PSASlicing: Perpetual SLA-Aware Reinforcement Learning for O-RAN Slice Management

Mingrui Yin*, Yang Deng†, Ahan Kak†, Nakjung Choi†, Tao Han*
*Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ
†Nokia Bell Labs, Murray Hill, NJ
{my72, tao.han}@njit.edu; {kody.deng, ahan.kak, nakjung.choi}@nokia-bell-labs.com

GLOBECOM 2024 - 2024 IEEE Global Communications Conference | 979-8-3503-5125-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/GLOBECOM52923.2024.10901152

***Abstract*—Network slicing has been widely recognized as one of the flagship use cases for Open Radio Access Network (O-RAN), enabling the provisioning of isolated network services over a shared physical infrastructure. Each slice is characterized by a set of distinct service level agreements (SLAs) tailored to meet the needs of various industries and applications. At the same time, industry-critical applications often require strict adherence to the SLA even in the worst-case scenarios. However, existing network slicing strategies merely incorporate SLA violations as penalties within the reward function, thus failing to consistently ensure perpetual SLA compliance. To address these challenges, this paper introduces PSASlicing, an intelligent resource allocation system designed for RAN slice management across the access network. More specifically, PSASlicing introduces a new reinforcement learning algorithm for maximizing resource utilization while perpetually guaranteeing the diverse SLA requirements across slices. Furthermore, PSASlicing also incorporates a trace-driven network emulator that effectively replicates the dynamic behavior of cellular networks by integrating a transition model with real-world data from an over-the-air 5G Standalone testbed. A comprehensive experimental evaluation showcases that PSASlicing achieves an average resource savings of approximately 24.0% when compared to the state-of-the-art, while guaranteeing no SLA violations.**

I. Introduction

As cellular networks advance towards 6G, the the Open Radio Access Network (O-RAN) architecture has been leading the charge on RAN programmability and intelligence [1]. Through its introduction of the RAN Intelligent Controllers (RICs), O-RAN harbors the potential for machine learning-driven RAN management, ultimately leading to improved network performance and lower operational costs. One of the flagship use cases in the O-RAN ecosystem is RAN slicing, which allows for multiple virtual networks to be created on the same physical infrastructure, simultaneously catering to different service requirements and user demands [2]. Given the diverse range of end-user applications expected to be served by 6G, the efficient management of RAN slices is of utmost importance, and thus forms the primary focus of this work.

To that end, we identify two significant research challenges. First, we note that optimal management of radio resources across multiple base stations is necessary for achieving efficient resource utilization and ensuring compliance with the diverse requirements of network slices. Second, there is also a need for such slice management mechanisms to continuously adapt to evolving network dynamics, ensuring the continued satisfaction of service level agreements (SLAs). It bears mentioning that recent studies have laid the groundwork for navigating these challenges with promising results. In particular, Wang *et al.* [3] have utilized Deep Q-Networks for dynamic resource allocation in network slicing, demonstrating a reduction in SLA violations, while Li *et al.* [4] have introduced a hierarchical intelligent RAN slicing framework that achieves differential SLA guarantees through hierarchical decision-making. Furthermore, Zhang *et al.* [5] have presented a hybrid slicing framework combining hard and soft slicing, and employed deep reinforcement learning for resource allocation to guarantee SLAs, while Gao *et al.* [6] have adopted deep learning to predict network states and SLA demands.

The prior art cited above primarily leverages reinforcement learning (RL) algorithms, such as DQN [7], DoubleDQN [8] and DDPG [9], as part of the proposed slice management frameworks. With a view to ensuring compliance with slice or user SLAs, the aforementioned methods incorporate violations of the SLA as part of the reward function as penalties. However, these approaches may still allow for a small probability of SLA violation by design, failing to guarantee total SLA satisfaction at all times. This follows from the fact that the existing works focus on optimizing the expected reward while constraining the expected cost over random dynamics. At the same time, highlighting a significant challenge associated with RL algorithms, we note that cost in a specific episode can still be unsatisfactorily high [10], leading to practical limitations of the prior art.

To overcome the identified challenges as well as shortcomings of the prior art, through this paper, we introduce PSASlicing–a novel model-based RL framework tailored for O-RAN-driven network slice management. Our proposed framework is tailor-made for scenarios that demand strict adherence to SLAs. More specifically, we provide the following contributions.

- **Novel SLA-driven RAN Slice Management Algorithm:** The PSASlicing algorithm uniquely balances the efficient utilization of RAN resources with the assurance of diverse SLA requirements for network slices.
- **Robust Trace-driven Network Emulator:** We developed a trace-driven network emulator that effectively reproduces the dynamic behavior of cellular networks by integrating a transition model with real-world data from a 5G Standalone (SA) testbed, allowing for accurate predictions of network performance across various traffic scenarios and configurations. We utilize this emulator to assess and optimize network parameters under

diverse theoretical conditions, ensuring robust system performance by predicting latency and throughput with enhanced precision.

- **Comprehensive Performance Evaluation and Validation:** We benchmark the PSASlicing algorithm against two state-of-the-art baselines, showcasing its superior performance in enhancing SLA compliance and resource efficiency within cellular networks.

II. System Model and Problem Formulation

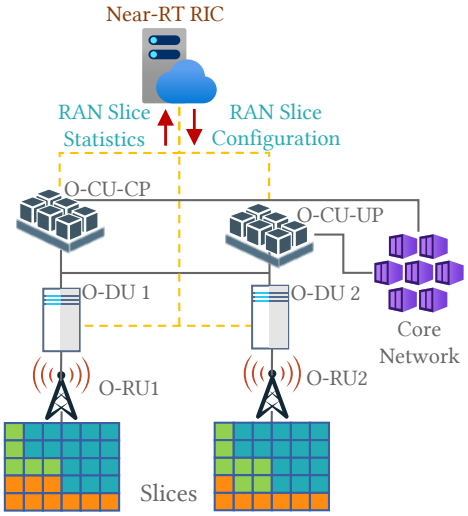

Fig. 1: System model overview.

As shown in Fig. 1, we consider an O-RAN architecture that consists of a near-Real-Time RAN Intelligent Controller (near-RT RIC) and several base stations (or gNodeBs) within a given service area that are associated with that RIC. The near-RT RIC has access to performance metrics, e.g., latency, throughput, buffer queue status, from all gNodeBs (gNBs) associated with it, and uses the PSASlicing xApp to perform RAN slice management. We assume that the same set of RAN slices is deployed across all gNBs within the service area, and that all slices have service queues that buffer the arrival traffic of their associated users.

The set of network slices is denoted by $\mathcal{I} = \{1, 2, \ldots, I\}$ and the set of gNBs is denoted by $\mathcal{J} = \{1, 2, \ldots, J\}$. For the PSASlicing xApp, we consider radio resource blocks (RBs) as the smallest resource unit available for allocation, with the total available bandwidth being denoted by B_j^{tot}. Furthermore, let $\mathcal{K} = \{1, 2, \ldots, K\}$ denote the set of possible RB assignments. Furthermore, we consider that the network is time-slotted with $\mathcal{T} = \{1, 2, \ldots, T\}$, and the PSASlicing xApp dynamically adjusts the resource allocation across slices with a predefined interval $t \in T$.

Let $\mathbf{x}_{i,j}^{(t)} = [x_{i,j,k}^{(t)} | \forall k \in \mathcal{K}]$, where $x_{i,j,k}^{(t)}$ represents k RBs allocated to the ith slice in the jth gNB. The resource allocated to network slices in the jth gNB cannot exceed the maximum amount of the radio resource, i.e.,

$$\sum_{i \in \mathcal{I}} \mathbf{x}_{i,j}^{(t)} \leq B_j^{tot}, \qquad \forall j \in \mathcal{J}, t \in \mathcal{T}, \tag{1}$$

where $B_j^{tot} = [b_{j,k}^{tot} | \forall k \in \mathcal{K}]$ represents the total amount of each resource in the jth gNB.

We note that the number of bytes that each RB can transmit depends on several factors, including the modulation and coding scheme (MCS) and channel quality (CQ). The MCS determines the number of bits that can be transmitted within each symbol of a RB, while, CQ affects the actual usable MCS level, which in turn impacts the data rate per RB. Therefore, we define a new parameter B_{RB} to represent the number of bytes an RB can transmit, as follows,

$$B_{RB} = h(\text{MCS}, \text{CQ}). \tag{2}$$

Here, h is a function whose output is the number of bytes that can be transmitted per RB based on the current MCS and CQ.

In our system, we categorize the requirements of the SLA into two parts, i.e., throughput and latency. In practical deployments, throughput is subject to the influence of several factors such as the capabilities of end-user device (or UE), environmental conditions, and noise and interference from various sources. To make our model robust and adaptable to real-world conditions, we introduce a parameter δ, which represents random and unpredictable variations, yet within a certain range. We subsequently delineate a novel cost function, $Cost$, that encapsulates the latency constraint, and employs the proposed algorithm to ensure that there are no SLA violations at any given time t. We denote that $SLA_{throughput(i,j)}$ is the minimum value of throughout SLA requirement and $SLA_{latency(i,j)}$ is the maximum value of latency SLA requirement for the ith slice in the jth gNB. The resource allocated to each network slice in the jth gNB should meet the minimum throughput requirement, while ensuring that latency does not exceed the maximum limits defined in the SLA. The throughput and latency constraints are defined in (3) and (4), respectively, as follows,

$$x_{i,j}^{(t)} \cdot (\delta_{\min} + B_{\text{RB}}) \geq \text{SLA}_{\text{throughput}}(i,j), \tag{3}$$

$$Cost \leq \text{SLA}_{\text{latency}}(i,j) \tag{4}$$

$$\forall i \in \mathcal{I}, j \in \mathcal{J}, t \in \mathcal{T}.$$

Next, the network utilization, often referred to as the occupancy rate of network resources, can be determined by the ratio of the total number of RBs allocated to all network slices at a certain time to the total number of RBs available in the entire network. The network utilization of each gNB at time t can be defined as

$$U_{i,j}^{(t)} = \frac{\sum_{i \in \mathcal{I}} x_{i,j}^{(t)}}{B_j^{\text{tot}}}. \tag{5}$$

The objective of RAN slicing is to optimize the overall network system efficiency while guaranteeing the quality-of-service (QoS). Hence, the network slicing problem $\mathscr{P}_1$ can be defined as,

$$\mathscr{P}_1: \quad \min_{\{\mathbf{x}_{i,j} \geq 0\}} \quad \sum_{t \in \mathcal{T}} \sum_{j \in \mathcal{J}} U_{i,j}^{(t)} \quad s.t. \quad (1), (3), (4). \tag{6}$$

III. RL-BASED NETWORK SLICING ALGORITHM

The major challenges of the above problem include dynamically allocating radio resources to maintain service levels in the face of unpredictable network conditions and traffic demands, ensuring compliance with strict throughput and latency SLAs, and managing finite and shared radio resources efficiently. We note that the Anytime-Competitive Markov Decision Process (A-CMDP) framework is well-suited to address these challenges due to its quick adaptability, which ensures that the resource allocation can keep pace with rapid changes in operating conditions and maintain SLA compliance. Furthermore, A-CMDP can handle multiple constraints, which is critical for balancing resource utilization and guaranteeing QoS.

A. Anytime-Competitive Markov Decision Process Definition

The Anytime-Competitive Markov Decision Process (A-CMDP) [10] is defined by the tuple $\mathcal{M}(\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{G}, T, r, c, \tau, \pi)$. In A-CMDP, each episode consists of T rounds. The network state at each round is denoted by $s_t \in \mathcal{S}$, where $t \in [T]$. At each round of an episode, the agent within the PSASlicing xApp selects an action a_t from an action set $\mathcal{A}$, which means that the agent decides how many resources to allocate to each gNB per slice. Through the PSASlicing xApp, the RAN then returns a reward $r_t(s_t, a_t)$ representing network efficiency and a cost $c_t(s_t, a_t)$ representing network latency, where $r_t \in \mathbb{R}$ and $c_t \in \mathbb{C}$. The network state dynamics are defined as $s_{t+1} = f_t(s_t, a_t)$, where $f_t \in \mathcal{F}$ is a random transition function sampled from an unknown distribution $g(f_t)$ with density $g \in \mathcal{G}$.

The agent cannot access the random function f_t directly but can observe the network state s_t at each round t. Additionally, this dynamic network model can be equivalently represented by the transition probabilities in conventional MDP models as $\mathbb{P}(s_{t+1}|s_t, a_t) = \sum_{f_t} \mathbf{1}(f_t(s_t, a_t) = s_{t+1}) g(f_t)$. A network resource allocation policy π prescribes an action a_t for each round $t \in [T]$. Under the policy π, the function $V^{\pi}t(s)$ represents the expected cumulative reward starting from round t for a given state s. The aim of A-CMDP is to maximize the expected overall reward beginning from the initial round, formulated as $Es_1[V_1^{\pi}(s_1)] = E\left[\sum_{t=1}^{T} r_t(s_t, a_t)\right]$, focusing on enhancing network system efficiency while upholding the quality-of-service (QoS) standards.

B. PSASlicing Agent Design

In line with the A-CMDP definition, we need to design a new RL model with a customized state sapce, action space, reward function and cost function.

State. The state represents the current network state. The state in the jth gNB at time interval t can be expressed as,

$$\mathbf{s}_t = \left[l_{i,j}^{(t)}, \forall i \in \mathcal{I}\right], \tag{7}$$

where $l_{i,j}^{(t)}$ is the queue length in slice i's buffer.

Action. The action at time interval t is defined as the resources allocated to each slice within the network as follows,

$$\mathbf{a}_t = \left[\mathbf{x}_{i,j}^{(t)}, \forall i \in \mathcal{I}\right]. \tag{8}$$

According to the constraints defined in (3), the action at time interval t must satisfy the minimum resource allocation requirements to uphold the throughput SLA. Specifically, each element of the action vector a_t, which represents the resource allocation to network slices, should be greater than or equal to the quotient of the throughput SLA requirement and the sum of $\delta_{\min}$ and B_{RB}. For each network slice, the action a_t must adhere to the following condition,

$$a_t \geq \frac{SLA_{\text{throughput}}(i,j)}{\delta_{\min} + B_{RB}}. \tag{9}$$

This ensures that the allocated resources per time interval do not fall below the threshold required to maintain the agreed-upon performance metrics set forth in the SLA. In certain extreme and exceptional scenarios where the sum of allocated resources in the action vector a_t surpasses the available total resources B_j^{tot} at the jth gNB, as delineated by (1), a proportional reduction strategy is implemented to ensure adherence to the resource constraints. Under this scheme, a scaling factor $\zeta \leq 1$, is computed as the ratio of the total available resources to the aggregate requested resources with $\zeta = \frac{B_j^{tot}}{\sum_{i \in I} x_{i,j}^{(t)}}$. Subsequently, each resource allocation $x_{i,j}^{(t)}$ within a_t is scaled down uniformly by ζ, yielding revised allocations $x_{i,j}^{(t)} \leftarrow \alpha \cdot x_{i,j}^{(t)}$, which cumulatively do not exceed B_j^{tot}. This approach preserves the proportionality of the original allocations while conforming to the total resource capacity of the gNB.

Reward. Since the overall objective is to maximize the long-term utility of RAN system, we define the reward at time interval t as,

$$\mathbf{r}_t(s_t, a_t) = \sum_{j \in \mathcal{J}} U_{i,j}^{(t)}. \tag{10}$$

Besides the reward, QoS is also crucial for the PSASlicing agent. Different from conventional reinforcement learning, our proposed RL algorithm necessitates the definition of a cost function, which serves as a constraint in the model.

Cost. Within the scope of our problem, the cost is defined as the network system's latency, and by imposing restrictions on the cost, we can always guarantee the SLA requirements. By comprehensively considering the current state, MCS, channel quality, and path loss, we can obtain the cost function of each time interval t as,

$$\mathbf{C}ost_t = \alpha_1 \cdot s_t^2 + \frac{\alpha_2 \cdot s_t \cdot \phi(MCS)}{\log_2(1 + \text{SNR})} \cdot e^{-\eta_d} + \alpha_3, \tag{11}$$

where α_1 and α_2 are the factors indicating the impact of queue size on latency. The function $\phi(\text{MCS})$ denotes the effect of the chosen MCS on latency, incorporating the modulation strategy's influence. The signal-to-noise ratio (SNR) reflects channel quality, and, along with the queue size and MCS selection, determines the efficiency of signal processing. The exponential term $e^{-\eta_d}$ captures the effect of path loss on signal attenuation, with costs increasing as distance grows. Finally, α_3 is a constant representing the minimum latency experienced under ideal conditions (e.g., an empty or near-empty buffer).

C. Transition Function

The state s_t is defined as the remaining demand for each round t and is updated as,

$$s_t = \left[V_s(s_{t-1}) + \sum_{m=1}^{n} \mu_{t,m} - (\delta + B_{RB}(t)) \cdot a_t \right]^+, \quad (12)$$

where $V_s(s_{t-1})$ is a function that models the state of the network buffer, taking into account the remaining queues in the buffer from the previous round. The term $\mu_{t,m}$ represents a vector corresponding to the arrival traffic from the mth user at time t, and is dimensionally consistent with the state vector s_t. The total arrival traffic at round t is given by the sum over all m UEs, represented as $\sum_{m=1}^{M} \mu_{t,m}$. In this case, we can use (2) and parameter δ to represent processed workload. The amount of workload that can be processed or served in a given round can depend on several random factors, such as resource availability, processing delays, network congestion, and other stochastic elements within the RAN infrastructure. Furthermore, the state of the network buffer can be influenced by random events like packet losses, re-transmissions, and other unpredictable factors that affect the remaining workload from the previous round. As a result, V_s is modeled as random functions because the network behavior and workload processing tend to have random or stochastic elements that are difficult to predict precisely using deterministic formulae or models.

D. The PSASlicing Algorithm

To solve the problem $\mathscr{P}_1$, we propose an RL-based network slicing algorithm. We define an augmented state x_t which includes the original state s_t, the allowed deviation $D_t \in \mathcal{D}$, and historical information $\{c_l\}_{l=1}^{t-1}$. The augmented state reflects the deviation between the policy provided by the PSASlicing agent and the safest prior policy, which represents the network system without latency deviation. This deviation can be controlled by two parameters (λ, b), thereby regulating the system's current latency to always remain below the SLA requirements. The transition of s_t is defined in (12) and needs to be learned, while the transition of $\mathcal{D}_t$ is defined in (13) and is known to the PSASlicing agent as,

$$D_t = \max\{D_{t-1} + \lambda e + b - \Gamma_{t-1,t-1} d_{t-1}, R_{t-1} + \lambda e + b\}, \quad (13)$$

where $R_{t-1} = \sum_{l=1}^{t-1} \left((1+\lambda) c_l^+ - c_l - \Gamma_{l,t} d_l \right)$ and the weight $\Gamma_{l,t} d_l$ indicates the total impact of the action deviation at round l on the sum of the cost differences from rounds t through T. The (λ, b)–anytime competitive constraints are satisfied if it holds at each round t that $\|a_t - \pi^t(x_t)\| \leq D_t$.

A baseline ML policy $\tilde{\pi}$ gives the output $\tilde{a}_t$ and the selected action is the projected action $a_t = P_{\mathcal{A}_t}(\mathcal{D}_t)(\tilde{a}_t)$. The prior policy is considered the safest policy for our purposes. Hence, in our problem, we define the set of prior policies as,

$$\tilde{\pi} = \frac{s_t + \sum_{m=1}^{M} \mu_{t,m}}{B_{RB}}. \quad (14)$$

Ideally, the network resources allocated at each step t are sufficient to process all waiting queues in the buffer, thereby eliminating any latency. This constitutes a sufficient condition that is guaranteed to satisfy the SLAs. Then, the network generates a reward $r_t(x_t, P_{\mathcal{A}_t}(\mathcal{D}_t)(\tilde{a}_t))$ and a cost $c_t(x_t, P_{\mathcal{A}_t}(\mathcal{D}_t)(\tilde{a}_t))$. Thus, the value function corresponding to the ML policy $\tilde{\pi}$ can be expressed as $V_t^{\tilde{\pi}}(x_t) = \mathbb{E}\left[\sum_{l=t}^{H} r_l(x_l, P_{\mathcal{A}_l}(\mathcal{D}_l)(\tilde{a}_l))\right]$ with $\tilde{a}_t$ being the output of the ML policy $\tilde{\pi}$.

Given an estimation of the transition distribution g^n at episode n, we perform value iteration to update $\hat{Q}$ functions for $t = 1, \ldots, T$ as,

$$\hat{Q}^n(x_t, a_t) = r_t(s_t, a_t) + \max_{\tilde{g} \in \mathcal{G}_n} \mathbb{E}_{\tilde{g}} \left[V_{t+1}^n(x_{t+1}) | x_t, a_t \right], \quad (15)$$

$$\hat{V}^n(x_t) = \max_{a \in \mathcal{A}} \hat{Q}^n(x_t, a), \quad (16)$$

$$\mathbb{E}_g \left[V_{t+1}^n(x_{t+1}) | x_t, a_t \right] = \sum_{f \in \mathcal{F}} \tilde{V}_{t+1}^n(x_{t+1}) g(f), \quad (17)$$

where $a_t = P_{\mathcal{A}_t}(D_t)(\tilde{a}_t)$, $\hat{Q}^{n+1,t}(s,a) = 0$, $V^{n+1,t}(s) = 0$, $\mathcal{G}_n$ is the confidence set based on the estimation of the transition model $\tilde{g}$. The transition model $\tilde{g}$ is estimated as,

$$\tilde{g}^n = \arg\min_{\tilde{g} \in \mathcal{G}} \sum_{l=1}^{n-1} \sum_{t=1}^{T} \left(\mathbb{E}_{\tilde{g}} \left[V_{t+1}^l(x_{t+1}) | x_t, a_t \right] - \hat{V}_{t+1}^l(x_{t+1}) \right)^2. \quad (18)$$

Based on the transition estimation, we can calculate the confidence set as

$$\mathcal{G}_n = \Bigg\{ \tilde{g} \in \mathcal{G} \Bigg| \sum_{l=1}^{n-1} \sum_{t=1}^{T} \Bigg(\mathbb{E}_{\tilde{g}} \left[V_{t+1}^l(x_{t+1}) \mid x_t, a_t \right] - \mathbb{E}_{\tilde{g}^n} \left[V_{t+1}^l(x_{t+1}) \mid x_t, a_t \right] \Bigg)^2 \leq \beta_n \Bigg\}. \quad (19)$$

E. Model Training

Algorithm 1 Model Training

1: **Initialization:** Transition model set $G_1 = \{g^1\}$.
2: **for** each episode $n = 1, \ldots, N$ **do**
3: Observe the initial state x_1^n.
4: Select $g^n = \arg\max_{g \in G_n} \mathbb{E}_g [V_1(x_1^n)]$.
5: Perform value iteration in Eqn.(15),(16),(17) and update $\tilde{Q}$ functions $\tilde{Q}_1^n \ldots, \tilde{Q}_T^n$.
6: **for** each round $t = 1, \ldots, T$ **do**
7: Initialize an allowed deviation: $D_t = \lambda e + b$.
8: Obtain the output of the ML policy $\tilde{\pi}$ as $\tilde{a}_t$.
9: Select the action a_t by projecting $\tilde{a}_t$ into the safe action set $A_t(D_t)$.
10: Update the allowed deviation D_{t+1} by Eqn.(13).
11: Observe state x_{t+1}^n and store values V_{t+1}^n.
12: **end for**
13: Update transition model g^{n+1} using Eqn.(18) and calculate confidence set G_{n+1}.
14: **end for**

The training workflow of the model is summarised in Alg. 1. We initialize the transition model set as $G_1 = \{g^1\}$. With a learned ML policy $\tilde{\pi}^k$ at each episode k, the policy used for action selection is the policy π^k. For state s_t at round t, π^k selects actions as $\pi^k(s_t) = P_{A_t(D_t)}(\tilde{\pi}^k(x_t))$.

For each round t, We first initialize an allowed deviation as $D_1 = \lambda e + b$. When the output $\tilde{a}_t$ of the ML model is obtained

at each round t, it is projected pnto a safe action set $A_t(D_t)$ depending on the allowed deviation D_t, i.e. $a_t = P_{A_t(D_t)}(\tilde{a}_t)$ where $P_{A_t(D_t)}$ is the projection onto the set defined by D_t, and a_t is the action chosen to minimize the deviation from the output $\tilde{a}_t$, formally $a_t = \arg\min_{a \in A_t(D_t)} \|a - \tilde{a}_t\|$. The allowed deviation D_t is then updated based on (13), allowing for greater or lesser flexibility in action selection based on the history of deviations $\{d_l\}_{l=1}^{t-1}$ and the resultant cumulative adjustments R_{t-1}. At the end of each episode, the transition model is refined to g^{*n+1} using (18) and the confidence set G_{n+1} is updated. This process iteratively enhances the policy and model with each episode, continuously seeking a balance between optimizing rewards and respecting constraints.

IV. Performance Validation and Evaluation

A. Experimental Environment

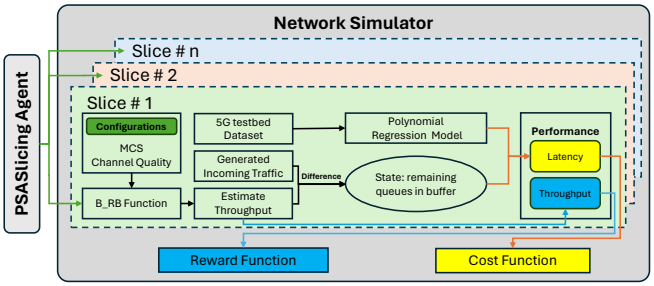

Fig. 2: The PSASlicing RAN emulator.

With a view to conducting a practical performance evaluation, we developed a trace-driven RAN emulator. The emulator is constructed upon the transition model in (12) that incorporates the B_{RB} function and a polynomial regression model, both derived from real RAN performance measurements obtained from a lab-grade 5G SA testbed. This integration of the transition model with the testbed data ensures an accurate representation of the dynamic cellular network environment. The PSASlicing agents are trained offline by using this emulator, as shown in Fig. 2.

In the emulation process, each network slice i is initially configured with specific parameters such as the MCS and channel quality. Traffic within each slice is generated following a Poisson distribution to reflect the stochastic nature of real-world network demands. The B_{RB} function is derived from empirical analysis of the 5G testbed data and governs the estimation of throughput. This function dynamically calculates throughput based on the allocation of RBs as determined by the agent's actions a_t and the prevailing network conditions.

In an endeavor to encapsulate the inherent variability of network environments, a random fluctuation dataset, symbolized by δ in (12), is integrated into the model, thereby introducing a degree of randomness reflective of in-field scenarios. The emulator juxtaposes this estimated throughput with actual traffic inflow to compute the residual queues in the buffer, which, in turn, is instrumental in predicting network latency.

As detailed above, we train the emulator using data from an over-the-air 5G SA testbed. The testbed represents an end-to-end 5G SA O-RAN network that leverages the HexRAN

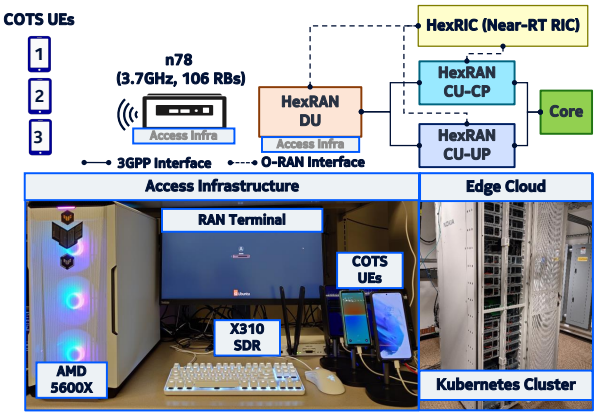

Fig. 3: Testbed setup for data collection.

project [11], as shown in Fig. 3. As shown in the figure, the testbed consists of a dissaggregated HexRAN gNB comprising the CU-CP, CU-UP, DU, and RU components. The gNB broadcasts a single 40 MHz n78 (3.7 GHz) carrier, and is connected to a 5G core network from the Open5GS project [12], along with a near-RT RIC from the HexRIC project [13]. The network hosts three slices having one user each, with varying traffic patterns. The DU is deployed on an AMD 5900X server attached to a USRP X310 SDR, while all other components are deployed on a Kubernetes cluster. Furthermore, we deploy the RANSight xApp [14] on HexRIC to assist with telemetry collection for the purpose of generating the training dataset for the emulator.

B. Performance Evaluation

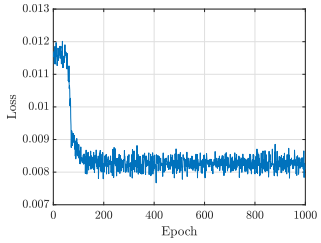

Fig. 4: Algorithm convergence.

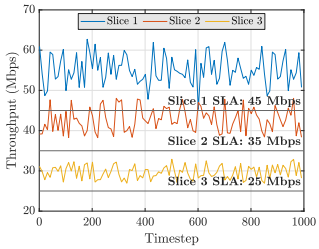

Fig. 5: Per-slice throughput.

The experimental setup involves three user equipments (UEs), each assigned to a different network slice, with incoming traffic following a Poisson distribution – UE1 in Slice 1 at 51 Mbps, UE2 in Slice 2 at 42 Mbps, and UE3 in Slice 3 at 28 Mbps. The time interval t is 50 ms and the time period T is composed of 20 time intervals. For RL training, we use a policy neural network with two hidden layers of 40 neurons each, initialized with a Gaussian distribution, trained across 2000 reinforcement learning episodes, with updates every 50 episodes using a 10^{-3} learning rate, and leveraging the Adam optimizer for weight adjustments. We select $\lambda = 2$ and $b = 4$ as our constraint parameters.

The relationship between training loss and epochs is illustrated in Fig. 4. The loss decreases significantly as the number of epochs increases, indicating that the proposed algorithm is capable of swift improvements in early training

stages. Notably, the proposed algorithm exhibits signs of convergence within the initial 100 epochs. This substantiates the effectiveness of the convergence of the proposed algorithm in learning the optimal policy. Post-convergence, the training loss stabilizes, displaying fluctuations that can be attributed to the exploration inherent in the learning process, rather than significant learning strides. Fig. 5 illustrates that PSASlicing is capable of meeting distinct SLA throughput criteria for different slices, with Slice 1 exceeding 45 Mbps, Slice 2 exceeding 35 Mbps, and Slice 3 exceeding 25 Mbps.

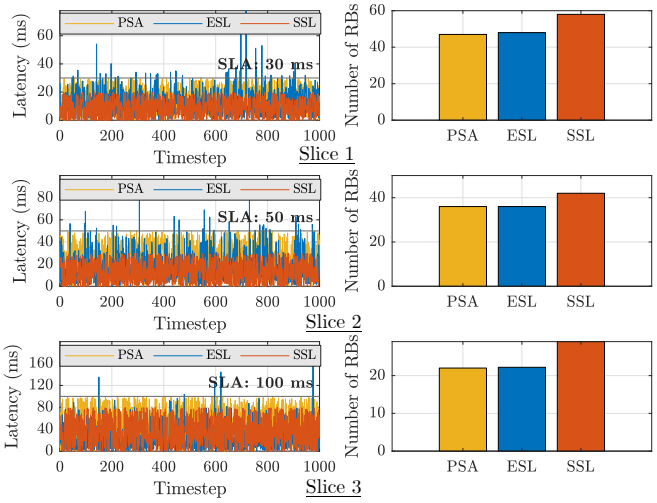

Fig. 6: Per-slice latency performance and resource allocations.

We set distinct SLA latency thresholds for each network slice in our evaluation – 30 ms for Slice 1, 50 ms for Slice 2, and 100 ms for Slice 3. We tested the capability of the PSASlicing (PSA) algorithm to allocate resources across slices with varying configurations and SLA standards, aiming to minimize the consumption of network resources while ensuring adherence to the SLA benchmarks. When compared with two baselines in Fig. 6, EdgeSlicing (ESL) [15] and SafeSlicing (SSL) [16], we found that both PSASlicing and SafeSlicing methods consistently stayed within the SLA latency thresholds.

For Slices 1, 2, and 3, SafeSlicing consumed approximately 23.40%, 16.67%, and 31.82% more network resources respectively. This follows from the fact that SafeSlicing utilizes the Lagrangian primal-dual method to incorporate SLA constraints into the reward system using multipliers. However, there exists a gap between the policy obtained from the dual function and the optimal solution. PSASlicing and EdgeSlicing exhibited similar resource usage, however, EdgeSlicing incorporated SLA considerations as part of the RL reward function, wherein SLA violations were treated merely as penalties. Consequently, it experienced SLA violations with probabilities of 3.7% for Slice 1, 2.6% for Slice 2, and 0.5% for Slice 3. Conversely, PSASlicing uses a cost function to individually represent network latency. Under the proposed algorithm mechanism, it can ensure compliance with the SLA constraint at every moment. Therefore, PSASlicing maintains no SLA violations across all slices and demonstrates exceptional performance in terms of resource efficiency.

V. Conclusion

In this paper, we have proposed a novel RL algorithm for network resource allocation that consistently ensures SLAs while optimizing network resource utilization. We developed a trace-driven network emulator that integrates a transition model with data from an over-the-air 5G Standalone testbed, allowing for precise simulations of network behavior across diverse traffic scenarios. In our experimental evaluations, we compared our algorithm with two advanced baseline algorithms. We showcased that our proposed method consistently matched or surpassed the performance of both baselines, achieving an outstanding balance across all tested metrics.

Acknowledgement

This work was partially supported by the U.S. NSF under Grants 2147623 and 2147624

References

[1] M. Polese *et al.*, "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 2, pp. 1376–1411, 2023.

[2] A. Kaloxylos, "A Survey and an Analysis of Network Slicing in 5G Networks," *IEEE Communications Standards Magazine*, vol. 2, no. 1, pp. 60–65, 2018.

[3] X. Wang and T. Zhang, "Reinforcement Learning Based Resource Allocation for Network Slicing in 5G C-RAN," in *2019 Computing, Communications and IoT Applications (ComComAp)*, 2019, pp. 106–111.

[4] J. Li *et al.*, "Hierarchical Intelligent Radio Access Network Slicing for Differential Service Level Agreement Guaranteeing," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 3, pp. 4124–4136, 2024.

[5] H. Zhang *et al.*, "A Hard and Soft Hybrid Slicing Framework for Service Level Agreement Guarantee via Deep Reinforcement Learning," in *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, 2022, pp. 1–5.

[6] S. Gao *et al.*, "Prediction-based Resource Slicing for Service Level Agreement Guarantee: A Deep Learning Approach," in *2022 31st Wireless and Optical Communications Conference (WOCC)*, 2022, pp. 68–73.

[7] V. Mnih *et al.*, "Human-level Control Through Deep Reinforcement Learning," *Nature*, vol. 518, pp. 529–33, 02 2015.

[8] H. van Hasselt *et al.*, "Deep Reinforcement Learning with Double Q-learning," *CoRR*, vol. abs/1509.06461, 2015.

[9] T. P. Lillicrap *et al.*, "Continuous Control with Deep Reinforcement Learning," in *ICLR*, 2016.

[10] J. o. Yang, "Anytime-Competitive Reinforcement Learning with Policy Prior," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 77 852–77 866.

[11] A. Kak *et al.*, "HexRAN: A Programmable Multi-RAT Platform for Network Slicing in the Open RAN Ecosystem," 2023.

[12] Open5GS Project, "Open5GS: 5G Core and EPC," https://open5gs.org/open5gs/, 2023.

[13] V.-Q. Pham *et al.*, "HexRIC: Building a Better near-Real Time Network Controller for the Open RAN Ecosystem," in *Proceedings of the 24th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '23, 2023, p. 15–21.

[14] A. Kak *et al.*, "RANSight: Programmable Telemetry for Next-Generation Open Radio Access Networks," in *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, 2023, pp. 5391–5396.

[15] Q. Liu *et al.*, "EdgeSlice: Slicing Wireless Edge Computing Network with Decentralized Deep Reinforcement Learning," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 2020, pp. 234–244.

[16] ——, "Constraint-Aware Deep Reinforcement Learning for End-to-End Resource Orchestration in Mobile Networks," in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, 2021, pp. 1–11.