Environment Aware Adaptive Q-Learning to Deploy SFC on Edge Computing

Suman Pandey
POSTECH
Pohang ,South Korea
suman17july@gmail.com

James W. Hong
POSTECH
Pohang ,South Korea
jwkhong@postech.ac.kr

Jae-Hyung Yoo*
POSTECH
Pohang ,South Korea
jhyoo78@postech.ac.kr

***Abstract*—Biggest challenge in deploying Service Function Chain (SFC) in the Edge Computing environment is the lack of resources at the edge. Hence while finding the optimum path for SFC deployment, the resource constraint environment should be observed and incorporated well in deployment scenarios. In this paper, we developed an environment aware adaptive Q-Learning algorithm to find an optimal SFC deployment path in edge computing environment. The available servers are divided into hierarchical network structure with local, neighbor, and datacenter servers to model an edge computing environment. The resource dynamics in the environment is modeled as a state transition probability. We compared the new algorithm with our base case algorithm that solely depends on Q-Learning and doesn't incorporate the state transition probabilities. An intuitive reward function is designed to give maximum reward to complex deployment with minimum delays. We integrated our algorithm with physical testbeds using OpenStack and open source REST APIs. We evaluated SFC deployment on physical testbed using 42 different scenarios by measuring RTT.**

Keywords—SFC, OpenStack, SDN, Edge Computing, Q-Learning, Reinforcement Learning

I. INTRODUCTION

In Network Management research, several models have been proposed to install SFC[1] and VNFs in Datacenter (DC) efficiently [2]. However, in recent years computational resources are positioned near to the end-users through edge cloud [3] and there is a need to devise a mechanism for deploying SFC in an edge computing environment. AT&T has also proposed a Central Office Re-architected as a Datacenter (CORD) infrastructure [4] to transform the edge into an agile service delivery platform. Minimal research has been done to exploit this edge infrastructure to deploy SFC.

In this paper we propose to install SFC across multiple COs in the nearest proximity if resources are not enough at local CO. As per our knowledge there are no standards to install an SFC across multiple COs in edge cloud environment. We propose to install SFC across neighbor COs to reduce the overall latency. When multiple COs are involved in installing a single SFC, it is important to allocate resources intelligently to maximize the utilization and minimize the latency. We modeled our network topology as a hierarchical structure of local, neighbor, and DC servers to simulate the edge computing infrastructure. We have also deployed a physical testbed to evaluate our proof of concept. Our physical testbed topology resembles Multi-access Edge Computing (MEC) scenario.

978-3-903176-31-7

On top of the testbed hardware, we installed OpenStack [5] to manage our servers and finally integrated our SFC deployment algorithm with testbeds. To integrate our machine learning algorithm with testbed we used open-source NI-Mon and NFVO APIs [6].

Our machine learning algorithm for SFC deployment is based on Reinforcement Learning (RL). RL has been proven successful for pathfinding problems in game programming and robotics etc. Recently it is also used for Routing [7] and SFC path selection [8]. We chose the RL based Q-Learning algorithm to deploy SFC in an edge computing environment. This work is an extension of our previous work [9]. Our previous algorithm deployed SFC by selecting the best path with minimum delay using Q-Learning based approach. Q-Learning typically involves an agent, action, states, policies, and rewards. In the Q-Learning algorithm, the environment is the task or simulation and a learning agent interacts with the environment and tries to solve the task.

In our SFC deployment scenario, the environment consists of servers and their capacity. Server capacity is devised in terms of CPU and memory. Moreover, servers are deployed in a hierarchical manner including local, neighbor, and datacenter servers. The agent interacts with servers in a given hierarchy, first with a local edge environment and then with the neighbor environment and finally with the DC environment looking for the appropriate server to deploy the VNFs based on the rewards. To add to this complexity, our environment isn't static, it changes after every VNF deployment.

It is important to opt these environments into the actionable insight in our algorithm. The accuracy of our algorithm increased, by making our agent accountable to environment changes. Hence in this paper, we changed the model to incorporate the resource availability probability of the hostile states and dynamically changing states. We achieved that by introducing a transition probabilities $p(s' \mid s, a)$. In the new model, we dynamically update the state transition probabilities after each action and selectively reduce the action space by removing the hostile states. Our proposed method contribute in the following ways

1. To the best of our knowledge, this is the early research conducted to deploy SFC on the edge computing environment. Edge Computing is realized by dividing the infrastructure into, edge, neighbor, and DC servers.

2. Environment aware adaptive Q-learning agent is designed, where the environment is represented with servers and their available capacity in three hierarchy.
3. The Algorithm is integrated and verified on the physical testbed with the OpenStack management layer. SFC deployment is verified and RTT is measured using traceroute command.

The rest of the paper is organized in the following way. Section II describes related work. Section III elaborates on our proposed network topology. Section IV explains the proposed method and the Q-Learning algorithm. Section V explains our evaluation mechanism. Section VI discusses the results and challenges encountered during this research and Section VII concludes this research with future work.

II. Related Work

We identified a few leading research that applied machine learning method to dynamically deploy SFC. Runyu Shi et al. [11] used RL based Markov Decision Processes (MDP) to dynamically allocate VNFs. The authors used the Bayesian learning method to monitor the historical resource usage to predict future resource reliability. However, they did not consider edge computing. All the resources were deployed at DCs away from users. They showed very promising computation time of 1 to 2 milliseconds but didn't clarify about the SFC length and number of iterations (epochs) required for learning. Jian Sun et al. [12] also used Q-Learning Framework Hybrid Module Algorithm (QFHLM) which was derived from RL. They also didn't consider edge infrastructure, besides their computation time is 1 minute and learning takes 2.25 million iterations. Moreover, their VNF resource model is considering all VNFs with an equal amount of resource requirement (CPU, mem, and bandwidth of 1 unit). These make their algorithm unrealistic. Aris Leivadeas et al. [10] did consider the edge infrastructure, however they relied on Mixed Integer Programming (MIP) formulation solved using a CPLEX Branch-and-Cut search algorithm resulting in high computation time as compare to Q-learning algorithms. This paper extends our previous work [9] by incorporating the environment variables such as state transition probabilities to design an environment aware Q-learning algorithm. The new feature improves our algorithm accuracy and reduces SFC deployment latency significantly. Apart from that, this algorithm is also integrated and tested on the physical testbed with the OpenStack management layer.

III. Network Topology

We implemented and tested our algorithm on a simulated topology as well as on a physical test bed. The simulated topology consists of Top of the Rack (TOR) switches, Leaf switch, Spine switch, Edge, and Core router. Servers are attached to the TOR, and have less capacity at the edge and higher capacity at the core. The latency is modeled based on the number of hops to cross for installing SFC. If the VNF is deployed on the same server as previous VNF then latency will be 0. If its in same TOR latency is 1, same edge will result in latency 3, same neighbor installation will result in latency 7 and DC will result in latency 9. To get the detail of our simulated topology please refer to our previous work [9].

The same model explained in simulated testbed is emulated on a physical testbed with more simplified network structure by omitting TOR, Leaf, Spine switches. Our testbed consists of three edge switch representing three COs, and one core switch connecting these COs with DCs as shown in Figure 1. This testbed is subjected to improvement in the future. Delay is emulated on this test bed using DEMU[15]. It is a software-based network emulator implemented as a Data Plane Development Kit (DPDK) application. The delay between edge and core switch is 5ms and DC and core switch is 12 ms.

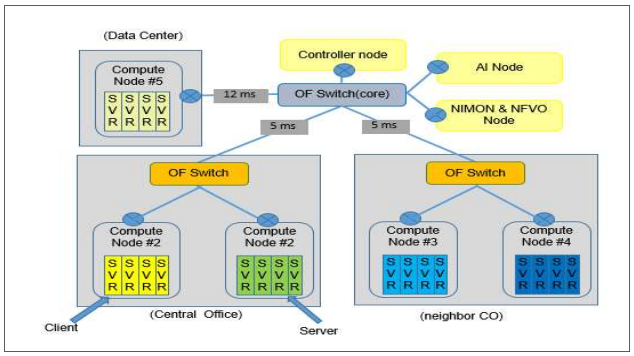

Figure 1. Physical Infrastructure with OpenStack Management Layer

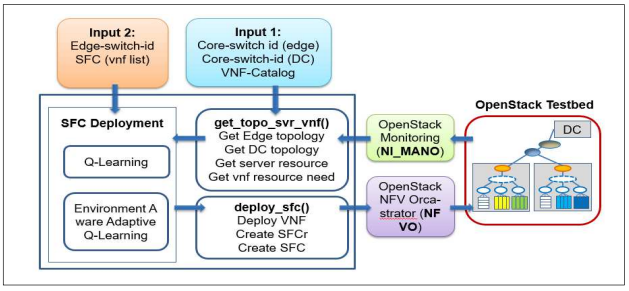

Figure 2. Overall Design

IV. Method

Figure 2 explains our overall design. The algorithm is developed using python 3.7. The algorithm takes two types of the input files. The first input is related to the network topology, where the algorithm takes core-switch-id to which the edge network and DC is connected. Once the algorithm gets the core-switch-id, it requests for edge switch, servers, and links information using REST APIs of the NI-Mon module. Using this information, the algorithm builds edge as well as DC topology. Based on the edge-switch-id the algorithm divides the edge topology into local and neighbor servers. So ultimately the infrastructure is divided into three hierarchy, 1) Local edge servers, 2) Neighboring servers and 3) DC servers. It also gathers the available CPU and memory on each compute node. It also requests for available VNF-flavors, the VNF-flavor objects give CPU and memory needs of each VNFs. The second input is SFC request. The algorithm passes the topology information to the SFC deployment module. The Q-learning agent will learn the best deployment for VNFs in the chain. The algorithm then uses

NFVO open-source APIs to deploy each VNF on the selected server.

A. Q-Learning SFC deployment

Usually, an RL problem is modeled by 4-tuple states (*s*), actions (*a*), state transition probabilities (*pi*), and rewards (*r*). In our implementation states represent the VNFs in SFC. Actions represents the selection of the appropriate server for deployment, state transition probabilities are represented with the resource availability on nodes, and rewards are based on the overall latency and resource demand to deploy the SFC. Q-Learning is a model-free RL learning approach, where learning happens in several *episodes* (epochs). In each *episode*, the agent in state *s* performs an action *a*, receives a reward *r*, and moves to the next state *s_next*. The action value is updated in the Q matrix of size [s × a] with the formula given in Eq. 1.

$$Q(s,a) = (1 - eta) * Q(s,a) + eta * [R + gamma * \max\{Q(s\_next, a\}] \quad (1)$$

Eta represents the learning rate (*Eta* = 0.05). Gamma represents a discount rate (*Gamma* = 0.5) for a given Q-Learning algorithm. Refer to Eq. 1, to find how they are used in Q-Learning. Periodic decay of Q-values is taken care of using gamma (discount factor). Epsilon (*epsilon* = 0.8) represents the *e-greedy* coefficient used later in the algorithm. For each episode, the epsilon is also reduced by a factor of 1.5 (*epsilon* = *epsilon*/1.5). In machine learning, the learner tries to improve the current solution while switching between exploration and exploitation of the solution space. *ε-greedy* selection in addition to greedy selection helps the learner uses a small amount of randomness to explore new solutions.

B. Reward

A good reward functions determines Q-learning algorithm accuracy. We devised an intuitive reward function as shown in Eq. 2. Rewards in SFC deployment scenario would depend on various factors including the E2E number of hops of the SFC path, and SFC length and resource ratio.

In reward equation, *l* represents latency of SFC path and *z* is the length of the SFC. We took a ratio of *l* and *z* to normalize the latency parameter. Reward should be low for high value of *l/z*. Hence we chose an exponentially decaying type of profile. Then this exponential decay equation is multiplied with resource ratio.

Resource ratio is calculated by dividing resource demand and the lowest available resource at the edge. *x* represents the CPU resources and *y* represents memory resource. High resource ratio represents a complex deployment. We assign a higher rewards to the complex deployment by multiplying it to the exponential decay parameter. Hence this function assigns high reward to complex deployment with lower latency. α and β are the constants where α represents a higher value such as 50 to manipulate the reward points giving higher rewards to high resource demand with low latency, and β represents a lower value such as 2, which is used for adjusting the reward point to give very low reward for low resource demand served with high latency.

$$r = \alpha * \left(\frac{x+y}{2}\right) * e^{-\frac{\beta l}{z}} \quad (2)$$

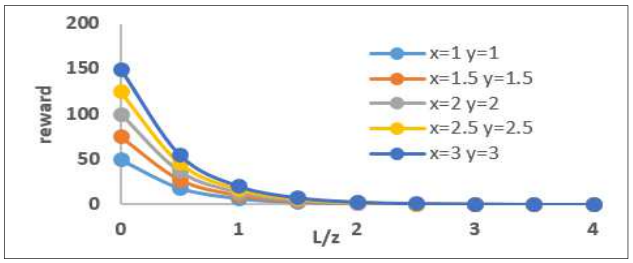

Figure 3. Reward function for given resource ratio, SFC latency and SFC length.

Most of the related work [10] defines a reward as 1 for successful deployment and 0 for unsuccessful deployment, however our reward function is very intuitive and considers several parameters. Figure 3, further clarifies equation behavior for different value of *x*, *y*, *l* and *z*.

C. Environment

We introduced two environment variables *theta* and *Pi* in the already existing algorithm in previous work [9]. These variables represent state transition probability in the Q-learning algorithm. *theta* will store 1 if the server has enough capacity (CPU and memory) to deploy the VNF, or else it will store 0. Later this array is converted into the state transition probability and stored in *Pi* using the Eq. 3.

$$Pi_i = \frac{theta_i}{\sum_{i=0}^{n} theta_i} \quad (3)$$

If the SFC chain is long, very often it will hit a hostile local server. We solved this by reducing the action space by removing the servers with state transition probability as 0 (*Pi = 0*) as shown in Eq. 4. This step is taken at each level, of local, neighbor, and DC servers.

$$S_e = S_e - S_r \mid S_r \in Pi_r = 0 \quad (4)$$

By introducing these variables we were able to solve two important issues with our previous algorithm. 1) An encounter of the hostile servers 2) Incorporating the dynamically changing environment.

V. Evaluation And Discussion

We did evaluation of our method in two phase, 1) Q-learning algorithm evaluation. 2) Deployment evaluation on the testbed.

A. Q-learning algorithm evaluation

We conducted 1200 tests with 100 episodes each. The SFC length varies between 3~8. The SFC could consist of 8 types of VNF flavors with different CPU and memory demands ranging from 1 to 4 CPU and 1 to 8 memory units. The simulated topology for testing the algorithm consists of 9 local servers, 18 neighbor servers, and 27 DC servers, however, the algorithm can support any number of servers. The edge server has the capacity of 6 CPU and 8 memory

units and DC servers have a capacity of 34 CPU and 48 memory units. Figure 4, shows the Q-learning behavior of our algorithm for SFC of length 8 with high, medium, and low resource demands. The environment aware adaptive Q-learning (Figure 4 (a)) shows lower latency and better learning behavior as compared to a simple Q-learning algorithm (Figure 4 (b)). The Latency for medium and high resource demand is much improved.

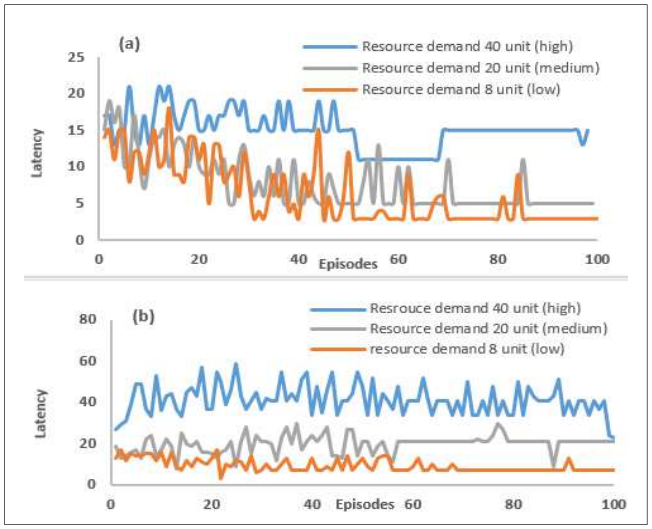

Figure 4. Q-Learning algorithm results of SFC-Length 8 for 100 episodes. (a) Adaptive-Q-Learning, (b) Q-Learning

B. *Deployment testing on the testbed*

In second phase of validation, we deployed our algorithm on a physical testbed with 9 servers, including 2 local, 6 neighbor, and 1 DC servers. We deployed SFC of size 3~8 successfully subjected to the resource availability. SFCs contains several VNFs of different types including firewall, flow monitor, deep packet inspection (DPI), intrusion detection system (IDS), and proxy. To this end, we use open-source software such as iptables, ntopng, nDPI, Suricata, and HAProxy. While creating the VNF instances we choose the appropriate images with these applications installed. The SFC can be tested using iperf, apache-bench, nuttcp or traceroute. These mechanisms gives us response time, and throughput and RTT. We used Traceroute command in this paper to validate if the request is passing through the SFC in the appropriate manner. The traceroute command sends a request through all the VNFs in the chain making it's way to the destination, and it sends back a short message at each intermediate VNFs containing name, address and RTT of the VNF.

Figure 5 represents 42 different deployment scenarios chosen by the Q-learning algorithm based on resource availability. These results validate our proof of concept as RTT is least for local server deployment and highest for the deployment across local-neigh-DC. Figure 5 also shows that the SFC of varying length have similar latency. This is because, the VNFs deployed at the same server have negligible delay. To achieve each deployment scenario, we have increased and decreased load the servers. In reality this result could differ based on other parameters that influence RTT.

We also measured the SFC deployment time for each scenario. We found that deployment time steadily increased with increasing SFC size and is least impacted by the choice of deployment, local, neighbor or DC as shown in Figure 6.

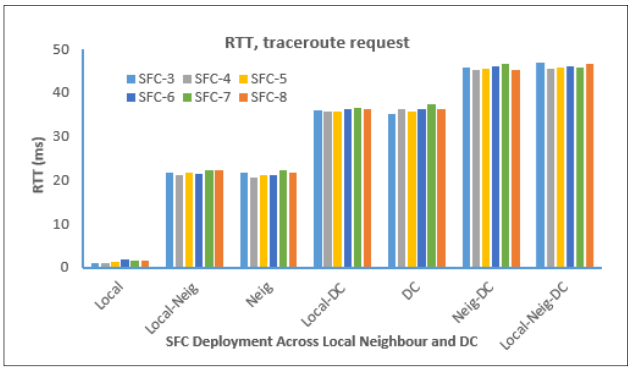

Figure 5. RTT through traceroute request for each SFC deployment across local, neighbor and DC.

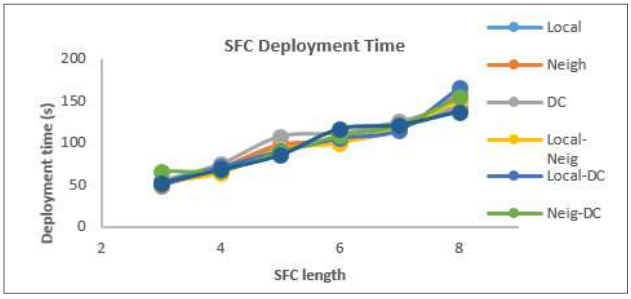

Figure 6. SFC deployment time

VI. Conclusion & Future Work

The dynamic optimal resource allocation of SFC is a critical research topic. In this paper, we have presented a method based on environment aware adaptive Q-Learning to dynamically allocate edge and DC computing resources to SFC. The hierarchical network model based on the local server, neighbor server, and dc server in this paper are more realistic as compared to dynamic graph-based models. We designed an intuitive reward model based on cumulative latency, VNF resource demand, and SFC length. We evaluated this research by running 1200 tests with varying SFC-length, resource demand, and server capacity. In the first phase, we evaluated Q-learning behaviors of the algorithm and in the second phase of evaluation we tested our algorithm on Openstack testbed. We verified the deployment using traceroute command. We measured the RTT for 42 different deployments. Based on the availability of the resources in the local, neighbor and DC the algorithm chooses the best possible deployment. RTT is highest for SFC deployment across local, neighbor and DC. In future, we will also consider network bandwidth and server NIC card capacity as a resource requirement apart from CPU and memory. Another possible consideration is to upgrade the algorithm to Deep Q-Leaning.

ACKNOWLEDGMENT

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (2018-0-00749, Development of Virtual Network Management Technology based on Artificial Intelligence).

REFERENCES

[1] "IETF SFC specification," 2015

[2] S. Jeong, H. Kim, J. H. Yoo, and J. W. Hong, "Machine Learning Based Link State Aware Service Function Chaining," The 20th Asia-Pacific Network Operations and Management Symposium (APNOMS 2019), Matsue, Japan, Sep. 18-20, 2019

[3] "ETSI Mobile Edge Computing (MEC). Framework and Reference Architecture," 2016.

[4] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow "Central Office Re-Architected as a Data Center", IEEE Communications Magazine, October 2016, pp 96~101

[5] "OpenStack Nova," https://opendev.org/openstack/nova

[6] "GitHub Repository for Network Intellegnec," https://github.com/dpnm-ni/

[7] Z. Mammeri, "Reinforcement learning based routing in networks: Review and classification of approaches," IEEE Access, vol. 7, pp. 55916– 55950, 2019.

[8] D. Lee, J. H. Yoo, and J. W Hong , "Q-learning based Service Function Chaining using VNF Resource-aware Reward Model," APNOMS 2020, Daegu, South Korea, Sep. 23-25, 2020

[9] S. Pandey, J. W. Hong, and J. H. Yoo, "Q-Learning based SFC deployment on Edge Computing Environment", APNOMS 2020, Daegu, South Korea, Sep. 23-25, 2020

[10] A. Leivadeas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, "VNF Placement Optimization at the Edge and Cloud," Future Internet, vol. 11, issue. 3, article number 69, 9 March 2019

[11] R. Shi et al., "MDP and Machine Learning-Based Cost-Optimization of Dynamic Resource Allocation for Network Function Virtualization," IEEE International Conference on Services Computing, USA, pp. 65~73, 2015

[12] J. Sun, G. Huang, G. Sun, H. Yu, A. K. Sangaiah and V. Chang , "A Q-Learning-Based Approach for Deploying Dynamic Service Function Chains," Symmetry, vol. 10, issue 11, article number 646, 2018