On Using Deep Reinforcement Learning for VNF Forwarding Graphs Placement

Pham Tran Anh Quang, Yassine Hadjadj-Aoul, Abdelkader Outtagarts

On Using Deep Reinforcement Learning for VNF Forwarding Graphs Placement

Pham Tran Anh Quang*, Yassine Hadjadj-Aoul*, Abdelkader Outtagarts†
*Inria, Univ Rennes, CNRS, IRISA, France
Email: quang.pham-tran-anh@inria.fr, yassine.hadjadj-aoul@irisa.fr
†Nokia-Bell Labs, France
Email: abdelkader.outtagarts@nokia-bell-labs.com

***Abstract*—The placement of Virtual Network Function – Forwarding Graphs (VNF-FGs) is one of the fundamental operations in the next generation networks. However, embedding a VNF-FG is a complicated process which has been proved as an NP-hard problem. In this demo, we leverage one of the most advanced approaches in Deep Reinforcement Learning (DRL) in order to efficiently solve the VNF-FG embedding problem. In particular, we have proposed an extension of this approach in order to better explore the space of solutions while making it safer. The experimental platform we have developed, which is based on Mininet and containers, allows us to clearly show the advantage of our approach over existing approaches.**

I. Introduction

With the advent of Network Function Virtualization (NFV), today's network infrastructures are becoming more flexible, allowing them to deploy services almost in real time and on-demand [1], making the concept of network slicing a reality [2].

Resolving the placement of complex services, in the form of a Virtual Network Function-Forwarding Graph (VNF-FG), remains a very difficult problem that can hardly be scaled up [3]. Indeed, the problem is known to be NP-hard, which limits the exact resolution of this kind of problem to very small service instances. Therefore, several heuristics have been developed in the literature [4]. However, although very fast, these approaches often get stuck in local minima and adapt very poorly to the addition of constraints or change of objective. The use of metaheuristics does not solve the problem either. Indeed, the latter do not benefit from past experiences, hence the benefit of using learning strategies [5].

Reinforcement learning is certainly the most appropriate approach among the existing learning techniques, in view of the conditions prevailing in the networks (i.e. variable bit-rate traffic, presence of congestion, unpredictable behavior of certain services, etc.). However, only few approaches exist in this category [5].

In this paper we present a demonstrator of a reinforcement learning technique that combines deep learning and a heuristic, which makes it safer [5]. Indeed, the proposed solution, named EDDPG, not only improves the exploration mechanism suggested for the Deep Deterministic Policy Gradients (DDPG) technique [6], it allows, by combining it with a heuristic, to have at worst the performance of the latter.

The rest of this paper is structured as follows. In Section II, we introduce the architecture of the proposed solution. In Section III, we introduce the experimentation carried out and discuss the obtained results. We conclude our paper in Section IV.

II. The proposed system architecture

A. Overview

The emergence of deep reinforcement learning has unveiled effective solutions to complex problems, in particular in networking. In [1], [2], DRL has been adopted to solve routing problems. Furthermore, it can also be exploited to solve more complicated problem such as virtual network embedding [3]. However, the high dimensional, in both state space and action space of VNF-FG embedding problems, poses a challenge to DRL. Thanks to recent advances in DRL, e.g., DDPG, we can tackle the high dimensional issues. In the previous study [5], we confirmed the capabilities of DRL in coping with VNF-FG embedding problems via simulations. This paper is the continuation in which we implement our solution in an emulation environment based on Mininet[1], with a component-based SDN controller framework (RYU[2]). The DRL agent that we proposed is integrated within an orchestrator designed by the authors.

B. Architecture description

Fig. 1 presents the architecture of the demo. The `MininetController` loads the substrate network stored in the shared memory. Then, it can initialize the substrate network via `MininetRunner` . Within the `Orchestrator` , the `LoadSN` (Load Substrate Network) function loads the substrate network from the shared memory and feed substrate network information to the algorithm. For each iteration, the `GenerateVNFFGs` function generates a set of VNF-FGs which will be loaded into the algorithm via the `LoadVNFFGs` function. The algorithm computes the solution based on the substrate networks and the virtual network request. Then, it writes the solution to the shared memory and it sends a command to `MininetController` to execute the simulation via the `RestClient` . The

[1]http://mininet.org/
[2]https://ryu-sdn.org/

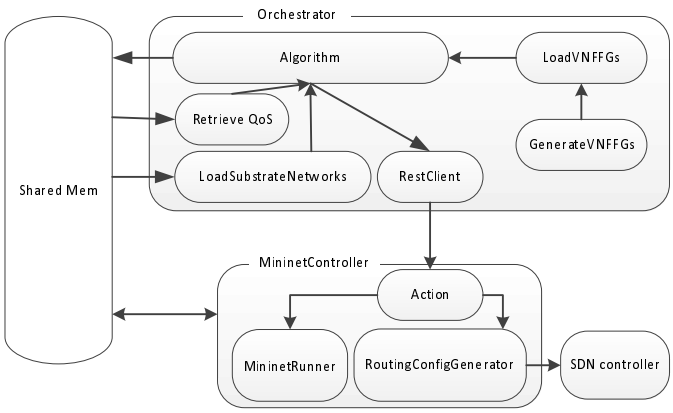

Fig. 1: Demo architecture

`MininetController` receives the command and executes the simulation via the `MininetRunner` and the `RCG` (RoutingConfigGenerator).

1) Mininet Controller:

- `MininetRunner` : Defined in the file mininet_runner.py to enable controlling the mininet emulator.
- `RCG` : Defined in routing_config_generator.py to convert a solution calculated by the algorithm into routing configurations to be sent to the SDN controller.
- `MininetController` : Defined in mininet_controller.py.
  - __init__: Intialize `MininetRunner` module. This module loads the substrate network defined in ../runs/nodes.txt and ../runs/edges.txt
  - initialize: Set up the `MininetRunner` module and initialize `RCG`. In the setup function of `MininetRunner` , the network is created with IP addresses and links. Then, the switches and the connections of the substrate networks are entered as arguments to initialize the RoutingConfigGenerator module.
  - load_data: Load VNFs and VLs configuration to RoutingConfigGenerator.
  - setup_routes: Send routes of the VLs to the SDN controller.
  - send_traffic: Running the Distributed Internet Traffic Generator (D-ITG).
  - del_route: Delete all the routes at the SDN controller.
  - del_data: Delete all the QoS measures.
- `Action` : Defined in mininet_controller.py. It comprises the definitions of the REST interface to control the `MininetController`.

2) SDN Controller: The main file is reset_router.py, which enables to configure the controller via a REST API. It could be deployed by running the following command line:

```
ryu-manager rest\_router.py
```

3) Orchestrator: All functions of the `Orchestrator` are defined in orchestrator.py

- `GenerateVNFFGs` : It generates the VNF-FGs.
- `LoadVNFFGs` : It loads VNF-FGs to the algorithm.
- `LoadSN` : It loads the substrate network to the algorithm.
- `RestClient` : It is in charge of communicating with the `MininetController`.
- `RetrieveQoS` : This function role is to load the QoS results from the shared memory. This allows computing the rewards.
- Algorithm (defined in orchestrator.py, main function): It defines the available algorithms to compute the VNR embedding. `ALG` variable can be $0, 1, 2$ corresponding to FirstFit-Dijkstra algorithm (hueristic), DDPG, and EED-DPG, respectively. We can define a new algorithm in this section.
- Constants (defined in const.py): It defines the following metrics
  - The number of episodes for the training (EPISODE_COUNT).
  - The number of steps in each episode (MAX_STEPS).
  - The minimum and maximum request bandwidth (MIN_REQ_BW, MAX_REQ_BW).
  - Latency, and lossrate of VLs (MIN_REQ_LATENCY, MAX_REQ_LATENCY, MIN_REQ_LOSS, and MAX_REQ_LOSS).
  - The number of VNF-FGs (NUM_VNFFGS), and the minimum and maximum VNFs in each VNF-FG (MIN_VNFS and MAX_VNFS)

III. Experiments

All the modules of our demonstrator have been installed on a laptop, although we can consider putting each module in a separate element. We considered three placement strategies: the First-Fit heuristic, the DDPG algorithm and the proposed EDDPG solution. The First-Fit heuristic consists in placing the virtual nodes, each time in the first available physical node, and then placing the paths using the Dijkstra algorithm.

To assess the performance of the proposed approach, we consider, for the first experiments, the network topology of BtEurope [7] with 24 nodes and 37 full-duplex links. We considered the placement of 10 VNF-FGs, comprising between 2 and 10 VNFs each. For the physical nodes, we considered normalized CPU, RAM and Storage resources in the interval $[0.5, 2.0]$. For physical links we have considered latencies in the interval $[1.0, 10.0]$*ms*[3], and a loss rates in the interval $[1\%, 4\%]$. For the virtual nodes, we considered resources in the interval $[0.1, 1.0]$. For the virtual links we considered latencies in the interval $[10, 100]$*ms* and loss rates in the interval $[1\%, 10\%]$.

Figures 2, 3 and 4 show the results obtained with the different strategies: First-Fit, DDPG and EDDPG, respectively. The reward, in the different figures, represents the percentage of VNF-FGs that have been placed.

Note that a successful placement is when you manage to place all the nodes of a request but also all its links. Note as well that each point in the plots represents a request for

[3]The effective latency results from the placement, it is therefore calculated dynamically.

placement of a batch of VNF-FGs, for which the difficulty of placement may vary, hence the variability of the results obtained.

We can easily notice that the heuristic has an average success rate of 20.6% while DDPG has a success rate of only 17.1%, which reveals the limits of applying a reinforcement learning approach directly, without adapting it to the placement problem. On the other hand, the strategy we proposed has an average success rate of 28.75%, which represents an improvement of 68.1% over DDPG and 39.6% over the heuristic.

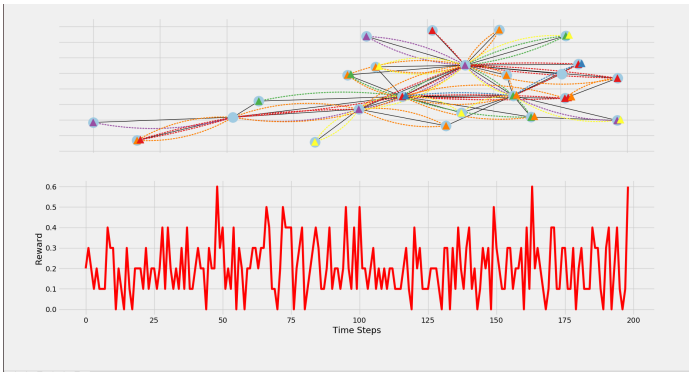

Fig. 2: First-Fit strategy

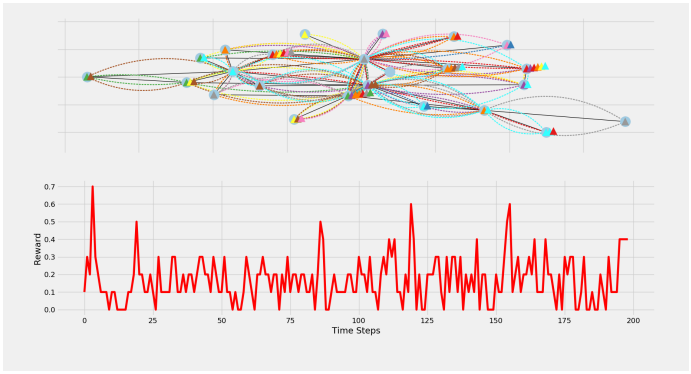

Fig. 3: DDPG strategy

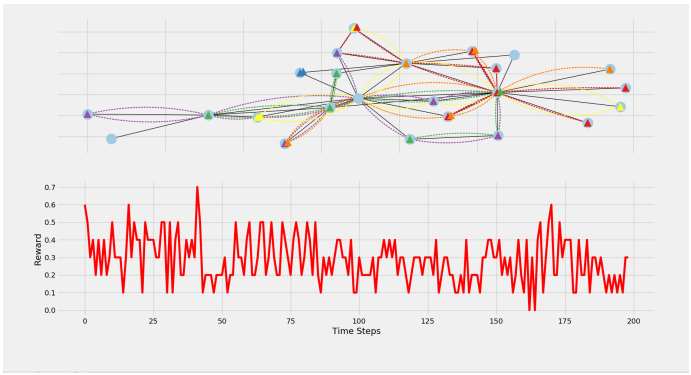

Fig. 4: EDDPG strategy

Figures 2, 3 and 4 also show the placement of virtual nodes and links on the physical network. It is quite easy to see that the number of nodes placed varies from one approach to another. As mentioned above, the number of placed VNFs does not reflect the quality of the placement since a VNF-FG is placed only when all its nodes are placed as well as all its links.

When performing experiments over a larger number of episodes (i.e., 1000 episodes), we observed that the results remained in the same order. In other words, learning converges fairly quickly thanks to the technique we proposed. It is, indeed, easier to learn how to assign weights (i.e. how to make a particular shuffle when assigning virtual nodes), which is then exploited by the heuristic we have proposed, than to learn the general model for the placement. Thus, the methodology we have proposed not only ensures that we have at least the performance of the heuristic right from the start, but also allows to converge towards optimal performance in a very short period of time.

IV. Conclusions

Service placement combined with deep reinforcement learning techniques are certainly part of the technological building blocks to converge towards a completely autonomous network, known as Zero-touch network.

Based on a learning strategy that we have previously proposed, we provide in this paper a demonstrator to illustrate the potential of this type of solution in dealing with service placement. This demonstrator also shows that such a solution can be considered for the control of a real network infrastructure. Indeed, the approach that we have proposed provides guarantees in the quality of the placement, since in the worst case, and from the beginning of the learning process, it allows to obtain the performance of the heuristic.

In our future work, we will consider the use of an even more realistic environment by integrating an orchestrator such as Kubernetes while considering the placement of real network services.

References

[1] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *CoRR*, vol. abs/1709.07080, 2017.

[2] T. A. Q. Pham, Y. Hadjadj-Aoul, and A. Outtagarts, "Deep Reinforcement Learning Based QoS-Aware Routing in Knowledge-Defined Networking," in *Proc. EAI QShine*, Dec. 2019, pp. 14–26.

[3] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "Deepvine: Virtual network embedding with deep reinforcement learning," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2019, pp. 879–885.

[4] D. Li, P. Hong, K. Xue, and J. Pei, "Virtual network function placement and resource optimization in nfv and edge computing enabled networks," *Computer Networks*, vol. 152, pp. 12 – 24, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128618305000

[5] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A deep reinforcement learning approach for vnf forwarding graph embedding," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2019.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[7] The internet topology zoo. [Online]. Available: http://www.topology-zoo.org/dataset.html