000 GB	160 GB
federation	480	12000 GB	960 GB

(iii) the DQN-based agent proposed in this paper. We also use an offline optimal oracle approach, which solves Problem 1 for a sufficiently large time horizon and assuming known future prices. Such an approach is unfeasible in practice because future prices are unknown; we only use it as a means to assess the optimality of our algorithms empirically.

The scenario we set up for evaluation is based on a mobile network operator (MNO) federation study case [35] from a large Spanish provider. In addition, we use the price evolution of a large cloud provider presented in §IV-A as a reference of federation and local service fees. Specifically, both the *federation cost* $f^{(t)}(\sigma)$, and local *deployment cost* $l^{(t)}(\sigma)$ correspond to the service prices of our reference cloud provider in the *eu-west-3a* region (see Fig 2). Hence, $\sigma \in \{$*t3a.small, c5.2xlarge, c5d.4xlarge*$\}$.

A. Experimental setup & environment

All experiments and results in the upcoming sections have been derived using a Dell Power Edge C6220 with two Intel Xeon CPU E5-2670 0 @ 2.60GHz, and 95GB of memory. We use Python 3 and TensorFlow 2.1.0 to implement all the schemes under evaluation; and AMPL/Gurobi 9.0.2 to implement our optimal (OPT) benchmark.

As in our reference case study [35], the scenario we assess is prone to encounter resource scarcity. Our goal is to emulate the business scenario explained in §III. The MNO (service provider) employs the above algorithms to generate deployment decisions for the incoming arrivals of service requests. Once a service is deployed, it books the requested resources for the requested lifetime period. Upon reaching the lifetime period, the deployed service leaves the system. To this end, we consider two data centers (local and federated) with different capacities. For our local domain, we select a medium-size data center from [35]; for our federated domain, we select [35]'s large data center. The details are depicted in Table III; the federation domains' data center has 6x the capacity of the local one, whereas the local's has the capacity to host 5x *c5d.4xlarge* services from Table II.

We generate service requests following a Poisson process with arrival rate $f(p^{(t)}(\sigma))$, as described in §IV-B, using the price evolution data from a major cloud provider as mentioned before. In more detail, k from eq. (4) is set to fit [35]'s data when the service price reaches its average value $\overline{p^{(t)}(\sigma)}$ (see Table II); and parameters a, b from eq. (4) are set to $a = 2, b = \frac{1}{2}$ as in [36]. Moreover, we set the marginal benefit P in eq. (2) to $P = 0.2$ unless otherwise stated (we will evaluate the sensitivity of our approach to P later). Finally, the departure times are also obtained from [35], i.e., the lifetime of each service is determined by a truncated normal distribution centered in the intervals $\frac{1}{192}L, \frac{1}{8}L, L$ specified in Table II.

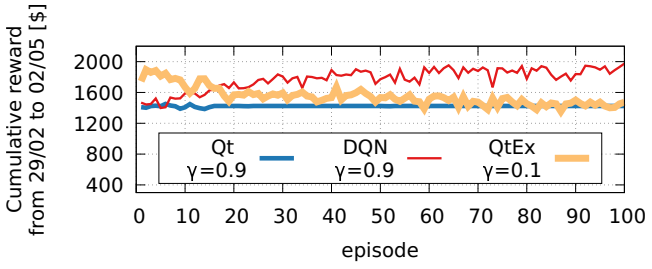


Fig. 6: Convergence of DQN, QtEx, and Qt after 100 training episodes.

B. Training

Among the three algorithms presented in §VI, both the Q-table and the DQN approaches required a preliminary training phase, which let us fine-tune the different hyper-parameters empirically to attain the best performance.

In detail, in the case of the Q-table algorithm, the learning rate and discount factor of the legacy strategy were selected as in [40], that is $\alpha = 0.95$ and $\gamma = 0.9$, respectively. For the Q-table explore strategy (QtEx), we used the same learning rate as in the Q-table legacy strategy, and the epsilon values decreased from $\varepsilon_{max} = 0.9$ in episode 1, down to $\varepsilon_{min} = 0.1$ in episode 100. We also tested a variety of discount factors γ between 0.1 and 0.9.

For our DQN approach, we used RMSprop to implement gradient descend, $\alpha = 0.001$, and a moving average parameter $\beta = 0.9$. As in the Q-table explore strategy, we tested out different discount factors (in the loss function (19)) between 0.1 and 0.9. The RMSprop gradient descend was computed using mini-batches of size $M = 30$ taken from the Experience (see line 11 of Algorithm 3). Fig. 6 shows that convergence was achieved within 20 episodes for the legacy Q-table solution, while QtEx and DQN converged within 60-70 episodes.

Both the Q-table and DQN algorithms were trained over $E_p = 100$ episodes. Each episode spanned over the service arrivals generated between 29/02/2020 and 02/05/2020. Fig. 7 shows how the discount factor impacts the cumulative reward of the DQN and the QtEx approaches during the training stage. None of the algorithms show a monotonic increasing/decreasing tendency with respect to the discount factor. Although, in general, DQN achieved a higher cumulative reward for higher values of γ, QtEx obtained the highest cumulative reward

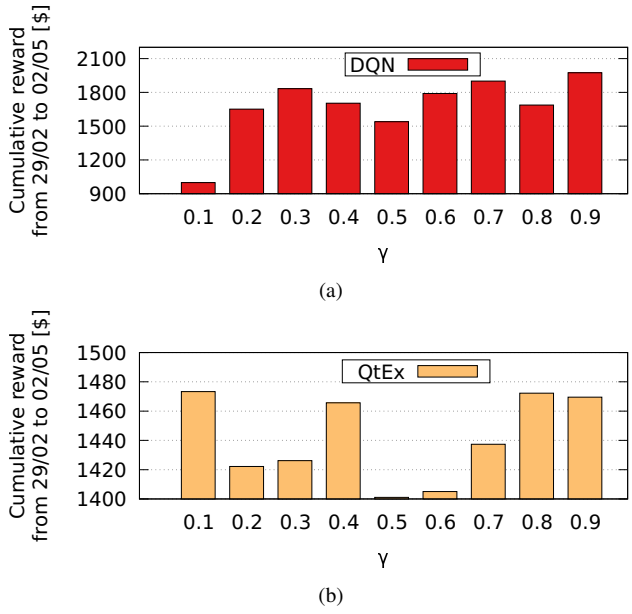


Fig. 7: Impact of discount γ parameter on DQN (a) and Q-table with the exploring strategy (b).

given $\gamma = 0.1$. This means that QtEx behaves better by relying on instantaneous rewards, while DQN did better when taking more into account the future rewards. The non-stationarity of the service arrivals and the transition probabilities make it harder to meet a monotonic change in the cumulative reward over γ. Arrivals are more likely to be bursty, which means there is no rule of thumb on how much to take into account the future rewards.

C. Performance

This section shows the performance of the algorithms described in §VI and trained as detailed in §VII-B. Every algorithm was tested over the service arrivals generated between 03/05/2020 and 31/07/2020.

1) Cumulative reward and evolution of decisions: Fig. 8a illustrates the cumulative reward of each solution over the aforementioned time-span, Fig. 8b depicts the price evolution of each service type in the federation domain, and Figs. 8c-g detail the decision-making evolution as a percentage of all service requests received at each time instant, for each of the algorithms under evaluation. Note that OPT refers to the optimal oracle, which we use as an ideal benchmark as introduced earlier. Greedy is a simple policy that only federates when local resources are exhausted and only rejects services when resources across all domains are exhausted.

Fig. 8a shows that OPT achieves a cumulative reward of \$3117.1, and DQN obtained \$2798.88, that is, the DQN algorithm was 90% as good as an optimal oracle that knows future service arrivals and prices a priori. On the other hand, Qt and QtEx only attained a cumulative reward of \$1793.82 and \$1892.7, respectively. However, both resulted in a higher benefit than the greedy baseline approach, which

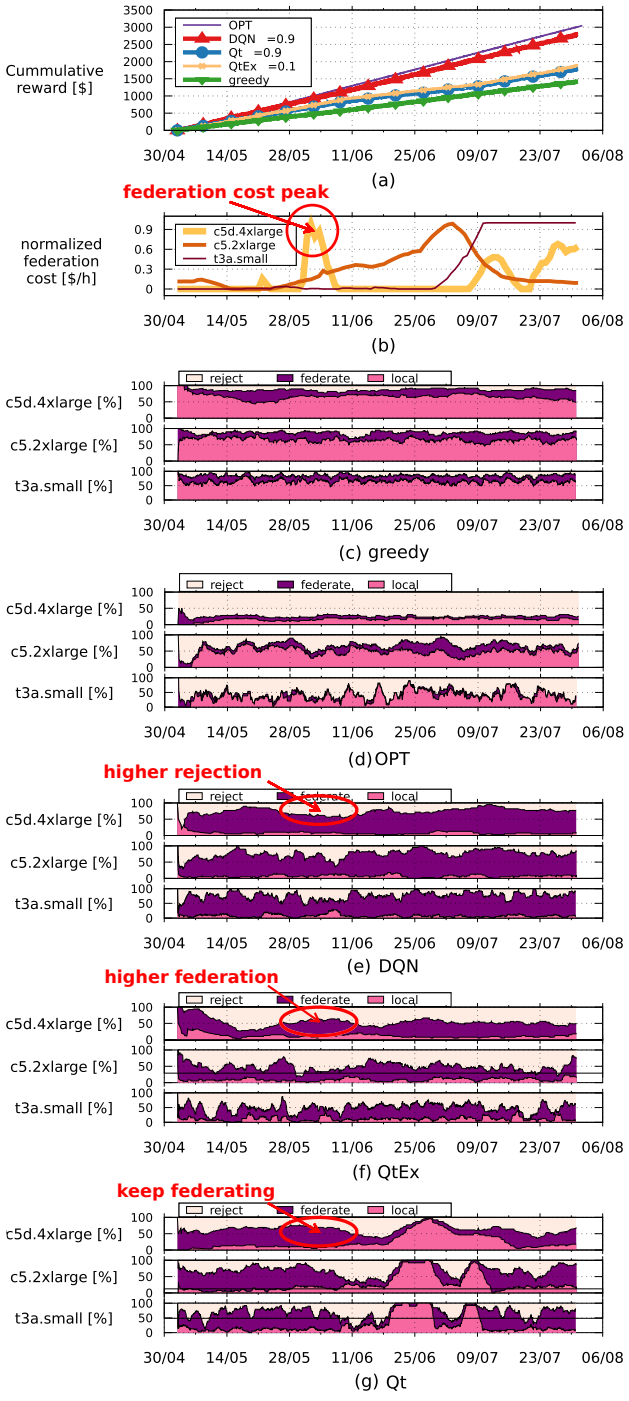


Fig. 8: (a) Cumulative reward of each solution during the May to July dataset; (b) the normalized *federation cost* $f^{(t)}$ over time; and the percentage of instances rejected, federated, or locally deployed by (c) greedy, (d) OPT, (e) DQN, (f) QtEx, and (g) Qt solution.

barely obtained a cumulative reward equal to $1418.22. The cumulative reward of each solution starts to diverge noticeably right after the *federation cost* for *c5d.4xlarge* reaches its peak, between 28/05 and 11/06 (see Fig. 8b). This service is the most demanding in terms of resources, and accordingly, the one with the highest associated *federation cost*.

Fig. 8c depicts the evolution of the decisions made by our greedy baseline. The plot shows that a tiny portion of service requests are rejected because the system is properly dimensioned. It is also evident that this policy is not affected by price fluctuations (Fig. 8d), which yields inferior performance in terms of revenue (Fig. 8b). An optimal oracle (Fig. 8c) is actually more conservative when granting service requests, with a much larger service rejection rate. This is especially true for the largest service (*c5d.4xlarge*). The reason is that this type of service incurs much larger net price fluctuations. Consequently, a better policy is to handle this type of service request conservatively.

Let us now explore the evolution of the decisions made by the DQN, QtEX, and Qt methods under assessment (Fig. 8e-g). The DQN algorithm follows a policy π_{DQN} that federates almost every service, except upon the prospect of price bumps when this policy rejects requests *even if resources are indeed available*. This allows preserving local resources available during periods when federated resources are expensive, and so this approach can also keep a low number of rejections. We can observe that after *c5d.4xlarge* federation cost reaches its peak, it increases the percentage of rejections, which helps to prevent future losses. This is key to deal with the uncertainty a practical approach such as this one has to deal with (in contrast to OPT) without being penalized in terms of reward. Conversely, QtEx and Qt increase or at least keep the same ratio of *c5d.4xlarge* services being deployed at the federated domain, as shown in Fig. 8f and Fig. 8g, respectively. Consequently, both Q-table-based algorithms achieve a lower cumulative reward because of the high fees these methods have to pay during sudden price increases.

2) Resource dynamics: We now plot in Fig. 9 the evolution of the resource consumption over time for our DQN approach between 03/05 and 31/07. This plot also shows, with different colors, whether the resource is consumed by a service deployed locally or federated, or whether the resource was requested by a service that was rejected. These results show that disk is the bottleneck resource in this case; note how this resource is around 100% of usage at almost any time, both for the local domain and the federated domain. Even though we have dimensioned our resources according to the case study of a large operator [35], these results suggest that both the CPU and memory resources should be shrunk by 50% without compromising reward, attaining substantial capital cost savings with no impact on revenue.

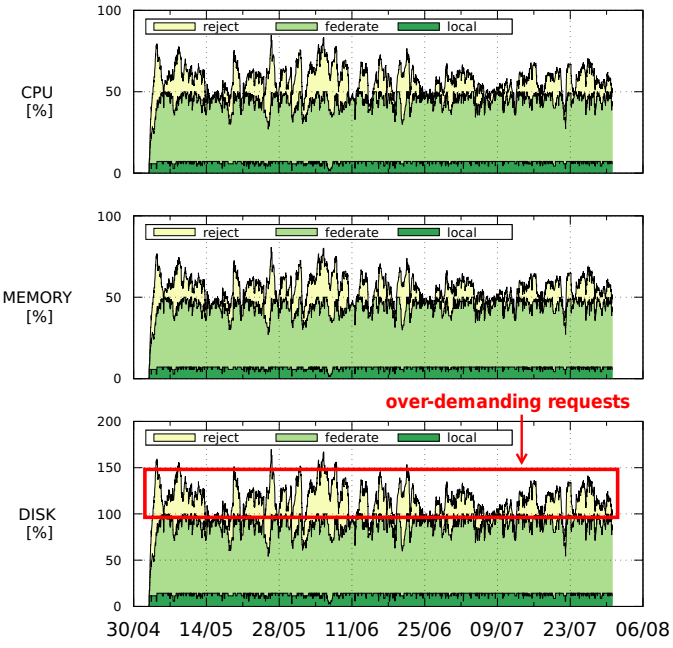


Fig. 9: Resources consumption by DQN $\gamma = 0.9$.

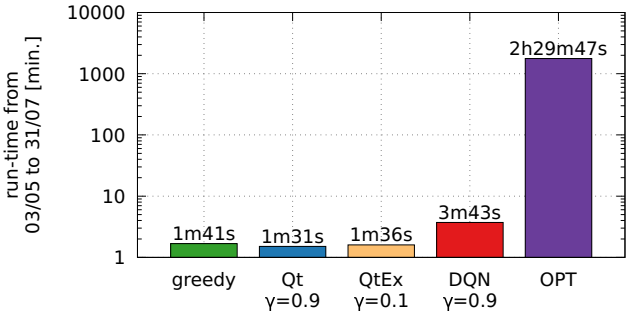


Fig. 10: Run-time comparison.

D. Computing complexity

We next present in Fig. 10 the actual time it takes by each algorithm to expedite decisions over three months (2918 requests). This provides empirical evidence of their computing complexity. Unsurprisingly, OPT takes an order of magnitude more time to run, as it has to explore a vast action space in a combinatorial problem (note the logarithmic scale of the y-axis) that is NP-complete (see Lemma 1). More importantly, both the DQN, QtEx, and Qt approaches perform similarly, reaching well within 4 minutes with a slight increase in computing time for our DQN.

To explain the slight increase, we have to look at the number of operations performed by each solution. Both Qt and QtEx have to access a Q-table entry to take an action, and that is a single hash operation since the Q-table is implemented using Python dictionaries. In contrast, the DQN has to perform a feed-forward pass in the Q-network to derive an action. Given that our DQN has two densely connected layers with input size $|s^{(t)}|$, and output size 3, the feed-forward performs two matrix multiplications, in particular $\tanh\left(s^{(t)} \cdot W_1\right) \cdot W_2$. Here, W_1, W_2 are the weight matrices of the hidden and output layer, respectively. Since W_1 has dimension $|s^{(t)}| \times |s^{(t)}|$, and W_2 has dimension $|s^{(t)}| \times 3$, the DQN feed-forward does

TABLE IV: Resource capacities for deployment A

data center	CPUs	memory	disk
local	[48,..., 80]	[1200,..., 2000] GB	[96,..., 160] GB
federation	480	12000 GB	960 GB

TABLE V: Resource capacities for deployment B

data center	CPUs	memory	disk
local	80	2000 GB	160 GB
federation	[0,..., 480]	[0,..., 12000] GB	[0,..., 960] GB

$\left|s^{(t)}\right|^2 + 3\left|s^{(t)}\right|$ multiplications, which explains the slightly worse run-time of DQN with respect to Qt and QtEx in Fig. 10.

Last, it is worth mentioning that the greedy algorithm spends a bit more time than the Qt and QtEx solutions, as it performs between 3 and 6 divisions to determine which action to take (see lines 2 and 5 of Algorithm 1).

E. Resource dimensioning

Our results above are based upon a specific deployment choice proposed by a large operator in Spain [35]. We now evaluate the impact of our algorithms on different deployments. Importantly, we do not re-train our learning approaches on the new setups; we simply use the same models trained on [35]'s scenario. This should give us an insight into the portability of our approach to generic environments.

To this end, we set up two different deployments: In Deployment A, we take [35]'s as a baseline and vary the local resources proportionally from 60% to 100%; In Deployment B, we take again [35]'s as a baseline and vary the federated domains from 0% to 600% (that is, 6x the resources available in our baseline's federated domain). The details of these two deployments can be found in Table IV and V, respectively. Figs. 11 and 12 depict the cumulative reward attained by each of our algorithms over the same 3-month time period used before, for Deployment A and B.

On the one hand, in Deployment A, DQN presents a substantial performance gain over the other approaches, no matter the size of the local domain. And apart from the QtEx, which outperforms the greedy approach by roughly $400, every other solution's gain has a similar and monotonic growth with respect to the available local resources. On the other hand, in Deployment B, the gain achieved by our DQN approach grows much faster with the dimension of the federated domain than all other approaches, e.g., reaching an 80% revenue increase with 6x resources than [35]'s. This is because the DQN federates more services than the greedy and Q-table solutions (see Fig. 8), and growth in the federation pool allows it to accommodate more services than the other solutions. Additionally, DQN takes advantage of the *federation cost* fluctuations to federate even more services when the *federation cost* is low, thus, increasing the benefits.

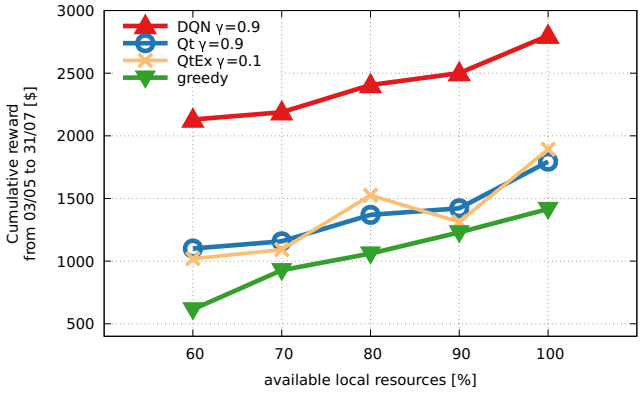


Fig. 11: Impact of available local resources in the cumulative reward achieved by each solution – deployment A.

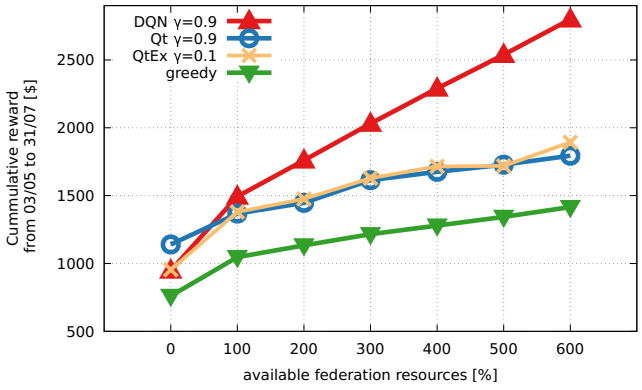


Fig. 12: Impact of the federation size in the cumulative reward achieved by each solution – deployment B.

F. Marginal Profit

Finally, we perform a sensitivity analysis on the marginal profit associated with the price of the services, P in eq. (2). Intuitively, P has an impact on reward via two phenomena: (i) higher P increases the net revenue associated with each incoming service, but (ii) higher P reduces incentives for customers to make requests into the system.

To this end, we deploy our baseline scenario (Table III), and test all our algorithms during the same 3-month period we used before for a variety of marginal profits between 0 (the *service price* equals that of the local *deployment cost*) and 3 (the *service price* is 4 times that of the local *deployment cost*). The results are depicted in Fig. 13.

It is worth noting that, when $P \geq 1$, the greedy approach outperforms both Q-table based solutions. This is due to the fact that we get a lower rate of service requests as we increase P (see Fig. 4), which allows the π_g policy to accommodate all incoming services locally. Conversely, the DQN algorithm reaches its maximum cumulative reward with $P = 0.8$, yielding a drastic drop in reward with $P > 0.8$. This is due to the fact that we have a drop in service requests of type *c5d.4xlarge* (see Fig. 4c for $P \geq 1$). With $P > 1$, DQN obtains a profit mostly from *c5.2xlarge* and *t3a.small* services, and reaches

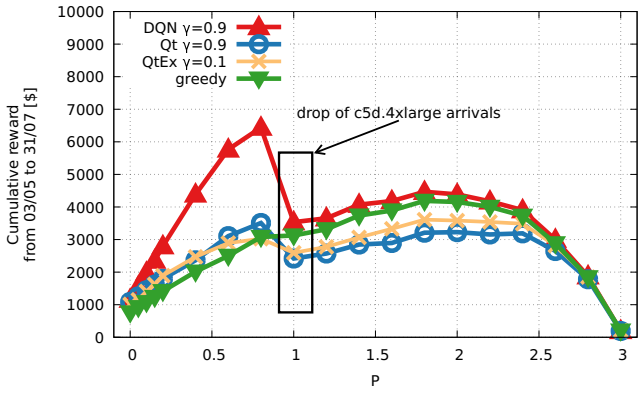


Fig. 13: Impact of the marginal benefit P in the commutative reward achieved by each solution.

its peak at $P = 1.8$. From that point on, all service requests decay prominently and so does the cumulative reward.

G. Practical considerations

In the following, we discuss some practical considerations on the realization of the DQN agent in a real cloud service provider:

1) A real service provider should revisit the revenue function $r^{(t)}$ proposed in §III, and substitute it according to its business model. Consequently, the environment used to train the DQN (see Fig. 5) should be refactored to yield the new instantaneous reward $r^{(t)}$ set by the real service provider.
2) The service provider should also revisit the assumption 1 on the services arrival rate $f\left(p^{(t)}\right)$, and how we defined it in this paper (4). In particular, the service provider could use real traces of service arrivals, and use such traces in the training stage of the DQN. If the Assumption 1 of [36] is right, the arrival of services should follow a concave decrease as the service provider increases the prices $p^{(t)}(\sigma)$.
3) The service provider should refer to the resources it has on the local infrastructure, i.e.: C_l, M_l, and H_l. It should also refer to the amount of resources in the federation domain, i.e.: C_f, M_f, and H_f. Note that in this paper we used the information reported by a large Spanish provider [35]; thus, we believe the aggregated resources used in our work are close to realistic scenarios.
4) The service provider should follow the training procedure explained in §VII-B using up-to-date data traces to capture real price dynamics (as we do with the prices of a major Europe/US cloud provider – see Fig. 2). Then, following our own workflow, an empirical evaluation of parameter γ should be performed, which may differ to ours if the provider has changed $r^{(t)}$.

Note that the timescale of our DQN approach is rather fast[5]. However, faster decisions may be obtained if required by using specialized machine learning hardware such as GPUs or TPUs, which are nowadays quite standard for this type of solutions.

[5]As shown in Fig. 10, DQN took 3 minutes and 43 seconds to take the deployment decision of 2918 service arrivals during 03/05/2020 and 31/07/2020.

VIII. Conclusions

Recent research activities have already proposed architectures to achieve service federation. However, they lack analyzing the service cost dynamic-effects upon making federation decisions. The contribution of this research is to fill such a gap by proposing a DQN agent, that solves the deployment decision problem of whether to federate services upon dynamic price changes in federation. Experiments were conducted taking as reference a real dataset of cloud provider prices, and service arrivals that decreased with the rise of prices. Simulations show that the proposed DQN agent reached 90% of the optimal revenue in a time-lapse of 3 months, and it learned the federation pricing dynamics, e.g., to stop federation upon incoming pricing peaks. Additionally, the DQN agent achieved fewer rejections than all the analyzed solutions, is able to achieve higher rewards upon the growth of available resources; and its computation time is in the same order of magnitude as other solutions.

The future work aims to explore scenarios consisting of (i) multiple federated domains; and (ii) service providers – each of them offering its own pricing. In that way, we can achieve maximum revenue by either (i) selecting the best federated domain; or (ii) offering more competitive prices to prevent users from selecting other service providers, thus, increasing the incoming service requests. Additionally, future work will also consider the fulfillment of location and latency constraints.

Acknowledgment

This work has been partially funded by the EC H2020 5GROWTH Project (grant no. 856709), and "Excelencia para el Profesorado Universitario" program of Madrid regional government.

References

[1] J. Nam, J. Seo, and S. Shin, "Probius: Automated approach for vnf and service chain analysis in software-defined nfv," in *Proceedings of the Symposium on SDN Research*, 2018, pp. 1–13.
[2] J. Gong, Y. Li, B. Anwer, A. Shaikh, and M. Yu, "Microscope: Queue-based performance diagnosis for network functions," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 390–403.
[3] G. Garcia-Aviles, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, P. Serrano, and A. Banchs, "Nuberu: Reliable ran virtualization in shared platforms," in *The 27th Annual International Conference on Mobile Computing and Networking (ACM MobiCom 2021)*, 2021.
[4] C. J. Bernardos, O. Dugeon, A. Galis, D. Morris, C. Simon, and R. Szabó, "5g exchange (5gex)-multi-domain orchestration for software defined infrastructures," *focus*, vol. 4, no. 5, p. 2, 2015.
[5] X. Li, A. Garcia-Saavedra, X. Costa-Perez, C. Bernardos, C. Guimarães, K. Antevski, J. Mangues-Bafalluy, J. Baranda, E. Zeydan, D. Corujo, P. Iovanna, G. Landi, J. Alonso, P. Paixão, H. Martins, M. Lorenzo, J. Ordonez-Lucena, and D. Lopez, "5growth: An end-to-end service platform for automated deployment and management of vertical services over 5g networks," *IEEE Communications Magazine*, 2021.

[6] G. George, R. Wolski, C. Krintz, and J. Brevik, "Analyzing aws spot instance pricing," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2019, pp. 222–228.

[7] A. Banchs, A. Garcia-Saavedra, P. Serrano, and J. Widmer, "A game-theoretic approach to distributed opportunistic scheduling," *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1553–1566, 2013.

[8] X. Li, J. Mangues-Bafalluy, I. Pascual, G. Landi, F. Moscatelli, K. Antevski, C. J. Bernardos, L. Valcarenghi, B. Martini, C. F. Chiasserini *et al.*, "Service orchestration and federation for verticals," in *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, 2018, pp. 260–265.

[9] "Unify-d2.2-final_architecture.pdf," https://www.eict.de/fileadmin/redakteure/Projekte/Unify/Deliverables/UNIFY-D2.2-Final_Architecture.pdf, (Accessed on 12/10/2020).

[10] J. Baranda, J. Mangues-Bafalluy, L. Vettori, R. Martínez, G. Landi, and K. Antevski, "Composing services in 5g-transformer," in *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2019, pp. 407–408.

[11] J. B. Hortiguela, J. Mangues-Bafalluy, R. Martinez, L. Vettori, K. Antevski, C. J. Bernardos, and X. Li, "Realizing the network service federation vision: Enabling automated multidomain orchestration of network services," *IEEE Vehicular Technology Magazine*, vol. 15, no. 2, pp. 48–57, 2020.

[12] J. Baranda, J. Mangues-Bafalluy, R. Martínez, L. Vettori, K. Antevski, C. J. Bernardos, and X. Li, "5g-transformer meets network service federation: design, implementation and evaluation," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 175–179.

[13] J. Baranda, J. Mangues-Bafalluy, L. Vettori, R. Martínez, K. Antevski, L. Girletti, C. Bernardos, K. Tomakh, D. Kucherenko, G. Landi *et al.*, "Nfv service federation: enabling multi-provider ehealth emergency services," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2020, pp. 1322–1323.

[14] C. Papagianni, J. Mangues-Bafalluy, P. Bermudez, S. Barmpounakis, D. De Vleeschauwer, J. Brenes, E. Zeydan, C. Casetti, C. Guimarães, P. Murillo *et al.*, "5growth: Ai-driven 5g for automation in vertical industries," in *2020 European Conference on Networks and Communications (EuCNC)*. IEEE, 2020, pp. 17–22.

[15] K. Antevski and C. J. Bernardos, "Federation of 5g services using distributed ledger technologies," *Internet Technology Letters*, p. e193.

[16] D. Stezenbach, M. Hartmann, and K. Tutschku, "Parameters and challenges for virtual network embedding in the future internet," in *2012 IEEE Network Operations and Management Symposium*. IEEE, 2012, pp. 1272–1278.

[17] P. T. A. Quang, A. Bradai, K. D. Singh, G. Picard, and R. Riggio, "Single and multi-domain adaptive allocation algorithms for vnf forwarding graph embedding," *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 98–112, 2019.

[18] A. F. T. Martins, M. A. T. Figueiredo, P. M. Q. Aguiar, N. A. Smith, and E. P. Xing, "Ad3: Alternating directions dual decomposition for map inference in graphical models," *Journal of Machine Learning Research*, vol. 16, no. 16, pp. 495–545, 2015. [Online]. Available: http://jmlr.org/papers/v16/martins15a.html

[19] P. T. A. Quang, A. Bradai, K. D. Singh, and Y. Hadjadj-Aoul, "Multi-domain non-cooperative vnf-fg embedding: A deep reinforcement learning approach," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 886–891.

[20] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[21] G. Li, H. Zhou, B. Feng, and G. Li, "Context-aware service function chaining and its cost-effective orchestration in multi-domain networks," *IEEE Access*, vol. 6, pp. 34 976–34 991, 2018.

[22] Q. Zhang, X. Wang, I. Kim, P. Palacharla, and T. Ikeuchi, "Service function chaining in multi-domain networks," in *2016 Optical Fiber Communications Conference and Exhibition (OFC)*, 2016, pp. 1–3.

[23] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 135–146. [Online]. Available: https://doi.org/10.1145/1807167.1807184

[24] J. C. Cisneros, S. Yangui, S. E. Pomares Hernández, J. C. Pérez Sansalvador, and K. Drira, "Coordination algorithm for migration of shared vnfs in federated environments," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, 2020, pp. 252–256.

[25] Yang Wang, Gaogang Xie, Zhenyu Li, Peng He, and K. Salamatian, "Transparent flow migration for nfv," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, 2016, pp. 1–10.

[26] B. Sonkoly, R. Szabó, B. Németh, J. Czentye, D. Haja, M. Szalay, J. Dóka, B. P. Gerő, D. Jocha, and L. Toka, "5g applications from vision to reality: Multi-operator orchestration," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 7, pp. 1401–1416, 2020.

[27] B. Németh, B. Sonkoly, M. Rost, and S. Schmid, "Efficient service graph embedding: A practical approach," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016, pp. 19–25.

[28] L. Ma and X. Wang and X. Wang and L. Wang and Y. Shi and M. Huang, "TCDA: Truthful Combinatorial Double Auctions for Mobile Edge Computing in Industrial Internet of Things," *IEEE Transactions on Mobile Computing*, no. 01, pp. 1–1, 2021.

[29] B. Zhou, S. N. Srirama, and R. Buyya, "An auction-based incentive mechanism for heterogeneous mobile clouds," *Journal of Systems and Software*, vol. 152, pp. 151–164, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219300548

[30] L. Toka, M. Zubor, A. Korosi, G. Darzanos, O. Rottenstreich, and B. Sonkoly, "Pricing games of nfv infrastructure providers," *Telecommunication Systems*, pp. 1–14, 2020.

[31] G. George, R. Wolski, C. Krintz, and J. Brevik, "Analyzing aws spot instance pricing," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2019, pp. 222–228.

[32] M. Baughman, S. Caton, C. Haas, R. Chard, R. Wolski, I. Foster, and K. Chard, "Deconstructing the 2017 changes to aws spot market pricing," in *Proceedings of the 10th Workshop on Scientific Cloud Computing*, ser. ScienceCloud '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 19–26. [Online]. Available: https://doi.org/10.1145/3322795.3331465

[33] A. Sarah, K. Lee, and H. Kim, "Lstm model to forecast time series for ec2 cloud price," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2018, pp. 1085–1088.

[34] J. Lancon, Y. Kunwar, D. Stroud, M. McGee, and R. Slater, "Aws ec2 instance spot price forecasting using lstm networks," *SMU Data Science Review*, vol. 2, no. 2, p. 8, 2019.

[35] A. Solano and L. M. Contreras, "Information exchange to support multi-domain slice service provision for 5g/nfv," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 773–778.

[36] H. Xu and B. Li, "Dynamic cloud pricing for revenue maximization," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 158–171, 2013.

[37] H. R. Lewis, "Computers and intractability. a guide to the theory of np-completeness," 1983.

[38] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957. [Online]. Available: http://www.jstor.org/stable/24900506

[39] Watkins, Christopher JCH and Dayan, Peter, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[40] K. Antevski, J. Martín-Pérez, A. Garcia-Saavedra, C. J. Bernardos, X. Li, J. Baranda, J. Mangues-Bafalluy, R. Martnez, and L. Vettori, "A q-learning strategy for federation of 5g services," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[41] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[42] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[43] K. S. Geoffrey Hinton, Nitish Srivastava, "Overview of mini-batch gradient descent," 2012. [Online]. Available: http://www.cs.toronto.edu/\%7Etijmen/csc321/slides/lecture_slides_lec6.pdf

Jorge Martín Pérez is a telematics engineering Ph.D. student at Universidad Carlos III de Madrid. He obtained a BSc in mathematics in 2016, and a BSc in computer science in 2016, both at Universidad Autónoma de Madird. Latter on 2017 he obtained a M.Sc. in telematics engineering in Universidad Carlos III de Madrid. His research focuses in optimal resource allocation in 5G networks, and since 2016 he participates in EU funded research projects.

Kiril Antevski received the B.S. degree in telecommunication engineering from the Saints Cyril and Methodius University of Skopje, Macedonia in 2012 and the M.S. degree in telecommunication engineering from the Politecnico di Milano, Milan, Italy in 2016. He is currently pursuing the Ph.D. degree in telematics engineering at University Carlos III Madrid (UC3M), Spain. His research interest includes the development of mechanisms to integrate and enhance NFV and MEC for 5G Networks in dynamic and heterogeneous environments.

Andres Garcia-Saavedra received his M.Sc and Ph.D. from University Carlos III of Madrid (UC3M) in 2010 and 2013, respectively. He then joined the Hamilton Institute, Ireland, as a research fellow until the end of 2014 when he moved to Trinity College Dublin. Since July 2015 he has been working as a research scientist at NEC Laboratories Europe. His research interests lie in the application of fundamental mathematics to real-life computer communications systems, and the design and prototyping of wireless communication systems and protocols.

Xi Li received her M.Sc in electronics and telecommunication engineering from Technische Universität Dresden in 2002 and her Ph.D. from the University of Bremen in 2009. Between 2003 and 2014 she worked as a research fellow and lecturer at the Communication Networks Group in the University of Bremen, leading a team working on several industrial and European R&D projects on 3G/4G mobile networks and future Internet design. From 2014 to 2015 she worked as a solution designer in Tele-fonica Germany GmbH & Co. OHG, Hamburg. Since March 2015 she has been a senior researcher in 5G Networks R&D at NEC Laboratories Europe.

Carlos J. Bernardos received a Telecommunication Engineering degree in 2003, and a PhD in Telematics in 2006, both from the University Carlos III of Madrid, where he worked as a research and teaching assistant from 2003 to 2008 and, since then, has worked as an Associate Professor. His research interests include IP mobility management, network virtualization, cloud computing, vehicular communications and experimental evaluation of mobile wireless networks. He has published over 70 scientific papers in international journals and conferences. He has participated in several EU funded projects, being the project coordinator of 5G-TRANSFORMER and 5Growth.