A Model for Traffic Forwarding through Service Function Chaining using Deep Reinforcement Learning Techniques

Silvio Romero de Araújo Júnior[1], **Reinaldo A. C. Bianchi**[1]

[1]Centro Universitário da FEI
Av. Humberto de Alencar Castelo Branco, 3972-B – Assunção
São Bernardo do Campo - SP - Brasil, 09850-901

sraj@fei.edu.br, rbianchi@fei.edu.br

Abstract. *The development of new communication networks to offer innovative services has increased the volume of data. With the introduction of Deep Reinforcement Learning and Service Function Chaining architecture, new research opportunities have emerged to propose solutions to the new challenges. This work proposes a model to demonstrate through computational simulations how these techniques can be applied. The model was evaluated using two variations of the Deep Q-Network algorithm over the CIC-Darknet dataset. Results showed that both variations are a promising mechanism to make the networks more autonomous and intelligent.*

1. Introduction

Communication systems such as the Internet have been developing quickly and due to this evolution, the infrastructure, devices, and resources of networked systems have become more complex and heterogeneous. Fifth-generation mobile networks (5G) have been designed to be the key element that will enable the offer of new services and technologies, such as Big Data and the Internet of Things (IoT). However, a deep change in how the networks are designed is necessary, both in the access layer and in the core of the network, to deploy the emergent networks architectures [Li et al. 2018].

To support the 5G networks and help them to deal with new traffic characteristics, such as the dynamic behavior and high volume of data required by new services, some technologies have arisen as candidates: Software Defined Networks (SDN) [Haleplids et al. 2015 and ITU-T 2014], Network Function Virtualization (NFV) [ETSI ISG 2014] and Service Function Chaining (SFC) [Fu et al. 2019].

The SFC technology allows the flow of information to travel through Virtual Network Functions (VNFs) which offers better flexibility in terms of resources usage [Li et al. 2020]. The SFC architecture definitions are found in RFC7665 [Halpern and Pignataro 2015] and in the ITU recommendation Y.2242 [ITU-T 2018]. However, such technologies bring with them some challenges for their implementation, such as: the efficient use of the radio spectrum, the use of hardware computing resources that support the virtualized functions and new security threats. Therefore, it is crucial in this new scenario that efforts are directed towards investigating how to use resources efficiently and intelligently [Li et al. 2018].

Deep Reinforcement Learning (DRL) methods have been used successfully to solve the most diverse computational problems [Luong et al. 2019]. Therefore, it is

natural that they are also used in environments related to communication systems [Fu et al. 2019]. Given the complexity and size of the networks and the size of the volume of traffic generated, defining the parameters used by Reinforcement Learning (RL), such as state spaces and actions can become tough, as the RL may not find the policy ideal in a reasonable time, which limits its application in the dynamic environment of the new networks. The combination of Deep Learning with Reinforcement Learning helps to overcome these limitations.

Part of the process for achieving the goals of optimizing computing resources and responding to security threats is to identify and handling network traffic intelligently. This work aims to investigate how Deep Reinforcement Learning techniques can be combined with the Service function Chaining architecture in order to provide a mechanism for identifying and routing traffic based on profiles by forwarding the packets to the correct path according to decisions made by a learning agent. The main contribution of this paper is the implementation of a model to evaluate the application of these techniques with real network traffic.

This work is organized as follows: section 2 presents the theoretical background of Service Function Chaining and Deep Reinforcement Learning. In section 3 some related works are discussed. In section 4 the proposed architecture and methodology are described. Section 5 presents our experiments and results. And Section 6 presents conclusions and suggestions for future works.

2. Background

The main concepts of Service Function Chaining architecture (its components and terminology) and Deep Reinforcement Learning are presented in this section.

2.1. Service Function Chaining (SFC)

In order to create a virtual chain of network functions for packet forwarding the SFC architecture establishes a mechanism that orders the Virtual Network Functions (firewalls, load balancers and routers) to form the SFC or a chain of service functions. SDN and NFV technologies are the pillars that allow the deployment of SFC to offer a method of allocating network resources efficiently and flexibly and using DRL methods allows it to do so dynamically without the need for manual configuration. This provides advantageous options for new networks, such as 5G systems and virtual network services in data centers [Zhang et al. 2018]. Efforts by standardization bodies such as the Internet Engineering Task Force (IETF) and the International Telecommunications Union (ITU) have been made to propose the concepts of how to implement the SFC [Halpern and Pignataro 2015 and ITU-T 2018].

Halpern and Pignataro [2015] described in RFC 7665 the components and functions, leaving it up to the implementations how they will be done, for example in a single module or separately. Such components can be interconnected using the encapsulation (SFC encapsulation) defined by Quinn et al. [2018] and form a path called SFP (Service Function Path) that establishes the route to be taken by the packets. Figure 1 illustrates a generic topology of the SFC architecture.

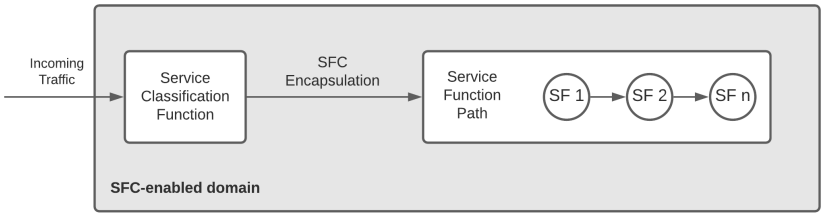

Figure 1. SFC Architecture, adapted from [Halpern and Pignataro 2015]

According to RFC 7665, 4 logical components form the SFC architecture:

- Classifiers (Service Classification Function): entry point into the SFC domain. Its function is to establish the criteria for classifying packets based on policies, the traffic that complies with such policies has its data encapsulated (VLAN tags, MPLS labels, etc.) and then forwarded to the SFF (Service Function Forwarder) function. Such classification can be done by criteria such as 5-tuple (source/destination IP addresses, source/destination ports and transport protocol) or with Deep Packet Inspection (DPI).

- Forwarders (SFFs): after defining the classification policy, the SFF directs traffic to the Service Functions (SFs) connected according to the assigned encapsulation, besides processing the traffic received from the SFs. You can also change the information on packages so that they are reclassified or even close the SFP.

- Service Functions (SFs): a component that can be implemented in hardware or software, whose function is to handle the packets received by the forwarding function (SFF). Such functions can be performed at any layer of the protocol stack.

- SFC Proxies: responsible for communication between SFC domains and those that are not part of this architecture (legacy networks) allowing interoperability between them, for that they remove or add header information according to the traffic direction (adds in the direction from non-SFC to SFC and remove in the opposite direction).

2.2. Deep Reinforcement Learning (DRL)

In Reinforcement Learning (RL) an autonomous agent learns to make the best decisions based on the interaction with the environment. This is done through random experimentation (trial and error) given a set of actions (***a***), such actions change the state (***s***) of the environment, and based on their consequence, the agent receives a reward or penalty (***r***). This illustrated in Figure 2 [Nguyen and Reddi 2019].

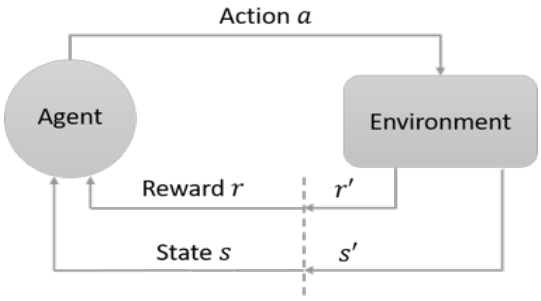

Figure 2. Reinforcement Learning [Nguyen and Reddi 2019]

An RL problem can be described as a Markov Decision Process (MDP) [Arulkumaran et al. 2017]. An MDP can be defined as a tuple (S, A, T, R), where: S is a set of states, A is a set of actions, T is the transition probability function that maps the state-action pair at time t in a state distribution at time $t + 1$ and R is the function that gives the cost (or reward) for making a decision $a \in A$ when the process is in a state $s \in S$ [Pellegrini and Wainer 2007]. The agent's objective in RL is to learn a policy (π) that maximizes rewards or minimizes accumulated penalties after choosing a sequence of actions. In general, policy (π) is the mapping of states to a distribution of probabilities about actions, $\pi : S \rightarrow p(A = a / S)$. A commonly used algorithm in RL is the Q-Learning, in which a value known as Q-value = $Q(s, a)$ is iteratively estimated for each pair (s, a), its objective is to maximize the accumulated reward after the sequence of actions, equation (1) describes this process, in which $\gamma \in [0, 1]$ is the discount factor [Nguyen and Reddi 2019]:

$$Q(s_t, a_t) = E[r_{t+1} + \gamma . r_{t+2} + \gamma^2 . r_{t+3} + \ldots | s_t, a_t] \qquad (1)$$

Although Q-learning has been shown to be important in several RL applications, it needs to use a table called a Q-table to store the expected rewards (Q-values) of actions given a set of states. As the action/state pair increases, the Q table also increases, which requires greater amounts of memory and processing, which can make Q-learning application prohibitive in some real-world scenarios. This feature is known as a "state-space explosion" [Restuccia and Melody 2020].

To allow Reinforcement Learning to deal with this limitation, one of the main characteristics of Deep Learning (DL) is used, which is able to find low-dimensional data representations from high-dimensional data through the approximation of learning functions. Therefore, the combination of DL and RL techniques leveraged research in the area known as Deep Reinforcement Learning (DRL). With DRL, problems considered intractable in RL can now be solved through Deep Neural Networks (DNNs). In the case of Q-learning, instead of a Q-table, a DNN is used in order to efficiently approximate the value $Q(s, a)$, this network is known as Deep Q-Network (DQN) [Mnih et al . 2015].

3. Related Work

The topic about applications of Deep Reinforcement Learning in the context of emerging networks, such as 5G networks, has been the subject of research by the academic community. The SFC architecture with the ability to dynamically adapt and provision resources has been applied in research environments in order to make such networks more autonomous. This section briefly discusses some of these efforts.

Sethi et al. [2020] proposed an adaptive IDS (Intrusion Detection System) architecture in the cloud based on Deep Reinforcement Learning, which tries to deal with some limitations of the systems studied so far: scalability and autonomy to self-adapt. The architecture consists of three components: the devices are in the host network, the architecture's intelligence is in the agent network responsible for executing the algorithm based on Deep Q-Network [Mnih et al. 2015], the administrator network

detects the state of the system in the presence of malicious activities and interacts with the agent network. The dataset by Moustafa and Slay [2015] was used to simulate attacks and, through a Python script, the intrusion detection model was implemented. The experiments balanced accuracy and a low false positive rate. The Service Function Chaining technology was not addressed, this work has chosen it because it offers a better flexibility from the point of view of packet classification and forwarding.

The NFQ (Neural Fitted Q-learning) algorithm [Riedmiller 2005] was used by Akbari et al. [2020] to implement an autonomous threat mitigation framework in SDN (Software Defined Networks). Threat mitigation was formulated as a Reinforcement Learning issue. The prototype has three modules: the SDN infrastructure where is located the *network observer*, responsible for traffic monitoring and for providing information about the state of the network for *autonomous management* module. In this module is the solution's intelligence made up by the Reinforcement Learning agent that uses API (Application Programing Interface) commands to establish policies to be implemented in the network via SDN controller. Such policies define the path that traffic from a given host must take. The third module called *host behavior profiling* accommodates both benign and malicious traffic generating devices. The SDN paradigm adopted by the authors can be adapted for use in the environment of this research through the Service Function Chaining.

Ning et al. [2020] implemented a network model to solve the Service Function Chaining optimization problem in a multiservice environment, a method based on Deep Reinforcement Learning to obtain optimized operations was introduced. The network was modeled as a directed graph, where the Virtual Network Functions (VNFs) are the vertices and the network links compose the edges, this is important to adapt the objective functions. To verify the experiment's results, the data obtained by the author were compared with those resulting from the Mixed-Integer Linear Programming (MILP) method [Taskin, 2008]. The solution based on Deep Reinforcement Learning achieved an almost ideal performance when compared to MILP and proved better when compared with traditional solutions (without learning mechanisms). The work did not address the topic of packet classification, an opportunity that was used in this research.

An intelligent provisioning framework for VNFs was proposed by He et al. [2020] to optimize resource scheduling in a cloud environment. A traffic identification mechanism based on Deep Learning was combined with a Virtual Network Functions path selection based on Deep Reinforcement Learning. The simulation was performed in two phases: the first to assess the accuracy of the traffic identification, where a dataset with 8 types of traffic classes was used. The second part consisted of VNFs' provisioning using PyTorch to build the test architecture. It was demonstrated that the proposed solution obtained a superior performance when compared to other Deep Reinforcement Learning algorithms, improving the Quality of Service (QoS) parameters of the users. The concept of VNF path selection is widely used in Service Function Chaining environments, reason why it is used in this work

A limitation in applying Deep Reinforcement Learning in dynamic Service Function Chaining implementation according to Xiao et al. [2019] is to assume that network resource requests are predetermined, regardless of real-time variations. The authors' proposal consists of NFVDeep, an adaptable and online approach based on the Policy Gradient [Peters and Bagnell 2010] that treats dynamic network state transitions

as a Markov Decision Process (MDP). In this way, the SFC requests are automatically implemented according to their QoS characteristics. The results showed that the architecture significantly outperforms the solutions found in other research. Although the focus is on QoS parameters, with correct adaptation the concepts can also be applied to the issue of traffic routing using SFC.

4. Methodology

In this section we describe the dataset used to traffic generation and the proposed model to simulate the network based on the Service Function Chaining architecture.

4.1. Darknet Dataset Overview

The dataset for this research was the CIC-Darknet2020 from Lashkari et al. [2020]. The dataset consists of 141530 IP packets (raw data) with several types of traffic. There are 85 columns representing the feature variables, but we used only the column with the destination port (dst.port), the application traffic that should be assigned an SFC path according to the policies received as actions from the *Agent* module by the *Classifier* module. Packets with dst.port = 0 or dst.port = 1 were deleted (data cleaning), since they do not represent a practical application. From these applications, three profiles were created according to the Table 1 that should be assigned to an SFC. In this study we followed the process as shown in the Figure 3 to handle the dataset. It was selected 10000 packets randomly for the experiments each time we ran the simulations.

Figure 3. Dataset Handling

Table 1. Traffic Type per SFC

Traffic Type	Port Number	SFC (Traffic Profile)
HTTP/HTTPS	80/443	1
FTP/SSH/Telnet/DNS/NTP/SNMP	21/22/23/53/123/161	2
Other	Other	3

4.2. Proposed Model

The proposed model is shown in Figure 4. It consists of four modules: **1-Traffic Generator**, this module is responsible to simulate an access network that sends Internet Protocol (IP) traffic to the core network (module 2 and module 3). Here, we used the Darknet dataset, describe in the previous session. **2-Classifier**, this module receives the traffic generated by the *Traffic Generator* module and assigns an SFC creating a tuple (IP destination port, SFC) where the SFC must be assigned according to the Table 1. This tuple is sent to the next module, **3-Monitoring Module**, this one has two sub-modules: *Packet Processing* and *Get Reward*, the first one calculates the percentage of

packets processed successfully based on the SFC assigned by the module 2 and the later one calculates the rewards. The module 3 is also responsible to simulate the VNFs (Service Functions – SFs) that receives and processes the packet sent by the *Classifier*. Modules 1, 2 and 3 are the environment for the Deep Reinforcement Learning.

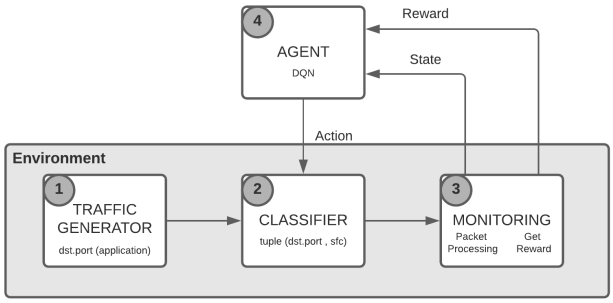

Figure 4. Proposed Model

The percentage calculated by the *Monitoring* module is the state space (observations) in terms of the Deep Reinforcement Learning environment and consists of an array with two elements, percentage of packet processing successfully in SFC1 and SFC2 [***SFC1%, SFC2%***], we discarded the information for SFC3 just for the sake of simplicity, since most part of the application types is relevant to SFC1 and SFC2. And the reward mechanism is shown in (2):

Table 2. Reward Mechanism

Parameter	Reward Value
If (SFC1% or SFC2%) = 100%	+2
If next state value after action is higher than actual state	+1
If next state value after action is lower than actual state	-1
If packets that supposed to be sent to SFC1 or SFC2 were sent to SFC3	-5

The information (states and rewards) provided by module 3 is sent to the module **4-Agent Module**. This one is the agent from Deep Reinforcement Learning perspective which function is to run the Deep Q-Network (DQN) algorithms. It calculates and sends the action to the *Classifier* module in order to move the packet to a new SFC. Here the action space is defined as shown in the Table 3.

Table 3. Action Table

Action Value	Action Meaning
0	Move packet to SFC1
1	Move packet to SFC2
2	Move packet to SFC3

From here a new episode is started, the *Classifier* sends the new tuple (dst.port, SFC) to the *Monitoring* module, this module calculates the percentage as follow:

$$SFC\% = (packet\_ok / (packet\_ok + packet\_nok)) * 100 \qquad (2)$$

Where *packet_ok* is the correct packet in the correct SFC, and *packet_nok* is the packet in wrong SFC. This percentage is sent as an array [*SFC1%, SFC2%*] to the *Agent* module. The *Monitoring* module send to the *Agent* rewards evaluated for each step. All modules are running as Python scripts. We used the Python (version 3.7), OpenAI Gym as the tool for the Agent, PyTorch (release 1.9), Matplotlib (version 3.4.2) and Pandas (version 1.3.0).

5. Evaluation

To evaluate the solution's performance, we used a Python application with the modules described in Section 4. It was executed experiments with two variations of the Deep Q-Network algorithm with some improvements to the original DQN as suggested by Hasselt et al. [2015], the Double DQN, and as suggested by Wang et al. [2015], the Dueling DQN. The hardware set up used was a server with 2 Intel X5650 processors (6 cores each, 2.66GHz), 32 GB RAM memory and a NVDIA card GeForce GTX 1050Ti (Graphical Processor Unit – GPU), running Ubuntu 20.04 LTS.

For each algorithm, it was executed 1000 episodes with 10000 steps each: this is the number of packets used for sampling from the dataset. We repeated the experiments 5 times. Due to the fact they showed similar results, only one graph for each algorithm is presented. That means the solution selects an SFC (it takes action) for each packet based on the policy decision evaluated by the agent per step. The *Packet Processing* module calculates the percentage of successfully assigned pairs (dst.port, SFC number) and sends as an array [*SFC1%, SFC2%*] to the agent, and it also evaluates the rewards to inform the agent. The hyperparameters for the DQN algorithms are shown in the Table 4.

Table 4. Hyperparameters Table

Parameter	Value
Hidden Layers	1
Neuron (Hidden Layer)	50
Discount Factor (gamma)	0.9
Learning Rate	0.01
Exploration Rate (epsilon)	1
Epsilon Decay	0.99
Minimum Epsilon	0.1
Batch or Replay size	20

The rewards received from the environment for each episode during the agent training are shown in Figure 5. The points in the graphic point out the sum of rewards received for each episode.

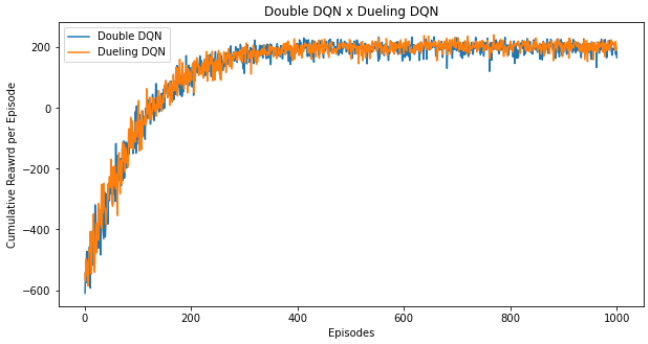

Figure 5. Rewards according to training episodes

We can conclude that the agent learns over the episodes, selecting the correct SFC path according to the traffic type. The shape of the figure is due to the randomness of the traffic in the training environment. It is also possible to check that for the last 50 episodes (some episodes after the agent converged and started to show stability), the Double DQN showed a similar performance as the Dueling DQN, shown in Figure 6.

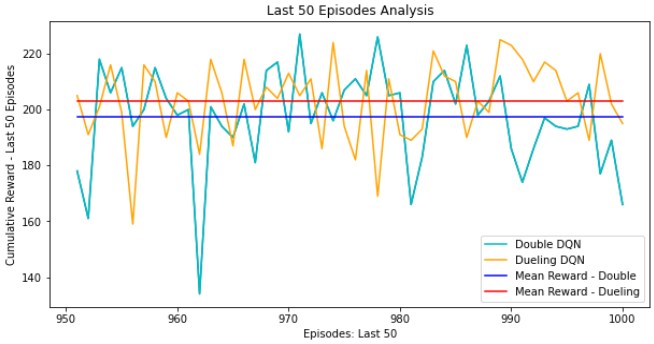

Figure 6. Comparison for the last 50 episodes

Table 5 shows the average and standard deviation values for each algorithm for the last 50 episodes. It demonstrates that after the convergence both algorithms have similar behavior for the suggested scenario (10000 packets and 3 different SFCs). Student T-test [Student 1908] can be used to compare if the result of algorithms is statistically different or not. The t-value computed for the results in Table 5 is 1.79, which gives a p-Value of 0.145. As the p-Value is larger than 0.05 (the threshold for a 95% confidence interval), we can state that both results are statistically the same, confirming the null hypothesis.

Table 5. Values for the last 50 episodes

Algorithm	Average	Standard Deviation
Double DQN	197.48	17.621
Dueling DQN	203.20	14.035

To compare how DQN variants are better than a random policy we present the Figure 7.

Figure 7. Comparison Random Policy x Dueling DQN

6. Conclusion and Future Work

In this work, we proposed a model to combine Deep Reinforcement Learning techniques and Service Function Chaining architecture to effectively and intelligently route traffic through a path to reach the correct Virtual Network Function. We did that by deploying a modular Python application that simulates a network environment and used a well-known dataset to simulate the traffic generation with real IP packets.

To evaluate the proposed model, we executed a comparison between two Deep Q-Networks variants (Double and Dueling). The experimental results elucidate that the agent in charge of making the routing decisions can learn how to do that over time (episodes), applying DQN policies to get optimized rewards. Such mechanisms aim to help the networks to be more resilient, intelligent, and secure. Both algorithms presented similar results. However, a possible choice to use in a real scenario would be the Dueling DQN since it has shown promising results in recent research.

For future work, we plan to run the agent in a more real network scenario composed of an SDN controller (OpenDayLight) running on a cloud environment based on Openstack to be compliant with the ETSI Network Functions Virtualization architectural framework. Besides we plan to use Docker containers and Kubernetes to be aligned with the newest technologies. Also, we will try to run the Deep-Q Learning with more hidden layers and other types of Deep Reinforcement Learning algorithms based on Policy Gradient, for example, to replace the random policy.

7. Acknowledgement

We would like to thank the reviewers for their comments and suggestions that certainly helped to improve our paper. The authors acknowledge the São Paulo Research Foundation (FAPESP Grants 2018/25225-9 and 2019/07665-4) for supporting this project.

References

Akbari, I., Tahoun, E., Salahuddin, M.A., Limam, N., and Boutaba, R. (2020). ATMoS: Autonomous Threat Mitigation in SDN using Reinforcement Learning, in IEEE/IFIP Network Operations and Management Symposium, pages 1-9.

Arulkumaran, K., Deisenroth, M.P., Brundage, M., and Bharath, A.A. (2017). Deep Reinforcement Learning: A Brief Survey. IEEE Signal Processing Magazine, 34, pages 26-38.

ETSI ISG. (2014). ETSI GS NFV 002 V1.2.1. Network Functions Virtualisation (NFV); Architectural Framework. ETSI Group Specification.

Fu, X., Yu, F. R., Wang, J., Qi, Q. and Liao, J. (2019). Dynamic Service Function Chain Embedding for NFV-Enabled IoT: A Deep Reinforcement Learning Approach, in IEEE Transactions on Wireless Communications, vol. 19, no. 1, pages 507-519.

Haleplids, E., Pentikousis, K., Denazis, S., Salim, J. H., Meyer, D. and Koufopavlou, O. (2015). Software-Defined Networking (SDN): Layers and Architecture Terminology. RFC 7426, IETF.

Halpern, J. M. and Pignataro, C. (2015). Service Function Chaining (SFC) Architecture. RFC 7665, IETF.

He B., Wang, J., Qi, Q., Sun, H. and Liao, J. (2020). Towards Intelligent Provisioning of Virtualized Network Functions in Cloud of Things: A Deep Reinforcement Learning based Approach, in IEEE Transactions on Cloud Computing, vol., no. 01, pages 1-1.

ITU-T. (2018). Y.2242: Service Function Chain in Mobile Networks. ITU-T, Y series.

ITU-T. (2014). Y.3300: Framework of Software-Defined Networking. ITU-T, Y series.

Lashkari, A.H., Kaur, G., and Rahali A. (2020) "DIDarknet: A Contemporary Approach to Detect and Characterize the Darknet Traffic using Deep Image Learning", 10th International Conference on Communication and Network Security, Tokyo, Japan, pages 1-13.

Li, G., Feng, B., Zhou, H., Zhang, Y., Sood, K., and Yu, S. (2020). Adaptive service function chaining mappings in 5G using deep Q-learning, in Computer Communications, vol. 152, pages 305-315.

Li, R., Zhao, Z., Sun, Q., Chih-lin, I., Yang, C., Chen, X., Zhao M. and Zhang, H. (2018). Deep Reinforcement Learning for Resource Management in Network Slicing, in IEEE Access, vol. 6, pages 74429-74441.

Luong, N.C., Hoang, D.T., Gong, S., Niyato, D., Wang, P., Liang, Y. and Kim, D.I. (2019). Applications of Deep Reinforcement Learning in Communications and Networking: A Survey, in IEEE Communications Surveys & Tutorials, vol. 21, no. 4, pages 3133-3174.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M.A., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning, in Nature, 518, pages 529-533.

Moustafa, N., and Slay, J. (2015). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set), in Military Communications and Information Systems Conference (MilCIS), pages 1-6.

Nguyen, T.T., and Reddi, V.J. (2019). Deep Reinforcement Learning for Cyber Security. ArXiv, abs/1906.05799.

Ning Z., Wang N. and R. Tafazolli. (2020). Deep Reinforcement Learning for NFV-based Service Function Chaining in Multi-Service Networks: Invited Paper, in IEEE 21st International Conference on High Performance Switching and Routing (HPSR), pages 1-6.

Pellegrini, J. and Wainer, J. (2007). Processos de Decisão de Markov: um tutorial. Revista de Informática Teórica e Aplicada; Vol. 14, No 2, páginas 133-179.

Peters, J. and Bagnell, J.A. (2010). Policy Gradient Methods. Encyclopedia of Machine Learning.

Quinn, P., Elzur, U. and Pignataro, C. (2018). Network Service Header (NSH). RFC 8300, IETF.

Restuccia, F., and Melodia, T. (2020). DeepWiERL: Bringing Deep Reinforcement Learning to the Internet of Self-Adaptive Things.

Riedmiller, M. (2005) Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method, in European Conference on Machine Learning (ECML). Springer, pages 317–328.

Sethi, K., Kumar, R., Prajapati, N., and Bera, P. (2020). Deep Reinforcement Learning based Intrusion Detection System for Cloud Infrastructure, in International Conference on Communication Systems and Networks (COMSNETS), pages 1-6.

"Student" Gosset, W. S. (1908). "The probable error of a mean", in Biometrika. 6 (1) pages 1–25.

Taskin, Z.C. (2008). Tutorial Guide to Mixed-Integer Programming Models and Solution Techniques.

Xiao, Y., Zhang, Q., Liu, F., Wang, J., Zhao, M., Zhang, Z., and Zhang, J. (2019). NFVdeep: Adaptive Online Service Function Chain Deployment with Deep Reinforcement Learning. 2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS), pages 1-10.

Zhang, J., Wang, Z., Ma, N., Huang, T., and Liu, Y. (2018). Enabling Efficient Service Function Chaining by Integrating NFV and SDN: Architecture, Challenges and Opportunities, in IEEE Network, vol. 32, no. 6 pages 152-159.