Stepwise Optimal Inter-Slices Radio Resource Scheduling for Service-Level Agreement Assurance

Daniel Campos[1], Gabriel M. F. de Almeida[1], William T. P. Junior[1], Cleverson V. Nahum[2], Aldebaro Klautau[2], Mohammad J. Abdel-Rahman[3,4], and Kleber V. Cardoso[1]

[1] Universidade Federal de Goiás (UFG), Goiânia, GO – Brazil,
[2]Universidade Federal do Pará (UFPA), Belém, PA – Brazil,
[3]Data Science Dept., Princess Sumaya University for Technology - Jordan,
[4]Electrical & Computer Engineering Department, Virginia Tech, USA.

{danielcampos, gabrielmatheus, williamjunior, kleber}@inf.ufg.br;
{cleverson, aldebaro}@ufpa.br; mo7ammad@vt.edu

Abstract. *In 5G networks and beyond, radio access networks (RANs) must be able to support multiple services with different service level agreements (SLAs). Network slicing is a critical concept in this context and it depends on an efficient approach for radio resource scheduling (RRS). Inter-slices RRS is responsible for allocating resource block groups (RBGs) to the slices to ensure their SLAs. Mainly motivated by the O-RAN initiative, several works in the literature have presented proposals based on machine learning (ML) to solve this problem. However, there is still a lack of problem formalization and an optimal strategy, which are both introduced in this work. Through simulations, we compare our approach with a state-of-the-art deep reinforcement learning (DRL) agent. The results show the excess resources employed by the agent when they are plentiful, suggesting an unnecessary increase in energy consumption. Additionally, we show the relevant gap between solutions when the resources are scarce. Finally, we discuss guidelines on how to improve ML-based approaches to the inter-slices RRS problem.*

1. Introduction

While the adoption of Open RAN is still in the very beginning and surrounded by discussions, the impact of the O-RAN Alliance and its standards is already huge in the telecommunications ecosystem. ML-based approaches in the RAN are not new since 3GPP Release 8 (from 2006) already had self-organizing networks (SON) related exactly to this type of approach. The several advances in the ML field and the design adopted for the O-RAN architecture [Polese et al. 2023] have kept most of the academic interest strongly guided in this direction when investigating RAN-related issues. Two key components of the O-RAN architecture are the near real-time RAN intelligent controller (Near-RT RIC) and the non-real-time RAN intelligent controller (Non-RT RIC). O-RAN compatible solutions must be developed as xApps (running in the Near-RT RIC) and rApps (running in the Non-RT RIC), which are generally ML-based applications. As a consequence, commonly, the ML-based proposals found in the literature are compared only against other ML-based counterparts. In the context of resource allocation, this may raise a basic question: how far from the optimal are the solutions?

In the RAN, network slicing is critical and involves non-trivial resource allocation. Network slicing is the main enabler for supporting multiple services with different SLAs

over the same infrastructure. An SLA takes into account a set of requirements that the network operator must ensure to provide the necessary quality of service (QoS) to the users' applications. To ensure the QoS, the network operator creates slices to serve the services with SLAs that may be very different. For example, video streaming from a smartphone requires a high throughput but tolerates packet loss and some latency, thus it may belong to an enhanced mobile broadband (eMBB) slice. On the other hand, self-driving vehicles need a very low packet loss and latency, best supported by ultra-reliable low-latency communication (URLLC) slices. This means that each slice must receive a specific amount of resources to satisfy the correspondent SLA and this resource allocation is highly dynamic in the RAN due to two main reasons. First, the number of user equipment (UEs) associated with each base station (BS) changes as the users move, thus, the consumption of resources also varies. Second, wireless channel conditions of each UE also vary, not only due to user mobility but also due to other environmental characteristics. This context is appealing to model-free approaches such as ML-based ones, but we argue that this does not preclude the pursuit of problem formalization and optimal solutions. Thus, we can build consistent performance references and find important insights that can contribute to designing better approaches, including the ML-based ones as we will discuss later.

Related work – There are several papers in the literature investigating issues related to network slicing in the RAN. In the following, we do not try to be comprehensive, but cite some critical works on the topic, mainly considering inter-slices allocation and including some recent state-of-the-art papers. [Kokku et al. 2012] is a well-known solution for inter-slice scheduling referenced by other inter-slice schedulers. Because it was developed in the early 2010s, its assumptions are outdated. The main problem of its scheduling is not considering orthogonal frequency division multiple access (OFDMA), thus scheduling radio resources by allocating all RBGs to a single slice at every transmission time interval (TTI). Another issue is that the resources are distributed in a weighted round-robin way, where each slice weight is defined by its historical throughput. This weight may not represent the needed resources to ensure each slice's SLA, as its restrictions may not directly relate to throughput.

[Chen et al. 2023] shows that the spectral efficiency can differ between RBGs for a single UE. It then develops a heuristic that maximizes the total throughput by selecting the best pair of RBG and UE. The intra-slice scheduling must be greedy to enable predicting its allocation to select the best RBGs for each slice. Nonetheless, this solves only the problem of selecting the resources, but the number of RBGs for each slice is determined similarly to [Kokku et al. 2012], using a pre-determined slice weight. [Nahum et al. 2023] develops a DRL agent to solve the inter-slices scheduling among eMBB, URLLC, and BE slices. We call this solution the DRL agent. It uses *intents*, which are equivalent to SLAs from the scheduler's perspective, to determine *intent-drifts*: the normalized difference between a required metric and its required value. Therefore, the agent's objective is to minimize the intent-drift for all slices.

[Khodapanah et al. 2020] provides a framework for inter-slice RRS using artificial neural networks to maximize the number of fulfilled SLA requirements assuming the allocation of fractional resources. In [Lotfi et al. 2023], an attention-based DRL agent is presented for scheduling resource blocks (RBs), not RBGs, between eMBB, URLLC, and massive machine-type communication (mMTC) slices. [Polese et al. 2022] uses proximal policy optimization to train a DRL agent that executes as an xApp in O-RAN architecture, scheduling resources to maximize or minimize metrics for each slice. Additionally, [Mei et al. 2021] proposes a DRL framework combining a deep deterministic poly gradient and

a deep-Q-network algorithm to address the inter-slices scheduling problem. A common aspect of the related works is that every solution always allocates 100% of the BS RBGs, which may be inefficient in several scenarios. Allocating only the minimal resources necessary to satisfy the SLAs brings benefits, such as the possibility to serve more users or minimize energy consumption.

Our contributions and paper organization – In this work, we optimally solve the stepwise optimal inter-slices RRS for SLA assurance problem, i.e., employing the minimum amount of RBGs to assure the SLA of every UE. In summary, our main contributions are:

- The formalization of the problem of stepwise inter-slices RRS for SLA assurance.
- The design of an algorithm that solves the problem in polynomial time.
- A comparison of our approach with a state-of-the-art DRL agent, using simulation, which illustrates the room for improvement and provides insights on how to improve ML-based approaches.

Section 2 presents the system model and the problem formulation. Section 3 describes how to solve the formulated problem with a polynomial algorithm. Section 4 evaluates our solution in comparison with the DRL agent and a weighted round-robin in a simulated environment. Lastly, Section 5 contains our conclusions and future works.

2. System model and problem formulation

In this section, we first introduce the system model employed to define the problem, delineating parameters pertinent to our study. Subsequently, we present the problem formulation, specifying the metrics associated with the SLA of each slice type and outlining the objective of our formulation.

2.1. System model

We assume that each UE u possesses a buffer represented as an array $L_u = [B_u(0), B_u(1), \ldots, B_u(L-1)]$, where L is the maximum buffer latency, in TTIs. Each element $B_u(i) \in L_u$ indicates the number of packets awaiting transmission for i timesteps. For example, if $L_u = [1, 3, 5]$, it means that 1 packet has just arrived (waiting for 0 timesteps), 3 packets have been waiting for 1 timestep, and 5 packets have been waiting for 2 timesteps. Furthermore, we denote the cumulative count of transmitted packets with an array $B_u^{sent} = [B_u^{sent}(0), B_u^{sent}(1), \ldots, B_u^{sent}(\eta - 1)]$. Each $B_u^{sent}(i)$ represents how many packets waited for i timesteps until been transmitted since the beginning of the environment. For instance, if $B_u^{sent} = [3, 5, 4]$, it indicates that, until now, 3 packets were immediately sent (no waiting), 5 packets waited for 1 timestep before transmission, and 4 packets waited for 2 timesteps before being transmitted.

The BS bandwidth is discretized into RBs with $2^{\zeta} \cdot 180$ kHz each, where $\zeta \in [0, 1, 2, 3, 4]$ is the BS option, as in the 5G standards [ETSI 2020a]. The TTI time length is directly related to the RB bandwidth, expressed as $I = 2^{-\zeta}$ ms. Moreover, ρ RBs are aggregated into one RBG with a total bandwidth of $R = \rho \cdot 2^{\zeta} \cdot 180$ kHz [ETSI 2020b]. This way, we discretize the time in our model as timesteps lasting 1 TTI each. We define a set $\mathcal{T} = \{0, 1, \ldots, \eta\}$ comprising the timesteps, where η is the current one. Additionally, we introduce a time window of $W \in \mathbb{N}_+$ timesteps for calculating historical metrics. As W may be greater than the number of past steps, $\omega = \min(W, \eta + 1)$ is used as the current window. The stages of a timestep are as follows: (*i*) packets arrive in each UE's buffer, then (*ii*) the inter-slice schedulers allocate the available RBGs of the BS among the slices, which (*iii*) distribute the received resource among the UEs to (*iv*) transmit packets from the buffer to the BS in an uplink communication.

2.2. Problem formulation

We address the stepwise optimal inter-slices RRS for SLA assurance problem by defining the minimal number of RBGs necessary for ensuring the SLA requirements of every UE in the actual timestep. We denote by $\mathcal{S}$ the set of slices in the environment, while $\mathcal{U}_s$ is the set of users assigned to the slice $s \in \mathcal{S}$. In this way, we express $\alpha_s \in \mathbb{N}$ as the amount of RBG served to slice $s \in \mathcal{S}$ in timestep η. To calculate the predicted metrics for the UEs, we define β_u as the number of RBGs that will be allocated for $u \in \mathcal{U}_s$ if $s \in \mathcal{S}$ receives α_s RBGs. In this work, we address three types of slices: best effort (BE), eMBB, and URLLC. The BE slice has services with tolerant throughput requirements [Khodapanah et al. 2020]. The SLA for BE is defined as minimum values for fifth-percentile throughput and long-term throughput. Moreover, the SLA for eMBB and URLLC comprises a minimum value for the served throughput and maximum values for the packet loss rate and the average buffer latency [Nahum et al. 2023]. To achieve a generic formulation that may be expanded to include new slices and SLAs, we categorize the set of slices into two subsets $\mathcal{S} = \{\mathcal{S}^{buf} \cup \mathcal{S}^{thr}\}$. The first, $\mathcal{S}^{buf}$, comprises slices whose SLA relates to buffering, while $\mathcal{S}^{thr}$ consists of slices with SLA depending solely on throughput metrics. In this work, we assume $\mathcal{S}^{thr} = \{BE\}$ and $\mathcal{S}^{buf} = \{eMBB, URLLC\}$. In the following, we describe the five SLA constraints and the objective function of our problem.

Maximum tolerated average buffer latency – This constraint prohibits the solution from surpassing the average buffer latency threshold defined for each slice $s \in \mathcal{S}^{buf}$, denoted as l_s^{req}. We use $\psi_u(i)$ as the predicted number of packets in $B_u(i)$ that will be transmitted if u receives β_u RBGs. The constraint is defined as follows:

$$\frac{\sum_{i=0}^{L-1}(\psi_u(i) + B_u^{sent}(i)) \cdot i}{\sum_{i=0}^{L-1}(\psi_u(i) + B_u^{sent}(i))} \cdot I \leq l_s^{req}, \qquad \forall u \in \mathcal{U}_s, s \in \mathcal{S}^{buf}. \tag{1}$$

Packet loss rate – This constraint prevents the packet loss rate from violating the packet loss requirement for a slice $s \in \mathcal{S}^{buf}$, denoted by p_s^{req}. We consider a scenario where packets can be dropped due to two reasons: (*i*) the buffer reaches its maximum capacity at the moment a packet arrives, and (*ii*) a packet achieves its maximum latency L. We denote $\overline{D_u^{arr}}(t)$ as the number of packets dropped at timestep $t \in \mathcal{T}$ due to maximum buffer capacity and $\overline{D_u^{lat}}(t)$ as the number of packets dropped at timestep $t \in \mathcal{T}$ due to the maximum latency constraint. As $\overline{D_u^{lat}(\eta)}$ can only be known after scheduling, we call ϕ_u^{lat} the number of packets that will be dropped in this timestep if u receives β_u RBGs. Similarly, we write ϕ_u^{arr} as the packets that will drop during arrival in the $\eta + 1$ timestep, given β_u. We assume that λ_u packets will arrive for the UE u in the timestep $\eta + 1$ when calculating ϕ_u^{arr}. Considering $\overline{B_u^{start}(t)}$ as the number of packets in the buffer at the beginning of timestep $t \in \eta$, before packet arrival, and $\overline{A_u}(t)$ as the number of packets that arrived the buffer at timestep $t \in \mathcal{T}$, we define the packet loss rate constraint as:

$$\frac{\sum_{t=\eta-\omega+1}^{\eta} \overline{D_u^{arr}}(t) + \sum_{t=\eta-\omega+1}^{\eta} \overline{D_u^{lat}}(t) + \phi_u^{lat} + \phi_u^{arr}}{\overline{B_u^{start}}(\eta - \omega + 1) + \sum_{t=\eta-\omega+1}^{\eta} \overline{A_u}(t) + \lambda_u} \leq p_u^{req}, \ \forall u \in \mathcal{U}_s, s \in \mathcal{S}^{buf}. \tag{2}$$

Served throughput – This constraint dictates that the required served throughput t_s^{req} must be upheld for $u \in \mathcal{U}_s$. We denote E_u as the spectral efficiency of the user $u \in \mathcal{U}_s$ in the current timestep. The constraint is defined as:

$$\beta_u \cdot R \cdot E_u \geq t_s^{req} \qquad \forall u \in \mathcal{U}_s, s \in \mathcal{S}^{buf}. \tag{3}$$

Long-term throughput – This constraint prevents the solution from violating the long-term throughput requirement for all slices $s \in \mathcal{S}^{thr}$, denoted by g_s^{req}. We represent the historically served throughput of user $u \in \mathcal{U}_s$ at timestep $t \in \mathcal{T}$ as $\overline{T_u}(t)$. The long-term throughput is defined as the average throughput of the UE over the time window of the last ω timesteps. Therefore, the long-term throughput constraint is defined as follows:

$$\frac{1}{\omega} \cdot \Big(\beta_u \cdot R \cdot E_u + \sum\nolimits_{t=\eta-\omega+1}^{\eta-1} \overline{T_u}(t)\Big) \geq g_s^{req}, \qquad \forall u \in \mathcal{U}_s, s \in \mathcal{S}^{thr}. \tag{4}$$

Fifth-percentile throughput – This constraint assures the fifth-percentile throughput requirement f_s^{req} for $s \in \mathcal{S}^{thr}$. This constraint prevents the solution from letting the fifth-percentile throughput of a user $u \in \mathcal{U}_s$ be below f_s^{req}. The fifth-percentile throughput is calculated by obtaining the h-th element of the sorted list $[\overline{T_u}(\eta-\omega+1), \ldots, \overline{T_u}(\eta-1), \beta_u \cdot R \cdot E_u]$ of the throughput in the last ω timesteps, where $h = \lfloor \frac{5}{100}\omega \rfloor$. Considering $W < 20$, we can simplify the metric to represent the constraint as:

$$\min\Big(\overline{T_u}(\eta-\omega+1), \ldots, \overline{T_u}(\eta-1), \beta_u \cdot R \cdot E_u\Big) \geq f_s^{req}, \qquad \forall u \in \mathcal{U}_s, s \in \mathcal{S}^{thr}. \tag{5}$$

We aim to minimize radio resource usage while ensuring QoS for every user. Thus, the stepwise optimal inter-slices RRS for SLA assurance problem is formalized as:

$$\begin{array}{l} \text{minimize} \sum\nolimits_{s \in \mathcal{S}} \alpha_s \\ \text{Subject to Equation}(1) \text{ to } (5) \end{array} . \tag{6}$$

However, linearizing the formulated problem is a complex task, which includes restricting β_u to how the intra-slice scheduler would work and expressing ψ_u, ϕ_u^{lat} and ϕ_u^{arr} as variables that depend on β_u for each UE u. Fortunately, this problem can be solved by a polynomial algorithm, as explained in Section 3. It is important to note that our stepwise problem is not equivalent to solving the scheduling for all the timesteps all at once, which is an NP-hard problem.

3. Stepwise Optimal Algorithm

The problem in Equation 6 can be solved with a polynomial greedy algorithm. We call our strategy the stepwise optimal algorithm (SOA), which considers a scenario where it is possible to respect the QoS restrictions for all UEs at each timestep. In the following, we describe how SOA assures SLA and predicts intra-slice scheduling.

3.1. Minimum throughput necessary

The output of the scheduling must be an allocation of RBGs, which is directly related to the UE's throughput. Based on this fact, we convert every SLA requirement in a Minimum Throughput Necessary (MTN) to respect the restriction of $u \in \mathcal{U}_s$ at the actual timestep. This strategy enables SOA to be expanded to new scenarios by calculating the MTN for new SLA restrictions, which may describe different slices.

The MTN to respect the served throughput constraint of Equation 3 is expressed as $MTN_s^t(u) = t_s^{req}$. The same simple expression happens to the fifth-percentile throughput MTN, noted as $MTN_s^f(u)$. Because SOA respects all restrictions at every timestep, then $\overline{T_u}(t) \geq f_s^{req}, \forall t \in \mathcal{T} \setminus \{\eta\}$. Thus, $MTN_s^f(u) = f_s^{req}$ ensures the constraint in Equation

5. We denote by $MTN_s^g(u)$ the MTN to assure the long-term throughput requirement of Equation 4, obtained by isolating the served throughput of the actual timestep:

$$MTN_s^g(u) = g_s^{req} \cdot \omega - \sum_{t=\eta-\omega+1}^{\eta-1} \overline{T_u}(t). \tag{7}$$

The average buffer latency MTN, denoted as $MTN_s^l(u)$, ensures the constraint of Equation 1. We noted that respecting this constraint while minimizing resources, in the long term, tends to a scenario where packets are sent at the last moment before l_s^{req}. Hence, we approximate $MTN_s^l(u)$ as the MTN to send the packets that will have waited for more than l_s^{req} in the next timestep:

$$MTN_s^l(u) = B_u\left(\left\lfloor \frac{l_s^{req}}{I} \right\rfloor\right) \cdot Z_s \cdot \frac{1}{I}, \tag{8}$$

where Z_s is the packet size for the slice $s \in \mathcal{S}$. The packet loss rate MTN $MTN_s^p(u)$ must ensure that ϕ_u^{lat} and ϕ_u^{arr} are small enough to respect the constraint of Equation 2. We approximate the number of packets that will arrive in the next timestep as $\lambda_u = \overline{A_u}(\eta)$. Considering that no resource is allocated to the UE u, we calculate $\phi_u^{lat} = B_u(L-1)$ and $\phi_u^{arr} = \left\lceil \frac{1}{Z_s} \cdot \max(0,\ (\overline{A_u}(\eta) + \sum_{i=0}^{L_u-1} B_u(i)) \cdot Z_s - B^{max}) \right\rceil$, with B^{max} as the buffer capacity in bits. Then, the number of packets that will drop between this scheduling and the next is $\max(\phi_u^{lat}, \phi_u^{arr})$, as $\phi_u^{lat} + \phi_u^{arr}$ may count the same packet twice. We express the denominator of Equation 2, the total of packets, as $\theta = \overline{A_u}(\eta) + \overline{B_u^{start}}(\eta - \omega + 1) + \sum_{n=\eta-\omega+1}^{\eta} \overline{A_u}(n)$. Thus, the MTN to send packets under drop risk and respect p_s^{req} is:

$$MTN_s^p(u) = \left\lceil Z_s \cdot \max(0, \sum_{n=\eta-\omega+1}^{\eta} \overline{D_u^{arr}}(n) + \sum_{n=\eta-\omega+1}^{\eta-1} \overline{D_u^{lat}}(n) + \max(\phi_u^{lat}, \phi_u^{arr}) - p_s^{req} \cdot \theta) \right\rceil \frac{1}{I}. \tag{9}$$

Lastly, we express as $MTN_s(u)$ the MTN to ensure the SLA of a UE $u \in \mathcal{U}_s$ in the actual step. It is calculated as the maximal MTN among the restrictions of u:

$$MTN_s(u) = \begin{cases} \max(MTN_s^t(u), MTN_s^l(u), MTN_s^p(u)), & \text{if } s \in S^{buf} \\ \max(MTN_s^f(u), MTN_s^g(u)), & \text{if } s \in S^{thr} \end{cases}. \tag{10}$$

3.2. RBG allocation

The SOA achieves $MTN_s(u)$ at every timestep by ensuring the allocation of at least $\beta_u^{min} = \left\lceil \frac{MTN_s(u)}{E_u \cdot R} \right\rceil$ RBGs for each $u \in \mathcal{U}_s$, $s \in \mathcal{S}$. Although the intra-slice scheduler allocates RBGs for the UEs, we can predict its scheduling similarly to [Chen et al. 2023]. As in [Nahum et al. 2023], we consider that the intra-slice scheduler for each slice $s \in \mathcal{S}$ is a Round-Robin, which cycles through $\mathcal{U}_s$ uniformly distributing the slices' RBGs among the users. The Round-Robin state is saved as an `offset`, the index of the next UE in the cycle. Knowing the `offset`, we act as a Round-Robin, allocating 1 RBG at a time until the number of RBGs for $u \in \mathcal{U}_s$ is $\beta_u \geq \beta_u^{min}$. Hence, SOA allocates $\alpha_s = \sum_{u \in \mathcal{U}_s} \beta_u$ for each slice $s \in \mathcal{S}$ as the minimal number of RBGs needed to assure the SLA of every $u \in \mathcal{U}_s$. The SOA allocation is described in Algorithm 1. The $MTN_s(u)$ function can be implemented in time $\mathcal{O}(1)$ using cumulative variables for the summations. Considering a scenario with limited resources, the loop at line 6 could stop the algorithm if the total

allocated RBGs reaches G, which is the number of available RBGs in the BS. Hence, as every RBG is allocated only once, the time complexity of SOA is $\mathcal{O}(G)$.

Algorithm 1: SOA allocation process.

Data: Set of slices $\mathcal{S} = \{s_1, \ldots, s_{|\mathcal{S}|}\}$ and set of users for each slice $\mathcal{U}_{s_1}, \ldots, \mathcal{U}_{|\mathcal{S}|}$
Result: Number of RBGs allocated for each slice $s \in \mathcal{S}$: $\alpha_{s_1}, \ldots, \alpha_{s_{|\mathcal{S}|}}$

for $s \in \mathcal{S}$ **do**
  `offset` $\leftarrow$ getIntraSliceRoundRobinOffset(s)
  for $u \in \mathcal{U}_s$ **do**
    $\beta_u \leftarrow 0$
    $\beta_u^{min} \leftarrow \left\lceil \frac{MTN_s(u)}{E_u \cdot R} \right\rceil$
  while $\exists u \in \mathcal{U}_s$ *such that* $\beta_u < \beta_u^{min}$ **do**
    $u \leftarrow \mathcal{U}_s$[`offset`] `// Mimicking Round-Robin`
    `offset` $\leftarrow$ `offset` $+ 1 \pmod{|\mathcal{U}_s|}$ `// intra-slice allocation`
    $\beta_u \leftarrow \beta_u + 1$
  $\alpha_s \leftarrow \sum_{u \in \mathcal{U}_s} \beta_u$
return $\alpha_{s_1}, \ldots, \alpha_{s_{|\mathcal{S}|}}$

4. Evaluation

In this section, we present the evaluation results of the proposed SOA. First, we describe the simulation environment used to implement the wireless network and its UEs. Then, we compare the SOA solutions with two state-of-the-art models: a weighted round-robin algorithm and the DRL agent proposed in [Nahum et al. 2023]. All evaluation results are publicly available at GitHub[1].

4.1. Simulation

To evaluate the SOA in different scenarios, we implemented a simulator that leverages [Nahum et al. 2023] dataset of realistic spectral efficiency values generated with the channel impulse responses for a wireless network simulated with QUAsi Deterministic RadIo channel GenerAtor (QuaDRiGa)[2] [Jaeckel et al. 2014]. The dataset contains 50 different trials of uplink communication that consider a massive Multiple Input Multiple Output (MIMO) system, the dual-slope path loss statistical models of 3GPP 38.901 UMi [Mondal et al. 2015,Zhu et al. 2021], the interference from the six more interfering nearby cells, shadow fading, and the MIMO spectral efficiency estimate equation from [Heath Jr and Lozano 2018]. Each trial collects the data of 10 UEs throughout 2000 timesteps lasting 1 ms each. A timestep in our simulation has a time length of 1 TTI and is structured as represented in Figure 1. The main parameters of the simulation are listed in Table 1.

Parameter	W	I	L	R	B^{max}	G	ρ	ζ
Value	10 timesteps	1 ms	100 TTIs	720 KHz	32 kbytes	138 RBGs	4 RBs	0

Table 1. Simulation parameters.

We consider a base station with 100 MHz of bandwidth, resulting in 138 RBGs. The packet arrival for a UE in each timestep is dictated by a Poisson distribution with mean μ_s, where $s \in \mathcal{S}$ is the assigned slice to the UE. Our evaluation scenario considers

[1]https://github.com/LABORA-INF-UFG/paper-DGWCAMK-2024
[2]https://quadriga-channel-model.de/

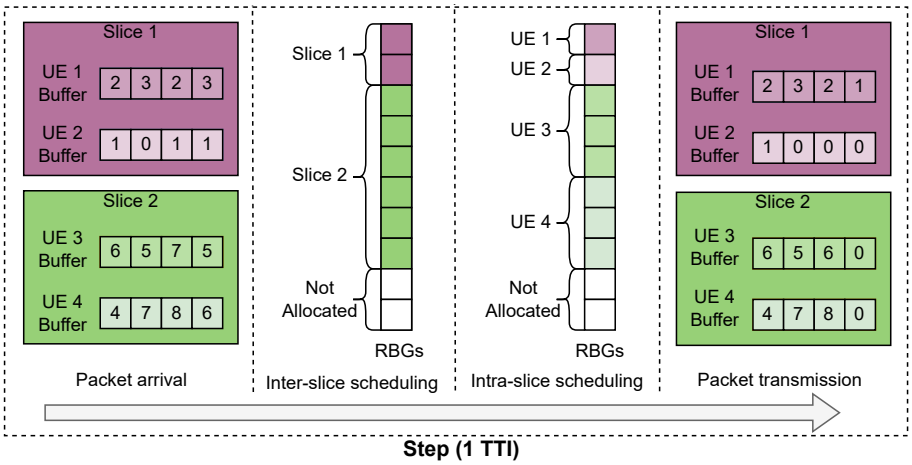


Figure 1. Example of one timestep in a scenario with 10 RBGs and 2 slices (2 UEs in each). One RBG is enough for a UE to send 2 packets during 1 TTI. Note that UE 1 will drop a packet when advancing the step due to reaching the maximum buffer latency (3 TTIs).

three different slices: an eMBB slice, a URLLC slice, and a BE slice. The slices are instantiated following the parameters of Table 2. Each slice has a round-robin algorithm as its intra-slice scheduler, as in [Nahum et al. 2023].

Slice type	eMBB	URLLC	BE
Requirements	t^{req}_{urllc} 10 Mbps l^{req}_{embb} 20 ms p^{req}_{embb} 20%	t^{req}_{embb} 1 Mbps l^{req}_{urllc} 1 ms p^{req}_{urllc} 0.001%	f^{req}_{be} 2 Mbps g^{req}_{be} 5 Mbps
$\|\mathcal{U}_s\|$	3	3	4
Z_s	1500 bytes	500 bytes	1500 bytes
μ_s	15 Mbps	1 Mbps	15 Mbps

Table 2. Slice parameters.

Figure 2 illustrates the spectral efficiency of the trial used in our evaluation scenario. Since SOA assures the SLAs of all UEs, the ones with the worst channel conditions will be allocated more RBGs. To do this, the SOA must schedule more resources to every UE in the slice, since we consider a round-robin intra-slice scheduler. Therefore, the dynamic of SOA's scheduling follows Figure 2b instead of Figure 2a.

4.2. Baselines

To assess the quality of the SOA for the inter-slice scheduling problem, we compare its performance to two baselines: a heuristic and a state-of-the-art DRL scheduler. The first is the weighted round-robin (RR) scheduler, which distributes resources uniformly among slices based on their assigned weights. The second is the DRL agent introduced by [Nahum et al. 2023], trained using the same Soft Actor-Critic algorithm of the original work. Both baselines rely on weights that define the priority of each slice type, such as eMBB, URLLC, and BE. In the RR algorithm, the weights indicate the priority of slices

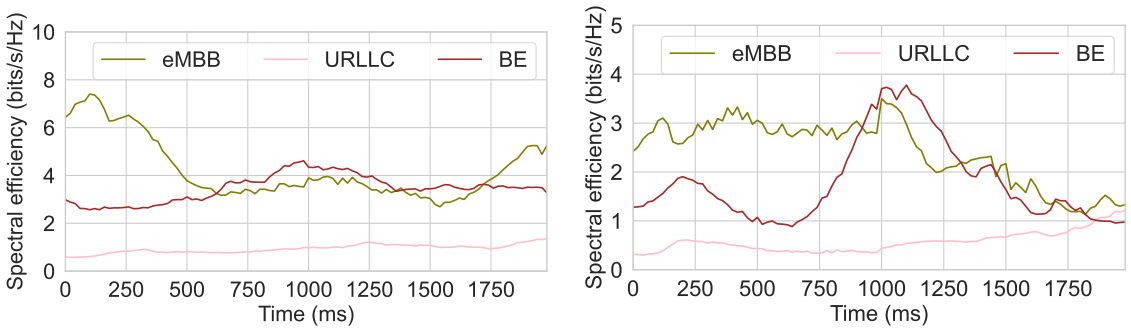


Figure 2. Spectral efficiency in the evaluated scenario.

in receiving RBGs and the proportion of the scheduling. In our scenario, the proportion in RR scheduling is 30% for URLLC, 30% for eMBB and 40% for BE, as the first two have 3 UEs and the latter has 4 UEs.

The weights used by the DRL agent, however, determine not the RBG allocation itself, but the impact of respecting the SLA requirements. In the original work [Nahum et al. 2023], for each SLA requirement, an intent-drift is calculated normalizing the distance to the required value if the requirement is not respected, otherwise, it is zero. Each intent-drift is then multiplied by a normalized weight associated with the SLA requirement. The authors consider the same five distinct SLA requirements we described in Section 2. We denote their weights as $\{w_s^t, w_s^l, w_s^p, w_s^f, w_s^g\}$, corresponding to the weights for served throughput, average buffer latency, packet loss rate, fifth-percentile throughput, and long-term throughput requirements of a slice s, respectively. Therefore, we use the same values from [Nahum et al. 2023]: $w_{eMBB}^t = 0.2$, $w_{eMBB}^l = 0.05$, $w_{eMBB}^p = 0.05$, $w_{URLLC}^t = 0.1$, $w_{URLLC}^l = 0.25$, $w_{URLLC}^p = 0.25$, $w_{BE}^f = 0.05$, and $w_{BE}^g = 0.05$. The agent's reward function is then defined as the summation of the product between each intent-drift and the weight, a sum that must be minimized.

Besides the reward, the DRL agent is defined by two other components: (*i*) the action space, which represents the possible schedulings for the agent, determined in the original work as a number for each slice, and (*ii*) the observation space, which describes the state of the environment and is used as input for the agent to select the action that minimizes the intent-drift. In [Nahum et al. 2023], two observation space strategies are defined: (*i*) the limited observation space, which includes the SLA requirement values and 9 metrics for each slice, calculated as the average for its UEs, and (*ii*) the full observation space, which includes also the 9 metrics for each UE. Between the metrics are the spectral efficiency and the 5 metrics used in the intent-drift calculation. However, as highlighted by [Nahum et al. 2023], both observation spaces demonstrate similar performance. Consequently, we adopt the limited observation space to streamline the training complexity. The training process for the DRL agent follows the approach described by [Nahum et al. 2023], where we create a training dataset with 45 trials from the spectral efficiency dataset and the agent undergoes training 10 times over the 2000 timesteps in each trial. This results in a total of 900,000 training steps. One trial, not included in the training dataset, is reserved for evaluating the three schedulers.

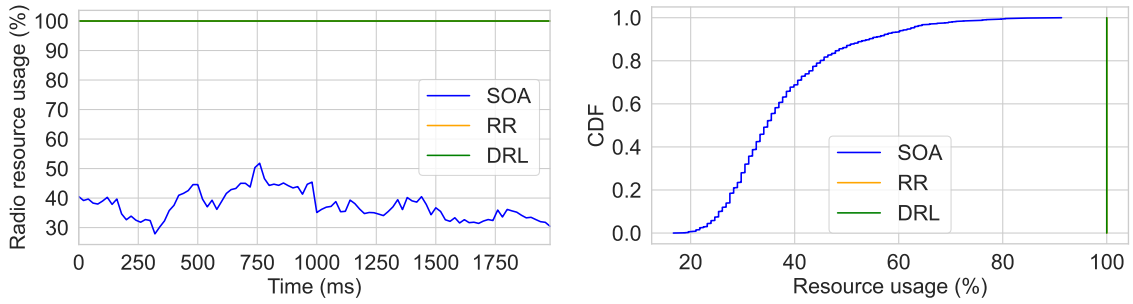
It is important to notice that the DRL agent was originally evaluated in a different scenario, where assuring the SLA for every UE is not possible. We use a different number of UEs per slice, higher spectral efficiencies, and lower buffer sizes, and we do

not consider changes in the traffic or requirements during the experiment. The number of RBGs is also higher, since we divide the resources into RBs with a lower bandwidth to follow the specifications of [ETSI 2020a]. Hence, the comparisons we perform consider the scenario of our evaluation and differ from the results in [Nahum et al. 2023].

4.3. Results

To evaluate SOA solutions, we consider two scenarios, a standard and a limited scenario. In the standard scenario, the SOA allocates the minimum RBGs necessary, while RR and DRL schedule all available RBGs in the BS at each timestep. This is a consequence of how the two baselines are formulated to not consider minimizing resource usage. While this strategy offers lower complexity for scenarios with lower demands, it does not prioritize radio resource optimization, resulting in the usage of more resources than is strictly necessary, difficulting the comparison of SOA with the baselines. Therefore, we consider a limited scenario where the number of available RBGs in the BS dynamically changes to the minimum needed to assure the SLA of all UEs at each timestep, defined by the SOA resource usage. In this scenario, we emphasize the significance of considering RBG minimization and how it can be difficult for the baselines to ensure SLA requirements for each UE with fewer resources.

Standard scenario – In this scenario, we compare the RBG usage for all models. Figure 3 depicts the radio resource usage for the RR algorithm, the DRL agent, and SOA across all timesteps. It is important to observe that, while the RR algorithm and the DRL agent utilize all radio resources in every timestep (lines are overlapped at 100%), our SOA dynamically adjusts the radio resource usage based on the current demand. Consequently, we achieved an average reduction of 62% in radio resource usage over the 2000 timesteps, while the worst-case demand, around 750 ms, has a peak of 91% allocated resources.



(a) Radio resource usage as a function of time **(b) CDF of radio resource usage**

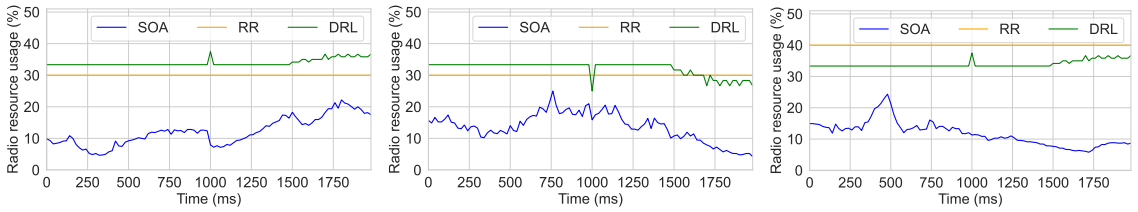
Figure 3. Radio resource usage of the schedulers in the standard experiment.

Although both baselines utilize all available radio resources for each timestep, it is important to note that neither method explicitly prevents the violation of slice requirements. Despite the DRL agent incorporating an intent-drift in its formulation, penalizing the agent when slice requirements are broken, this approach does not entirely eliminate the possibility of such faults occurring. To quantify SLA violations, we count the instances where UEs fail to meet the specified requirements for each slice at every timestep. For instance, if the URLLC slice receives no resource allocation throughout all 2000 timesteps, the served throughput requirement is violated for each of its 3 UEs, resulting in a total of $3 \times 2000 = 6000$ instances.

As expected, the SOA respected all SLAs throughout the 2000 timesteps. The same happened to the RR algorithm but with a less efficient resource utilization. Despite

also allocating 100% of the resources, the DRL agent performs worse than RR and exhibits SLA violations during the experiment. It especially violates the served throughput requirement for the URLLC slice a total of 210 times. This is due to the set of actions chosen by the agent throughout the simulation: (*i*) equal distribution for all slices, chosen in 1930 timesteps, and (*ii*) 50% for eMBB, 0% for URLLC, and 50% for BE, chosen in 70 timesteps and thus violating $70 \times 3 = 210$ times this SLA requirement.

We also analyze the radio resource allocation for each slice during the experiment. Figure 4 illustrates the resource allocation for eMBB, URLLC, and BE slices, comparing the RR algorithm, the DRL agent, and our SOA. As expected, the RR algorithm exhibits a static radio resource allocation among all slices during all timesteps due to its formulation. A similar behavior is observed in the DRL agent allocation. As it chooses an equal distribution across 1930 timesteps, the overall allocated resources are mostly constant. Its only fluctuations in allocation happen around 1000 ms and after 1400 ms, when the channel conditions for eMBB and BE are worse, therefore they are prioritized. This is the reason why URLLC gets no resources from the DRL agent in some timesteps.

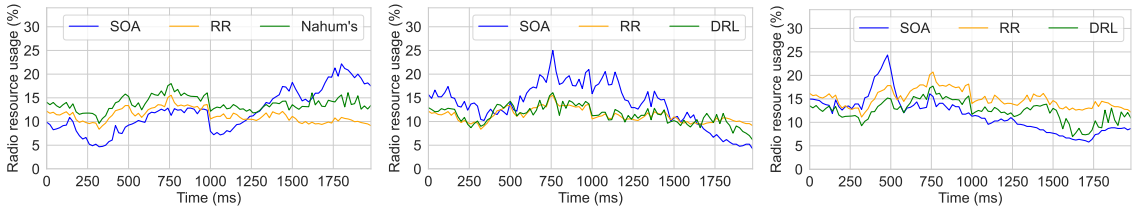


(a) eMBB **(b) URLLC** **(c) BE**

Figure 4. Allocated radio resources for each slice in the standard experiment.

Limited scenario – In this scenario, we use the SOA's minimal resource allocation represented in Figure 4 as the total number of RBGs available at every timestep to the RR algorithm and the DRL agent. It is noteworthy that the DRL agent was trained to use 100% resources, so it may not have learned how to solve scarce scenarios.

Figure 5 illustrates the radio resource allocation for each slice during this experiment. Comparing the slices, we can see that both RR and DRL allocate more RBGs to eMBB and BE than the SOA's optimal solution at most of the time, while the inverse happens to URLLC. The DRL agent consistently distributes resources equally among the slices for the majority of the experiment. This allocation strategy is similar to the RR algorithm, where the allocation is determined as a constant fraction of the available RBGs at each timestep. Additionally, we observe that the SOA strategy to allocate resources among the three slices has a highly different dynamic from the baselines. This occurs as the SOA decision is based on channel and buffer conditions, thus leading to an adaptable solution responding to the current demand.



(a) eMBB **(b) URLLC** **(c) BE**

Figure 5. Allocated radio resources for each slice in the limited experiment.

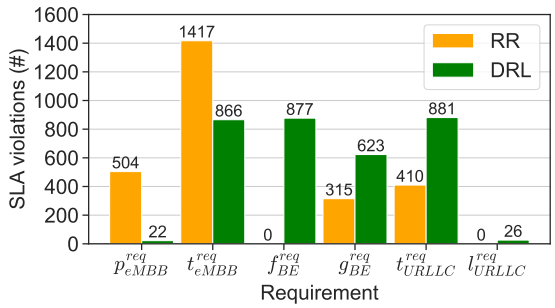


Figure 6. SLA violations in the limited experiment. SOA has no SLA violation.

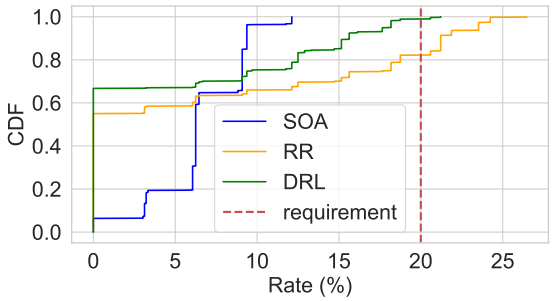
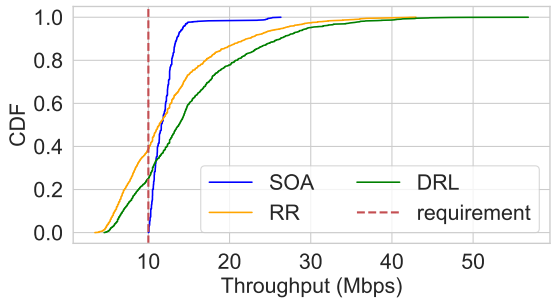
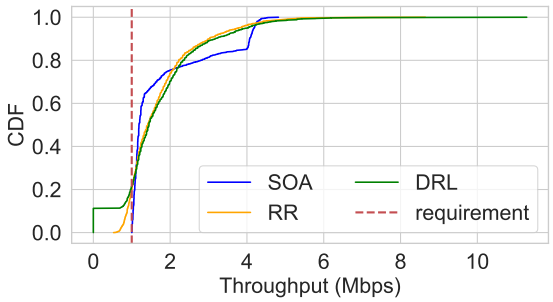


Figure 7. CDF of worst packet loss rate for eMBB in the limited experiment.



(a) eMBB



(b) URLLC

Figure 8. CDF of worst served throughput in the limited experiment.

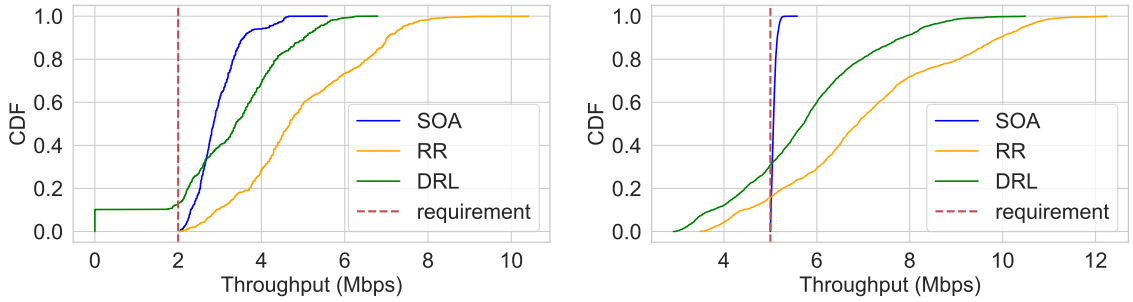
As expected, the RR algorithm and the DRL agent solutions in scarce scenarios are worse, as shown by the SLA violations in Figure 6. We can see that as the DRL agent allocation for eMBB is higher than RR's, it has fewer SLA violations for the requirements of this slice. The same does not happen to BE requirements, which are more respected by the RR algorithm since the slice has a higher weight. We recall the BE intent-drifts having the lowest weights in the DRL's reward as a reason for not prioritizing it. Lastly, although the URLLC allocation is similar between the baselines, the DRL agent performs worse. This happens due to the DRL agent scheduling zero resources to the slice in a few timesteps.

As eMBB has larger packets and a higher demand than URLLC, if high throughput is not achieved, it leads to high packet losses. Figure 7 illustrates the CDF for the worst eMBB packet loss, calculated as the maximum packet loss for an eMBB UE in each timestep. We note that the DRL agent maintains zero packet loss most of the time due to over-provisioning, but violates the requirement when the channel quality is low.

Figure 8 depicts the worst served throughput for eMBB and URLLC. Again, the DRL agent has a better performance than the RR algorithm regarding the eMBB slice. However, we can see that URLLC has a throughput of 0 Mbps for the DRL scheduling during a considerable portion of the simulation. This is explained by its set of selected actions: (*i*) equal distribution for the three slices, (*ii*) 100% for eMBB, (*iii*) half for eMBB and half for BE, and (*iv*) half for eMBB and half for URLLC. An option where no resource is scheduled to URLLC and BE is selected in 11% and 4%, where eMBB always receives at least 33% of the available RBGs.

The BE performance is represented by Figure 9. Since the fifth-percentile throughput is calculated as the minimum throughput for our time window of 10 timesteps,

scheduling no RBGs to BE impacts the actual timestep and the next 9 ones. This explains why the DRL agent has a value of 0 Mbps for the metric almost 10% of the time. The long-term throughput does not achieve such low values as it is more tolerant to fluctuations in allocation. This metric also exemplifies how SOA respects the SLA of each UE while reducing resource usage, as it allocates exactly the required value at every timestep.



(a) Fifth-percentile throughput **(b) Long-term throughput**

Figure 9. CDF of worst metrics for BE in the limited experiment.

Guidelines for RL solutions – A critical aspect of RL methods is designing an efficient reward function. Defining a constant weight for summing different intent-drift may be a solution for prioritizing slices, but it also may lead to SLA violations. For example, when the DRL agent allocates half resources for eMBB and half for BE, it chooses to disrespect a requirement of weight $w^t_{URLLC} = 0.2$ while assuring the requirements of weight $w^g_{BE} + w^t_{eMBB} = 0.25$. Another crucial aspect of RL solutions is defining the observation space. Using the average spectral efficiency for a slice may invisibilize the heterogeneous channel conditions of UEs in a slice. A scenario with two UEs u_1 and u_2 where $E_{u_1} = 0.5$ and $E_{u_2} = 5.5$ bits/s/Hz will be equivalent to another where $E_{u_1} = E_{u_2} = 3$ bits/s/Hz, as both scenarios have the same average spectral efficiency. Lastly, much of the resource usage can be reduced if the agent is designed to not allocate every RBG, as seen in SOA's allocation. An improvement can be made in the DRL agent by adding an action value for the not-used resources and then adding it as part of the reward function.

5. Conclusions and future works

In this work, we presented the stepwise optimal inter-slices RRS for SLA assurance problem. We described SOA, a polynomial algorithm that solves this problem by ensuring the SLA requirements of each UE and minimizing resource allocation. Our solution is used to evaluate a state-of-the-art DRL scheduler in a scenario where every SLA can be ensured. We showed that the ML approach may fail in ensuring the SLA of every UE even with a less efficient resource utilization. Moreover, the SOA achieves zero SLA violations while reducing resource usage by an average of 62%. We expect that new ML solutions could leverage our strategy as a baseline to improve their approaches. Lastly, our future works include expanding the scenarios where SOA can be used to address competition and prioritization when resources are scarce.

Acknowledgements

This work was supported by CAPES, MCTIC/CGI.br/São Paulo Research Foundation (FAPESP) through the Project Smart 5G Core And MUltiRAn Integration (SAMURAI) under Grant 2020/05127-2, by CNPq through the Project Universal under Grant 405111/2021-5, by RNP/MCTIC, Grant No. 01245.010604/2020-14, under the 6G Mobile Communications Systems project, and Program OpenRAN@Brasil.

References

Chen, Y., Yao, R., Hassanieh, H., and Mittal, R. (2023). Channel-Aware 5G RAN Slicing with Customizable Schedulers. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1767–1782.

ETSI (2020a). 5G; NR; Physical channels and modulation (3GPP TS 38.211 version 16.2.0 release 16). Technical report, European Telecommunications Standards Institute (ETSI).

ETSI (2020b). ETSI TS 136 213. LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Layer Procedures (3GPP TS 36.213 version 15.10.0 Release 15). Technical report, European Telecommunications Standards Institute (ETSI).

Heath Jr, R. W. and Lozano, A. (2018). *Foundations of MIMO communication*. Cambridge University Press.

Jaeckel, S., Raschkowski, L., Börner, K., and Thiele, L. (2014). Quadriga: A 3-d multi-cell channel model with time evolution for enabling virtual field trials. *IEEE transactions on antennas and propagation*, 62(6):3242–3256.

Khodapanah, B., Awada, A., Viering, I., Barreto, A. N., Simsek, M., and Fettweis, G. (2020). Framework for Slice-Aware Radio Resource Management Utilizing Artificial Neural Networks. *IEEE Access*, 8:174972–174987.

Kokku, R., Mahindra, R., Zhang, H., and Rangarajan, S. (2012). NVS: A Substrate for Virtualizing Wireless Resources in Cellular Networks. *IEEE/ACM Transactions on Networking*, 20(5):1333–1346.

Lotfi, F., Afghah, F., and Ashdown, J. (2023). Attention-based Open RAN Slice Management using Deep Reinforcement Learning. arXiv:2306.09490 [cs, eess].

Mei, J., Wang, X., Zheng, K., Boudreau, G., Sediq, A. B., and Abou-Zeid, H. (2021). Intelligent Radio Access Network Slicing for Service Provisioning in 6G: A Hierarchical Deep Reinforcement Learning Approach. *IEEE Transactions on Communications*, 69(9):6063–6078.

Mondal, B., Thomas, T. A., Visotsky, E., Vook, F. W., Ghosh, A., Nam, Y.-H., Li, Y., Zhang, J., Zhang, M., Luo, Q., et al. (2015). 3D channel model in 3GPP. *IEEE Communications Magazine*, 53(3):16–23.

Nahum, C. V., Lopes, V. H., Dreifuerst, R. M., Batista, P., Correa, I., Cardoso, K. V., Klautau, A., and Heath, R. W. (2023). Intent-aware Radio Resource Scheduling in a RAN Slicing Scenario using Reinforcement Learning. *IEEE Transactions on Wireless Communications*, pages 1–1.

Polese, M., Bonati, L., D'Oro, S., Basagni, S., and Melodia, T. (2022). ColO-RAN: Developing Machine Learning-based xApps for Open RAN Closed-loop Control on Programmable Experimental Platforms. *IEEE Transactions on Mobile Computing*, pages 1–14.

Polese, M., Bonati, L., D'Oro, S., Basagni, S., and Melodia, T. (2023). Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges. *IEEE Communications Surveys & Tutorials*, 25(2):1376–1411.

Zhu, Q., Wang, C.-X., Hua, B., Mao, K., Jiang, S., and Yao, M. (2021). 3GPP TR 38.901 channel model. In *the wiley 5G Ref: the essential 5G reference online*, pages 1–35. Wiley Press.