Policy Controlled Multi-domain cloud-network Slice Orchestration Strategy based on Reinforcement Learning

Asma Islam Swapna*, Raphael Vicente Rosa*, Christian Esteve Rothenberg*, Rafael Pasquini†, Javier Baliosian‡
*University of Campinas (UNICAMP), Sao Paulo, Brazil
†Federal University of Uberlândia (UFU), Brazil
‡University of the Republic, Uruguay

***Abstract*—The concept of network slicing plays a thriving role as 5G rolls out business models vouched by different stakeholders. The dynamic and variable characterization of end-to-end cloud-network slices encompasses the composition of different slice parts laying at different administrative domains. Following a profit-maximizing Slice-as-a-Service (SaaS) model, such a multi-domain facet offers promising business opportunities in support of diverse vertical industries, rendering to network slicing marketplace members the roles of Infrastructure Provider, Slice Provider, and Tenants. The effective realization of SaaS approaches introduces a dynamic resource allocation problem, manifested as challenging run-time decisions upon on-demand slice part requests. The Orchestrator is hence responsible to perform an optimized decision on-the-fly on which elasticity requests to address based on an orchestration policy defined within the context of Network Slice architecture for the followed revenue model. This paper presents a slice management strategy for such an orchestrator can follow, based on reinforcement learning, able to efficiently orchestrate slice elasticity requests to comprehend the maximum revenue for the stakeholders of end-to-end network slice lifecycle. The proposed strategy orients a Slice Orchestrator to learn which slice requests to address as per availability of the required resources at the different participating Infrastructure Providers. The experimental results show the Reinforcement Learning based Orchestrator outperforms several benchmark heuristics focused on revenue maximization.**

I. INTRODUCTION

The telecommunications and the cloud businesses are evolving towards offering a richer set of services beyond basic connectivity and machine virtualization, taking advantage of network programmability and service virtualization as core tenets behind 5G network slicing [1]. In this context, a versatile execution environment is required, capable of running different workloads at different times and scales, while minimizing resource over-provisioning. This adaptability is known as elasticity, and many approaches to perform it has been proposed, for example deep-neural networks (DNN) and reinforcement learning (RL) [2] [3]. Those are especially useful to control systems that are heterogeneously complex to identify and model, in particular cloud-network systems where elasticity needs to follow over-the-top service dynamics.

Virtualized cloud-network environments are embracing the slicing concept [4] to define an arbitrary amount of logically independent network partitions, each comprising different resources and functions, which are interconnected and involved in the delivery and the operation of a specific service. By instantiating these so-called cloud-network slices, an operator is able to dynamically provide and operate different services over shared infrastructures. Each slice behaves and appears as a fully-functional and isolated instance, despite actually running over the same physical infrastructure. When it comes to slice creation spanning shared infrastructures, the on-demand allocation and scaling of resources becomes a critical decision.

With the architectural vision to realize cloud-network slicing, the NECOS project [5], [6] upholds such slice elasticity concept as one of the critical success factors. This work references NECOS as a candidate platform to exercise a Reinforcement Learning-based solution for the management of resources in an end-to-end (E2E) cloud-network slicing environment. As well as for other authors [7], in the NECOS project, elasticity comprises the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning physical and virtual resources (computing, networking and storage) in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible. Furthermore, the proposed elasticity features consider that the slice is provisioned in a multi-domain and multi-technological environment, and the run-time change of resources as such requires a sophisticated mechanism of orchestration.

In the scope of our resource allocation policy, the slice elasticity control is extended to cloud providers along with a proposition for new elasticity flavor conceptualized through the efficient allocation of unused resources from other slices [8]. To satisfy the slice resource requests, the Slice Provider has an Orchestrator module that allocates compute, bandwidth, and storage resources to the slice simultaneously. Implementing the resource allocation policies of a NECOS-like environment is a challenging task for the Orchestrator for reasons such as:

i Unknown request arrival process and resource requirements from users or the triggering monitoring abstractions.
ii Heterogeneity in slice resource capabilities (i.e., CPU, memory, bandwidth, etc.) and notoriously hard to optimize performance metrics, i.e., service level agreements (SLAs).
iii Dynamic and slice run-time decisions of triggered elastic slice part requests on finite resource capacities (e.g., horizontal and/or vertical scaling).

In this work, we try to answer the following question: *can an orchestrator benefit from Reinforcement Learning to* *control the elasticity of a cloud-network slice*?

Application of machine learning in decision-making domains is being increasingly explored in [9], [10], and [11]. In most cases, RL has been stretched for further studies in addressing the decision-making problems of complex network systems [12], [13], [14], [15]. Reinforced learning emphasis on agents learning to make better decisions by interacting directly with the environment through a goal-oriented approach. The agent stays on-line with the environment knowing nothing about the task and learns by reinforcement of rewards it gets upon each performed task.

Our research hypothesis states that, behaving as an RL agent, a cloud-network Slicing Orchestrator can learn to address elasticity requests to take slice provisioning decisions by, first, learning over simulated and synthetic workload data and, then, applying those learned policies in real-time deployments. As the functional challenges mentioned earlier prevails, as a first step, this work designs and evaluates a Orchestrator to perform efficient scheduling of Slice Resource Requests. The proposed RL-Orchestrator operates on a simulated on-line environment where the resource requests, or jobs, arrive dynamically and therefore are cleared/scheduled by the orchestrator to achieve certain desired objectives. In this case, the RL-Orchestrator focused on learning to optimize the completion time for each job and minimize the slowdown rate.

We advocate that an RL-Orchestrator approach must operate under the umbrella of a policy-based decision-making engine. The sole trust in RL-based decisions can potentially harm cloud-network slice SLAs. DNN and RL approaches, in spite of detaining high accuracy and efficiency still lack behind on transparency. A critical component such a cloud-network slice orchestrator indeed must determine clear mechanisms to debug, trace, and explain its decisions. When adopting a solution based on RL or DNN, such concerns are still open research issues within the machine learning community. Therefore, we determine a policy-based decision-making engine that must be responsible for taking the RL-Orchestrator scheduling as a recommendation source, set proper adjustments on them, and transparently decide upon stable SLAs.

Section II presents the background about cloud-network slicing and RL Policy parameters, while Section III describes the model design and learning algorithms. We designed a simulation environment, described in Section IV, to perform the experiments on RL learning performance over time and the comparison with standards heuristics to achieve maximum revenues depending on prioritized stakeholders. Section V discusses the main limitations of this proposal and Section VI concludes this paper with some future work directions.

II. Background

This section provides a primer on the reference architecture of NECOS and presents the adopted Cloud-Network Slice (CNS) concept along with the reference cloud environment and Reinforcement Learning (RL) techniques.

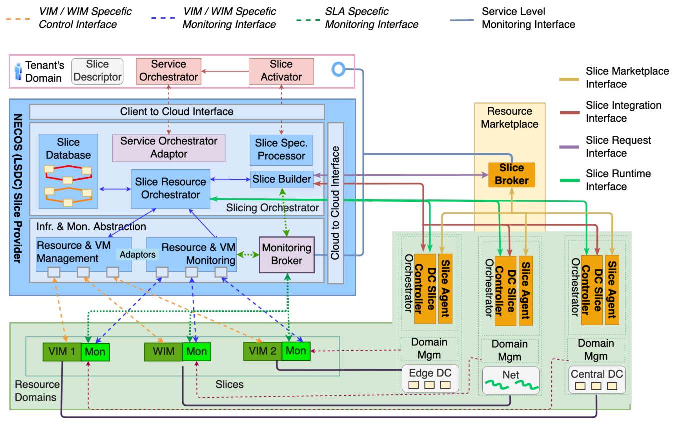

Fig. 1. NECOS platform architecture for multi-domain cloud-network slicing.

A. *NECOS*

1) *Architecture:* The NECOS architecture depicted in Figure 1 has four subsystems: Tenant's Domain, Slice Provider (SP), Marketplace, and Infrastructure Providers (InPs). All these subsystems support the proposed Slice-as-a-Service paradigm of NECOS, offering on-demand slices to tenants. Figure 1 depicts a subset of the NECOS Architecture, in which the four subsystems along with the modules for slice creation are highlighted. The full architectural description, with all modules can be found in [16]. In this architecture, in order to have a slice, the tenant provides a description of a slice request, which, among other aspects, may include QoS goals to an SP, the entrance domain (cloud or network operator). The Marketplace is a subsystem in which all domains, part of a NECOS federation, can advertise the resources they can provide. The Slice Provider consults the marketplace to find the options available in the federation to create the requested slices. The InPs, participating domains in a given NECOS federation, offer resources, called *slice parts*, to compose slices under the slice-as-a-service paradigm proposed in the NECOS project. Examples of Infrastructure Providers include network operators, cloud and edge providers.

2) *Cloud-Network Slice (CNS):* A CNS is defined as an independent E2E logical network running on a shared infrastructure (*i.e.,* compute, storage, connectivity resources) capable of providing a negotiable service quality agreed among its consumers and providers [1]. Slices can be composed of virtual components (*e.g.,* Virtualized Network Functions (VNFs), Virtual Machines (VM), containers, and virtual links) and/or infrastructure resources (*e.g.,* CPUs, routers, tunnels), and connectivity services spanning multiple administrative domains at the same time and deployed across different networking settings (*i.e.,* access, transport, core). The concept of cloud-network slicing conveys the approach to provide isolated slice parts offered by diverse InPs based on specific E2E service requirements. To feed the E2E flow with slice requirements, the Orchestrator performs elastic resource allocation to the creation of the new slices or the running slice. The design of efficient elasticity control mechanisms for dynamic resource allocation is crucial to increase the efficiency of cloud-based

slice-defined networked systems.

3) Slice Resource Orchestrator (SRO): The NECOS SRO component [17] is responsible for combining the slice parts that make up an E2E cloud-network slice, orchestrating slices and service elements over slices parts. Slice builder takes part in slice creation and tear-down process while the monitoring entity audits the resource usage and confirms the final tear-down of the slice or deactivation of slice parts. Hence, apart from provisioning and decommissioning the E2E slices, the main functions of the SRO are: *(a)* instantiating virtual resources for the services, which refers to the allocation of the virtual resources of each selected slice, and *(b)* performing the slice Elasticity. In a real-time multi-domain CNS environment, the orchestrator directly deals with the decision of which request to address and when, from a vast pool of requests, mostly dependent on resource availability at InPs.

4) Elasticity: The elasticity architecture proposed in [8] guarantees the performance of the cloud-network slices as much as possible through reactive decision-triggering while maintaining committed levels of SLA and application performance. Nonetheless, the proposed distributed architecture simultaneously acts in different cloud and network providers, in order to apply elasticity control functions for readjusting (scale up and down) resources at CPU, RAM, and network levels to each of the activated cloud-network slices. Therefore, the orchestrator faces another decision-making problem – scheduling the slice part requests, which we classify as a slice management problem and the scope of this research as well.

5) Slice-as-a-Service (SaaS) Profit Maximization: The NECOS system enables resource trading among different actors for the dynamic provision of slices following a Slice-as-a-Service (SaaS) model, where the primary stakeholders are *(i)* Tenants – slice users; *(ii)* Slice Provider – cloud-network operators and service providers; *(iii)* Marketplace Providers – service providers with cloud-network asset trading platforms; and *(iv)* Resource Providers – mainly the InPs. A consolidated business model following a cost-based analysis for E2E slice provisioning is required to maximize all profits.

Figure 2 presents a generic view of the cost model to better portrait the revenue architecture of cloud-network slicing in NECOS. If analyzed, the depicted figure conveys the more resource requests the Slice Provider can handle, the higher the profits. Alongside this, faster E2E provisioning adds up to the profit surplus for a SP. In this scenario, the profit of an InP can be maximized by: (*i*) accepting as many slice requests as possible, and (*ii*) matching the elastic dynamic request of the slice requirements as closely as possible, for instance avoiding penalties of performance degradation caused when slice elasticity requests are not satisfied up due to resource constraints. Since the SP is dependent on the SRO's performance of processing tenants' new slice requests, InPs profit can be controlled by processing the elasticity triggers from slice monitoring VIM/WIM residing at InPs. Therefore, the SRO falls in the heuristic dilemma of processing two ways inward slice part requests containing consolidated workload.

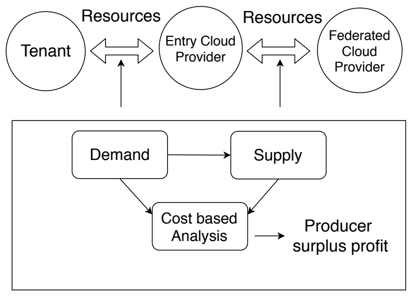

Fig. 2. NECOS cost based business model.

B. Reinforcement Learning

In a generic setup, an RL agent interacts with the environment and learns from the observation. Consider an RL model, where at each time step t, the agent observes an environment state s_t. For an unforeseen scenario, the agent (e.g., inside an Orchestrator embodiment) chooses an action a_t and in the next state s_{t+1} receives a reward r_t for such action, assuming the state transition and rewards have stochastic Markov property. Without prior knowledge of the current state, the agent learns to make a decision by interacting with the environment, unaware of the rewarding actions. During the training of this learning process, the goal consists in maximizing the expected cumulative discounted reward:

$$R_t = \mathbb{E}[\sum \gamma^t, r_t], for \gamma \in [0,1] \tag{1}$$

where, γ weighs the possible immediate future reward with possibilities of better rewarding actions for the agent.

In the preliminary proposition of RL [18] , an agent decides an action based on a learned *policy*, defined in the literature as a probabilistic distribution over the plausible actions for the occurring state. π: *π(s,a)* $\rightarrow$ [0,1], where *π(s,a)* denotes the probability of taking action *a* in state *s*. For the decision problem in consideration of this work, the action, state pairs are exponential as described in Section *III*. Appropriate function approximators are applied to store scalable non-tabular pairs [19]. In the learning process, typically, policy approximators learn to cluster the behavior of similar states such that at encountering a new state the approximator can find the action for the closest state experienced so far. In the literature, the exponential use of DNN combined with RL as a function approximator [20] has influenced this work to incorporate a similar policy.

Our focused policy learning algorithms fall in the RL class that learns by performing *gradient-descent* on policy parameters, the reward. The key idea of the policy gradient-descent method is to estimate the gradient of the method by observing the trajectory of the agent-following policy execution. The objective lies in maximizing the cumulative rewards in equation *(1)*, which presents:

$$\nabla_\theta[\sum_{t=0}^{\infty} \gamma^t r_t] = \mathbb{E}_{\pi\theta}[\nabla_\theta log\pi_\theta(s,a) Q^{\pi\theta}(s,a)], \tag{2}$$

Here, $Q^{\pi\theta}$ is the expected cumulative discounted reward for choosing a deterministic action a in a state s. If followed

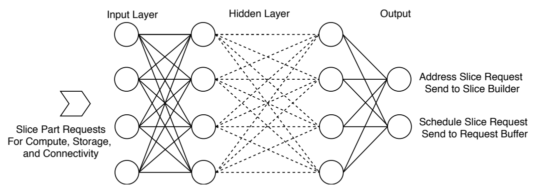
Input Layer
Hidden Layer
Output
Slice Part Requests For Compute, Storage, and Connectivity
Address Slice Request Send to Slice Builder
Schedule Slice Request Send to Request Buffer

Fig. 3. Stochastic Policy-based Deep Neural Network architecture feeding the orchestrator.

a simple form of Monte Carlo Method [21], agent samples multiple trajectories to get an unbiased estimate of $Q^{\pi\theta}$, v_t and updates the policy parameters:

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta log\pi_\theta(s,a)v_t, \tag{3}$$

Here, α is the gradient ascent step size. This is the positioned Reinforce algorithm in the literature [19] and thus used in this work, which is later expressed in pseudo-code learning algorithms, such as Q-learning, temporal difference learning, etc., can solve general Markov Decision Problems to converge to optimal policies. However, the decision to choose each of them vastly depends on the action space over the environment.

We provide Table I with accompanying discourse factor for heuristics behind preferring policy-gradient over Q-learning given the slice management environment in consideration.

III. Model Design

Our reinforcement learning-based stochastic policy network for the designed orchestration model is represented in Figure 3 with the formation of deep neural networks. The agent is embedded into a slice management loop running at the orchestrator and composed of two parts illustrated in Figure 4. The inner loop includes request, slice provisioning, and reward computation, whereas the outer loop contains elasticity triggering space to feed the orchestrator with elastic slice part requests, and merges with inner loop until a reward computation takes place again. To design a learning model following appropriate policies for the scheduling of multi-domain slice part requests with RL, we formulate our problem in the section below and present an RL model in the subsections.

A. Environment Model

We formulate the types of the slice part request with reference to NECOS resource variations (e.g., CPU, memory, connectivity-bandwidth). Slice part requests arrive at the orchestrator in an online fashion with discrete time steps. The requests can be created by tenants or triggered by NECOS monitoring abstractions. Considering the previous studies [22], we assume that the resource demand for each job is known upon arrival. We specify the requested profile for each job by a given vector $r_j = (r_{j,1}, ..., r_{j,d})$, where d is the demanded resource type for the slice part. Within the scope of NECOS, the resources are allocated to the representative slice as per SLAs reflecting the completion time of the job or slice request to the orchestrator. However, scheduling the slice requests to allocate the demanded resources upon availability is as we represent – an optimization problem.

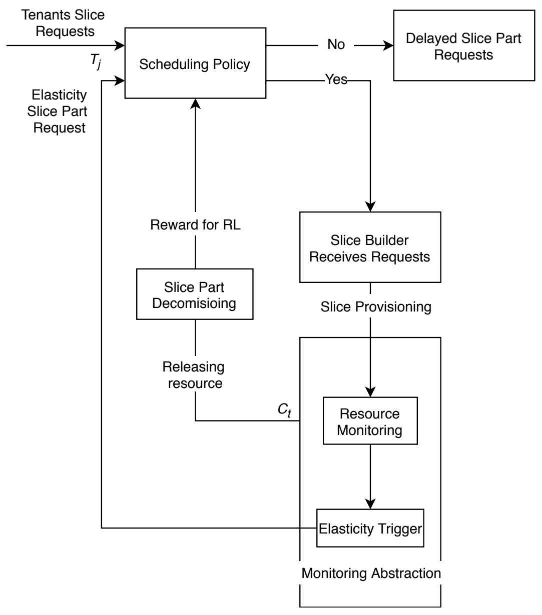

Fig. 4. Reinforcement Learning-based slice management loop feeding the orchestrator.

Objectives: maximize the performance of slice management by the Orchestrator and use the *average job slowdown* as a primary objective calculated as $S_j = C_j/T_j$, where C_j is the completion time of the job and T_j is the duration of the job or lock time for that resource in the scenario under consideration; within NECOS consists of the time from the request arrival at orchestrator until the tear-down confirmation.

B. RL Formulation

We solve the above optimization problem with a RL algorithm. Table II presents the state, action, and reward of the RL formulation.

1) State Space: The state space model, if represented in blocks of resource requests, can be illustrated as in Figure 5, where different colors represent different request types, tenants, and elasticity. For the sake of experiments we designed a fixed state representation. The *backlog* component defines any request that is in waiting state to be processed by the orchestrator for a given number of requests at arrival. The intelligent orchestrator designed learns to avoid penalties and to maximize the profit of the stakeholders.

2) Action Space: The scheduler may admit any subset of requests upon arrival; the designed RL can handle a large action space, however we keep the action space small and straightforward, allowing the agent to execute multiple actions at a given time step. *a='void'* for time-steps when the orchestrator does not perform scheduling tasks. The agent either allocates the blocks of resources to the requests and shifts the current block or stays in a similar block. Decoupling the decision sequence from time-steps, we allow the agent to schedule multiple jobs at the same time-step while keeping the action space linear.

3) Reward: The design moves the reward signal to guide the agent towards making a better decision over time for the

TABLE I

DISCOURSE FACTORS OF USING POLICY GRADIENT OVER Q-LEARNING TO ADDRESS REQUEST SCHEDULING DECISION PROBLEM OF ORCHESTRATOR IN CLOUD-NETWORK SLICE MANAGEMENT.

Factor	**Policy Gradient-descent**	**Q-Learning**	**Discourse**
Action Space	Policy-based model can solve problems with continuous action space	Q-learning finds a maximum value from a discrete set of actions	Orchestrator deals with a large continuous action space for request scheduling.
Action Selection	Learns a stochastic action map from action space	Learns to take a deterministic action from a discrete set of actions	Continuous inward slice requests to Orchestrator ask for stochastic decision map to avoid future penalties
Policy Optimization	Performs stochastic policy optimization in the learning process through direct policy searching	Learns from value function to maximize the reward.	Orchestrator needs a map to refer its next action on the scheduling which can be performed through optimized learning policy and speed up through value-based learning model at the same time.

TABLE II

STATE, ACTION, AND REWARD TO FORMULATE RL FOR ADDRESSING REQUEST SCHEDULING DECISION PROBLEM OF ORCHESTRATOR IN CLOUD-NETWORK SLICE MANAGEMENT.

State	Number of arrived slice parts request within a specific time window and resource profiles of the job requests waiting to be scheduled by Orchestrator
Action	Slice part requests sent to the slice builder for further processing, i.e. contact Marketplace for slice resources to complete E2E slice composition
Reward	Weighted sum of average delays in processing requests (Slowdown)

designed objective – minimize the average slowdown time. The reward calculation for each time-step is as follows,

$$\sum_{j\in J}\frac{-1}{T_j}, \tag{4}$$

where *J* is the number of requests currently in the system. The Orchestrator receives the reward of the action on the current state upon completion of the request, and no reward will occur for intermediate decisions within a given time-step. Minimizing delays or slowdown maximizes the reward.

4) Training Algorithms: Using the REINFORCE algorithm and policy gradient stated in equation (2), we execute our training procedure. However, a slight modification is performed in adoption to match the training algorithm with the cloud-network slice orchestration scenario. To reduce the high variance policy gradient we introduced a baseline which is calculated following a popular training algorithm [23] for resource management problems. In this exercise,we compute the baseline b_t by calculating the average mean of return v_t and subtracting the baseline from policy divergence. This changes of computation in the algorithm should reduce the gradient-descent value to a presentable figure for further evaluation. The RL training stages with such design modification in it's computation follow the steps as presented in Algorithm 1.

Algorithm 1: REINFORCE training algorithm with baseline subtraction. [24]

```
Initialize $b$;
for each iteration do
    $\nabla\theta \leftarrow 0$;
    for each requests do
        for each episodes do
            compute return $v_t$;
            for each time-steps do
                compute baseline $b_t$;
                if resource request > availability then
                    put requests in buffer;
                else
                    $\nabla\theta \leftarrow \nabla\theta + \alpha\nabla_\theta \log\pi_\theta(s_t^i, a_t^i)(v_t^i - b_t^i)$;
```

IV. EVALUATION

The described model evaluation is performed using a reference environment for DNN-based RL experiments as proposed in [12]. For the purpose of event-driven simulations, we modified the environment and policy-gradients, and other comparable agents to use as benchmarking heuristics.

A. Experiments

Mincing the model design described in Section III, the average request arrival rate is chosen for different workloads varying between 10-100 percent forming a Poisson distribution. We prototype the DNN with 20 neurons in it's hidden layers with a total of 49,746 parameters performing 1000 iterations with 100 experiments over 25 different requests sets in parallel during training. The test results are averaged over 200 seeds. The constant learning rate throughout the experiments is 0.001. For simulation purposes, we designed elasticity requests conforming to fewer resource demands than Tenant's slice part request profiles.

B. Performance Comparison

The Orchestrator's performance benchmark scheme is focused on maximizing the revenues coming from the increasing order of performing requests. Resource requests from elasticity triggers, if cleared for resource allocation by slice builder, give profit to the InPs, while Tenant's slice requests with less slowdown ensure higher revenue for the slice providers. The performance of our RL approach was compared against three benchmarks: *(i)* RevInP - in which the Orchestrator chooses the Elasticity request to maximize the InP revenue; *(ii)* RevSP - in which the orchestrator serves tenant's slice requests over elasticity triggers; and *(iii)* Random - in which the orchestrator randomly serves any of the slice resource requests, regardless of the arrival time. To observe the the RL policy convergence, the total reward at each training iteration is shown as well.

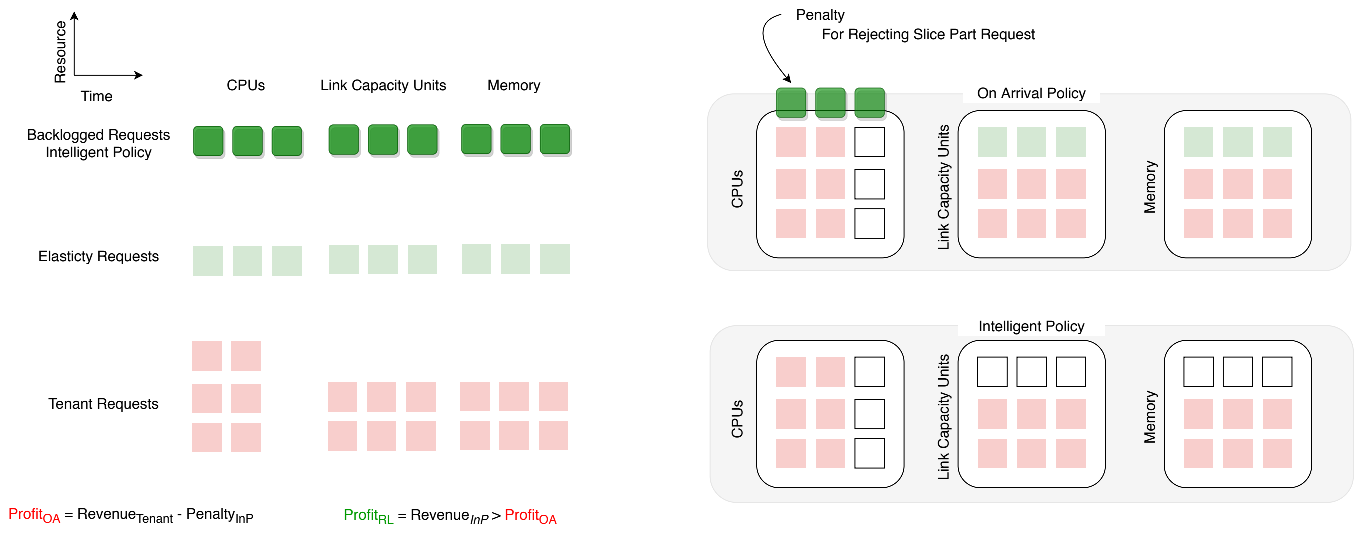
Resource
Time
CPUs
Link Capacity Units
Memory
Backlogged Requests Intelligent Policy
Elasticty Requests
Tenant Requests
$Profit_{OA}$ = $Revenue_{Tenant}$ - $Penalty_{InP}$
$Profit_{RL}$ = $Revenue_{InP}$ > $Profit_{OA}$
Penalty
For Rejecting Slice Part Request
On Arrival Policy
Intelligent Policy

Fig. 5. State representation with different requests types and profit heuristics.

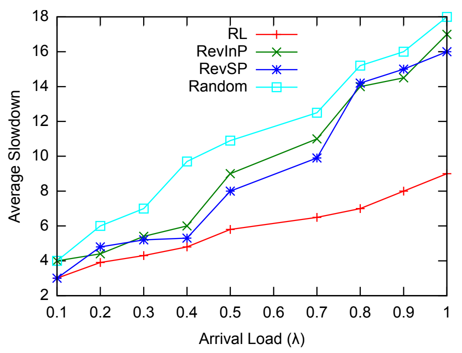

Fig. 6. Performance of DNN-based RL orchestrator upon varying workloads.

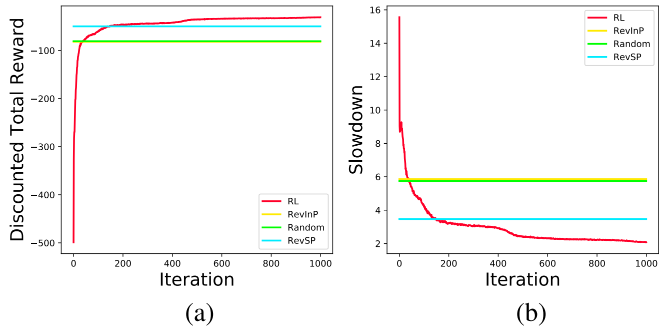

Fig. 7. Learning curve of DNN-based RL Slice Orchestrator showing improved request scheduling for incoming slice part requests.

Figure 6 presents the average slowdown and total rewards discounted over all the 1000 iterations. Each data point is an average of 100 new unseen experiments. As noticed in the figure, the average slowdown increases with the workload. When Orchestrator is addressing elasticity triggers during RevInP aiming to increase InPs revenue results in a smaller load of slice resource demands for the calculated time period. RevSP demonstrates the inverse of RevInP's behavior. However, it outperforms random selection of incoming slice resource requests in the queue, waiting to be processed. RL based Orchestrator, on the other hand, performs better during the highest workload decreasing about 59% of slowdown than the calculated slowdown occurred during random request selection. Eventually, with increased number of request loads, RL-based policy controlled Orchestrator learns how to address the slice elasticity requests to minimize the slowdown.

C. Behaviour Analysis

To understand the learning behaviours of our designed RL model in the network slice management scenario, this work refers to one specific data point at Figure 6 and investigates the learning curve for the agent to achieve that same point. The output learning curve during the training phase as shown in Figure 7 (a). Other benchmark heuristics for the decision making schedules, following the revenue model as described in the earlier section, is compared within one graphic. As expected, we observe that the RL-based orchestrator learns to take better decision as the iteration count increases.

Observed the graphics a question emerges: does the iteration cycles keep improving the performance of the RL approach *ad infinitum*? As seen in the Figure 7 (b), the higher iteration count increases the average reward. The learning algorithm mainly optimizes the reward over the timeline by achieving more experience according to the probability distribution. The performance gap between the RL agent and other agents narrows over time and tends to the convergence. As seen in Figure 7 (b), near the 1000^{th} iteration the algorithm converges.

V. Discussion

We briefly discuss the main current limitations that motivate a set of future work improvements:

Machine Boundaries and Locality. We performed the training simulation in a local server without accessing the higher processing units. We also did not take into account the data locality. The performance analysis is kept generic without considering the time horizon, inter-task dependencies, and partial observation issues [12].

Simulation Assumptions. For the sake of simplified experiments to demonstrate the slice management scenario targeting a larger cloud-network system of NECOS, we programmed the environment states on previously prototyped Python-based simulators. However, the real-time or on-line emulation test bed experiments may provide interesting results for future analysis of the problem.

Transparency. Any RL approach suffers from the lack of mechanisms that explain its internal structure, the means to debug and backtrace its behavior. Stated in the introduction section, we propose the RL approach presented in this paper to be utilized jointly with a policy-based decision making engine, which can enable the guarantees of SLA maintenance, taking the RL solutions as recommendations. Further studies will elucidate on the means to realize such composition.

VI. Conclusion and Future Work

This work revolves around the feasibility of applying deep RL in decision-making problems in SDN/NFV realms, more specifically for a cloud-network slice orchestrator. This experimental excercise as a novel contribution in the research field of NS, explains and reasonate that the results obtained with RL approach outperform conventional revenue-focused slice policy-oriented orchestrators, designed for comparison. Intrigued by the possibilities of widely applied RL algorithm in interactive environments, as applied in the context of this exercise, this work looks ahead to confront other incremental heuristics to perform complex and large-scale cloud-network slice management on an emulated environment added to the NECOS testbed. The discussed limitations present different challenges as prominent research topics for future work around ML-assisted cloud-network slicing.

Acknowedgements

This work was supported by the H2020 4th EU-BR Collaborative Call, under grant agreement 777067 (NECOS - Novel Enablers for Cloud Slicing), funded by the European Commission and the Brazilian Ministry of Science, Technology, Innovation, and Communication through RNP and CTIC.

References

[1] NGMN Alliance, "5G White Paper," NGMN Alliance, Tech. Rep., 2015. [Online]. Available: https://www.ngmn.org/5g-white-paper.html

[2] Z. Zhang, L. Ma, K. Poularakis, K. K. Leung, J. Tucker, and A. Swami, "Macs: Deep reinforcement learning based sdn controller synchronization policy design," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, Oct 2019, pp. 1–11.

[3] C. Zhang, X. Wang, F. Li, Q. He, and M. Huang, "Deep learning–based network application classification for sdn," *Transactions on Emerging Telecommunications Technologies*, vol. 29, 11 2017.

[4] G. Alex, T. Francesco, C. Stuart, R. Christian, and S. Joan, *Slicing 5G Networks: An Architectural Survey*. American Cancer Society, 2020, pp. 1–41. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119471509.w5GRef095

[5] F. S. D. Silva, M. O. O. Lemos, A. Medeiros, A. V. Neto, R. Pasquini, D. Moura, C. Rothenberg, L. Mamatas, S. L. Correa, K. V. Cardoso, C. Marcondes, A. ABelem, M. Nascimento, A. Galis, L. Contreras, J. Serrat, and P. Papadimitriou, "Necos project: Towards lightweight slicing of cloud federated infrastructures," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, June 2018, pp. 406–414.

[6] S. Clayman and A. Galis, "NECOS Deliverable D3.2: NECOS system architecture and platform specification V2," *NECOS Project Deliverable*, April 2019. [Online]. Available: http://www.h2020-necos.eu/documents/deliverables/

[7] M. Jinno, T. Ohara, Y. Sone, A. Hirano, O. Ishida, and M. Tomizawa, "Introducing elasticity and adaptation into the optical domain toward more efficient and scalable optical transport networks," in *2010 ITU-T Kaleidoscope: Beyond the Internet? - Innovations for Future Networks and Services*, Dec 2010, pp. 1–7.

[8] A. Medeiros, A. Neto, S. Sampaio, R. Pasquini, and J. Baliosian, "End-to-end elasticity control of cloud-network slices," *Internet Technology Letters*, vol. 2, no. 4, p. e106, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/itl2.106

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[11] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[12] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016, pp. 50–56.

[13] I. Arel, C. Liu, T. Urbanik, and A. G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," *IET Intelligent Transport Systems*, vol. 4, no. 2, pp. 128–135, 2010.

[14] J. Koo, V. B. Mendiratta, M. R. Rahman, and A. Walid, "Deep reinforcement learning for network slicing with heterogeneous resource requirements and time varying traffic dynamics," *arXiv preprint arXiv:1908.03242*, 2019.

[15] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[16] S. Clayman and A. Galis, "NECOS Deliverable D3.1: NECOS system architecture and platform specification V1," *NECOS Project Deliverable*, October 2018. [Online]. Available: http://www.h2020-necos.eu/documents/deliverables/

[17] F. L. Verdi, "NECOS Deliverable D5.2: Intelligent Management and Orchestration," *NECOS Project Deliverable*, October 2019. [Online]. Available: http://www.h2020-necos.eu/documents/deliverables/

[18] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 16, pp. 285–286, 1988.

[19] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[20] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[21] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, p. 3320–3328.

[22] J. Koo, V. B. Mendiratta, M. R. Rahman, and A. 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)*, Oct 2019, pp. 1–5.

[23] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ser. HotNets '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 50–56. [Online]. Available: https://doi.org/10.1145/3005745.3005750

[24] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 2, no. 4.