A Reinforcement Learning-Based Routing Strategy for Elastic Network Slices

Zhouxiang Wu and Jason P. Jue
Department of Computer Science, The University of Texas at Dallas, Richardson, Texas 75080, USA
Email: {Zhouxiang.Wu, jjue}@utdallas.edu

***Abstract*—This paper addresses a routing selection strategy for elastic network slices that dynamically adjust required resources over time. When admitting elastic initial slice requests, sufficient spare resources on the same path should be reserved to allow existing elastic slices to increase their bandwidth dynamically. We demonstrate a deep Reinforcement Learning (RL) model to intelligently make routing choice decisions for elastic slice requests and inelastic slice requests. This model achieves higher revenue and higher acceptance rates compared to traditional heuristic methods. Due to the lightness of this model, it can be deployed in an embedded system. We can also use a relatively small amount of data to train the model and achieve stable performance. Also, we introduce a Recurrent Neural Network to auto-encode the variable-size environment and train the encoder together with the RL model.**

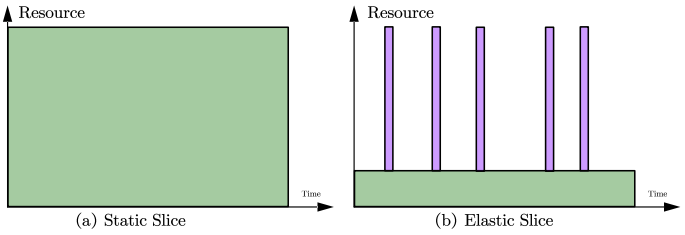
***Index Terms*—network slice, reinforcement learning, routing, policy gradient**

I. Introduction

In the 5G era, network customers expect more tailored services. With network slicing technology, network resource can be easily partitioned into different slices to fulfill the different requirements of different customer requests. Traditionally, the resources assigned to a slice do not change over time. However, as the usage of resources on a slice varies over time, if a slice is provisioned for its peak resource usage, resources may be wasted. For this reason, elastic slices are proposed to address this issue. With elastic slices, the provisioned resources vary according to the needs of the slice during service time, resulting in cost savings and more efficient utilization of system resources. However, the elastic slices add more complexity to resource allocation and scheduling.

The slice allocation strategy, which makes the slice admission decision and maps the slice request to proper resource, is an important problem. For example, without an effective strategy, the network system may accept too many slice requests such that the system is unable to accommodate a request from an existing slice to increase its resources, leading to a degradation of slice users' quality of experience and a potential loss of revenue. In this paper, we propose a Reinforcement Learning slice allocation strategy which can effectively map a slice to a path in the topology.

To the best of our knowledge, this paper is the first to apply RL to the elastic slice request routing problem. The remainder of the paper is organized as follows. We discuss related work in Section II. Section III introduces the system model and problem formulation. Section IV gives a brief background on related machine learning concepts, and describes the procedure of feature selection, the proposed RL model, and the RL algorithm. In Section V, we describe experiments for evaluating the proposed approach, make essential observations, and give suggestions on the training model. Finally, we conclude the paper in Section VI.



(a) Static Slice (b) Elastic Slice

Fig. 1. Elastic and Static Slice

II. Related Work

We extend the problem from [1], which only considers the admission strategy for the elastic slice. Several papers apply RL to routing problems. However, they do not consider elastic requests. In [2], the authors use an RL algorithm to learn an SFC scheduling policy to increase the success rate of SFC requests. In [3], the authors provide an online routing reinforcement learning algorithm with multiple agents for multiple service requirements. The algorithm is scalable because the algorithm is implemented in each node of the network. The author also normalizes different performance metrics.

III. System Model and Problem Formulation

A. Physical Network Infrastructure

The physical network is represented as a graph G with a node set V, and an edge set E. Each link $e \in E$ has bandwidth bw_e. For each source-destination pair (s, d), there is a predefined path set $P_{(s,d)}$ where each $p \in P$ is a subset of edges in E.

B. Network Slice

In general, a slice consists of a set of virtual networking and computing resources configured in a virtual topology. In this paper, we consider a linear slice topology; thus, a slice is characterized by a source and destination node, as well as a number of intermediate nodes corresponding to functions that

have some computing requirements. Virtual links of the slice require some amount of bandwidth. In this paper, we consider two types of slices: static slices, in which the amount of bandwidth on each slice link remains fixed for the duration of the slice, and elastic slices, in which the amount of bandwidth on slice links may increase or decrease during the lifetime of the slice. We assume for simplicity that all edges of a given slice have the same bandwidth at any given time. We further assume that each path p has sufficient computing resources to accommodate the functions of a slice. Fig. 1 illustrates the bandwidth allocation for a static slice and a dynamic slice. Note that an elastic slice offers the potential for significant resource and cost savings versus allocating for the peak capacity requirement in a static slice.

Slice $deployment\ requests$ arrive to the network dynamically over time and can be for either a new static slice or a new elastic slice. Furthermore, existing elastic slices may generate slice $scaling\ requests$, which are requests to increase or decrease the bandwidth of the slice. A slice deployment request can be represented as a tuple $r = (src, des, bw, t_{arr}, t_{hold}, type)$, where src is the source node, des is the destination node , bw is the bandwidth requirement, t_{arr} is the arrival time, t_{hold} is the holding time, and $type$ indicates its type. If the request is for a static slice, bw is a scalar. Otherwise, bw is a list that contains all the potential bandwidth levels for an elastic slice. The request for an elastic slice also includes the distribution $dist$, which indicates the fraction of time that the slice is expected to be at each of the potential bandwidth levels. Slice scaling requests will identify the slice and the new bandwidth level required by the slice. Note that requests to reduce bandwidth are always satisfied, while requests to increase bandwidth are only rejected if insufficient resources are available.

The network operator gains revenue when it successfully allocates a slice request, but it must pay a penalty if it rejects a slice scaling request. The revenue of a static slice i is given by $R_{si} = r_s \cdot bw_i \cdot t_i$, where r_s is the revenue per unit bandwidth per unit time for a static slice, bw_i is the bandwidth of the slice, and t_i is the holding time of the slice. The revenue of an elastic slice i is given by $R_{ei} = r_e \cdot \sum_{b \in BW_i} b \cdot d_b \cdot t_i$, where r_e is the revenue per unit bandwidth per unit time for an elastic slice, BW_i is the set of bandwidth levels for elastic slice i, d_b is the fraction of time the slice is at bandwidth level b, and t_i is the holding time of the slice. For each slice scaling request that must be rejected due to insufficient resources, the network operator must pay a fixed penalty, R_p.

Thus, the network operator must decide whether or not to accept an incoming slice deployment request with the potential risk of blocking future scaling requests if too many slices are admitted. Furthermore, the network operator must select a route for an admitted slice request in a manner that attempts to minimize interference among slices that share common links in the network. Note that the network operator must also achieve the proper balance between static slices, which are expected to generate higher revenue per slice but consume more resources, and elastic slices, which generate less revenue per slice but allow for a greater number of slices.

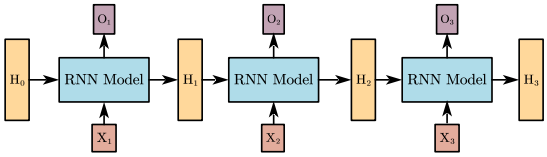


Fig. 2. RNN Procedure

C. Problem Formulation

We state the slice admission and routing problem as follows. Given 1) a network $G = (V, E)$ with bandwidths bw_e for $e \in E$, 2) a slice deployment request $r = (src, des, bw, t_{arr}, t_{hold}, type)$, 3) a set of potential paths, $P_{(s,d)}$, for slice r, and 4) a set of existing static and elastic slices in the network, along with their routes and bandwidth levels, the problem is to 1) decide whether or not to admit slice r, and 2) select a route, $p \in P_{(s,d)}$, for r if the slice is admitted. The objective of the problem is to maximize the long-term revenue for the network operator over time.

IV. Feature Selection, Model and Algorithm

In this section, we introduce the background of related machine learning concepts and discuss the selection of features from the environment, the model we use as an agent, and the RL algorithm we chose.

A. Background on related machine learning

A DNN model includes multiple layers where each layer consists of a weight matrix W and a bias vector b. The input of the DNN is a vector $X = (x_1, x_2, x_3...)$, and between two layers, there is an activation function. The objective is to make the output approach ground truth by tuning parameters of the activation function. Usually, the tuning process employs the gradient descent method.

A DNN's input is a fixed-size vector; however, many observations cannot be represented in a fixed-size vector, including graphs. A recurrent neural network (RNN) can solve this problem. As shown in Fig. 2, the input contain three vectors, (X_1, X_2, X_3), and there is an initial hidden state H_0. The RNN model concatenates of the previous hidden state H_{i-1} and input X_i, and it outputs vector O_i and hidden state H_i. After the loop ends, we obtain the final output O_3 as an encoding of (X_1, X_2, X_3). This procedure is independent of the input size, and the output contains all the information of (X_1, X_2, X_3) because each RNN model takes in a hidden state from the previous step.

In RL, three crucial elements are the environment, the agent with policy π_θ, the reward function. The agent chooses an action based on the environment and the policy. The environment returns a reward based on the action. In our problem, the agent attempts to maximize the profit of the network provider over time.

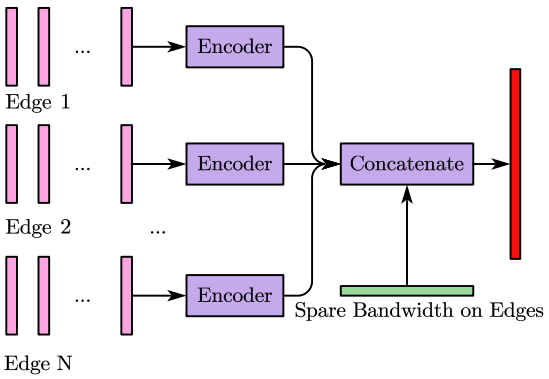


Fig. 3. Encoder Procedure

B. Feature Selection

A slice request follows the previously defined slice request type. Each slice request has a specific arrival time, service time, and type. We use tuple $R_i = (bandwidth_i, time_i, type_i)$ to represent the request i, where $bandwidth_i$ indicates the bandwidth of the slice required, $time_i$ indicates the service time, and $type_i$ is type of the slice. One-hot encoding is used to indicate the type of slice request. $type_i \in (1,0,0),(0,1,0),(0,0,1)$, corresponding static deployment slice type, elastic deployment slice type, and elastic scaling slice type, respectively.

The features of the RL model include two parts. The first part is the current slice request, which is encoded in a 1×5 vector, as described the above paragraph.

The second part encodes the environment, which includes the remaining bandwidth resource on each network edge and the set of slices that are currently in the network. We encode the remaining bandwidth resources on each edge using a vector of size $|E|$ because the number of edges is fixed. The slices currently deployed in the network are grouped by edge. However, since the number of slices on each edge varies over time, we employ an RNN to encode the information. For example, suppose there are three slices on an edge, and each slice is represented as a 1×5 vector. As shown in Fig. 2, these three vectors can be transformed into a fixed-size vector. Fig. 3 shows the entire encoding procedure.

C. DNN Model

We employ a simple flat layer in deep learning. For this simple graph, the model only contains 3 layers with 256 neurons on each layer. The activation function between each layer is $tanh$. The output of the model is an action distribution. For example, if we assign three paths to every source-destination pair, the action space is 4. Thus, the model needs to output a 1×4 vector, where the first element corresponds to the probability of rejection, and the remaining elements correspond to the probability of choosing the related path. Since we need the output to be a distribution, the softmax

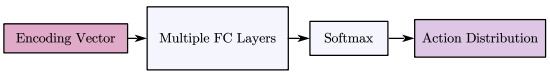


Fig. 4. Agent Model

function is used before the output. The softmax function is defined as follows:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1,\ldots,K \text{ and } \mathbf{z} = (z_1,\ldots,z_K) \in \mathbb{R}^K. \tag{1}$$

We illustrate our model in Fig. 5.

D. RL algorithm

There are two major categories of RL algorithms, policy-based algorithms, and value-based algorithms. This paper implements Proximal Policy Optimization (PPO), which is a policy-based algorithm. Before introducing PPO, we first briefly introduce the Policy Gradient algorithm, a classic algorithm in RL. We name the DNN model mentioned in the previous subsection as policy π_θ. The policy π_θ interacts with the environment to produce a trajectory τ: $[(s_1,a_1,r_1),(s_2,a_2,r_2),...(s_n,a_n,r_n)]$, where s_i is the environment encoding, a_i is $\pi(s_i)$, and r_i is the reward from the environment. The total reward for τ is given by:

$$R_\theta(\tau) = \sum_{t=1}^{T} r_t. \tag{2}$$

Sine the environment always has randomness, we usually need to collect more trajectories $\{\tau^2,\tau^3...\tau^N\}$. In this problem, we assume that each action has little affect on the following reward. Thus, instead of using the total reward, we assign $A_\theta(s_i,a_i)$ to each (s_i,a_i) pair, which is defined as follows:

$$A^{\theta^k}(s_t^n,a_t^n) = \sum_{t'=t}^{T_n} \gamma^{t'-t} r_{t'}^n, \tag{3}$$

where $\gamma \in [0,1]$ is the discount factor, determining how long the current action affects the reward. We take gradient of $A_\theta(s_i,a_i)$ to update θ. [4] describes the details to calculate the gradient.

One problem with policy gradient is the efficiency of trajectory usage. We can only use trajectories produced by π_θ once, since the original data is updated after the θ is out of date.

PPO employs importance sampling [5] to bypass this problem. We define two more values, $J^{\theta^k}(\theta)$ and $J_{PPO}^{\theta^k}(\theta)$

$$J^{\theta^k}(\theta) \approx \sum_{(s_t,a_t)} \frac{p_\theta(a_t \mid s_t)}{p_{\theta^k}(a_t \mid s_t)} A^{\theta^k}(s_t,a_t) \tag{4}$$

$$J_{PPO}^{\theta^k}(\theta) = J^{\theta^k}(\theta) - \beta KL\left(\theta,\theta^k\right). \tag{5}$$

The PPO algorithm is shown in Algorithm 1. We can reuse trajectories multiple times before the policy, which interacts with the environment, diverges too far from the training policy.

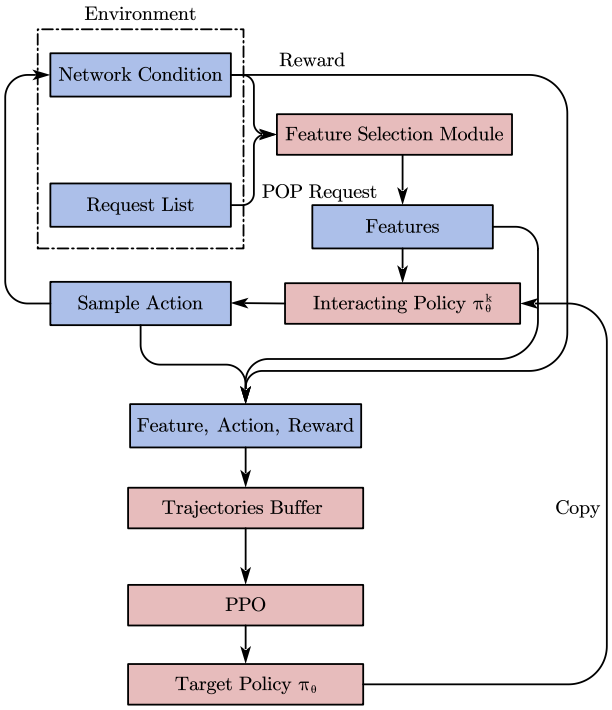


Fig. 5. Complete Procedure

Algorithm 1 PPO

Initialize the policy parameter θ^0, β, KL_{max} and KL_{min}
for each iteration **do**
  Using θ^k to interact with the environment to collect $\{s_t, a_t\}$and compute advantage $A^{\theta^k}(s_t^n, a_t^n)$
  Find θ optimizing $J_{PPO}^{\theta^k}$
  if $KL\left(\theta, \theta^k\right) > KL_{max}$ **then**
    increase β
  end if
  if $KL\left(\theta, \theta^k\right) < KL_{min}$ **then**
    decrease β
  end if
end for

While the policy π_{θ^k} interacts with the environment, we employ the exploration strategy, which samples actions from the output action distribution rather than select the action with the highest possibility. While this policy π_θ predicts the real situation, we apply an exploitation strategy.

We combine the elements, from feature selection to training strategies, into a complete procedure, as shown in Fig. 5.

V. Performance Evaluation

In this section, we describe the experimental setup, compare the RL algorithm's performance with the Shortest-Path-First (SPF) method, which always chooses the shortest available path, and we make some observations.

A. Reward Function

The reward Function is critical in reinforcement learning and relatively subjective, since, in most scenarios, we cannot define an exact reward for each action. In this paper, the basic format of the reward function is shown as follows:

$$reward = \begin{cases} bw \times time, & \text{if accepted} \\ 0, & \text{otherwise} \end{cases}, \tag{6}$$

where bw is the bandwidth required by the slice request and $time$ is the service time of the slice.

Besides the primary reward, we consider four additional factors that may affect the reward. First when the network operator accepts a request for the deployment of a new elastic slice, it should also be expected to accept future scaling requests for this slice, as long as such requests fall within the bandwidth profile agreed upon between the network operator and the slice owner. If the operator rejects a scaling request due to insufficient resources, there should be a penalty for the network operator, since it would be considered a violation of the agreement between the network operator and the slice owner when the elastic slice request was accepted. Second, the penalty for rejecting a scaling request should be eliminated if the bandwidth requested by the slice exceeds the amount indicated by the slice's bandwidth distribution profile. For example, if a slice's distribution $dist$ is $[0.8, 0.2]$, and its bandwidth bw is $[2, 8]$ but during service, the slice's actual distribution $dist$ is $[0.5, 0.5]$ based on the history of this user, then the user is using more bandwidth than was indicated in the initial request. When rejecting such scaling requests, the punishment should be lower. We use a variant of KL-divergence to indicate the difference between the reported $dist$ and actual $dist$. KL-divergence is defined as follow:

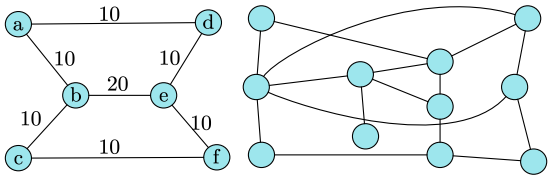
$$D_{\mathrm{KL}}(P\|Q) = \sum_{x\in\mathcal{X}} P(x)\log\left(\frac{P(x)}{Q(x)}\right) \tag{7}$$

where the P and Q are two distributions and $\mathcal{X}$ is the set of potential values of random variable x. In this example, P is the reported distribution, and Q is the actual distribution. We calculate the KL-divergence for the example above as follows:

$$D_{\mathrm{KL}} = 0.2 \times \log\left(\frac{0.2}{0.5}\right) + 0.8 \times \log\left(\frac{0.8}{0.5}\right) \approx 0.193. \tag{8}$$

If two distributions are identical, then the D_{KL} is 0. D_{KL} increases with the difference between two distributions. However, we wish the variant of the KL-divergence to be 1 when the reported distribution and the actual distribution are identical. Since the slice's behavior is valid in this scenario, the punishment should be full. On the other hand, when these two distributions diverge significantly, the variant of the KL-divergence should be a small number. Thus, we define the variant as $V_{KL} = \exp(-D_{KL})$. When a penalty is incurred, it is multiplied by V_{KL}.

The third factor is that, for an elastic request, the revenue should be higher than a static request when the average required bandwidth is the same, due to the increased network complexity associated with providing elastic slices. For example, if an elastic request's reported distribution is $[0.8, 0.2]$, with bandwidth levels $[2, 8]$, then the average bandwidth is $2 \times 0.8 + 8 \times 0.2 = 3.2$. The cost of such an elastic request



(a) Simple Topology (b) Complex Topology

Fig. 6. Experiment Topologies

should be relatively higher compared to a static request with 3.2 units of bandwidth. Thus, we multiply the reward with a coefficient e.

The fourth factor is that different paths may have different rewards. For example, paths with fewer nodes should yield greater revenue since less resources are used. In this paper, we introduce a coefficient $p = 0.9^{n-2}$, where n is the number of nodes in the path. For example, for source-destination pair (a, d) in Fig. 6 (a) , there are two path, $a \rightarrow d$ and $a \rightarrow b \rightarrow e \rightarrow d$. The coefficient p of the first path is 1, and the coefficient of the second path p is 0.9^2.

Overall, the reward function for a static slice request $reward_{SS}$ is shown as follows:

$$reward_{SS}(req) = \begin{cases} bw \times time \times p, & \text{if accepted} \\ 0, & \text{otherwise} \end{cases} . \quad (9)$$

The reward function for a elastic slice initial request $reward_{ESI}$ is shown as follows:

$$reward_{ESI}(req) = \begin{cases} bw \times time \times e \times p, & \text{if accepted} \\ 0, & \text{otherwise} \end{cases} . \quad (10)$$

The reward function for a elastic slice scaling request $reward_{ESS}$ is shown as follows:

$$reward_{ESS}(req) = \begin{cases} bw \times time \times e \times p, & \text{if accepted} \\ -bw \times time \times e \times V_{KL}, & \text{otherwise} \end{cases} . \quad (11)$$

B. Environment Setup

1) Network Topology: We set up two scenarios, the first scenario is a simple graph with only 6 nodes, shown as Fig. 6 (a), and the second scenario is a more complex graph, which shares the same structures as the Sprint backbone network, as shown in Fig. 6 (b).

For simplicity, we primarily focus on the experiments and results from the simple graph environment, and we verify the concepts on the complex graph to prove that algorithm is robust.

2) Request Type: We assume slice requests arrive to the network according to Poisson process with parameter λ and that the service time for a request follows an exponential distribution with parameter μ. We list the request types used in the simple graph in Table. I. Similar request types are used in the complex graph scenario.

3) Path: We assign three paths for each source-destination pair and summarize these paths in Table. II. The last column is path coefficient for each path.

TABLE I
REQUEST TYPE

src, des	ID	bw	λ	μ	$dist$
a,d	0	[6]	0.75	0.5	[1]
	1	[9]	1.5	1	[1]
	2	[4,9]	3	2	[0.8,0.2]
c,f	0	[5]	1.5	1	[1]
	1	[9]	0.75	0.5	[1]
	2	[5,8]	3	2	[0.70.3]
b,e	0	[4]	0.75	0.5	[1]
	1.5	[12]	1	1	[1]
	2	[3,13]	3	2	[0.9,0.1]

TABLE II
PREDEFINED PATH

s, d	path	p	s, d	path	p	s,d	path	p
a,d	a,d	1	c,f	c,f	1	b,e	b,e	1
	a,b,e,d	0.81		c,b,e,f	0.81		b,a,d,e	0.81
	a,b,c,f,e,d	0.66		c,b,a,d,e,f	0.66		b,c,f,e	0.81

C. Training Phase

We set the trajectory time to 20 units in training, while we set the trajectory time for the test to 200. The model π_θ's performance converged after 1000 iterations. Because the model is relatively shallow, the total training time is short. Fig. 7 shows the total evaluation reward change during training.

D. Result and Observation on Simple Graph

Tables III and IV show the reward values and the slice acceptance rates of static requests (SRAR) and elastic requests (ERAR) for different discount factors and baseline values.

1) Discount Factor: While tuning the hyper-parameters, we noticed that the discount factor γ plays a critical role in performance. The discount factor is usually set to 0.9 to 0.99 in common RL scenarios; however, in our situation, the agent does not have a specific endpoint. The endpoint is actually set up by the total trajectory time. If we set the γ as high as 0.9 or 0.99, as shown in Eqn. 3, the state s and action a occur earlier in a trajectory, corresponding to a larger accumulated reward and a greater effect on θ updating. To alleviate this problem, we set the γ to a relatively low value. As observed in Table III, when $\gamma = 0.3$, the model has the best performance.

2) Baseline: As shown in Eqn. 3, we notice that the reward is always a non-negative number. This situation leads to a decrease in the probability that actions are not sampled. An action that is not sampled could be a potential promising action. Thus, not sampling the action causes a loss of performance. We set a very small, non-negative baseline bs value and calculate the advantage value as follows:

$$A^{\theta^k}(s_t^n, a_t^n) = \sum_{t'=t}^{T_n} \gamma^{t'-t} r_{t'}^n - bs. \quad (12)$$

We show the improvement of setting the baseline to 2 in Table III.

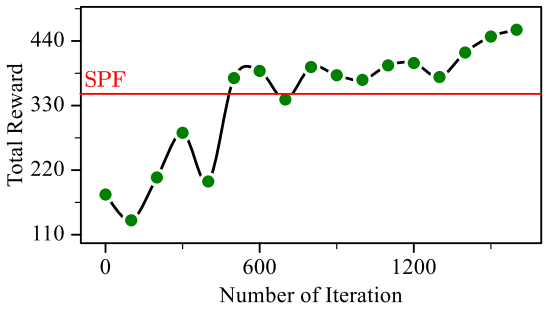


Fig. 7. Training Phase

TABLE III
DISCOUNT FACTOR AND BASELINE TUNING

	Baseline = 0			Baseline = 2		
γ	Reward	SRAR	ERAR	reward	SRAR	ERAR
SPF	327.344	0.435	0.408	327.344	0.435	0.408
1.0	278.42	0.432	0.307	450.895	0.538	0.491
0.9	437.705	0.543	0.508	400.366	0.483	0.558
0.8	435.170	0.551	0.581	447.692	0.538	0.650
0.7	422.184	0.538	0.543	473.358	0.451	0.650
0.6	416.190	0.527	0.593	451.766	0.522	0.594
0.5	456.955	0.561	0.576	473.782	0.556	0.598
0.4	466.268	0.554	0.623	469.049	0.557	0.645
0.3	**483.348**	**0.608**	**0.651**	**502.634**	**0.609**	**0.652**
0.2	437.182	0.525	0.499	472.070	0.609	0.635

3) Encoder Selection: The vanilla RNN tends to vanishing or exploding its gradient. There are two common tricks to alleviate this problem. The first is gradient clipping. As the name says, we can clip the gradient if the gradient exceeds a given threshold. The second solution is to use a gated structure RNN, such as Long Short-Term Memory Networks (LSTM) or Gated recurrent unit (GRU) [6]. We choose GRU in this experiment, which is more lightweight compared to LSTM and has a similar performance with LSTM.

4) Safety Boundary: It is possible that the agent may choose to accept a request, even though there is not enough bandwidth for this request. We will check the remaining bandwidth before allocating resources according to the agent's decision to prevent this situation. If the remaining resource cannot fulfill the request, we reject the request.

E. Result on Complex Graph

Because the input size increases linearly with the number of graph edges, for this graph, we choose deeper and wider DNN as an agent, which contains 6 fully connected layers, with the widest layer containing 1024 neurons. The training time grows accordingly. The result is shown in Table IV. Overall, compared with SPF, the RL agent has better performance with suitable parameters.

In our RL approach, the agent could adjust its behavior without saving history and gain more revenue over a relatively long period of time. We don't provide the RL agent with the type of requests; thus, the algorithm is robust. Besides RL, we implement an Auto Encoder to encode the network environment. With the Auto Encoder's help, we convert variable-sized environments into a fixed-sized vector. Another benefit of the Auto Encoder is that we can train the encoder with the RL agent and provide an end-to-end experience.

TABLE IV
RESULT IN COMPLEX GRAPH

	Baseline = 0			Baseline = 2		
γ	Reward	SRAR	ERAR	reward	SRAR	ERAR
SPF	482.23	0.613	0.355	482.23	0.613	0.355
0.5	585.72	0.545	0.459	660.13	0.564	0.575
0.4	**621.47**	**0.543**	**0.512**	**754.97**	**0.518**	**0.597**
0.3	543.89	0.459	0.362	623.66	0.502	0.538

VI. CONCLUSION

The primary contributions of this works are (i) defining a framework for virtualizing slice requests and network environments, (ii) designing an RL agent and employing a Proximal Policy Optimization algorithm to train the agent to route slice requests, and (iii) designing a pricing strategy to prevent a user from abusing elastic policy and to help to allocate resource efficiently.

The model is tested in both simple and complex network topologies and outperforms heuristic methods. The algorithm only requires a small amount of data to train a satisfied model. However, we still need to tune two hyper-parameters, discount factor and baseline, in addition to the normal hyper-parameters in DNN, such as the learning rate. In the future, we wish to find a better way to decide the baseline parameter. Another problem of this algorithm is that the model's complexity grows at least linearly with the size of edges in the network graph. We wish to have a more reliable scalable performance by employing a more advanced encoder, such as GNN.

VII. ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation under Grant No. CNS-2008856.

REFERENCES

[1] Z. Wu, G. Ishigaki, R. Gour, and J. P. Jue, "A reinforcement learning-based admission control strategy for elastic network slices," in *2021 IEEE Global Communications Conference (GLOBECOM)*, pp. 01–06, IEEE, 2021.

[2] J. Jia, L. Yang, and J. Cao, "Reliability-aware dynamic service chain scheduling in 5g networks based on reinforcement learning," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pp. 1–10, 2021.

[3] C. Liu, M. Xu, Y. Yang, and N. Geng, "Drl-or: Deep reinforcement learning-based online routing for multi-type service requirements," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pp. 1–10, IEEE, 2021.

[4] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*, pp. 387–395, PMLR, 2014.

[5] T. Kloek and H. K. Van Dijk, "Bayesian estimates of equation system parameters: an application of integration by monte carlo," *Econometrica: Journal of the Econometric Society*, pp. 1–19, 1978.

[6] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.