The Cost of Learning: Efficiency *vs.* Efficacy of Learning-Based RRM for 6G

Seyyidahmed Lahmer[*], Federico Chiariotti[*†], Andrea Zanella[*]
[*]Department of Information Engineering, University of Padova, via G. Gradenigo 6B, 35131 Padova, Italy
[†]Department of Electronic Systems, Aalborg University, Fredrik Bajers Vej 7C, 9220 Aalborg Øst, Denmark
Emails: {federico.chiariotti, seyyidahmed.lahmer, andrea.zanella}@unipd.it

***Abstract*—In the past few years, Deep Reinforcement Learning (DRL) has become a valuable solution to automatically learn efficient resource management strategies in complex networks. In many scenarios, the learning task is performed in the Cloud, while experience samples are generated directly by edge nodes or users. Therefore, the learning task involves some data exchange which, in turn, subtracts a certain amount of transmission resources from the system. This creates a friction between the need to speed up convergence towards an *effective* strategy, which requires the allocation of resources to transmit learning samples, and the need to maximize the amount of resources used for data plane communication, maximizing users' Quality of Service (QoS), which requires the learning process to be *efficient*, i.e., minimize its overhead. In this paper, we investigate this trade-off and propose a dynamic balancing strategy between the learning and data planes, which allows the centralized learning agent to quickly converge to an efficient resource allocation strategy, while minimizing the impact on QoS. Simulation results show that the proposed method outperforms static allocation methods, converging to the optimal policy (i.e., maximum efficacy and minimum overhead of the learning plane) in the long run.**

***Index Terms*—Resource allocation, Reinforcement learning, Cost of learning, Edge networking, Network slicing.**

I. Introduction

The use of Artificial Intelligence (AI) in communication networks has become pervasive with the transition from 4G to 5G, and learning is at the core of the 6G standardization process [1]: mobile networks have become exponentially more complex, with multiple QoS targets and extremely fast dynamics, and computational and energetic considerations have become inextricable from communications with the rise of green networking [2] and Mobile Edge Computing (MEC) [3]. The 6G vision acknowledges that hand-designed resource allocation strategies are not up to the challenge of managing all these elements, proposing the use of DRL as an adaptable and robust alternative for network orchestration [4] and resource allocation [5], along with a variety of other optimization tasks. DRL's *effectiveness* in dealing with complex scenarios is well-established: these agents can find foresighted policies aiming for long-term objectives [6], significantly improving network performance after the DRL agent has been trained.

However, large and complex DRL models, such as those required to control modern communication networks, are also computational processes that require non-negligible computational [7], transmission [8], and energetic resources. Local training at the edge puts a significant strain on MEC nodes with limited energetic and computational budgets, and offloading training to the Cloud incurs a significant communication overhead. As the use of DRL in 6G is aimed at optimizing the allocation of communication and computational resources, not considering the *cost of learning* might lead to suboptimal outcomes [9]. In this sense, their *efficiency* becomes questionable: the performance during the training process might be affected by the cost of the process itself, particularly for more complex models and agents, leading to a trade-off between it and effectiveness at convergence.

Online training, which is necessary in environments with time-varying statistics, is particularly vulnerable to this issue [10], as the DRL agent needs to keep training to update its policy to the changes in the environment. In our previous work [11], we considered the trade-off between effectiveness and efficiency of Cloud-based DRL training by optimizing the resource allocation between the data plane, i.e., the network resources allocated to the end users, and the *learning plane*, i.e., the portion of the network resources used by the learning process. However, in that work, we considered a static resource allocation to the learning plane, thus fixing the overhead cost of the learning process on the system and, at the same time, fixing the speed at which the DRL could converge towards an efficient strategy for the allocation of the remaining resources to the users.

In this work, we go beyond a simple static allocation and define a more general learning plane control loop, which decides the allocation of resources based on a greedy optimization strategy. This dynamic resource allocation is computationally simple, and maintains a similar efficiency to fixed allocation strategies, while reaching the same effectiveness as ideal out-of-band (i.e., with negligible overhead) approaches after the training period is over. We demonstrate the performance of the proposed framework in a simple network slicing scenario, showing that the dynamic approach leads to significantly better performance after an initial transition. To be noticed that the optimization framework we propose is not tied to the networking scenario, but works in any allocation problem in which the pool of resources that the DRL agent needs to

This work was supported by the EU H2020 MSCA ITN project Greenedge (grant no. 953775), by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, under the REDIAL Young Researchers grant and the partnership on "Telecommunications of the Future" (PE0000001 - program "RESTART"), and by the Villum Foundation, Denmark, under Villum Investigator grant "WATER."

allocate to the users is also required for the training of the DRL agent itself.

The rest of this paper is divided as follows: first, Sec. II presents the system model and the learning plane optimization. The slicing use case is then described in Sec. III, while Sec. IV presents the simulation results. Finally, Sec. V concludes the paper and presents some possible future work on the subject.

II. System Model

Let us consider a generic resource allocation problem, which is modeled as an infinite horizon Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathbf{P}, R, \gamma)$: $\mathcal{S}$ represents the state space, $\mathcal{A}$ is the action space (which is potentially different for each state), $\mathbf{P} \in [0,1]^{|\mathcal{S}|\times|\mathcal{A}|\times|\mathcal{S}|}$ is the transition probability matrix, which considers both the state and the action chosen by the agent, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. The ultimate objective of a DRL agent is to find the optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$, which maximizes the expected long-term reward:

$$\pi^* = \underset{\pi:\mathcal{S}\to\mathcal{A}}{\arg\max}\, \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(\mathbf{s}_t, \pi(\mathbf{s}_t), \mathbf{s}_{t+1})\right]. \tag{1}$$

Time is divided in slots, and the slot index is denoted by $t \in \mathbb{Z}^+$. Let us assume that, in each time slot t and independently from the state $s \in \mathcal{S}$, the system can allocate N resource blocks, which may represent communication bandwidth, computational cycles, or energy, depending on the specific application: the type of resource may affect the definition of the specific MDP, but is irrelevant at this point. In general, we refer to *requests* in the following: a request may be a packet that needs to be transmitted, a computing job that needs to be executed, or an action that requires some energy, but each request requires exactly one resource block.

The system resources are allocated among M different *slices*, where a slice may represent a single user, or a group of users with the same features. The action space then contains all possible resource allocation vectors that split the N resources among the M slices:

$$\mathcal{A} = \left\{ \mathbf{a} \in \{0, \dots, N\}^M : \sum_{m=1}^{M} a_m = N \right\}. \tag{2}$$

Furthermore, we assume that each slice is associated to a First-In First-Out (FIFO) queue of requests: each queue has a limited size Q, after which the system starts dropping older requests for that slice to make room for newer ones.

In this work, we focus on Key Performance Indicators (KPIs) tied to the latency with which the requests of the different slices are served. However, the approach can be generalized to consider other metrics.

We hence indicate by $T_{m,i}$ the latency of the i-th request from slice m, which depends on the time it spends in the queue before being assigned a resource. Dropped or rejected requests have an infinite latency by definition. The i-th request from slice m is generated at time $t_{m,i}$, and age $\Delta_{m,i}(t)$ is defined as:

$$\Delta_{m,i}(t) = t - t_{m,i}. \tag{3}$$

We can then define the reward function:

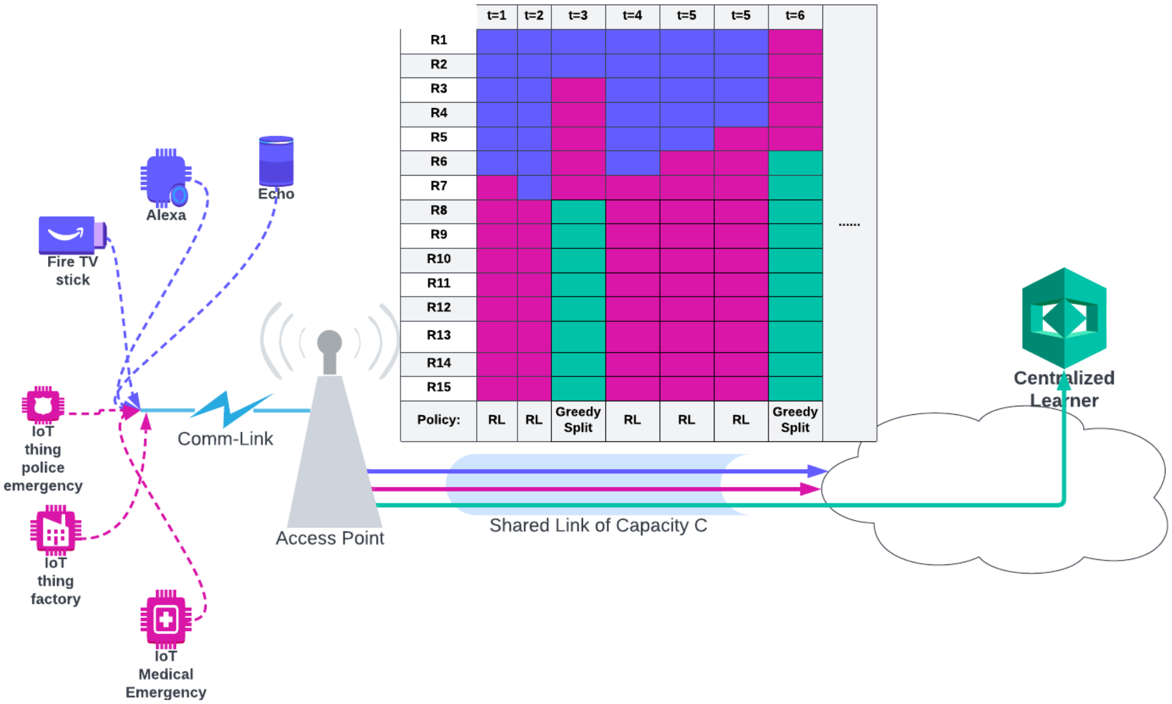
$$R(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \sum_{m=1}^{M} \sum_{i=1}^{a_m} f_m\left(\Delta_{q_m(i)}\right), \tag{4}$$

where $\Delta_{q_m(i)}$ is the age of the packet in position i of the m-th queue at the current time t, and $f_m : \mathbb{N} \to [0, 1]$ is a function mapping the latency of each request to slice m's resulting QoS. With a slight abuse of notation, we define $f(\varnothing) = 0$, where $\varnothing$ indicates that there is no packet in that position in the queue. We can distinguish between slices with *hard* timing requirements, for which the QoS of a request is 1 if it is served within a maximum latency, and 0 if it exceeds that deadline; and *soft* timing requirements, for which the QoS is a generic monotonically decreasing function of the latency. It should also be noted that dropped or rejected requests do not generate any rewards, as they are never included in the sum. The state of the system is then represented by the age of each request contained in each queue, so that in the most general case, $\mathcal{S} = (\{\varnothing\} \cup \mathbb{N})^{M\times Q}$.

The objective of the learning agent is then to learn how to allocate resources among users, so as to maximize their QoS parameters; it should also be aware of the slices that have a higher risk of violating hard timing requirements and schedule resources to avoid missing deadlines. However, learning is also a computational process, and the DRL agent may take up some of the same resources that may be allocated to the users in order to improve its policy. As we highlighted in our previous work [11], considering the cost of learning can lead to significantly different choices, limiting the amount and type of experience samples that are selected for training: this is also true regardless of the type of resource the learning requires.

However, even our previous work only considered static policies, which set up a separate virtual channel (either divided in time or in frequency) for the learning data, strictly separating the learning and data planes. Equivalently, an agent learning how to schedule tasks in an edge server could reserve a certain percentage of computation time to self-improvement, but the amount was decided in advance. This is clearly suboptimal: intuitively, the relative returns from policy self-improvement decrease over time, as the agent gradually converges to the optimal policy. After convergence, and as long as the environment statistics are stable, the value of further improvements to the policy is zero by definition. A dynamic policy for adapting the allocation between requests and learning should then take this into account.

Furthermore, the current state of the system also needs to be taken into account: if delaying the queued requests further does not have a large impact on the QoS, the system can take away resources from the slices in order to improve the resource allocation policy, but if the impact is big, e.g., if some requests from a slice with hard timing requirements are already close to



	t=1	t=2	t=3	t=4	t=5	t=5	t=6	
R1	🟦	🟦	🟦	🟦	🟦	🟦	🟪	
R2	🟦	🟦	🟦	🟦	🟦	🟦	🟪	
R3	🟦	🟦	🟪	🟦	🟦	🟦	🟪	
R4	🟦	🟦	🟪	🟦	🟦	🟦	🟪	
R5	🟦	🟦	🟪	🟦	🟦	🟪	🟪	
R6	🟦	🟦	🟪	🟦	🟪	🟪	🟩	
R7	🟪	🟦	🟪	🟪	🟪	🟪	🟩	
R8	🟪	🟪	🟩	🟪	🟪	🟪	🟩	
R9	🟪	🟪	🟩	🟪	🟪	🟪	🟩	......
R10	🟪	🟪	🟩	🟪	🟪	🟪	🟩	
R11	🟪	🟪	🟩	🟪	🟪	🟪	🟩	
R12	🟪	🟪	🟩	🟪	🟪	🟪	🟩	
R13	🟪	🟪	🟩	🟪	🟪	🟪	🟩	
R14	🟪	🟪	🟩	🟪	🟪	🟪	🟩	
R15	🟪	🟪	🟩	🟪	🟪	🟪	🟩	
Policy:	RL	RL	Greedy Split	RL	RL	RL	Greedy Split	

Fig. 1: Schematic of the learning control loop in a communication scenario.

the deadline, they need to be prioritized, choosing immediate gains over potential future improvements.

This is particularly important for non-stationary environments, in which the coherence time of the MDP statistics is finite: in this kind of system, the learning agent needs to adapt the allocation to the changing statistics of the environment, and cannot rely on offline training, but must keep learning from experience and adapt to the changes proactively.

A. Learning Plane Resource Allocation

One of the problems of including the learning plane in the resource allocation policy is the circularity of the policy: in order to learn when to allocate resources to policy improvement, a DRL agent needs to first learn when learning is important. As the policy evolves over time and learning becomes less of a priority, this makes the reward that the agent perceives dependent on the agent's own reduced demands, making the environment non-stationary.

In order to avoid this problem, we need to set an external rule to regulate learning, so that the environment that the agent sees is stationary. We define a generic resource allocation vector space $\mathcal{Z}$ as follows:

$$\mathcal{Z} = \left\{ \mathbf{z} \in \{0, \ldots, N\}^M : \sum_{m=1}^{M} z_m \leq N \right\}. \tag{5}$$

We remark here that $\mathcal{Z}$ is not the action space, but rather a superset of it, i.e., $\mathcal{A} \subseteq \mathcal{Z}$: the definition of the action space in (2) only considers allocation that gives all of the available resources to slices. If $\sum_{m=1}^{M} z_m < N$, the remaining resources are allocated to the learning.

We can then divide time slots in two categories: in DRL slots, the DRL agent applies its current policy and allocates all the resources to the slices requesting them, while in learning slots, we use a different policy, which divides resources between the learning process and the slices. Naturally, this policy must be simpler than the one defined by the DRL problem, and should not be learning-based, to avoid the circularity problem. We also remark that learning slots are not considered as experience samples for the DRL training, as the resource allocation $\mathbf{z}$ might not be a valid action in the MDP.

Fig. 1 shows a basic schematic of the process in the communication use case: the two classes of users, corresponding to Internet of Things (IoT) and human communications, transmit over a shared link, and the resources in each time slot (which correspond to bandwidth and time resources in the uplink to the Cloud) are allocated following the dynamic division. Slots 3 and 6 in the figure are learning slots, as a significant portion of the resources is allocated to the learning plane.

We can then define a randomized selection between DRL and learning slots: in each time slot t, the learning plane can be allocated some resources with probability $\rho(t)$, which decreases linearly over time, following a similar profile to the ε-greedy policy's exploration parameter. If we consider a coherence time for the scenario of τ slots, i.e., the statistics of the environment will be approximately stationary for τ steps, so that previous experience is still useful and the optimal policy is static, we can adapt the learning curve. However, we still need to define an allocation strategy in learning slots.

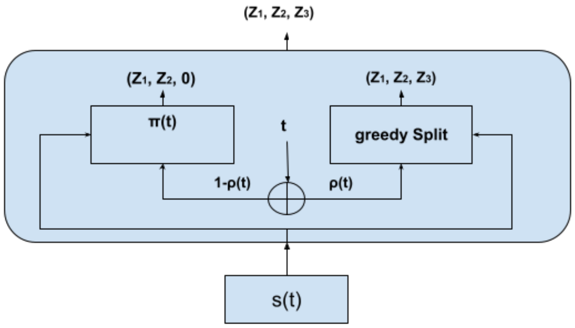


Fig. 2: Schematic of the learning plane resource allocation policy.

B. Greedy Allocation Strategy

Firstly, we define a function $\hat{R} : \mathcal{S} \times \mathcal{Z} \to \mathbb{R}$, which represents the best approximation of the instantaneous reward for each resource allocation, considering only the information available in the current state. If the f_m QoS functions are known, we can consider the following approximated reward:

$$\hat{R}(\mathbf{s},\mathbf{z}) = \sum_{m=1}^{M}\left[\sum_{i=1}^{z_m} f_m\left(\Delta_{q_m(i)}\right) - \sum_{j=z_m+1}^{L} f_m\left(\Delta_{q_m(i)}+1\right)\right]. \tag{6}$$

Naturally, this only considers the instantaneous reward, and an allocation based on this function will often lead to worse outcomes: however, this is a simple policy that can be evaluated instantly, and its optimization is relatively simple.

The second element that we must take into account when designing the policy in learning slots: after each DRL slot, we generate an experience sample, which is queued for transmission or training. Due to memory limitations, the maximum number of unprocessed samples is E, and each sample is split into ℓ packets, each of which requires one resource allocated to the learning. We then define a second function $S(\mathbf{z}, e)$, where $e \in \{0, \dots, E\}$ is the number of samples in the experience queue. The greedy strategy is then the solution to the following optimization problem:

$$\mathbf{z}^*(\mathbf{s},\mathbf{z},e) = \arg\max_{\mathbf{z}} \left(M^{-1}\hat{R}(\mathbf{s},\mathbf{z}) + E^{-1}S(\mathbf{z},e)\right). \tag{7}$$

If f_m is concave for all slices with a soft timing requirement, we can exploit the FIFO nature of the queue to provide a simple iterative solution, starting from the empty assignment and gradually assigning resources to either one of the slices or the learning process, depending on its value.

Fig. 2 shows a schematic of the full learning plane resource allocation strategy, in a simple case with $M = 2$: at each time step, the node randomly selects either the DRL agent or (with probability $\rho(t)$) the greedy allocation, which reserves some resources for the learning plane, while taking care to avoid damaging the reward.

TABLE I: Use case and learning parameters.

	Parameter	Symbol	Value
	Communication system		
	Number of subchannels	N	15
	Slot time duration	τ	1 ms
	Packet queue length	Q	1500
	Packet size	L	512 B
	Link capacity	C	7.68 MB/s
	Traffic model		
Slice 1	Total users	U_1	16
	Active user rate	R_1	512 kB/s
	Activity transition matrix	$\mathbf{O}^{(1)}$	$\begin{pmatrix}0.5 & 0.5\\0.92 & 0.08\end{pmatrix}$
	Total expected traffic	$\mathbb{E}\left[G_1\right]$	2.88 MB/s
Slice 2	Total users	U_2	17
	Active user rate	R_2	512 kB/s
	Activity transition matrix	$\mathbf{O}^{(2)}$	$\begin{pmatrix}0.5 & 0.5\\0.5 & 0.5\end{pmatrix}$
	Total expected traffic	$\mathbb{E}\left[G_2\right]$	4.35 MB/s
	Packet deadline	$T_{\max}^{(2)}$	70 ms
	Learning plane		
	Discount factor	γ	0.95
	Learning queue length	E	1500
	Packets required for each sample	ℓ	3
	Initial learning slot probability	ρ_0	0.2
	Final learning slot probability	ρ_f	0.01
	Learning slot probability decay	σ	8×10^{-4}
	Learning slot decay pace	H	1000
	Queue pressure parameter	χ_1	1400

III. Network Slicing Use Case

We consider a simple, but representative scenario, in which an Orthogonal Frequency Division Multiple Access (OFDMA) link with N orthogonal subchannels is used to transmit the data packets generated by two classes of data sources, as well as traffic on the learning plane. The scenario fits the general model presented in the previous section, as the communication resources are shared between the data and learning planes. The full parameters for the scenario, which we will describe in this section, are given in Table I.

A. Communication System Model

The number of active users in each slice is variable, making traffic non-deterministic. We consider two slices, corresponding to the two types of data sources:

- Slice 1 represents file transfer traffic, for which we do not set any strict latency constraints. However, we want the system to have the highest possible reliability to ease the burden on the higher layers. As such, $f_1(T) = 1$ for all finite values of T, but the QoS is 0 if T is infinite (i.e., if the packet is dropped);
- Slice 2 represents interactive traffic, such as video conferencing or Virtual Reality (VR) traffic, with a strict

latency deadline: packets need to be transmitted within a maximum latency $T^{(2)}_{\max}$, or the QoS drops to 0.

We consider a maximum number of active users $U_m \in \mathbb{N}$ for each slice, each of which follows a Gilbert-Elliott on-off model, which can be modeled as a binary Markov chain with transition probability matrix $\mathbf{O}^{(m)}$. In state 0, the user does not transmit, while in state 1, it transmits packets of size L with a constant bitrate R_m.

The aggregate traffic generated by slice m is then represented by the number of active users at time t $u_m(t)$, multiplied by R_m. We can define a Markov chain over $u_m \in \{0,\dots,U_m\}$, with the following transition probabilities:

$$P(u_m(t+1)=v|u_m(t)=u)=\sum_{w=\max(0,u+v-U_m)}^{\min(u,v)}(O_{11}^{(m)})^w(O_{10}^{(m)})^{u-w} \times\binom{u}{w}\binom{U_m-u}{v-w}(O_{01}^{(m)})^{v-w}(O_{00}^{(m)})^{U_m-u-v+w}. \tag{8}$$

The expected traffic G_m from slice m can be computed as:

$$\mathbb{E}\left[G_m\right]=\frac{O_{01}^{(m)}U_mR_m}{O_{01}^{(m)}+O_{10}^{(m)}}. \tag{9}$$

On the other hand, the total channel capacity is simply given by:

$$C=\frac{NL}{\tau}=7.68\text{ MB/s}. \tag{10}$$

We also consider different policies for each slice's queue: in both queues, packets that find a full queue are *rejected*, i.e., they do not enter the queue and are dropped immediately. However, in the second queue, packets still in the queue whose age is higher than the deadline are also dropped, as they do not contribute to the QoS of the slice and transmitting them would just waste resources.

B. Learning Plane

We can then define the DRL agent and learning plane optimization parameters. We used a Deep Q-Network (DQN) [12] for the agent, as the problem is simple enough not to require more advanced architectures. We also simplified the state and action space, limiting the possible resource allocation vectors to the following:

$$\mathbf{a}_{t+1}=\mathbf{a}_t+\delta_t, \tag{11}$$

with $\delta_t \in \{(1,-1),(0,0),(-1,1)\}$. In other words, the change in the allocation is at most 1 resource with respect to the previous DRL step. The outputs of the DQN correspond to the estimated long-term value of selecting each δ_t, so the network only has 3 output values. The state was also simplified: for each slice $m \in \{1,2\}$, the input to the network is given by the following values:

- The number $q_m \in \{0,\dots,Q\}$ of packets in the queue;
- The minimum latency $T_m^{\min}$ for packets transmitted in the previous slot;
- The maximum latency $T_m^{\max}$ for packets transmitted in the previous slot;
- The average latency T_m^{avg} for packets transmitted in the previous slot;
- The number d_m of dropped or rejected packets in the previous slot;
- The current number a_m of resources allocated to the slice.

TABLE II: DQN architecture.

Layer size		**Activation function**
Input	Output	
13	64	ReLU
64	32	ReLU
32	3	Linear

The values for each queue are contained in the tuple $\mathbf{s}^{(m)}=(q_m,T_m^{\min},T_m^{\max},T_m^{\text{avg}},d_m,a_m)$, to which we add another parameter: $\xi^{(2)}$, i.e., the number of packets in the queue for slice 2 that would miss their deadline if they are not transmitted in the next slot. We can define it as follows:

$$\xi^{(2)}=\sum_{i=1}^{Q} f_2\left(\Delta_{q_2(i)}\right)-f_2\left(\Delta_{q_2(i)}+1\right). \tag{12}$$

As the first slice does not have latency requirements, there are no equivalent parameters for it. The input to the DQN is then given by $\mathbf{s}^{(m)},\mathbf{s}^{(m)},\xi^{(m)}$, for a total of 13 values, which are normalized to fit in the range between 0 and 1. The full network architecture is given in Table II[1]

We then consider the learning plane optimization. First, we set the size of the experience sample queue $E=1500$, and implement an early rejection policy. When a sample is generated, its rejection probability is equal to $\frac{e}{E}$, i.e., to the current pressure on the queue. Consequently, samples that find a full queue are always rejected, but sometimes samples that could fit in the queue are dropped in favor of new experiences, avoiding too many correlated samples filling the queue.

The probability of selecting a slot as a learning slot decays linearly, starting from an initial value ρ_0 and gradually decaying to a value ρ_f. The decay is applied every H steps, after which the learning rate decreases by a constant value σ:

$$\rho(t)=\max\left(\rho_f,\rho_0-\left\lfloor\frac{t}{H}\right\rfloor\sigma\right). \tag{13}$$

Finally, we consider the greedy allocation in the learning slots. As the first slice has no latency requirements, we consider allocating resources to it greedily only when the number of packets in the queue is higher than a threshold χ_1: in this way, we avoid packet rejections, but also leave more resources for learning plane and latency-sensitive packets.

We can then define the following estimated rewards:

$$\hat{R}_1(\mathbf{s},\mathbf{z})=\min(0,z_1-\min(q_1-\chi_1,N)); \tag{14}$$

$$\hat{R}_2(\mathbf{s},\mathbf{z})=\min(0,z_2-\min(\xi_2,N)); \tag{15}$$

$$S(\mathbf{z},e)=\min(e,N)-\left(N-\sum_{m=1}^{2}z_m\right). \tag{16}$$

[1] The complete implementation of the DQN agent and dynamic resource allocation is available at https://github.com/slahmer97/costoflearning

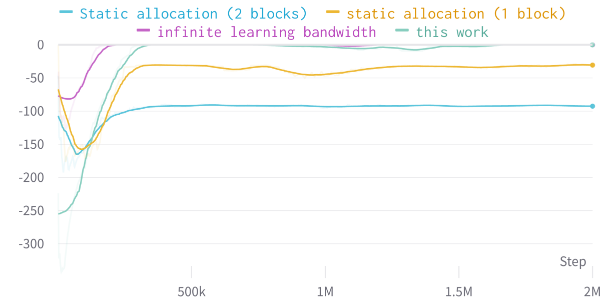


Fig. 3: Average cumulative reward per second.

The minimum operation ensures that resources will not be allocated to a slice once the queue pressure is below the limit ξ_1 or all packets with a close deadline are served, respectively. We can define the following problem:

$$\mathbf{z}^*(\mathbf{s},\mathbf{z},e) = \arg\max_{\mathbf{z}\in\mathcal{Z}} S(\mathbf{z},e) + \sum_{m=1}^{2} \hat{R}_m(\mathbf{s},\mathbf{z}). \qquad (17)$$

As the problem can easily be converted to an integer linear problem, we can easily solve it through iterative methods by assigning each resource.

IV. Simulation Results

We can then measure the performance of the dynamic learning plane resource allocation, running the DRL agent in the slicing use case for 2000 seconds. We define two benchmarks against which we compare our dynamic solution:

- *Out-of-band* learning plane: in this ideal scenario, training data is transmitted over a side channel with infinite capacity. This corresponds to the common assumption in the literature of free training;
- *Static* allocation: in this case, we set aside a fixed number of resources for the training process in each time slot. This is equivalent to the solution in our previous work [11], which considered the cost of learning but only proposed static policies.

We run two versions of the static allocation, which give 1 and 2 resources per time slot to the learning plane, respectively.

The overall reward over the training process is shown in Fig. 3: as the figure shows, the dynamic allocation initially performs worse than static allocation, but manages to converge slightly faster and maintain close to ideal performance after approximately 4×10^5 samples, corresponding to slightly more than 10 minutes of real time, while the static algorithms have a lower reward at convergence. This is also due to the fact that, unlike in our previous work, the resources allocated to the learning plane change the nature of the problem, so that resources allocated to the learning plane cannot just be transferred to the data plane: as the DQN agent trained with $N' < N$ resources, it will perform suboptimally if it is asked to allocate N resources.

We can also consider the effect of learning slots on the learning and data planes: Fig. 4 shows how many experience samples are forwarded to the Cloud during the training process. Following the linear decay of $\rho(t)$, the number of new experience samples transmitted for training is initially very high, but decreases over about 300 seconds to reach the minimum, which is between 40 and 50 samples per second. This rate is high enough to guarantee that changes in the environment statistics are captured, but does not impact the final performance, as we discussed above. Furthermore, we can analyze the impact of learning slots on the instantaneous reward by looking at the empirical Cumulative Distribution Function (CDF) of the reward penalty from using the greedy allocation, shown in Fig. 2: the reward loss is 0 in 40% of cases, and below 0.1 in 80% of cases. This means that the greedy allocation can still guarantee good performance in most cases, and as such, is a robust strategy for the learning slots. The static allocation with only 1 resource per time slot works better than with 2, even though its convergence is slightly slower, but still ends up dropping or rejecting hundreds of packets per second: slice 2 tends to suffer more, as it has more stringent requirements which need to be optimized better. On the other hand, our dynamic resource allocation manages to allocate resources as well as the ideal system, after the initial training period.

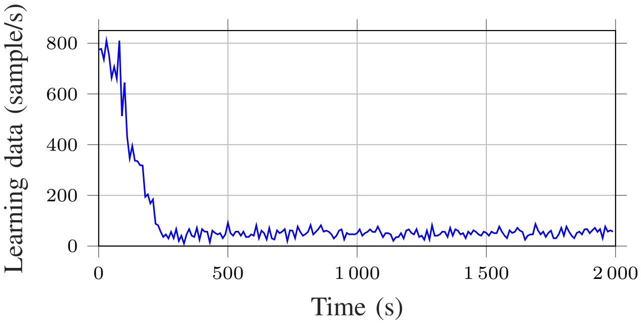


Fig. 4: Average number of forwarded learning experiences per second.

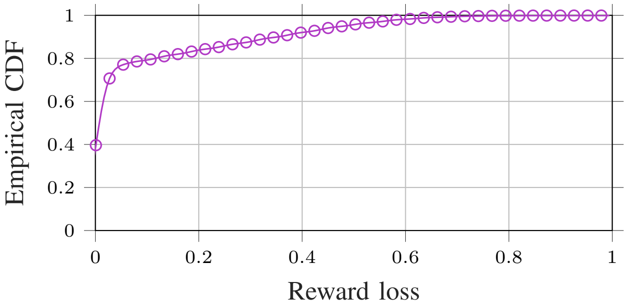
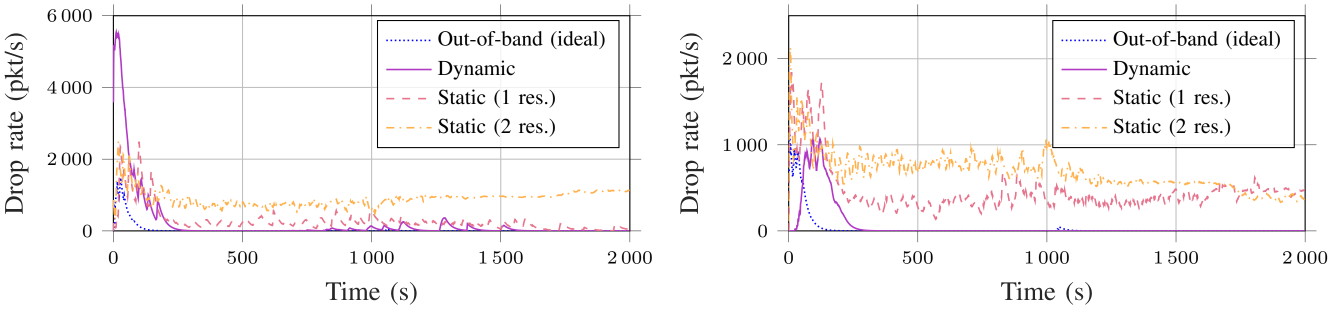


Fig. 5: Empirical CDF of the reward loss during learning slots.

We can further analyze the performance of slice 2 by considering the average latency experienced by packets, as shown in Fig. 7: the out-of-band solution quickly converges to immediately serving all packets from the slice, ensuring the lowest possible risk of violating the hard timing requirement. The dynamic resource allocation needs more time, and approaches the limit during training as several resources are taken up by the learning plane, but quickly reduces the latency



(a) Slice 1 (rejected packets).

(b) Slice 2 (rejected and dropped packets).

Fig. 6: Drop and rejection rates of each slice queue over time.

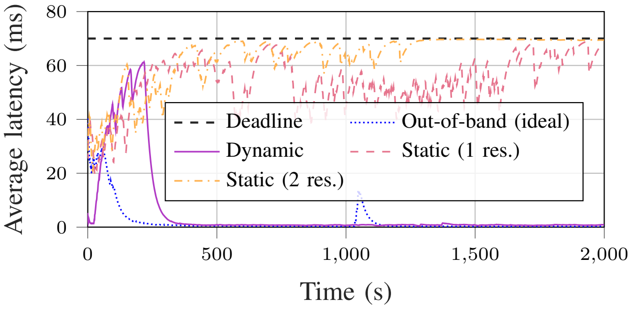


Fig. 7: Average queuing latency in slice 2.

to the same level as the ideal policy, with only minor spikes that remain below 5 ms. On the other hand, the static strategies have a latency that is close to the limit, as the limited resources available to the data plane require much riskier choices to avoid having too much of an impact on the other slice. The dynamic allocation manages to achieve better performance for both slices, even during training, only suffering an initial performance drop in the first minute of training.

V. Conclusions and Future Directions

In this work, we have designed a dynamic resource allocation policy which can mediate between the learning and data planes, controlling the trade-off between effectiveness and efficiency of DRL models. Unlike most works in the learning-based networking literature, we specifically consider the cost of learning, i.e., the resources required by the training process of a DRL agent, and shown that our dynamic policy can outperform static schemes and, after a short transition phase, match the performance of an ideal system with an out-of-band learning plane.

Possible extensions of the work include a wider deployment with more resources and a more difficult problem, with multiple slices and more stringent requirements, such as those for Ultra-Reliable Low-Latency Communications (URLLC). Another interesting theoretical question is the interplay between cost of learning and active learning: when resources in the learning plane are scarce, transmitting the most valuable samples can significantly accelerate training. In DRL, this also has an important effect on the trade-off between exploration and exploitation, which we plan to study in a future work.

References

[1] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y.-J. A. Zhang, “The roadmap to 6G: AI empowered wireless networks,” *IEEE Communications Magazine*, vol. 57, no. 8, pp. 84–90, 2019.

[2] I. Chih-Lin, “AI as an essential element of a Green 6G,” *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 1, pp. 1–3, 2021.

[3] N. Hu, Z. Tian, X. Du, and M. Guizani, “An energy-efficient in-network computing paradigm for 6G,” *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 4, pp. 1722–1733, 2021.

[4] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, “Applications of deep reinforcement learning in communications and networking: A survey,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[5] H. Sami, H. Otrok, J. Bentahar, and A. Mourad, “AI-based resource provisioning of IoE services in 6G: A deep reinforcement learning approach,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3527–3540, 2021.

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

[7] N. Shalavi, G. Perin, A. Zanella, and M. Rossi, “Energy efficient deployment and orchestration of computing resources at the network edge: a survey on algorithms, trends and open challenges,” 2022. [Online]. Available: https://arxiv.org/abs/2209.14141

[8] O. Beaumont, L. Eyraud-Dubois, and A. Shilova, “Efficient combination of rematerialization and offloading for training DNNs,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 23 844–23 857, 2021.

[9] I. Jang, H. Kim, D. Lee, Y.-S. Son, and S. Kim, “Knowledge transfer for on-device deep reinforcement learning in resource constrained edge computing systems,” *IEEE Access*, vol. 8, pp. 146 588–146 597, 2020.

[10] R. S. Villaça and R. Stadler, “Online learning under resource constraints,” in *International Symposium on Integrated Network Management (IM)*. IFIP/IEEE, 2021, pp. 134–142.

[11] F. Mason, F. Chiariotti, and A. Zanella, “No free lunch: Balancing learning and exploitation at the network edge,” in *International Conference on Communications (ICC)*. IEEE, 2022, pp. 631–636.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.