Reconfiguring Network Slices at the Best Time With Deep Reinforcement Learning

Adrien Gausseran, Redha A. Alliche, Hicham Lesfari, Ramon Aparicio-Pardo,
Frédéric Giroire, Joanna Moulierac

HAL Id: hal-03786887
https://inria.hal.science/hal-03786887

Submitted on 23 Sep 2022

Reconfiguring Network Slices at the Best Time With Deep Reinforcement Learning

Adrien Gausseran, Redha A. Alliche, Hicham Lesfari, Ramon Aparicio-Pardo, Frédéric Giroire, Joanna Moulierac
Université Côte d'Azur, CNRS, Inria Sophia Antipolis, France

Abstract—The emerging 5G induces a great diversity of use cases, a multiplication of the number of connections, an increase in throughput as well as stronger constraints in terms of quality of service such as low latency and isolation of requests. To support these new constraints, Network Function Virtualization (NFV) and Software Defined Network (SDN) technologies have been coupled to introduce the network slicing paradigm. Due to the high dynamicity of the demands, it is crucial to regularly reconfigure the network slices in order to maintain an efficient provisioning of the network. A major concern is to find the best frequency to carry out these reconfigurations, as there is a trade-off between a reduced network congestion and the additional costs induced by the reconfiguration. In this paper, we tackle the problem of deciding the best moment to reconfigure by taking into account this trade-off. By coupling Deep Reinforcement Learning for decision and a Column Generation algorithm to compute the reconfiguration, we propose `Deep-REC` and show that choosing the best time during the day to reconfigure allows to maximize the profit of the network operator while minimizing the use of network resources and the congestion of the network. Moreover, by selecting the best moment to reconfigure, our approach allows to decrease the number of needed reconfigurations compared to an algorithm doing periodic reconfigurations during the day.

I. INTRODUCTION

To meet the growing and increasingly diverse demands of users and companies, networks have evolved, adopting new technologies to make their management more efficient and more responsive to dynamic changes in traffic. The first fundamental evolution to the management of modern networks is Software Defined Networking (SDN). SDN is a network paradigm that decouples the control plane from the data plane and enables centralized control of the network. The network becomes programmable and routing rules can be adjusted in real time leading to a better responsiveness of network management in case of traffic changes.

Network Function Virtualization (NFV) comes with SDN as a paradigm allowing the decoupling of network functions from the hardware. Functions can be virtualized on generic servers located in data centers dispersed all over the network. NFV allows first of all the reduction of capital expenditure (Capex) by avoiding the purchase of a dedicated equipment for each function. It also allows a reduction of operational costs (Opex) since a function can be easily stopped when not used.

Finally, NFV allows a more flexible network management since functions can be instantiated on demand anywhere in the network when needed due to traffic dynamics.

By combining SDN and NFV technologies, network management thus is greatly enhanced by making the network programmable, dynamic, and flexible, and by allowing for controlled sharing of resources between different services and users. The increasing importance of wireless networks and the emergence of 5G bring out new needs such as massive device connectivity, high mobility and a great diversity in the quality of service (QoS) requirements. Network slicing has been proposed to meet this challenge and to satisfy these diversified service needs. By dividing the network infrastructure into multiple logical isolated networks, network slicing allows the support of a wide range of communication scenarios with a diversified set of service demands, requirements, and performance. To meet a demand, a slice needs to fulfill an end-to-end service which requires joint allocation of different types of resources. A slice must be deployed in real time and, thus, the corresponding provisioning of network, computing, and storage resources has to be done dynamically.

In 5G networks, traffic is considered to be highly dynamic, and network requests may be subject to frequent changes such as arrivals and departures. This dynamicity may fragment the slice resource usage and make the use of network resources less efficient. To counter this effect, network operators need to regularly reconfigure the network slices. Indeed, thanks to SDN and NFV, the routing of flows and the allocation of the network functions can be easily modified. The slice allocation can thus be adjusted in order to reduce the resource utilization with the goal of minimizing operational costs.

In this work, *we propose a method to efficiently reconfigure the network without severe interruptions of traffic* (as it is done in [1]). We focus on the case where 5G slices correspond to service chains. We use a *make-before-break* mechanism, in which we first allocate the resources for a secondary route while keeping the first one intact (i.e., two redundant routes are reserved in parallel). Then, we migrate the flow to the new route and release the resources of the old one. There is no disruption of the traffic impacting severely the Quality of Service (QoS) of the slices. The computations of the new routes are done by using a *scalable optimization decomposition method making use of a column generation approach*. Even when using such a mechanism to avoid degrading the QoS, network operators do not want to reconfigure their network

too frequently, as it may lead to additional management costs. On the opposite, reconfiguring too rarely during the day may lead to a sub-optimal network usage. A simple policy with low computational cost is to regularly reconfigure after a fixed number of minutes. However, reconfiguring in response to variation of traffic can reduce the number of reconfigurations required each day without impacting the overall improvement obtained.

In this paper, we propose a *reconfiguration management agent* that chooses when to initiate reconfiguration as a function of different parameters such as the traffic dynamics and the level of network congestion. We use a *Deep Reinforcement Learning technique* (DRL) by implementing it with Tensorflow [2] and their Deep Q-learning Network (DQN) agent. We then show that our agent improves the efficiency of reconfigurations by performing less reconfigurations while still minimizing the network operational costs compared to doing periodic and frequent reconfigurations.

II. RELATED WORK

Routing and provisioning of slices. The VNF allocation and SFC provisioning problems have been widely studied in recent years. Some works focus on static scenarios such as [3], [4] in which the authors develop efficient methods coupling chained allocation of VNFs and traffic routing within SFCs. The dynamic nature of network traffic raises a range of problems concerning the acceptance of incoming slices, resource management, and Service Level Agreement compliance. Cheng *et al.* [5] use method to deploy and manage slice provisioning using deep learning and Lyapunov stability theories. In [6], the authors present a Mixed Integer Linear Program and a heuristic to add new slices by minimizing the bandwidth consumption and the slice provisioning cost while taking into account the VNF migrations. In [7], a method is presented to manage the creation, modification or deletion of slices by adapting to the traffic. Their goal is to minimize the number of slices while having enough bandwidth available to serve the traffic.

Reconfiguration using standard techniques. The reconfiguration of SFCs and/or slices aims to maintain a near-optimal state of the network over time in order to optimize the network usage and the acceptance of demands. Wang *et al.* [8] develop an algorithm that manages two types of reconfiguration to maximize the operator's profits. First, a reconfiguration to adapt the slices to the current traffic. Second, a reconfiguration modifying the flows traversing the slices. The algorithm then schedules the reconfigurations and reserves resources for future traffic to reduce the number of potential future reconfigurations. Each reconfiguration includes the service interruption and resource usage as costs. In [9] authors proposed a slice reconfiguration technique in which the new state of the network is pre-computed. The reconfiguration is done in several steps in which the VNFs and routes are modified while taking into account capacities and delays. In [10], authors proposed an integer linear program

and an heuristic to efficiently reconfigure Service Function Chains using a *make-before-break* strategy. In [11], column generation techniques are used to optimize the reconfiguration of hundreds of network slices in only few seconds.

Learning-based reconfiguration Some recent works use reconfiguration techniques based on reinforcement learning and try to predict the dynamicity of the network. Liu *et al.* [12] propose a VNF migration strategy based on Double-Deep Q-Network. Their goal is to equally place VNFs between Mobile Edge clusters and core clouds in order to avoid congestion at the Edge. The migration takes into account future traffic and tries to reduce the number of migrations while minimizing the number of congested links. In [13] the authors use reinforcement learning to perform dynamic SFC resource allocation in optical networks. They define an agent that decides for each SFC when and which reconfiguration to perform (migration, scaling-up, scaling-down) in order to meet the QoS criteria. Each SFC has a bound on the maximum number of authorized reconfigurations and the optimization objective is the total number of reconfigurations. Each reconfiguration has a penalty for service interruption.

In [14] the authors use DRL to predict when to reconfigure in order to minimize the resources consumed. Unlike our work, the authors focus on intra-slice reconfigurations with a fixed number of requests throughout the experiment and with unchanging service sources and destinations. Their algorithm optimises slices only locally and uses a pre-computed set of paths using Depth-first search algorithm. Guan *et al.* [15] use a Markov Renewal Process to predict changes in the resource occupancy of slices and reserve resources for slices that obtain higher revenues at lower cost. They use Deep dueling neural network combined with Q-Learning to choose for each slice whether to reconfigure it or not. Their goal is to maximize long term revenue: the increased user utility minus a cost for the resource utilization and the service interruptions. Similar to the last two works mentioned, we establish an agent based on DRL to choose when to reconfigure the slices.

To the best of our knowledge, we are the first to propose a methodology to reconfigure a dynamic network with incoming and outgoing slices and not being dependent on a fixed number of slices throughout the day. Our deep reconfiguration learning algorithm adapts its behavior based on the variations of the whole network traffic and not to a fixed set of flows within a slice. Moreover, we do not fix a limit of the reconfigurations per slice or per day. The agent determines the best time to reconfigure and performs the needed number of reconfigurations depending on the traffic variations. The reconfiguration is computed independently using a column generation algorithm based on a *make-before-break* reconfiguration that chooses which slices to reconfigure. This allows our method to deal with a large number of slices, taking only a few seconds to compute the reconfiguration.

III. SYSTEM MODEL AND PROBLEM FORMULATION

We consider the network as a directed capacitated graph

$G = (V, L)$ where V represents the node set and L the link set. Using the resources available in this network, we must allocate a set of slices requests D. A network slice request $d \in D$ is modeled as a Service Function Chain (SFC) (as in [16]) with a quintuplet: (i) the source v_{SRC}, (ii) the destination v_{DST}, (iii) the required bandwidth BW_d in traffic units, (iv) the delay requirement γ_d, and, (v) the ordered sequence of network functions c_d that need to be performed, where $f_i^{c_d}$ is the $i - th$ function of chain c_d. Each network function instance $f \in F$ has an installation cost c_f accounting for all the VNF usage costs (licenses, energy consumption, etc). Each slice $d \in D$ provides a revenue u per bandwidth unit.

We aim to find an allocation of the slice requests such that the network operator profit accumulated over a certain time window is maximized. This cumulative profit is the sum of the instantaneous profits p_t at each observation time (i.e. minute). The profits p_t are computed as the difference between the overall revenue of the allocated slices and the overall cost of the deployed VNFs at time t:

$$p_t = \sum_{d \in D_t} u \cdot \text{BW}_d - \sum_{f \in F_t} c_f \qquad (1)$$

where D_t and F_t is the set of slices and function instances allocated at observation time t, respectively.

In a dynamic scenario with no information on future traffic, the impact of recently arrived requests onto the cumulative profit (at the operator time horizon) is still not known: slices routed using long paths will consume too many resources preventing the allocation of future requests. To take into account that, at each time, we target to place new requests on paths such that resources (i.e., bandwidth at each link and network functions at each node) are minimized. Therefore, in the method detailed in Section IV, the slice revenue maximization (first term in Eq. 1) is obtained by minimizing the resources used by the slices (see Eq. 2).

As a consequence of the lack of information about the future, the trivial mechanics of requests coming and leaving over time will bring the network to a global sub-optimal state. Hence, we are forced to periodically reconfigure the network. To do that, we use the *make-before-break* mechanism [1] that avoids network service disruption due to traffic rerouting. We detail the process of the reconfiguration of a request in the following example of Figure 1.

A. Example

Two requests, *B* to *C* and *F* to *E* are routed during step (b). Four VNFs have been installed in *B*, *C*, *E* and *F* to satisfy the needs of these requests. To avoid the usage cost of new VNFs, the route from *A* to *F* with minimum cost is a long 5–hops route and the VNF already installed in node *B* is shared by the two slices (step (c)). When requests from *B* to *C* and from *F* to *E* leave, the request is routed on a non-optimal path (step (d)), which uses more resources than necessary. We compute one optimal 3-hop path and reroute the request on it (step (f)) with an intermediate *make-before-break* step (step (e)) in which

(a) Requests   (b) Two requests   (c) A 3rd request

(d) First two requests leave   (e) Reconfiguration phase   (f) Optimal routing for the third request
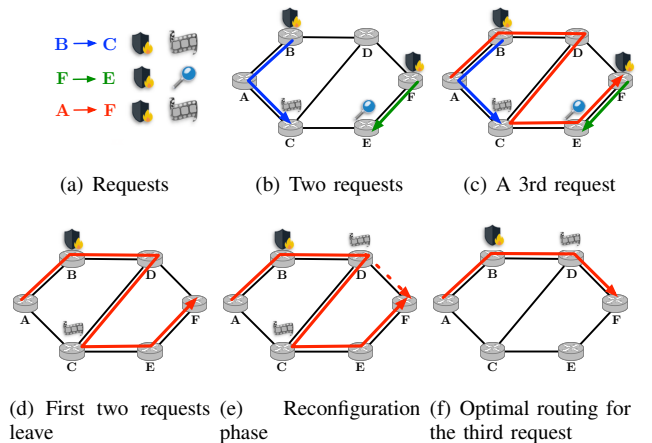
Fig. 1: An example of the reconfiguration of a request using a *make-before-break* approach with one step.

both routes co-exist. During this intermediate step, traffic can follow both paths, resources are accordingly reserved. The old path is removed when all the allocation and provisioning are ready to be used. In doing so, no packet losses occur and the traffic is not interrupted. In this example, the reconfiguration can be done in only one step of reconfiguration, but we will consider in the following up to 3 steps of reconfiguration.

IV. COLUMN GENERATION OPTIMIZATION MODELS

In this section, we first describe the main principles of the column generation algorithm to solve the problem of reconfiguration of network slices (first proposed in [11]), and then we show how we will use this for our deep reinforcement learning algorithm.

Column generation (CG) [17] is a model allowing to solve an optimization model without explicitly introducing all variables. It thus allows solving larger problem instances than a compact model with an exponential number of variables. In this work, the master problem (MP) seeks a possible global reconfiguration for all slices with a path-formulation. This means that the problem decision variables (columns) correspond to paths on the layered graph. In the restricted master problem (RMP), only a subset of potential paths is used for each slice. At the initialization, the set of paths is the one used before reconfiguration. Each pricing problem (PP) then generates a new path for a request, together with the placement of the VNFs. During a reconfiguration, slices are migrated from one path to another. Note that, as the execution of each pricing problem is independent of the others, their solutions can be obtained in parallel.

A. Layered graph

In order to model the chaining constraint of a demand, we associate to each demand d a layered graph $G^L(d)$ with $|c_d|$ layers where $|c_d|$ denotes the number of VNFs in the chain of the demand. Each layer is a duplicate of the original graph, and the capacities of both nodes and links are shared among layers.

A path on the layered graph starts at layer 0 and ends at layer c_d and corresponds to an assignment of both a path and the locations where functions are being run (the links between layers).

Representing the original graph as a layered graph is a *modeling trick* first proposed in [18]. It allows to simplify the problem by reducing it to a routing problem with shared capacities. This allows a drastic reduction of computation time compared to usual strategies using a large number of binary variables due to the ordering constraints of VNFs in the slice.

B. Master Problem

The Master Problem of the column generation algorithm is described in this section.

Parameters:
- δ_l^p is the number of times the link ℓ appears on path p (considering all the layers in the layered graph).
- $\theta_{i,u}^p = 1$ if node u is used as a VNF on path p on layer i.

Variables:
- $\varphi_p^{d,t} \in [0,1]$ is the amount of flow of demand d on path p at time step t.
- $y_p^{d,t} \in [0,1]$ is the maximum amount of flow of demand d on path p between time step $t-1$ and t.
- $z_{u,f} \in [0,1]$, is equal to 1 if function f is activated on Node u at time step T in the final routing.

We assume an initial configuration is provided with fixed values for $\varphi_p^{d,0}$. The optimization model is written as follows.

Objective: minimize the amount of network resources consumed during the last reconfiguration time step T, which is the sum of the bandwidth used (BW) added to the sum of the costs of the deployed VNFs multiplied by a factor β.

$$\min \sum_{d \in D} \sum_{p \in P_d} \sum_{\ell \in E} \mathrm{BW}_d \, \varphi_p^{d,T} \, \delta_\ell^p + \beta \sum_{u \in V^{\mathrm{VNF}}} \sum_{f \in F} c_{u,f} \, z_{u,f} \quad (2)$$

Note that maximizing the accepted bandwidth in equation (1) implies minimizing the link bandwidth used by the paths (first term in equation (2)); whereas the second term in equation (2) represents the cost of the VNFs deployed at time t in equation (1). Since the actual cost of the bandwidth and the VNFs can vary depending on the network operator, we considered a value of β for which the bandwidth and the VNFs have the same weight in the objective optimization.

Constraints:
One path constraint. For $d \in D$, time step $t \in \{0, ..., T\}$.

$$\sum_{p \in P_d} \varphi_p^{d,t} = 1 \quad (3)$$

Path usage over two consecutive time periods. For $d \in D$, $p \in P_d$, $t \in \{1, ..., T\}$.

$$\varphi_p^{d,t} \leq y_p^{d,t} \text{ and } \varphi_p^{d,t} \leq y_p^{d,t-1} \quad (4)$$

Make Before Break - Node capacity constraints. The capacity of a node u in V is shared between each layer and cannot exceed C_u considering the resources used over two consecutive time periods. Δ is the amount of computational units

required by function $f \in F$ per unit of bandwidth processed. For $u \in V^{\mathrm{VNF}}$, $t \in \{1, ..., T\}$.

$$\sum_{d \in D} \sum_{p \in P_d} \sum_{i=0}^{|c_d|-1} y_p^{d,t} \cdot \theta_{i,u}^p \cdot \mathrm{BW}_d \cdot \Delta_{f_i^{c_d}} \leq C_u \quad (5)$$

Make Before Break - Link capacity constraints. The capacity of a link $\ell \in E$ is shared between each layer and cannot exceed C_ℓ considering the resources used over two consecutive time periods. For $\ell \in E$, $t \in \{1, ..., T\}$,

$$\sum_{d \in D} \sum_{p \in P_d} \mathrm{BW}_d \, y_p^{d,t} \, \delta_\ell^p \leq C_\ell. \quad (6)$$

Function activation. To know which functions are activated on which nodes in the final routing. For $u \in V$, $f \in F$, $d \in D$, $i \in \{0, ..., |c_d| - 1\}$,

$$y_p^{d,T} \, \theta_{i,u}^p \leq z_{u,f_i^{c_d}}. \quad (7)$$

C. Pricing Problem

The pricing problem searches for a possible placement for the slice. Since a reconfiguration can be done in several steps, a pricing problem is launched for each demand, at each time step. The objective of the pricing problem for each demand d at time t is called the reduced cost and is expressed using the equation in [17].

Parameters:
- μ are the dual values of the master's constraints. The number written in upperscript is the reference of the master's constraints.

Variables:
- $\varphi_{\ell,i} \in \{0,1\}$ is the amount of flow on link ℓ in layer i.
- $\alpha_{u,i} \in \{0,1\}$ is the amount of flow on node u in layer i.

Objective: minimize the amount of network resources consumed for the demand d at time t.

$$\min \sum_{\ell \in E} \sum_{i=0}^{|c_d|} \varphi_{\ell,i} \, \mathrm{BW}(1 + \mu_{l,t}^{(6)}) + \mathrm{BW} \sum_{u \in V^{\mathrm{VNF}}} \mu_{u,t}^{(5)} \sum_{i=0}^{|c_d|-1} \Delta_{f_i^{c_d}} \alpha_{u,i}$$
$$- \mu_{d,t}^{(3)} + \beta \sum_{u \in V^{\mathrm{VNF}}} \sum_{f \in F} c_{u,f} \, z_{u,f} \, \mu_{d,u,f}^{(7)} \quad (8)$$

where $\mu_{d,u,f}^{(7)} = 0$ when $t \neq T$, see constraints (7). **Constraints:**
Flow conservation constraints for the demand d. For $u \in V^{\mathrm{VNF}}$.

$$\sum_{\ell \in \omega^+(u)} \varphi_{\ell,0} - \sum_{\ell \in \omega^-(u)} \varphi_{\ell,0} + \alpha_{u,0} = \begin{cases} 1 \text{ if } u = v_s \\ 0 \text{ else} \end{cases} \quad (9)$$

$$\sum_{\ell \in \omega^+(u)} \varphi_{\ell,|c_d|} - \sum_{\ell \in \omega^-(u)} \varphi_{\ell,|c_d|} - \alpha_{u,|c_d|-1} = \begin{cases} -1 \text{ if } u = v_d \\ 0 \text{ else} \end{cases} \quad (10)$$

$$\sum_{\ell \in \omega^+(u)} \varphi_{\ell,i} - \sum_{\ell \in \omega^-(u)} \varphi_{\ell,i} + \alpha_{u,i-1} - \alpha_{u,i-1} = 0$$
$$0 < i < |c_d| \quad (11)$$

Delay constraints. The sum of the link delays Γ_ℓ of the flow must not exceed the delay requirement of demand d.

$$\sum_{i=0}^{|c_d|} \varphi_{\ell,i}\, \Gamma_\ell \leq \gamma_d \tag{12}$$

Function activation. To know which functions are activated on which nodes. For $u \in V^{\text{VNF}}, f \in F$, layer $i \in \{0, ..., |c_d| - 1\}$

$$\alpha_{u,i} \leq z_{u,f_i^{c_d}} \tag{13}$$

Location constraints. A node may be enabled to run only a subset of the virtual network functions. For $u \in V^{\text{VNF}}$, $i \in \{0, ..., |c_d|-1\}$, if the $(i+1)^{\text{th}}$ function of c_d cannot be installed on u, we have

$$\alpha_{u,i} = 0. \tag{14}$$

D. Motivation for `Deep-REC`

Our algorithm reconfigures a given set of network slices from an initial routing and placement of network functions to another solution that improves the usage of the network resources (both in terms of link bandwidth and VNFs). This reconfiguration is done with a make-before-break approach to avoid interruptions of the flows.

In a dynamic scenario, due to the frequent arrival and departure of slices, the network is regularly in a sub-optimal state. Nevertheless, the frequency to run this reconfiguration algorithm was found on a empirical manner. Indeed, works [1], [11] showed that reconfiguring every 15 minutes allowed a good ratio between cost reduction, quality of reconfigurations, acceptance rate and computation time.

A fixed frequency is easy to set up but in practice, at some specific time of a day, reconfiguration may not be needed as traffic remains stable and the network is already in an optimal state. A new reconfiguration at this time won't bring any gain. On the opposite, during high-dynamic traffic period, more frequent reconfigurations may be suitable to maintain an acceptable state of the network with efficient network resource usage. Therefore, a network operator might be interested to adapt the reconfiguration frequencies depending on the congestion of the network, and the nature of the traffic. This is the main goal of this work.

Indeed, even if we use a *make-before-break* reconfiguration model which allows to reconfigure without disrupting the traffic, reconfiguring generates network management costs to migrate VNFs, and to compute and instantiate the intermediate and the new paths [19].

We present in the new section our DRL model named `Deep-REC` to choose when to reconfigure in order to optimally adapt to the evolving network state. Our objective is to maximize the cumulative profit as presented before: the sum of the instantaneous profits p_t (See (1)).

V. Deep Reinforcement Learning (DRL) Algorithm: `Deep-REC`

The reinforcement learning paradigm formalises a discrete time stochastic control process (as our networking problem) where an *agent* interacts with an *environment* (in our case, the network). At each time step t, the *agent* interacts with its *environment* by (i) observing from the *environment* the current state s, (ii) accordingly, taking a decision (an *action*) a, (iii) receiving a reward $r(s,a)$, and, (iv) observing a new state s' (the network has transitioned from s to s'). The agent can repeat this process for a potential infinite number of time steps, giving rise to a *trajectory*. The sum of the discounted rewards over a trajectory from time t, or *discounted return*, is calculated as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

where $\gamma \leq 1$. The expectation of G_t over all possible trajectories initiated at a state s after taking an action a is the so-called *Q-value function* $Q(s,a)$. We aim to take, at each state s, the action a maximizing the *Q-value function*. Then, we need to estimate $Q(s,a)$.

In [20], authors proposed the *Q-learning* algorithm to learn $Q(s,a)$ from a sequence of agent interactions with the environment. Unfortunately, when the state and action spaces are huge, Q-learning needs a prohibitive computation time. To overcome that, Deep Q-learning Network (DQN) [21] makes use of a deep neural network to approximate the *Q-value function* for high-dimensional state-space problems, as our case. Finally, we also opt for DQN since, conversely to other reinforcement learning algorithms, it can learn efficiently from past experiences without introducing bias.

Description. For the implementation, we use the DQN agent from *tf_agents.agents.dqn.dqn_agent*, and the neural network from *tf_agents.networks.q_network* [2]. The network is composed of a pre-processing layer from Keras [22] used for batch normalization and 3 layers of 64, 64 and 2 neurons, respectively. The batch size is 288, which is large enough to properly normalize. We use Adam optimizer with a learning rate of 1e-3, and we update the network every 16 states. The discount factor γ is set to 0.9, a value large enough to show the importance of future actions. We use an epsilon-greedy policy, where ϵ is set to 0.99 and decay to 0 in 200 instances. The replay buffer has a size of 50 instances, and we train the agent on 250 instances.

Context. A 24-hour day consists of 1440 minutes. We decided to discretize it into 288 periods of 5 minutes in order to optimize the training of our agent. It is recalled that the objective is to maximize the profit, while reconfiguring as efficiently as possible. The agent can potentially choose to reconfigure 288 times. To make the agent aware of the implicit trade-off between reconfiguring now or later, an artificial cost per reconfiguration v_R is introduced. Reconfiguring a network will never decrease the profit, but the agent has to learn when a reconfiguration is really worth it.

The agent will then learn the optimal number of reconfigurations to maximize the profit with this artificial cost. This cost can be real (management cost) or it can be fixed to get a given number of reconfigurations per day. The advantage

of this technique compared to having a maximum number of reconfigurations allowed is that allows the agent to make more or less reconfigurations, adapting its behavior to the current period of the day.

State and Action Spaces. The network state can be described based on the next five quantities: (i) the number of minutes since the last reconfiguration ΔT, (ii) the number of slices added since the last reconfiguration λ, (iii) the number of slices released since the last reconfiguration μ, (iv) the current profit p_t (1) and, (v) the current time t. ΔT represent the current allocation oldness, λ and μ estimate the current network load.

The action space consists of two actions: to perform or not a reconfiguration at current time based on the agent decision.

Reward Function. If the agent has chosen not to perform a reconfiguration, the reward is 0. Otherwise, the agent selects the reconfiguration, and two scenarios are possible:

1) *The reconfiguration was worth it.* A reconfiguration at time t is computed. To have a long-term vision, we simulate the network behavior (slices arrivals and departures) with the new network configuration for the next three time slots, which presents a good balance between accuracy and computational overhead(in training, we can simulate the future requests). Finally, we estimate the accumulated profit gained with the reconfiguration as:

$$\Delta p_R = \{\sum_{k=t}^{t+3} p_k | \text{reconf at } t\}$$

2) *The reconfiguration was actually not worth it.* The reward is estimated differently. We suppose that no reconfiguration was performed at t and we also simulate the network behavior (slices arrivals and departures) for the next three time slots. We estimate the accumulated profit gained without the reconfiguration as:

$$\Delta p_{NR} = \{\sum_{k=t}^{t+3} p_k | \text{no reconf at } t\}$$

Finally, we compute the reward as:

$$r = \Delta p_R - \Delta p_{NR} - v_R.$$

We therefore have a positive reward when the profit increase Δp_R (if *reconfiguring* was the good decision) compensates both the profit increase Δp_{NR} (if *not reconfiguring* was the good decision) and the reconfiguration cost.

Training. We are now studying the efficiency of the learning of our agent trained on 250 instances. In the Figure 2(a), the return of the agent on the training environment increases during the 250 trained instances, which implies that it learns to maximize the accumulated rewards on each instance.

We should not reconfigure too often during a day, so in Figure 2(b), we study the variation of the number of reconfigurations made by the agent on each instance. The agent starts by reconfiguring randomly: 1 time out of 2 and thus about 144 times per instance, and it learns that it must reduce

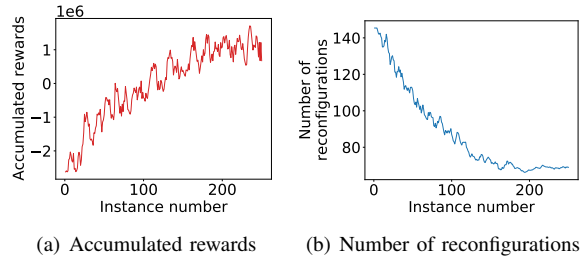(a) Accumulated rewards   (b) Number of reconfigurations

Fig. 2: Learning Curves

the number of reconfigurations to maximize the accumulated reward. When the agent has trained on around 200 instances, the epsilon reaches 0 and the number of reconfigurations converges to around 70 reconfigurations per instance.

VI. DATA SET

Topology. We conduct simulations on a real-world topology from SNDlib [23], `ta1` (24 nodes, 55 links), which includes 6 datacenters on which all VNFs can be instantiated. The cost of VNF c_f is equal to the revenue of 2000 times the revenue u of a megabyte served.

Slice demands Each slice is composed of a chain of up to 5 VNFs, requires a specific amount of bandwidth, and has latency constraints. We consider four different types of demands corresponding to four services: Video Streaming, Web Service, VoIP, and online gaming. The characteristics of each service are reported in Table I and have been already used by [24]. The bandwidth usage was chosen according to the distribution of Internet traffic described in [25]. The latency requirements are expressed in milliseconds and represent the maximum delay between the source and destination.

Each minute, 1 to 5 slice requests arrive (uniform random distribution) and slices that have reached the end of their life are removed from the network. By varying the lifetime of the slices, we can vary the maximum number of slices present at the same time on the network and so that the load on the network follows a curve representing a real distribution of traffic measured on a dedicated network operator. We divided this traffic in five different periods, where D1 is a low-traffic period, and in D5, the network is highly congested. There are between 30 and 180 slices present at each moment on the network and in 24 hours, there are about 4320 arrivals of slices.

Reconfiguration Cost. To train `Deep-REC`, we define a fixed and artificial cost to the reconfiguration v_R. This cost can be adapted to reconfigure more or less. In our study, it is equal to the cost of deploying a VNF for 15 minutes, which means that a reconfiguration is considered useful if it allows to shut down a VNF for at least 15 minutes. To be usable in practice, a reconfiguration must be done quickly. Thanks to the column generation, we can reduce the computation time of each reconfiguration up to 30 times less [11] without affecting its efficiency.

Slice Types	VNF chain	Latency	bw (Mbps)
Web Service	NAT-FW-TM-WOC-IDPS	10ms	100
Video Streaming	NAT-FW-TM-VOC-IDPS	5ms	256
VoIP	NAT-FW-TM-FW-NAT	3.5ms	64
Online Gaming	NAT-FW-VOC-WOC-IDPS	2.5ms	50

TABLE I: Characteristics of network slices

VII. NUMERICAL RESULTS

We compare the results obtained with three solutions in this section:

- `No-REC`: the slices are added in and removed from the network over time, and no reconfiguration is performed,
- `REC-15`: the reconfiguration is carried out every 15 minutes using a *make-before-break* strategy,
- `Deep-REC`: our deep-learning *make-before-break* reconfiguration proposal.

We first show that reconfiguring the network leads to significant gains in terms of profit. We then discuss the importance of selecting the best moments to carry out the reconfigurations, allowing to perform fewer reconfigurations while achieving similar gains.

A. Improved network operational Cost

Figure 3 presents the network cost ((second term in equation (2))) per megabyte of data sent over the network throughout the day. We observe that the costs achieved by `REC-15` and `Deep-REC` are very similar with a clear improvement compared to `No-REC`: `REC-15` allows an improvement of 36.82% when `Deep-REC` performs a little better with 38.05%.

We also see that the cost is rather stable throughout the day, while with `No-REC` the cost increases strongly during low-congestion period (periods D1 and D2, between 2am and 7am). The global cost improvement through a day (not presented in this paper due to lack of space) of `REC-15` is 34.18% versus 35.55% with `Deep-REC`.

Figure 4 presents the network operational cost for the three strategies in terms of VNF costs (second term in equation (2)). This shows that both `Deep-REC` and `REC-15` reduce the network operational cost compared to `No-REC`. This justifies the necessity of reconfiguring. Moreover, the two reconfiguring strategies are comparable for this parameter.

B. Improved Profit and link utilisation

Figure 5 shows the achieved profit, whose maximization is the objective of the reconfiguration: `Deep-REC` and `REC-15` have similar performance and improve the profit compared to `No-REC`. Indeed, the profit improvement of `REC-15` is 32.75% versus 32.53% for `Deep-REC`.

Finally, Figure 6 shows that even when minimizing VNF costs, reconfiguring the network does not lead to an increase of congestion: we observe a slightly higher utilization of links during periods D1-D2, but when the network is heavily loaded (periods D4-D5), there is a reduction of the congestion of the network.

As a conclusion, reconfiguring the network reduces congestion while reducing costs. Moreover, we validate with these results the performance of `Deep-REC` as it leads to similar
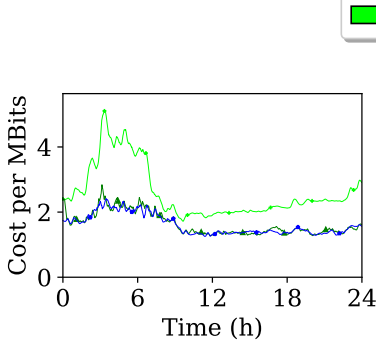
No-REC   REC-15   Deep-REC
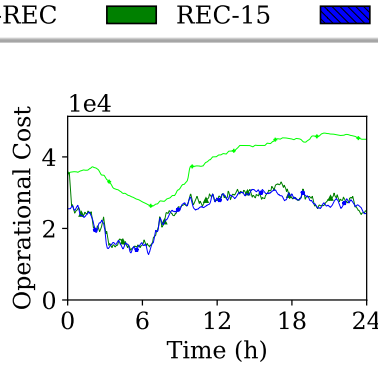
Fig. 3: Cost per MB
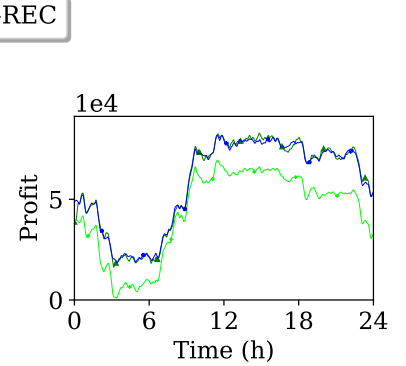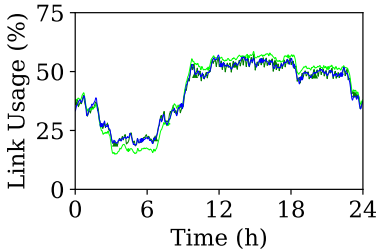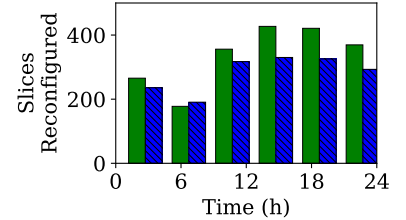
Fig. 4: Operational Cost

Fig. 5: Profit

Fig. 6: Percentage of links capacity used
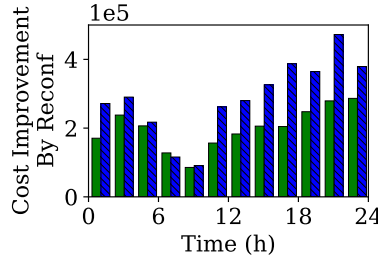
Fig. 7: Cost gain per Reconf

Fig. 8: Number of slices modified

profit as a periodic reconfiguration strategy such as `REC-15`. We show in the following that `Deep-REC`, by performing reconfigurations at the ideal moment, achieves this efficiency while reducing the total number of reconfigurations through a day compared to a regular and fixed reconfiguration strategy such as `REC-15`.

C. Number of reconfigurations

Figure 7 presents the cost improvement divided by the number of reconfigurations over periods of two hours. This shows that `Deep-REC` performs more efficient reconfigurations than `REC-15`. Each reconfiguration leads to a better improvement in terms of the network costs.

Figure 8 presents the number of slices modified during the reconfiguration. Our algorithm `Deep-REC` modified approximately 20% less slices than `REC-15`, and thus there is less impact on the network (less modifications, less computation).

Finally, Figure 9 shows the distribution of the number of reconfigurations during a day over two-hour periods. The green line on the figure represents `REC-15` which does a constant number of reconfiguration, namely 8 (a reconfiguration every 15-minutes). In contrast, `Deep-REC` adapts its actions to the network load and does not carry out reconfigurations when they are not necessary, leading to a reduction of their number during the majority of the day, see the period between 10am and 4pm. Moreover, `Deep-REC` performs more reconfigurations than `REC-15` during the ascending phase (between D1 and D5) in order to react to the rapid change of the network and, thus, to maintain a good profit. With 96 reconfigurations during a day (against 73.2 in average for `Deep-REC`), `REC-15` has 31.15% more reconfigurations, for only 0.22% profit improvement.
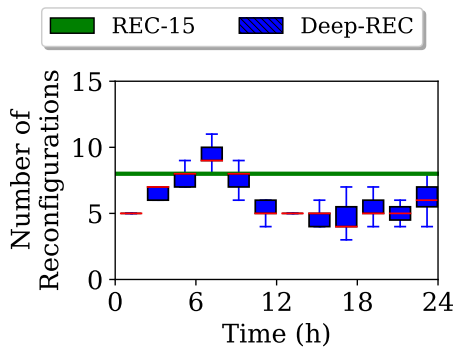

Fig. 9: Reconfiguration distribution for `Deep-REC` compared to `REC-15`.

VIII. Conclusion

We presented in this paper a DRL strategy to carry out an efficient reconfiguration of network slices with dynamic network demands. Our proposal, `Deep-REC`, chooses adequately the best time to reconfigure, it reconfigures few times during low-congestion periods compared to a fixed-frequency reconfiguration strategy. Moreover, when the network is highly congested, `Deep-REC` adapts his behavior, and reconfigures more in order to maximize the network profit. Finally, `Deep-REC` reduces by almost a quarter the number of reconfigurations needed over a day, while achieving similar performances in terms of network operational costs, achieved profit, and congestion of the network.

As a future work, we plan to extend the simulations of `Deep-REC` for several networks and to study the efficiency of our proposal by experimentation on a real platform.

References

[1] A. Gausseran *et al.*, "Be scalable and rescue my slices during reconfiguration," in *IEEE ICC*, 2020.

[2] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[3] N. Huin *et al.*, "Optimal network service chain provisioning," *IEEE/ACM Transactions on Networking*, 2018.

[4] A. Tomassilli *et al.*, "Provably efficient algorithms for placement of service function chains with ordering constraints," in *IEEE INFOCOM*, 2018.

[5] X. Cheng *et al.*, "Safeguard network slicing in 5g: A learning augmented optimization approach," *IEEE JSAC*, 2020.

[6] D. Harutyunyan *et al.*, "Orchestrating end-to-end slices in 5g networks," in *15th International Conference on Network and Service Management (CNSM)*, 2019.

[7] S. Sharma *et al.*, "Dynamic network slicing using utility algorithms and stochastic optimization," in *IEEE HPSR*, 2020.

[8] G. Wang *et al.*, "Reconfiguration in network slicing—optimizing the profit and performance," *IEEE TNSM*, 2019.

[9] M. Pozza *et al.*, "On reconfiguring 5g network slices," *IEEE JSAC*, 2020.

[10] A. Gausseran *et al.*, "Don't interrupt me when you reconfigure my service function chains," *Computer Communications*, 2021.

[11] A. Gausseran *et al.*, "Be Scalable and Rescue My Slices During Reconfiguration," *The Computer Journal*, vol. 64, no. 10, pp. 1584–1599, 08 2021. [Online]. Available: https://doi.org/10.1093/comjnl/bxab108

[12] Y. Liu *et al.*, "Network function migration in softwarization based networks with mobile edge computing," in *IEEE ICC*, 2020.

[13] S. Troia *et al.*, "Reinforcement learning for service function chain reconfiguration in nfv-sdn metro-core optical networks," *IEEE Access*, 2019.

[14] F. Wei *et al.*, "Network slice reconfiguration by exploiting deep reinforcement learning with large action space," *IEEE Transactions on Network and Service Management*, 2020.

[15] W. Guan *et al.*, "Slice reconfiguration based on demand prediction with dueling deep reinforcement learning," in *IEEE GLOBECOM*, 2020.

[16] N. Zhang *et al.*, "Network slicing for service-oriented networks under resource constraints," *IEEE JSAC*, 2017.

[17] G. Desaulniers *et al.*, Eds., *Column Generation*, ser. Springer Books. Springer, December 2005, no. 978-0-387-25486-9. [Online]. Available: https://ideas.repec.org/b/spr/sprbok/978-0-387-25486-9.html

[18] A. Dwaraki *et al.*, "Adaptive service-chain routing for virtual network functions in software-defined networks," in *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, 2016, pp. 32–37.

[19] K. A. Noghani *et al.*, "On the cost-optimality trade-off for service function chain reconfiguration," in *IEEE CloudNet*, 2019.

[20] C. J. Watkins *et al.*, "Q-learning," *Machine learning*, 1992.

[21] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[22] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[23] S. Orlowski *et al.*, "SNDlib 1.0–survivable network design library," *Wiley Networks*, 2010.

[24] M. Savi *et al.*, "Impact of processing costs on service chain placement in network functions virtualization," in *IEEE Conference NFV-SDN*, 2015.

[25] *Cisco Visual Networking Index: Forecast and Methodology, 2014–2019*, CISCO, 2015.