Received 10 November 2022; accepted 30 November 2022. Date of publication 9 December 2022; date of current version 4 January 2023.

Digital Object Identifier 10.1109/OJCOMS.2022.3227591

DRL-Based Slice Admission Using Overbooking in 5G Networks

SHIVANI SAXENA AND KRISHNA M. SIVALINGAM (Fellow, IEEE)

Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai 600036, India

CORRESPONDING AUTHOR: K. M. SIVALINGAM (e-mail: skrishnam@cse.iitm.ac.in)

This work was supported by the VMWare University Research, USA.

ABSTRACT In 5G-and-beyond networks, the concept of *Network Slicing* is used to support multiple independent and co-existing logical networks on a physical network infrastructure. The infrastructure provider (InP) owns the set of virtual and physical resources that are used to support the tenant slice requests. Each slice request specifies a service level agreement (SLA) that contains the required slice-level resources (computation and communication) and the revenue provided by the tenant. Due to limited resources, the InP cannot accommodate all requests made by the tenants. In general, it has been found that tenants tend to overestimate their resource demands (e.g., for 5G Core computation) to reduce possible SLA violations. In this paper, we consider two major slice types: Elastic (low priority and low revenue) and Inelastic (high priority and high revenue). We apply the concept of overbooking, where the InP accepts more slices while considering slice priorities in order to maximize the overall revenue and utilization. We consider a multi-tenant environment and propose a slice admission system named PRLOV, which is based on Reinforcement Learning (RL) and prediction methods. The system predicts the future resource demands of Elastic slices and applies an opportunistic overbooking technique to overbook InP for accepting more slices. In addition, the admission decision is formulated as an Markov Decision Process (MDP) problem and solved using standard RL techniques (Policy Iteration, Q-Learning, DQN). The performance of our proposed work is compared against three other heuristics (Basic, Prediction, PRL) that do not use overbooking. Data traces from the Materna data center networks were used for prediction purposes. The important performance metrics measures include InP total revenue, the acceptance rate of respective slices and overall resource utilization for different slices. The results show that the proposed work significantly outperforms the other mechanisms in terms of revenue gain and resource utilization. Simulation results show that PRLOV provides a revenue gain of 6%, 26%, and 102% compared to the PRL, Prediction and Basic scheme.

INDEX TERMS 5G networks, network slicing, infrastructure provider, overbooking, reinforcement learning, prediction, slice priority.

I. INTRODUCTION

NETWORK Slicing (NS) is an important aspect of 5G networks [1], [2], where a shared physical network is divided into multiple virtual networks having tailored capabilities. In 5G networks, the network infrastructure provider shares its underlying resources for specific use cases standardized by 3GPP and ITU-R [3], such as enhanced Mobile Broadband communication (eMBB), Ultra-reliable low-latency communication (URLLC) and machine-to-machine type communication (MMTC) [4].

The concepts of Software Defined Networking (SDN) [5] and Network Function Virtualization (NFV) [6] enable the implementation of network slicing. NFV enables a slice to perform its functionality by emulating hardware with virtual machines, whereas SDN manages the network traffic using concepts such as control and data-plane separation. The functionality of a network slice is realized using a group of Network Functions (NFs) that run on virtual machines as Virtualized Network Functions (VNFs) [7]. Slice resources can be generalized into two types: link and node resources.

Link resources relate to bandwidth, link buffers, etc. whereas node resources consist of main memory and computing (CPU/GPU cores).

In this paper, we focus on node NFs for a slice, assuming that enough link capacity is provided unless there is a topology change or any fault, which occurs rarely. We consider two network entities: Tenant and Infrastructure Provider (InP). Tenants are the users of a slice, while the InP provides slice-as-a-service (*SlaaS*) to the tenants to manage the required service resources. The overall resource requirements for the NFs of a slice are determined by the tenant and the corresponding physical resources are requested from the InP.

Slices are categorized into different slice types, based on the type of resources and services that a slice needs to support. Slice types can be defined based on two scenarios: Use cases or verticals (eMBB, mMTC, URLLC) and Elasticity of resource requirements. In this paper, we consider slice types based on their Elastic nature. In Inter-slice admission, a tenant requests an independent slice from the InP. The InP can accept a slice only when it can allocate the desired resources, maintaining the Service Level Agreement (SLA) and Quality of Service (QoS) [8]. A 5G network will have a limited amount of InP resources resulting in some of the tenants' demands being not satisfied at a given instant of time. Due to resource limitations and multiple tenants requesting the same type of resources, InP cannot accept all the slices and a brokering scheme needs to be applied. The problem of deciding whether a slice should be accepted or not is termed as *slice admission control*. The request may be accepted or rejected immediately, or the tenant may choose to wait till resources are available. In this paper, we consider the case where a slice is rejected if adequate resources are not available when the request is made.

One of the main goals of the InP is to earn maximum revenue, while admitting as many tenants (slices) as possible without violating SLA constraints. This goal can be achieved by: (1) admitting more slices, by utilizing resources efficiently, and (2) giving preference to higher priority slices while admitting. The two approaches are contrary to each other since one wants to admit more and more slices which can lead to SLA violations (SLAV) due to resource deficiency. In comparison, admitting higher priority slices will require guaranteed QoS, which means reserving the entire requested demand throughout their lifetime.

In this work, we have proposed an approach that considers the two conflicting goals and present a balanced way of maintaining resource utilization. It has been observed from past trends that tenants tend to over-estimate resources demand in order to reduce SLA violations for the slice's end-users [9]. Hence, some resources will remain idle. This situation can be improved by an admission technique that admits more slices to use these idle resources. InP, on the other hand, has to provide service quality to make a profit.

We use the concept of *overbooking* given that it does not use all resources and at the same time gives higher priority to slices that pay higher revenue on admission. Overbooking [10] refers to accepting more requests when requested demand exceeds the available system capacity. Overbooking is frequently practiced in airlines and hostel bookings to increase their profit, assuming cancellations and no-shows. Here, overbooking is considered since tenants tend to request more resources, during admission request, than the actual usage. Hence, we overbook the InP to accept more slices than the capacity permits [9].

Slices are classified into two categories: *Inelastic* slices, also known as higher priority slices; and *Elastic*, also known as best effort slices. Inelastic slices are very stringent in their resources requirements but offer higher revenues. Any SLA violation in Inelastic slices will lead to the InP paying a high penalty. Elastic slices, on the other hand, have flexible SLA requirements but offer lower revenues.

We propose a method called Prediction and RL based Overbooking (PRLOV) that tries to balance the twin goals of InP, increased resource utilization and providing better services. We present a Slice Admission architecture where, a Slice Forecasting block (SRF) predicts the future resource (CPU, Memory, Bandwidth) usage of currently active Elastic slices for the next time window using the Long Short-Term Memory (LSTM) approach [11]. This predicted information is used by an opportunistic overbooking heuristic, where the system allocates the required resources to each slice. Thereafter, we admit the new slices using an intelligent policy on the basis of available resources.

This work is the extension of our previous work [12] which solves admission using a simple overbooking method and Q learning. In this paper, the state entity is modified to a reduced form. Also, the admission technique is improved by using Deep Reinforcement Learning (DRL) [13] over Q-Learning [14] for reducing state space and time complexity. The admission block is generalized, and a benchmark algorithm (policy iteration), which is a dynamic approach, is also formulated to compare the performance of the DRL scheme. An efficient and rigorous overbooking method is also proposed rather than a simple method, which uses RL as feedback to adjust the overbooking amount through exploration and exploitation techniques.

The proposed PRLOV method is evaluated on the Materna data centre traces which is used to generate the slice resource data [15]. It consists of three months data for virtual machines (VMs) that consists of different performance metrics. The performance of our proposed work is compared against three other heuristics (Basic, Prediction, PRL) that do not use overbooking. The results show that the proposed work significantly outperforms the other three mechanisms in terms of revenue gain and resource utilization. PRLOV show about 102%, 26%, 6% revenue gains against the other three algorithms respectively.

The main contributions of this paper are: (i) use of Long Short-Term Memory (LSTM) [11] based forecasting with the application of opportunistic overbooking to allocate resources at InP efficiently; (ii) consideration of an online setting based on a reinforcement learning [16] model to

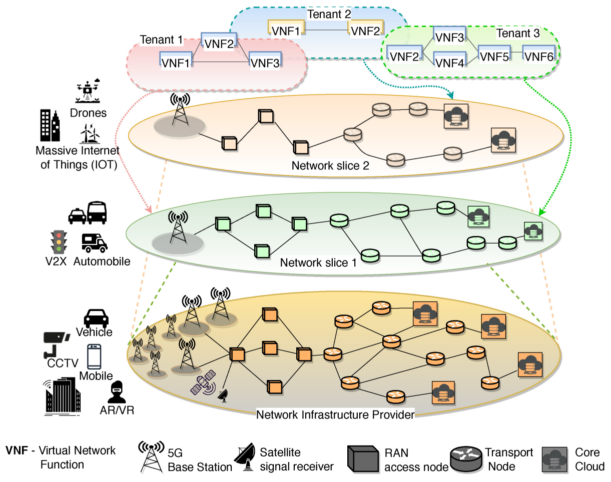


FIGURE 1. **Network Slicing in 5G networks.**

dynamically accept or reject slices; and (iii) training of slice usage based on the Materna data center traces [15] and a detailed performance study based on this training.

With respect to existing work, the main novelty is the use of overbooking concept coupled with LSTM and RL. The overbooking concept uses the RL feedback mechanism and converges automatically to the best overbooking factor, with the help of predicted information from LSTM based slice resource forecasting. The design of RL policy to accept or reject slices in the framed system is novel. Most of them have not considered a real dataset; and none of them are considering the actual resources requirements and wasting a lot of resources. A detailed comparison is presented in Section II.

The remainder of the paper is organized as follows. Section II presents a literature review. Section III presents the system model and gives a detailed view of system architecture used in designing the model. Section IV details how the problem is formulated for the RL agent to optimize the slice admission decision. Section V presents the performance evaluation results for different scenarios. Section VI summarizes the paper.

II. BACKGROUND AND RELATED WORK

This section presents the relevant background material and related work.

The system architecture of a 5G network is composed of three major components: Access Network, Transport Network, and Core [17]. A network slice is defined as a logical end-to-end network that utilizes resources from the underlying network for a specific application vertical or a client. Fig. 1 depicts the concept of network slicing, where each slice spans these network components; this is known as end-to-end network slicing [18]. In this paper, we consider crucial 5G Core domain resources, i.e., Compute, Memory and Bandwidth; however, the work can be extended to include access and transport network resources.

The system consists of two network entities: Infrastructure Provider (InP) and Tenants. InP provides relevant services, and tenants are the consumer of services. When a tenant requests its slice to the InP, the Communication Service Management Function (CSMF) translates communication-related resources into slice requirements. The Network Slice Management Function (NSMF) uses the information to create a slice blueprint called a network slice template (NST). The NFV resource requirement for NST is mapped to a network slice descriptor (NSD) available in the InP slice catalog, managed by the NFV Management and Network Orchestration (MANO) entity. When a slice blueprint is created, we install the admission control module to determine if the requested slice should be accepted.

The InP has to decide whether to admit each slice or not, based on the available and requested resources, priorities among slices and revenue generated. One of the chief objectives of slicing admission control is to maximize the InP's profit, while ensuring efficient resource utilization, fairness among slice types and other factors.

Machine Learning techniques have been applied for various networking problems in recent years [30]. In particular, several works in Network Slicing have considered the application of Machine Learning (ML) techniques [31] to address the slice admission problem. Multiple deep learning solutions have been applied for predicting the resource demand of slices, and based on the remaining resources; a slice admission decision is taken. The work in [23] presents an admission scheme for slices using predictions using big data analytics for resource requirements of existing slices. The admission is performed based on whether the requested slice and existing slices could be scaled up or not. The objective is to increase InP profit by minimizing losses incurred when the accepted slices can not be scaled when required. In [21], an RL-based 5G network slice broker (RL-NSB), built on predictions using Holt-Winters [32] theory, is presented. The slices are accepted and scheduled on resource availability in First-Come-First-Serve (FCFS) order. The number of SLAVs for each slice is fed back to the forecasting engine using RL, and parameters of the prediction algorithm are updated on frequent violations.

The work in [25] uses probabilistic forecasting based DeepAR to predict slice resources from a real dataset and admits slices based on resource availability. In [20], two slice classes, namely Best Effort and Guaranteed Service are considered. It uses a genetic algorithm for accepting slices intelligently and shows that it is better than Q-Learning. The drawback of genetic algorithms is that they rely on the quantized fitness values [33] of different overall strategies instead of the reward value of every single action. The authors of [22] present a Deep RL-based admission policy that uses neural networks in the context of 5G radio access networks (RAN). The objective is to minimize the losses from rejected slices and degraded services. The proposed algorithm performs greedily such that it learns to selectively reject the slices

TABLE 1. Summary of related works on slice admission control in 5G networks.

Paper	Domain(s)	Dataset	Priority	Approach(es)	Objective(s)	Admission Policy
Bega et al. 2017 [19]	RAN	Synthetic	**Y**	RL	Revenue maximization, QoS control	Optimal
Han et al. 2018 [20]	End-to-End	Synthetic	**Y**	Genetic Algorithm	Resource utilization	Greedy
Salvat et al. 2020 [9]	End-to-End	Synthetic	N	Heuristic	Loss minimization, Resource utilization	Overbooking
Sciancalepore et al. 2019 [21]	–	Synthetic	**Y**	Prediction, Heuristic, RL	Resource Utilization, Minimizing SLA violation	User spatial distribution
Raza et al. 2019 [22]	RAN	Synthetic	**Y**	DRL	Revenue maximization	Greedy
Raza et al. 2019 [23]	RAN, Core	Synthetic	N	Prediction	Revenue maximization	FCFS
Bega et al. 2020 [24]	RAN	Synthetic	**Y**	DRL	Revenue maximization, QoS control	Optimal
Jiang et al. 2022 [25]	–	**Real**	**Y**	Prediction	Revenue maximization, Resource utilization	FCFS
Sulaiman et al. 2022 [26]	RAN	Synthetic	**Y**	DRL	Revenue maximization, Slicing	Multi agent
Haque et al. 2022 [27]	–	Synthetic	**Y**	DRL	Revenue maximization	Optimal
Wu, Zhouxiang et al. 2021 [28]	–	Synthetic	**Y**	RL	Revenue maximization	Offline
Villo et al. 2022 [29]	Core	Synthetic	N	DRL	Revenue maximization, Resource utilization	Optimal
Our work	Core	**Real**	**Y**	Prediction, Heuristic, DRL	Revenue maximization, Resource utilization, QoS control	Optimal, RL Overbooking

that generate low revenues (i.e., Low Priority) in favor of those that generate higher revenues (i.e., High Priority).

In [19], the authors derive an analytical model of slices traffic requests from tenants to InP considering only the spectrum resource. It provides a relationship between slices that can be accepted inside the admissibility region. In this work, the admission problem is modeled as a Markov Decision Problem (MDP), where an optimal policy is derived using Q-Learning in RL for the admission of Inelastic and Elastic slices. The work in [24] extends that of [19] by using the DRL technique over Q-Learning to reduce high state-space complexity and early convergence.

The work in [26] solves the joint admission and network slicing problem using DRL for multi-RL agents. The work in [29] considers use Core VNF resources and consider different slices types using a DRL approach. It does not consider slice priorities. The scheme in [28] uses a recurrent neural network (RNN) as encoder and RL to train the model for slice admission decision, in an offline manner. The work in [27] also addresses slice admission using the DRL approach. All these works have a common objective to increase InP revenue.

Apart from using ML and RL techniques, the work presented in [9] proposes a probabilistic model for yield-driven end-to-end slice orchestration and applies slice overbooking. In this work, a controller decides on the admission control and resource reservation (AC-RR) using monitoring and forecasting information. The AC-RR problem is formulated as an optimization problem to minimize the penalty caused due to SLAV, which is solved using an offline method. The profit gained from different slices is not taken care of in this work, and the slices are served on an FCFS basis.

Table 1 presents a summary of the existing approaches and our proposed work. The drawbacks of the these approaches are that some consider effective resource utilization with QoS control, and some consider intelligent admission for revenue management, but none consider both. Also, none of the mentioned works use real data except [25]. However, it uses hard coding to admit slices concerning slice priorities and only focus on prediction techniques. Our work focuses on effective resource utilization while maintaining the QoS and increasing revenue using real-time ML techniques. The proposed work predicts the future resources demand of existing Elastic slices based on history using Recurrent Neural Networks (RNN) [34]. We consider over-estimation of resources, particularly in the core domain, and apply overbooking InP to reduce it. Further, we consider online slice request arrivals that are dynamically admitted based on the availability of resources using prediction information and overbooking factor.

III. SLICE ADMISSION FRAMEWORK

This section presents the proposed admission control framework and related definitions.

A. SLICE MODEL

The slice's Service Level Agreement (SLA) defines the service class of slice as Inelastic (I) type or Elastic (E). Inelastic slices have strict service quality requirements and offer higher bids for using its slice. The Elastic slices are flexible. We consider a set of service classes, $l \in \{I, E\}$, where every slice class is represented as $<\lambda, \mu, \rho, \nu, \phi>$. Here, λ is the arrival rate of slices belonging to class l based on an arrival process, μ is the slice duration mean, ρ

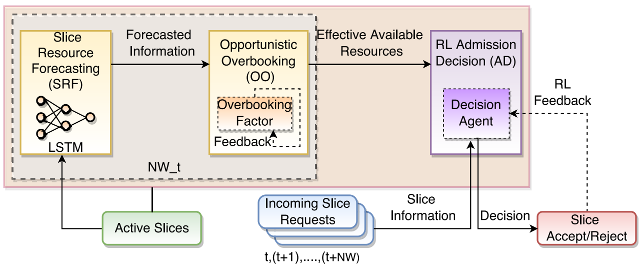


FIGURE 2. Slice admission control framework.

is the bid value, ν is the penalty given on any SLA violation depending on service class, and ϕ is the specific slack value which is degradation tolerance level of any slice. We define $\phi = 0$ for $l = I$ since Inelastic slices cannot suffer from any service degradation, whereas ϕ ranges from 0.2 to 0.3 for elastic slices.

In the admission control block, we represent the incoming slice request u as $<N_r, d, p, g, l>$, where N_r denotes resource demand for r type of resources, d is slice holding duration, p is per-unit cost (bid value) of using a unit InP resource, g is the slack value and l is the service class of slice u. Here, r denotes different types of physical resources required such as CPU, Memory and Bandwidth. The p and g values of a slice u are dependent on its slice class l, with $p_u = \rho_l$ and $g_u = \phi_l$. Also, it is assumed that $\rho_I > \rho_E$.

Three 5G Core resource types are considered: Computation as the number of CPU Cores, Storage as Memory (GB), and Network as Bandwidth (Mbps). The InP has a fixed capacity of available resources. At every point in time, the system constraints listed in Eq. (1) must be satisfied, such that the summation of all the allocated resources to existing slices is less than or equal to the overall system resources.

$$\sum_{k=1}^{n} c_k \leq C, \sum_{k=1}^{n} m_k \leq M, \sum_{k=1}^{n} b_k \leq B \tag{1}$$

Here, C, M, B are the maximum available system capacities of resources in CPU cores, memory, and bandwidth, respectively; c_k, m_k, b_k are the allocated core, Memory, and bandwidth resources to an existing slice k.

B. PREDICTION-RL-OVERBOOKING (PRLOV)

Fig. 2 presents Prediction-RL-Overbooking (PRLOV), the proposed Admission Control (AC) based 5G system architecture. It includes an LSTM-based prediction framework (*SRF*) to predict the following resource requirements of active E slices in each network window (*NW*). With the help of prediction, we apply opportunistic overbooking (*OO*) at InP to improve resource utilization by creating space for more slices acceptance. In addition, an adaptive decision algorithm (*AD*) is designed to decide the acceptance or rejection of an incoming slice. According to the overall framework, opportunistic overbooking is applied to each *NW* in accordance with the prediction results and historical data. The net available resource information is then fed to an intelligent decision algorithm for future slices admission decisions.

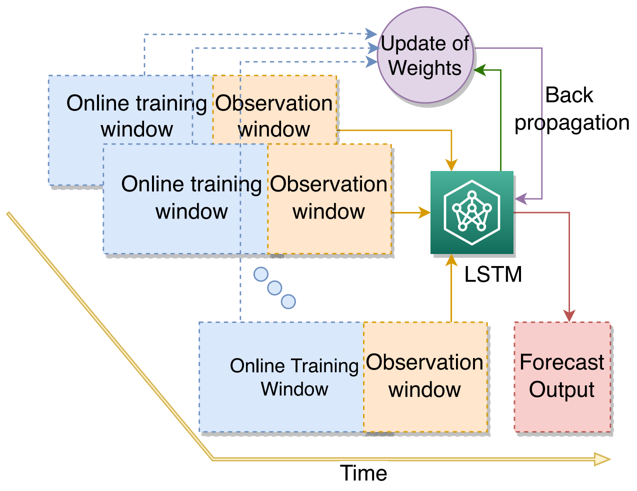


FIGURE 3. Training & Forecasting using LSTM.

The Admission Decision block, shown in Fig. 3, is responsible for deciding whether to accept or reject any incoming slice. A learning-based policy drives the decision making process. Considering the two slice classes, it becomes crucial to admit slices considering different factors that define a class rather than just random or an FCFS accept. Different slice classes have different priorities and offer different bid values to InP. The slice classes can be generalized such that higher priority slices pay a higher profit. To increase InP profit, it needs to make the best decision for itself. Once a decision is taken to accept the slice, it is created and allocated resources. Once a slice is rejected, there are two possibilities; either the tenant has to wait, which can lead to balking; or it is discarded, and the tenant must request a new slice again. We opt for the second option in this paper. We propose a decision policy aiming to maximize InP's overall profit, as described in Section IV.

C. SLICE RESOURCE FORECASTING (SRF)

The Slice Resource Forecasting (SRF) component is used to predict the future resource demand of CPU, Memory, and Bandwidth for all active slices at InP. Since none of the predictions are perfect, if we allocate resources based on forecasting results in I class slices, any under-prediction will cost a very high penalty on InP. Hence, we apply predictions only for accepted slices that belong to E type service class. The system maintains a list of currently active slices. At the start of every new window NW_{t+1}, the maximum resource demand of all active slices is predicted for the current window. Since our focus is not on designing a new prediction algorithm, we use existing deep neural networks techniques for prediction.

Due to high uncertainty in 5G data, we use the Long Short-Term Memory (LSTM) technique for prediction [11].

An LSTM neural network is designed such that it consists of an input, an output layer, and a single hidden layer. As a thumb rule, only one hidden layer is enough to train a neural network for simple predictions. We construct three separate neural networks for each CPU, Memory, and Bandwidth prediction. We use n-step prediction where a single RNN takes as input the past T observations and outputs the next n steps resource demand for each slice. In time series prediction, if a slice resource usage is given as $\{r(t)|t = 1, 2, \ldots, T\}$, the n^{th} prediction on resource usage for time $t+n$ is a function f such that $q_k(t+n) = f(r(t), r(t-1), \ldots, r(1))$. Here, f is a mapping from the past resource demands to the predicted resource usage q for n time steps ahead; n is defined as the Network Window (NW) in this paper.

This is represented by:

$$\text{InputLayer} : c_k(t), c_k(t-1), \ldots, c_k(2), c_k(1)$$
$$\text{OutputLayer} : q_k(t+1), q_k(t+2), \ldots, q_k(t+n)$$

Here, $c_k(1)$ denotes the actual CPU usage of past T^{th} time from t and $q_k(t+n)$ denotes the predicted CPU usage of n^{th} step ahead of time t.

A similar model is used for Memory (m) and Bandwidth (b) predictions. The LSTM weights are updated such that the algorithm learns the neuron's weights by running the back-propagation through time [35]. Fig. 3 shows the weights update process for traffic flow forecasts using an online training window. Here, the oldest features are discarded, and only the most recent observations are retained. The observation window here is the Input layer for LSTM. Using trial and error method, the number of hidden neurons is chosen to be 8 while using the Mean Squared Error (MSE) [36] loss function.

When a slice k is accepted, the maximum resource demand ($N_{r,k}$) is allocated since there is not enough data for training. The agent retrains its slice model to forecast better when enough new data become available or if the model consistently under-predicts. Slices that the agent accepts are dropped when they depart. In the next block, overbooking is managed based on the predicted information. This process is repeated for each start of NW.

D. OPPORTUNISTIC OVERBOOKING (OO)

The need for overbooking arises since prediction may not be perfect for all slices. This may result in under-predictions, accurate prediction, or over-prediction of the resource demand. Resources are allocated to each slice based on prediction results, so in case of under-prediction, we have no choice but to penalize the InP. In case of over-prediction, we have an opportunity to overbook the InP to accept more slices. The system goal is to maximize the InP profit through overbooking with minimal or no Service Level Agreement Violation (SLAV). When there are not enough resources for new slices to be accepted, the system attempts to overbook slices. One of the main challenges with overbooking is finding the exact amount of each type of resource that needs to be overbooked. We rely on an RL feedback mechanism to manage the overbooking.

TABLE 2. Symbols and notations.

Notations	Meaning
I, E	Inelastic, Elastic
l	Slice class
λ	Poisson arrival rate of a slice class
μ	Exponential mean duration of a slice class
ρ	Bid value of slice class
ν	Service degradation penalty of a slice class
ϕ	Degradation tolerance level of a slice class (in %)
$N_{r,k}$	Demand for resource r by a slice k
d	Holding duration of a slice
p	Price (bid) per unit time to use a resource of InP
g	Slack value of a slice
C, M, B	Total CPU Cores, Memory, Bandwidth of InP
c, m, b	CPU Cores, Memory, Bandwidth usage of a slice
RC_r	Total resource capacity of InP of resource r
NC_r	Net Capacity of InP resources of resource r
AL_r, AV_r	Total Allocated & Available resources of InP
IO_r, FO_r	Initial & Final overbooking factor
PU_r	Predicted resource usage of all Elastic slices
Υ	Overbooking regulating factor
ζ	Overbooking increment value
β	Overbooking future assurance factor
L_v, L_{nv}	Likelihood of SLAV and No SLAV
$\mathcal{TR}$	Total Final Revenue
$s, \mathcal{S}$	State & State Space
$a, \mathcal{A}$	Action & Action Space
$\mathcal{R}$	Reward Function
e	Event
$\hat{\mathcal{P}}$	Transition Probability
A	Set of Active Slices
V	Value Function
γ, α	RL Agent Discount Factor & Learning Rate
$\mathcal{PY}_{rla}$	RL Agent Penalty due to slice rejection
$\mathcal{PY}_{inp}$	InP penalty due to SLA violations

During execution, the InP at any time t has slices that may belong to either the Inelastic or Elastic category. At the start of every network window, we calculate the amount of each resource allocated to all Inelastic slices and obtain the net available capacity for each resource r from the following Eq. (2).

$$NC_r = RC_r - \sum_{k \in I} AL_r \tag{2}$$

After finding the resources allocated to Inelastic slices, the predicted usage of resource r for all the Elastic slices (PU_r) is calculated. An initial overbooking factor IO at time t for resource r is introduced using Eq. (3).

$$IO_r(t) = \frac{NC_r(t)}{PU_r(t)} \tag{3}$$
$$FO_r(t) = \Upsilon_r * IO_r(t) \tag{4}$$

The final overbooking factor FO_r is approximated by regulating the value of Υ_r. The initial value of Υ is set to 1. The value of FO_r is converged to an optimum value by exploring the state space defined using RL. The state space consists of a vector that keeps information on the Net capacity, (PU_r) and FO_r. Each vector in the state space is mapped to two values that denote the likelihood of SLAV (L_v) and no SLAV (L_{nv}), respectively. The two values are filled using the exploration of state space.

When any SLAV is encountered for a corresponding state, L_v is updated with a penalty of -1; and L_{nv} is updated by giving a reward of $+1$. The values are updated using Eq. (5)-(6).

$$L_v = L_v - 1 + \beta L_v^{max} \tag{5}$$
$$L_{nv} = L_{nv} + 1 + \beta L_{nv}^{max} \tag{6}$$

Here, β is a measure of assurance based on the next state. In case of SLAV, L_v^{max} is a maximum value of the next state with a lower FO by value ζ, whereas L_{nv}^{max} is a maximum value of the state with a higher FO. A greedy approach is used here, such that if an upper value of FO does not result in a SLA violation, then a lower value will also not reveal it and vice versa. A higher magnitude of L_v than L_{nv} denotes a higher chance of SLAV occurrence. Initially, both variables are set to 0. For any state, if the magnitude of L_v is more than L_{nv}, the FO_r is calculated by decreasing Υ by ζ; else Υ is increased by value ζ and a new FO_r is derived using Eq. (4). The traversal process is repeated, keeping the rest of the entities of the vector same and only changing FO_r until we reach a state where the magnitude of $L_v \leq L_{nv}$. The value of ζ is kept small so that the overbooking factor is gradually increased or decreased for a better convergence value to minimize service degradation due to excess overbooking. The above process is repeated for every resource r.

When FO_r ($r \in (c, m, b)$) converges to the optimum value, the available amount of resources (AV) of each type is calculated for admitting new slices using Eq. (7):

$$AV_r = NC_r FO_r - PU_r \tag{7}$$

E. ADMISSION DECISION (AD)

This block is responsible for deciding whether to accept or reject any incoming slice. The major challenge is that, InP is unaware of the arrivals of future slices. When Inelastic slices arrive, InP should have space to admit. The InP must be aware of the arrivals of Inelastic slices to make better reject some Elastic slices, which makes the problem NP hard to solve. This is solved by interacting with the environment through RL. An artificial intelligent policy drives this decision by learning the environment. Different slice classes have different priorities and offer different bid values to InP. The slice classes can be generalized such that higher priority slices pay a higher profit. Once a decision is taken to accept the slice, it is created and allocated resources. Once a slice is rejected, there are two possibilities; either the tenant has to wait, which can lead to balking, or it is discarded, and the tenant must request a new slice again. We opt for the second option in this paper. We propose a decision policy aiming to maximize InP's overall profit. Section IV focuses on designing an effective approach for slice admission.

The objective of admission policy is to maximize the accumulated revenue of InP ($\mathcal{TR}_t$), which is a function of revenue obtained from accepted slices and penalty from SLAV, until time t. This is similar to the objective defined in [23]. This is defined as follows.

$$\mathcal{TR}_t = \sum_k (p_k - \nu_k) \tag{8}$$

Here, p_k is the profit and ν_k is the penalty accrued to SLA violations, of all admitted slices from system start till time t. The value of ν depends on p and a slice degradation level ϕ ($0 \leq \phi < 1$)and given as $\nu = 3p\phi$.

Eq. (8) presents the net revenue obtained by InP at any point of time t. This is the amount of total revenue obtained from all active slices and the penalty from SLA violations faced by the slices (ν). The RL admission control agent, that takes the decision to accept or reject a slice, imposes a penalty of $\mathcal{PY}_{rla}$ for rejecting any Inelastic slice considered during the internal training of the RL agent. It has no relation with the InP net profit or net revenue.

The overall procedure is explained in Algorithm 1, where the input is the set of slice requests (U) at time t, and the output is the revenue $\mathcal{TR}_t$. The algorithm predicts the demand of currently active E slices at every (NW). The net available resources are then calculated after performing the overbooking mechanism. Now, at each time t in NW, the slice admission decision is taken by AD. Based on the admission decisions, the total reward and penalties (P) from slice SLAV is added to obtain net revenue at InP. Meanwhile, any departing slice's resources are also freed.

IV. RL-BASED PROBLEM FORMULATION

This section presents the reinforcement learning (RL)-based formulation of the slice admission decision problem and the different approaches considered, including the proposed DRL-based approach.

A. MARKOV DECISION PROCESS (MDP)

A typical RL consists of an agent that learns and decides on interaction with an environment. The environment sets up the basis of the decision and can not be changed. Whenever an agent takes action, it reaches a state and provides a reward. In RL, the agent's decision at any state is not pre-trained but learns the best action through learning. The Slice admission decision problem can also be solved using RL since the InP does not know which slices will be requested in the future, and the best decision can be learned using the RL technique. Such problems are often mathematically modeled using Markov Decision Process (MDP) [37] in RL. It represents system behavior as a sequence of states with the Markovian Property. An MDP is a five-sized tuple that defines the environment. It consists of States ($s \in S$), Actions ($a \in A$), Reward ($R(s, a)$), Transition Function ($T(s, a)$) and a Discount Factor γ. The state-space, action, and reward functions considered in our work are defined as follows:

1) State Space ($\mathcal{S}$): A state s in state-space $\mathcal{S}$ consists of a four sized tuple consisting of all Available resource (AV_r) from Eq. (7), and event (e). The event $e \in \{l_u, -l_u\}$ represent either the arrivals of

TABLE 3. State transition probabilities.

current state (s)	action (a)	Next Event (e)	Next state (s')	Probability $\hat{\mathcal{P}}(s,a,s')$	Reward $\mathcal{R}(s,a)$
$< c, m, b, l_u >$	0/1	arrival of I	$< c, m, b, I >$ / $< c - c_u, m - m_u, b - b_u, I >$	$\frac{\lambda_I}{\omega}$	0/$(p_u * d_u)$
$< c, m, b, l_u >$	0/1	arrival of E	$< c, m, b, E >$ / $< c - c_u, m - m_u, b - b_u, E >$	$\frac{\lambda_E}{\omega}$	0/$(p_u * d_u)$
$< c, m, b, l_u >$	0/1	departure of I	$< c, m, b, -I >$ / $< c - c_u, m - m_u, b - b_u, -I >$	$\frac{\mu_I}{\omega}$	0/$(p_u * d_u)$
$< c, m, b, l_u >$	0/1	departure of E	$< c, m, b, -E >$ / $< c - c_u, m - m_u, b - b_u, -E >$	$\frac{\mu_E}{\omega}$	0/$(p_u * d_u)$
$< c, m, b, -l_u >$	2	arrival of I	$< c + c_u, m + m_u, b + b_u, I >$	$\frac{\lambda_I}{\omega}$	0
$< c, m, b, -l_u >$	2	arrival of E	$< c + c_u, m + m_u, b + b_u, E >$	$\frac{\lambda_E}{\omega}$	0
$< c, m, b, -l_u >$	2	departure of I	$< c + c_u, m + m_u, b + b_u, -I >$	$\frac{\mu_I}{\omega}$	0
$< c, m, b, -l_u >$	2	departure of E	$< c + c_u, m + m_u, b + b_u, -E >$	$\frac{\mu_E}{\omega}$	0

a slice class $l_u \in \{I, E\}$ or departure of a slice $-l_u \in \{-I, -E\}$ belonging to either Inelastic or Elastic class. It is represented as $<AV_c, AV_m, AV_b, e>$, where u is the requested or departed slice and $c, m, b \in$ (core, Memory, Bandwidth) respectively.

2) Action Space ($\mathcal{A}$): $\mathcal{A}$ represents a set of all possible actions $\mathcal{A} \in \{0, 1, 2\}$ based on events e, where $a \in \mathcal{A}$ represents a single action. Actions $a = 0$ and $a = 1$ indicate the decision to reject and accept any incoming slice respectively and $a = 2$ means do nothing. The action $a = 2$ is triggered upon departure of a slice, in which case the agent does nothing and is not rewarded.

3) Reward Function ($\mathcal{R}$): The reward defines the goodness of taking action a at state s. The agent receives a bonus based on the actions it takes. The possible events are the arrival and departure of a slice. When a new slice u arrives, the agent either rejects or accepts it. On departure, no action is taken, and no reward is generated. Based on the events, the reward function $R(s, a)$ is given as below:

$$\mathcal{R}(s, a) = \begin{cases} d_u p_u; & a = 1 \\ 0; & else \end{cases} \tag{9}$$

Whenever an agent accepts a new slice u, a reward equal to the product of the slice holding duration (d_u) and the price per unit time (p_u) is given to it. A zero reward is given to the agent in case of rejection and departure of any existing slice.

The transition between states from s to s' is taken depending on the transition probabilities $\hat{\mathcal{P}}(s, a, s')$. It is defined as the probability of moving from state s to a new state s' on taking action a. The average time for being on state s is given in Eq. (10) which is a function of arrivals rates and departures, and A is a set of active or existing slices at InP. If a slice k with current resources allocation as (c_k, m_k, b_k) departs at InP or is requested from InP, then the state transition probabilities for possible events and their corresponding next states are listed in Table 3. We can determine the transition probability at a state by dividing the favourable outcome by the total number of possible outcomes using Eq. (10).

$$\omega(\lambda, \mu) = \lambda_I + \lambda_E + \sum_{k \in A, I} \mu_I + \sum_{k \in A, E} \mu_E \tag{10}$$

Algorithm 1: PRLOV Slice Admission Algorithm

```
Input: slice request set $U$: $< N_r, d, p, g, l >$
Output: $\mathcal{TR}[t]$
1  $\mathcal{TR} \leftarrow \mathcal{TR}[t-1]$
2  for each NetworkWindow(NW) do
3      for slice $k \in$ Active slices do
4          if k.l == $E$ then
5              SliceForecastingAgent(k)              // SRF
6      OpportunisticOverbooking()                    // OO
7      for each time t in NW do
8          $\mathcal{PY}_{inp} \leftarrow 0$
9          for slice k ∈ Active slices do
10             $\mathcal{TR} \leftarrow \mathcal{TR} + p_k$
11         for each slice u ∈ U              // this is
           another comment
12         do
13             admitstatus = AdmissionDecision($u$)
               // AD
14             if admitstatus == 1 then
                   /* Add u to active slices
                   queue */
15                 $\mathcal{TR} \leftarrow \mathcal{TR} + p_u$
16                 AllocateResource($u$)
17                 UpdateAvailableResource
18         if SLAViolation for k ∈ Active slices then
19             $\mathcal{PY}_{inp} = \sum v_k$
20         for each slice k.t == DepartureTime do
                   /* Remove k from Active
               slices queue */
21             ReleaseResource(k)
22             UpdateAvailableResources
23     $\mathcal{TR}[t] \leftarrow \mathcal{TR} - \mathcal{PY}_{inp}$
```

B. DYNAMIC ALGORITHM: POLICY ITERATION

Policy Iteration [38] is one of the dynamic algorithms in RL that is used to learn the optimal policy in an MDP environment when we are aware of transition probabilities. Similar to [39], we have applied policy iteration for our goal, which tries to

Algorithm 2: Policy Iteration

```
Output: $\pi^*(s)$
1  $\pi(s) \in \mathcal{A}$, $V_\pi(s)$=0 $\forall$ $s$
   /* Policy Evaluation */
2  while $\delta > min$ do
3      t ← t+1
4      $\delta \leftarrow 0$
5      for $s \in \mathcal{S}$ do
6          $V = V_\pi(s)$
7          $V_\pi(s) \leftarrow \sum_a \pi(s). \sum_{s'} \hat{\mathcal{P}}^a_{ss'}(\mathcal{R}(s,a) + \gamma V_\pi(s')$
8          $\delta \leftarrow \max(\delta, |V_\pi(s) - V|)$
   /* Policy Improvement */
9  for $s \in \mathcal{S}$ do
10     $a = \pi(s)$
11     $\pi(s) \leftarrow argmax_a(\sum_{s'} \hat{\mathcal{P}}^a_{ss'}(\mathcal{R}(s,a) + \gamma V_\pi(s'))$
12     if $\pi(s) \neq a$ then
13         Goto Step 2
14 $\pi^* \leftarrow \pi$
```

maximize the long-term reward of the InP. It follows three steps to find the best policy as illustrated in Algorithm 2. This is defined as one of the baseline approaches.

Algorithm 2 learns the optimal policy π^* using the optimal value function V^*. It first considers an arbitrary policy π and assigns an arbitrary value to V then iteratively improves the policy and value function. An optimal policy π^* is optimal only when $V_{\pi^*}(s) \geq V_\pi(s)$ $\forall$ s and $\pi^* \geq \pi$ $\forall$ π.

$$V^*(s) \leftarrow max_\pi V_\pi(s) \tag{11}$$

The policy is randomly initialized to accept every request if resources are available and the state values $V(s)$ are set to be 0. In the evaluation part, the values are updated using the transition probabilities in Table 3 and a difference between the current value and updated value is calculated until it reaches a minimum. After evaluating the policy that stores the discounted reward for every state as $V(s)$, the next step is taken to improve the policy. γ is the discount factor which is a constant between 0 and 1. The improvement in policy means it learns the best action that should be taken in each state.

This algorithm fails when we do not know the transition probabilities as defined in Table 3. We do not know the arrivals of future slices, so we can not be sure of the same transition. Also, when state space is increased, it is hard for the algorithm to converge faster, and both the time and space complexities increases. To solve this problem of unknown transition probabilities, we propose another existing algorithm for our framework, i.e., Q Learning.

C. Q LEARNING

The problem with Policy Iteration is the high computation and convergence time. Also, the InP does not have any information about the future incoming slices and is unaware of the arrival rates and their duration. Q Learning is a well-known algorithm in RL. It is different from Policy Iteration since it is an off-policy method. The goal is to learn the optimal policy by exploring the environment without being greedy for action values. Q is also known as a model-free technique since its transition probabilities are absent. Minor modifications are made in the above state, action, and reward function for Q Learning. In Q, we do not consider and departure of a slice as an event since no action needs to be taken. The state space is kept quite the same as in Algorithm 2 except the last value. A state s is represented as a number of available resources and the slice class that is being requested. When $a \in \{0, 1\}$ is taken, it either rejects or accepts a slice. The reward function $\mathcal{R}(s, a)$ is modified as explained in Eq. (14) below.

$$s \in \ <AV_c, AV_m, AV_b, l_u> \tag{12}$$

$$a \in \{0, 1\} \tag{13}$$

$$\mathcal{R}(s,a) = \begin{cases} d_u p_u; \ a = 1 \\ 0; \quad\ a = 0, l_u = E \\ -Z; \quad a = 0, l_u = I \end{cases} \tag{14}$$

Here, $l_u \in \{I, E\}$. As a reward, the agent gets a zero reward for rejecting slices in the E class, and a penalty equal to $\mathcal{PY}_{rla}$ is rewarded for rejecting slices in the I class.

The Q Learning agent maintains a Q table that keeps Q values for all possible states in $\mathcal{S}$ and actions A each, denoted as $Q(s, a)$. Q values are the agent's expected reward at state s for taking action a. The initial $Q_0(s, a)$ is set to 0, $\forall s \in \mathcal{S}$. Whenever the agent takes an action a, it transitions from its current state s to next state s'. The $Q(s, a)$ is updated using the following Bellman optimality Eq. (15). In this case, α represents the rate of learning of new Q values; γ represents the discount factor that balances the current and future rewards and decides how much importance to give to future rewards; $\max_{a'}(Q(s', a'))$ represents the maximum expected future reward in the case where the next state will be s'.

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left\{R(s,a) + \gamma \max_{a'}\big(Q(s',a')\big)\right\} \tag{15}$$

In Q Learning, when a slice departs, its allocated resources are released, and the agent takes no action. On acceptance of a new slice, the agent transitions from the previous state s to the next state s' and receives a reward which is added to the InP's expected revenue ($\mathcal{TR}$). When the agent decides to reject a slice, it remains at its previous state s. When enough resources are not available to fulfill the slice request, the slice is always rejected. The transitions from one state to other in Q Learning is explained in Table 4. We aim to maximize the revenue in the long run, so we accept more I slices since they produce a higher profit. It is accomplished by imposing a penalty of $\mathcal{PY}_{\nabla\updownarrow\dashv}$ on the agent for rejecting each slice of I. After the learning phase of Exploration-Exploitation principle [40] is over, the policy is learned that maximizes the revenue of InP.

TABLE 4. Q transitions.

current state (s)	action (a)	Next state (s')	Reward $\mathcal{R}(s,a)$
$< c, m, b, I >$	0	$< c, m, b, I >$	0
$< c, m, b, E >$	0	$< c, m, b, E >$	0
$< c, m, b, I >$	1	$< c - c_u, m - m_u, b - b_u, I >$ / $< c - c_u, m - m_u, b - b_u, E >$	$(p_u * d_u)$
$< c, m, b, E >$	1	$< c - c_u, m - m_u, b - b_u, I >$ / $< c - c_u, m - m_u, b - b_u, E >$	$(p_u * d_u)$

Algorithm 3: Q Learning

```
Input: slice request $u < N_r, d, p, g, l >$, Available
       Resources $< AV_c, AV_m, AV_b >$
Output: Accept(1)/Reject(0)
1  $\alpha, \epsilon, \gamma$
2  $s \leftarrow (AV_c, AV_m, AV_b, l_u)$
3  if $(N_{r,u} > AV_r \forall\ r)$ then
4      $a \leftarrow 0$                              // action
5  else
6      if $uniform.random() \leq \epsilon$ then
7          $a \leftarrow$ choice(0,1)                // Explore
8      else
9          $a \leftarrow argmax_a Q(s, a)$           // Exploit
10     $s', \mathcal{R}(s, a) \leftarrow$ next state, Reward
11     $Q' \leftarrow \mathcal{R}(s, a) + \gamma \max_{a'}(Q(s', a'))$
                                     /* Update Q value */
12     $Q(s, a) \leftarrow (1 - \alpha).Q(s, a) + \alpha.Q'$
13 return $a$
```

1) EXPLORATION AND EXPLOITATION

This algorithm uses the Exploration-Exploitation feature in Q Learning. The trade-off between exploration and exploitation is balanced by setting a parameter ϵ, determining how often we want to explore other possible states. Exploitation chooses the action of being greedy, which basically exploits the agent's current estimated value to get the most reward. The exploration is taking action randomly from a set of all possible actions with probability ϵ. In exploitation, the agent considers estimated value which is not the actual value, and this may lead the agent to take action which might not be profitable. The exploration is important, which restricts the agent from being greedy all the time and explores other states by taking random actions which may end up being the best action in that state. The best possible method to decide the occurrence of exploration-exploitation is to set ϵ to use a decaying epsilon greedy approach.

The Q Learning algorithm is explained in Algorithm 3, where the input is a slice requests u at time t and output is the admission decision. The Admission decision algorithm gives the decision based on the Q decision. The agent provides the Q decision for the action with the maximum Q value at that state. This decision is given to InP to accept or reject the slice. The Q agent learns the optimal policy on interaction with the environment.

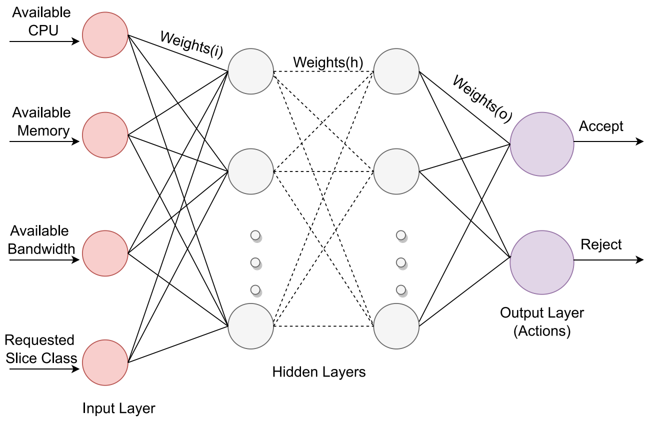


FIGURE 4. Deep neural network.

D. DEEP Q LEARNING (DQL)

In Q Learning, the main problem is the size of the Q table. In the learning phase, it is difficult to maintain the state space when it increases, which hinders exploring all states. Due to improper learning, the agent cannot take the desired action. The problem of state space can be solved using neural network techniques where we need not store the full Q table; instead, we train the neurons and update weights to predict the required action.

The RL algorithms that use neural networks to reduce the state space are known as Deep RL (DRL) algorithms. Since we use the Bellman equation in Q Learning for update of weights while training, it is termed Deep Q Learning, a standard RL algorithm that solves Q using a generalization. It replaces the Q table with a neural network.

In the DQL [13] technique, a neural network's input states are mapped to the Q states and it outputs as all possible actions. The value of output nodes is equal to the Q's action values. Fig. 4 depicts the neural network used to solve AD using DQL. It consists of an input layer, one hidden layer, and an output layer. The input layer consists of four neurons, and the output layer has two. Each output represents action and gives the Q value for that action. The neural network predicts the best action that has the highest Q value among other possible actions, i.e., $a = argmax_a Q(s, a)$ for the state provided as input to the network. Every neuron passes its input through an activation function that decides whether the neuron should be fired or not and sends its weighted summation to the neurons of the next layer of the network.

There are two neural networks in DQL: the main network, also known as the Q network, and the target network, which shares the same architecture but carries different weights. In DQL, the past experiences are stored in a memory buffer known as the Experience replay buffer in the format of <state, next state, action, reward> to make the network update more stable. Q network uses a mini-batch of transitions sampled from experience replay during training the network for computing the loss and gradient. After a certain number of steps, the weights from the Q network are copied to the target network. The Q network approximates the Q values for the state-action pair. Deep neural network is trained with parameter θ which estimates the Q values, i.e., $Q(s, a; \theta) \approx Q(s, a)$. The weights are updated by minimizing loss function $L_t(\theta_t)$ at each step t, which is calculated by taking the means square error of TD (temporal difference) error as in Eq. (17), over a batch of transitions sampled from replay buffer. y_t is the TD target.

$$y_t = \mathcal{R} + \gamma \max_{a}^{\prime} Q\big(s', a'; \theta_{t-1}\big) \tag{16}$$

$$L_t(\theta_t) = \mathop{\mathbb{E}}_{s,s',a,\mathcal{R}}\Big[(y_t - Q(s, a; \theta_t))^2\Big] \tag{17}$$

The DQL approach is explained in Algorithm 4. DQL agent follows the same state space $\mathcal{S}$ as input, action space $\mathcal{A}$ and reward function $\mathcal{R}(s, a)$ as in Eqs. (12), (13), and (14) respectively. First, the weights of neural networks are initialized randomly. For every event e of the arrival of a new request, the agent takes the input and predicts the best action using the epsilon greedy exploration approach. The action a is chosen randomly with a probability of ϵ and a greedy action with a probability of 1-ϵ. In the case of a neural network, the Q values for each action are predicted. The action with the highest predicted Q value is the best at that state. The observations are stored in A replay memory. According to the Bellman Equation, the agent trains the neural network and updates its weights. Training first involves the calculation of TD target y_t, which is calculated using the target network, and weights are updated for the Q network. After every N steps, the weights of the Q network are copied into the target network.

To summarize, the reason to apply Deep Q-networks (DQN) is that the state transition probabilities are not known. If they are known, one can apply dynamic algorithms and solve using value iteration or policy iterations method in RL. The dynamic approach will give us the best solution. However, in practical scenarios, the slice arrival probabilities are not known making the problem hard to solve. This is addressed using by Q-Learning. The drawback of Q-Learning is that it takes a lot of memory. In order to handle the space constraints, we apply deep queuing networks that helps reduce the state space. The dynamic algorithm approach, namely Policy Iteration, is used as a baseline to determine the performance of applying Q Learning and DQN.

Algorithm 4: Deep Q Learning

```
1  Initialize weights of neural networks randomly.
2  Initialize $n, \epsilon, \gamma, \theta, t \leftarrow 0$
3  $s \leftarrow (AV_c, AV_m, AV_b, l_u)$
4  if $(N_{r,u} > AV_r \forall\ r)$ then
5      $a \leftarrow 0$                                  // action
6  else
7      Provide Input($s$) to DQL Agent
8      if uniform.random() $\leq \epsilon$ then
9          $a \leftarrow$ Choice(0,1)                    // Explore
10     else
11         actionsQval=PredictQScore($s$)
12         $a \leftarrow argmax_a$(actionsQval)          // Exploit
13     $s', \mathcal{R}(s, a) \leftarrow$ next state, Reward
14     ReplayBuffer.append([$s, a, s', \mathcal{R}(s, a)$])
15     S=ReplayBuffer.sample()
16     for every entity in S do
17         Target($y_t$) = $\mathcal{R}(s, a) + \gamma \max'_a Q(s', a'; \theta_{t-1})$
18     Train QNetwork using SGD.
19     if $t$ % $n$ == 0 then
                                         /* Copy weights */
20         TargetNetwork.W() $\leftarrow$ QNetwork.W()
21 $t \leftarrow t + 1$
22 Repeat Steps 3 to 21 for every slice request $u$
```

V. PERFORMANCE EVALUATION

This section presents the implementation details and performance studies of the approaches presented in the previous section. The experiments were conducted on a computing system with an Intel i7-8700 3.20 GHz CPU, 32 GB RAM, and 6 CPU Cores. The results are generated by averaging the results obtained by running the overall experiments ten times.

A. IMPLEMENTATION DETAILS

1) DATASET

The *GWA-T-13 Materna-trace* dataset [15] has been used for creating 5G network slice data. The original dataset consists of Materna-trace-1, Materna-trace-2, and Materna-trace-3. Materna-trace-1 consists of performance metrics of 520 Virtual Machines (VMs), Materna-trace-2 consists of 527 VMs, and the third trace consists of 547 VMs. Each trace represents one month of data. In each file of its VM, data is recorded every 5 minutes. Three metrics are extracted, i.e., CPU usage, which is in terms of MHz, Memory usage, which is actively used in terms of KB, and network transmitted throughput in Kbps. The three metrics are converted to the required format for creating a slice for C, M, B. The CPU usage, which is in MHz, is divided by the speed per core and obtained CPU in number of Cores. Memory usage is converted into GBs, and Network throughput is used as Bandwidth in Mbps. To create a resource usage slice in a time series manner, we have summed up these three metrics of 10 continuous VMs in every trace. The number of CPU cores and network bandwidth information is scaled by 10 in each slice, so that all resource requirements are on the same scale, while maintaining the same distribution.

TABLE 5. System parameter values.

Parameters	C(Cores)	M(GB)	B(Mbps)	N_c(Cores)	N_m(GB)	N_b(Mbps)	λ_I (/min)	λ_E (/min)	μ_I (hr)	μ_E (hr)	ρ_I	ρ_E
value	15000	15000	15000	50	50	50	5	16	9	9	10	2

2) LSTM

The slice data is first preprocessed before being trained for forecasting. In preprocessing, the outliers are removed using the Interquartile range (IQR) approach. The slice forecasting agent based on LSTM is implemented using Tensorflow [41] and Keras [42] packages. We employ an online training window for LSTM, which is trained for the past 10 hours of data. The observation window for history data is kept to be 80 minutes, and resource demand is predicted for 40 minutes. Without loss of generality, the data is normalized in range of [0,1] using the MinMaxScalar [43] technique from the *sklearn* Python library [44]. We arrived at different hyperparameter values of the LSTM neural network by hyper tuning. LSTMs are trained for 300 epochs in batches of 64 with eight neurons in a hidden layer using *Adam* [45] optimizer at an initial learning rate of 0.1 with an exponential decay rate of 0.96. The neural network is trained on Materna slice data for time-series prediction using the *ReLU* [46] activation function.

3) DQL AGENT

The entire framework is implemented in Python 3.6.9. The Q learning agent and DQL agent are trained at a learning rate (α) and discount factor (γ) using Tensorflow [47]. In DQL, the Experience Replay Buffer size can store 50,000 observations. A mini-batch of 128 observations is sampled from Replay Buffer for training the neural network. Both the Q and target network consist of two dense hidden layers with size 32 and 16 neurons and use ReLu activation function each. The Q Network is trained after every four steps using Bellman and a model with a batch size of 32, 64, 128 is tried where we chose a batch size of 128. The Q Weights are copied into the target network after every 100 steps. The DQL agent calculates the loss using *Mean-Squared-Error* [36]. The weights are updated through back-propagation by using various optimizers where the Stochastic Gradient Descent [48] gives best results.

4) SLICE SIMULATION

The slice arrivals and their departures are generated using the SimPy discrete-event simulator package [49]. Slice arrivals are modeled using a Poisson process with an arrival rate of λ slices per minute. The arrival rate and mean exponential distribution μ for the departure of slice classes vary such that $\lambda_E \geq \lambda_I$ and $\mu_I = \mu_E$ respectively. A rejection penalty of $\mathcal{PY}_{rla} = 1000$ is set for the RL agent when it rejects an I slice. The bid value of the I slice class is assumed to be ten times higher than E, i.e., $\rho_I = 5\rho_E$ per hr. The bid value for using each resource is kept the same. The value of β is set to 0.5 in opportunistic overbooking, since it gives significant importance to the future states.

The InP capacity and slice parameters are listed in Table 5. In simulation results, it is assumed that each slice requests an equal and fixed amount of resource r from InP denoted as N_r. The simulation is run for a continuous time t. Each unit t represents five minutes, and a network window NW size is considered 40 minutes. A Poisson process determines the number of slices arriving at each time t, where t represents 5 minutes. Based on arrival rates, a total of 5352 slices are generated for $t = 2200$x5 minutes in which 1313 belong to I and 4039 to E class for λ_I=5 and λ_E=16 respectively. The decision to accept or reject a slice is taken every unit time that AD decides.

Whenever a slice gets accepted, it is added to the active slices queue and generates revenue each time until its lifetime. The revenue from each slice is proportional to bid value and requested resources. This paper considers SLAV as occurring when the InP does not provide enough resources to a slice. In case any slice experiences resource deficiency, dependent on its class's ϕ value, the InP is penalized to 3 times its bid value for insufficient resources. The waiting queue for slices is considered to be infinite. The AD runs when requests arrive, whereas slice forecasting and overbooking decisions are made once at the start of every network window to reallocate resources to all active slices.

5) ALGORITHMS USED FOR COMPARISON

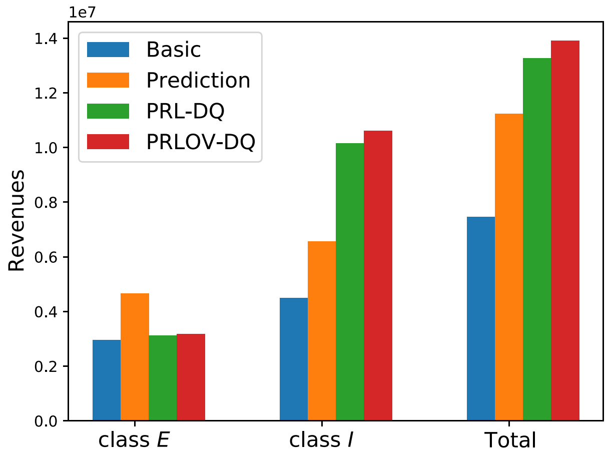
The performance of the proposed framework (PRLOV) is compared with three other heuristic algorithms. These are:

1) Basic Admission:
It simply admits slice requests if requested resources are available at InP in FCFS order. It does not give any preference in accepting higher priority slices.
2) Prediction Admission: It includes the Slice Forecasting (SRF) that predicts the resources demand of currently active E slices at each NW and then accepts the incoming slices based on remaining resources in FCFS order. It also does not account for slice priorities.
3) PRL: It is similar to the proposed algorithm PRLOV except considering the slice opportunistic overbooking.

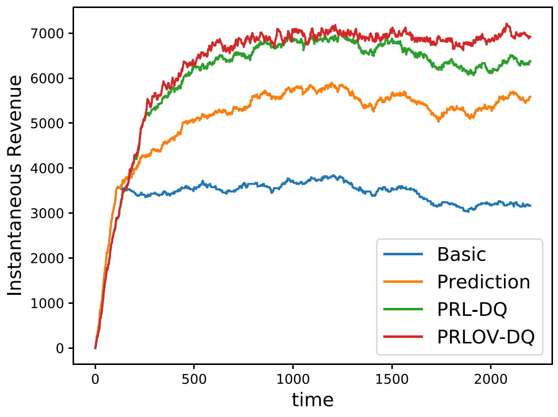
B. PERFORMANCE ANALYSIS

1) INP NET REVENUE

Fig. 5(a) shows the performance of the proposed framework PRLOV in terms of revenue, for 15,000 units per resource, after $t = 2200$ units. The results for using Deep RL in AD in the case of PRL and PRLOV are presented. The instantaneous revenues averaged over the last 1200 times for Basic, Prediction, PRL-DQ, and PRLOV-DQ are 3428, 5520, 6556, **6942** respectively, as shown in Fig. 5(b). The results



(a) Overall revenue at $t = 2200$



(b) Instantaneous Revenue Values

FIGURE 5. Comparison of overall InP revenue.

TABLE 6. Results analysis at InP resource size of 15,000.

	Algorithms			
Parameters	**Basic**	**Prediction**	**PRL-DQ**	**PRLOV-DQ**
Admitted I Slices	349	538	785	815
Admitted E Slices	1182	1853	1246	1284
Total Admitted Slices	1531	2391	2031	2099

show the effectiveness of using prediction and overbooking concepts, that results in higher revenue with PRLOV.

The revenues obtained from two classes that constitute the overall revenue are also analyzed. Table 6 shows the number of overall accepted slices over 5352 slices by InP for each slice class. It shows the individual contribution of revenue at InP by different slice classes I and E along with the total revenue. The overall revenue obtained for the proposed scheme PRLOV-DQ is the highest, whereas it is the least for Basic scheme. The revenue obtained for the E slices class is maximum for Prediction, since it accepts the maximum number of Elastic slices, whereas, for I, it is maximum for PRLOV-DQ.

2) RESOURCE UTILIZATION

Fig. 6 and Fig. 7 present the percentage of resources (CPU, Memory, Bandwidth) utilized for the four different mentioned algorithms by slice class I and E, respectively. The maximum resources utilization for Basic Admission is lowest, i.e., (47%, 37%, 47%) for each resource type. Considering only Prediction, PRL and PRLOV, the maximal resources utilization is similar; however, PRLOV achieves high resource utilization along with higher revenue. The revenue gains for Prediction and PRL are lower than that of PRLOV, indicating inefficient resource utilization.

Fig. 6 shows that PRLOV achieves the highest usage of resources for slice class I for all type of resources. PRL shows higher than Prediction for I but less than the proposed algorithm due to overbooking, which admits more slices. The plot in Fig. 7 gives the resources used by E slices. The resources utilization curves for each resource for E do not show a significant difference; still, PRLOV shows the highest utilization. The reason is that PRL and PRLOV try to reject E if there is more possibility of incoming an I slice.

Since PRLOV uses overbooking, it accepts more slices, whereas PRL only focused on priority, resulting in lower resource utilization. The Prediction scheme shows increased resource utilization for E slices because it only accepts slices based on FCFS order without considering slice priorities. The maximum utilization is not 100% since slices that belong to I are assigned the full requested resource N_r, but may not use all the resources. To be totally free from any SLA violation for I slices, we do not perform any prediction and overbooking, and the InP provides them with all the requested resources. The results show that the proposed PRLOV scheme shows a balanced way of obtaining higher revenue while maintaining better resource utilization.

TABLE 7. Accepted slices for different arrival rates.

λ_I (/min)	Total I	Total E	Admitted I	Admitted E
5	1313	4039	815	1284
8	2046	4039	965	993
10	2532	4039	1049	880
12	3028	4039	1063	786
16	4039	4039	1157	629

3) VARYING ARRIVAL RATE

Fig. 8(b) shows the total revenues obtained after $t = 2200$ at a fixed rate of E, $\lambda_E = 16$ and varying the arrival rate, $\lambda_I \in \{5, 8, 10, 12, 16\}$. As the rate of arrival for Inelastic slices increases, the acceptance rate of Inelastic slices increases whereas rate of acceptance for Elastic slices decreases. This is due to the priority given to Inelastic slices. The revenues obtained from each slice class show similar behaviour. The overall revenue is also increased with the increase in λ_I, as expected. Table 7 presents the total number of simulated and accepted slices of I and E by varying the arrival rates.

4) CONVERGENCE TIME

The time taken to converge the QL and DQL is expressed in the number of episodes. The maximum time is taken by policy iteration, a model-based approach used as a benchmark to evaluate the results obtained from PRLOV-Q

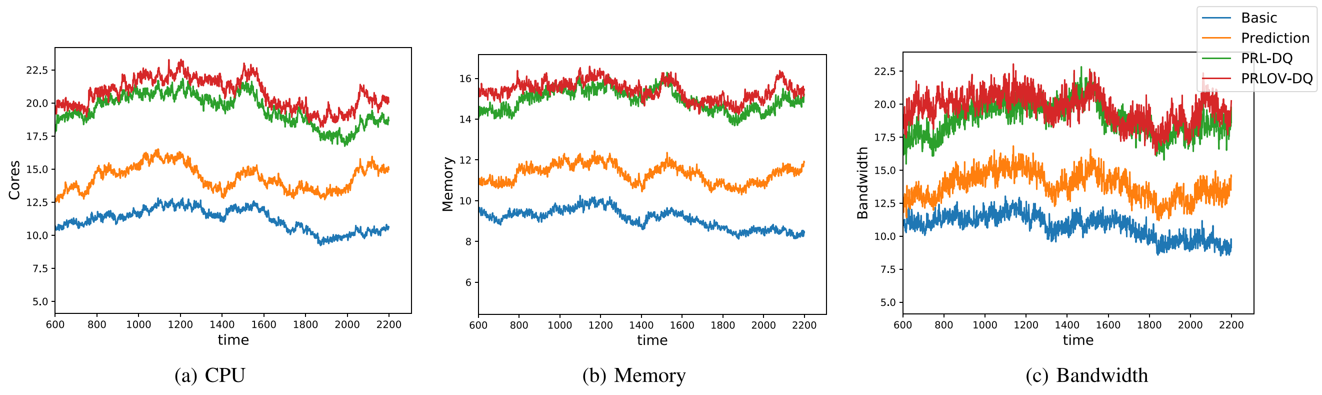


FIGURE 6. Instantaneous resource usage in percentage of inelastic slices (I), for time $t = 2200$.

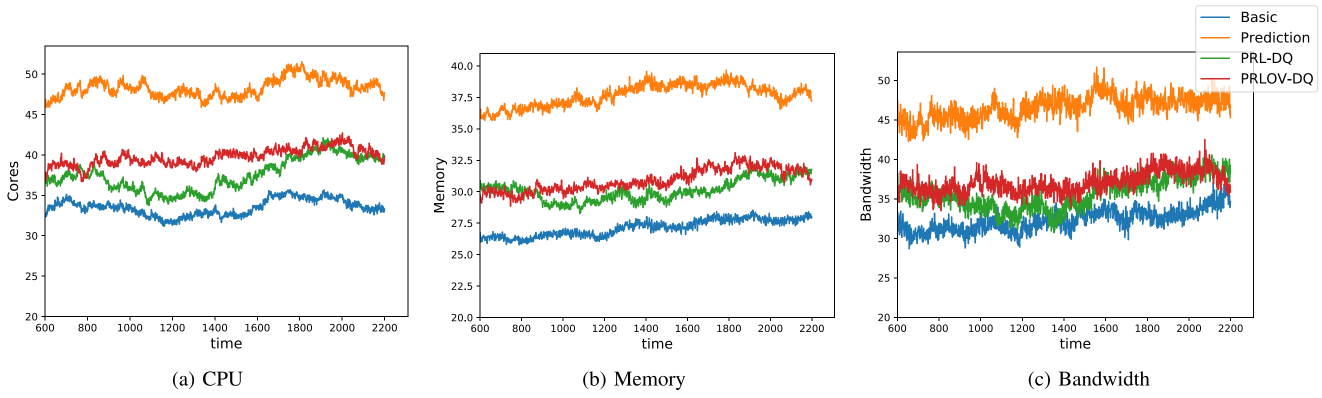


FIGURE 7. Instantaneous resource usage in percentage of elastic (E) slices, for time $t = 2200$.

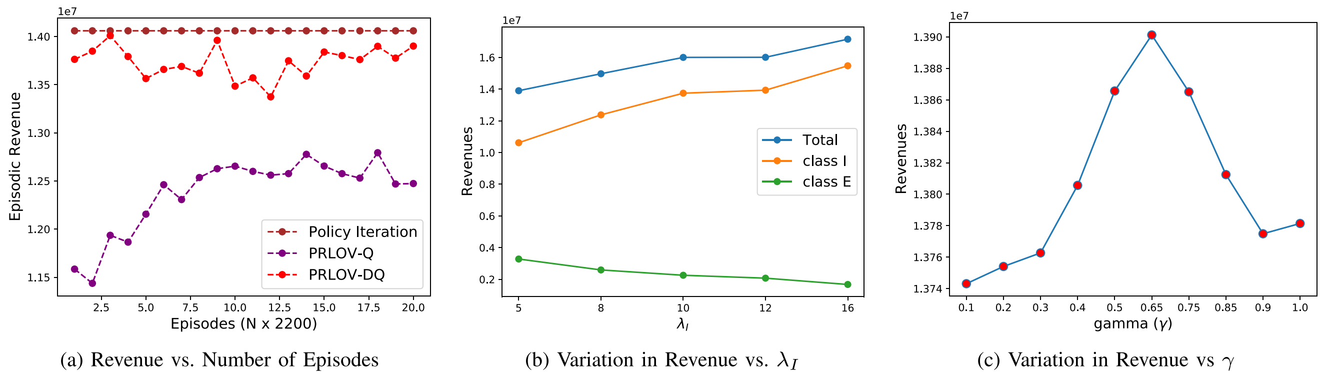
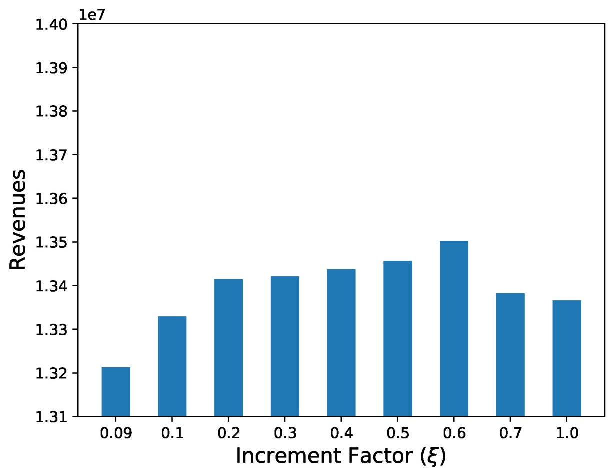


FIGURE 8. Convergence study and effect of varying λ_I and γ.

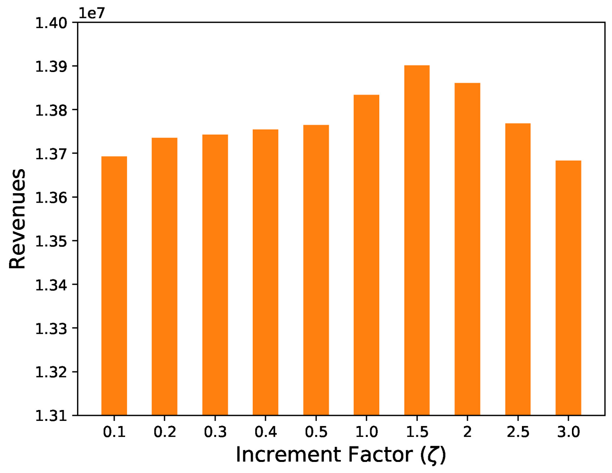
and PRLOV-DQ. Fig. 8(a) shows the number of episodes required for convergence of the Q-Learning and Deep Q Learning. DQL converges faster within 300k-450K episodes, whereas, Q-Learning requires more episodes to converge. The convergence time of QL and DQL also depends upon the learning rate α. The results of varying the value of γ are shown in Fig. 8(c). From running multiple experiments, the learning rate (α) of 0.09 and a discount factor (γ) of 0.65 is chosen, respectively.

5) COMPARISON WITH OUR EARLIER SCHEME [12]

There may be cases when resources are over-predicted and under-predicted. In case of over-prediction, the Υ is increased by the ζ factor and converged to a new overbooking value. In contrast, when actual usage for any slice is higher than predicted, the proposed mechanism reduces the overall overbooking and admits fewer slices. The revenues obtained from varying the regulating factor ζ that use the RL feedback mechanism to regulate overbooking are compared against the existing overbooking scheme in [12] by varying ξ value. The results obtained for different values incrementing the overbooking regulating factors are shown in Fig. 9(a) and 9(b). We see an enhanced overbooking solution using RL is far better than the existing scheme. The maximum revenue obtained by varying the ξ values is always lower than the revenue obtained by the proposed RL feedback scheme.

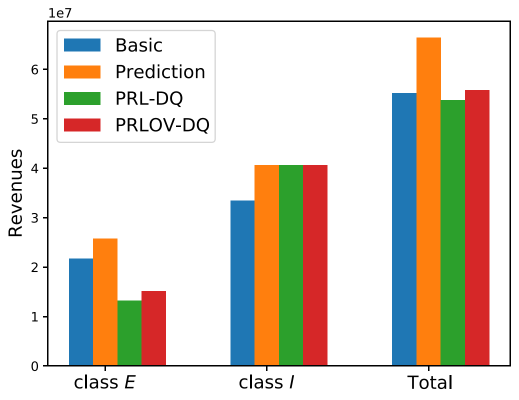


(a) ξ

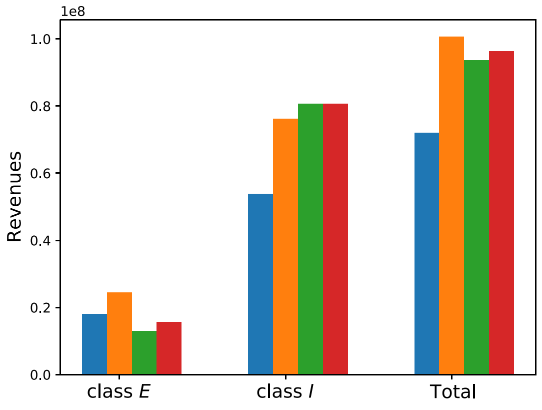


(b) ζ

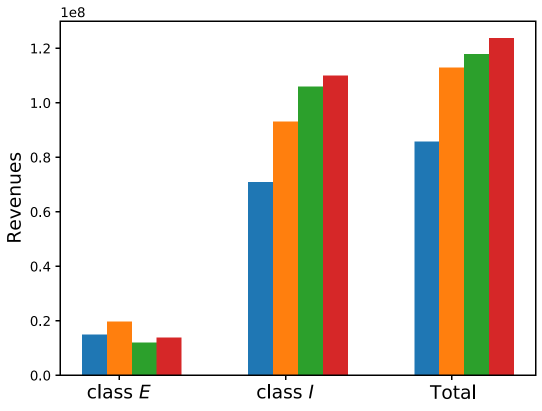
FIGURE 9. **Impact of varying regulating Factors ξ and ζ.**



(a) $\lambda_I = 5$



(b) $\lambda_I = 10$



(c) $\lambda_I = 15$

FIGURE 10. **Overall Revenues at InP Resource size of 50,000 at $t = 5000$.**

6) COMPARING RL ALGORITHMS

Simulation experiments were conducted for InP resource size of 50,000 for $t = 5000$, at different arrival rates of $\lambda_I \in \{5, 10, 15\}$ and $\lambda_E = 16$ per minute and $\mu = 9$ hours. The number of slice requests generated were 11751, 14555, and 17283 slices respectively for the three values of λ_I. Of this, 2813, 5617, and 8345 respectively belonged to the I class. Since different slice types arrived at different rates, the revenue values are different for the four algorithms.

Fig. 10(a) presents the revenues obtained when $\lambda_I = 5$ and 50,000 units of each InP resource are available. The maximum revenue is received using Prediction, since the InP has enough resources to accommodate all slices and there is no need to make decisions about accepting or rejecting slices. Due to the low arrival rate, there is no need to apply an RL model with rigid rejection rules to create space for I slices. The RL algorithms, i.e., PRL-DQ and PRLOV-DQ, accept all Inelastic slices for $\lambda_I = 5$ and reject Elastic ones, which is unnecessary when enough resources are left unallocated.

A similar behavior is also shown at the arrival rate $\lambda_I = 10$ in Fig. 10(b), where PRL-DQ and PRLOV-DQ reject some Elastic slices in order to accept most of the Inelastic slices. When there are enough slices such that InP cannot accommodate all the slice requests. In such cases, we see an improvement using the RL models. As seen in Fig. 10(c), the proposed model PRLOV yields the highest revenue at InP size of 50,000 and λ_I=15, which is similar to the InP size of 15,000 for λ_I=5 in Fig. 5(a).

The number of I and E slices accepted by different algorithms at different values of λ_I are given in Table 8. As seen,

TABLE 8. Results analysis at InP resources size of 50,000.

	Algorithms			
Parameters	**Basic**	**Prediction**	**PRL-DQ**	**PRLOV-DQ**
λ_I= 5 per min.				
Admitted I Slices	2329	2813	2813	2813
Admitted E Slices	7636	8938	4570	5254
λ_I= 10 per min.				
Admitted I Slices	3783	5316	5617	5617
Admitted E Slices	6295	8550	4531	5400
λ_I= 15 per min.				
Admitted I Slices	4929	6490	7349	7607
Admitted E Slices	5240	6929	4167	4872

PRLOV accepts more inelastic slices and less elastic slices, which results in higher revenue generation.

Experiments were also conducted for InP resource size of 2,000 at λ_I for $t = 2200$. We generated 1313 I slices, and 4039 E slices. The framework accepts 45, 68, 90, and 121 slices belonging to I and 149, 244, 198, and 205 slices from E class for Basic, Prediction, PRL, and PRLOV respectively. We obtain a total revenue of 998,484; 1,518,000; 1,777,494, 2,106,787 at $t = 2200$ for the four algorithms respectively.

VI. CONCLUSION

This paper presented a Deep Reinforcement Learning approach for 5G network slice Admission in a service provider's network infrastructure. Two types of slices, Elastic and Inelastic, have been considered, where the latter offers higher revenue but has strict resource requirements. The framework is driven by prediction using LSTM and overbooking of active Elastic slices. The RL mechanism mainly focuses on accepting higher priority slices to increase the service providers' revenue. Since Elastic slices' service requirements are relaxed, the notion of overbooking has been applied where slices are admitted even if the total requirements exceed the InP's available capacity. A detailed performance study was done using discrete event simulation coupled with implementation of the different schemes in TensorFlow and Keras. In comparison with other schemes, the proposed PRLOV scheme is shown to perform much better in terms of revenue generation and resource utilization.

Further improvements can be made by analyzing slice priorities using multiple agents, slice duration, the number of resources requested, and their impact on InP's revenue and resources utilized.

ACKNOWLEDGMENT

The authors are thankful to Dr. Sujata Banerjee of VMware Research USA and her team for their valuable comments and suggestions.

REFERENCES

[1] A. Gupta and R. K. Jha, "A survey of 5G network: Architecture and emerging technologies," *IEEE Access*, vol. 3, pp. 1206–1232, 2015.

[2] S. Zhang, "An overview of network slicing for 5G," *IEEE Wireless Commun.*, vol. 26, no. 3, pp. 111–117, Jun. 2019.

[3] "Minimum requirements related to technical performance for IMT-2020 radio interface(s)," Int. Telecommun. Union Radio Commun. Sector (ITU-R), Geneva, Switzerland, ITU-R Rep. M.2410-0, Nov. 2017.

[4] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi, "5G wireless network slicing for eMBB, URLLC, and mMTC: A communication-theoretic view," *IEEE Access*, vol. 6, pp. 55765–55779, 2018.

[5] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[6] C. Bouras, A. Kollia, and A. Papazois, "SDN & NFV in 5G: Advancements and challenges," in *Proc. Conf. Innov. Clouds Internet Netw. (ICIN)*, 2017, pp. 107–111.

[7] "Network functions virtualisation (NFV) release 3; evolution and ecosystem; report on network slicing support with ETSI NFV architecture framework." 2017. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV-EVE/001_099/012/03.01.01_60/gr_nfv-eve012v030101p.pdf

[8] I.-H. Hou, V. Borkar, and P. R. Kumar, "A theory of QoS for wireless," in *Proc. IEEE INFOCOM*, 2009, pp. 486–494.

[9] J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, and X. Costa-Perez, "Overbooking network slices through yield-driven end-to-end orchestration," in *Proc. ACM CoNEXT*, 2018, pp. 353–365.

[10] C. Sexton, N. Marchetti, and L. A. DaSilva, "On provisioning slices and overbooking resources in service tailored networks of the future," *IEEE/ACM Trans. Netw.*, vol. 28, no. 5, pp. 2106–2119, Oct. 2020.

[11] "Long short-term memory layer." Oct. 2021. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM

[12] S. Saxena and K. M. Sivalingam, "Slice admission control using overbooking for enhancing provider revenue in 5G networks," in *Proc. IEEE/IFIP Netw. Oper. Manag. Symp.*, 2022, pp. 1–7.

[13] V. Mnih et al., "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[14] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. 7, pp. 133653–133667, 2019.

[15] "GWA-T-13-materna-trace." Sep. 2013. [Online]. Available: http://gwa.ewi.tudelft.nl/datasets/gwa-t-13-materna

[16] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[17] *5G; System Architecture for the 5G System*, 3GPP Standard TS 23.501 version 15.3.0 release 15, 2018. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/15.03.00_60/ts_123501v150300p.pdf

[18] M. Chahbar, G. Diaz, A. Dandoush, C. Cérin, and K. Ghoumid, "A comprehensive survey on the E2E 5G network slicing model," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 1, pp. 49–62, Mar. 2021.

[19] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez, "Optimising 5G infrastructure markets: The business of network slicing," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[20] B. Han et al., "Admission and congestion control for 5G network slicing," in *Proc. IEEE Conf. Stand. Commun. Netw. (CSCN)*, 2018, pp. 1–6.

[21] V. Sciancalepore, X. Costa-Perez, and A. Banchs, "RL-NSB: Reinforcement learning-based 5G network slice broker," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1543–1557, Aug. 2019.

[22] M. R. Raza, C. Natalino, P. Öhlen, L. Wosinska, and P. Monti, "Reinforcement learning for slicing in a 5G flexible RAN," *J. Lightw. Technol.*, vol. 37, no. 20, pp. 5161–5169, Oct. 15, 2019.

[23] M. R. Raza, A. Rostami, L. Wosinska, and P. Monti, "A slice admission policy based on big data analytics for multi-tenant 5G networks," *J. Lightw. Technol.*, vol. 37, no. 7, pp. 1690–1697, Apr. 1, 2019.

[24] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, and X. Costa-Pérez, "A machine learning approach to 5G infrastructure market optimization," *IEEE Trans. Mobile Comput.*, vol. 19, no. 3, pp. 498–512, Mar. 2020.

[25] W. Jiang, Y. Zhan, G. Zeng, and J. Lu, "Probabilistic-forecasting-based admission control for network slicing in software-defined networks," *IEEE Internet Things J.*, vol. 9, no. 15, pp. 14030–14047, Aug. 2022.

[26] M. Sulaiman, A. Moayyedi, M. A. Salahuddin, R. Boutaba, and A. Saleh, "Multi-agent deep reinforcement learning for slicing and admission control in 5G C-RAN," in *Proc. IEEE/IFIP Netw. Oper. Manag. Symp. (NOMS)*, 2022, pp. 1–9.

[27] M. A. Haque and V. Kirova, "5G network slice admission control using optimization and reinforcement learning," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2022, pp. 854–859.

[28] Z. Wu, G. Ishigaki, R. Gour, and J. P. Jue, "A reinforcement learning-based admission control strategy for elastic network slices," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2021, pp. 1–6.

[29] W. F. Villota-Jacome, O. M. C. Rendon, and N. L. S. da Fonseca, "Admission control for 5G core network slicing based on deep reinforcement learning," *IEEE Syst. J.*, vol. 16, no. 3, pp. 4686–4697, Sep. 2022.

[30] K. M. Sivalingam, "Applications of artificial intelligence, machine learning and related techniques for computer networking systems," Apr. 2021, *arXiv:2105.15103*.

[31] "AI and ML—-Enablers for beyond 5G networks version 1.0." May 2021. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2021/05/AI-MLforNetworks-v1-0.pdf

[32] A. B. Koehler, R. D. Snyder, and J. Ord, "Forecasting models and prediction intervals for the multiplicative Holt–Winters method," *Int. J. Forecast.*, vol. 17, no. 2, pp. 269–286, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207001000814

[33] S. D. Immanuel and U. K. Chakraborty, "Genetic algorithm: An approach on optimization," in *Proc. Int. Conf. Commun. Electron. Syst. (ICCES)*, 2019, pp. 701–708.

[34] M. I. Jordan, "Chapter 25—Serial order: A parallel distributed processing approach," in *Neural-Network Models of Cognition* (Advances in Psychology), vol. 121, J. W. Donahoe and V. P. Dorsel, Eds. Amsterdam, The Netherlands: North-Holland, 1997, pp. 471–495. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166411597801112

[35] Y. Chauvin and D. E. Rumelha, *Backpropagation: Theory, Architectures, and Applications*, Taylor & Francis Group, 2013.

[36] "MeanSquaredError." Accessed: Nov. 21, 2022. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/losses/MeanSquaredError

[37] M. A. Alsheikh, D. T. Hoang, D. Niyato, H.-P. Tan, and S. Lin, "Markov decision processes with applications in wireless sensor networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1239–1267, 3rd Quart., 2015.

[38] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107–1149, Dec. 2003.

[39] B. Bakhshi, J. Mangues-Bafalluy, and J. Baranda, "R-learning-based admission control for service federation in multi-domain 5G networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2021, pp. 1–6.

[40] C. Sun, "Fundamental Q-learning algorithm in finding optimal policy," in *Proc. Int. Conf. Smart Grid Elect. Autom. (ICSGEA)*, 2017, pp. 243–246.

[41] "TensorFlow." Google Brain. Oct. 2021. [Online]. Available: https://www.tensorflow.org

[42] "Keras: The python deep learning API." Keras. Oct. 2021. [Online]. Available: https://keras.io

[43] "MinMaxScalar." Accessed: Nov. 21, 2022. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[44] "Scikit-learn 1.1.1." Accessed: Nov. 21, 2022. [Online]. Available: https://scikit-learn.org/stable/auto_examples/release_highlights/plot_release_highlights_1_1_0.html

[45] "Adam." Accessed: Nov. 21, 2022. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam

[46] "ReLU." 2022. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/ReLU

[47] "Train a deep Q network with TF-agents." 2022. [Online]. Available: https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial

[48] "Stochastic gradient descent (SGD)." Accessed: Nov. 21, 2022. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD

[49] "SimPy 4.0.1: Event discrete, process based simulation for python." SimPy Team. Apr. 2020. [Online]. Available: https://pypi.org/project/simpy/