2021 IEEE 29th International Conference on Network Protocols (ICNP) | 978-1-6654-4131-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICNP52444.2021.9651934

Constraint-Aware Deep Reinforcement Learning for End-to-End Resource Orchestration in Mobile Networks

Qiang Liu
University of Nebraska-Lincoln
qiang.liu@unl.edu

Nakjung Choi
Nokia Bell Labs
nakjung.choi@nokia-bell-labs.com

Tao Han
New Jersey Institute of Technology
tao.han@njit.edu

***Abstract*—Network slicing is a promising technology that allows mobile network operators to efficiently serve various emerging use cases in 5G. It is challenging to optimize the utilization of network infrastructures while guaranteeing the performance of network slices according to service level agreements (SLAs). To solve this problem, we propose *SafeSlicing* that introduces a new constraint-aware deep reinforcement learning (CaDRL) algorithm to learn the optimal resource orchestration policy within two steps, i.e., offline training in a simulated environment and online learning with the real network system. On optimizing the resource orchestration, we incorporate the constraints on the statistical performance of slices in the reward function using Lagrangian multipliers, and solve the Lagrangian relaxed problem via a policy network. To satisfy the constraints on the system capacity, we design a constraint network to map the latent actions generated from the policy network to the orchestration actions such that the total resources allocated to network slices do not exceed the system capacity. We prototype *SafeSlicing* on an end-to-end testbed developed by using OpenAirInterface LTE, OpenDayLight-based SDN, and CUDA GPU computing platform. The experimental results show that *SafeSlicing* reduces more than** 20% **resource usage while meeting SLAs of network slices as compared with other solutions.**

***Index Terms*—End-to-End Slicing, Resource Orchestration, Deep Reinforcement Learning, Constraint-Awareness**

I. Introduction

5G is designed to support three primary use cases, i.e., enhanced Mobile Broadband (eMBB), Ultra Reliable Low Latency Communications (URLLC) and Massive Machine Type Communication (mMTC), which consequently enable various new applications such as augmented/virtual reality (AR/VR), 360-degree video streaming, and vehicle-to-everything (V2X) [1]. These new applications have diverse requirements of quality of services (QoS), e.g., data rates, latency, jitters, and reliability. It is thus desirable to cost-efficiently customize mobile networks and provision networking and computing resources for individual applications according to their demands [2].

Network slicing allows mobile network operators (MNOs) to virtualize physical network infrastructures, e.g., base stations and servers, and provide network slices with isolated resources to slice tenants [3]. A network slice is an end-to-end virtual network customized to fulfill the QoS requirements of a particular application based on the service level agreement (SLA) [4]. To accommodate more slices simultaneously and thus improve the network revenue, MNOs aim to satisfy the QoS requirement of slices with the minimum resource usage in multiple technical domains [5]. As shown in Fig. 1, the slice management [6] is composed of multiple levels, e.g., function, programmability, slice and service levels. At the programmability level, multiple domain managers are developed to enable the abstraction of network functions (NFs), e.g., physical, virtual and containerized NFs, and implement the orchestration decision from the resource orchestrator. An end-to-end resource orchestrator at the service level dynamically orchestrates the virtual resources (e.g., virtual UL radio resource) to diverse slices for meeting their performance requirements. It observes high-dimension network states from domain managers and slice tenants, and makes orchestration decisions to assure performance of slices.

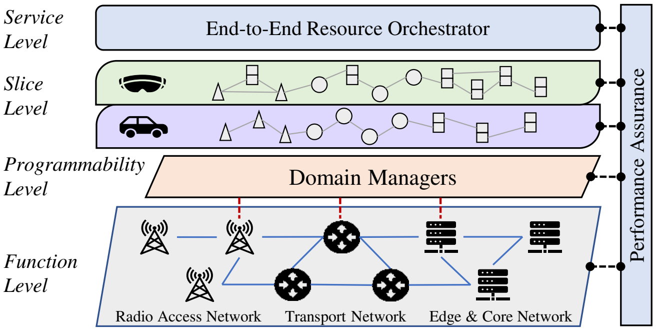


Fig. 1: The architecture of network slicing management

It is challenging to orchestrate cross-domain resources due to the complicated interdependencies in end-to-end network systems. On one hand, the slice performance is correlated to many distinct types of resources, e.g., bandwidth in transport network and CPU/IO in edge servers, where an accurate mathematical model can hardly be derived [7], [8]. Consequently, model-based approaches fail to precisely represent the complex network slicing system and might result in performance degradation if applied. On the other hand, the varying high-dimensional network states also affect the slice performance. For example, given a resource orchestration, the round-trip application latency of an AR/VR slice may change under different radio channel qualities and traffic of slice users. In addition, the resource orchestration problem shows the Markov property, i.e., the next slice performance and network states depend only upon the current orchestration decision and observed network states such as service queuing in edge servers. Therefore, a model-free solution that can

[§]This work is completed at Nokia Bell Labs. Dr. Tao Han's work is partially supported by the US National Science Foundation under Grant No. 1910844, No. 2008447, No. 2047655, and No. 2049875.

[§]We would like to thank the shepherd, Prof. Kate Ching-Ju Lin for the guidance in refining this paper and anonymous reviewers for their insightful comments.

978-1-6654-4131-5/21/$31.00 ©2021 IEEE

handle the high-dim interdependencies is needed to orchestrate cross-domain resources in end-to-end network slicing.

Deep reinforcement learning (DRL) has gained increasing popularity in managing and optimizing dynamic and complex network systems in a model-free approach [7], [9], [10], and achieves promising performance improvements by parameterizing policies with the deep neural networks (DNNs). However, most of the existing DRL solutions derive their policies within simulated environments and directly apply them into real networks without any adaptations. These offline simulated environments are usually built by leveraging approximated models, e.g., queuing, which cannot completely represent the complex network systems, especially in end-to-end domains. In other words, there is a discrepancy between the offline environment and the real network [9], [8], which is not negligible in practice and cannot be overlooked. As a result, deploying the offline-trained policy for resource orchestration in real networks could cause performance degradation of slices and thus lead to the violation of slice SLAs. Thus, an online DRL solution is indispensable for eliminating the simulation-to-reality discrepancy, which allows the policy to be learned by directly interacting with a real network.

On slicing end-to-end networks, DRL needs to be constraint-aware for two main reasons. First, the policy should satisfy the statistical performance requirement derived from slice SLAs, e.g., 95-percent end-to-end application latency in a day. Reward-shaping methods are proposed to solve this problem by shaping the reward function with static weighted constraints. However, these methods may result in either violations of SLAs when the weights are too small or the suppression of exploitation when the weights are too large. Second, the policy needs to maintain the instantaneous system limitations at any time slots, e.g., the summation of the bandwidth allocated to all slices cannot exceed its total link bandwidth. On learning the orchestration policy, a DRL algorithm needs to explore the whole action space containing all possible actions, i.e., resource orchestration of slices, for seeking better rewards. For example, a widely-adopted random action exploration strategy adds small deviations, which are sampled from a Normal distribution, onto the actions generated by the policy. As these deviated actions are implemented in the real network, multiple system limitations could be breached.

In this paper, we propose *SafeSlicing* that allows MNOs dynamically slice end-to-end network system, i.e., optimizes the cross-domain resource usage while guaranteeing the performance requirements of slices and maintaining the system limitations. *SafeSlicing* is accomplished by a novel constraint-aware deep reinforcement learning (CaDRL) algorithm in two steps, i.e., offline training in a simulated environment and online learning with the real network. First, we develop a simulated environment to imitate the resource orchestration in the real network, by using real experimental dataset and exploiting domain knowledge such as queuing service model. The environment is designed to reduce the simulation-to-reality gap, and used to offline train the policy, which thus boosts the policy performance before online learning. Second, we allow the policy to be refined in an online learning manner by interacting with the real network directly. The policy can quickly adapt to the real network with the CaDRL algorithm and hence eliminate the simulation-to-reality discrepancy.

The CaDRL algorithm incorporates the statistical performance constraints in its reward function by using Lagrangian primal-dual method and uses a novel constraint neural network as the output layer appending to the end of the policy network to regulate the resource orchestration actions. This constraint network ensures all the actions generated by the policy satisfy the constraints on instantaneous resource allocations at any time slots. Because of the constraint awareness, CaDRL can directly interact with the real network and learn the resource orchestration policy. We validate the performance of the CaDRL algorithm and prototype *SafeSlicing* on an end-to-end network slicing testbed designed with OpenAirInterface, OpenDayLight SDN, and CUDA GPU computing platform[1].

The contributions of this paper are summarized as follows:

- We design *SafeSlicing*, the first integrated offline-online learning system, that can dynamically orchestrate networking and computing resources for end-to-end network slicing and optimize the resource utilization of network infrastructures while meeting the SLAs of slices.
- We develop a novel constraint-aware deep reinforcement learning (CaDRL) algorithm that can learn the orchestration policy directly from the real network. CaDRL can effectively handle the constraints on the statistical slice performance and instantaneous resource allocation during the policy optimization so that the violations of slices' SLAs approximate to zero.
- We prototype *SafeSlicing* on a small-scale end-to-end network slicing testbed, by using OpenAirInterface LTE to implement the radio access network, SDN with the OpenDayLight controller to emulate the transport network, and CUDA GPU computing platform for computing acceleration.
- We conduct extensive experiments to evaluate the performance of *SafeSlicing*. The experimental results show that *SafeSlicing* outperforms other existing solutions in terms of resource usage and slice performance.

II. Background and Motivation

With dynamic network slicing, the MNO aims to minimize the usage of cross-domain resources while satisfying the statistical performance requirements of network slices with the capacity constraints on physical networking and computing infrastructures. For example, an AR/VR slice needs uplink radio resources to upload environment sensing data, transport resources to transmit data to edge servers, computing resources to process data, and downlink radio resources for downloading augmentation contents. To accomplish this slice's SLA, e.g., 500 ms average round-trip latency, multiple resource

[1] *SafeSlicing* fits well with 5G architecture that provides open interfaces and flexible control functions such as O-RAN (open RAN) [11], NWDAF (NetWork Data Analytics Function), and NEF (Network Exposure Function).

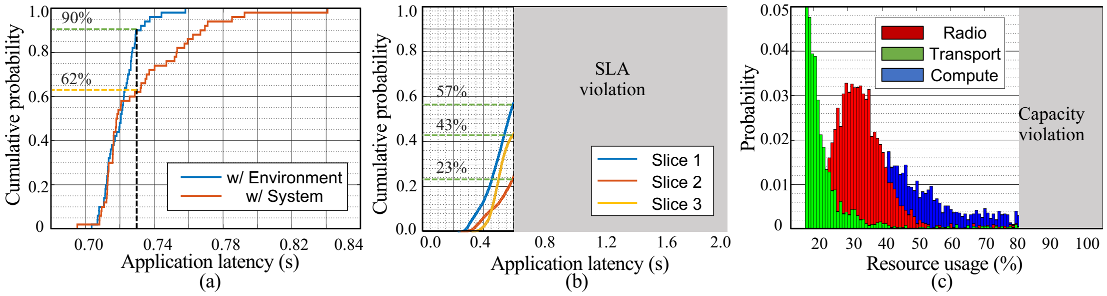


Fig. 2: a) CDF of app. latency within system and simulated environment under identical policy; b) CDF of app. latency of slices during online learning with reward shaping; c) the probability distribution of resource usages during online learning with reward shaping.

orchestrations may be feasible, e.g., 10 UL/DL PRBs in RAN, 100Mbps bandwidth in transport network, and 1 dedicated GPU in the edge server. However, the minimal resource usage closely depends on the high-dimensional network state, e.g., slice traffic, radio channel quality, transport congestion, and service queuing in servers. For instance, this slice requires more UL/DL PRBs to maintain a similar performance if the radio channel quality of slice users is poor, as each PRB carries less data. Besides, the capacity of resources are constrained by physical network infrastructures, e.g., the total number of PRBs, and thus the orchestration of all slices need to be jointly optimized in case of resource over-requesting. Therefore, we resort to the deep reinforcement learning (DRL) technique to deal with complicated interdependencies in the complex end-to-end slicing system.

Applying DRL in dynamic network slicing faces two challenges. The first one is the discrepancy between the simulated environment and real networks, which causes that a DRL policy trained offline within a simulated environment may not work well in real networks. The second one is the constraint violation during the online learning as the intrinsic DRL exploration mechanism.

Simulation-to-reality discrepancy. To study the impact of the discrepancy, we develop an end-to-end network slicing testbed (detailed in Sec. VI) and build a simulated environment that mimics the resource orchestration in the real network. We train the DRL policy in the simulated environment and then apply the policy in the real network without online learning. Fig. 2 (a) shows the performance comparison of a resource orchestration policy in the simulated environment and system testbed. In the simulated environment, the policy obtains the application latency of slices range from $0.7s$ to $0.76s$ with about 90% of it being less than $0.73s$. In contrast, when the same policy is applied in the system testbed, the application latency of slices range from $0.69s$ to $0.83s$, and only 62% of it is less than $0.73s$. This result shows the performance degradation when applying an offline-trained policy to a real network. This simulation-to-reality discrepancy also exists in various systems, e.g., data center networks [9], robot system [12], and video telephony [8], and can be hardly eliminated as real networks are too complicated to be precisely represented by simulated environments.

Constraint violation. The problem caused by the simulation-to-reality discrepancy requires us to design an online DRL approach that can directly interact with and learn the network slicing policy from real networks. However, during the online learning, the DRL agent may violate the SLAs and system capacity constraints. Assume that the maximum allowable application latency of a slice is $0.6s$, and the resource usage of all slices cannot exceed 80% of the capacity of individual resources, where the remaining 20% resources are reserved for conventional non-slicing users. Fig. 2 (b) and (c) show the performance of an online DRL algorithm that uses the reward shaping technique to shape the reward function with fixed weighted constraints [13]. In Fig. 2 (b), it shows that the probability of satisfying the maximum latency constraint in slice 1, 2, and 3 are only 57%, 23% and 43%, respectively. In Fig. 2 (c), we can see that there are many instances in which the resource usages exceed 80% of the capacity of individual resources.

These experiment results demonstrate the need of a safe network slicing system that enables MNOs to dynamically optimize the resource orchestration among slices without violating the SLA and capacity constraints.

III. *SafeSlicing* Overview

As shown in Fig. 3, *SafeSlicing* consists of a resource orchestrator, a simulated environment, and the real network[2]. The resource orchestrator centrally manages the allocation of virtual resources in radio access network, transport network, and edge servers, to all slices. The simulated environment is designed to imitate the resource orchestration in the real network for training the CaDRL agent offline. The domain managers (DMs) implement the resource orchestration issued by the orchestrator in radio access network, transport network, and edge servers, respectively. The orchestrator is deployed in the core network and DMs are instantiated in proximity to infrastructures, where they exchange data, e.g., state collection and action notification, through socket communication[3]. The network state, e.g., slice traffic and queue, is collected by a system monitor from domain managers and slice tenants. The system monitor also collects the performance of slices (i.e., end-to-end latency) and stores state-action-performance pairs

[2] *SafeSlicing* is aligned with industry efforts on network slicing, e.g., ETSI ZSM [14], 3GPP Slice LifeCycle Management (LCM) [15].

[3] The deployment of *SafeSlicing* can be extended in large-scale operational networks to support distributed DMs. The communication between the orchestrator and DMs can be enhanced by adopting standardized architectures (O-RAN [11]) and interfaces (ETSI Network Resource Model [16]).

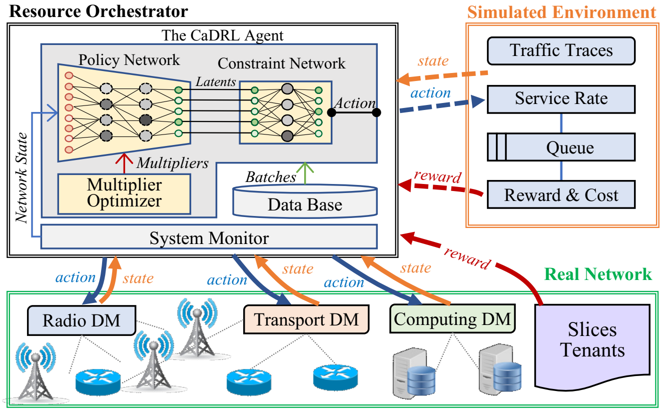


Fig. 3: The *SafeSlicing* system.

into a database. The *SafeSlicing* is accomplished in two steps, i.e., offline training in the simulated environment and online learning with the real network, where the offline trained policy serves as the start point of online learning.

The core of the resource orchestrator is the constraint-aware deep reinforcement learning (CaDRL) agent. It learns the orchestration policy either in the simulated environment or with the real network, and ensures the slice SLAs and system limitations to be adhered during the exploration of the policy. The CaDRL agent consists of three main components, i.e., a *policy network*, a *constraint network*, and a multiplier optimizer. The *policy network* is responsible for generating latent actions based on the network *states* to maximize the long-term reward. The statistical constraints, e.g., average application latency of slices, are incorporated in the reward function so that they can be enforced during the policy optimization. During the learning, *policy network* is updated by using actor-critic method [17] under dynamic multipliers. The multipliers are updated by an optimizer that runs at a slower timescale than that of the update of *policy network* to maintain the statistical constraints in the network slicing (detailed in Sec. V-A). The latent actions are then fed into the pre-trained *constraint network* which generates the final resource orchestration actions. The *constraint network* prevents the orchestration actions from violating the instantaneous resource constraints, e.g., the system capacity of cross-domain resources (detailed in Sec. V-B).

IV. System Model

We consider an end-to-end mobile network that consists of a cellular base station (BS) in radio access network (RAN), an edge computing server, and a transport network connecting the RAN and the server. Denote $\mathcal{I}$ and $\mathcal{K}$ as the set of slices and multi-domain resources, respectively. $i \in \mathcal{I}$ and $k \in \mathcal{K}$ represent the ith slice and kth resource, respectively. The end-to-end mobile network hosts multiple slices according to their service level agreements (SLAs) that define the performance requirements, e.g., the maximum latency [18]. To optimize the performance of slices, the network operator adjusts the resource orchestration in each domain periodically.

With the consideration of temporal interdependencies and various constraints in the end-to-end network slicing, the cross-domain resource orchestration problem can be naturally formulated as a constrained Markov decision process (CMDP) denoted as $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{C}, \mathcal{P}, \mu \rangle$. Here, $\mathcal{S}, \mathcal{A}$ are the set of states and actions, and $\mathcal{R}, \mathcal{C}$ are the reward and cost functions, respectively. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function, and $\mu : \mathcal{S} \to [0, 1]$ is the initial state distribution. A parameterized resource orchestration policy $\pi_\theta : \mathcal{S} \to Pr(\mathcal{A})$ is a mapping from states to probability distribution over actions, where θ is the neural network parameters.

The state $\mathbf{s}_t \in \mathcal{S}$ is observed by the policy π_θ, which represents the status of the end-to-end network. Thus, we define ***state*** as the combination of following components: 1) the average traffic of slices $[f_i^{t-1}, \forall i \in \mathcal{I}]$, which provides useful information about the user traffic at the current time slot t; 2) the number of users waiting in the queue of each slice $[l_i^{t-1}, \forall i \in \mathcal{I}]$, which describes traffic loads inherited from last time slot $t-1$; and 3) the slice performances in last time slot that consists of the number of users served by each slice $[u_i^{t-1}, \forall i \in \mathcal{I}]$, the resource usages $[g_k^{t-1}, \forall k \in \mathcal{K}]$, and the performance of slices $[y_i^{t-1}, \forall i \in \mathcal{I}]$. This information shows how the resource orchestration in last time slot performs.

Based on the observed state, the policy π_θ generates an action, where $\mathbf{a}_t \in \mathcal{A}$ corresponds to the cross-domain resource orchestration for slices. We define ***action*** $\mathbf{a}_t \triangleq [a_{i,k}^t, \forall i \in \mathcal{I}, k \in \mathcal{K}]$, where $a_{i,k}^t \in [0, 1]$ is the kth resource allocated to the ith slice at time slot t. In the system, we consider four types of resources (see Sec. V-C), i.e., the uplink and downlink physical resource blocks (PRBs) in radio access network, the bandwidth in transport network, and the GPU computing resources in edge servers. By taking action $\mathbf{a}_t$ to the system under the state $\mathbf{s}_t$, a reward can be obtained from the system. We define ***reward*** function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as $r_t \triangleq -\sum_{i \in \mathcal{I}} \sum_{k \in \mathcal{K}} a_{i,k}^t$. By maximizing the long-term rewards achieved by the policy π_θ, we are able to minimize the utilization of cross-domain resources.

Meanwhile, the resource orchestration has to satisfy two types of constraints, which are mapped to the cost functions $\mathcal{C} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. On one hand, the network operator needs to maintain the slice performances specified by SLAs. Denote y_i^t is the performance of the ith slice at time slot t, we define the first ***cost*** as $c_i \triangleq y_i^t, \forall i \in \mathcal{I}$. Then, we can maintain the SLA of slices by letting

$$\mathbb{E}_{\pi_\theta} \left[\frac{1}{|\mathcal{T}|} \sum\nolimits_{t \in \mathcal{T}} \gamma^t c_i(\mathbf{s}_t, \mathbf{a}_t) \right] \geq Y_i, \forall i \in \mathcal{I}, \tag{1}$$

where Y_i is the performance requirement of the ith slice. For instance, the $c_i(\mathbf{s}_t, \mathbf{a}_t)$ can be defined as 90-percent application latency of slice users in a slice and Y_i is 0.5s.

On the other hand, the resource orchestration subjects to the constraints of the system capacity, e.g., limited radio and computing resources, at any time slots. Hence, we define the second ***cost*** as $d_k \triangleq \sum_{i \in \mathcal{I}} a_{i,k}^t, \forall k \in \mathcal{K}$. Then, we can satisfy the constraints on the system capacity by ensuring

$$d_k(\mathbf{a}_t) \leq Z_k, \forall k \in \mathcal{K}, t \in \mathcal{T}, \tag{2}$$

where Z_k is the total amount of the kth resource.

On slicing the network, we aim to derive an orchestration policy π_θ^* that minimizes the utilization of cross-domain resources without violating constraints defined by the SLA and system capacity, respectively. Therefore, given a time period $\mathcal{T}$, e.g., 24 hours, we formulate the cross-domain resource orchestration problem in end-to-end network slicing as

$$\begin{aligned} \mathscr{P}_1 : \max_{\pi_\theta} \quad & \mathbb{E}_{\pi_\theta}\left[\sum\nolimits_{t\in\mathcal{T}} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right] \\ s.t. \quad & (1), (2). \end{aligned} \tag{3}$$

The difficulties of solving the above problem is multi-fold. First, the correlations between the orchestration action $\mathbf{a}_t$ and the slice performance $r(\mathbf{s}_t, \mathbf{a}_t)$ are complicated by the high-dim network state $\mathbf{s}_t$ (e.g., slice traffic, radio channel quality and queuing servers), and thus cannot be accurately represented by mathematical models. Second, the orchestration action in a time slot would influence not only the current slice performance but also the future network state and performance, which shows a Markov property in end-to-end network slicing. Therefore, we resort to DRL approaches to deal with the above problem in complex network slicing system.

V. Constraint-Aware DRL

In the cross-domain resource orchestration problem, we consider two types of constraints. The first type of constraints are the performance requirements, i.e., the 90-percent end-to-end latency in a day, derived from SLAs (Eq. 1). Since these constraints are on the statistical performance of networks over a period of time, we define them as the statistical constraints. The second type of constraints are the system capacity constraints, e.g., total number of radio resource blocks in the system (Eq. 2). The violation of these constraints depends only on every output resource orchestration action. Hence, we define the second type of constraints as the instantaneous constraints. In this section, we develop a constraint-aware deep reinforcement learning (CaDRL) algorithm that handles the statistical and instantaneous constraints via the policy network and constraint network, respectively.

A. Handling Statistical Constraints

To handle the statistical constraints in Eq. 1, we utilize Lagrangian primal dual method to incorporate the constraints into the reward with multipliers [19]. The Lagrangian function can be expressed as

$$\mathcal{L}(\theta, \lambda) = \mathbb{E}_{\pi_\theta}\left[\sum_{t\in\mathcal{T}} \gamma^t \left(r(\mathbf{s}_t, \mathbf{a}_t) + \sum_{i\in\mathcal{I}} \frac{\lambda_i}{|\mathcal{T}|} c_i(\mathbf{s}_t, \mathbf{a}_t) \right)\right] + C, \tag{4}$$

where $C = -\sum_{i\in\mathcal{I}} \lambda_i Y_i$ and λ are multipliers. The dual function, i.e., the point-wise maximization of Lagrangian with respect to parameters θ of *policy network*, can be written as

$$\mathcal{D}(\lambda) = \max_{\theta\in(2)} \mathcal{L}(\theta, \lambda), \tag{5}$$

where the dual function provides a lower bound on the value of the Lagrangian in Eq. 4. The tighter the bound, the closer the gap between the policy obtained from dual function and the optimal solution of problem $\mathscr{P}_1$. Thus, the dual problem, which finds the tightest bound, can be expressed as

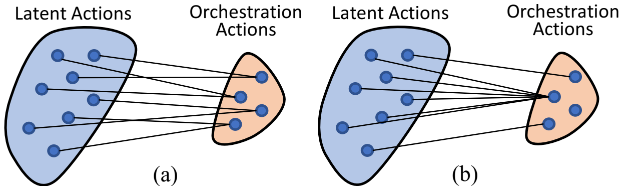


Fig. 4: Illustration of a) inefficient mapping, b) efficient mapping.

$$\min_{\lambda\geq 0} \mathcal{D}(\lambda). \tag{6}$$

As indicated in [20], the duality gap between the primal and dual problem approaches nearly zero if the policy is parameterized by neural networks. In other words, the problem $\mathscr{P}_1$ can be effectively addressed by alternatively solving the primal problem in Eq. 5 and the dual problem in Eq. 6.

On one hand, the dual problem is solved by updating the Lagrangian multipliers with sub-gradient descent as

$$\lambda_i^{(m+1)} = \left[\lambda_i^{(m)} - \eta_i \left(\frac{1}{N|\mathcal{T}|} \sum_{n=0}^{N} \sum_{t\in\mathcal{T}} \gamma^t c_i(\mathbf{s}_t, \mathbf{a}_t) - Y_i \right)\right]^+ \tag{7}$$

where $\eta_i, \forall i \in \mathcal{I}$ are non-negative step sizes and $[x]^+ = \max(0, x)$. Instead of evaluating $\mathbb{E}_{\pi_\theta}\left[\sum_{t\in\mathcal{T}} \gamma^t c_i(\mathbf{s}_t, \mathbf{a}_t)\right]$, we approximate it with N times Monte-Carlo sampling.

On the other hand, we note that solving the primal problem, i.e., minimizing the Lagrangian w.r.t. the parameters θ, approximately corresponds to learning a policy with reinforcement learning algorithms such as policy gradient [21] or actor-critic method [17]. Thus, we solve the primal problem by updating

$$\theta^{(m+1)} \approx \arg\max_{\theta\in(2)} \mathcal{L}(\theta, \lambda^{(m)}). \tag{8}$$

B. Handling Instantaneous Constraints

When updating *policy network* with parameters θ, we need to maintain that the orchestration actions satisfy the instantaneous constraints in Eq. 2 at any time slots. Since the intrinsic DRL exploration seeks the better reward by exploring the nearby space of actions (e.g., adding noise sampled from Normal distributions), the instantaneous constraints could be easily violated by these deviated actions. To address this issue, we design a *constraint network* with parameters μ to map the latent actions generated by *policy network* to the resource orchestration actions that satisfy the instantaneous constraints. The *constraint network* is pre-trained and appended to *policy network* as illustrated in Fig. 3.

On the designing the *constraint network*, we aim to map the latent actions to final orchestration actions such that the DRL agent can efficiently explore the entire space of the orchestration actions as illustrated in Fig. 4(a). A common mapping method, i.e., clamping [22], may result in an inefficient mapping in which a disproportional number of latent actions are mapped to a single orchestration action as shown in Fig. 4(b). With the clamping method, the latent actions that not satisfying the instantaneous constraints are mapped to the boundary of the space defined by the constraints. When the DRL is exploring actions outside the constraint space, the random action exploration mechanism fails as all these explored actions are clamped. As a result, the efficiency of exploring the action space is decreased, and thus degrading the performance of DRL.

To achieve an efficient mapping, we formulate the problem of designing the *constraint network* as

$$\begin{aligned} \mathscr{P}_2: \quad \min_{\mu} \quad & \mathbb{E}_{\mu}\left[D_{KL}\left(p(\mathbf{a})||q(\hat{\mathbf{a}})\right)\right] \\ s.t. \quad & \mathbb{E}_{\mu}\left[d_k(\hat{\mathbf{a}})\right] \leq Z_k, \forall k \in \mathcal{K}, \end{aligned} \tag{9}$$

where D_{KL} is the Kullback-Leibler divergence [23], $p(\mathbf{a})$ and $q(\hat{\mathbf{a}})$ are the distribution of input actions $\mathbf{a}$ and output actions $\hat{\mathbf{a}}$, respectively. We solve problem $\mathscr{P}_2$ by using the Lagrangian primal-dual method. Specifically, the training loss of the *constraint network* is designed as

$$Loss_{\mu} = D_{KL}\left(p(\mathbf{a})||q(\hat{\mathbf{a}})\right) + \sum_{k\in\mathcal{K}} \beta_k \left(d_k(\hat{\mathbf{a}}) - Z_k\right), \tag{10}$$

where $\beta_k, \forall k \in \mathcal{K}$ are Lagrangian multipliers that are updated

$$\beta_k^{(m+1)} = \left[\beta_k^{(m)} + \xi_k \left(A - Z_k\right)\right]^{+}, \forall k \in \mathcal{K}, \tag{11}$$

where $A = \max\{d_k(\hat{\mathbf{a}}_n), \forall n = 1, 2, ..., N\}$ is the maximum value of the instantaneous constraint under N times Monte-Carlo sampling, and $\xi_k, \forall k \in \mathcal{K}$ are non-negative step sizes.

On training the *constraint network*, the inputs are sampled from the space of the latent actions uniformly. Here, we aim to build a deterministic mapping from the latent actions to the orchestration actions. As the pre-trained *constraint network* is appended to *policy network*, we can rewrite the primal problem in Eq. 8 as

$$\theta^{(m+1)} \approx \arg\max_{\theta} \mathcal{L}(\theta, \lambda^{(m)}). \tag{12}$$

From the perspective of *policy network*, its learning becomes unconstrained, and thus various exploration techniques can be applied without concerning the violation of instantaneous constraints. Since the *constraint network* is based on neural network architecture, which is differentiable, it is compatible with the backpropagation of gradients during *policy network* learning. Hence, the orchestration actions outputted from the *constraint network* are not required to be close to the latent actions outputted from *policy network* in terms of Euclidean distance.

C. The SafeSlicing Workflow

Deploying Safeslicing mainly consists of three stages: offline constraint network training, offline policy network training, and online policy network learning.

Offline Constraint Network Training. The *constraint network* is trained to translate latent actions into resource orchestration actions that satisfy the system constraints. First, the input batches are sampled uniformly from the latent action space. Then, these batches are fed into *constraint network* that generates the output actions $\hat{\mathbf{a}}$. The loss function in Eq. 10 is calculated, which is used to update *constraint network* accordingly. After multiple *constraint network* updates, we follow Eq. 11 to update the multipliers $\beta_k, \forall k \in \mathcal{K}$. This procedure repeats until the multipliers become stable.

Algorithm 1: The CaDRL Algorithm

```
   Input: $Y_i$, $\forall i \in \mathcal{I}$; $Z_k, \forall k \in \mathcal{K}$; $\eta$, $\xi$.
   Output: $\pi_{\theta+\mu}$
1  $/ * *$ constraint network training phase $* * /$;
2  Initialize parameters $\mu$, multipliers $\beta_k, \forall k \in \mathcal{K}$;
3  for $m = 0, 1, ...$ do
4      Sample batches $\mathcal{B} \leftarrow$ latent actions;
5      $\mu^{(m+1)} \leftarrow \arg\min_{\mu} Loss_{\mu}$;
6      if time to update then
7          $\beta_k^{(m+1)} \leftarrow Eq.\ 11, \forall k \in \mathcal{K}$;
8      if convergence then
9          break;
10 Append constraint network to policy network;
11 $/ * *$ offline training phase $* * /$;
12 Initialize parameters $\theta$, multipliers $\lambda_i, \forall i \in \mathcal{I}$;
13 for $m = 0, 1, ...$ do
14     for $t = 0, 1, ..., T - 1$ do
15         $a_{i,k}^t, \forall i \in \mathcal{I}, k \in \mathcal{K}, \leftarrow \pi_{\theta+\mu}$;
16         $y_i^t, \forall i \in \mathcal{I}, \leftarrow$ system;
17         $\sum_{i\in\mathcal{I}} a_{i,k}^t, \forall k \in \mathcal{K}, \leftarrow$ system;
18     $\theta^{(m+1)} \leftarrow \arg\min_{\theta} \mathcal{L}(\theta, \lambda^{(m)})$;
19     if time to update then
20         $\lambda_i^{(m+1)} \leftarrow Eq.\ 7, \forall i \in \mathcal{I}$;
21     if convergence then
22         break;
23 $/ * *$ online learning phase $* * /$;
24 Repeat the procedures between Line 13 and Line 22;
25 return $\pi_{\theta+\mu}$;
```

Offline Policy Network Training. We offline train the CaDRL agent with a simulated environment before we conduct the online learning. The simulated environment is developed to imitate the resource orchestration in the real network by leveraging experimental dataset and exploiting domain knowledge. As shown in Fig. 5, we implement a FIFO queue in the environment to simulate the service process for individual slices. Since the service rates of the queues determine the performance of the slices, it is important to accurately predict the service rater under different resource orchestrations to approximate the performance of the real network. Hence, we design a regression model with *scikit-learn* [24] tool in each slice to predict the service rates according to the experimental dataset collected from a real network. By using the real traffic trace that records the arrival timestamps of user requests, and the predicted service rates, we simulate the enqueuing, serving and dequeuing process of user requests in slices in each time slot. At the end of a time slot, the application latency of

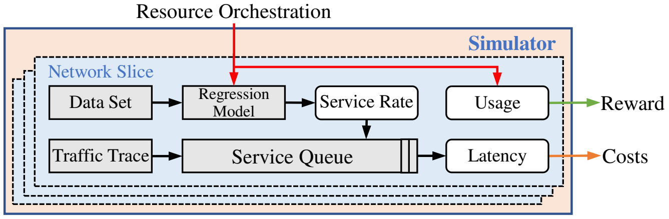


Fig. 5: The design of the simulated environment.

all slices and resource usages are outputted as the costs and rewards, respectively.

During the offline training of *policy network*, we design the agent to observe the following states, i.e., *the average traffic of slices*, *the number of slice users in service queues*, *the number served users of slices*, *the performance of slices* and *the resource usage at the last time slot*. Based on the observed state, the CaDRL agent generates 4-dim resource orchestration for each slice, which are *uplink and downlink physical resource blocks (PRBs) in RAN*, *bandwidth in transport network*, and *the GPU computing resources in edge servers*. The resource orchestration is then fed into the simulated environment shown in Fig. 5, which updates slice users in service queues according to the traffic trace of slices. The CaDRL agent retrieves the *resource usage* and *end-to-end latency* of slices from the simulated environment as the reward and cost, respectively.

Online Policy Network Learning. In this phase, we allow the CaDRL agent to directly interact with domain managers, continuously update its *policy network*, and gradually adapt to the real network. At runtime, the resource orchestrator collects the state space (same with that of offline training) from domain managers and app servers of slices by using TCP sockets. As the state space is fed into *policy network* and *constraint network* in the CaDRL agent, an orchestration action is generated. The action is sent to the corresponding domain manager for the action implementation in network infrastructures. For example, the transport bandwidth allocation of slices are sent to the transport domain manager. The orchestrator adapts the resource orchestration every 15 minutes. The reward and cost of the orchestration action are defined as the average resource usage and average end-to-end application latency of slices over the 15 minutes period, respectively. The rewards and costs are collected from domain managers and app servers of slices, respectively[4]. We define the length of an episode as a day, i.e., 96 transitions per episode[5]. In the online learning, the CaDRL agent continues exploring the action space and learning from interactions toward the optimal policy in the real network.

VI. System Implementation

Testbed Setups. We prototype *SafeSlicing* on an end-to-end network slicing testbed as depicted in Fig. 6. The system testbed is composed of radio access network (RAN) with an eNodeB, transport network with an OpenFlow switch, core network, and edge server with a CUDA GPU. The details of hardware are summarized in Table I. The domain managers of radio, transport and compute resources are deployed in the computers running eNodeB, a SDN controller and core network, respectively. The eNodeB is configured with 10MHz (50 PRBs) wireless bandwidth. The total bandwidth of the transportation network between the eNodeB and the edge server is 100Mbps. The total amount of the computing resource in the edge server is 51200 CUDA threads.

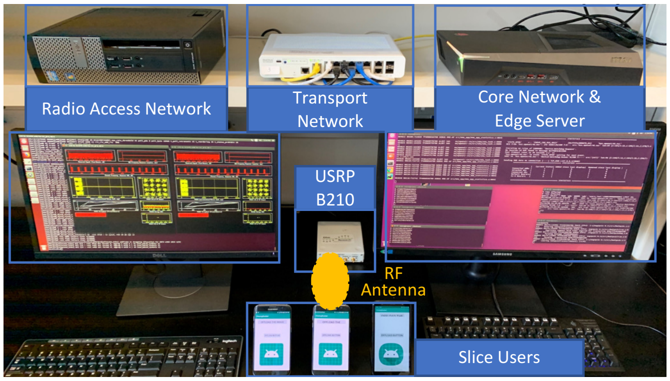


Fig. 6: The overview of system testbed.

Component	Hardware	Software
UEs	3x Smartphones	Android 7.0
eNodeB	1x Intel i5 Computer	OpenAirInterface [29]
RF Front-End	1x Ettus USRP B210	N/A
Transport	1x OpenFlow Switch	OpenDayLight [30]
Core Network	1x Intel i7 Computer	OpenAir-CN [31]
Edge Server	1x Nvidia GTX 1070	CUDA 9.0 [32]

TABLE I: Details of the system testbed

Orchestrator. We implement the resource orchestrator by using Python 3.7 and PyTorch 1.40 [26] and co-locate the resource orchestrator with the core network. We employ a state-of-the-art reinforcement learning algorithm, i.e., deep deterministic policy gradient (DDPG) [27], to update *policy network*. In particular, we use a 2-layer fully-connected neural network, i.e., [512, 512], in both actor and critic networks in DDPG. The *constraint network*, which is appended at the end of actor network of DDPG, is implemented with 2-layer fully-connected neural network, i.e., [128, 128]. For both *policy network* and *constraint network*, we use Leaky Rectifier and $sigmoid$ [28] activation functions for the intermediate layers and output layer, respectively. On training the resource orchestrator, we conduct extensive and empirical tuning on the hyper-parameters. The initial multipliers $\beta_k = 0.01, \forall k \in \mathcal{K}, \lambda_i = 0.01, \forall i \in \mathcal{I}$. The step sizes of multiplier update $\xi_k = 0.1, \forall k \in \mathcal{K}, \eta_i = 0.1, \forall i \in \mathcal{I}$. The learning rates of both actor and critic networks in DDPG and *constraint network* are 5e-4. The batch size is 512. The discounted factor for cumulative reward is $\gamma = 0.99$. We add a decaying Gaussian noise starts from $\mathcal{N}(0, 0.1)$ to $\mathcal{N}(0, 0.001)$ on actions during the training phase for balancing the exploitation and exploration.

Experimental Dataset. We collect experimental dataset

[4] For the deployment in large-scale operational networks, the rewards and costs can be measured via standardized interfaces (e.g., DRM [16]) that support data collections from domain managers and app servers.

[5] Although modern cellular networks have relatively long configuration intervals [25], e.g., 30 mins or 1 hour, we are seeing several efforts on open network architectures that enable real-time or near-RT configurations such as O-RAN [11], which would significantly improve the sampling efficiency for online learning.

from the testbed and use it in the simulated environment for training the regression models as shown in Fig. 5. We obtain the dataset of the resource orchestration and application latency pairs for each slice, by traversing different combinations of resource orchestration with the grid search method. Since the dimension of the resource orchestration space is very high, it is impractical to collect all pairs of the resource orchestration and application latency. Hence, we collect the data with the granularity of resource allocation equaling 6% for all the resources. At the runtime, the resource orchestration generated by the resource orchestrator may not be found in the dataset. To predict its application latency, we use the adjacent resource orchestration pairs in the dataset to fit the regression model. Once the model is fitted, the regression model predicts the application latency of the slice under the resource orchestration.

Domain Managers. Domain managers are developed to virtualize the physical infrastructures and dynamically implement cross-domain resource orchestrations received from the resource orchestrator into multiple technical domains. In the radio access network, the radio domain manager is designed based on OpenAirInterface [29], that allocates both uplink and downlink radio resources to slices. The transport domain manager is designed based on an OpenDayLight [30] controller, that allocates bandwidth of the links between radio access network and edge servers through OpenFlow (Southbound API) and RESTful (Northbound API) [33]. The computing domain manager is designed based on CUDA GPU computing platform, which dynamically allocates computing resources, e.g., the number of CUDA threads, to slices.

VII. Performance Evaluation

In this section, we aim to study 1) how *SafeSlicing* optimizes the network slicing via learning; 2) what's the performance comparison between *SafeSlicing* and existing algorithms; 3) how the design of constraint-awareness impacts the performance of the network slicing; and 4) whether *SafeSlicing* can adapt to the traffic dynamics and learn the actual resource demands of different applications.

Experiment Setups. We create three slices on the system testbed, where each slice hosts an application. The application traffic is generated in smartphones that are wirelessly connected to the testbed. For any individual networking and computing resources, the amount allocated to slices should be less than 80%, i.e., $Z_k = 0.8, \forall k \in \mathcal{K}$. We consider the average end-to-end application latency as the main SLA of a slice in the experiments and sets $0.6s$ as the latency requirement, i.e., $Y_i = 0.6s, \forall i \in \mathcal{I}$. Here, the value is selected based on the capability of the testbed and the traffic loads generated from the traffic data trace. A slice is allocated at least 6% of each resource to keep the slice live.

Performance Metrics. To evaluate the performance of *SafeSlicing*, we define the following performance metrics. The application latency is defined as the average end-to-end latency of the application in an episode. The system latency is defined as the average application latency of all slices. The resource usage is defined as the average usage of a resource in the system in an episode. The system resource usage is defined as the average usages of all resources in the system in an episode.

Network Dynamics. The network dynamics are composed of varying slice application traffic and different workload of slice applications. We use *Tsinghua App Usage Dataset* to generate the slice application traffic [34]. This dataset consists of traffic traces of nearly 2000 mobile applications. We select the social networking category in the dataset and scale down the volume of the requests according to the capability of our end-to-end network slicing testbed. We split the whole selected traffic traces into a train set with 80% traces and a test set with 20% traces for training and evaluation, respectively.

To generate different workload of slice applications, we develop three mobile apps that have diverse requirements of networking and computing resources. The first, second and third app have the demands of high-networking & low-computing, medium-networking & medium-computing, low-networking & high-computing, respectively. To simplify the application development, we use a video analytics framework in which video frames are sent to an edge server and analyzed by the YOLO object detection framework [35]. The amount of transmission data, i.e., images with different resolutions, determines the demand of the networking resources, and the complexity of computing model, i.e., YOLO, determines the demand of the computing resources. Hence, the image resolution and computing model of app 1 are 500x500 and tiny YOLOv3, respectively. The image resolution and computing model of app 2 are 300x300 and YOLOv3 416x416, respectively. The image resolution and computing model of app 3 are 100x100 and YOLOv3 608x608, respectively. The app server of slices are deployed in the core network, which record the performance of slice users (latency) and report the average slice performance to the CaDRL agent as rewards.

Comparison Algorithms. We compare *SafeSlicing* with the following algorithms: Reward Shaping (RS): solves the constrained reinforcement learning problem by incorporating the static weighted constraints into the reward function as penalties [7]. After extensive trials with different weights, we select 1.0 as the weight for both statistical and instantaneous constraints. Reward Constrained Policy Optimization (RCPO) [36]: is a two-layer training algorithm for constrained reinforcement learning designed based on the Lagrangian primal-dual method. In the upper layer, the Lagrangian multipliers are updated according to the sub-gradient decent method. In the lower layer, the policy network is updated based on the actor-critic method.

A. Experimental Results

1) Optimizing Network Slicing via learning: Fig. 7 (a) shows the entire learning-based optimization process of *SafeSlicing*. The offline phase starts from episode 0 to episode 250, in which the first 200 episodes are offline learning, and the episodes 201 to 250 are offline evaluation. After the evaluation, the offline policy is derived. When the offline

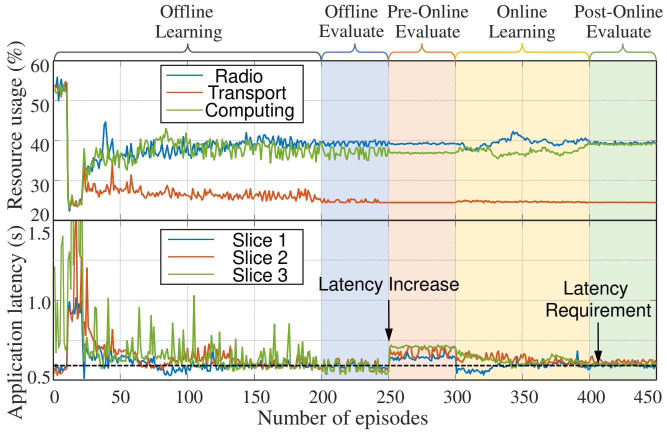


Fig. 7: Optimizing resource orchestration with *SafeSlicing*.

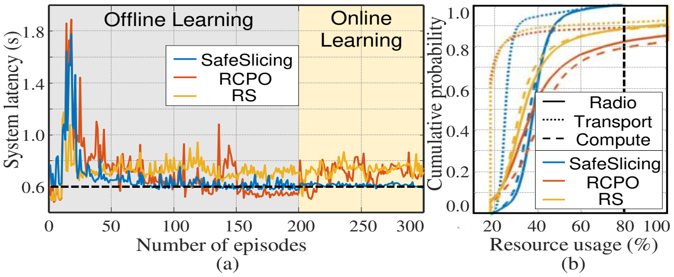


Fig. 8: The learning performance of different algorithms, a) the system latency; b) the resource usage.

policy is evaluated in the system (from episodes 251 to 300), the application latency is larger than that in the simulated environment. In other words, the online policy has a poor performance in the real network because of the simulated environment cannot fully reflect the dynamic of the real network. The online learning starts with episode 301 and ends at episode 400. During the online learning, the CaDRL agent interacts with the real network and learns the orchestration policy. In the online evaluation phase, we can see that the application latency of the slices are reduced to around $0.6s$ which is the minimal latency requirement.

2) Performance Comparison: We compare the performance of *SafeSlicing* with that of RCPO and RS. Fig. 8 (a) shows the system latency of these algorithms. *SafeSlicing* converges to $0.6s$ which the latency requirement defined in the SLA. Meanwhile, both RCPO and RS cannot converge, and the system latency under these algorithms are considerably higher than the latency requirement. This indicates that *SafeSlicing* has better performance in handling the statistical constraints. Fig. 8 (b) shows the cumulative probability of the resource usage of these algorithms. We can see that *SafeSlicing* can follow the resource usage constraints, i.e., less than 80% total resources. Both the RCPO and RS violate the resource usage constraints. This proves that *SafeSlicing* can effectively satisfy the instantaneous constraints.

Table II lists the performance of these algorithms in the post-online evaluation phase. *SafeSlicing* uses the least resources, i.e., $34.3\% \pm 1\%$, and satisfies the system latency constraint, i.e., $0.6s$, with 0% violation of the constraints

Metrics	SafeSlicing	RCPO	RS
Sys. Resource Usage(%)	**34.3±0.1**	42.8±0.2	47.1±0.9
System Latency (s)	**0.60±0.03**	0.70±0.25	0.71±0.16
Resource Usage Violation	**0.0%**	12.7%	12.4%

TABLE II: The performance of different algorithms.

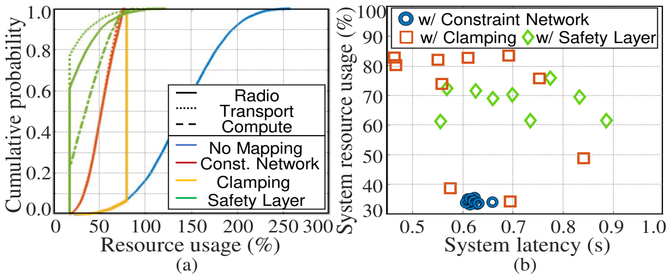


Fig. 9: a) The cumulative probability of resource usage under different mappings; b) *SafeSlicing* performance under different mappings.

on the instantaneous resource allocations. Here, *SafeSlicing* maintains the system latency at $0.6s$ with a $\pm 5\%$ error which is tolerant in the SLA. In contrast, although both RCPO and RS consume more resources, they still cannot meet the latency requirement and violate constraints on the resource usages.

3) Design of Constraint Awareness: Although the function of the *constraint network* is simple (i.e., mapping latent actions to orchestration actions), its design determines the performance of the network slicing. Fig. 9 (a) shows the cumulative probability of resource usages in the system with different mapping methods. Without any mapping, i.e., no constraint-awareness, the resource usages range from ∼20% to ∼260%. With *constraint network*, the range of the resource usage is condensed to ∼18% to ∼80%, which satisfies the constraints on the resource usage, i.e., $Z_k = 80\%, \forall k \in \mathcal{K}$. Besides, the distribution of the resource usage after the mapping is uniform, which benefits the searching of the orchestration policy. The safety layer method introduces an output layer at the end of *policy network* to correct the actions by minimizing the distance between the input and output actions [22]. The mapping results show that the safety layer method leads to a non-uniform distribution of the resource usage. The clamping method limits the resource usage to the nearest allowable value. Hence, whenever the resource usage exceeds 80%, the clamping method maps the usage to 80%.

Fig. 9 (b) shows the system latency and resource usage with different mapping methods used in *SafeSlicing*. It shows that the *constraint network* method significantly outperforms the other methods and achieves the shortest system latency with the least resource usage. The reason for the performance gain is that the *constraint network* method allows efficient exploration during *policy network* learning while satisfying the constraints of the system capacity.

In addition, we show the range of the resource usage during the training of *constraint network* under different static weights in Fig. 10. It can be observed that these static weighted *constraint networks* either cannot enable the resource usage to approach the upper bound (80%) and the lower bound (18%) or violate the resource usage constraints. In other words,

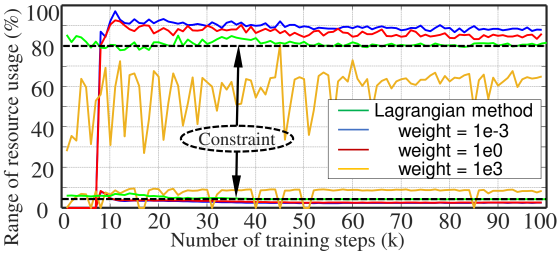
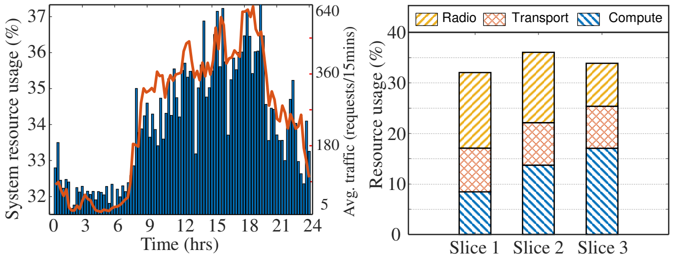


Fig. 10: The impact of weights of constraint network.



(a)

(b)

Fig. 11: a) system resource usage vs. average traffic within an episode; b) resource usage of slices.

constraint network with static weights may generate resource orchestrations that violate the instantaneous constraints and cannot explore some resource orchestrations whose resource usages are close to the boundary. In contrast, when the *constraint network* is trained using Lagrangian primal-dual method, the resource usage can approach the upper and lower bounds, which allows the CaDRL agent to explore the entire action space without violating the resource usage constraints.

4) Traffic and Resource Demand Awareness: We show the system resource usage of *SafeSlicing* (post-online) and the corresponded avg. traffic of slices within a episode in Fig. 11 (a). The system resource usage track the avg. traffic of slices, which means *SafeSlicing* can react to varying slice traffic and dynamically adjust its resource orchestrations, i.e., traffic awareness, to meet the latency requirement of slices. Fig. 11 (b) depicts the resource usage of *SafeSlicing* for slices. It shows that the resource usage in slices align with the resource demands of their application, i.e., resource demand awareness. For example, slice 1 and slice 3 adopt the highest frame resolution (500x500) and computation model (YOLOv3 608x608), respectively. As a result, slice 1 and slice 3 are allocated the largest radio and computing resources, respectively. For these applications, the transportation network is not the bottleneck, and thus the resource allocations in the transportation network are similar.

VIII. Related Work

This work relates to resource management problem in network slicing and machine learning techniques for networking.

Resource Management in Network Slicing: Salvat *et. al.* [5] proposed a Benders decomposition based algorithm that manages the resource provisioning of slices to maximize the revenue of mobile operators in an end-to-end network slicing system. Han *et. al.* [37] proposed a utility-based admission control mechanism based on multi-queuing systems to improve the resource efficiency for accommodating heterogeneous slices in network slicing. D'Oro *et. al.* [38] proposed a slice admission and resource provisioning algorithm to instantiate slices for heterogeneous services by considering the interdependencies among multiple domain resources in multi-access edge computing (MEC). However, these works consider that the performance of slices are mathematically modeled with closed-form expressions. In contrast, *SafeSlicing* is a model-free approach that is capable of handling the dynamic network slicing system with complicated interdependence among cross-domain resources.

Machine Learning for Networking: Ayala-Romero *et. al.* [10] developed a reinforcement learning based approach that dynamically allocates the coupled computing and radio resources to meet heterogeneous QoS targets in virtualized radio access network (vRAN). Bega *et. al.* [39] proposed DeepCog with deep learning techniques to predict network capacity within individual slices and balance the tradeoff between resource over-provisioning and service request violations. Liu *et. al.* [7] developed an end-to-end network slicing system and proposed a decentralized reinforcement learning approach to orchestrate cross domain resources to meet service level agreement (SLA) of slices. Liu *et. al.* [40] proposed a constrained DRL algorithm by using the Interior-point Policy Optimization (IPO) and clamping for dealing statistical and instantaneous constraints, which is trained offline. Zhang *et. al.* [8] designed OnRL, an online DRL solution within real networks, to improve the performance of real-time mobile video telephony, which fails to maintain various constraints in slicing system. However, these works follow the offline training and online deployment strategy which suffers from the discrepancy between the simulated and real network. In contrast, with the novel two-stage design of *SafeSlicing*, we achieve constraint-aware deep reinforcement learning that learns the orchestration policy directly with the real network while maintaining various constraints.

IX. Conclusion

In this paper, we have designed *SafeSlicing* that allows dynamically optimization of network slicing without violating the SLAs of slices. To accomplish *SafeSlicing*, we have designed a constraint-aware deep reinforcement learning (CaDRL) algorithm that learns the resource orchestration policy while meeting the practical constraints in two steps, i.e., offline training in a simulated environment and online learning with the real network. The statistical slice performance constraints are incorporated into the reward function with the Lagrangian primal-dual method and the instantaneous resource orchestration constraints are handled by a novel constraint network. We have developed a system testbed of *SafeSlicing* with OpenAirInterface LTE network, OpenDayLight SDN network, and CUDA GPU computing platform. We have conducted extensive experiments to evaluate the performance of *SafeSlicing*, and the results have demonstrated that *SafeSlicing* significantly improves on system performance and adheres to the SLAs in the dynamic network slicing optimization.

References

[1] I. WP5D, "Minimum requirements related to technical performance for IMT-2020 radio interface (s)," 2017.

[2] I. Afolabi, T. Taleb *et al.*, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[3] X. Foukas, G. Patounas *et al.*, "Network slicing in 5G: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[4] J. Ordonez-Lucena, P. Ameigeiras *et al.*, "Network slicing for 5G with SDN/NFV: Concepts, architectures, and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 80–87, 2017.

[5] J. X. Salvat, L. Zanzi *et al.*, "Overbooking network slices through yield-driven end-to-end orchestration," in *ACM CoNEXT*. ACM, 2018, pp. 353–365.

[6] 3GPP, "Management and orchestration, Architecture framework," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 28.533, 2020, version 16.4.0.

[7] Q. Liu, T. Han, and E. Moges, "Edgeslice: Slicing wireless edge computing network with decentralized deep reinforcement learning," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 234–244.

[8] H. Zhang, A. Zhou, J. Lu, R. Ma, Y. Hu, C. Li, X. Zhang, H. Ma, and X. Chen, "OnRL: improving mobile video telephony via online reinforcement learning," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.

[9] H. Mao, M. Schwarzkopf *et al.*, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*. ACM, 2019, pp. 270–288.

[10] J. A. Ayala-Romero, A. Garcia-Saavedra *et al.*, "vrAIn: A Deep Learning Approach Tailoring Computing and Radio Resources in Virtualized RANs," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.

[11] O. R. Alliance, *O-RAN-WG1-O-RAN Architecture Description - v01.00.00*. Open RAN Alliance, 2020.

[12] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[13] A. D. Laud, "Theory and application of reward shaping in reinforcement learning," Tech. Rep., 2004.

[14] ETSI. ETSI Zero touch network & Service Management (ZSM). [Online]. Available: https://www.etsi.org/technologies/zero-touch-network-service-management

[15] 3GPP, "3GPP TR 28.801, Study on management and orchestration of network slicing for next generation network," 3GPP, Tech. Rep., 2018.

[16] ETSI, "ETSI 5G Management and orchestration, 5G Network Resource Model (NRM), v15.0.0," ETSI, Tech. Rep., 2018. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/128500_128599/128540/15.00.00_60/ts_128540v150000p.pdf

[17] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[18] G. Association, *Generic Network Slice Template, V2.0, https://www.gsma.com/newsroom/wp-content/uploads//NG.116-v2.0.pdf*. GSM Association, 2019.

[19] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2004.

[20] S. Paternain, L. Chamon *et al.*, "Constrained reinforcement learning has zero duality gap," in *Advances in Neural Information Processing Systems*, 2019, pp. 7553–7563.

[21] R. S. Sutton, D. A. McAllester *et al.*, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[22] G. Dalal, K. Dvijotham *et al.*, "Safe exploration in continuous action spaces," *arXiv preprint arXiv:1801.08757*, 2018.

[23] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[25] C. Marquez, M. Gramaglia, M. Fiore *et al.*, "How should i slice my network?: A multi-service empirical evaluation of resource sharing efficiency," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 2018, pp. 191–206.

[26] A. Paszke, S. Gross *et al.*, "Automatic differentiation in pytorch," 2017.

[27] T. P. Lillicrap, J. J. Hunt *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

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

[29] OpenAirInterface Software Alliance. OpenAirInterface repository. https:gitlab.eurecom.fr/oai/openairinterface5g, 2017.

[30] J. Medved, R. Varga *et al.*, "Opendaylight: Towards a model-driven SDN controller architecture," in *IEEE WoWMoM 2014*. IEEE, 2014, pp. 1–6.

[31] OpenAirInterface Software Alliance. Openair-cn repository. https:gitlab.eurecom.fr/oai/openair-cn, 2017.

[32] C. Nvidia, "Nvidia CUDA C programming guide," *Nvidia Corporation*, vol. 120, no. 18, p. 8, 2011.

[33] N. McKeown, T. Anderson *et al.*, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[34] D. Yu, Y. Li *et al.*, "Smartphone app usage prediction using points of interest," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, p. 174, 2018.

[35] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE CVPR*, 2016, pp. 779–788.

[36] J. Achiam, D. Held *et al.*, "Constrained policy optimization," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 22–31.

[37] B. Han, V. Sciancalepore *et al.*, "A utility-driven multi-queue admission control solution for network slicing," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 55–63.

[38] S. D'Oro, L. Bonati *et al.*, "Sl-EDGE: Network slicing at the edge," *arXiv preprint arXiv:2005.00886*, 2020.

[39] D. Bega, M. Gramaglia *et al.*, "Deepcog: Cognitive network management in sliced 5G networks with deep learning," 2019.

[40] Y. Liu, J. Ding, and X. Liu, "A constrained reinforcement learning based approach for network slicing," in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*. IEEE, 2020, pp. 1–6.