Hierarchical Multi-Agent Deep Reinforcement Learning for SFC Placement on Multiple Domains

Nassima Toumi[1], Miloud Bagaa[2], Adlen Ksentini[1]
[1] EURECOM, Sophia-Antipolis, France
[2] CSC, Espoo, Finland
Email: {nassima.toumi, adlen.ksentini}@eurecom.fr, miloud.bagaa@csc.fi

Abstract—**Service Function Chaining (SFC) is the process of decomposing a network service into multiple functions that successively process packets to deliver the end-to-end service. In a multi-domain context, SFC placement is a challenging problem due to limited knowledge of the infrastructure of the local domains, which complicates the process of finding the optimal placement solutions. On the other hand, Reinforcement Learning has gained momentum as a tool for decision-making, allowing agents to construct and improve policies using feedback from the environment. In this paper, we leverage Deep Reinforcement Learning (DRL) to perform SFC placement on multiple domains. We devise a hierarchical architecture where the local domain agents and the multi-domain agent are trained using different DRL models to perform SFC and sub-SFC placement while satisfying the SLA requirements.**

I. Introduction

The emerging use cases of 5G and beyond bring additional heterogeneous requirements that need to be accommodated. To achieve this, network operators leverage Network Function Virtualization (NFV) and Software Defined Networking (SDN) to allow increased flexibility and dynamic management of services. Service Function Chaining (SFC) leverages both aforementioned technologies to decompose services into multiple blocks that process data in an ordered manner.

In some cases, service operators deploy the functions of the same SFC on different administrative domains for reasons such as resource shortage, data aggregation for use cases such as Edge Computing where data is processed on Edge clouds closer to the clients to reduce bandwidth consumption [1], or the Function-as-a-Service (FaaS) cloud service model [2] where a client might compose its service using functions provided by multiple vendors. However, in the multi-domain context, multiple challenges arise for the SFC orchestration and management operations. In the SFC placement process in particular, it is difficult to determine the optimal Virtual Network Function (VNF) placement and link mapping solution considering that for security considerations, the domain operators disclose minimal information on their infrastructure. Furthermore, SFC placement has been proven to be an NP-Hard problem [3] [4]; indeed, it is subject to multiple constraints such as resource availability and SLA-related requirements and should improve multiple objectives. Classic optimization methods such as Integer Linear Programming, Game Theory, and different algorithms suffer from scalability issues for NP-Hard problems as their computation time increases exponentially, making them impractical for real-time usage.

This paper tackles the above-mentioned issues by applying Deep Reinforcement Learning (DRL) methods to construct decision policies based on the environment's feedback and provide placement decisions in significantly lower times. The proposed solution for multi-domain SFC placement relies on a hierarchical multi-agent architecture, where a centralized Multi-Domain Orchestrator and the local domain orchestrators each dispose of their DRL agents to perform placement on their respective views on the network topology. The architecture also includes an interface between the MDO and the local domain orchestrators that provides an abstracted view of the network to the MDO, performs SFC partitioning and sends the sub-SFCs to each local domain, then collects the local placement rewards to return the final placement reward to the MDO. We train the agent models using Deep Q-Networks (DQN) and evaluate their performance on a multi-domain architecture.

The main contributions of this paper are two-fold: First, we propose a hierarchical multi-agent Deep Reinforcement Learning framework for multi-domain SFC placement. Second, we train the agents using different DRL algorithms to perform the multi-domain SFC placement while optimizing cost and latency, and discuss their results. The remainder of this paper is organized as follows. Section II provides an overview on the existing contributions. Section III describes our proposed framework and IV details the DRL model's states, actions and rewards. The simulation settings and results are discussed in Section V, and Section VI concludes the paper.

II. Background

In a multi-domain scenario, the SFC is partitioned between multiple domains, which implies that the placement of the VNFs is performed in two steps: first, a domain is selected for each VNF, then, a physical node is selected within that domain. However, due to limited knowledge of the network infrastructure of each domain, the first step is challenging. Indeed, for security reasons, the domains disclose as little information as possible on their infrastructure, therefore, it is difficult to determine the optimal solution. Note that in this work, we consider a single domain as an autonomous network infrastructure; therefore, different administrative entities and independent divisions of the same administrative entity are considered as separate domains.

Multiple mathematical models and algorithms have been used to optimize SFC placement, such as Integer Linear Programming, different heuristics, or Game Theory [5]. In particular, in the multi-domain setting, two approaches have been applied: a distributed one, where the domains exchange messages until the convergence to an optimal solution, and a hierarchical approach, where a logically centralized entity is entrusted by the domains to partial information on their infrastructure and performs a preliminary placement and partitioning of the SFC request. However, most of these proposals fall short on scaling to bigger problem instances. Indeed, the processing time exponentially increases with the size of the problem. Recently, DRL methods have gained momentum as a tool for decision-making. They rely on Deep Neural Networks (DNN) that are trained using feedback from the environment, where the agent gradually adjusts its policy depending on the perceived rewards and penalties. Once the agents have been trained, they can provide near-optimal solutions in a short time. This makes them particularly suitable for real-time service orchestration and management. Multiple works have applied DRL for adaptive SFC placement, traffic prediction, and resource allocation [6]–[9].

The contributions in [10] and [11] proposed solutions for Network Slice orchestration on multiple domains using DRL models, but supposed that the agent had access to complete information on the domains' infrastructure, therefore ignoring the limited visibility aspect. Tang *et al.* [12] devise on a DRL model for SFC placement on geo-distributed data-centers with separate decentralized algorithms for routing optimization and task routing, however, the authors suppose that the agent disposes of full knowledge on the network infrastructure of the data-centers. In [13], DRL is employed for VNF Forwarding Graph (VNFFG) embedding on multiple non-cooperative domains, where the domains compete for the SFC requests and update their pricing policies accordingly to get the client to place more VNFs on their network, thus increasing their profit, while the client trains its own model by evaluating the different domains using the observed QoS metrics of the deployed SFCs to assess whether the SLAs have been breached. However, the model is only tested on a small network setup of 3 domains.

In this contribution, we propose a multi-agent DRL-based framework for the placement of multi-domain SFCs while jointly optimizing cost and latency. Our proposal takes into account the limited visibility aspect, and allows the agents to communicate through an interface that acts as an intermediary.

III. Proposed Architecture

In the following, we describe our proposed architecture as depicted in Figure 1, and detail the learning process of the different agents and their interactions to perform end-to-end SFC placement. SSimilar to the work in [14], the proposed architecture adopts a hierarchical approach. It is composed of multiple orchestrators that employ separate DRL agents and exchange requests and rewards through a Multi-Domain Interface (MDI). Our framework supports the use of different DRL methods for each agent of the architecture, which allows increased flexibility in combining algorithms. The Multi-Domain Orchestrator (MDO) uses its DRL agent to perform an initial domain placement of the VNFs using the partial view on the network that has been provided by the MDI. Once each VNF has been mapped to a domain, the MDI partitions the SFC accordingly and sends the sub-SFCs to the domain orchestrators for a local placement using their own DRL agents. We provide details on the main components of our framework in the following:

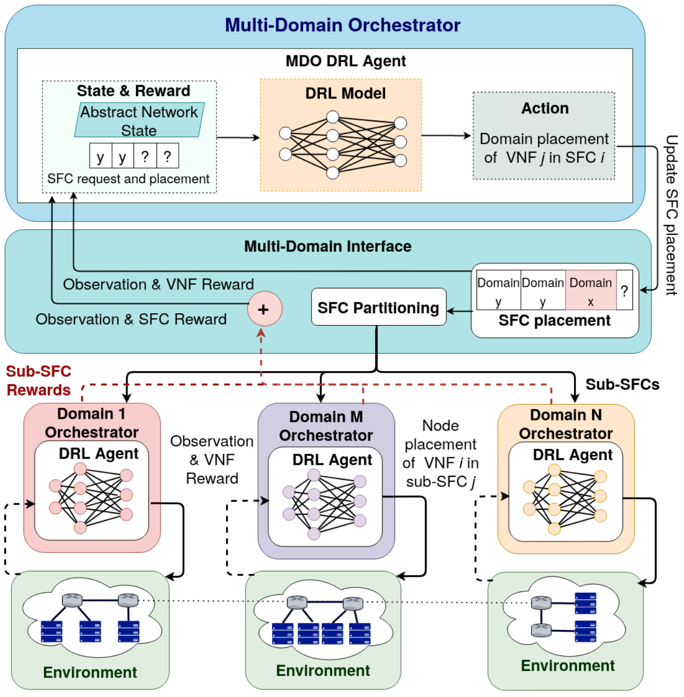

Fig. 1: Multi-domain SFC Embedding Framework

A. Multi-Domain Orchestrator

The MDO is in charge of performing the initial placement of the multi-domain SFC, by selecting the domains where each VNF is placed. The MDO is composed of an agent that uses the abstract network state and the rewards from the environment to construct a policy that maximizes the SFC rewards. Through an exploitation/exploration trade-off, the agent creates associations between states, actions, and rewards and selects the best actions for each state. Note that the MDI plays the role of the environment for the MDO, since it feeds the states and rewards to the DRL agent. Furthermore, due to the limited knowledge of the local domain infrastructure, the MDO agent's environment is Partially Observable.

B. Multi-Domain Interface

The MDI presents to the MDO a partial view of the global network constructed using information disclosed by the local domains such as the available resources, their unit price, and the inter-domain vertices [15]. In this work, the DRL agents perform placement one VNF at a time. For each VNF placement decision, the MDI is in charge of updating the total

SFC placement and the abstract network topology accordingly. It returns that updated network state along with the partial reward for that VNF. Once the domain placement of the whole SFC has been performed, the MDI partitions the SFC into sub-SFCs by grouping the VNFs that have been mapped to the same domain and sends the sub-SFC requests to their associated domain orchestrators. The MDI then collects the sub-SFC rewards from each agent, and returns the total SFC reward and the updated network state to the MDO DRL agent.

C. Local Domain Orchestrators

Each local domain orchestrator performs a local placement of the incoming sub-SFCs with a full view of the environment and uses the rewards to update its policy. The rewards are also sent back to the MDI to compute the complete SFC placement reward. Note that for each local orchestrator, the environment is composed of the local topology that provides the network state and rewards, as well as the MDI that provides the sub-SFCs used to complete the state fed to the agent. Note that the agents of each level of the architecture are trained separately. First, the local orchestrator's DRL agents are trained using randomly generated sub-SFCs, then switched to the full exploitation mode to train the MDO DRL agent. This choice is due to the fact that the local orchestrator's placement efficiency has an impact on the placement that is performed by the MDO. Indeed, a local orchestrator performing an exploration action might cause a decreased reward for the MDO agent, which would lead this latter to under-evaluate the value of that specific state-action pair.

IV. System Model

This section describes the environment's states, actions, and rewards for the local domain agents and the MDO agent. We also formulate the constraints of the SFC placement problem.

A. States

At each time step, the state observation for the DRL agents is composed of information on the VNF that should be placed by the next action, the placement of the previous VNFs, the available resources of the network topology, and their unit price, and the link latency. For each physical node or domain n, the remaining available capacity for each resource type r of the network topology is denoted by $\mathcal{R}_{r,n}$, and the unit price is expressed using $\zeta_{r,n}$. For each domain or inter-domain link l from the set $\mathcal{L}$, we denote by $\mathcal{R}_{\omega,l}$, ζ_l and ϕ_l the bandwidth capacity, the unit price and latency respectively. Each VNF i from the set of VNFs $\mathcal{V}_i$ of an SFC i is characterized by its resource requirements $\nabla_{r,i,j}$ for each resource type r from the set $\mathcal{R}$, the required amount of bandwidth $\mathcal{W}_i$, the maximal latency ϕ_i^+ and $\mathcal{C}_{i,paid}$, which is the price that has been paid by the tenant to deploy the SFC. For the MDO agent, the state of a VNF also includes a set of authorized domains $\mathcal{M}_{i,j}$ on which the VNF can be placed, while for the local domain agents, no such restriction applies. If a domain isn't allowed for a certain VNF, the state for that time step is updated to set the resource capacity of that domain and its connecting inter-domain links to 0.

B. Actions

The actions of each agent determine the domain or physical node where the current VNF should be placed. For the sake of simplicity and to reduce the action space of our model, we assume that the shortest paths between the domains and physical nodes have been computed before the placement process. Therefore, the link capacity, cost, and latency from the previous VNF can be directly computed. For a VNF j, for the MDO agent, the number of actions would correspond to the number of its allowed domains $|\mathcal{M}_{i,j}|$, while for a local domain orchestrator, the number of actions is the number of physical nodes of the domain d topology $|\mathcal{N}_d|$. We use the boolean $\mathcal{X}_{i,j}^n$ as a decision variable which takes the value 1 if the VNF j of SFC i has been mapped to the domain or physical node n, and 0 otherwise. The placement of an SFC or sub-SFC i must satisfy the following constraints that apply on both levels of the architecture, unless stated otherwise:

1) Mapping Constraints: Each VNF i must be placed on only one domain or physical node from its set of allowed placements. For the sake of simplicity, in this formulation, we denote by $\mathcal{M}_{i,j}$ the set of allowed nodes for both the MDO agent and the local domain orchestrators. Two successive VNFs can be mapped to two domains or nodes n and m if and only if they are connected by an available physical path, which is expressed using the boolean $\rho_{n,m}$:

$$\sum_{n\in\mathcal{M}_{i,j}} \mathcal{X}_{i,j}^n = 1, \qquad \forall j \in \mathcal{V}_i \tag{1}$$

$\forall j \in \mathcal{V}_i, \forall n \in \mathcal{M}_{i,j}, \forall m \in \mathcal{M}_{i,j+1}$:

$$\mathcal{X}_{i,j}^n \cdot \mathcal{X}_{i,j+1}^m \leq \rho_{n,m}, \tag{2}$$

2) Capacity Constraints: To place a VNF j on a certain node or domain n, sufficient amounts of all of the resource types r (CPU, RAM, disk space...) must be available on that node. This constraint is expressed as follows:

$$\sum_{j\in\mathcal{V}_i} \nabla_{r,i,j} \cdot \mathcal{X}_{i,j}^n \leq \mathcal{R}_{r,n}, \qquad \forall n \in \mathcal{M}_{i,j+1}, \forall r \in \Re \tag{3}$$

For each pair of successive VNFs j and $j+1$, each link l that is part of the path between their selected placements must dispose of sufficient remaining bandwidth capacity. Denoting by the boolean $\tau_l^{n,m}$ whether a link is part of the physical path between domains n and m, the constraint can be expressed as follows: $\forall l \in \mathcal{L}$:

$$\sum_{j\in\mathcal{V}_i} \sum_{n\in\mathcal{M}_j} \sum_{m\in\mathcal{M}_{j+1}} \mathcal{W}_i \cdot \mathcal{X}_{i,j}^n \cdot \mathcal{X}_{i,j+1}^m \cdot \tau_l^{n,m} \leq \mathcal{R}_{\omega,l} \tag{4}$$

3) Latency Constraint: To provide a satisfactory Quality of Service (QoS) depending on the SLA selected by the client, the end-to-end latency for an SFC i cannot exceed a certain limit ϕ_i^+.

$$\phi_i \leq \phi_i^+ \tag{5}$$

The latency ϕ_a for each agent a is computed as follows:

$$\phi_a = \sum_{j\in\mathcal{V}_i} \sum_{l\in\mathcal{L}} \sum_{n\in\mathcal{M}_{i,j}} \sum_{m\in\mathcal{M}_{i,j+1}} \phi_l \cdot \mathcal{X}_{i,j}^n \cdot \mathcal{X}_{i,j+1}^m \cdot \tau_l^{n,m} \tag{6}$$

Note that the end-to-end latency can be obtained by summing the ϕ_a values from each local domain orchestrator d from the set $\mathcal{D}$ which are collected by the MDI, and the inter-domain latency value from the MDO.

$$\phi_i = \phi_{\text{MDO}} + \sum_{d \in \mathcal{D}} \phi_d \tag{7}$$

4) Cost Constraint: The total deployment cost $\mathcal{C}_i$ should remain below a certain amount to guarantee a profit percentage of at least 5% for the SFC provider:

$$\mathcal{C}_i \leq 0.95 \cdot \mathcal{C}_{i,paid} \tag{8}$$

Where $\mathcal{C}_{comp}$ and $\mathcal{C}_{link}$ the computing and link costs for each domain d are computed as follows:

$$\mathcal{C}_{comp} = \sum_{n \in \mathcal{N}_d} \sum_{r \in \Re} \sum_{j \in \mathcal{V}_i} \zeta_{r,n} \cdot \nabla_{r,i,j} \cdot \mathcal{X}_{i,j}^{n} \tag{9}$$

$$\mathcal{C}_{link} = \sum_{j \in \mathcal{V}_i} \sum_{l \in \mathcal{L}} \sum_{n \in \mathcal{M}_j} \sum_{m \in \mathcal{M}_{j+1}} \zeta_l \cdot \mathcal{W}_i \cdot \mathcal{X}_{i,j}^{n} \cdot \mathcal{X}_{i,j+1}^{m} \cdot \tau_l^{n,m} \tag{10}$$

This latter equation can also be used to compute the inter-domain link cost. The total cost is computed by summing the link and node costs for each domain, and adding the inter-domain link costs.

$$\mathcal{C}_i = \mathcal{C}_{link,MDO} + \sum_{d \in \mathcal{D}} (\mathcal{C}_{link,d} + \mathcal{C}_{comp,d}) \tag{11}$$

C. Rewards

The goal of this model is to perform multi-domain SFC placement while minimizing deployment cost $\mathcal{C}_i$, the end-to-end latency ϕ_i, and satisfying the aforementioned constraints to minimize the request rejection rate. The reward function for an SFC is a fixed placement reward minus a penalty which depends on the quality of the solution, expressed as the difference with the optimal value. This difference is weighted by the importance of each objective depending on the selected SLA (e.g., Best Effort, Ultra-Low Latency...). To formulate the reward function, we first formulate the penalties related to cost and latency separately, expressed as $P_{\mathcal{C}}$ and P_{ϕ}, respectively. Their values are normalized using the maximal values for each objective for the selected SLA.

$$P_{\mathcal{C}} = \alpha_{\mathcal{C}} \cdot \frac{\mathcal{C}_i - 0.95 \cdot \mathcal{C}_{i,paid}}{0.95 \cdot \mathcal{C}_{i,paid}} \tag{12}$$

$$P_{\phi} = \alpha_{\phi} \cdot \frac{\phi_i - \phi_i^{+}}{\phi_i^{+}} \tag{13}$$

With $\alpha_{\mathcal{C}}$ and α_{ϕ} being the weights associated to cost and latency which depend on the selected SLA. The final reward function that the agent aims to maximize can be formulated as follows:

$$reward = \mathcal{A}_i \cdot (R_S - P_{\mathcal{C}} - P_{\phi}) - (1 - \mathcal{A}_i) \cdot P_F \tag{14}$$

With R_S being the reward obtained for successfully placing the SFC, and P_F the penalty for failing to place that SFC. And the boolean $\mathcal{A}_i$ expresses whether a complete SFC has been placed, its value can be computed using the following equation:

$$\mathcal{A}_i = \prod_{j \in \mathcal{V}_i} \sum_{n \in \mathcal{M}_{i,j}} \mathcal{X}_{i,j}^{n} \tag{15}$$

V. Evaluation

To evaluate our solution, we deploy an implementation on a physical machine with 12 Intel Core i7-10710U 1.10GHz CPU Cores and 32GB of memory, hosting an Ubuntu 20.04 x64 Operating System. The simulation environment is implemented using Python. To learn the optimal policy π_*, we use PyTorch [16] to implement fully connected Deep Neural Networks (DNN) for the DRL agents.

A. DRL Model

Tabular RL methods, such as Q-learning [17] and SARSA [18], update the action-value function estimates to converge to the optimal policy. While their convergences have been proven, they are limited to the small problem of limited states and actions. Therefore, to enlarge the scope of RL problems, the DQN Algorithm has been introduced [19]. DQN leverages neural networks to estimate the Q-values and hence considers continuous state spaces. A considerable enhancement that has been introduced to the original DQN algorithm in [20] is the use of a separate Q-network referred to as *target network*, which is used to approximate the state-action values for the TD-error computation in the loss function. The target network is a clone of the original Q-Network with a delayed weight vector that is updated (copied from the original Q-network) periodically after a certain number of time steps.

For our simulation, we employ Deep Q-Networks on the MDO level and on the local domains. For both levels, the size of the input and output layers are similar to the size of the state observation and action space, respectively. Therefore, for the MDO agent, the output layer size would be the number of domains in the topology, while for the local domains, it would correspond to the number of physical nodes in the network. Note that our framework can be extended to support multiple combinations of additional DRL algorithms.

For the local domain agents, we implement the main DQN and target network DNNs with three hidden layers of 512, 256, and 128 nodes for our simulation. While for the DQN agent of the MDO, we implement two hidden layers of 256 and 128 nodes. For both levels, we apply the Rectified Linear Unit (ReLU) activation function to the hidden and output layers. We apply a discount factor γ of 0.99, and the learning rate is set to 10^{-2} for both the main and target networks. We also implement experience replay with a batch size of 256, and the target network is updated every eight episodes. The neural network parameters are updated using the ADAM optimizer [21]. The decayed epsilon-greedy policy is used to determine whether an action is chosen randomly or according to the current policy, where the epsilon value is gradually decreased as the episodes are processed by the agent.

B. Model Training

As stated earlier, we train the MDO and domain agents separately. We first train the local domain model by running 30000 independent episodes. For each episode, we randomly generate local domain topologies with a 3-level Fat-Tree structure composed of 16 servers using the *networkx* library.

We also generate sub-SFC requests of 1-10 VNFs, where each VNF is characterized by a type (small, medium, large) that defines the amounts of required resources for each resource type. Each request specifies the number of users and the SLA that the SFC must satisfy. Table I shows the maximum latency limit, the bandwidth per user requirement, the cost per user, and the weights associated to cost and latency for the reward function for each SLA. The first two SLAs have a low latency requirement with a high latency weight. The first one has a high bandwidth per user requirement, while SLA 2 seeks a trade-off between cost and latency. In contrast, SLA 3 favors reducing cost over latency. The last SLA corresponds to best-effort services; hence it doesn't have a maximum latency requirement and gives a higher weight to cost. For each episode, a set of 1-10 sub-SFCs is randomly generated, as the number of sub-SFCs that the local orchestrator receives might vary depending on the MDO placement. To obtain the full reward for the episode (10 minus cost and latency-related penalties), the agent must successfully place all of the sub-SFCs; otherwise, the episode is terminated, the resources of the topology are freed, and the agent receives a penalty of -100. Figure 2 illustrates the rewards in blue and 100 episode average in orange for the DQN model of the local domains, where the x-axis represents the number of episodes, and the y-axis represents the perceived rewards. It can be observed that the model progressively starts converging to higher reward values after 15000 episodes, with an average oscillating between 40 and 80 depending on the quality of the solutions, meaning that most of the placement actions succeed with occasional placement failures that can be imputed to the exploration actions that are performed by the agent. Furthermore, early SFC placement failures terminate the episode and cause higher penalties. It can be noticed that after 17000 episodes, the maximum observed penalties decrease, which indicates that SFC placement failures occur later in the episode, meaning that a higher number of SFCs are placed by episode.

Once the models of the local agents have been trained, we switch them to a full exploitation mode and train the MDO agent using 8 local domain agents with their randomly generated Fat-Tree topologies and the corresponding abstracted network view. For this training, we generate SFC requests of 4-10 VNFs, adding the allowed domain sets in their characteristics. For each episode, 10 SFCs must be placed successfully by the MDO and all of the selected local domain orchestrators in order to get the full placement reward collected by the MDI, otherwise, if the placement of an SFC fails, the episode is terminated, and the agent receives a penalty. To train our MDO model, since the observation space and number of possible actions is smaller than for the local domain agent, we run 5000 independent episodes. Figure 3 shows the rewards and 100 episodes running average for the DQN model of the MDO, which is also composed of the local domain model rewards. It can be seen that the 100 episode running average of the rewards slowly increases as the SFC requests are placed. The agent converges after 3000 episodes, reaching an average reward value between 70 and 90, meaning that

SLA	0	1	2	3	4
Cost/user	8000	6000	4000	3000	1000
Max Latency (ms)	100	100	200	300	-
Bw/user (Mb/s)	500	50	200	50	20
Cost Weight	0.1	0.25	0.5	0.75	0.9
Latency Weight	0.9	0.75	0.5	0.25	0.1

TABLE I: SLA Characteristics

despite the partial observability of the environment, the model successfully captured the environment's dynamics. Indeed, most of the episodes completed successfully with all of the SFCs being placed, which is confirmed by the rewards plot where a small number of episodes produced penalties.

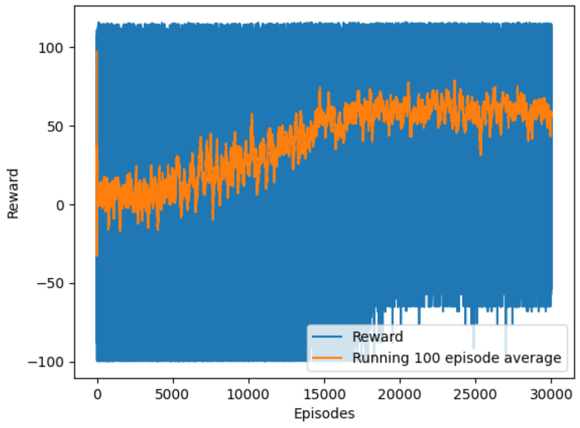

Fig. 2: Local Domain DQN Model Training Reward

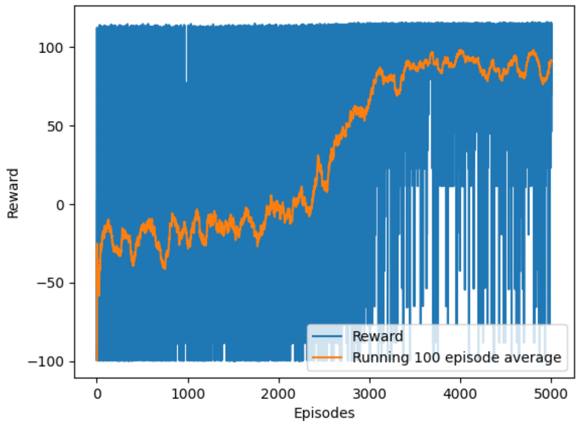

Fig. 3: MDO DQN Model Training Reward

C. Model Evaluation

After training the DQN model, we switch it to the full exploitation mode and observe three key metrics: the acceptance rate of the SFC requests, to assess the model's ability to identify the problem's constraints and adapt its parameters, and the relative cost and latency compared to the SLA objectives, to evaluate the quality of the provided

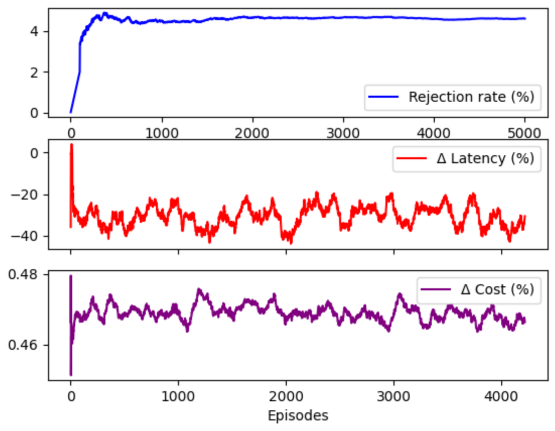

Fig. 4: Multi-Domain SFC Placement Metrics

solutions. For this second part of the experiment, we generate 5000 new episodes. Figure 4 displays the obtained results for the selected evaluation metrics.

The first sub-figure in blue represents the running average of the individual SFC request rejection rate, where it can be seen that the rejection rates oscillate between 4 and 5%.The sub-figures in red and purple display the 100 episodes, showing the average difference between the obtained latency and cost, for each successful episode, and their optimal values, respectively. We set the optimal value for latency at 60% under the maximum limit of the SLA, and the optimal cost value at 50% of the value paid by the customer to deploy that SFC. The obtained results show that the latency values are on average around $20-40\%$ over the optimal value for that SFC, but remain largely below the maximum SLA limit, and the cost values are on average 0.47% below (thus better than) the optimal value of 95% of the SFC's revenue. Therefore, it can be concluded that our proposal's SFC placement results achieve latency values that satisfy the SLA requirements while maximizing the profit margin of the SFC providers.

VI. Conclusion

In this contribution, we designed a multi-agent framework for multi-domain SFC placement using Deep Reinforcement Learning. We formulated the problem's constraints and defined the states, action space, and rewards for the MDO and local domain agents. Our proposal was validated on a simulation testbed using the DQN algorithm, and the results showed that despite the limited visibility on the network infrastructure, the agents were able to capture the environment's dynamics and learn efficient policies, where 94% of the SFCs were successfully placed while minimizing cost and latency. In future works, we aim to enhance this framework to support post-deployment orchestration operations such as migration or scaling. Further, we aim to experiment additional DRL algorithms such as actor-critic methods to reduce the learning time and improve performance.

References

[1] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An overview on edge computing research," *IEEE Access*, vol. 8, pp. 85 714–85 728, 2020.

[2] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, and P. Suter, *Serverless Computing: Current Trends and Open Problems*. Singapore: Springer Singapore, 2017, pp. 1–20.

[3] M. Ghaznavi, N. Shahriar, S. Kamali, R. Ahmed, and R. Boutaba, "Distributed service function chaining," *IEEE J. Sel. Areas Commun.*, vol. PP, no. 99, pp. 1–1, 2017.

[4] L. Yala, P. A. Frangoudis, and A. Ksentini, "Latency and availability driven VNF placement in a MEC-NFV environment," in *IEEE Global Communications Conference, GLOBECOM 2018, Abu Dhabi, United Arab Emirates, December 9-13, 2018*. IEEE, 2018, pp. 1–7.

[5] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Trans. on Netw. and Serv. Manage.*, vol. 13, no. 3, pp. 518–532, Sept 2016.

[6] N. Jalodia, S. Henna, and A. Davy, "Deep reinforcement learning for topology-aware vnf resource prediction in nfv environments," in *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2019, pp. 1–5.

[7] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "Nfvdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*, 2019, pp. 1–10.

[8] H. A. Shah and L. Zhao, "Multi-agent deep reinforcement learning based virtual resource allocation through network function virtualization in internet of things," *IEEE Internet of Things Journal*, pp. 1–1, 2020.

[9] S. Troia, R. Alvizu, and G. Maier, "Reinforcement learning for service function chain reconfiguration in nfv-sdn metro-core optical networks," *IEEE Access*, vol. 7, pp. 167 944–167 957, 2019.

[10] G. Kibalya, J. Serrat, J. Gorricho, R. Pasquini, H. Yao, and P. Zhang, "A reinforcement learning based approach for 5g network slicing across multiple domains," in *2019 15th International Conference on Network and Service Management (CNSM)*, 2019, pp. 1–5.

[11] A. I. Swapna et al., "Policy controlled multi-domain cloud-network slice orchestration strategy based on reinforcement learning," in *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2020, pp. 167–173.

[12] T. Tang, B. Wu, and G. Hu, "A hybrid learning framework for service function chaining across geo-distributed data centers," *IEEE Access*, vol. 8, pp. 170 225–170 236, 2020.

[13] P. T. A. Quang, A. Bradai, K. D. Singh, and Y. Hadjadj-Aoul, "Multi-domain non-cooperative vnf-fg embedding: A deep reinforcement learning approach," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops*, 2019, pp. 886–891.

[14] N. Toumi, O. Bernier, D.-E. Meddour, and A. Ksentini, "On cross-domain service function chain orchestration: An architectural framework," *Computer Networks*, vol. 187, p. 107806, 2021.

[15] M. Shen, K. Xu, K. Yang, and H. H. Chen, "Towards efficient virtual network embedding across multiple network domains," in *2014 IEEE 22nd Int. Symp. of Quality of Serv. (IWQoS)*, May 2014, pp. 61–70.

[16] P. Adam et al., "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[18] G. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," *Technical Report CUED/F-INFENG/TR 166*, 11 1994.

[19] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2016.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 02 2015.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.