Deep Reinforcement Learning Based Flexible Preamble Allocation for RAN Slicing in 5G Networks

Ahmet Melih Gedikli*†, Mehmet Koseoglu*, Sevil Sen*

*WISE Lab of Hacettepe University; †Corresponding Author

Abstract

One of the most difficult challenges in radio access network slicing occurs in the connection establishment phase where multiple devices use a common random access channel in order to gain access to the network. It is now very well known that random access channel congestion is a serious issue in case of sporadic arrival of machine-to-machine nodes and may result in a significant delay for all nodes. Hence, random access channel resources are also needed to be allocated to different services to enable random access network slicing. In the random access channel procedure, the nodes transmit a selected preamble from a predefined set of preambles. If multiple nodes transmit the same preamble at the same random access channel opportunity, a collision occurs at the eNodeB. To isolate the one service class from others during this phase, one approach is to allocate different preamble subsets to different service classes. This research proposes an adaptive preamble subset allocation method using deep reinforcement learning. The proposed method can distribute preambles to different service classes according to their priority providing virtual isolation for service classes. The results indicate that the proposed mechanism can quickly adapt the preamble allocation according to the changing traffic demands of service classes.

Index Terms

Deep Reinforcement Learning, Preamble Allocation, Network Slicing, 5G, RAN, M2M

I. INTRODUCTION

Internet of Things (IoT) is becoming more widespread in a wide range of usage areas such as smart grids, personal communications, controlling traffic flow on roads, smart driving, and smart healthcare [1]. It was recently reported that a total of 75 billion IoT devices are expected to be in operation globally by 2025 [2]. Communication among these IoT devices is named machine-to-machine (M2M) communication and the connections resulting from M2M communication are expected to be more than half of the global connected devices and connections by 2022 [3]. Hence, M2M communication is one of the main drivers of the evolution from 4G to 5G. While nearly half of these connections have resulted from home applications, the number of connections resulting from connected work and connected cities applications is showing an increasing trend in recent years.

M2M communication is the main driver of fully-automated production lines and connected cities. It has different characteristics than human-to-human communication (H2H) who has been the main driver of Long-Term Evolution (LTE). In H2H, most of the traffic is carried on the downlink, sessions are longer and less frequent. On the other hand, M2M devices mostly utilize the uplink and sessions are shorter and more frequent. For example, a smart meter may wake up, send its data, and then immediately go back to sleep, thus, just for a short data transmission, an uplink physical resource allocation request has to be made. In short, these devices generally require low bandwidth, however, due to the large number of them and their wake-up nature, the up-link physical resource allocation for them are quite different from H2H [4].

5G has not only pledged to provide services for H2H and M2M communication but it promised more, yet, up-to LTE it was sufficient. Therefore, LTE and 3G generally divided the service characteristics into two as M2M and H2H. However, with the 5G concept at least 3 but up to 5 different service characteristics are defined [5], [6]. The priorities of these service types may differ from each other. For example, while a remote surgery service should have the highest

priority since it must be ultra reliable and must provide ultra low latency [7], an IoT service that collects sensor data can have the lowest priority in a network. Thus, the increasing number of devices and services with changing characteristics cause us to review the bottlenecks of the communication.

One of the bottlenecks in Radio Access Network (RAN) communication is the Physical Random Access Channel (PRACH) allocation procedure [8]. This RACH opportunity period is called Random Access Opportunity (RAO) and the whole procedure is maintained by base stations. The limited number of orthogonal distinguishable signals (preambles) arise the need of dynamic distribution of them to the services. In both LTE and 5G, there are 54 PRACH preambles each one selected by devices randomly and transmitted for channel allocation requests. If multiple nodes transmit the same preamble at the same RAO, a collision occurs at the base station. To isolate one service class from others during this phase, one approach is to divide RACH resources to different service classes. Static allocation of those resources, however, may result in inefficiencies when the traffic generated by each service changes significantly over time. For example, in the event of a power outage and restore, all IoT devices in the network may try to reconnect to the network at the same time since they wake up at the same time. There may also be temporary dense networks, such as networks of thousands of spectators gathered in a stadium for a football match [9], [10]. Moreover, since 5G requires more heterogeneous and smaller cells, the number of users in each cell could vary greatly and dynamically due to user mobility [11]. All these example scenarios highlight the need for dynamic allocation mechanism for up-link resources, since the contention on resources may cause delays and collisions. Hence, slicing the resource into dynamic groups is more suitable to be able to keep the delay and collision probabilities around the desired levels.

Dynamic slicing of resources of the RACH preambles is a challenging problem since the eNodeB has limited information about the number of nodes waiting for different services. Previous studies on service differentiation in the RACH context either focused on heuristics

methods or analytical models. This research focuses on a reinforcement learning (RL) approach for this complex and dynamic problem. RL methods have the advantage of being able to operate without exact system models. RL-based approaches have become very attractive for solving problems in networking and communication due to their very characteristics such as autonomous decision making and obtaining the best policy with the minimum information for network entities [12]. Moreover, it is stated in the literature [13] that DRL-based network resource allocation schemes outperform conventional resource allocation schemes. Also, they have the ability to adapt to changes in an environment and to give a suitable action automatically with the help of their reward mechanism [14]. Furthermore, where a large amount of data may not be available to train complex deep learning models, RL might provide a practical solution by learning from a network environment [15]. These features make RL very attractive for dynamic preamble allocation and suitable for changing service requirements over time. Therefore, in this study, the use of reinforcement learning for automatically RAN slicing is investigated.

An adaptive preamble subset allocation method is proposed in this research using deep reinforcement learning (DRL) in order to increase QoS by preserving the balance between service types such that the PRACH could be shared with respect to priorities of services and traffic in the network. The proposed method can distribute preambles to different service classes according to their priority providing virtual isolation of service classes. The results indicate that the proposed mechanism can adapt the preamble allocation according to the changing traffic demands of service classes quickly. Besides, several reward functions are proposed for the RL algorithm and mathematically analyze the behavior of these functions. Simulations indicate that the behavior of the proposed RL mechanism closely follows the proposed mathematical analyses. The contribution of this current study could be summarized as follows:

- DRL is proposed in order to solve the preamble allocation problem in 5G. To the best of our knowledge, this is the first study in the literature that explores the use of DRL for allocating preambles dynamically based on the priorities of services, and traffic in the network.

- The proposed method is evaluated thoroughly on varying simulated networks where traffic could increase or decrease suddenly or constantly. Since at least three services are defined in 5G and this number could increase up to 5 services [6], network scenarios with 2, 3, 5 slices are considered in the experiments. Please note that, evaluations are generally carried out for 2 or 3 slices in the literature, however 5 slices are also taken into account here.
- The experimental results show that the proposed approach successfully allocates preambles dynamically and improves the access probability of slices to the PRACH based on their priority levels. Moreover, it can adapt very quickly to changes in the network. The proposed approach is compared with a recent approach [16] in the literature and shown to produce results comparable to [16].
- Three reward functions, namely Successful Preambles Reward Function (SRF), Proportional Reward Function (PRF) and Collision Penalizing Reward (CRF) are proposed for the reinforcement learning and mathematically analyzed. CRF has been shown to be the most successful, since it penalizes collisions.

The rest of the paper is organized in the following manner. In Section 2, related work is introduced. The system model is given in Section 3. In Section 4, RL-based preamble grouping is explained. The analysis of the proposed 3 reward functions of RL is carried out in Section 5. The evaluation and results are discussed in Section 6. The discussion on results, limitations and possible future work is given in Section 7. Finally, Section 8 is devoted to concluding remarks.

II. Related Work

In this section, studies on RACH congestion control, reinforcement learning in wireless networks and RAN slicing in the literature are reviewed.

The most well known proposed congestion control technique, Access Class Barring (ACB), prevents user equipment (UE) to access RACH resources when there is congestion in the network. The eNodeB periodically distributes a probability factor and barring time. Then, the UEs select

a random number. If the selected random number is lower than the probability factor, the UE is permitted to access RACH resources. Otherwise, UE waits a random amount of time based on the barring time distributed by the eNodeB.

Since ACB does not consider multiple service classes, Extended Access Barring (EAB) is proposed as an extension of ACB for service differentiation among different M2M devices. In this scheme, the main aim is to reduce the number of collisions among delay-tolerant M2M devices, so generally, the delay is higher than ACB [17]. There are also few studies to incorporate service differentiation into random access channel by distributing multiple barring factors to different service classes in combination with different techniques [18], [19], [20]. While the study in [18] bases their strategy on three (low, medium and high) ACB factors and names it multiple ACB (MACB), the authors in [19] proposes a method for delay-sensitive devices by combining ACB and RACH partitioning. Moreover, they assert the work in [18] does not realize partition of RACH and it ignores the quality of delay-tolerant devices. Another multiple ACB method [20] introduces a priority-based access class barring (PACB) algorithm that performs higher throughput when compared to MACB and some other relevant techniques.

There are also a limited number of studies proposing to prioritize RACH access through preamble separation [21], [22], [23], [24]. Most of these studies propose a fixed preamble separation configuration which divides the preambles into two groups. For example, the total set of preambles are divided into two groups in [21], [23], one group is exclusively reserved for H2H devices and the other group is either reserved for M2M or can be used by both M2M and H2H devices. The authors in [24] proposed to partition the preambles into 3 groups by introducing a higher priority traffic type for the smart grid. However, the groupings are not adaptive. Another static approach where PRACH slots are assigned to different service classes is also proposed in [22]. The issue with these works is that the preamble groupings are static which may be inadequate to respond to changes in the traffic of different service classes.

There are few studies that investigate the use of dynamic preamble subset allocation in the

literature [6][25][16]. The main difference between the approach in [6] and this research is that the authors use heuristic algorithms for subset allocation whereas reinforcement learning is used in this research. Besides they consider a setup where nodes transmit multiple consecutive preambles whereas this research considers single preamble transmission as in the LTE standard. Another dynamic preamble separation method which is combined with binary exponential back-off with respect to three different priority classes is proposed in [26]. A load adaptive dynamic preamble allocation method called LATMAPA [25] is proposed for prioritizing in the context of 5G Random Access. However, it proposes a solution for only 2 slices which are delay tolerant and delay intolerant.

Very recently, an online control method for dynamic preamble distribution over prioritized preamble groups is proposed in [16]. At first, the number of active devices in each priority is estimated in a recursive Bayesian way. Then, together with this estimation, a heuristic novel algorithm that distributes the preambles over the service groups with respect to their priority levels in a dynamic manner is applied. Finally, they extend their approach with ACB. In terms of separating the preambles and prioritizing the services, [16] shows similarity to the proposed approach in this current study. For example, while they classify the devices into different priorities by assigning priority weights, priority coefficients are employed in this study. Furthermore, both studies support any number of services. Therefore, the approach in this current study is compared with mentioned approach by using a scenario given in [16]. Moreover, the ideal algorithm given as the baseline in [16] is used in the experiments.

RL-based approaches have recently been proposed for several problems in radio access networks. In general, these works are focused on optimizing the allocation of resources and increasing the quality of service (QoS). One relevant study on RACH proposes ACB barring rate adaptation using Q-learning to increase the success probability of M2M communications with low impact on H2H communication [27]. Another study employs RL to optimize the joint allocation of fiber and radio resources for Cloud RANs. The authors note that RL improves performance

with respect to genetic and tabu search algorithms [28]. There are also several studies about mobile edge computing (MEC) to optimize the allocation of network and computing resources [29], [30], [31]. The study in [29] proposes a DRL solution to allocate the networking resources adaptively to reduce service time for users under the diversified MEC environment. The authors in [30] exploit Deep Q-Network (DQN) in order to solve computational offloading problem in multi-user MEC systems and to avoid the curse of dimensionality. A similar work [31] focuses vehicle-assisted computational offloading using DQN in a similar manner. Another application of RL is to improve the energy efficiency of heterogeneous networks through user scheduling and resource allocation [32].

A DRL-based random access optimization method is proposed in [33] in order to dynamically adjust the ACB parameter. The prioritization is addressed using different ACB parameters for service types, however, only three types are sampled. Furthermore, the types are assumed to have default traffic characteristics. Hence, the diversity of service types and flexible characteristics of traffic are not addressed in [33]. To the best of our knowledge, RL has not been applied to the problem of preamble subset allocation for network slicing. The advantage of using RL with respect to approaches based on heuristic methods is that there is no assumption on the traffic model in RL. RL can learn to maximize channel performance regardless of the traffic distribution.

Network slicing is also gaining popularity as a key enabler technology for the 5G vision and RAN slicing is one of its main components. The main aim of RAN slicing is to enable dynamic on-demand allocation of radio resources among multiple services. A run time slicing method that isolates RAN slices and a set of algorithms in order to partition inter-slice resources are proposed in [34]. Another RAN slicing method allocates radio resources between enhanced mobile broadband and vehicle-to-everything services using RL combined with a heuristic algorithm to maximize utilization [35]. A two-layer scheduler approach is proposed in [36] in order to manage balance between isolation and efficiency, where the first layer is used to allocate resources to

each slice and the second one is used to allocate resources for each user. Another RL-based slice admission controller which makes its decisions based on resource availability is also proposed in [37].

III. System Model

In a typical LTE configuration, 64 preambles are used in the random access procedure. While 10 preambles are reserved for contention-free access, 54 preambles are allocated for contention-based access [38]. Hence, it is assumed that the total number of available preambles is 54 and it is assumed that a system where there are N slices with different service requirements. Therefore, 54 preambles used in the random access procedure are divided into N different groups. Each preamble group is assigned to a slice and a UE is only allowed to select a preamble from the group of its service class. Hence, each slice is isolated from the rest of the slices in the sense that the preamble transmitted by a UE can only collide with the other UEs in the same slice. There are studies in the literature that aim to model and analyze the PRACH considering the factors like path loss, low transmission power of nodes, deafness of nodes [39], [40], [41]. In this research, these factors are ignored and the system is assumed to have an ideal channel where preambles losses only occur due to collisions.

In the model, in each RAO, the nodes which have been collided will remain backlogged in the system and will attempt to transmit a new preamble in the next round. The preamble groupings are adaptive and it is assumed that these groupings are announced by the eNodeB before each RAO. Hence, the backlogged nodes will respect the new preamble groupings announced by the eNodeB in the next round. As well as the backlogged nodes, newly arriving nodes will also be attempting to transmit in the next slot.

The notations used throughout this study and how they are calculated are listed in Table I. Here, M represents the total available contention based PRACH orthogonal distinguishable signal (preamble) count in each RAO period (54). N shows the number of slices in the network. W is

the maximum number of allowed preamble transmission attempts, if it is reached, the preamble requests are removed from the backlog and evaluated as dropped. m_j represents the number of preambles reserved for slice j and it is announced by eNodeB periodically. At the end of each RAO period, the contention on the PRACH is resolved, and the number of collided preambles for slice j (c_j), number of successful preambles for slice j (s_j) and number of unused preambles for slice j (u_j) are obtained in that period. These values with priority coefficients (k_j) are used in the reward functions, hence the reward value for slice j (r_j) could be calculated. n_j here represents the number new arrival preamble requests for slice j. n_j values are determined for each scenario in testing and these requests are added to the testing backlogs (b_j^t) in every RAO period ($10ms$). On the other hand, n_j values are randomly generated in training and, the requests are added to the training backlogs (b_j^{tr}). Finally, λ_j, which is widely used in the literature, represents the normalized arrival rate for slice j.

A sample RAO is shown in Fig. 1. In the figure, the eNodeB has announced that 9 preambles are reserved for slice-1, 12 preambles are reserved for slice-2, etc. There were 22 UEs transmitted a preamble from slice-1 and four of them became successful as they are the ones that did not experience a preamble collision. Hence, the number of backlogged UEs is reduced to 18 after the RAO. Slicing the preamble resources to services given in Fig. 1 as an example can be called as preamble grouping.

IV. Reinforcement Learning Based Preamble Grouping

In recent years, there has been a tremendous increase in the use of deep reinforcement learning in order to solve highly-complex problems. Here, it is used for the adaptive selection of the preamble groups. In the following parts, the proposed RL-based preamble grouping technique is explained in detail. Firstly, the focused problem in this research is formulated in the RL context and, then, the RL algorithm that is used to solve the problem at hand is described. Further, the training procedure used in the system model is clarified.

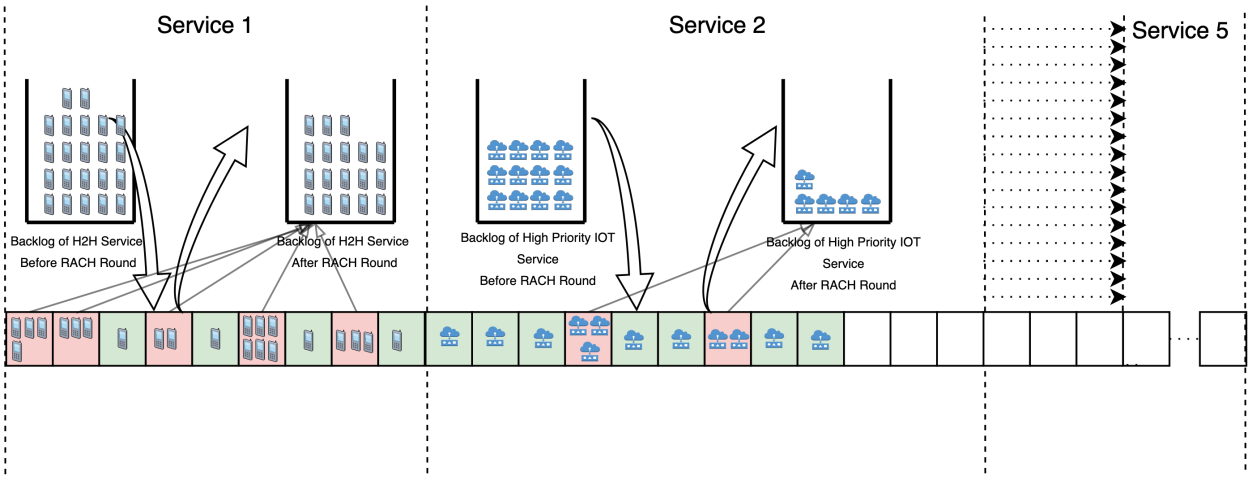


Fig. 1: RACH Process of eNodeB Simulator

Notation	Explanation	Value	Formula
M	Total number of available preambles for contention-based RACH	54	-
N	Number of slices in the available system model	-	-
W	Maximum number of allowed preamble transmission attempt	10	-
m_j	Number of preambles reserved to slice j for the RAO	-	-
c_j	Number of collided preambles in the RAO period for slice j	-	-
s_j	Number of successful preambles in the RAO period for slice j	-	-
u_j	Number of unused preambles in the RAO period for slice j	-	$m_j - c_j - s_j$
n_j	Number of new arrival preamble requests in the RAO period for slice j	-	-
b_j^{tr}	Number of backlogged preamble requests waiting for the next RAO period for slice j used in the training	-	-
b_j^{t}	Number of backlogged preamble requests waiting for the next RAO period for slice j used in the testing	-	-
b_j	Number of backlogged nodes waiting for the next RAO period for slice j	-	-
λ_j	Normalized arrival rate for slice j	-	$\lambda_j = n_j/M$
k_j	Reward or priority coefficient for successful transmission of a preamble request for slice j	j	-
r_j	Collected reward value in a RAO period for slice j	-	$s_j k_j - c_j$ for CRF $k_j \frac{s_j}{m_j - u_j}$ for PRF

TABLE I: Notations

A. Problem Representation

In the RL context, the *agent* takes *action*s according to its *policy* to interact with the environment. The policy is a function that maps the environment's *state* into agent's actions. As a result of the actions, the *environment* returns a *reward* and the state of the environment changes. Observing this new state, the agent chooses a new action and the interaction goes on. During this interaction, the agent refines its policy based on the collected rewards to reach an optimal policy that maximizes the expected cumulative reward.

In this problem, the learning agent is the eNodeB and the action in RL corresponds to the preamble groupings for each slice. More specifically, the eNodeB decides on the number of preambles for each slice such that their sum will be 54. Before each random access opportunity, the groupings is distributed in System Information Grouping (SIB) along with other RACH parameters. The grouping information includes slice id, the number of preambles reserved for this slice and the first preamble index of the group. The environment is the random access channel where a number of UEs are waiting to transmit a preamble. Based on the announced preamble groupings, each UE transmits one of the preambles from their respective preamble group. Each node randomly selects its preamble and the number of UEs transmitting is not known to the eNodeB. Moreover, the number of UEs waiting to transmit a preamble changes as new UEs arrive at the channel over time. Hence, the environment is stochastic.

B. Training

Fig. 2 illustrates the interaction between the agent and the simulation environment. There are two neural networks in policy optimization based DRL. The policy network plays the actor role and produces the action space to the environment. Meanwhile, the value network produces state values by assigning each state a score calculated using the sum of rewards and the state of the previous round so that the states with higher rewards have more value in the network. Actions which results in a better state is preferred since it produces a higher reward.

Here, a scenario where the eNodeB policy is trained offline but tested with actual traffic after deployment is assumed. RL algorithms start by exploring the action space. At the initial state, the first actions are mostly random. Moreover, their initial behavior may be very far from optimum. Hence, RL algorithms take a considerable amount of time in order to converge, especially if the training process starts from scratch.

The most crucial aspect in the RL framework is the reward function. The reward function defines the objective of the problem. In this study, the reward is defined as a function of the number of successful preambles and collided preambles for each service class. For each service class j, a reward function is calculated as follows:

$$R_j(t) = s_j k_j - c_j \tag{1}$$

where s_j and c_j are the number of successful and collided preambles from class j, respectively. k_j is defined as the reward or priority coefficient for class j which is used to prioritize among different slices. Higher the priority of a slice, the higher its reward coefficient should be. Then, the total reward is calculated as the sum of rewards for all service classes.

$$R(t) = \sum_j R_j(t). \tag{2}$$

The selection of this reward function is discussed in detail in Sec. V.

The state of the environment is the number of UEs from each service class waiting to transmit a preamble. However, this information is not available to the eNodeB, hence, it uses the most recent observation about the channel in terms of successful and collided preambles. The state is defined as

$$S(t) = \bigcup_j S_j(t) \tag{3}$$

where

$$S_j(t) = \{c_j, s_j\}. \tag{4}$$

The policy of the agent maps the state to actions and the RL algorithm optimizes the policy. Here, trust region policy optimization (TRPO) which has recently achieved state-of-the-art results in various AI tasks is employed in RL [42]. In the TRPO algorithm, the two dependent neural networks shown in Fig. 2 are for approximating the value function and the policy function. From the TRPO perspective, the value network is used to calculate the expected future reward from the current policy. The policy network is used for approximating the policy function. This network gets the current state of the environment as input and produces the probability distributions of each action using the value function. Then, the algorithm selects the most probable action [43].

In this study, an open implementation of TRPO [44] is used. This framework uses ADAM optimizer for both neural networks due to its fast convergence [45], [46]. The default values of the framework [44] is used in the experiments. Only the *number of episodes* and *batch size* parameters are modified in these settings. *Batch size* is set to 1 in order to increase the frequency of policy evaluation and the *number of episodes* is set to 100,000 in order to increase the training time. The simulation parameters and their values used in the experiments are given in Table II. The details of other parameters used in the framework can be found in [44].

Over-fitting is a crucial problem to avoid in all types of machine learning approaches. RL algorithms are also prone to over-fitting to the environment and the trained policies may not generalize very well to newer environments. In the scenario it is considered that the system could over-fit to the traffic arrival distributions used in training. In order to avoid over-fitting, the policy is trained by using a randomly generated traffic model. In the model, the generator redistributes the number of preamble requests randomly in 5 seconds periods. Recalling M is the total number of available preamble requests, it is shown in [47] that the number of successful transmissions in a round yields to M/e if the number of arrival of preamble requests is equal to M in a round. In the next round, the number of backlogged request is $M-M/e$ if no new request is made. As a result, if the arrival request count is greater than M/e constantly, the number of backlogged preamble request increases in time and the RACH becomes unstable irreversibly.

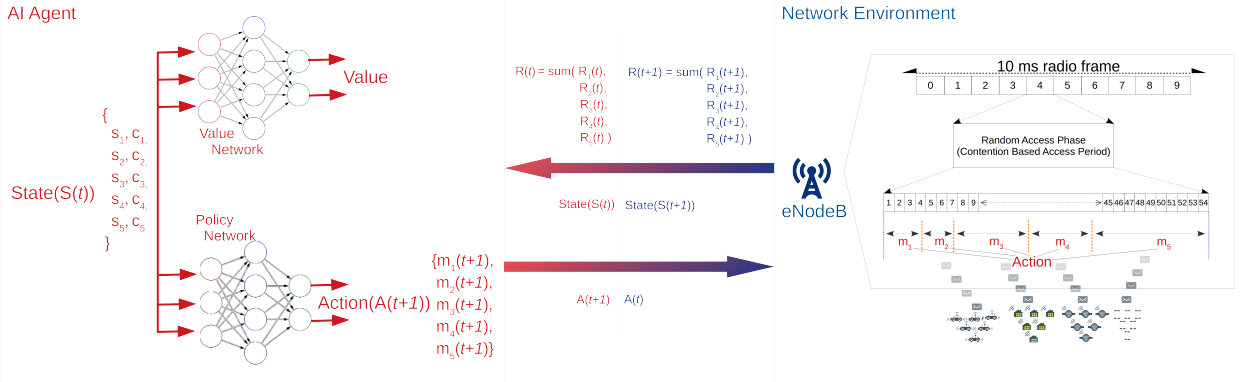
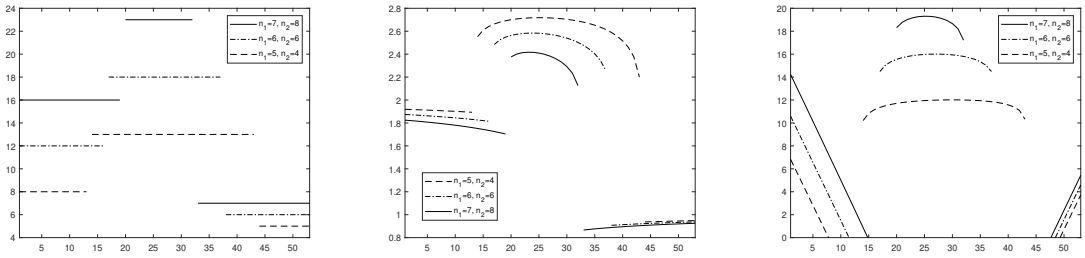


Fig. 2: Interaction Between DRL Agent and Network Environment



(a) (b) (c)

Fig. 3: Change in the (a) SRF (b) *PRF* (c) *CRF* as m_1 changes for $k_1 = 1$ and $k_2 = 2$ for different arrival rates.

Therefore, the number of preamble requests are distributed to slices in a way that their average sum yields M/e in the long run. Since this given randomly generated traffic does not show any patterns as might be in a real traffic, it is less prone to over-fitting. Therefore, it is preferred to be used in training. Then the generated model is evaluated in testing. The performance of the model in varying network scenarios also supports that the model does not over-fit.

	Networks Scenarios		
	2-slice	**3-slice**	**5-slice**
Parameters	**Value**	**Value**	**Value**
N	2	3	5
k_j	$k_1 = 1$ $k_2 = 2$	$k_1 = 1$ $k_2 = 2$ $k_3 = 3$	$k_1 = 1$ $k_2 = 2$ $k_3 = 3$ $k_4 = 4$ $k_5 = 5$
Reward Functions	CRF PRF	CRF PRF	CRF
Discount Factor	0.9		
Number of Episodes	100,000		
Batch Size	1		
GAE (λ)	0.98		
D_{KL} target	0.003		
Size of the First Hidden Layer	10		
Initial Policy Log-variance	-1		

TABLE II: Simulation Parameters

V. Reward Function Analysis

Reward function is the objective function used for evaluating the model that has to be maximized over successive steps. The choice of the reward function is crucial, since a poor selection may result in sub-optimal allocation of resources. In this part, analytical expressions are derived for several possible reward functions for a two-slice scenario. Then, the behavior of these functions is evaluated. Finally, these analytical results are compared with the experimental results presented in Sec. VI.

Let m_1 and m_2 are the number of preambles assigned to each slice such that $M = m_1 + m_2$. Let n_1 and n_2 are the average number of arrivals per RAO for each slice. First, consider the case where $n_1 < m_1/e$ and $n_2 < m_2/e$. In this case, all incoming traffic can be supported because the capacity of the RACH with m preambles is approximated as m/e [48].

When there are b_j nodes randomly selecting preambles from a set with size m_j, the expected number of successful preambles is given by [49]:

$$s_j = b_j(1 - m_j^{-1})^{b_j - 1}. \tag{5}$$

Similarly, the expected number of unused preambles can be found as [49]:

$$u_j = m_j(1 - m_j^{-1})^{b_j}. \tag{6}$$

The remaining preambles are collided:

$$c_j = m_j - s_j - u_j. \tag{7}$$

When the RACH is stable, the expected number of successful preambles must be equal to the average number of arrivals in the long run, i.e., $s_j = n_j$. To satisfy this equality, the number of attempting nodes at each slot needs to increase to a level $b_j > n_j$. This value can be found by solving (5) which gives $b_j = -m_j W(-n_j/m_j)$ where $W()$ is the principal branch of the Lambert W function [48]. Using the approximation $(1 - 1/x)^n \approx e^{-n/x}$, u and c can be further simplified as

$$u_j \approx m_j e^{-b_j/m_j} = m_j e^{W(-n_j/m_j)} \tag{8}$$

and

$$c_j \approx m_j(1 - e^{W(-n_j/m_j)}) - n_j. \tag{9}$$

A. Successful preambles reward function (SRF)

The most intuitive and simple reward function is the number of successfully transmitted preambles at each RAO. In this case, the reward for a slice is $r_j = s_j$ and the total reward can be defined as the weighted sum of rewards of each slice:

$$r_{succ} = k_1 r_1 + k_2 r_2 = k_1 s_1 + k_2 s_2 \tag{10}$$

where k_1 and k_2 are used to prioritize different slices. This reward function, however, has some undesirable properties. For any choice of m_1 and m_2 which satisfies $n_1 < m_1/e$ and $n_2 < m_2/e$,

RACH is stable and the number of successful preambles equal to the number of arrivals, $s_j = n_j$. Hence, the reward function is flat in this region. For $n_1 > m_1/e$ and $n_2 < m_2/e$, all preambles will be unsuccessful for slice-1 and all traffic can be supported in slice-2, again leading to a flat reward function. In general, the reward function can be written as:

$$r_{succ} = \begin{cases} k_1 n_1 + k_2 n_2 & \text{if } n_1 < m_1/e, n_2 < m_2/e \\ k_1 n_1 & \text{if } n_1 < m_1/e, n_2 > m_2/e \\ k_2 n_2 & \text{if } n_1 > m_1/e, n_2 < m_2/e \\ 0 & \text{if } n_1 > m_1/e, n_2 > m_2/e \end{cases} \tag{11}$$

The behavior of this reward function is illustrated in Fig. 3a for $n_1 = n_2 = 5$. As the reward function is flat for the stable region, the RL agent does not have any incentive to change the groupings as long as the RACH is stable. This behaviour may bring the system very close to instability when the number of preambles reserved for one slice is barely sufficient for the traffic of that slice. In that case, a slight increase in traffic may result in serious congestion which could have been easily avoided if the agent used a more robust allocation of preambles. Hence, a reward function is needed which would "steer" the system towards a better operating point.

B. Proportional reward function (PRF)

Another intuitive reward function for the RACH channel is the ratio of successful preambles among the number of transmitted preambles. Weighting each ratio corresponding to each slice, it is possible to differentiate preambles among slices according to their priority. Let k_1 and k_2 denote the priority coefficients of each slice. Then, for $n_1 < m_1/e$ and $n_2 < m_2/e$, total reward for a single RAO is given by

$$r_{pr} = r_1 + r_2 = k_1 \frac{s_1}{m_1 - u_1} + k_2 \frac{s_2}{m_2 - u_2} \tag{12}$$

For $n_1 > m_1/e$, there are not enough preambles for slice-1 which will result in a growing backlog and all preambles will collide resulting in $r_1 = 0$. Similarly, for $n_2 > m_2/e$, $r_2 = 0$. Hence,

$$r_{pr} = \begin{cases} \sum_{j=1}^{2} \frac{k_j n_j}{m_j(1-e^{W(-n_j/m_j)})} & \text{if } n_1 < m_1/e, n_2 < m_2/e \\ \frac{k_1 n_1}{m_1(1-e^{W(-n_1/m_1)})} & \text{if } n_1 < m_1/e, n_2 > m_2/e \\ \frac{k_2 n_2}{m_2(1-e^{W(-n_2/m_2)})} & \text{if } n_1 > m_1/e, n_2 < m_2/e \\ 0 & \text{if } n_1 > m_1/e, n_2 > m_2/e \end{cases} \tag{13}$$

For a given n_1, n_2, k_1 and k_2, this function can be numerically maximized to find the values of m_1 and m_2 such that $m_1+m_2 = m$. The behavior of the *PRF* is shown in Fig. 3b for $n_1 = n_2 = 5$. This reward function does not suffer from the problem mentioned in the previous part. The optimum values of m_1 for different priority coefficients is 25, 24 and 23 for $(n_1, n_2) = (5, 4)$, $(6, 6)$ and $(7, 8)$, respectively. Even the traffic of low-priority is higher than the traffic of high-priority, this reward function allocates a lower number of preambles to the lower-priority traffic.

C. Collision-penalizing reward function (CRF)

Another proposed reward function is defined as $r_j = s_j k_j - c_j$ where k_j is the reward coefficient. Hence, the total reward for $n_1 < m_1/e$ and $n_2 < m_2/e$ can be written as:

$$r_{cp} = r_1 + r_2 \tag{14}$$

$$= k_1 s_1 + k_2 s_2 - c_1 - c_2 \tag{15}$$

$$= (1 + k_1)n_1 - m_1(1 - e^{W(-n_1/m_1)}) + \tag{16}$$

$$(1 + k_2)n_2 - m_2(1 - e^{W(-n_2/m_2)}) \tag{17}$$

For $n_1 > m_1/e$ and $n_2 < m_1/e$, all preambles reserved for slice-1 will experience collisions,

$c_1 = m_1$ and $s_1 = 0$. Hence, the reward will be

$$r_{cp} = k_2 s_2 - m_1 - c_2 \tag{18}$$

$$= (1 + k_2) n_2 - m_1 - \tag{19}$$

$$m_2 (1 - e^{W(-n_2/m_2)}). \tag{20}$$

Similarly, for $n_2 < m_2/e$:

$$r_{cp} = k_1 s_1 - m_2 - c_1 \tag{21}$$

$$= (1 + k_1) n_1 - m_2 - \tag{22}$$

$$m_1 (1 - e^{W(-n_1/m_1)}) \tag{23}$$

For the case $n_1 > m_1/e$ and $n_2 > m_1/e$, all preambles will be collided:

$$r_{cp} = -m_1 - m_2. \tag{24}$$

The behavior of the *CRF* is shown in Fig. 3c. For $k_1 = 1$ and $k_2 = 2$, the optimum values of m_1 for different priority coefficients is 30, 27 and 25 for $(n_1, n_2) = (5, 4)$, $(6, 6)$ and $(7, 8)$, respectively.

VI. Evaluation and Results

In this section, the proposed technique is evaluated using simulations. Different models are trained for each reward function for different number of slices and evaluated. The models are trained by using only *CRF* and *PRF* reward functions. When *SRF* is used as the reward function, the agent cannot distribute preambles successfully and cannot handle the load of the higher priority slices effectively. As a result, a higher number of dropped preamble requests and longer waiting times are observed in comparison to *CRF* and *PRF*. Therefore, only the results of the models trained by using *CRF* and *PRF* are demonstrated in the results.

In the previous section, 3 different reward functions are analyzed for the 2-slice scenario since the complete state space is relatively small. However, with the context of 5G there are defined at

least 3 service types: extreme mobile broadBand (xMBB), massive machine-type communications (mMTC) and ultra-reliable machine-type communications (uMTC) [5]. This number, however, can increase up to 5 as delay-tolerant IoT, emergency, high priority IoT, human-to-human and mobile broadband [6]. On the other hand, increasing the number of slices also increases the state space drastically and finding the optimum policy through analysis gets intractable. Here, the simulations are also conducted for 3 and 5-slice scenarios besides the 2-slice scenario.

In the simulations, slices are prioritized using their reward coefficients, k_j. Slice numbers are used as the prioritization factor such that $k_j = j$. In other words, assuming there are N slices, the N^{th} slice has the highest priority. This prioritization scheme is employed in all simulations through this paper. Therefore, the slice numbers in all figures also indicate their priority. In addition, the first and lowest priority slice will be named as low-priority slice and the N^{th} and highest priority slice will be named as high-priority slice in the rest of the paper. The Figs. 4-8 are plotted using the average of collected data of 10 simulation runs.

A. Traffic Distribution

In the simulations, n_j preamble requests are generated every 10 ms and these requests are added to the message backlogs of each slice. While lower priority slices have a constant arrival rate during the whole simulation, the rate of the high-priority slice is increased every 200 ms. In the 2-slice case, low-priority slice has a constant normalized arrival rate of $\lambda_1 \approx 0.09$ where the normalized arrival rate of slice j is defined as $\lambda_j = n_j/M$. The normalized arrival rate for the high-priority slice starts from $\lambda_2 \approx 0.04$ and increases up to $\lambda_2 \approx 0.39$. In the 3-slice case, the low-priority slice has a constant rate of $\lambda_1 \approx 0.07$ and medium-priority slice also has a constant rate of $\lambda_2 \approx 0.09$. The normalized arrival rate of the high-priority slice starts from $\lambda_3 \approx 0.04$ and increases up to $\lambda_3 \approx 0.39$. The x-axis of the Figs. 4, 5, 6, 7 and 8 demonstrate this increase. Since the rate of increase of the high-priority slice is constant, the x-axis is proportional to the simulation time. Each simulation runs 7 seconds.

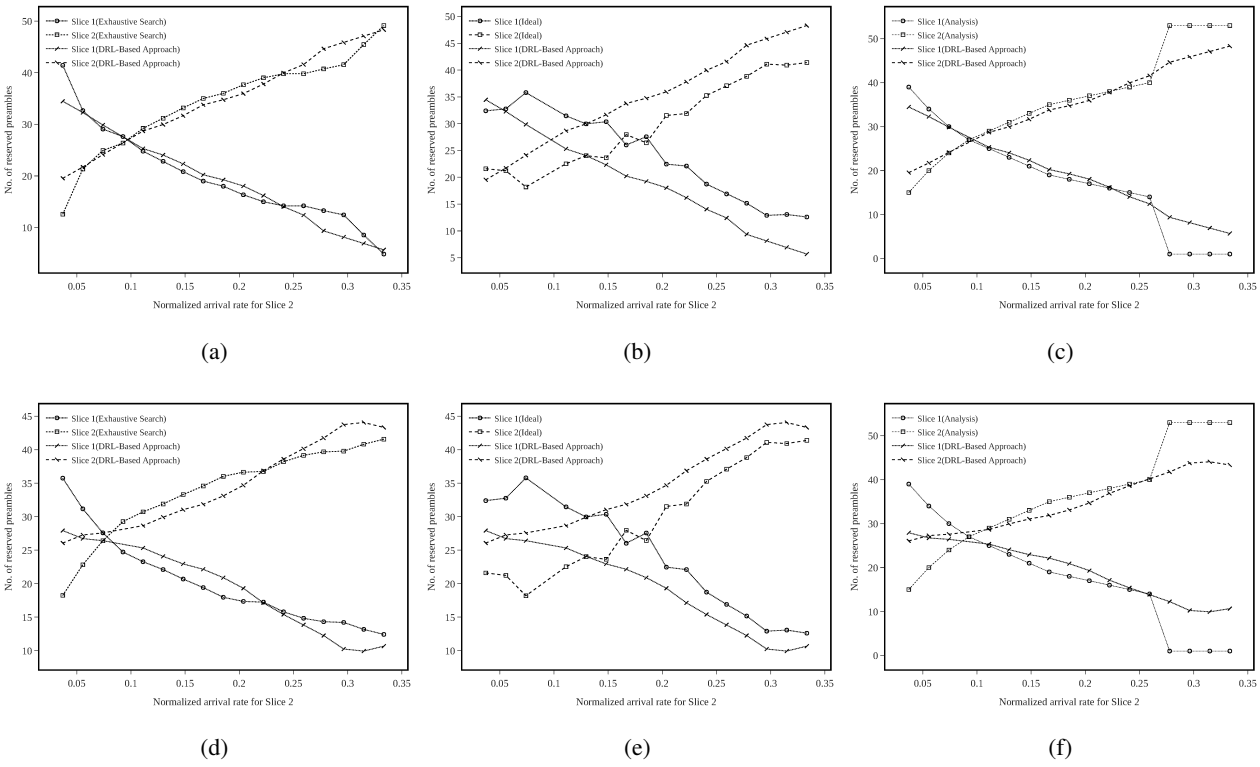
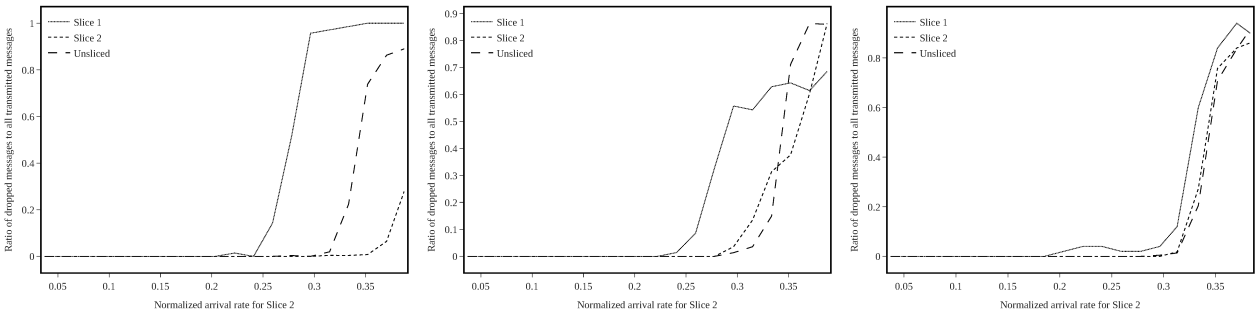


Fig. 4: Preamble allocations computed by the exhaustive search, *ideal algorithm* and mathematical analysis against the DRL-based approach in 2-slice scenario (a)-(b)-(c) respectively for the *CRF* and (d)-(e)-(f) for the *PRF* reward function.

B. Benchmarks for Simulations

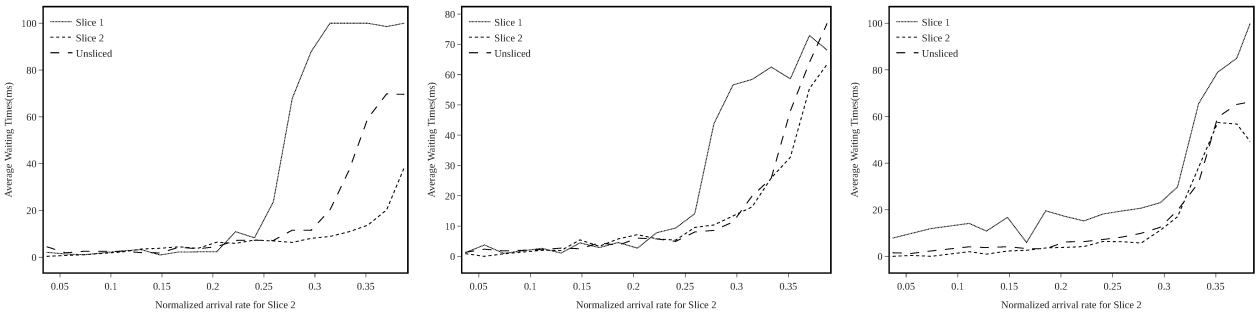
In order to evaluate how the proposed technique based on RL gets close to the optimal solution, it is compared with an exhaustive search technique in addition to the analyses presented in Sec. V. In exhaustive search, all possible preamble allocations for a given traffic load are searched through and the best preamble allocation is chosen. The same system parameters are used in the exhaustive search.

The performance of the proposed method is also compared with an unsliced scenario. In this scenario, there is only one slice in the system and all preamble requests belong to that slice. In



(a) (b) (c)

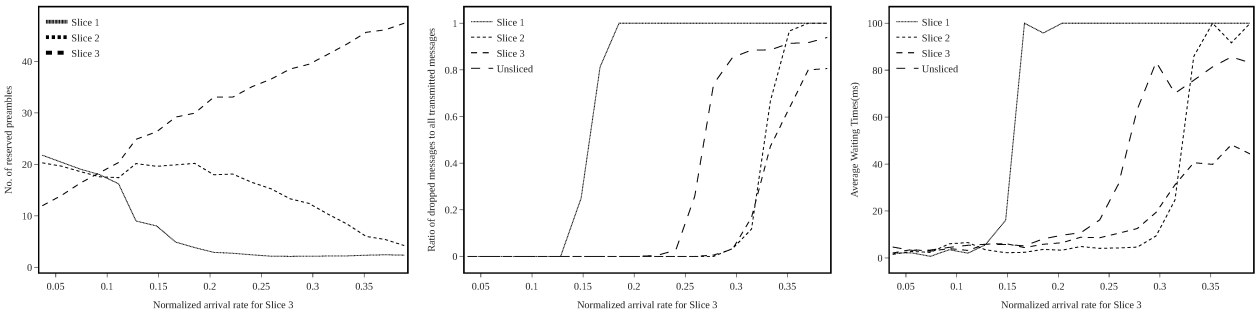
Fig. 5: The ratio of dropped messages to all transmitted messages for 2-slice scenario while the arrival rate of second slice consistently increased. (a)-(b)-(c) show the performance for *CRF*, *PRF* and *ideal* respectively.



(a) (b) (c)

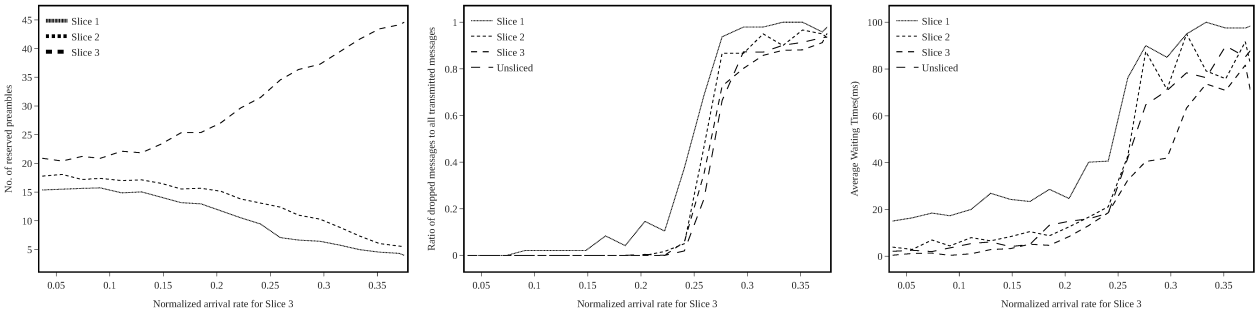
Fig. 6: The average waiting time for 2-slice scenario while the arrival rate of second slice consistently increased.(a)-(b)-(c) show the performance for the *CRF*, *PRF* and *ideal* respectively.

such a case, $n_{unsliced}$ is the sum of the number of new arrival preamble requests to all slices. As a result, the total number of new arrival requests does not change between the unsliced and sliced scenarios. That gives us a comparison basis between the performances of slices of the proposed model and the unsliced scenario.



(a) (b) (c)

Fig. 7: The performance of the proposed method to 3 network slices while the arrival rate of third slice consistently increased



(a) (b) (c)

Fig. 8: The performance of the ideal algorithm to 3 network slices while the arrival rate of third slice consistently increased

Finally, the *ideal algorithm* given as a baseline in [16] is used for comparison. The ideal algorithm assumes that the exact number of preamble request counts for a given RAO (b_j) is known for each priority. It is noted that the preamble request count for the next RAO for slice j, b_j, is found by adding the number of new arrivals (n_j) to the number of nodes which has not transmitted W times yet. Then, by using the Equation 25 in [16], the number of reserved

preambles to slice j (m_j) can be found in each RAO.

$$M = \sum_{j=1}^{N} m_j = \sum_{j=1}^{N} \frac{b_j}{-\ln(\frac{k_j}{\sum_{i=1}^{N} k_i}) - \ln(x)} \tag{25}$$

Here, x denotes the proportionality factor which satisfies the above equation. Assuming b_j values are known, x is also can be found by solving the equation and hence, m_j values are obtained.

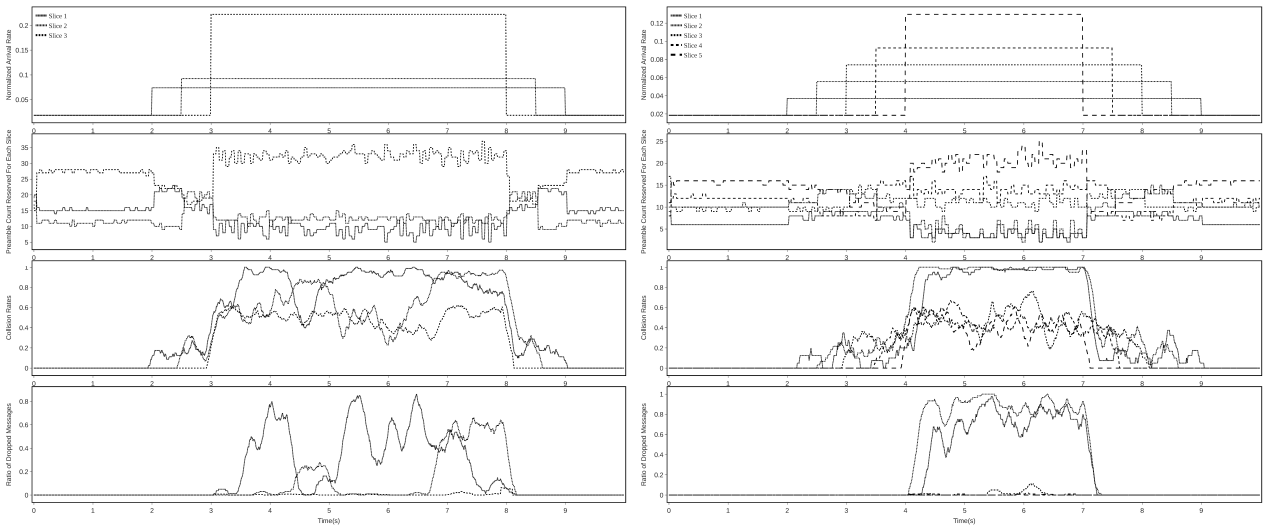
C. Simulation Results

Firstly, the proposed DRL-based approach is compared with the mathematical analysis given in Sec. V, the exhaustive search and the *ideal algorithm* for the scenario with 2 slices. Since the 2-slice scenario has a smaller state space, it is preferred for a baseline comparison. Fig. 4 plots the optimum number of reserved preambles for each slice that maximizes the reward functions against mathematical analysis, *ideal algorithm* and exhaustive search. The results show that the preamble allocations which yield maximum rewards for each technique behave similarly.

Fig. 5 demonstrates the ratio of dropped messages to all transmitted messages in a random access opportunity in the 2-slice scenario. The results for the reward functions *CRF*, *PRF* and *ideal algorithm* are given in Fig. 5a, Fig. 5b and Fig. 5c respectively. While the value of normalized arrival rate of the high-priority slice approaches to 0.3, the total load $n_1/M + n_2/M \approx 0.38$ exceeds the capacity $1/e \approx 0.37$. In both figures, the normalized arrival rate of the high-priority slice rises up to $n_2/M \approx 0.39$ and the total load in that maximum point passes $(n_1 + n_2)/M = 26/54 \approx 0.48$. In Fig. 5, while the high-priority slice outperforms the unsliced case for both reward functions, the low-priority slice also gets a considerable amount of reserved preambles. Fig. 5a points out that the high-priority slice using *CRF* can satisfy almost all preamble requests so that preamble request messages are nearly not dropped even when the total load passes $(n_1+n_2)/M = 25/54 \approx 0.46$. On the other hand, the unsliced plot has a sharp increase after passing the maximum normalized traffic load limit that the random access channel can handle ($1/e \approx 0.37$).

For the same scenario, Fig. 6 presents the average waiting time, which denotes how much time successful preamble requests waited in the message backlog on average. If there is no successful preamble request in the random access opportunity, the average waiting time can not be calculated for that random access opportunity. Nevertheless, to show the performance, the value is set to the maximum average waiting time which is $10 \times W$ ms. The average waiting time for the high-priority slice almost stays constant despite the increasing traffic rate. On the other hand, the average waiting time for the high-priority slice for *PRF* is higher in comparison to the waiting time of the same slice for *CRF*. Moreover, while *PRF* scheme distributes the preambles more closely to two slices as seen in Fig. 4, when the total load passes $(n_1 + n_2)/M = 21/54 \approx 0.39$, the dropped preamble requests from high-priority slice are dramatically increased. Therefore, the results which use only *CRF* are presented in the rest of the experiments.

The simulation results for the 3-slice scenario is given in Fig 7. As noted above, here the following performance metrics are evaluated for only the *CRF* reward function. At time $t = 0$, $n_1 = 5, n_2 = 4, n_3 = 2$ and the relation between number of reserved preambles is $m_1 > m_2 > m_3$. Since the total load is $((n_1 + n_2 + n_3)/54 \approx 0.20) < (1/e \approx 0.37)$, the proposed method allocates more preambles to $1st$ and $2nd$ slices as shown in Fig. 7a in order to relieve the traffic to lower priority slices when the high-priority slice easily handles the traffic. As n_3 increases, the proposed method aggressively increases the reserved preambles count for the $3rd$ slice in order to avoid dropping messages and long average waiting. The proposed method manages not to drop any preamble allocation requests from the $3rd$ slice even the total load passes $(n_1 + n_2 + n_3)/M = 26/54 \approx 0.48$. The average waiting time for the $3rd$ slice follows a horizontal line up to the normalized arrival rate of 0.3. After that, it slightly increases due to high load on the random access channel; yet, the proposed method is able to prevent a sharp increase. On the other hand, the unsliced curve shows a sharp increase after passing the maximum normalized traffic load limit that the random access channel can handle ($1/e \approx 0.37$), as in the two-slice scenario.



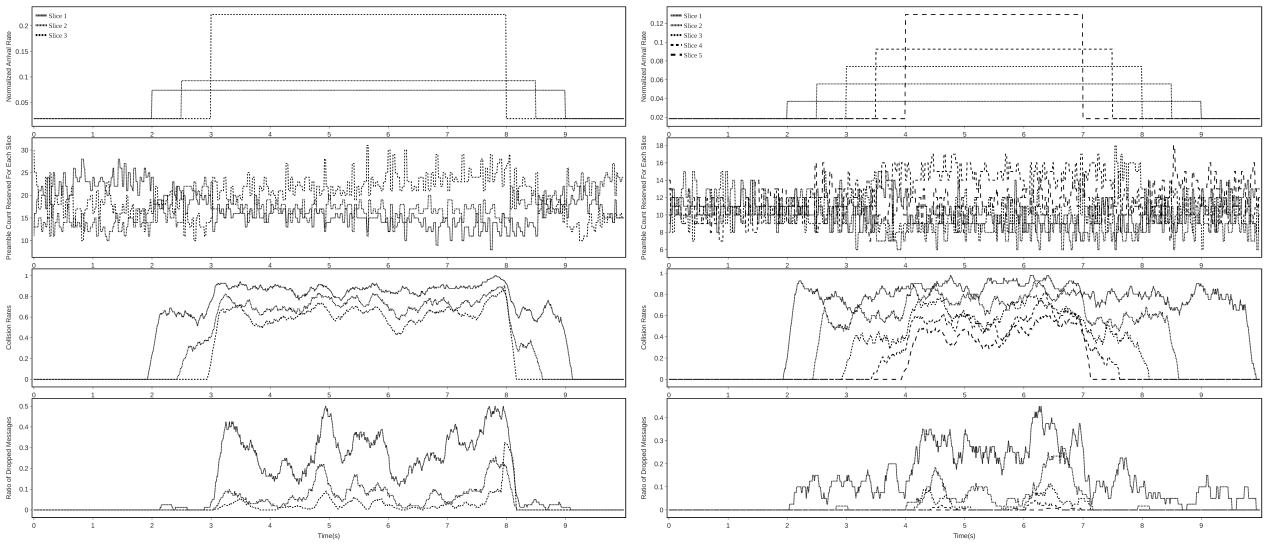
(a) (b)

Fig. 9: The timeline simulation graphs for 3-slice (a) and for 5-slice (b) using *CRF*

Fig. 5c, 6c and 8 demonstrates the behaviour of the *ideal* algorithm for scenarios with 2-slice and 3-slice. It slightly prioritize the slices in a similar way to the unsliced plot with respect to their prioritization levels in the figures. Furthemore, Fig. 8a shows that for the 3-slice scenario it prefers to allocate more number of preambles to the 3rd slice even in the beginning of the scenario when the total load is much less than $1/e \approx 0.37$ and normalized arrival rate for the 1st slice is highest. As a result of using the *ideal algorithm*, there are dropped preamble requests observed in the 1st slice even at the early stage of the scenario when the arrival rate of the 3rd is slice between 0.05 and 0.1 as and the total load is 0.24 shown in Fig. 8b.

D. Performance of Reinforcement Learning Based Method in a Dynamic Environment

In this part, the temporal behavior of the proposed method is presented to show how it reacts to changes in the traffic demands and how much time it is needed to rearrange the preamble allocations to meet the requirements. Different from the previous experiments, here not only the requests of the high-priority service change, but all service requests may also vary in time. The



(a) (b)

Fig. 10: The timeline simulation graphs for 3-slice (a) and for 5-slice (b) using the ideal alg.

simulations for the 5-slice scenario are also run in addition to the 3-slice scenario. For these simulations, only *CRF* is used as the reward function. The traffic pattern and the behavior of the proposed method are shown in Fig. 9 for both scenarios. The uppermost plots in Fig. 9a and 9b show the normalized arrival rate for each slice over time. The following below plots show the response of the proposed method to the changing environment as reserved preamble counts for each slice. The next below plots demonstrate the collision rates of each slice in that random access opportunity which is found using c_j/m_j. Finally, the bottom plots show the ratio of dropped messages to all transmitted messages on that random access opportunity. The same scenarios are implemented with also the *ideal algorithm*. The performance of the *ideal algorithm* is plotted in Fig. 10.

Fig. 9a plots the simulation results for the 3-slice scenario. From $t = 0$ to $t = 2$, the reserved preamble count for each slice stays almost constant since there exists no congestion and change in traffic. At $t = 2$, λ_1 increases suddenly while other slices still have the same normalized

arrival rates. After a reaction time of approximately $50ms$, the proposed method gives most of the preambles to slice-1 immediately. First, the collision rate of slice-1 increases a little, then, immediately drops thanks to the reallocation of preambles. Similarly, at $t = 2.5$, λ_2 increases to a higher value than λ_1. Right after, m_2 increases and slice-2 gets a major part of the preambles. Up until $t = 3$, m_1 and m_2 stays around the same levels with little difference between them.

In Fig. 9a, the total load exceeds the total capacity at $t = 3$. At this point, since the preamble resources are not enough for all slices, the method allocates most of the preambles to the high-priority slice, slice-3. Therefore, m_3 spikes, m_2 is nearly halved and m_1 is dropped to around between 5 and 10. As all slices cannot be supported at the same time, the proposed method sacrifices the low-priority slice in favor of other slices. The collision rate for the slice-1 oscillate around 1 up to $t = 8$. The collision rates for the other two slices change between 0.2 and 0.8, and the collision rate for slice-2 is higher than slice-3 even if the traffic to slice-3 is more than twice. In the last two seconds, λ_3 falls of suddenly, right after, m_3 also drops. m_1 and m_2 increase to their previous values where right before λ_3 increases. Then, respectively λ_2 and λ_3 drops to their previous values and the preamble allocation returns to its initial assignment.

Similarly, Fig. 9b plots the timeline simulation for the 5-slice scenario. From $t = 0$ to $t = 2$, there is no congestion and the preamble allocation of slices does not change. During this period, the number of preambles reserved to each slice is proportional to their priority levels. Normalized arrival rates of slices are increased starting from the low-priority slice to the high-priority slice. At the end of the simulation, the rates are decreased in the reverse order. The preamble allocation behavior is similar to the 3-slice scenario and the RL agent successfully prioritizes the slices. The full load is exceeded when the normalized arrival rate of slice-5 increases. In that case, the majority of preambles are reserved for slice-5.

Figs. 10a and 10b plot the timeline simulation of the *ideal algorithm* for 3-slice and 5-slice scenarios respectively. As pointed out above, the algorithm slightly prioritizes the slices in a way that the collision rates in each figure are very close to each other, hence the allocation of

each slice j is slightly better than the allocation of slice $j-1$. The plots in the time intervals [2,3] and [8,9] for 3-slice in Fig. 10a and, in the time intervals [2,4] and [7,9] for 5-slice in Fig. 10b show that the *ideal algorithm* can not prevent collisions for lower priority slices when the total load in the network is less than ($1/e \approx 0.37$). Fig. 9 shows that DRL-based approach manages not to drop any preamble requests in these time intervals and achieves to keep the collision rates lower than the *ideal algorithm* does. On the other hand, when the arrival rate of the high-priority slice increases and the total load becomes bigger than ($1/e \approx 0.37$), the DRL-based approach sacrifices the lowest priority slices ($1st$ one in 3-slice scenario, $1st$ and $2nd$ ones in 5-slice scenario) in favour of higher priority ones. Although the *ideal algorithm* tries to keep the collision rates under control for even the lowest priority slice when the total load is bigger than ($1/e \approx 0.37$), it may not be the best behavior when the total load is such high. Especially for some real life scenarios such as where a remote surgery slice has the highest priority and a sensor-IoT slice has the lowest priority, the lowest priority slice could need to be sacrificed.

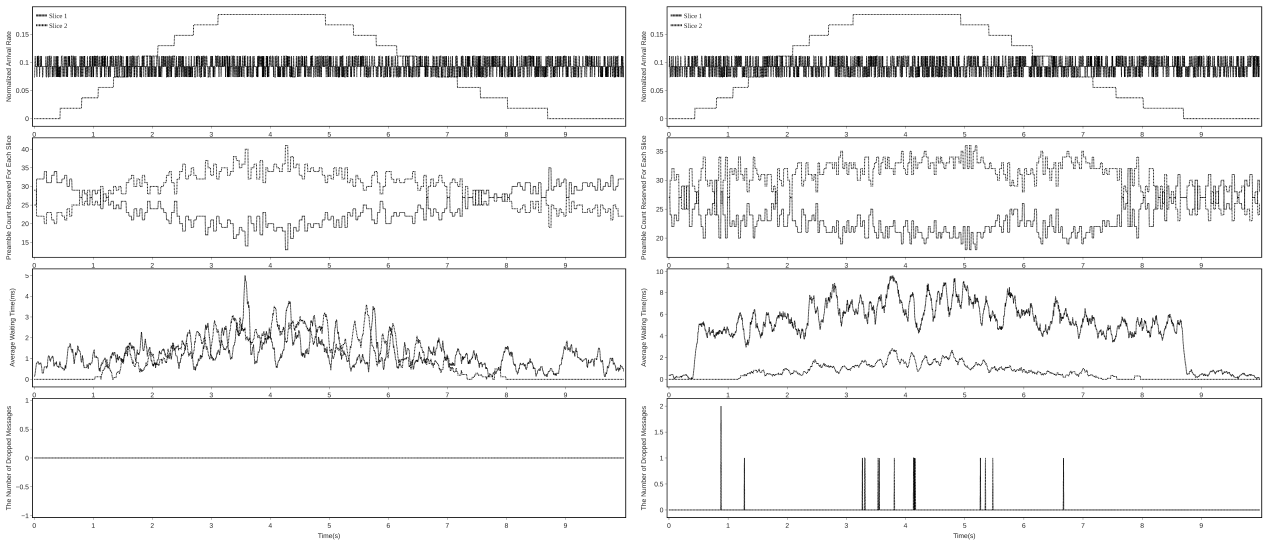
In summary, in timeline simulations, the sudden increase of arrival preamble requests to slices from low-priority to high-priority and the sudden decrease from high-priority to low-priority respectively are tested in the timeline. For both the 3-slice and 5-slice scenarios, when the traffic of the low-priority slice increases the preamble allocation is increased in favour of the low-priority slice. Following, the traffic to slices increases from lower to higher priority ones respectively and in each increase more preambles are reserved to the slice having lastly increased traffic. The moment when the traffic to high-priority slice increased, the total normalized arrival rate reaches 0.39 so that passes the full load ($1/e \approx 0.37$). After that, the proposed method suddenly allocates the vast majority of preambles to the high-priority slice. Nonetheless, Fig. 9a and 9b show that the proposed method does not neglect performances of lower priority slices, while the collision rate of the high-priority slice is kept under control. With approaching to the end of the timeline, the traffic assigned to slices decreases starting from the high-priority slice

to the low-priority slice respectively. The preamble allocation of the proposed method acts in a reverse direction. In the end, the preamble allocations return to the first position. These results confirm the effectiveness and adaptability of the proposed approach in a dynamic environment.

E. Two Priorities Scenario Used in [16]

The authors in [16] propose to assign weights γ_i to services based on their priorities. In this specific scenario, they use two services having priority weights $\gamma_1 = 2$ and $\gamma_2 = 1$. For each service, 10,000 devices are activated over 10 seconds (2000 RAO slots each $5ms$) with uniform arrival for the low-priority service and bursty beta arrival for the high-priority service. They stated that their algorithm produces results close to the ideal case in which the average delay (slots/device) is close to 0. In addition, they specify the average delay for the high-priority service is less than for the low-priority scenario. For a fair comparison, the same scenario is implemented for 2-slice and the results are plotted in Fig. 11. Please note that the RAO period is taken as $5ms$ here, whereas it is taken as ($10ms$) in previous experiments. Although the authors use $W = \infty$ and this value is taken as 10 in this current study, since no dropped preamble request is observed in these simulations, W has no effect on the comparison.

Fig. 11 shows the comparison of the proposed approach with the ideal algorithm on the scenario as defined above [16]. As shown in the figure, the average waiting time never reaches $5ms$ for both slices. The average waiting times for the high-priority slice are 0.98 ms in the DRL-based approach and 0.65 ms in the *ideal algorithm*. The average waiting times for the low-priority slice are 1.2 ms in the DRL-based approach and 4.7 ms in the *ideal algorithm*. Please note that the average waiting time is how much time successful preamble requests waited in the message backlog on average. Since it is taken as the time spent by one device/service for successfully accessing into the channel (slots/device) in [16], the average waiting time here is divided by the slot interval ($5ms$) in order to make a fair comparison with [16]. For the high-priority slice, it is calculated as $0.98ms/5ms \approx 0.2$ by the proposed approach and as $0.65ms/5ms \approx 0.13$



(a) (b)

Fig. 11: The timeline simulation graphs for 2-slice using the two priority scenario for DRL-based method (a) and for *ideal algorithm* (b) in [16]

by the *ideal algorithm*. For the low-priority slice, it is calculated as $1.2ms/5ms \approx 0.24$ and $4.7ms/5ms \approx 0.94$ for both algorithms respectively.

To sum up, the proposed method can successfully prioritize the slices in a way that the average delay (slots/device) of each slice is close to 0. Moreover, the high-priority slice has a lower average delay than the low-priority slice. It is also stated in [16] that they obtain average delays (slots/device) close to 0 without specifying exact values. Although the *ideal algorithm* manages to keep average delays (slots/device) lower than 1, there is a noticeable difference between the slices (0.94 for the low-priority slice and 0.13 for the high-priority slice). Even the *ideal algorithm* performs slightly better than the proposed approach for the high-priority slice based on the average delay metric, it drops a few number of preamble requests from the low-priority slice. As stated above, the proposed approach does not drop any requests.

VII. DISCUSSION

The experimental results show the effectiveness and adaptability of the proposed DRL-based approach in a dynamic environment. Different reward functions are evaluated and shown that penalizing collisions in addition to rewarding successful preambles gives the best performance. When penalizing is used in the reward function, reinforcement learning act more quickly and sharply due to the very nature of penalizing, which is fed into the system in a very short time (in a few RAO). However, using the other functions which do not include penalizing, the response time of the algorithm takes longer and it adapts more softly to the changing environment. Moreover, when penalizing is applied in reward functions, the number of reserved preambles are inclined to be given to lower priority slices when there is no traffic congestion. In other words, more importance is given to lower slices, when there is no high traffic for higher priority slices. This is the reason behind why the DRL-based approach using penalizing can perform better than *ideal algorithm* under some scenarios. On the other hand, in the event of traffic congestion DRL-based approach perform a steady and more prioritizing attitude in favour of higher priority slices when compared to *ideal algorithm*. There exists an obvious opposite attitude of DRL-based approach towards slices between low and high traffic congestion cases. These outcomes stand for DRL-based approach can change its pattern of prioritization behaviour under varying network traffic load.Please note that the same priority coefficient values for slices are employed in all reward functions in order to have a comparison basis. However, these coefficients could affect reward functions in different ways, therefore, in the future, different coefficients could be applied for different reward functions.

The running time of the proposed approach is proportional to the TRPO algorithm. However, an increase in the number of slices, which is varying in the experiments, causes an expansion in the state space. Recalling M is the number available preambles, the state space size for N number of slices can be calculated as $\frac{(M+3N-1)!}{M!(3N-1)!}$ since there are 3 state space members s_j,c_j

and u_j. The action space size is $\frac{(M+N-1)!}{M!(N-1)!}$ since the action list includes N members. Therefore, increasing N by 1, the state space size expands nearly M^3 times and the action space size expands M times. Since the solution space is the mapping from the state space to the action space, increasing N expands the solution space immensely. It is not easy to build a mathematical model for representing such a complex solution space. However, DRL is known to be able to diminish the curse of dimensionality caused by the state space theoretically [50].

In this study, the proposed approach is trained and evaluated by using simulated network traffic, since real world RACH preamble traffic data is not available. It is hard to collect such data from eNodeBs since they can not detect how many nodes are available in the environment. Moreover, since the nodes in the environment are distributed and mobile, collecting such data synchronously is hard. Due to all of these reasons, randomly generated traffic is used in training in this study. On other hand, having a real world data for different service types which have various QoS help to generate a more precise model for that environment. Moreover, in the experiments of this current study, the effects of some parameters such as transmission power of nodes, distance of nodes to eNodeBs, deafness of nodes, path loss and shadowing are not simulated. In the future, these parameters could be included in the experiments for more realistic simulations.

VIII. Conclusion

With the increasing significance of RAN resource allocation in 5G, flexible preamble allocation becomes an important problem. This study aims to find a solution to the RAN slicing problem by prioritizing slices. It explores deep reinforcement learning (DRL) to solve the optimum resource allocation problem in RAN slicing. Here, three reward functions are proposed for the RL formulation and mathematically analyzed.

Since at least 3 service types are defined for 5G [5] and, some studies propose 5 service types [6] for 5G in the literature, 3-slice and 5-slice scenarios are implemented in the experiments.

The proposed approach is evaluated by the following metrics: the number of reserved preambles, the average waiting time and the ratio of dropped messages to all transmitted messages. The proposed approach is compared with a recent study [16] in the literature. Moreover, in order to show its effectiveness it is compared with the unsliced scenario and the exhaustive search, in which all possible preamble allocations for a given traffic load are searched through and the best preamble allocation is chosen. The results show that the proposed method distributes preambles to different service classes according to their priorities successfully. It produces comparable results with [16]. In some network scenarios, it outperforms the ideal algorithm given in [16] by giving importance to lower priority slices in addition to higher priority slices when there is no traffic congestion.

To sum up, the proposed DRL-based approach is shown to be a suitable approach for RAN slicing, since it adapts to changes in the environment in a timely manner. It also can be deployed as an instantiable Virtual Network Function (VNF) to eNodeBs. Afterwards, eNodeBs slice PRACH preambles for network service classes that have different QoS needs under the dynamic environment. The VNF can be instantiated at the very beginning of eNodeB operation, then it can step in when there exists network congestion like in a power outage and restore scenario or in a temporary dense network formation scenario.

REFERENCES

[1] J. Kim, J. Lee, J. Kim, and J. Yun, "M2m service platforms: Survey, issues, and enabling technologies," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 61–76, First 2014.

[2] "Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions)," https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/, accessed: 2019-10-12.

[3] Cisco, "Cisco visual networking index: Forecast and trends, 2017–2022 white paper," (Visited December 2019) [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

[4] G. Wunder, P. Jung, M. Kasparick, T. Wild, F. Schaich, Y. Chen, S. T. Brink, I. Gaspar, N. Michailow, A. Festag, L. Mendes, N. Cassiau, D. Ktenas, M. Dryjanski, S. Pietrzyk, B. Eged, P. Vago, and F. Wiedmann, "5gnow: non-orthogonal,

asynchronous waveforms for future mobile applications," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 97–105, 2014.

[5] P. Marsch, Bulakci, I. Silva, P. Arnold, N. Bayer, J. Belschner, T. Rosowski, G. Zimmermann, M. Ericson, A. Kaloxylos, P. Spapis, A. Ibrahim, Y. Yang, S. Singh, H. Celik, J. Gebert, A. Prasad, F. Moya, M. Säily, and J. Monserrat, "D2.2 draft overall 5g ran design," 06 2016.

[6] S. Vural, N. Wang, P. Bucknell, G. Foster, R. Tafazolli, and J. Muller, "Dynamic preamble subset allocation for ran slicing in 5g networks," *IEEE Access*, vol. 6, pp. 13 015–13 032, 2018.

[7] W. Tian, M. Fan, C. Zeng, Y. Liu, D. He, and Q. Zhang, "Telerobotic spinal surgery based on 5g network: The first 12 cases," *Neurospine*, vol. 17, pp. 114–120, 03 2020.

[8] M. Vilgelm, S. Schiessl, H. Al-Zubaidy, W. Kellerer, and J. Gross, "On the reliability of lte random access: Performance bounds for machine-to-machine burst resolution time," in *2018 IEEE International Conference on Communications (ICC)*, May 2018, pp. 1–7.

[9] J. Oueis and E. Strinati, "Uplink traffic in future mobile networks: Pulling the alarm," 05 2016, pp. 583–593.

[10] Ericsson, "Ericssonmobility report, on the pulse of the networked society," 2015. [Online]. Available: http://www.ericsson.com/res/docs/2015/ericsson-mobility-report-june-2015.pdf

[11] Y. Dong, Z. Chen, P. Fan, and K. B. Letaief, "Mobility-aware uplink interference model for 5g heterogeneous networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 3, pp. 2231–2244, 2016.

[12] C. Nguyen, H. Dinh Thai, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," 10 2018.

[13] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L. Wang, "Deep reinforcement learning for mobile 5g and beyond: Fundamentals, applications, and challenges," *IEEE Vehicular Technology Magazine*, vol. 14, no. 2, pp. 44–52, 2019.

[14] R. Boutaba, M. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. Caicedo Rendon, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, 05 2018.

[15] Y. Shi, Y. E. Sagduyu, and T. Erpek, "Reinforcement learning for dynamic resource optimization in 5g radio access network slicing," in *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2020, pp. 1–6.

[16] J. Liu, M. Agiwal, M. Qu, and H. Jin, "Online control of preamble groups with priority in massive iot networks," *IEEE Journal on Selected Areas in Communications*, pp. 1–1, 2020.

[17] W. T. Toor and H. Jin, "Comparative study of access class barring and extended access barring for machine type communications," in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, Oct 2017, pp. 604–609.

[18] N. Zangar, S. Gharbi, and M. Abdennebi, "Service differentiation strategy based on macb factor for m2m communications

in lte-a networks," in *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan 2016, pp. 693–698.

[19] N. Li, C. Cao, and C. Wang, "Dynamic resource allocation and access class barring scheme for delay-sensitive devices in machine to machine (m2m) communications," *Sensors*, vol. 17, no. 6, p. 1407, Jun 2017. [Online]. Available: http://dx.doi.org/10.3390/s17061407

[20] Y. Sim and D.-H. Cho, "Performance analysis of priority-based access class barring scheme for massive mtc random access," *IEEE Systems Journal*, vol. 14, no. 4, pp. 5245–5252, 2020.

[21] K.-D. Lee, S. Kim, and B. Yi, "Throughput comparison of random access methods for M2m service over LTE networks," in *2011 IEEE GLOBECOM Workshops (GC Wkshps)*, Dec. 2011, pp. 373–377, iSSN: 2166-0077.

[22] T. Lin, C. Lee, J. Cheng, and W. Chen, "Prada: Prioritized random access with dynamic access barring for mtc in 3gpp lte-a networks," *IEEE Transactions on Vehicular Technology*, vol. 63, no. 5, pp. 2467–2472, Jun 2014.

[23] K. Lee, M. Reisslein, K. Ryu, and S. Kim, "Handling randomness of multi-class random access loads in lte-advanced network supporting small data applications," in *2012 IEEE Globecom Workshops*, Dec 2012, pp. 436–440.

[24] C. Kalalas, F. Vazquez-Gallego, and J. Alonso-Zarate, "Handling mission-critical communication in smart grid distribution automation services through lte," in *2016 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, Nov 2016, pp. 399–404.

[25] M. Vilgelm, M. Gürsu, W. Kellerer, and M. Reisslein, "Latmapa: Load-adaptive throughput-maximizing preamble allocation for prioritization in 5g random access," *IEEE Access*, vol. PP, pp. 1–1, 01 2017.

[26] X. Zhao, J. Zhai, and G. Fang, "An access priority level based random access scheme for qos guarantee in td-lte-a systems," in *2014 IEEE 80th Vehicular Technology Conference (VTC2014-Fall)*, Sep. 2014, pp. 1–5.

[27] D. Pacheco-Paramo, L. Tello-Oquendo, V. Pla, and J. Martinez-Bauset, "Deep reinforcement learning mechanism for dynamic access control in wireless networks handling mmtc," *Ad Hoc Networks*, vol. 94, p. 101939, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870519300976

[28] A. Mohammed Mikaeil, W. Hu, and L. Li, "Joint allocation of radio and fronthaul resources in multi-wavelength-enabled c-ran based on reinforcement learning," *Journal of Lightwave Technology*, vol. 37, no. 23, pp. 5780–5789, Dec 2019.

[29] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2019.

[30] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for mec," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2018, pp. 1–6.

[31] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 11, pp. 11 158–11 168, Nov 2019.

[32] Y. Wei, F. R. Yu, M. Song, and Z. Han, "User scheduling and resource allocation in hetnets with hybrid energy supply: An actor-critic reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 17, no. 1, pp. 680–692, Jan 2018.

[33] Z. Chen and D. B. Smith, "Heterogeneous machine-type communications in cellular networks: Random access optimization by deep reinforcement learning," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[34] C. Chang and N. Nikaein, "Ran runtime slicing system for flexible and dynamic service execution environment," *IEEE Access*, vol. 6, pp. 34 018–34 042, 2018.

[35] H. D. R. Albonda and J. Pérez-Romero, "An efficient ran slicing strategy for a heterogeneous network with embb and v2x services," *IEEE Access*, vol. 7, pp. 44 771–44 782, 2019.

[36] D. Marabissi and R. Fantacci, "Highly flexible ran slicing approach to manage isolation, priority, efficiency," *IEEE Access*, vol. 7, pp. 97 130–97 142, 2019.

[37] M. R. Raza, C. Natalino, P. Öhlen, L. Wosinska, and P. Monti, "Reinforcement learning for slicing in a 5g flexible ran," *Journal of Lightwave Technology*, vol. 37, no. 20, pp. 5161–5169, Oct 2019.

[38] B. M. Sesia S., Toufik I., *The UMTS Long Term Evolution: From Theory to Practice, 2nd Edition*, 2011.

[39] F. H. S. Pereira, C. A. Astudillo, T. P. C. De Andrade, and N. L. S. Da Fonseca, "Prach power control mechanism for improving random-access energy efficiency in long term evolution," in *2018 IEEE 10th Latin-American Conference on Communications (LATINCOM)*, 2018, pp. 1–6.

[40] J. Choi, J. Ding, P. Le, and Z. Ding, "Grant-free random access in machine-type communication: Approaches and challenges," 12 2020.

[41] V. Savaux, A. Kountouris, Y. Louët, and C. Moy, "Modeling of Time and Frequency Random Access Network and Throughput Capacity Analysis," *EAI Endorsed Transactions on Cognitive Communications*, vol. 3, no. 11, p. e2, 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01531239

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[43] Y. Li, "Deep reinforcement learning: An overview," *CoRR*, vol. abs/1701.07274, 2017. [Online]. Available: http://arxiv.org/abs/1701.07274

[44] P. Coady, "trpo," https://github.com/pat-coady/trpo, 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1183378

[45] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[46] S. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *International Conference on Learning Representations*, 2018.

[47] M. Koseoglu, "Pricing-based load control of m2m traffic for the lte-a random access channel," *IEEE Transactions on Communications*, vol. 65, no. 3, pp. 1353–1365, March 2017.

[48] M. Koseoglu, "Lower bounds on the lte-a average random access delay under massive m2m arrivals," *IEEE Transactions on Communications*, vol. 64, no. 5, pp. 2104–2115, 2016.

[49] M. Koseoglu, "Smart pricing for service differentiation and load control of the lte-a iot system," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec 2015, pp. 187–192.

[50] C. P. Andriotis and K. G. Papakonstantinou, “Managing engineering systems with large state and action spaces through deep reinforcement learning,” 2018.