CS798-001,002 Project Report: Deep Reinforcement Learning Based 5G Slicing with Admission Control

Muhammad Sulaiman m4sulaim@uwaterloo.ca
Arash Moayyedi arash.moayyedi@uwaterloo.ca

I. Introduction

With the recent boom in the Internet traffic volume, telecommunication operators are having a difficult time keeping up with the demand. Their systems serve a variety of users and communications, all with their own set of service expectations and requirements. As a result, conventional one size fits all network infrastructures are slowly vanishing and are giving their place to modern programmable networks. This programmability allows for improved resource utilization through leveraging network trends, which consequently, reduces operation costs and provides an opportunity to host more users. 5G slicing is one of these emerging technologies that isolates a set of users with similar requirements from the rest of the network, enabling the network operator to adapt the underlying infrastructure to cater for those users' needs.

In the 5G paradigm, a network Infrastructure Provider (InP) is able to serve multiple Mobile Virtual Network Operators (MVNOs), which may also subsume the role of Service Providers (SPs). Based on the type of service that needs provisioning, an MVNO requests a network slice from the InP. This slice request constitutes a particular Service Function Chain (SFC) and an associated service level agreement (SLA), such as latency and bandwidth thresholds. Based on the slice request parameters and the current state of the network, it is the job of the InP to optimally allocate resources in the physical infrastructure, such that it leads to maximization of the infrastructure utilization and revenue gains. To this end, the InP has to optimize multiple features of 5G slice management and orchestration (MANO), which include slice admission control, Virtual Network Embedding (VNE), resource allocation, slice monitoring, and resource rescaling.

Since the InP only has a limited amount of resources available, it inevitably has to decline some slice requests and also place the accepted requests optimally in the underlying network. These decisions are complicated, since the InP must consider several aspects such as the slice template, the current state of the network, the prospective revenue, and the slice-request traffic characteristics, among others. A mathematical formulation and optimization of this problem that includes all these aspects is, therefore, too complex. AI based approaches, specifically Reinforcement learning (RL), allow us to manage this complexity by having the a machine learning model learn and improve using historical data instead of having to design a complex mathematical model. In this project, we will be converting the 5G network slicing and admission control problem to an RL problem by formulating a cost function, and teaching an RL agent, by experience, to minimize this cost function. Although, some RL-based solutions for this control problem have been proposed so far, their perspectives of the underlying networks is somewhat too simplistic to fully leverage the available information at hand and to be able to be used in a real-life scenario. For instance, some of the approaches that have been proposed do not consider some influential factors in their solutions, such as the network link bandwidths and delays, the Quality of Service (QoS) requirements associated with individual slices, and their placement in case they are accepted. We have developed a 5G slicing and admission control unit that not only takes the slice-request traffic characteristics into account, but also leverages an extensive observation of the network topology. By combining the admission control agent with the intelligence obtained from a intelligent network slicing agent, we can make a more comprehensive reinforcement learning algorithm that learns to maximize the InP's revenue. In this report, we will be analyzing some of the recent challenges in 5G slice managements and orchestration (MANO) and their possible AI-based solutions with a focus towards 5G slicing and admission control. In the end, we present and analyze our novel RL-based 5G slicing and admission control solution.

The rest of the paper is structured as follows: Section 2 and 3 give a short overview of Reinforcement Learning and 5G management and orchestration (MANO) respectively. Section 4 introduces the problem of network slice resource reconfiguration and analyzes the RL approach proposed by [7]. Section 5 introduces the problem of VNF placement and resource allocation and analyzes the RL-approaches proposed by [17, 12]. Section 6 introduces the problem of slice admission control and analyzes the RL-based solutions proposed by [2, 5, 16, 10, 4, 14, 11]. Section 7 empirically evaluates the RL solutions to slice admission control. Section 8 introduces our novel approach the 5G network slicing and admission control, section 9 presents the results, discussion and possible future improvements. Finally, Section 10 gives conclusion.

II. Reinforcement Learning

Reinforcement Learning (RL) sits alongside supervised learning and unsupervised learning, and it encourages learning by providing an intelligent agent with a simulated environment to interact with. Each interaction influences the environment and the propriety of this action is reflected as a reward value to the agent. RL has re-emerged in recent years as Deep Reinforcement Learning (DRL) to provide unprecedented performance[9] in tasks previously too challenging for artificial intelligence-based solutions, by leveraging the power of Deep Neural Networks (DNNs). Artificial neural networks are a series of interconnected computation nodes that are loosely based on the neurons of a biological brain. Each of these nodes take input from a multitude of other nodes, perform their computations, and pass their results to the rest of the network, thus, constituting to the final output of the network.

In RL, there are two categories of possible actions, called exploration and exploitation. Exploration refers to wandering away and exploring the untouched parts of the environment in hopes of finding a new optimal path through the environment. This is crucial for the agent, since without experimenting with different actions and studying their effects, the agent will not be able to find the optimal solution. However, the agent cannot keep experimenting, as the massive scale and the complexity of the environment and all its possible set of actions and states would not allow the agent to do so. Therefore, the agent must use its previous findings to guide it towards the optimal path. This is called exploitation. A balance is required between these two categories to get closer to the optimal path in a timely manner. One suggested policy that takes a balanced approach is the "ϵ-greedy" policy, where the agent takes a random exploratory action - out of the action space - ϵ percentage of the time and chooses an action that it thinks is the optimal action $1 - \epsilon$ percentage of the time.

Reinforcement learning algorithms are divided into multiple categories based on their characteristics. The first category would be on-policy and off-policy methods. Off-policy algorithms update their learning tables based on the optimal action according to their current understanding of the environment, while the updates in on-policy algorithms are based on the optimal action according to the currently applied policy, which may not be the optimal action. For example, while applying the ϵ-greedy policy, when the agent takes a random action that leads to a new state, if it decides to learn based on the expected value of taking a random action in the new state, it would be acting on-policy. On the contrary, if it decides to learn based on the expected value of taking the optimal action, it would be considered off-policy.

Another attribute of RL algorithms is whether they are policy-based algorithms or value-based algorithms. Policy-based algorithms, as their name suggests, focus on optimizing the policy by mapping directly from a state to the action probability. They increase the probability of selecting higher paying actions in terms of the returned reward for each state, while decreasing the probability of selecting the other actions. This is in contrast with value-based methods, where the best action is selected by comparing the values of each possible action in that state, which had been previously calculated by considering the prospective rewards of that action. Actor-critic algorithms are a mix of these two.

The final feature of RL algorithms is if they are model-based algorithms or model-free ones. If the environment is deterministic and the agent has access to the environment model and can fully predict the consequences of its actions, the approach would be called model-based. However, not all environments can be predicted, as they might be too massive to model, too complex to model, or they might include noise and randomness. Thus, in contrast, there are model-free approaches, where the agent is not completely sure of the outcome of the selected action, and it tries to model the environment by experimenting and collecting results.

III. 5G Slice Management and Orchestration (MANO)

Software-defined networking (SDN) and network function virtualization (NFV) are among the key enablers of efficient 5G networks. Currently, the prices for compute, and storage resources are so low that it has become feasible to emulate the network functions, which are usually performed by specialized hardware equipment. These emulated network functions are called virtual network functions (VNFs), and this idea of emulating these network functions is referred to as NFV. SDN, on the other hand, is the use of logically centralized controllers to effectively manage these VNFs and create service function chains (SFC) or slices. One of the core tenants of 5G networks is their ability to efficiently serve customers with diverse requirements. For instance, a multiplayer gaming setup might not need high throughput, but may require very low latency, while a video streaming service might need higher throughput with flexible latency requirements. In legacy networks, the same underlying network infrastructure is used to serve all these use cases, which leads to both an inefficient use of the infrastructure's available resources and a diminished quality of experience (QoE) for the users. 5G networks can leverage NFV and SDN to solve this problem by creating different virtual networks that are logically isolated but run on the same underlying physical hardware. Each of these logical networks is called a network slice. The network slices can have different structures and different amounts of allocated resources based on the type of use case they are expected to serve. 5G network slicing and orchestration includes a few different aspects that need to be optimized and automated, namely network slice admission control, VNF placement and resource allocation, network slice resource reconfiguration, monitoring and fault detection.

Traditional approaches are unable to solve these problems optimally and some even require a human expert with domain knowledge to be present in the loop. RL-based approaches provide a promising solution to the 5G MANO problems due to their ability to learn the dynamics of the network from historical or simulated data and manage the network slicing and orchestration effectively. Nonetheless, as of now, the different RL-based solutions that have been proposed only look at a subset of the sub-problems. This paper analyzes the major open challenges in 5G orchestration and slicing, describes the drawbacks of traditional approaches, and surveys some of the promising RL-based approaches proposed to solve these problems. Additionally, for the problem of slice admission control, the authors empirically analyze some of the recent RL-based solutions. Finally, we present our own solution to the 5G slicing and admission control problem.

IV. Network Slice Resource Reconfiguration

The underlying infrastructure, on which the network slices operate, has a limited amount of resources available. Therefore, a network slice must be allocated only the resources it can utilize. Once a network slice has been made and allocated an initial amount of resources, the challenge of network slice resource reconfiguration is to learn the long term traffic characteristics of the slices and reconfigure the slice resources to maximize the resource utilization. Service Efficiency (SE) is defined as the bits transmitted per unit of allocated resources, and when network resources are allocated most effectively, SE is maximized. However, each network slice also has a service level agreement (SLA) associated with it. For example, an Ultra-Reliable Low-Latency Communication (URLLC) might have an SLA that indicates its latency must stay below 10ms and that it should always have a minimum throughput of 5Mbps. Quality of experience (QoE) of a slice is defined as the number of packets transmitted through that slice that satisfy its SLA requirements. Traditional approaches to resource allocation, which maximize the SE, take into account only the slice traffic characteristics and allocate resources based on the predicted slice traffic. The main issue with traffic prediction-based resource allocation is that it ignores the QoE of that slice. For instance, URLLC slices usually have very strict SLAs which require the allocation of a reasonable amount of resources, even if there is very small amount of traffic present.

[7] proposes an RL-based solution to this problem. The reward for the RL agent is formulated as a weighted sum of the SE and QoE, therefore, it can maximize the usage of allocated resources, while also taking into account the SLA. In addition to this reward, the environment's state consists of the amount of bandwidth demanded by different slices' traffic, while the agent's action is to allocate the bandwidth (radio resource) to these slices while trying to maximize the reward. The RL algorithm that the authors propose is deep double Q-learning (DDQN)[15] with experience replay. For experience replay, the past experience of the agent is stored in memory and random batches are sampled from this storage during training. It is more efficient in terms of experience collection, since the past experiences can be reused for training in multiple iterations. Additionally, it makes the data more independent and identically distributed (i.i.d) which improves the convergence properties. In network-cloning or double Q-learning, two neural networks - a target network and an evaluation network - are used. Target neural network is used to produce the next action values but its weights are not updated on every iteration. A separate evaluation network is updated on every iteration and its weights are copied to the target neural network after every C iterations. Since the target that the neural network is trying to converge towards is not stationary in RL, the neural network weights can diverge as training progresses. Producing the next action values from a neural network that is not updated for several iterations can rectify this problem of a constantly moving target and leads to better convergence properties.

The paper simulates the network traffic for three network slices, using stationary traffic models, where the incoming slice traffic is sampled from a static underlying distribution. They compare the RL-based approach with two traffic prediction-based approaches and with hard-slicing, where every network slice is allocated an equal share of the total available resources. As hypothesized, these traditional approaches lead to a higher SE but show a complete disregard of SLAs of the slices leading to low QoE, Whereas the RL-based approach leads to a high QoE with a slight SE cost. As opposed to traditional techniques, the RL-based approach has the obvious advantage of being the only approach that can optimize the balance of SE and QoE. However, as indicated in the paper, there are several issues with the RL-based approach. The major issue is that the RL algorithm is slow to converge. Even under a static traffic model, the authors mention that on a CPU-based setup, it takes 2 days for the Q-value function to converge. In actual operation, slices only operate for a time which is in order of minutes, their traffic distribution does not stay static for long, and multi-domain (compute, radio, storage) resource reconfiguration needs to be optimized concurrently. An interesting direction for future research on this topic would be to analyze the convergence properties of different Q-learning algorithms for this problem under dynamic traffic models. Possible changes that may lead to faster convergence could be replacing the deep neural networks with simpler linear function approximators, using a deep-dueling algorithm, and using feature engineering instead of using raw traffic values as the RL agent's input.

V. VNF Placement and Resource Allocation

VNF placement and initial resource allocation is one of the more complex sub-problems associated with 5G network orchestration and slicing. The main idea is that there are geographically and logically isolated servers/nodes available in the network, each with a limited amount of total resources, in which VNFs can be emulated or 'placed'. Each VNF placement requires a certain amount of resources, depending on the number of users that are expected to use that VNF. A network slice consists of chains of these VNFs placed at separate nodes. VNF placement at different nodes affects the QoE of the users as it affects the latency, throughput, coverage, and other slice properties. Additionally, the cost per resource to the operator at different servers is not the same. For example, a UPF may require 10 units of CPU resource and 20 units of storage resource and each server may cost a different amount to allocate these resources and emulate the VNF. Therefore, it is imperative to place the VNF on the available servers to maximize overall utility, i.e., the sum of users' QoE minus the operator's overall resource costs.

[17, 12] present an RL-based solution to this problem, however, they model the problem differently. [17] optimizes the VNF placement and radio resource (sub-channel and power) allocation to the users, using two-stage Q-learning (TSQL). The two-staged approach is used to prevent the Q-table from becoming too large. They use multi-dimensional mapping relationship matrices η and ϕ to denote the VNF-to-Node relationship and user-to-resource relationship. For example, $\eta^k_{m,n} \in \{0, 1\}$ denotes if a VNF of type m belonging to the slice k is placed on the node n. Then the authors formulate the reward or the utility function in terms of these relationship matrices. The input to the first stage's RL agent is the set of VNFs that need placing and the data

rate and latency thresholds for each slice. The output action of this RL agent is the VNF-to-Node relationship matrix. This relationship matrix is then used as input to the second stage's RL agent, where the user-to-resource relationship matrix is obtained as output. The reward is the quality of service (QoS) minus the cost of resource usage at each server, which are a function of the relationship matrices. Additionally, the agent is always given a reward of -1 if it violates a set of preset conditions such as allocating more than a server's available resources to a VNF. By using these formulations of state, action and reward with Q-learning, the RL agent can minimize the overall resource cost and maximize the QoS using the base Q-learning algorithm for both stages.

As opposed to the last paper, [12] do not do the placement for all slices' VNFs in one go. Instead, they look at an already running system, and when a new VNF placement request comes in, the agent decides to either place that VNF on an already running server by assigning it a part of the server's remaining resources, start up a separate server and allocate its resources to the VNF, or upload VNF to the cloud. Another difference, as compared to the last paper's modeling, is that instead of assigning resources to individual users, the authors assign server's resources to the individual VNFs which are then divided equally among its users. There are several costs associated with every state, such as a one-time cost of starting up a new server, a recurring cost as long as a server stays running, a cost for scaling up a server's resources, a cost for uploading the VNF to the cloud, and the costs associated with the assigned resources. These costs make the simulated environment closer to an actual system. The reward is formulated as the QoE minus the sum of all costs. For policy optimization, the authors use an RL algorithm which they call parameterized action twin (PAT). It uses the parameterized action Markov decision process (PAMDP), introduced by [8], with twin delayed DDPG (TD3). PADMPs eliminates the need to dichotomize between discrete and continuous action spaces. In TSQL's second stage, where the authors discretize the assigned resources to create Q-tables, the authors leverage PADMP to use a combination of discrete action space for VNF placement with continuous action space for resource allocation. Next, the authors use TD3 for policy optimization. TD3 algorithm prevents the over-estimation of action-values by using two critic networks during policy gradient.

The baseline approach to this problem is the greedy approach, where the current VNF under consideration is placed on the server having the most resources available and is allocated the highest required resources. This method understandably leads to a degradation of overall utility since it neither incorporates the QoS/QoE, nor the different resource costs at each server. Although the results of the RL approaches cannot be compared directly, due to different modeling of the problem and different definitions of the utility function used, their achieved utility can still be compared to the baseline greedy approach. [17] compare their results with the greedy approach and show that their RL approach always achieves a higher utility and the exact gain depends on the total number of users. Since the greedy approach is quite inefficient, it can be considered quite easy to beat. Therefore, [12] compare PAT with not only the greedy algorithm, but also with other RL algorithms, which include A3C, DDQN, and DDPG. They show that the PAT algorithm can achieve the highest utility out of all these algorithms. In this problem, once the RL agent has learned an optimal policy, it can enact that policy without any need to adapt to a changing environment. Therefore, for PAT, which uses policy gradient for optimization, being slow to converge is not a drawback, since it only needs to converge once in an offline setting. One interesting future direction to explore in this area would be to test these techniques out using an actual testbeds and compare them to find out if the TSQL's simpler formulation being faster to converge has any actual advantage.

VI. Slice Admission Control Qualitative Analysis

Due to limited resources, an operator can only accept a limited number of slice requests. Therefore, when a slice request comes-in, before its VNFs are placed and resources allocated, the operator needs to make sure that the requested resources are indeed available. And, if the resources are not available, the request has to be either rejected or queued until some in-service slices departs and free up the resources. Additionally, slice requests have different resource requirements, arrival and departure characteristics, and if accepted, pay a different amount of revenue to the operator. For example, ambulances in a city may request a URLLC slice an average of 3 times per hour, each time with a certain amount of resources, stop using the slice after an average of 30 minutes, and offer to pay revenue of 3 units to the operator if accepted. The operator needs to analyze these characteristics and accept the slice requests to use the available resource optimally and maximize the long term revenue gained. An instance of smart slice admission control would be that if an operator's infrastructure is operating close to full capacity and the operator knows a high paying URRLC request may arrive very soon, it may decline a low paying best-effort slice request to keep the resources free, and thereby increase its long-term revenue. A greedy algorithm, on the other hand, accepts whatever the highest paying request has arrived right now, no matter how much resources the slice is going to take and for how long.

A. Heuristics-based approaches

[10, 4] propose heuristics-based approaches for the problem of slice admission control. [10] considers a two-layered network model - the RAN layer and the core network (CN) layer. The CN layer contains multiple data center (DC) nodes, the RAN layer contains a set of next generation Node Bs (gNBs). All the gNBs are connected to all the DC nodes. A slice can be placed at multiple gNBs, but only at one DC node. The slice requests consists of the radio resource requirements from each gNB and additional compute, radio and storage requirements, which can be allocated at any of the DC nodes. To admit the optimal set of slice requests, the authors form a 2-step knapsack problem. In the first step, the authors consider all the DC nodes as a single node with cumulative resources. This simplifies the problem to the multidimensional knapsack problem (MKP) - a problem where you have to fit objects with multi-dimensional weights into knapsacks with multi-dimensional resources in order to maximize the reward. Since slices can only be placed at a single DC node, the optimal slice request set needs to now be assigned particular DC nodes. Therefore, in step 2, the authors form a multiple multidimensional knapsack problem (MMKP), which is the same as MKP, but with the addition of multiple knapsacks available to fit the objects into. The solution to this problem gives the optimal DC nodes to host the network slices at. To compare their solution, authors use Mosek toolbox to find the optimal solution. The authors shows that their solution can obtain a slice admission solution in polynomial time with little additional error. It is worth noting that this approach requires all the slice requests to be known in advance to solve the knapsack problem. This is a major flaw, since in real-world scenarios, the slice requests are not known beforehand. Additionally, although there is a semblance of VNF placement with the slices needing to be hosted at particular DC nodes, this is still quite different, because only one DC node is required to host the entire slice and there is no service chaining.

[4] proposes a heuristics-based admission control and resource allocation solution for optimizing the QoE of the users and the network resource utilization. The QoE of a UE depends on the radio resources allocated to it, its inter-user-equipment priority, and the priority of the slice it belongs to. Any new request consists only of the requested radio resources and the associated priorities. The objective of the heuristics-based model is to do admission control and radio resource allocation to these requests, in order to maximize the overall QoE. The admission controller essentially accepts a request if it leads to an increase in the overall QoE, otherwise,

the request is declined. If a request is accepted, then based on the priorities of all the UEs in the network, each one is assigned a new resource allocation such that it leads to a maximum overall QoE. It is worth noting that this is an admission control for UE resource requests, not for slice requests. The authors claim that their algorithm can be extended to slice admission control by changing relevant variables in the algorithm. Finally, the authors compare it with the 4G solution which is essentially the same algorithm, however, it does not take into account slice priorities, as slicing cannot be done in the 4G network. As expected, the 4G solution leads to a worse overall QoE. The authors also test their algorithm without admission control and show that it still leads to a better overall QoE compared to the 4G solution, as it still can take the slice priorities into consideration when allocating resources. Finally, the authors show that their solution can do admission control and resource allocation to UE resource requests that lead to a maximum overall QoE for the network. The issue with this approach is that it is overly simplified. Not only does it not consider multiple kinds of available resources, but it also models the network as a single entity with cumulative resources. In reality, there are multiple kinds of resources that need to be taken into account, and these can be allocated at multiple locations.

B. RL-based approaches

[14] models 5G slice the admission control problem as a variation of the classical multi armed bandit (MAB) problem. The MAB problem imagines a hypothetical scenario where a number of actions are available and each action provides a reward from an associated reward distribution which is unknown to the agent committing the action. The agent has to balance between exploration and exploitation i.e., it can either explore new actions to sample their reward or it can exploit the action that has historically provided the highest reward. The objective of the MAB problem is come up with optimal strategies to maximize the total reward given a limited number of steps. [14] extends the MAB problem for the 5G admission control scenario. They consider that the available actions are to accept slice requests from one of a given set of tenants which provide a reward in turn. And if a slice request is accepted from the tenant, another request cannot be accepted from that tenant until their previous in-service slice departs. Additionally, each slice request from the tenants includes a resource requirement and the sum of the resources for the in-service slices must be less than the available resource budget. To include the requirements, they modify the MAB problem such that the agent can commit multiple actions at any step but each action has a cost associated with it, and this cost must be less than the

maximum budget. Additionally, if any action is committed, it cannot be committed again for a given number of time steps. They call this variation of the MAB problem as the budgeted lock-up multi-armed bandit (BLMAB) problem. After the problem formulation, what is left is to propose a strategy to achieve the optimal exploration-exploitation balance. To this end, the authors propose a variation of the enhanced upper confidence bound (eUCB) [6] - called ONETS - where instead of a variable number of actions, only a fixed number of actions can be taken at any time step. They show that this sub-optimal solution is better suited for the 5G slice AC problem due to its lesser time complexity. Finally in addition to eUCB, the compare ONETS with the classical ϵ-greedy algorithm. The authors show that their solution achieves a execution time and mean achieved reward that lies between ϵ-greedy and eUCB. Additionally, the authors show the practicality of the BLMAB, using a real-life testbed with OpenEPC [1] used to emulate the substrate network.

[11] also proposes an RL-based admission control solution. The substrate network is divided into three locations - the central office (CO), the optical backhaul network (OBH) and the regional data center (RDC) - each one containing its respective cumulative resources. The incoming slice requests are categorized into two types - low priority (LP) and high priority (HP) - and contain the resources required from each location and the offered revenue. The LP slices require a small amount of resources from all three locations while the HP slices require a large amount of resources only from the CO. An RL-based admission controller is used to admit the incoming slice requests. Any in-service slices are allowed to alter their resource usage. The reward for the RL agent, at each time-step, is directly proportional to the slice's offered revenue and inversely proportional to the SLA violation penalty in case if a slice tries to increase its resource usage but no more resources are available in the network. This incentives the agent to not only analyze the slice-request traffic pattern but also their resource usage pattern. The authors include an array of system's remaining available resources and an array containing the requested slice type and its requirements in the observation space for the RL agent. The authors compare this strategy with a random policy which accepts slice requests randomly and a greedy strategy where slice requests are accepted if required resources are available. The authors show that the RL agent is able to learn not only the slice request traffic characteristics but also its resource usage as the RL agent starts to decline some LP requests in case if an in-service HP slice needs to increase its resource usage. Finally, the authors show that RL-based admission control can lead upto 54.5% more performance in terms of SLA violation penalty as compared to the random approach and

also always leads to a more performance as compared to the greedy approach as well.

[3] is a precursor paper of [2], therefore, we leave it out of this survey. [2, 5, 16] analyze the problem of optimal slice admission control using RL. In these papers, the terms *revenue* and *priority* are used interchangeably. The major factor that differentiates [5] from the other two papers is its modeling of the slice admission control problem as another resource allocation problem. In the latter two papers, the RL agent outputs an explicit action accepting or rejecting a slice request, but in this paper, the RL agent assigns a proportion of the infrastructure's remaining resources to each slice request, and if the assigned resources are greater than the required resources, the slice request is accepted, otherwise, it stays enqueued. The RL agent's input is the current queue/buffer levels and the requested resources by slice requests in those queues. The loss function is formulated as a weighted sum of the waiting time of all slice requests in the queues and the requested resources. This waiting time is weighted higher for slices with higher priorities. The negative of this loss is used as a reward, and this incentivizes the RL agent to assign the resources to slices of each priority in such a way that slices with higher priority do not stay queued for long, and slices that do not require too much resources are readily accepted. Since the assigned resource proportions output by the agent are continuous, the authors use the REINFORCE algorithm for policy gradient. By evaluating both synthetic data and using real-world data, the authors show that this RL-based technique can achieve higher rewards when compared to a greedy approach. Additionally, the authors observe that the algorithm allocates higher average resources to high priority slices, which is as expected.

[2, 16] both model the problem similarly. In the former, each slice type is labeled as either *best effort* or *guaranteed service*, which is similar to assigning a priority of 1 and 2, respectively. Whereas in the latter, slices are assigned priorities in the range of 1-3. The second difference in problem modeling is in the presence of a queue. In the former, there are separate finite FIFO queues for both types of slices, and any incoming slice request is dropped if the associated queue is full. In the latter, any slice requests that the RL agent does not accept are dropped instantly. Another caveat is that in [2], the authors add the concept of resource multiplexing and elastic VNFs to their modeling, but these are beyond the scope of basic slice admission control and therefore, not covered in this survey.

Both of these papers define the reward function as the operator's long-term revenue, which is also called the network reward. In [2], since the authors use queues, the state space is defined as the current status of the queue of each type, and the agent's action is to accept the slice requests from these FIFO queues. On the hand, in [16], since no queues are used, the state space is the current arrived slice request and the agent's action is to either accept or decline this request. Now, having defined the problem in terms of state, action, and rewards, what remains is an optimization algorithm. [2] propose using SARSA with linear function approximation (LFA). In SARSA with LFA, instead of using a neural network to learn action values, a linear function is used. The authors' formulation suggests that they used generalized linear regression, but they do not mention the basis functions used. [16] propose using DDQN with dueling. The advantage of using DDQN over DQN has been described above. The dueling architecture was introduced by [18]. In this architecture, instead of a fully connected neural network producing the action-value $Q^{\pi}(s, a_s)$ directly, two separate fully-connected streams produce the state value $V^{\pi}(s;\beta)$ and an advantage function $G^{\pi}(s, a_s;\alpha)$, where α and β are the weights of the two fully-connected streams. The state value denotes how good it is to be in a particular state, the action advantage denotes how favorable the action in that state is, and their sum equals the action-value. However, by using this formulation, given a particular Q, V and G cannot be found uniquely. Therefore, as given in the paper, some additional modifications are made to the architecture and the action-value is actually calculated as:

$$Q(s, a_s;\alpha,\beta) = V(s;\beta) + (G(s, a_s;\alpha) - \tfrac{1}{|A|}\textstyle\sum_{a_s} G(s, a_s;\alpha))$$

where $|A|$ denotes the total number of actions. The intuition behind this type of network architecture is that when the action space is large, it becomes quite difficult for the agent to learn the action-value for all actions in all states. Additionally, in some states, it does not matter much what action the RL agent takes. Therefore, by approximating the state-value and action advantage separately, the agent can learn which states are important and which are not, without having to learn the associated action values.

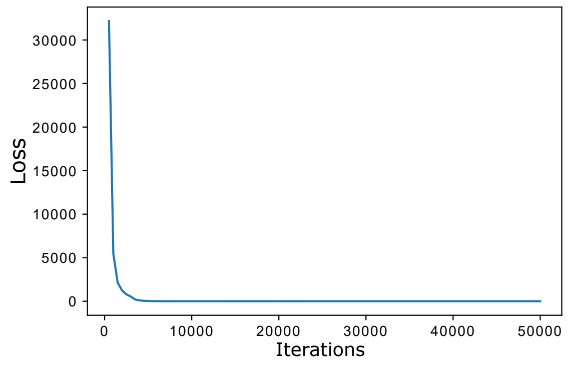
The authors' evaluation using synthetic data shows that this architecture achieves 40% higher reward on average, as compared to the greedy approach. Not only that, but the authors evaluate that the DDQN with dueling can obtain the optimal policy within $15,000$ iterations where the base Q-learning algorithm takes more than 10^7 iterations to converge to the optimal policy. Similarly, in the paper by [2], the authors show that SARSA with LFA can achieve a higher reward as compared to the greedy approach at different slice-request arrival probabilities.

VII. Slice Admission Control Quantitative Evaluation

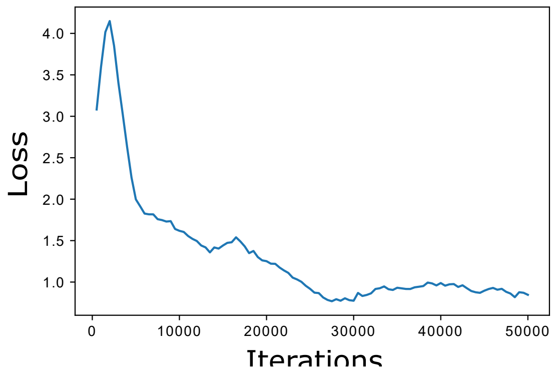
Due to the existence of numerous problem models, it becomes difficult to directly compare two different approaches in terms of the machine learning effectiveness. Therefore, we model a slice admission control problem based on common model features described in [2, 16], and compare the RL algorithms proposed - SARSA with LFA and DDQN with dueling. The basic problem stays the same - maximizing the operator's reward by intelligently accepting slice-requests. We assume that there are slices of three types 'Voice over LTE (VOLTE)', 'Video' and 'URLLC'. Each slice gives the same reward of 1 unit if accepted, but if a URLLC slice is dropped, the agent gets a reward of -5 units. This makes the 'VOLTE' and 'Video' slices of equal priority and the 'URLLC' slice of higher priority. Additionally, queueing is retained and the state space is the current status of the queues and the infrastructure's remaining resources. The agent's objective is to maximize the operator's long term reward by trying to maximize the total number of slice requests accepted, while minimizing the number of high priority 'URLLC' requests dropped.

We use a request arrival probability of 0.85 per time step for slices of each priority and a maximum of 2 requests of each priority can arrive at any time step. The environment has an overall 800 units of each radio, compute, and storage resources. Each slice request requires 200 units of each resource to be accepted. The slice requests when accepted, depart, and free up the resources with a probability of 0.35 at each time step. The FIFO queues for each priority have a maximum size of 4. For SARSA with LFA, we do not use basis functions. For both algorithms, an experience memory is used to store the most recent 10000 experiences, i.e., tuples of form $(s_t, a_t, s_t^{'}, R(s_t, a_t))$. During training, random batches of size 32 are sampled from this storage. For both of these algorithms, ϵ-greedy policy is used for exploration/exploitation, where ϵ starts at 1.0 and is decreased by 0.1 every 500 iterations, until it reaches the minimum value of 0.1. For DDQN with Dueling, the target network is updated after every 500 iterations, while the evaluation network is updated after every iteration. The loss, reward and slice dropping probabilities achieved are shown in figure 1.

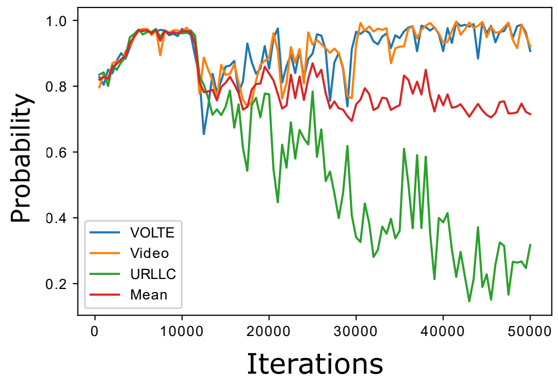
Figure 2 shows the mean reward achieved by using the greedy approach at different URLLC slice arrival probabilities, whereas figures 1(e) and 1(f) show the reward averaged over every 500 iterations using RL approaches at URLLC slice request arrival probability of 0.85 per time step. We can see that with the greedy approach, as the slice arrival probability of URRLC slices increases, it starts dropping more of these requests and incurs a negative reward. At a



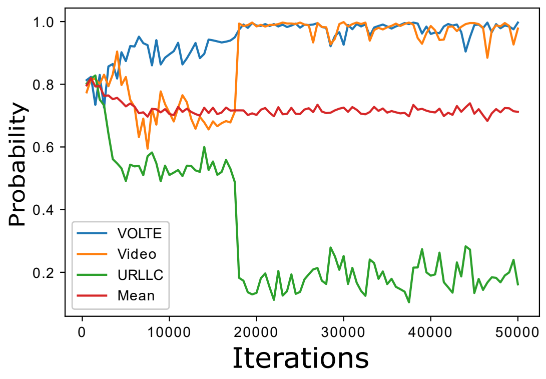
(a) Dueling DDQN MSE loss.



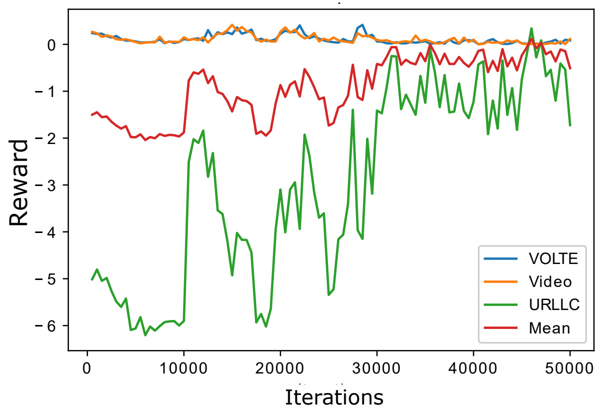
(b) SARSA LFA MSE loss



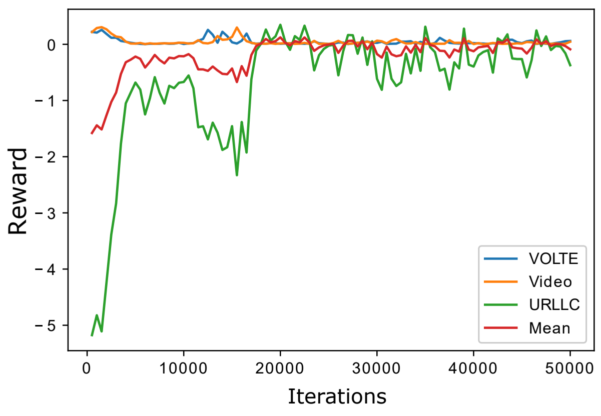
(c) DDQN slice drop probability.



(d) SARSA slice drop probability



(e) Dueling DDQN reward



(f) SARSA LFA reward

Fig. 1: RL Slice Admission Control Curves

probability of 0.85, its mean reward is less than -1. The RL approaches learn the cost associated with dropping URLLC slice requests, and after some iterations, their URLLC slice dropping

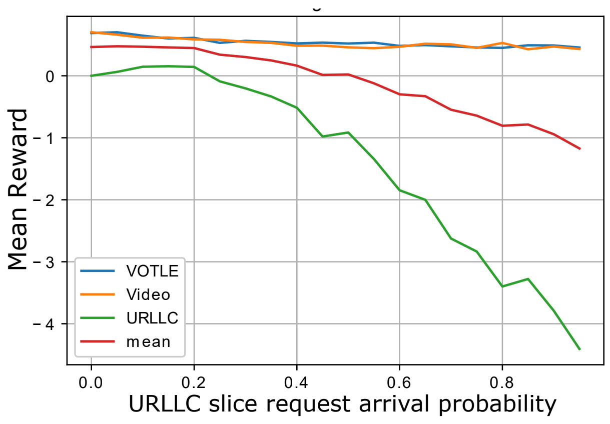


Fig. 2: Greedy reward

probabilities decreases and the reward increases. It can be observed that SARSA with LFA not only achieves a higher reward within $50,000$ iterations, but also shows less variance in the reward curve. Additionally, due to having much smaller number of trainable parameters, SARSA with LFA takes smaller amount of time and GPU resources to train. Therefore, we can say that for this problem, SARSA with LFA performs better and faster than the more complex DDQN with dueling algorithm.

VIII. Proposed 5G Slicing with Admission Control Solution

It is evident from the analysis so far that RL-based 5G MANO has a clear advantage over baseline techniques. However, none of the RL-based approaches present a solution to the overall problem, and instead, only address parts of the problem. It is not made clear how these approaches could be integrated with one another to address the broader issue of optimal 5G MANO. For instance, in the problem of admission control, the main focus of these approaches is to learn the distribution of incoming slice traffic, while ignoring how the slice is going to be actually embedded as a substrate network. In other words, the slicing agent does not interact with the admission control agent. This is a major flaw that makes these approaches impractical for real-world applications. As a result, we set out to develop an approach that takes into account all the major variables required to prevent it from being too simplistic. For the rest of the report, we use the terms Service Function Chain (SFC) and slice interchangeably.

First, we design a VNF catalogue containing the available VNFs, where each VNF v has its own set $(S(v), C(v))$ of storage and computation requirements. For the initial design, we decided to start with 10 VNF types. An SFC S_i has an ordered list $V_i = \{v_1, v_2, ..., v_{l_i}\}$ containing its requested VNFs from the catalogue, where $2 < l_i < 5$. In addition to these requirements, each VNF v in an SFC s has a compression ratio of $Z_s(v)$, which denotes the compression that will be applied to the incoming data when it passes through that VNF, reducing the required bandwidth after it. Moreover, each SFC S_i has delay, data rate, and operation time SLAs denoted by $Q_d(S_i)$, $Q_r(S_i)$, and $O(S_i)$, respectively. For a successful embedding, these requirements must be satisfiable during the admission.

The substrate network is also represented by an undirected graph $G(N, E)$ where the nodes N denote the server nodes with compute and storage resources and the edges E denotes the links between nodes, where each link has a latency associated with it. The graph is divided into two distinct clusters to simulate the Distributed Units (DUs) and Central Units (CUs) in a 5G environment. These two clusters are connected through a fronthaul link of limited bandwidth capacity. This functional split of the 5G infrastructure was defined in 3GPP release 15. It brings flexible hardware and software and consequently, a scalable and cost-effective network deployment. Dividing the network modules also allows for the acquisition of different software and hardware from different vendors, and thus, bringing adaptable and customizable services according to user needs. Figure 3 shows an example substrate network graph.

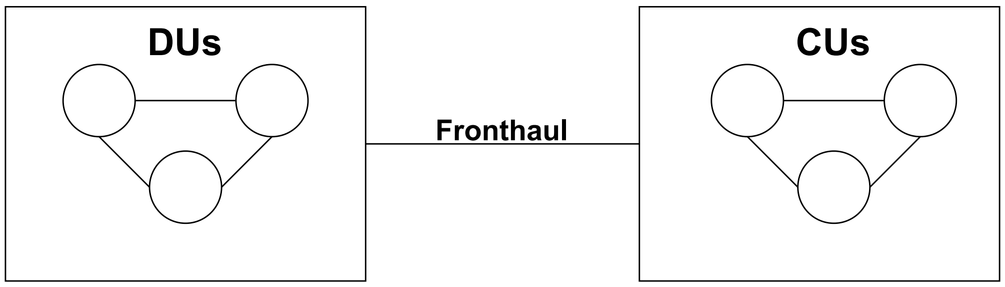


Fig. 3: Sample Network Graph

For embedding an SFC onto the substrate network, we use an SFC to substrate node relationship matrix ϕ of size $m * n$ where m denotes the maximum SFC length, and n denotes the number of substrate nodes. The links between the embedding nodes are determined using shortest-path algorithm. An embedding produced by the embedding relationship matrix is successful if the following conditions are met:

1) One VNF is not placed on multiple substrate nodes.
2) The substrate nodes have enough compute and storage resources available to host the VNF.
3) FH bandwidth limit is not exceeded.
4) The embedding delay is less than, or equal to the delay SLA of the SFC.

The embedding matrix is produced by the 5G slicing module. In addition to this, we introduce the admission controller module. The job of the admission controller is to interact with the 5G slicing module and decide whether to accept or decline an arriving slice request, by analyzing the network status and possible embedding location retrieved through the slicing module. The overall structure of the environment is given in Figure 4.

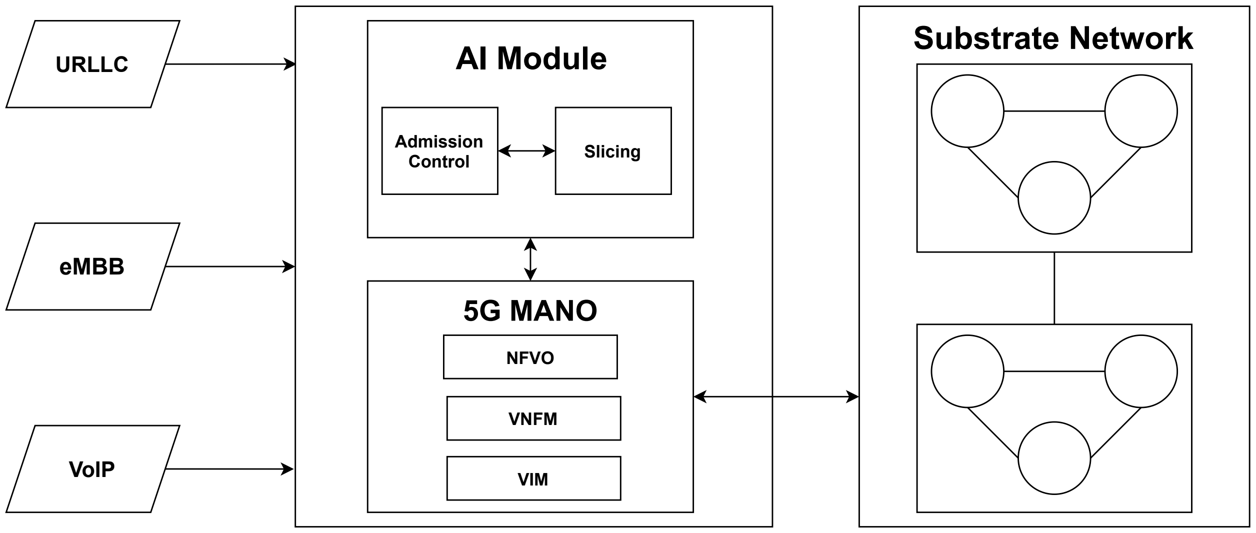


Fig. 4: 5G MANO with AI

Arriving slices are among 3 categories, URLLC, eMBB, and VoIP. Each of these slice requests have different SLA requirements. URLLC slices are used for time-critical use-cases such as remote controlled vehicles, where each ms of delay counts. For these type of slices, the requirements are set to a maximum 4.9 ms of delay and a minimum data rate of 1000 mbps. With the 5 ms delay occurring when data traverses the fronthaul, these slices require all their VNFs to be placed on the DU nodes.

Furthermore, eMBB slices require massive bandwidth and have a high delay tolerance. These slices are used to provide services to use-cases such as streaming video, where the video buffer at the receiver allows for some jitter. In our model, these slices require a maximum delay of 25 ms and a minimum data rate of 2500 mbps. Such delay eliminates any need for special care in terms of placement, as even the worst case placement results in a maximum delay of 25 ms. But if some of its VNFs are not placed in the DU for data compression, it can consume a high fronthaul capacity causing some of the next slices to be dropped. The last type of slice in our model is the VoIP model, which is not demanding in terms of the data rate. This slice type has an associated delay SLA of 9 ms, which sits between the previous two slice types. These slice types and their associated SLAs are given in Table 1.

Slice Type	Arrival Rate	Operation Time	Size	Delay SLA	Data Rate SLA	Offered Revenue per time-step
URLLC (0)	Exponential Distribution (Avg: 1 per 3 time-steps)	Exponential Distribution (Avg: 30 time-steps)	Uniform Distribution (Low:2, High:5)	5ms	1000Mbps	Uniform Distribution (Low: 6, High: 15)
eMBB (1)	Exponential Distribution (Avg: 1 per 3 time-steps)	Exponential Distribution (Avg: 30 time-steps)	Uniform Distribution (Low:2, High:5)	25ms	2500Mbps	Uniform Distribution (Low: 2.5, High: 6)
VoIP (2)	Exponential Distribution (Avg: 1 per 3 time-steps)	Exponential Distribution (Avg: 30 time-steps)	Uniform Distribution (Low:2, High:5)	9ms	50Mbps	Uniform Distribution (Low: 0.8, High: 2)

TABLE I: Slice Types

Arriving slice types are distributed randomly among the mentioned types and the slices arrive randomly with an average of once per time step. The operation time of the slices is also decided randomly with an exponential distribution with an average of 30 time steps.

A. RL Algorithm Design

The objective of an RL-agent is to maximize the mean reward, but this does not necessarily mean that it can be set equal to the revenue. The reason is that if the reward is not designed to make learning easier for the agent, it can lead an RL agent towards non-optimal pathways. We meticulously designed this reward such that it leads to the maximum revenue. To this end, we design and train an RL model with both a VNE module and a slice admission module by using the following model to convert our admission control and VNE problems to an RL scenario.

As already mentioned, each RL solution has three parts, the observation space, the action space, and the reward function. The observation space in our model includes the current state of the network and the arriving slice request information. The current state of the network comprises of the computation and storage resource usage at each node, in addition to the fronthaul link bandwidth usage. Moreover, as the last part of the observation space, the arriving slice is introduced to the agent by the slice revenue, its SLA requirements, its VNF compression ratios, and its operation time.

The action space includes a binary variable, indicating the admission status of the current slice at hand, as well as the embedding relationship matrix. A positive reward, relative to the slice's type and operation time is given to the agent in case of a successful SFC embedding. The reward value is equal to SFC's offered revenue per time-step multiplied by the slice's operation time if case of a successful embedding. A negative reward is given to the agent if either delay SLA or fronthaul bandwidth constraint are not met by the produced embedding matrix. An optimal agent would reject low paying high resource cost slices to reserve resources for the high paying low resource cost slice requests, in addition to successfully embedding the SFC while considering the potential future requests and the slice SLAs.

To optimize this reward we use the Proximal Policy Optimization[13] (PPO) algorithm, which as the name suggests, is an on-policy model-free policy-based algorithm. PPO scales out across our computation cluster, by employing multiple workers, each with their own copy of the environment and collecting experience. Our learning platform consists of 4 nodes, each with 8 CPU cores, which are controlled by the head node through Ray. Ray is a distributed processing platform that includes an RL module named RLLib. Additionally, our custom environment was implemented in OpenAI Gym, which is a platform dedicated to creating and testing RL algorithms. In the end, RLLib was integrated with OpenAI Gym to train and test our RL models.

The environment starts with 6 nodes, connected as shown in figure 3. The link delays between the nodes in each cluster are uniformly distributed between 0.2 and 0.8 ms, and the frounthaul delay is set to 5 ms. Each node in the distributed cluster (DU) has 800 units of storage and 800 unit of computation resource. These numbers are 1600 units of storage and 1600 units of computation resource for the nodes in the central cluster (CU), due to the nature of such clusters. The fronhaul capacity is set to 1000Mbps.

With each arriving slice request, the agent chooses the appropriate action to accept or reject the slice. If the agent decides to accept a slice, the environment attempts at embedding the request according to the SFC embedding matrix. During this time, it checks whether the substrate network addresses the delay SLA, whether the fronthaul link has enough available capacity to host the substrate network with the compressions applied, and whether the selected servers have enough computation and storage resources left to host the corresponding VNFs. If all the constraints are met, the SFC will be successfully embedded. This would result in the available resources being reduced in the servers hosting the new VNFs and the available fronthaul bandwidth being reduced by the amount consumed by the new SFC. The agent receives the reward as defined previously based on its output action.

IX. Results

The training session is run for 100 million steps, which takes 14 hours approximately, and in the end the agent takes 0.07114 ms to make each decision. The training was done with the following PPO parameters, which we have reached upon through hyperparameter optimization:

- Neural network model: 4 fully connected hidden layers with 256, 256, 128, and 128 nodes, respectively, and with RELU activation function.
- Learning rate: Decreasing linearly, starting from 0.003 at timestep 0 and reaching 0.001 and 0.0004 at timesteps 70,000,000 and 100,000,1000, respectively.
- KL coefficient: 0.4
- Lambda: 0.95
- Number of Stochastic Gradient Descents (SGD): 5
- SGD minibatch size: 4096
- Training batch size: 102400
- Framework: PyTorch

In order to evaluate the effectiveness of our trained model, we first create a close to ideal slicing algorithm based on the types of requests and the substrate network structure. This is done with prior knowledge of the environment and the structure of the arriving requests. Therefore, it would be not be feasible in real-life scenarios, since such information may not always be available and it would not be practical to manually design such an algorithm for every telecom center, specially in more complex scenarios. The purpose of this semi-ideal algorithm is to provide a rival to demonstrate the capabilities of reinforcement learning. Thus, the aim of the RL model would be to learn from the slice requirements and do slicing and admission control to achieve comparable or better performance than this near-ideal slicing algorithm

In this algorithm, for each requested VNF in the arriving SFC request, candidate substrate nodes are sorted according to their position, available compute resources, and their available storage resources, in order. The preferred node locations are the DU nodes for URLLC and eMMBC slices and the CU nodes for the VoIP slices. The reasoning behind this decision is as follows. For URLLC slices, the only possible substrate nodes are the DU nodes, so they are preferred. For eMMBC slices, the fronthaul link's available bandwidth is a bottleneck, therefore, the VNE module must put as many VNFs as possible on the DU nodes to utilize the compression ratios the most. For VoIP slices, since they use little fronthaul capacity and are tolerant to the fronthaul link delay, they can utilize the available resources of the CU. After the priorities are calculated, the first node in the list with the available resources to host the VNF is selected. This guarantees that for the current request and with the current network state, the lowest delay substrate network with the lowest fronthaul bandwidth usage is selected.

As the baseline, we employ a greedy approach that for each VNF, selects a substrate node with the highest available resources. The results of all these approaches, averaged over 100 episodes is provided in table 2.

As it can be seen in the table, our RL model achieves 16% higher revenue against the manual near-optimal approach, with the greedy approach set as the baseline. Although the manual approach has a slightly higher resource utilization and fronthaul bandwidth utilization, it does not take advantage of the higher profit of embedding the URLLC slices. This is evident from the 31 percent of slices of type URLLC embedded by the RL model, while this number is at 18 percecnt for the manual approach. Additionally, the VNE module is more successful than the other methods, with its 67 percent successful slice embedding.

Taking a look at the system state during the training provides us with great insight into the network trends and offers valuable tips to create better manual solutions, if needed.

For instance, figure 5 shows a clear decline in the number of eMBB slices admitted and a rise in the the number of VoIP slices admitted. This is due to the fact that the available fronthaul bandwidth is a bottleneck for eMBB slices, and therefore, these slices have to contend for the DU nodes. With the larger revenue offered by URLLC slices, the AC module will gradually learn to prefer slices of this type. As explained before, VoIP slices can be embedded comfortably in the CU nodes and thus, they do not have a bottleneck, which results in the gradual increase in their admission rate. Even though all this was apparent with the nature of these slice types and their requirements versus their offered revenue, calculation of the exact admission/rejection ratio

TABLE II

	Greedy	**Near-optimal**	**RL**
Average In-service slices	3.08	7.05	6.46
FH bandwidth available / Total bandwidth	0.80	0.48	0.50
Average DU computation resource utilization	0.10	0.82	0.70
Average CU computation resource utilization	0.35	0.57	0.53
Average DU storage resource utilization	0.10	0.82	0.70
Average CU storage resource utilization	0.35	0.57	0.53
Admitted slices / Total requests	1	1	0.69
Embedded slices / Total Admitted	0.24	0.55	0.67
Embedded slices of type URLLC / Total embedded	0.02	0.18	0.31
Embedded slices of type eMMB / Total embedded	0.07	0.29	0.17
Embedded slices of type VoIP / Total embedded	0.90	0.51	0.51
Server resource constraint not met / Total admitted	0.00	0.07	0.53
FH bandwidth constraint not met / Total admitted	0.62	0.31	0.16
Delay constraint not met / Total admitted	0.43	0.25	0.10
Total revenue	4.99	24.16	27.33

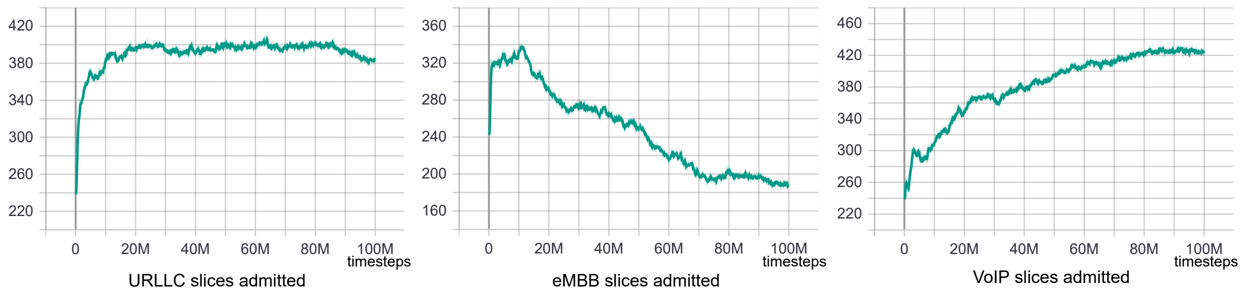


Fig. 5: Admitted slices per type

would have been too complicated to be carried out manually.

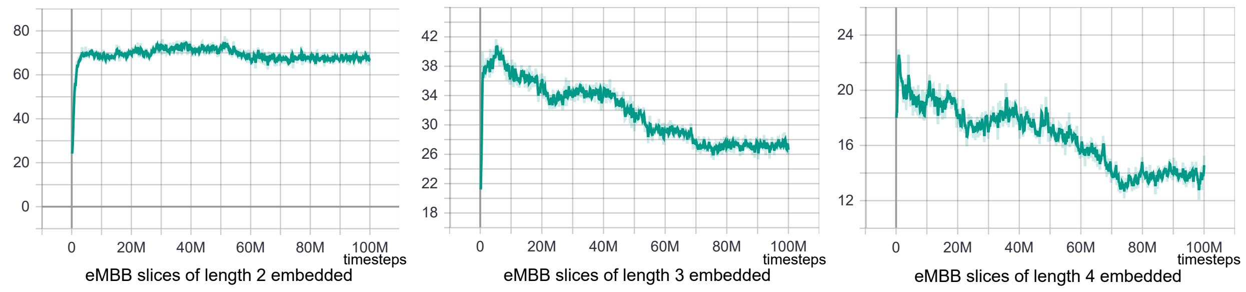


Fig. 6: Embedded eMBB slices per length

Figure 6 exhibits the optimality of accepting shorter slices, when compared to the lengthy ones as slices with longer SFCs consume more resources. The decreasing admission rate of the eMBB slices is reflected more in the longer SFCs, as they appear to have a steeper slope in their embedding count.

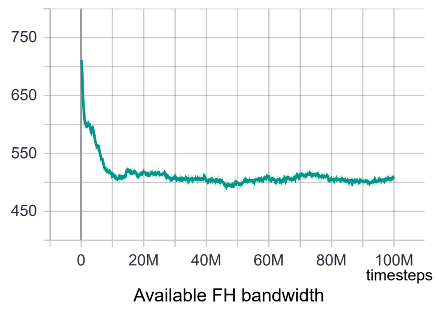


Fig. 7: Available fronthaul bandwidth (Mbps)

Figures 7 and 8 display the learning process of the agent, as it learns to utilize more resources. CU resource utilization plots combined with the VoIP slice admissions plot, verifies our hypothesis that in order to optimize the resource usage, we must host VoIP slices in the CU and dedicate DU nodes to the other slice types.

X. Conclusion

Modern telecommunication networks are all about automation and scaling. Therefore, in this project, we attempted at employing reinforcement learning, which is well-known for its autonomousity, to tackle the problem of 5G slice orchestration and management. It is clear from

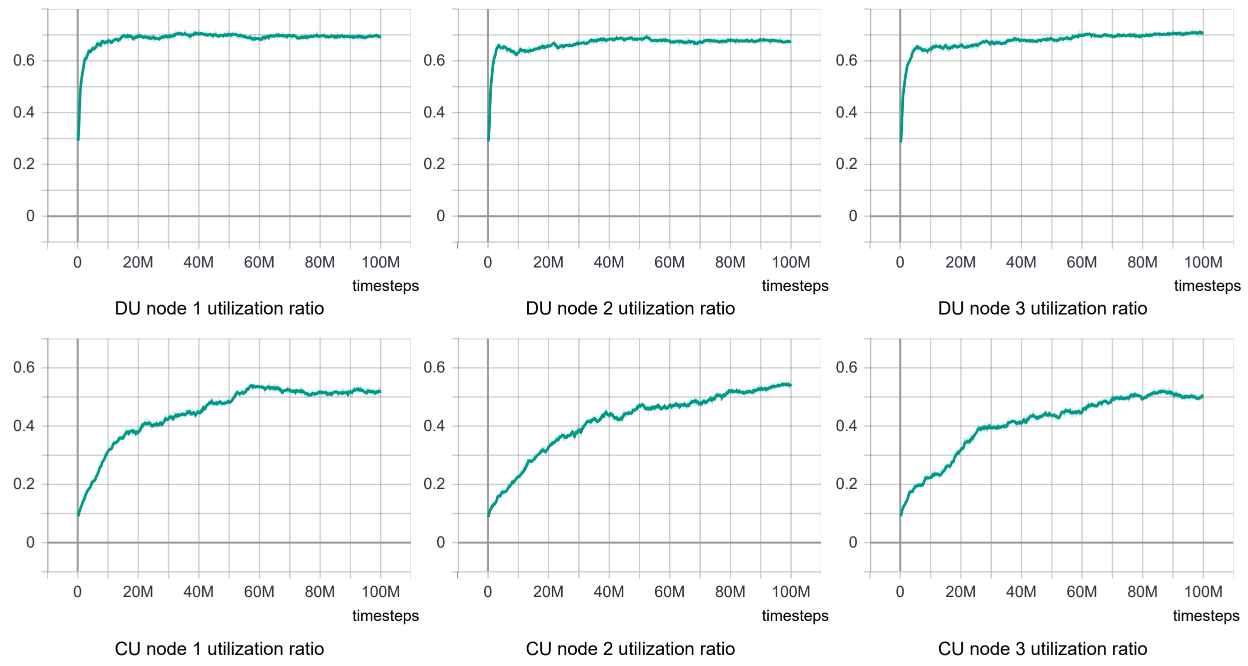


Fig. 8: Computation resource utilization per node

this survey that 5G MANO is a complex and multifaceted problem. There have been several AI-based solutions proposed to address each of these facets and reinforcement learning dominates this solution space. But there are two major issues with the current approaches - they either have an impractical model of the system or they can't dovetail with other approaches. Additionally, due to lack of standardization in system modeling, it becomes harder to compare any two solutions. We presented an integrated solution for 5G slicing and admission control that takes into all the major variables required to make it practical for a real-life scenario. Our initial results shows that our solution can improve the InP's revenue by over 4 times when comparing to greedy baseline approach. And it can also outperform near optimal approaches designed meticulously based on the system model and slice traffic.

In the future, one of the possible improvements that could be made is to use multi-agent RL. As things stand currently, the same reward is given to both AC and slicing agents, which leads to lesser control over their incentive structures. Moreover, we plan to add additional improvements such as variable offered revenue based on the SFC length, use more accurate distributions to generate slice request traffic, use more accurate compression ratios and resource requirements based on values from actual VNFs.

References

[1] Marius Corici, Fabricio Gouveia, Thomas Magedanz, and Dragos Vingarzan. Openepc: A technical infrastructure for early prototyping of ngmn testbeds. In Thomas Magedanz, Anastasius Gavras, Nguyen Huu Thanh, and Jeffry S. Chase, editors, *Testbeds and Research Infrastructures. Development of Networks and Communities*, pages 166–175, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[2] Ghina Dandachi, Antonio de Domenico, Dinh Thai Hoang, and Dusit Niyato. An artificial intelligence framework for slice deployment and orchestration in 5g networks. *IEEE Transactions on Cognitive Communications and Networking*, 6(2):858–871, 2020.

[3] B. Han, A. DeDomenico, G. Dandachi, A. Drosou, D. Tzovaras, R. Querio, F. Moggio, O. Bulakci, and H. D. Schotten. Admission and congestion control for 5g network slicing. In *2018 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 1–6, 2018.

[4] M. Jiang, M. Condoluci, and T. Mahmoodi. Network slicing management prioritization in 5g mobile systems. In *European Wireless 2016; 22th European Wireless Conference*, pages 1–6, 2016.

[5] Jaehoon Koo, Veena B. Mendiratta, Muntasir Raihan Rahman, and Anwar Walid. Deep reinforcement learning for network slicing with heterogeneous resource requirements and time varying traffic dynamics. In *2019 15th International Conference on Network and Service Management (CNSM)*, pages 1–5. IEEE, 2019.

[6] T.L Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

[7] Rongpeng Li, Zhifeng Zhao, Qi Sun, Chih-Lin I, Chenyang Yang, Xianfu Chen, Minjian Zhao, and Honggang Zhang. Deep reinforcement learning for resource management in network slicing. *IEEE Access*, 6:74429–74441, 2018.

[8] Warwick Masson, Pravesh Ranchod, and George Konidaris. Reinforcement learning with parameterized actions.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep

reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[10] K. Noroozi, M. Karimzadeh-Farshbafan, and V. Shah-Mansouri. Service admission control for 5g mobile networks with ran and core slicing. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019.

[11] M. R. Raza, C. Natalino, P. Öhlen, L. Wosinska, and P. Monti. A slice admission policy based on reinforcement learning for a 5g flexible ran. In *2018 European Conference on Optical Communication (ECOC)*, pages 1–3, 2018.

[12] Joan S. Pujol Roig, David M. Gutierrez-Estevez, and Deniz Gündüz. Management and orchestration of virtual network functions via deep reinforcement learning.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[14] Vincenzo Sciancalepore, Lanfranco Zanzi, Xavier Costa-Perez, and Antonio Capone. Onets: Online network slice broker from theory to practice, 2020.

[15] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[16] Nguyen van Huynh, Dinh Thai Hoang, Diep N. Nguyen, and Eryk Dutkiewicz. Optimal and fast real-time resource slicing with deep dueling neural networks. *IEEE Journal on Selected Areas in Communications*, 37(6):1455–1470, 2019.

[17] Xiaofei Wang and Tiankui Zhang. Reinforcement learning based resource allocation for network slicing in 5g c-ran. In *2019 Computing, Communications and IoT Applications (ComComAp)*, pages 106–111. IEEE, 2019.

[18] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning.