Bega, D.; Gramaglia, M.; Banchs, A.; Sciancalepore, V.; Costa-Pérez, X. A machine learning approach to 5G infrastructure market optimization, in: *IEEE transactions on mobile computing, 19(3), March 2020, Pp.* 498-512

DOI: https://doi.org/10.1109/TMC.2019.2896950

A Machine Learning approach to 5G Infrastructure Market optimization

Dario Bega, Marco Gramaglia, Albert Banchs, *Senior Member, IEEE,* Vincenzo Sciancalepore *Member, IEEE,* Xavier Costa-Pérez, *Senior Member, IEEE*

Abstract—It is now commonly agreed that future 5G Networks will build upon the network slicing concept. The ability to provide virtual, logically independent "slices" of the network will also have an impact on the models that will sustain the business ecosystem. Network slicing will open the door to new players: the infrastructure provider, which is the owner of the infrastructure, and the tenants, which may acquire a network slice from the infrastructure provider to deliver a specific service to their customers. In this new context, how to correctly handle resource allocation among tenants and how to maximize the monetization of the infrastructure become fundamental problems that need to be solved. In this paper, we address this issue by designing a network slice admission control algorithm that (i) autonomously learns the best acceptance policy while (ii) it ensures that the service guarantees provided to tenants are always satisfied. The contributions of this paper include: (i) an analytical model for the admissibility region of a network slicing-capable 5G Network, (ii) the analysis of the system (modeled as a Semi-Markov Decision Process) and the optimization of the infrastructure providers revenue, and (iii) the design of a machine learning algorithm that can be deployed in practical settings and achieves close to optimal performance.

Index Terms—Network Slicing, Admission Control, Neural Networks, Machine Learning, 5G Networks

1 Introduction

The expectations that build around future 5G Networks are very high, as the envisioned Key Performance Indicators (KPIs) represent a giant leap when compared to the legacy 4G/LTE networks. Very high data rates, extensive coverage, sub-ms delays are just few of the performance metrics that 5G networks are expected to boost when deployed.

This game changer relies on new technical enablers such as Software-Defined Networking (SDN) or Network Function Virtualization (NFV) that will bring the network architecture from a purely *hardbox* based paradigm (*e.g.*, a eNodeB or a Packet Gateway) to a completely *cloudified* approach, in which network functions that formerly were hardware-based (e.g., baseband processing, mobility management) are implemented as software Virtual Network Functions (VNFs) running on a, possibly hierarchical, general purpose *telco-cloud*.

Building on these enablers, several novel key concepts have been proposed for next generation 5G networks [1]; out of those, *Network Slicing* [2] is probably the most important one. Indeed, there is a wide consensus in that accommodating the very diverse requirements demanded by 5G services using the same infrastructure will not be possible with the current, relatively monolithic architecture in a cost efficient way. In contrast, with network slicing the infrastructure can be divided in different slices, each of which can be tailored to meet specific service requirements.

A network slice consists of a set of VNFs that run on a virtual network infrastructure and provide a specific telecommunication service. The services provided are usually typified in macro-categories, depending on the most important KPIs they target. Enhanced Mobile Broadband (eMBB), massive Machine Type Communication (mMTC) or Ultra Reliable Low Latency Communication (URLLC) are the type of services currently envisioned by, *e.g.*, ITU [3]. Each of these services is instantiated in a specific network slice, which has especially tailored management and orchestration algorithms to perform the lifecycle management within the slice.

In this way, heterogeneous services may be provided using the same infrastructure, as different telecommunication services (that are mapped to a specific slice) can be configured independently according to their specific requirements. Additionally, the *cloudification* of the network allows for the cost-efficient customization of network slices, as the slices run on a shared infrastructure.

Network Slicing enables a new business model around mobile networks, involving new entities and opening up new business opportunities. The new model impacts all the players of the mobile network ecosystem. For end-users, the capability of supporting extreme network requirements [3] enables new services that could not be satisfied with current technologies, providing a quality of experience beyond that of todays networks. For tenants such as service providers, network slicing allows to tailor the network service to the specific

- *D. Bega and A. Banchs are with the IMDEA Networks Institute, 28918 Leganés, Spain, and also with the Telematic Engineering Department, University Carlos III of Madrid, 28911 Leganés, Spain (e-mail:dario.bega@imdea.org; banchs@it.uc3m.es).*
- *M. Gramaglia is with the Telematic Engineering Department, University Carlos III of Madrid, 28911 Leganés, Spain(e-mail: mgramagl@it.uc3m.es).*
- *V. Sciancalepore and X. Costa-Pérez are with NEC Laboratories Europe, 69115 Heidelberg, Germany (e-mail: xavier.costa@neclab.eu).*

Manuscript received April 25, 2018; revised June 27, 2018 and November 27, 2018; accepted January 25, 2019. Date of publication xxxx xx, xxxx; date of current version xxxx xx, xxxx. (Corresponding author: Dario Bega.) For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. xx.xxxx/TMC.xxxx.xxxxxxx

needs of the service being provided, adjusting the slice's operation to the service provider's needs. For mobile network operators, network slicing allows to target new customers with specific service requirements, such as industrial sectors with stringent requirements, ultimately providing new revenue streams coming from the new customers.

This business model underlying network slicing is the Infrastructure as a Service (IaaS), which is expected to increase the number of available revenue streams in 5G. This model has already been successfully applied to the cloud computing infrastructure by providers such as Amazon AWS or Microsoft Azure. However, cloud computing platforms are selling to customers (*i.e.*, tenants) cloud resources (*e.g.*, CPU, memory, storage) only, while in a 5G Infrastructure market such as the one enabled by network slicing, the traded goods also include network resources (*e.g.*, spectrum, transport network). This entails a totally different problem due to the following reasons: (i) spectrum is a scarce resource for which over-provisioning is not possible, (ii) the actual capacity of the systems (*i.e.*, the resources that can actually be sold) heavily depends on the mobility patterns of the users, and (iii) the Service Level Agreements (SLAs) with network slices tenants usually impose stringent requirements on the Quality of Experience (QoE) perceived by their users. Therefore, in contrast to IaaS, in our case applying a strategy where all the requests coming to the infrastructure provider are admitted is simply not possible.

The ultimate goal of InPs is to obtain the highest possible profit from of the deployed infrastructure, thus maximizing monetization. The design of a network slice admission control policy that achieves such goal in this spectrum market is still an open problem. More specifically, the network capacity broker algorithm that has to decide on whether to admit or reject new network slice requests shall simultaneously satisfy two different goals: (i) meeting the service guarantees requested by the network slices admitted while (ii) maximizing the revenue of a network infrastructure provider.

The goal of meeting the desired service guarantees needs to consider radio related aspects, as a congested network will likely not be able to meet the service required by a network slice. Conversely, the goal of maximizing the revenue obtained by the admission control should be met by applying an on-demand algorithm that updates the policies as long as new requests arrive.

In this paper, we propose a Machine Learning approach to the 5G Infrastructure Market optimization. More specifically, the contributions of this paper are: (i) we provide an analytical model for the admissibility region in a sliced network, that provide formal service guarantees to network slices, and (ii) we design an online Machine Learning based admission control algorithm that maximizes the monetization of the infrastructure provider.

Machine learning is the natural tool to address such a complex problem. As discussed in detail along the paper, this problem is highly dimensional (growing linearly with the number of network slices classes) with a potentially huge number of states (increasing exponentially with the number of classes) and many variables (one for each state). Furthermore, in many cases the behavior of the tenants that request slices is not known *a priori* and may vary with time. For these reasons, traditional solutions building on optimization techniques are not affordable (because of complexity reasons) or simply impossible (when slice behavior is not known). Instead, machine learning provides a mean to cope with such complex problems while learning the slice behavior on the fly, and thus allows to develop a practical approach to deal with such a complex and potentially unknown system.

The rest of the paper is structured at follows. In Section 2 we review the relevant works related to this paper in the fields of resource allocation for network slicing aware networks, network slice admission control and machine learning applied to 5G Networks. In Section 3 we describe our System Model, while the analytical formulation for the network slice admissibility region is provided in Section 4. In Section 5 we model the decision-making process by means of a Markovian analysis, and derive the optimal policy which we use as a benchmark. In Section 6 we present a Neural Networks approach based on deep reinforcement learning, which provides a practical and scalable solution with close to optimal performance. Finally, in Section 7 we evaluate the proposed algorithm in a number of scenarios to assess its performance in terms of optimality, scalability and adaptability to different conditions, before concluding the paper in Section 8.

2 State of the art

While the network slicing concept has only been proposed recently [2], it has already attracted substantial attention. 3GPP has started working on the definition of requirements for network slicing and the design of a novel network architecture for supporting it [4], whereas the Next Generation Mobile Networks Alliance (NGMN) identified network sharing among slices (the focus of this paper) as one of the key issues to be addressed [5]. While there is a body of work on the literature on spectrum sharing [6]–[9], these proposal are not tailored to the specific requirements of the 5G ecosystem. Conversely, most of the work has focused on architectural aspects [10], [11] with only a limited focus on resource allocation algorithms. In [12], the authors provide an analysis of network slicing admission control and propose a learning algorithm; however, the proposed algorithm relies on an offline approach, which is not suitable for a continuously varying environment such as the 5G Infrastructure market. Moreover, the aim is to maximize the overall network utilization, in contrast to our goal here which is focused on maximizing InP revenues.

The need for new algorithms that specifically targets the monetization of the network has been identified in [13]. However, there are still very few works on this topic. The work in [14] analyzes the problem from an economical perspective, proposing a revenue model for the InP. The authors of [15] build an economic model that describes the Mobile Network Operator (MNO) profit when dealing with the network slice admission control problem, and propose a decision strategy to maximize the expected overall network profit. The proposed approach, however, is not on demand and requires the full

knowledge of arriving requests statistics, thus making it impracticable in real scenarios. Another work in this field is the one of [14], with similar limitations.

Learning algorithms are in the spotlight since Mnith et al. [16] designed a deep learning algorithm called "deep Q-network" to deal with Atari games, and further improved it in [17] making the algorithm able to learn successful policies directly from high-dimensional sensory inputs and reach human-levels performance in most of Atari games. Another approach is "AlphaGo", which builds on deep neural networks to play with the "Go" game [18]. Many more algorithms have been proposed [19]–[23], which are mostly applied in games, robotics, natural language processing, image recognition problems.

The application of Reinforcement and Machine learning approaches to mobile networks is also gaining popularity. To name a few examples, the work in [24] proposes a Q-learning algorithm for improving the reliability of a millimeter wave (mmW) non-line-of-sight small cell backhaul system, while in [25] Q-learning is implemented to solve the adaptive call admission control problem.

Machine learning has been applied to a wide span of applications in 5G networks, ranging from channel estimation/detection for massive MIMO channel to user behavior analysis, location prediction or intrusion/anomaly detection [26]. For instance, decision tree and information-theoretic regression models have been used in [27] in order to identify radio access networks problems. The authors of [28] employ a deep learning approach for modulation classification, which achieves competitive accuracy with respect to traditional schemes. The authors of [29] apply deep neural networks to approximate optimization algorithm for wireless networks resources management.

This work is an extension of the paper in [30]. In that paper, the problem of slice admission control for revenue maximization was addressed by employing Q-learning. While this provides the ability to adapt to changing environments while achieving close to optimal performance, an inherent drawback of Q-learning is its lack of scalability, as the learning time grows excessively when the state space becomes too large. In contrast, the algorithm proposed in this paper is based on Neural Networks, and it is shown to scale with the size of the network, quickly converging to optimal performance.

To the best of our knowledge, the work presented in this paper along with the previous version in [30] are the first ones that build on Machine Learning to address the problem of admission control for a 5G Infrastructure Market, with the aim of maximizing the InP's revenue while guaranteeing the SLAs of the admitted slices.

3 System Model

As discussed in Section 1, 5G networks necessarily introduce changes in the applied business models. With the legacy and rather monolithic network architecture, the main service offered is a generic voice and best-effort mobile broadband. Conversely, the high customizability that 5G Networks introduce will enable a richer ecosystem on both the portfolio of available services and the possible business relationships. New players are expected to join the 5G market, leading to an ecosystem that is composed of (i) users that are subscribed to a given service provided by a (ii) tenant that, in turn, uses the resources (*i.e.*, cloud, spectrum) provided by an (iii) infrastructure provider.[1] In the remainder of the paper we use this high level business model as basis for our analysis. In the following, we describe in details the various aspects related to our system model.

Players. As mentioned before, in our system model there are the following players: (i) the *Infrastructure Provider, InP*, who is the owner of the network (including the antenna location and cloud infrastructure) and provides the tenants with *network slices* corresponding to a certain fraction of network resources, (ii) the tenants, which issue requests to the infrastructure provider to acquire network resources, and use these resources to serve their users, providing them a specific telecommunication service, and finally (iii) the *end-users*, which are subscribers of the service provided by a tenant which uses the resources of the infrastructure provider.

Network model. The ecosystem described above does not make any distinction on the kind of resources an InP may provide to the tenants. From the various types of resources, spectrum will typically be the most important factor when taking a decision on whether to accept a request from a tenant. Indeed, cloud resources are easier to provision, while increasing the spectrum capacity is more complex and more expensive (involving an increase on antenna densification). Based on this, in this paper we focus on the wireless access network as the most limiting factor. In our model of the wireless access, the network has a set of base stations $\mathcal{B}$ owned by an infrastructure provider. For each base station $b \in \mathcal{B}$, we let C_b denote the base station capacity. We further refer to the system capacity as the sum of the capacity of all base stations, $C = \sum_{\mathcal{B}} C_b$. We let $\mathcal{U}$ denote the set of users in the network.[2] We consider that each user $u \in \mathcal{U}$ in the system is associated to one base station $b \in \mathcal{B}$. We denote by f_{ub} the fraction of the resources of base station b assigned to user u, leading to a throughput for user u of $r_u = f_{ub}C_b$. We also assume that users are distributed among base stations according to a given probability distribution; we denote by $P_{u,b}$ the probability that user u is associated with base station b. We assume that these are independent probabilities, i.e., each user behaves independently from the others.

Traffic model. 5G Networks provide diverse services which are mapped to three different usage scenarios or slice categories: eMBB, mMTC and URLLC [3]. As the main bottleneck from a resource infrastructure market point of view is spectrum, different slice categories need to be matched based to their requirements in terms of the spectrum usage. For instance eMBB-alike slices have a higher flexibility with respect to resource usage, and can use the leftover capacity of URLLC services which have more stringent requirements on

1. While some of these roles may be further divided into more refined ones, as suggested in [31], the ecosystem adopted in this paper reflects a large consensus on the current view of 5G networks.

2. The users of the network are the *end-users* we referred to above, each of them being served by one of the *tenants*.

the needed capacity.

Following the above, in this paper we focus on elastic and inelastic traffic as it is the main distinguishing factor for spectrum usage and thus provides a fairly large level of generality. In line with previous work in the literature [32], we consider that inelastic users require a certain fixed throughput demand which needs to be satisfied *at all times*,[3] in contrast to elastic users which only need guarantees on the *average* throughput, requiring that the expected average throughput over long time scales is above a certain threshold. That is, for inelastic users throughput needs to be always (or with a very high probability) above the guaranteed rate, while the throughput for elastic users is allowed to fall below the guaranteed rate during some periods as long as the average stays above this value.

We let $\mathcal{I}$ denote the set of classes of inelastic users; each class $i \in \mathcal{I}$ has a different rate guarantee R_i which needs to be satisfied with a very high probability; we refer the probability that this rate is not met as the outage probability, and impose that it cannot exceed $\bar{P}_{out}$, which is set to a very small value. We further let N_i denote the number of inelastic users of class $i \in \mathcal{I}$, and $P_{i,b}$ be the probability that a user of class i is at base station b. Finally, we let N_e be the number of elastic users in the network and R_e their average rate guarantee.

At any given point in time, the resources of each base stations are distributed among associated users as follows: inelastic users $u \in \mathcal{I}$ are provided sufficient resources to guarantee $r_u = R_i$, while the remaining resources are equally shared among the elastic users. In case there are not sufficient resources to satisfy the requirements of inelastic users, even when leaving elastic users with no throughput, we reject as many inelastic users as needed to satisfy the required throughput guarantees of the remaining ones.

Note that the above traffic types are well aligned with the slice categories defined in 3GPP, as the elastic traffic behavior is in line with the eMBB and mMTC services, while inelastic behavior matches the requirements of URLCC services.

Network slice model. By applying the network slicing concept discussed in Section 1, the network is divided into different logical slices, each of them belonging to one tenant. Thus, we characterize a network slice by (i) its traffic type (elastic or inelastic), and (ii) its number of users (*i.e.*, the subscribers of a given service) that have to be served.

A network slice comes with certain guarantees provided by an SLA agreement between the tenant and the infrastructure provider. In our model, a tenant requests a network slice that comprises a certain number of users and a traffic type. Then, as long as the number of users belonging to a network slice is less or equal than the one included in the SLA agreement, each of them will be provided with the service guarantees corresponding to their traffic type.

A network slice may be limited to a certain geographical area, in which case the corresponding guarantees only apply to the users residing in the region. In our model, we focus on the general case and consider network slices that span over the entire network. However, the model could be easily extended to consider restricted geographical areas.

Following state of the art approaches [11], network slicing onboarding is an automated process that involves little or no human interaction between the infrastructure provider. Based on these approaches, we consider a bidding system in order to dynamically allocate network slices to tenants. With this, tenants submit requests for network slices (*i.e.*, a certain number of users of a given service) to the infrastructure provider, which accepts or rejects the request according to an admission control algorithm such as the one we propose in this paper. To that aim, we characterize slices request by:

- Network slice duration t: this is the length of the time interval for which the network slice is requested.
- Traffic type κ: according to the traffic model above, the traffic type of a slice can either be elastic or inelastic traffic.
- Network slice size N: the size of the network slice is given by the number of users it should be able to accommodate.
- Price ρ: the cost a tenant has to pay for acquiring resources for a network slice. The price is per time unit, and hence the total revenue obtained by accepting a network slice is given by $r = \rho t$.

Following the above characterization, an infrastructure provider will have catalogs of network slice blueprinted by predefined values for the tuple $\{\kappa, N, \rho\}$, which we refer to as network slice classes. Tenants issue requests for one of the slice classes available in the catalogue, indicating the total duration t of the network slice. When receiving a request, an infrastructure provider has two possible decisions: it can reject the network slice and the associate revenue to keep the resources free or it can accept the network slice and charge the tenant r dollars. If accepted, the infrastructure provider grants resources to a tenant during a t-window.

To compute the profit received by the tenant, we count the aggregated revenue resulting from all the admitted slices. This reflects the net benefit of the InP as long as (i) the costs of the InP are fixed, or (ii) they are proportional to the network utilization (in the latter case, ρ reflects the difference between the revenue and cost of instantiating a slice). We argue that this covers a wide range of cases of practical interest such as spectrum resources or computational ones. Moreover, in the cases where costs are not linear with the network usage, our analysis and algorithm could be extended to deal with such cases by subtracting the cost at a given state from the revenue.

4 Bounding the Admissibility Region

An online admission control algorithm has to decide whether to accept or reject a new incoming network slice request issued by a tenant. Such a decision is driven by a number of variables such as the expected income and the resources available. The objective of an admission control algorithm is to maximize the overall profit while guaranteeing the SLA committed to all tenants. A fundamental component of such an algorithm is

3. Note that, by ensuring that the instantaneous throughput of inelastic traffic stays above a certain threshold, it is possible to provide delay guarantees. Indeed, as long as the traffic generated by inelastic users is not excessive, by providing a committed instantaneous throughput we can ensure that queuing delays are sufficiently low.

the *admissibility region*, *i.e.*, the maximum number of network slices that can be admitted in the system while guaranteeing that the SLAs are met for all tenants. Indeed, if admitting a new network slice in the system would lead to violating the SLA of already admitted slices, then such a request should be rejected. In the following, we provide an analysis to determine the admissibility region, denoted by $\mathcal{A}$, as a first step towards the design of the optimal admission algorithm.

4.1 Admissibility region analysis

We say that a given combination of inelastic users of the various classes and elastic users belongs to the admissibility region, *i.e.*, $\{N_1, \ldots, N_{|\mathcal{I}|}, N_e\} \in \mathcal{A}$, when the guarantees described in the previous section for elastic and inelastic traffic are satisfied for this combination of users. In the following, we compute the admissibility region $\mathcal{A}$.

In order to determine whether a given combination of users of different types, $\{N_1, \ldots, N_{|\mathcal{I}|}, N_e\}$, belongs to $\mathcal{A}$, we proceed as follows. We first compute the outage probability for an inelastic user of class $i \in \mathcal{I}$, $P_{out,i}$. Let R_b be the throughput consumed by the inelastic users at b. The average value of R_b can be computed as

$$\mathbb{E}[R_b] = \sum_{j \in \mathcal{I}} N_j P_{j,b} R_j, \tag{1}$$

and the typical deviation as

$$\sigma_b^2 = \sum_{j \in \mathcal{I}} N_j \sigma_{j,b}^2, \tag{2}$$

where $\sigma_{j,b}^2$ is the variance of the throughput consumed by one inelastic user of class j, which is given by

$$\begin{aligned} \sigma_{j,b}^2 &= P_{j,b}(R_j - P_{j,b}R_j)^2 + (1 - P_{j,b})(P_{j,b}R_j)^2 \\ &= P_{j,b}(1 - P_{j,b})R_j^2. \end{aligned} \tag{3}$$

Our key assumption is to approximate the distribution of the committed throughput at base station b by a normal distribution of mean R_b and variance σ_b^2, i.e., $\mathcal{N}(\mathbb{E}[R_b], \sigma_b^2)$. Note that, according to [33], this approximation is appropriate as long as the number of users per base station in the boundary of the admissibility region is no lower than 5, which is generally satisfied by cellular networks (even in the extreme case of small cells).

The outage probability at base station b is given by the probability that the committed throughput exceeds the base station capacity, i.e.,

$$P_{out,b} = \mathbb{P}(R_b > C_b), \tag{4}$$

where C_b be the capacity of base station b.

To compute the above probability with the normal approximation, we proceed as follows:

$$P_{out,b} \approx 1 - \Phi\left(\frac{C_b + \tilde{C}_b - \mathbb{E}[R_{b,i}]}{\sigma_{b,i}}\right), \tag{5}$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution and $\tilde{C}_b$ is a continuity correction factor that accounts for the fact R_b is not a continuous variable. In line with [34], where this is applied to a binomial distribution and the correction factor is one half of the step size, in our case we a set the continuity correction factor as one half of the average step size, which yields

$$\tilde{C}_b = \frac{1}{2}\frac{\sum_{j \in \mathcal{I}} P_{j,b} N_j R_j}{\sum_{j \in \mathcal{I}} P_{j,b} N_j}. \tag{6}$$

Once we have obtained $P_{out,b}$, we compute the outage probability of an inelastic user of class i with the following expression:

$$P_{out,i} = \sum_{b \in \mathcal{B}} P_{i,b} P_{out,b}. \tag{7}$$

Next, we compute the average throughput of an elastic user. To this end, we assume that (i) in line with [32], elastic users consume all the capacity left over by inelastic traffic, (ii) there is always at least one elastic user in each base station, and (iii) all elastic users receive the same throughput on average.

With the above assumptions, we proceed as follows. The average committed throughput consumed by inelastic users at base station b is given by

$$\mathbb{E}[R_b] = \sum_{i \in \mathcal{I}} N_i P_{i,b} R_i, \tag{8}$$

which gives an average capacity left over by inelastic users equal to $C_b - \mathbb{E}[R_b]$. This capacity is entirely used by elastic users as long as the base station is not empty. The total capacity usage by elastic users is then given by the sum of this term over all base stations. As this capacity is equally shared (on average) among all elastic users, this leads to the following expression for the average throughput of an elastic user:

$$r_e = \frac{\sum_{b \in \mathcal{B}} C_b - \mathbb{E}[R_b]}{N_e}. \tag{9}$$

Based on the above, we compute the admissibility region $\mathcal{A}$ as follows. For a given number of inelastic users in each class, N_i, $i \in \mathcal{I}$, and of elastic users, N_e, we compute the outage probability of the inelastic classes, $P_{out,i}$, and the average throughput of the elastic users, r_e. If the resulting values meet the requirements for all classes, i.e., $P_{out,i} \leq \bar{P}_{out}$ $\forall i$ and $r_e \geq R_e$, then this point belongs to the admissibility region, and otherwise it does not.

4.2 Validation of the admissibility region

In order to assess the accuracy of the above analysis, we compare the admissibility region obtained theoretically against the one resulting from simulations. To this end, we consider the reference scenario recommended by ITU-T [35], which consists of $|\mathcal{B}| = 19$ base stations placed at a fixed distance of 200m. Following the system model of Section 3, we have elastic and inelastic users. All inelastic users belong to the same class, and all users (elastic and inelastic) move in the area covered by these base stations following the Random Waypoint (RWP) mobility model, with a speed uniformly distributed between 2 and 3 m/s.

The association procedure of elastic and inelastic users with base stations is as follows. Inelastic users try to associate to the nearest base station $b \in \mathcal{B}$, if it has at least R_i capacity left.

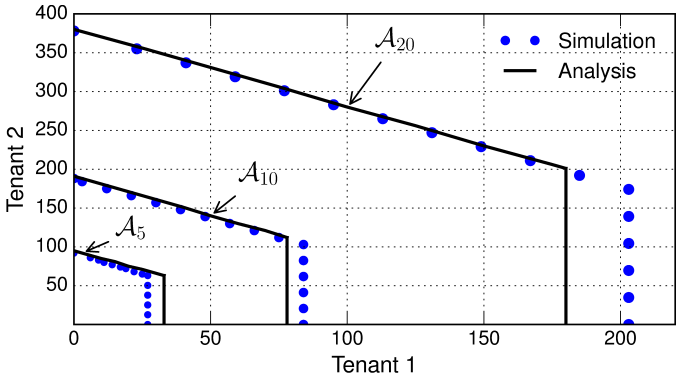


Fig. 1: Admissibility region: analysis vs. simulation.

Otherwise they do not associate and generate an outage event, joining again the network when their throughput guarantee can be satisfied. When associating, they consume a capacity R_i from the base station. The probability of association to each base station (i.e., the $P_{i,b}$ values) are extracted from the simulations and fed into the analysis.

Similarly to inelastic users, elastic users always associate to the nearest base station. All the elastic users associated to a base station fairly share among them the capacity left over by inelastic users. Upon any association event, the throughput received by the users associated to the new and the old base station changes accordingly.

Following the above procedure, we have simulated all the possible combinations of inelastic and elastic users, $\{N_i, N_e\}$. For each combination, we have evaluated the average throughput received by elastic users, computed over samples of 10 seconds time windows, and the outage probability P_{out} of inelastic users, computed as the fraction of time over which they do not enjoy their guaranteed throughput. If these two metrics (average elastic traffic throughput and inelastic traffic outage probability) are within the guarantees defined for the two traffic types, we place this combination inside the admissibility region, and otherwise we place it outside.

Fig. 1 shows the boundaries of the admissibility region obtained analytically and via simulation, respectively, for different throughput guarantees for elastic and inelastic users ($\mathcal{A}_5 : R_i = R_e = C_b/5$, $\mathcal{A}_{10} : R_i = R_e = C_b/10$ and $\mathcal{A}_{20} : R_i = R_e = C_b/20$) and $\bar{P}_{out} = 0.01$. We observe that simulation results follow the analytical ones fairly closely. While in some cases the analysis is slightly conservative in the admission of inelastic users, this serves to ensure that inelastic users' requirements in terms of outage probability are always met.

5 Markovian Model for the Decision-Making Process

While the admissibility region computed above provides the maximum number of elastic and inelastic users that can be admitted, an optimal admission algorithm that aims at maximizing the revenue of the infrastructure provider may not always admit all the requests that fall within the admissibility region. Indeed, when the network is close to congestion, admitting a request that provides a low revenue may prevent the infrastructure provider from admitting a future request with a higher revenue associated. Therefore, the infrastructure provider may be better off by rejecting the first request with the hope that a more profitable one will arrive in the future.

The above leads to the need for devising an admission control strategy for incoming slice requests. Note that the focus is on the admission of slices, in contrast to traditional algorithms focusing on the admission of users; once a tenant gets its slice admitted and instantiated, it can implement whatever algorithm it considers more appropriate to admit users into the slice.

In the following, we model the decision-making process on slice requests as a Semi-Markov Decision Process (SMDP).[4] The proposed model includes the definition of the state space of the system, along with the decisions that can be taken at each state and the resulting revenues. This is used as follows: (i) to derive the optimal admission control policy that maximizes the revenue of the infrastructure provider, which serves as a benchmark for the performance evaluation of Section 7, and (ii) to lay the basis of the machine learning algorithm proposed in Section 6, which implicitly relies on the states and decision space of the SMDP model.

5.1 Decision-making process analysis

SMDP is a widely used tool to model sequential decision-making problems in stochastic systems such as the one considered in this paper, in which an agent (in our case the InP) has to take decisions (in our case, whether to accept or reject a network slice request) with the goal of maximizing the reward or minimizing the penalty. For simplicity, we first model our system for the case in which there are only two classes of slice requests of fixed size $N = 1$, i.e., for one elastic user or for one inelastic user. Later on, we will show how the model can be extended to include an arbitrary set of network slice requests of different sizes.

The Markov Decision Process theory [36] models a system as: (i) a set of states $s \in S$, (ii) a set of actions $a \in A$, (iii) a transition function $P(s, a, s')$, (iv) a time transition function $T(s, a)$, and (v) a reward function $R(s, a)$. The system is driven by events, which correspond to the arrival of a request for an elastic or an inelastic slice as well as the departure of a slice (without loss of generality, we assume that arrivals and departures never happen simultaneously, and treat each of them as a different event). At each event, the system can be influenced by taking one of the possible actions $a \in A$. According to the chosen actions, the system earns the associated reward function $R(s, a)$, the next state is decided by $P(s, a, s')$ while the transition time is defined by $T(s, a)$.

The inelastic and elastic network slices requests follow two Poisson processes $\mathcal{P}_i$ and $\mathcal{P}_e$ with associated rates of λ_i and λ_e, respectively. When admitted into the system, the slices occupy the system resources during an exponentially distributed time of average $\frac{1}{\mu_i}$ and $\frac{1}{\mu_e}$. Additionally, they generate a revenue per time unit for the infrastructure provider of ρ_i and ρ_e. That is, the total revenue r generated by, e.g., an elastic request with duration t is $t\rho_e$.

4. Note that SMDP allows to model systems operating under continuous time such as ours, where slice requests may arrive at any point in time.

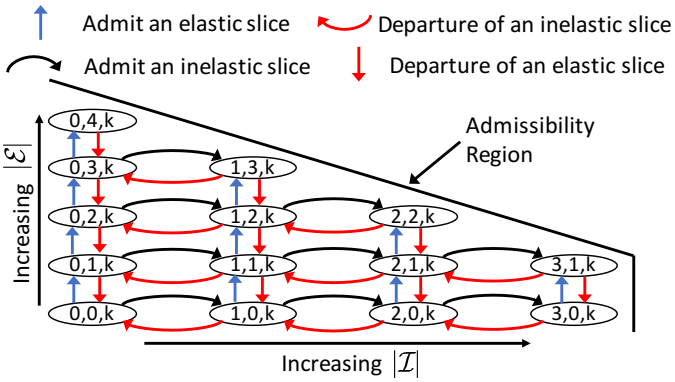


Fig. 2: Example of system model with the different states.

We define our space state S as follows. A state $s \in S$ is a three-sized tuple $(n_i, n_e, k \mid n_i, n_e \in \mathcal{A})$ where n_i and n_e are the number of inelastic and elastic slices in the system at a given decision time t, and $k \in \{i, e, d\}$ is the next event that triggers a decision process. This can be either a new arrival of a network slice request for inelastic and elastic slices ($k = i$ and $k = e$, respectively), or a departure of a network slice of any kind that left the system ($k = d$). In the latter case, n_i and n_e represent the number of inelastic and elastic slices in the system after the departure. Fig. 2 shows how the space state S relates to the admissibility region $\mathcal{A}$.

The possible actions $a \in A$ are the following: $A = G, D$. The action G corresponds to admitting the new request of an elastic or inelastic slice; in this case, the resources associated with the request are granted to the tenant and the revenue $r = \rho_{i,e} t$ is immediately earned by the infrastructure provider. In contrast, action D corresponds to rejecting the new request; in this case, there is no immediate reward but the resources remain free for future requests. Note that upon a departure ($k = d$), the system is forced to a fictitious action D that involves no revenue. Furthermore, we force that upon reaching a state in the boundary of the admissibility region computed in the previous section, the only available action is to reject an incoming request ($a = D$) as otherwise we would not be meeting the committed guarantees. Requests that are rejected are lost forever.

The transition rates between the states identified above are derived next. Transitions to a new state with $k = i$ and $k = e$ happen with a rate λ_i and λ_e, respectively. Additionally, states with $k = d$ are reached with a rate $n_i\mu_i + n_e\mu_e$ depending the number of slices already in the system. Thus, the average time the system stays at state s, $\bar{T}(s, a)$ is given by

$$\bar{T}(s, a) = \frac{1}{\upsilon(n_i, n_e)}, \tag{10}$$

where n_i, and n_e are the number of inelastic and elastic slices in state s and $\upsilon(n_i, n_e) = \lambda_i + \lambda_e + n_i\mu_i + n_e\mu_e$.

We define a policy $\pi(S)$, $\pi(s) \in A$, as a mapping from each state s to an action A. Thus, the policy determines whether, for a given number of elastic and inelastic slices in the system, we should admit a new request of an elastic or an inelastic slice upon each arrival. With the above analysis, given such a policy, we can compute the probability of staying at each of the possible states. Then, the long-term average revenue R obtained by the infrastructure provider can be computed as

$$R = \sum_{n_i, n_e, k} P(n_i, n_e, k)\left(n_i\rho_i + n_e\rho_e\right), \tag{11}$$

where ρ_i and ρ_e are the price per time unit paid by an inelastic and an elastic network slice, respectively.

The ultimate goal is to find the policy $\pi(S)$ that maximizes the long term average revenue, given the admissibility region and the network slices requests arrival process. We next devise the Optimal Policy when the parameters of the arrival process are known *a priori*, which provides a benchmark for the best possible performance. Later on, in Section 6, we design a learning algorithm that approximates the optimal policy.

5.2 Optimal policy

In order to derive the optimal policy, we build on Value Iteration [37], which is an iterative approach to find the optimal policy that maximizes the average revenue of an SMDP-based system. According to the model provided in the previous section, our system has the transition probabilities $P(s, a, s')$ detailed below.

Let us start with $a = D$, which corresponds to the action where an incoming request is rejected. In this case, we have that when there is an arrival, which happens with a rate λ_i and λ_e for inelastic and elastic requests, respectively, the request is rejected and the system remains in the same state. In case of a departure of an elastic or an inelastic slice, which happens with a rate of $n_e\mu_e$ or $n_i\mu_i$, the number of slices in the system is reduced by one unit (recall that no decision is needed when slices leave the system). Formally, for $a = D$ and $s = (n_i, n_e, i)$, we have:

$$P(s, a, s') = \begin{cases} \frac{\lambda_i}{\upsilon(n_i, n_e)}, & s' = (n_i, n_e, i) \\ \frac{\lambda_e}{\upsilon(n_i, n_e)}, & s' = (n_i, n_e, e) \\ \frac{n_i\mu_i}{\upsilon(n_i, n_e)}, & s' = (n_i - 1, n_e, d) \\ \frac{n_e\mu_e}{\upsilon(n_i, n_e)}, & s' = (n_i, n_e - 1, d) \end{cases}. \tag{12}$$

When the chosen action is to accept the request ($a = G$) and the last arrival was an inelastic slice ($k = i$), the transition probabilities are as follows. In case of an inelastic slice arrival, which happens with a rate λ_i, the last arrival remains $k = i$, and in case of an elastic arrival it becomes $k = e$. The number of inelastic slices increases by one unit in all cases except of an inelastic departure (rate $n_i\mu_i$). In case of an elastic departure (rate $n_e\mu_e$), the number of elastic slices decreases by one. Formally, for $a = G$ and $s = (n_i, n_e, i)$, we have:

$$P(s, a, s') = \begin{cases} \frac{\lambda_i}{\upsilon(n_i+1, n_e)}, & s' = (n_i + 1, n_e, i) \\ \frac{\lambda_e}{\upsilon(n_i+1, n_e)}, & s' = (n_i + 1, n_e, e) \\ \frac{(n_i+1)\mu_i}{\upsilon(n_i+1, n_e)}, & s' = (n_i, n_e, d) \\ \frac{n_e\mu_e}{\upsilon(n_i+1, n_e)}, & s' = (n_i + 1, n_e - 1, d) \end{cases}. \tag{13}$$

If the accepted slice is elastic ($k = e$), the system exhibits a similar behavior to the one described above but increasing

Algorithm 1 Value Iteration

1) Initialize the vector $V(s) = 0, \forall s \in S$. $V(s)$ represents the long term expected revenue for being in state s. Initialize the step number n to 1.
2) Update the expected reward at time $n+1$, $V_{n+1}(s)$ using the rule

$$V_{n+1}(s) = \max_{a \in A} \left[\frac{R(s,a)}{T(s,a)} \tau + \frac{\tau}{T(s,a)} \sum_{s'} P(s,a,s') V_n(s') + \left(1 - \frac{\tau}{T(s,a)}\right) V_n(s) \right] \quad \forall s \in S$$

3) Compute the boundaries

$$M_n = \max_{s \in S} \left(V_{n+1}(s) - V_n(s)\right)$$

$$m_n = \min_{s \in S} \left(V_{n+1}(s) - V_n(s)\right)$$

and check the condition

$$0 \leq (M_n - m_n) \leq \epsilon m_n$$

4) If the condition in step 3 is not fulfilled, then repeat from step 2

by one the number of elastic slices instead. Thus, for $a = G$, $s = (n_i, n_e, e)$, we have:

$$P(s,a,s') = \begin{cases} \frac{\lambda_i}{v(n_i,n_e+1)}, & s' = (n_i, n_e+1, i) \\ \frac{\lambda_e}{v(n_i,n_e+1)}, & s' = (n_i, n_e+1, e) \\ \frac{n_i \mu_i}{v(n_i,n_e+1)}, & s' = (n_i-1, n_e+1, d) \\ \frac{(n_e+1)\mu_e}{v(n_i,n_e+1)}, & s' = (n_i, n_e, d) \end{cases}. \quad (14)$$

A reward is obtained every time the system accepts a new slice, which leads to

$$R(s,a) = \begin{cases} 0, & a = D \\ t\rho_i & a = G, k = i \\ t\rho_e & a = G, k = e \end{cases}. \quad (15)$$

Applying the Value Iteration algorithm [37] for SMDP is not straightforward. The standard algorithm cannot be applied to a continuous time problem as it does not consider variable transition times between states. Therefore, in order to apply Value Iteration to our system, an additional step is needed: all the transition times need to be normalized to multiples of a faster, arbitrary, fixed transition time τ [38]. The only constraint that has to be satisfied by τ is that it has to be faster than any other transition time in the system, which leads to

$$\tau < \min T(s,a), \qquad \forall s \in S, \forall a \in A. \quad (16)$$

With the above normalization, the continuous time SMDP corresponding to the analysis of the previous section becomes a discrete time Markov Process and a modified Value Iteration algorithm may be used to devise the best policy $\pi(S)$ (see Algorithm 1). The discretized Markov Chain will hence perform one transition every τ interval. Some of these transitions correspond to transitions in continuous time system, while in the others the system keeps in the same state (we call the latter fictitious transitions).

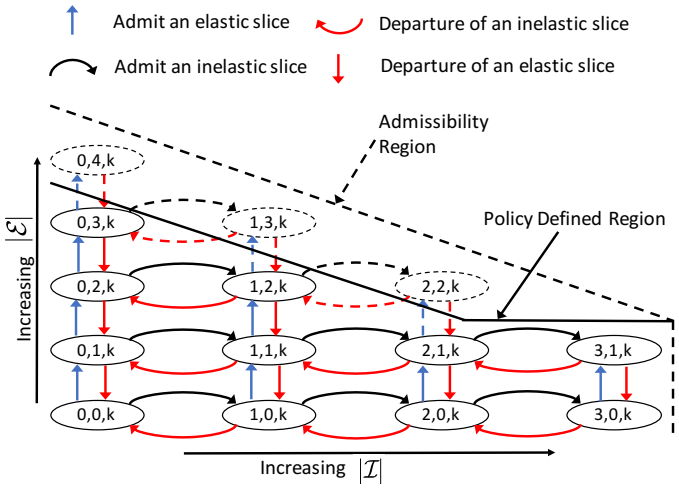


Fig. 3: Example of optimal policy for elastic and inelastic slices.

The normalization procedure affects the update rule of step 2 in Algorithm 1. All the transition probabilities $P(s,a,s')$ are scaled by a factor $\frac{\tau}{T(s,a')}$ to enforce that the system stays in the corresponding state during an average time $T(s,a')$. Also, the revenue $R(s,a)$ is scaled by a factor of $T(s,a)$ to take into account the fact that the reward $R(s,a)$ corresponds to a period $T(s,a)$ in the continuous system, while we only remain in a state for a τ duration in the discrete system. In some cases, transitions in the sampled discrete time system may not correspond to any transition in the continuous time one: this is taken into account in the last term of the equation, i.e., in case of a fictitious transition, we keep in state $V_n(s)$.

As proven in [30], Algorithm 1 is guaranteed to find the optimal policy $\pi(S)$. Such an optimal policy is illustrated in Fig. 3 for the case where the price of inelastic slice is higher than that of elastic slice ($\rho_i > \rho_e$). The figure shows those states for which the corresponding action is to admit the new request (straight line), and those for which it is to reject it (dashed lines). It can be observed that while some of the states with a certain number of elastic slices fall into the admissibility region, the system is better off rejecting those requests and waiting for future (more rewarding) requests of inelastic slice. In contrast, inelastic slice requests are always admitted (within the admissibility region).

The analysis performed so far has been limited to network slice requests of size one. In order to extend the analysis to requests of an arbitrary size, we proceed as follows. We set the space state to account for the number of slices of each different class in the system (where each class corresponds to a traffic type and a given size). Similarly, we compute the transition probabilities $P(s,a,s')$ corresponding to arrival and departures of different classes. With this, we can simply apply the same procedure as above (over the extended space state) to obtain the optimal policy.

Following [39], it can be seen that (i) Algorithm 1 converges to a certain policy, and (ii) the policy to which the algorithm converges performs arbitrarily close to the optimal policy.

6 N3AC: A DEEP LEARNING APPROACH

The Value Iteration algorithm described in Section 5.2 provides the optimal policy for revenue maximization under the framework described of Section 5.1. While this is very useful in order to obtain a benchmark for comparison, the algorithm itself has a very high computational cost, which makes it impractical for real scenarios. Indeed, as the algorithm has to update all the V values $V(s)$, $s \in S$ at each step, the running time grows steeply with the size of the state space, and may become too high for large scenarios. Moreover, the algorithm is executed offline, and hence cannot be applied unless all system parameters are known *a priori*. In this section, we present an alternative approach, the *Network-slicing Neural Network Admission Control* (N3AC) algorithm, which has a low computational complexity and can be applied to practical scenarios.

6.1 Deep Reinforcement Learning

N3AC falls under category of the deep reinforcement learning (RL). With N3AC, an agent (the InP) interacts with the environment and takes decisions at a given state, which lead to a certain reward. These rewards are fed back into the agent, which "learns" from the environment and the past decisions using a learning function $\mathcal{F}$. This learning function serves to estimate the expected reward (in our case, the revenue).

RL algorithms rely on an underlying Markovian system such as the one described in Section 5. They provide the following features: (i) high scalability, as they learn online on an event basis while exploring the system and thus avoid a long learning initial phase, (ii) the ability to adapt to the underlying system without requiring any *a priori* knowledge, as they learn by interacting with the system, and (iii) the flexibility to accommodate different learning functions $\mathcal{F}$, which provide the mapping from the input state to the expected reward when taking a specific action.

The main distinguishing factor between different kinds of RL algorithms is the structure of the learning function $\mathcal{F}$. Techniques such as Q-Learning [40] employ a lookup table for $\mathcal{F}$, which limits their applicability due to the lack of scalability to a large space state [41]. In particular, Q-Learning solutions need to store and update the expected reward value (*i.e.*, the Q-value) for each state-action pair. As a result, learning the right action for every state becomes infeasible when the space state grows, since this requires experiencing many times the same state-action pair before having a reliable estimation of the Q-value. This leads to extremely long convergence times that are unsuitable for most practical applications. Additionally, storing and efficiently visiting the large number of states poses strong requirements on the memory and computational footprint of the algorithm as the state space grows. For the specific case studied in this paper, the number of states in our model increases exponentially with the number of network slicing classes. Hence, when the number of network slicing classes grows, the computational resources required rapidly become excessive.

A common technique to avoid the problems described above for Q-learning is to *generalize* the experience learned from some states by applying this knowledge to other similar states, which involves introducing a different $\mathcal{F}$ function. The key idea behind such *generalization* is to exploit the knowledge obtained from a fraction of the space state to derive the right action for other states with similar *features*. There are different generalization strategies that can be applied to RL algorithms. The most straightforward technique is the linear function approximation [42]. With this technique, each state is given as a linear combination of functions that are representative of the system features. These functions are then updated using standard regression techniques. While this approach is scalable and computationally efficient, the right selection of the feature functions is a very hard problem. In our scenario, the Q-values associated to states with similar features (*e.g.*, the number of inelastic users) are increasingly non linear as the system becomes larger. As a result, linearization does not provide a good performance in our case.

Neural Networks (NNs) are a more powerful and flexible tool for generalization. NNs consist of simple, highly interconnected elements called *neurons* that learn the statistical structure of the inputs if correctly *trained*. With this tool, the design of neuron internals and the interconnection between neurons are the most important design parameters. RL algorithms that employ NNs are called Deep RL algorithms: N3AC belongs to this family. One of the key features of such a NN-based approach is that it only requires storing a very limited number of variables, corresponding to the weights and biases that compose the network architecture; yet, it is able to accurately estimate the $\mathcal{F}$ function for a very large number of state/action pairs. In the rest of this Section, we review the NNs design principles (Section 6.2) and explain how these principles are applied to a practical learning algorithm for our system (Section 6.3).

6.2 Neural networks framework

The fundamental building blocks of DRL algorithms are the following ones [43]:

- A set of labeled data (i.e., system inputs for which the corresponding outputs are known) which is used to train the NN (i.e., teach the network to approximate the features of the system).
- A loss function that measures the neural network performance in terms of training error (*i.e.*, the error made when approximating the known output with the given input).
- An optimization procedure that reduces the loss functions at each iterations, making the NN eventually converge.

There are many different Machine Learning (ML) schemes that make use of NNs, which are usually categorized as supervised, unsupervised and RL. A ML system is supervised or unsupervised depending on whether the labeled data is available or not, and it is a RL system when it interacts with the environment receiving feedback from its experiences. N3AC falls under the latter category, and, within RL, it falls under the category of deep RL. Since the seminal work in [16], deep RL techniques have gained momentum and are nowadays one of the most popular approaches for RL. In spite of the bulk of literature available for such techniques, devising the N3AC

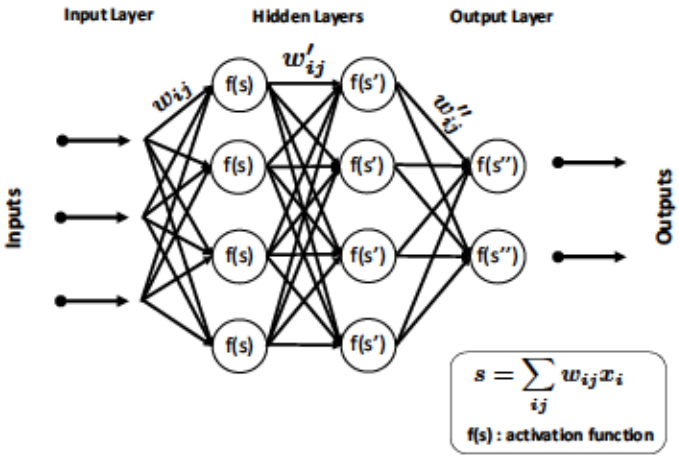


Fig. 4: Neural networks internals.

algorithm involves a number of design decisions to address the specificities of our problem, which are summarized in the following.

Neuron internal configuration. An exemplary NN is illustrated in Figure 4, where we have multiple layers of interconnected neurons organized as: (i) an input, (ii) an output and (iii) one or more hidden layers. A neuron is a non-linear element that executes a so-called *activation function* to the linear combination of the weighted inputs. Many different activation functions are available in the literature spanning from a linear function to more complex ones such as $tanh$, sigmoid or the Rectified Linear Unit (ReLU) [44].N3AC employs the latter.

Neural Network Structure. One of the design choices that needs to be taken when devising a NN approach is the the way neurons are interconnected among them. The most common setup is *feed-forward*, where the neurons of a layer are fully interconnected with the ones of the next. There are also other configurations, such as the *convolutional* or the *recurrent* (where the output is used as input in the next iteration). However, the best choice for a system like the one studied in this paper is the feed-forward. Indeed, convolutional networks are usually employed for image recognition, while recurrent are useful when the system input and the output have a certain degree of mutual relation. None of these match our system, which is memoryless as it is based on a Markovian approach. Furthermore, our NN design relies on a single hidden layer. Such a design choice is driven by the following two observations: (i) it has been proven that is possible to approximate any function using NN with a single hidden layer [45], and (ii) while a larger number of hidden layers may improve the accuracy of the NN, it also involves a higher complexity and longer training period; as a result, one should employ the required number of hidden layers but avoid building a larger network than strictly necessary.

Back-propagation algorithm selection. In classical ML applications, the NN is trained using a labeled dataset: the NN is fed with the inputs and the difference among the estimated output and the label is evaluated with the error function. Then, the error is processed to adjust the NN weights and thus reduce the error in the next iteration. N3AC adjusts weights using a Gradient Descent approach: the measured error at the output layer is back-propagated to the input layer changing the weights values of each layer accordingly [44]. More specifically, N3AC employs the RMSprop [46] Gradient Descent algorithm.

Integration with the RL framework. One of the critical requirements for N3AC is to operate without any previously known output, but rather interacting with the environment to learn its characteristics. Indeed, in N3AC we do not have any "ground truth" and thus we need to rely on estimations of the output, which will become more accurate as we keep exploring the system. While this problem has been extensively studied in the literature [43], [47], we need to devise a solution that is suitable for the specific problem addressed. In N3AC, we take as the output of the NN the average revenues expected at a given state when taking a specific decision. Once the decision is taken, the system transitions to the new state and we measure the average revenue resulting from the decision taken (0 in case of a rejection and ρt in case of an acceptance). Then, the error between the estimated revenue and the measured one is used to train the NN, back-propagating this error into the weights. N3AC uses two different NNs: one to estimate the revenue for each state when the selected action is to accept the incoming request, and another one when we reject the request. Upon receiving a request, N3AC polls the two NNs and selects the action with the highest expected revenue; then, after the transition to the new state is performed, the selected NN is trained. More details about the N3AC operation are provided in the next section.

Exploration vs exploitation trade-off. N3AC drives the selection of the best action to be taken at each time step. While choosing the action that maximizes the revenue at each step contributes to maximizing the overall revenue (referred to as *exploitation* step), in order to learn we also need to visit new (still unknown) states even if this may eventually lead to a suboptimal revenue (referred to as *exploration* step). This procedure is especially important during the initial interaction of the system, where estimates are very inaccurate. In N3AC, the trade-off between exploitation and exploration is regulated by the γ parameter, which indicates the probability of taking an exploration step. In the setup used in this paper, we take $\gamma = 0.1$. Once the NNs are fully trained, the system goes into exploitation only, completely omitting the exploration part.

6.3 Algorithm description

In the following, we describe the proposed N3AC algorithm. This algorithm builds on the Neural Networks framework described above, exploiting RL to train the algorithm without a ground truth sequence. The algorithm consists of the following high level steps (see Algorithm 2 for the pseudocode):

- *Step 1, acceptance decision:* In order to decide whether to accept or reject an incoming request, we look at the expected average revenues resulting from accepting and rejecting a request in the two NNs, which we refer to as the Q-values. Specifically, we define $Q(s, a)$ as the expected cumulative reward when starting from a certain state s with action a, compared to a baseline σ given by

the optimal policy reward when starting from state 0, i.e.,

$$Q(s,a) = \mathbb{E}\left[\lim_{t\to\infty}\sum_{n=0}^{D(t)} R_n - \sigma t | s_0 = s, a_0 = a\right] \quad (17)$$

where $D(t)$ is the number of requests received in a period t, R_n is the revenue obtained with the n^{th} request and $\sigma = \mathbb{E}[\lim_{t\to\infty}\frac{1}{t}\sum_{n=0}^{D(t)} R_n | s_0 = 0]$ under the optimal policy. Then, we take the decision that yields the highest Q-value. This procedure is used for elastic slices only, as inelastic slices shall always be accepted as long as there is sufficient room. When there is no room for an additional slice, requests are rejected automatically, regardless of their type.

- *Step 2, evaluation:* By taking a decision in Step 1, the system experiences a transition from state s at step n, to state s' at step $n+1$. Once in step $n+1$, the algorithm has observed both the reward obtained during the transition $R(s,a)$ and a sample t_n of the transition time. The algorithm trains the weights of the corresponding NN based on the error between the expected reward of s estimated at step n and the target value. This step relies on two cornerstone procedures:
  - *Step 2a, back-propagation:* This procedure drives the weights update by propagating the error measured back through all the NN layers, and updating the weights according to their gradient. The convergence time is driven by a learning rate parameter that is used in the weight updates.
  - *Step 2b, target creation:* This procedure is needed to measure the accuracy of the NNs estimations during the learning phase. At each iteration our algorithm computes the observed revenue as follows:

$$\omega = R(s,a,s') - \sigma t_n + \max_{a'} Q(s',a'), \quad (18)$$

    where $R(s,a,s')$ is the revenue obtained in the transition to the new state. As we do not have labeled data, we use ω to estimate the error, by taking the difference between ω and the previous estimate $Q_{n+1}(s,a)$ and using it to train the NN. When the NN eventually converges, ω will be close to the Q-values estimates.
- *Step 3, penalization:* When a state in the boundary of the admissibility region is reached, the system is forced to reject the request. This should be avoided as it may force the system to reject potentially high rewarding slices. To avoid such cases, N3AC introduces a *penalty* on the Q-values every time the system reaches the border of the admissibility region. With this approach, if the system is brought to the boundary through a sequence of highly rewarding actions, the penalty will have small effect as the Q-values will remain high even after applying the penalty. Instead, if the system reaches the boundary following a chain of poorly rewarding actions, the impact on the involved Q-values will be much higher, making it unlikely that the same sequence of decisions is chosen in the future.

Algorithm 2 N3AC algorithm.

1) Initialize the Neural Network's weights to random values.
2) An event is characterized by: s, a, s', r, t (the starting state, the action taken, the landing state, the obtained reward and the transition time).
3) Estimate $Q(s',a')$ for each action a' available in state s' through the NN.
4) Build the target value with the new sample observation as follows:

$$target = R(s,a,s') - \sigma t_n + \max_{a'} Q(s',a') \quad (19)$$

   where t_n is the transition time between two subsequent states s and s' after action a.
5) Train the NNs through RMSprop algorithm:
   5.1 If $s \notin$ admissibility region boundary, train the NN with the error given by the difference between the above target value (step 4) and the measured one.
   5.2 Otherwise, train the NN corresponding to accepted requests by applying a *"penalty"* and train the NN corresponding to rejected requests as in step 5.1.

- *Step 4, learning finalization:* Once the learning phase is over, the NN training stops. At this point, at a given state we just take the the action that provides the highest expected reward.

We remark that the learning phase of our algorithm does not require specific training datasets. Instead, the algorithm learns from the real slice requests on the fly, during the real operation of the system; this is the so-called *exploration phase*. The training corresponding to such an exploration phase terminates when the algorithm has converged to a good learning status, and is triggered again when the system detects changes in the system that require new training.

7 Performance Evaluation

In this section we evaluate the performance of N3AC via simulation. Unless otherwise stated, we consider a scenario with four slice classes, two for elastic traffic and two for inelastic. Service times follow an exponential distribution with $\mu = 5$ for all network slices classes, and arrivals follow a Poisson process with average rates equal to $\lambda_i = 2\mu$ and $\lambda_e = 10\lambda_i$ for the elastic and inelastic classes, respectively. We consider two network slice sizes, equal to $C/10$ and $C/20$, where C is the total network capacity. Similarly, we set the throughput required guarantees for elastic and inelastic traffic to $R_i = R_e = C_b/10$. Two key parameters that will be employed throughout the performance evaluation are ρ_e and ρ_i, the average revenue per time unit generated by elastic and inelastic slices, respectively (in particular, performance depends on the ratio between them). The rest of the network setup (including the users' mobility model) is based on the scenario described in Section 4.2.

Following the N3AC algorithm proposed in the previous section, we employ two feed-forward NNs, one for accepted requests and another one for rejected. Each neuron applies a ReLU activation function, and we train them during the exploration phase using the NNs RMSprop algorithm implementation available in Keras (https://keras.io/); the learning parameter of the RMSprop Gradient Descent algorithm [46] is

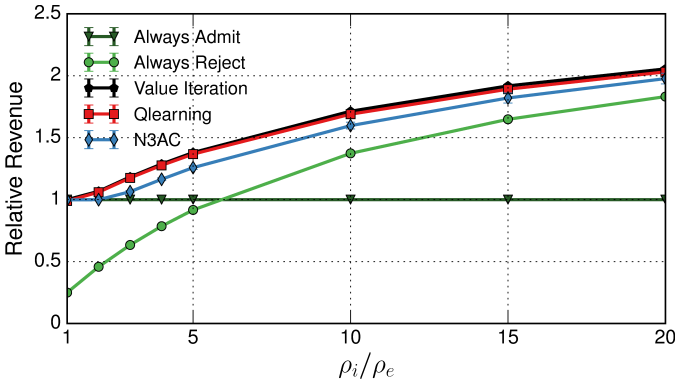


Fig. 5: Revenue vs. ρ_i/ρ_e.

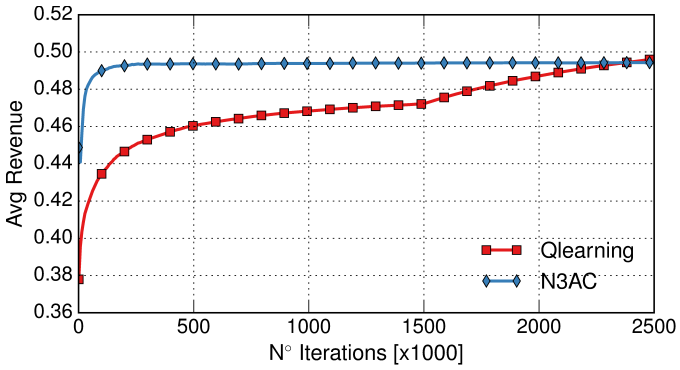


Fig. 6: Learning time for N3AC and Q-learning.

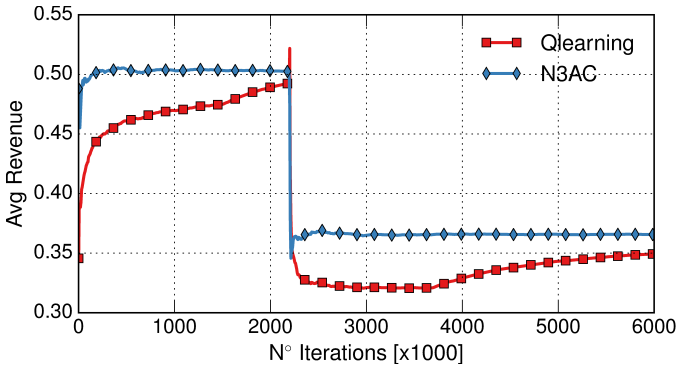


Fig. 7: Performance under changing conditions.

equal to 0.001. The number of input nodes in the NN is equal to the size of the space state (*i.e.*, the number of considered classes plus one for the next request k), the number of neurons in the hidden layer equal to 40 for the scenario described in Sections 7.3 and 7.4 and 20 for the others, and the output layer is composed of one neuron, applying a linear function. Note that, while we are dealing with a specific NN structure, one of the key highlights of our results is that the adopted structure works well for a wide range of different 5G networks.

In the results given in this section, when relevant we provide the 99% confidence intervals over an average of 100 experiments (note that in many cases the confidence intervals are so small that they cannot be appreciated).

7.1 Algorithm Optimality

We first evaluate the performance of the N3AC algorithm (which includes a hidden layer of 20 neurons) by comparing it against: (i) the benchmark provided by the optimal algorithm, (ii) the Q-learning algorithm proposed in [30], and (iii) two naive policies that always admit elastic traffic requests and always reject them, respectively. In order to evaluate the optimal algorithm and the Q-learning one, which suffers from scalability limitations, we consider a relatively small scenario. Figure 5 shows the relative average reward obtained by each of these policies, taking as baseline the policy that always admit all network slice requests (which is the most straightforward algorithm).

We observe that N3AC performs very closely to the Q-learning and optimal policies, which validates the proposed algorithm in terms of optimality. We further observe that the revenue improvements over the naive policies is very substantial, up to 100% in some cases. As expected, for small ρ_i/ρ_e the policy that always admits all requests is optimal: in this case both elastic and inelastic slices provide the same revenue. In contrast, for very large ρ_i/ρ_e ratios the performance of the "always reject" policy improves, as in this case the revenue obtained from elastic traffic is (comparatively) much smaller.

7.2 Learning time and adaptability

One of the key advantages of the N3AC algorithm as compared with Q-learning is that it requires a much shorter learning time. This is due to the fact that with N3AC the knowledge acquired at each step is used to update the Q-values of all states, while Q-learning just updates the Q-value of the lookup table for the state being visited. To evaluate the gain provided by the NNs in terms of convergence time, we analyze the evolution of the expected revenue over time for the N3AC and the Q-learning algorithms. The results are shown in Figure 6 as a function of the number of iterations. We observe that after few hundred iterations, N3AC has already learned the correct policy and the revenue stabilizes. In contrast, Q-learning needs several thousands of iterations to converge. We conclude that N3AC can be applied to much more dynamic scenarios as it can adapt to changing environments. Instead, Q-learning just works for relatively static scenarios, which limits its practical applicability. Furthermore, Q-learning cannot scale to large scenarios, as the learning time (and memory requirements) would grow unacceptably for such scenarios.

When the network conditions change, e.g., the arrival pattern of slice requests, this is detected by the system, and a new training period is triggered. To evaluate the system performance under such conditions, Figure 7 illustrates the behavior of N3AC and Q-learning. In this experiment, the arrival rate of elastic network slices is reduced to one half at a given point in time, and this is detected as the revenue drops beyond a given threshold (which we set to 10%). We observe that N3AC rapidly moves to the best point of operation, while Q-learning needs much more time to converge, leading to a substantially lower revenue. We further observe that, even in the transients, N3AC obtains a fairly good performance.

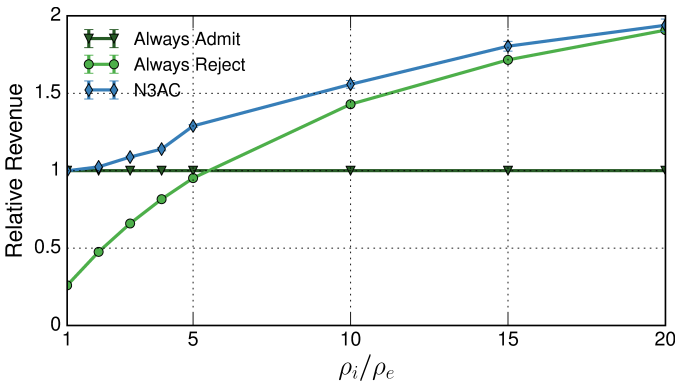


Fig. 8: Revenue vs. ρ_i/ρ_e.

7.3 Large-scale scenario

The previous results have been obtained for a relatively small scenario where the evaluation of the optimal and Q-learning algorithm was feasible. In this section, we assess the performance of the N3AC algorithm in a large-scale scenario; indeed, one of the design goals of this algorithm is its scalability to large scenarios. We consider a scenario with eight slice classes, four for elastic traffic and four for inelastic. For each traffic type, we allow four network slice sizes, linearly distributed among $C/10$ and $C/20$. We have the same throughput guarantees for elastic and inelastic traffic as in the previous experiment ($R_i = R_e = C_b/10$) and thus we have the same admissibility region (although the space state is much larger now). We set μ and λ parameters in a way that the load of the network is similar to the previous experiment.

In this larger scenario, the optimal and Q-learning algorithms are not feasible. Hence, we evaluate the performance of N3AC and compare it against the naive policies only. Figure 8 shows the relative average reward obtained by each of these policies, taking as baseline the policy that always admits all network slice requests. Similarly to the evaluation performed in the previous experiment, we observe that the N3AC algorithm always substantially outperforms the naive policies. As expected, for small ρ_i/ρ_e the policy that always admits all requests is optimal, while for very large ρ_i/ρ_e ratios the performance of "always reject" policy improves since the revenue obtained from the elastic traffic is much smaller.

7.4 Gain over random policies

While the result of the previous section shows that the proposed algorithm provides high gains, it is only compared against two naive policies and thus does not give an insight on the real revenue gains that could be achieved over smarter, yet not optimal policies. To this end, we compare the performance of the N3AC algorithm against a set of "smart" random policies which work as follows: (i) inelastic network slices requests are always accepted, and (ii) the decision of rejecting an elastic request is chosen randomly upon defining the policy for each different state. Then, by drawing a high number of random policies, it is to be expected that some of them provide good performance.

Figure 9 compares N3AC against the above approach with 1,000 and 10,000 different random policies, respectively. We note that the improvement achieved with 10,000 random policies over 1,000 is very small, which shows the the chosen setting for the random policies approach is appropriate and provides the best performance that can be achieved with such an approach. From the figure, we can see that N3AC provides substantial gains over the best performing random policy (around 20%). This confirms that a smart heuristic is not effective in optimizing revenue; indeed, with such a large space state it is very difficult to calibrate the setting for the acceptance of elastic slices that maximizes the resulting revenue. Instead, by using a NN-based approach such as N3AC, we are capable of accurately capturing such a large space state within a limited range of parameters and thus drive acceptance decisions towards very high performance.

7.5 Memory and computational footprint

One of the key aspects of the proposed framework is the memory footprint, which has a strong impact on scalability. By using NNs, N3AC does not need to keep track of the expected reward for each individual state-action $Q(s,a)$, but it only stores the weights of the NNs. Indeed, NNs capture the dynamics of the explored system based on a small number of weights, which are used to estimate the Q-values for all the states of the system. This contrasts with Q-learning, which requires to store data for each individual state. As the number of weights, fixed by the NN layout, is much smaller than the total number of states, this provides a much higher scalability, specially when the number of states grows substantially. For example, the large scale scenario evaluated in Section 7.3 has an internal space state of around 500 thousand states, which makes the Q-learning technique unfeasible for such a scenario. In contrast, N3AC only requires storing state for around 400 parameters, which represents a huge improvement in terms of scalability.

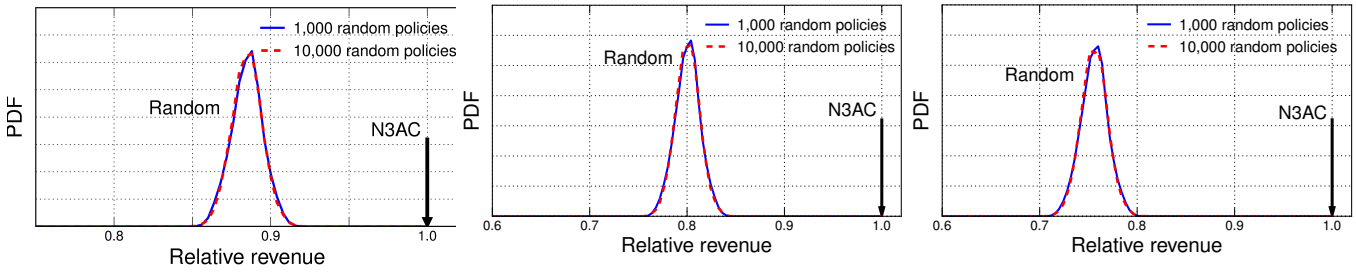
In addition to memory, the computational footprint also has a strong impact on scalability. In order to understand the computational load incurred by N3AC, we measured the time elapsed in the computation for one iteration. Table gives the results obtained with a NVIDIA GTX 1080 GPU platform for different system scenarios in terms of neurons, number of base stations and number of users. Results show that computational times are very low, and the differences between the various scenarios are almost negligible, which further confirms the ability of N3AC to scale up to very large network scenarios.

Number of neurons	Number of base stations	Number of users	Computational time (sec)
40	50	500	0.0181
40	100	1000	0.0194
40	250	1000	0.0195
100	250	2500	0.0192
100	500	2500	0.0197
100	500	5000	0.0199

TABLE 1: Computational load for different network scenarios.

7.6 Different traffic types

Our analysis so far has focused on two traffic types: elastic and inelastic traffic. In this section, we address a different



(a) $\rho_i/\rho_e = 5$ (b) $\rho_i/\rho_e = 10$ (c) $\rho_i/\rho_e = 15$

Fig. 9: The distribution of the revenues obtained by random smart policies compared to the N3AC algorithm.

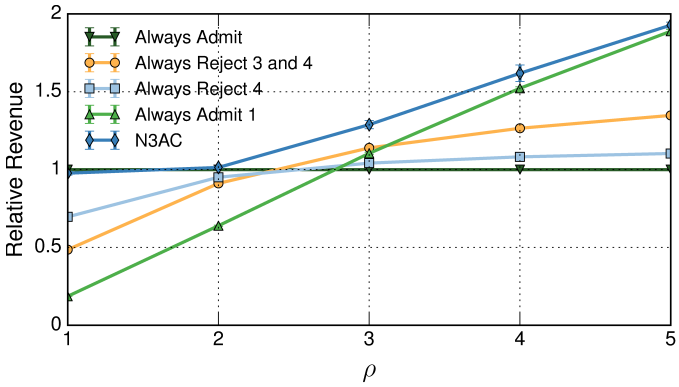


Fig. 10: Revenue vs. ρ_i/ρ_e.

scenario that includes the traffic types corresponding to the four service classes defined by 3GPP [48] (hereafter we refer to them as class 1 to class 4, where class 1 is the one with most stringent delay requirements). In line with the analysis of Section 4, for this scenario with 4 different traffic types we take the admissibility region $\mathcal{A}^*$ given by (i) $|\mathcal{T}_1| \leq \mathcal{T}_1^{max}$, (ii) $|\mathcal{T}_1| + |\mathcal{T}_2| \leq \mathcal{T}_2^{max}$, (iii) $|\mathcal{T}_1| + |\mathcal{T}_2| + |\mathcal{T}_3| \leq \mathcal{T}_3^{max}$, and (iv) $|\mathcal{T}_1| + |\mathcal{T}_2| + |\mathcal{T}_3| + |\mathcal{T}_4| \leq \mathcal{T}_4^{max}$. For this scenario, we run the same experiment as in Section 7.3, varying the price ratio ρ among different classes as follows: $r_k = \rho \cdot r_{k+1}$ $\forall k$, where r_k is the revenue generated by class k. Figure 10 compares the performance provided by the N3AC algorithm in this scenario against the one provided by naive policies which only accept a subset of classes. We can observe that N3AC provides very high gains when compared to all the naive policies, which confirms that our approach can be successfully applied to scenarios with more traffic types, such as, e.g., the 3GPP service classes.

8 Conclusion

Network Slicing will be one of the pillars of future 5G networks. It is expected that this new paradigm will bring new players to the business: Infrastructure Providers will sell their resources to tenants which, in turn, provide a service to their users. An open problem within this model is how to admit requests from the tenants, ensuring that the corresponding SLAs will be satisfied while maximizing the monetization of the Infrastructure Provider. In this paper we propose a machine learning approach to address this problem. To this aim, we first present a model based on SMDP for the decision-making process and formulate the optimal revenue problem. Then, building on this model, we design an algorithm based on Neural Network: the N3AC algorithm. Our evaluation shows that N3AC (i) performs close to the optimal under a wide range of configurations, (ii) substantially outperforms naive approaches as well as smart heuristics, and (iii) only requires a few hundred of iterations to converge to optimal performance. Furthermore, N3AC scales to large scenarios and can be used in practical settings.

Acknowledgments

The work of University Carlos III of Madrid was supported by the H2020 5G-MoNArch project (Grant Agreement No. 761445) and the 5GCity project of the Spanish Ministry of Economy and Competitiveness (TEC2016-76795-C6-3-R). The work of NEC Laboratories Europe was supported by the 5G-Transformer project (Grant Agreement No. 761536).

References

[1] D. Bega, M. Gramaglia, C. J. Bernardos Cano, A. Banchs, and X. Costa-Perez, "Toward the network of the future: From enabling technologies to 5G concepts," *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 8, pp. 3205–3216, Jun. 2017.

[2] NGMN Alliance, "Description of Network Slicing Concept," Public Deliverable, 2016.

[3] ITU-R, "Minimum requirements related to technical performance for imt-2020 radio interface(s)," Technical Report, 2017.

[4] 3GPP, "Study on Architecture for Next Generation System," TS 23.799, v2.0.0, Dec. 2016.

[5] NGMN Alliance, "5G White Paper," White Paper, Feb. 2015.

[6] A. Gudipati, L. Li, and S. Katti, "RadioVisor: A Slicing Plane for Radio Access Networks," in *Proc. ACM HotSDN*, Chicago, Illinois, Aug. 2014, pp. 237–238.

[7] I. Malanchini, S. Valentin, and O. Aydin, "Generalized resource sharing for multiple operators in cellular wireless networks," in *Proc. IEEE IWCMC*, Nicosia, Cyprus, Aug. 2014, pp. 803–808.

[8] R. Mahindra, M. A. Khojastepour, H. Zhang, and S. Rangarajan, "Radio Access Network sharing in cellular networks," in *Proc. IEEE ICNP*, Göttingen, Germany, Oct. 2013, pp. 1–10.

[9] S. Rathinakumar and M. Marina, "GAVEL: Strategy-proof Ascending Bid Auction for Dynamic Licensed Shared Access," in *Proc. ACM MobiHoc*, Paderborn, Germany, Jul. 2016, pp. 121–130.

[10] X. Zhou, R. Li, T. Chen, and H. Zhang, "Network slicing as a service: enabling enterprises' own software-defined cellular networks," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 146–153, Jul. 2016.

[11] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5G network slice broker," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 32–39, Jul. 2016.

[12] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs, "Mobile traffic forecasting for maximizing 5G network slicing resource utilization," in *Proc. IEEE INFOCOM*, Atlanta, USA, May 2017, pp. 2583–2591.

[13] M. A. Habibi, B. Han, and H. D. Schotten, "Network Slicing in 5G Mobile Communication Architecture, Profit Modeling, and Challenges," arXiv:1707.00852, 2017.

[14] B. Han, S. Tayade, and H. D. Schotten, "Modeling profit of sliced 5G networks for advanced network resource management and slice implementation," in *Proc. IEEE ISCC*, Heraklion, Greece, Jul. 2017, pp. 576–581.

[15] B. Han, D. Feng, L. Ji, and H. D. Schotten, "A Profit-Maximizing Strategy of Network Resource Management for 5G Tenant Slices," arXiv:1709.09229, 2017.

[16] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," arXiv:1312.5602, 2013.

[17] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[18] D. Silver, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[19] A. Graves *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, Oct. 2016.

[20] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. International Conference on Machine Learning*, New York, USA, Jun. 2016, pp. 1928–1937.

[21] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Proc. NIPS*, Barcelona, Spain, Dec. 2016, pp. 2154–2162.

[22] P. Mirowski *et al.*, "Learning to navigate in complex environments," arXiv:1611.03673, 2016.

[23] P. H. Su *et al.*, "On-line active reward learning for policy optimisation in spoken dialogue systems," arXiv:1605.07669, 2016.

[24] B. Malila, O. Falowo, and N. Ventura, "Intelligent NLOS Backhaul for 5G Small Cells," *IEEE Commun. Lett.*, vol. 22, no. 1, pp. 189–192, Jan. 2018.

[25] H. Tong and T. X. Brown, "Adaptive call admission control under quality of service constraints: a reinforcement learning solution," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 2, pp. 209–221, Feb. 2000.

[26] C. Jiang, H. Zhang, Y. Ren, Z. Han, K. C. Chen, and L. Hanzo, "Machine Learning Paradigms for Next-Generation Wireless Networks," *IEEE Wireless Commun.*, vol. 24, no. 2, pp. 98–105, Dec. 2017.

[27] A. P. Iyer, L. E. Li, and I. Stoica, "Automating Diagnosis of Cellular Radio Access Network Problems," in *Proc. ACM MobiCom*, Salt Lake City, USA, Oct. 2017, pp. 79–87.

[28] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Trans. on Cogn. Commun. Netw.*, vol. 3, no. 4, pp. 563–575, Dec. 2017.

[29] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for wireless resource management," arXiv:1705.09412, 2017.

[30] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez, "Optimising 5G Infrastructure Markets: The Business of Network Slicing," in *Proc. IEEE INFOCOM*, Atlanta, USA, May 2017, pp. 910–918.

[31] Will Townsend, "Will 5G Change Lives? Infrastructure Providers, Carriers, Even Dell EMC And HPE Are Betting It Will," Forbes, 2017. [Online]. Available: https://www.forbes.com/sites/moorinsights/2017/06/21/will-5g-change-lives-infrastructure-providers-carriers-even-dell-emc-and-hpe-are-betting-it-will/

[32] S. Shenker, "Fundamental design issues for the future Internet," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 7, pp. 1176–1188, Sep. 1995.

[33] M. Berenson, D. Levine, K. A. Szabat, and T. C. Krehbiel, *Basic business statistics: Concepts and applications*. Pearson higher education AU, 2014.

[34] J. L. Devore, *Probability and Statistics for Engineering and the Sciences*. Cengage learning, 2011.

[35] ITU-R, "Guidelines for evaluation of radio interface technologies for imt-advanced," Report ITU-R M.2135-1, 2016.

[36] R. Bellman, "A Markovian decision process," DTIC, Tech. Rep., 1957.

[37] R. Howard, *Dynamic Programming and Markov Processes*. Technology Press-Wiley, 1960.

[38] S. Lippman, "Applying a New Device in the Optimization of Exponential Queuing Systems," *Operation Research*, vol. 23, no. 4, pp. 687–710, Aug. 1975.

[39] H. Tijms, *A First Course in Stochastic Models*. J. Wiley & Sons, 2003.

[40] C. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[41] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial Intelligence: a Modern Approach*. Prentice hall Upper Saddle River, 2003, vol. 2, no. 9.

[42] F. S. Melo and M. I. Ribeiro, "Q-learning with linear function approximation," in *Proc. COLT*, San Diego, USA, Jun. 2007, pp. 308–322.

[43] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016.

[44] F. Chollet, *Deep learning with Python*. Manning Publications, 2018.

[45] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989.

[46] T. Tieleman and G. Hinton, "RMSprop gradient optimization," http://www.cs.toronto.edu/tijmen/csc321/slides/lecture_slides_lec6.pdf, 2014.

[47] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[48] 3GPP, "Technical specification group services ans system aspects; policy and charging control architecture," TS 23.203, v15.1.0, Dec. 2017.