Machine learning based metro edge computing network resource optimization algorithm for 5G and beyond applications

Luong Quoc Dat
Dept. of Electrical Engineering
Technical University of Eindhoven

***Abstract*—Nowadays, there is a huge demand for optical metro network infrastructure in terms of service expectations for applications in 5G and beyond. In the real-life scenario with multiple applications and different user patterns for each application and different applications' latency requirements, a resource optimization algorithm should be developed to automate the application deployment in the optical network. This project aims to develop a reinforcement learning based algorithm to optimize the deployment of 5G and beyond applications in the optical metro network to satisfy applications' requirements. In the first half part of the thesis, the real applications' deployment, user traffic pattern generation, set up an experiment to capture traffic from each application in the network and simulation software design are already achieved.**

I. Introduction

In recent years, it is undeniable that there is a significant increasing trend in the number of users as well as devices connected to the Internet [1]. This trend leads to huge demands in optical metro networks infrastructure in terms of service expectations. Therefore, optical metro networks should efficiently support a variety of access applications with different dynamic user traffic patterns from various sources such as enterprises, home applications, media streaming, and IoT applications. These applications not only require high bandwidth demands but also require low latency response. For those latency-sensitive applications, cloud computing in data centers is not efficient enough to support because of the long distance between the data center and end-users, causing a long delay from signal propagation leads to high latency. To address this issue, edge computing where the computing servers have been brought to the edge near the end-users is a good solution [2].

Edge computing refers to the enabling technologies that allow computation to be performed at the metro network so that computing happens near data sources. It works on both downstream data on behalf of cloud services and upstream data on behalf of IoT services. Edge computing can be any computing or networking resource residing between data sources and cloud-based data centers. For example, in a metro network, an optical metro node can be equipped with powerful servers as edge computing and handle requests and responses between users and the cloud datacenters. In edge computing, the end device not only consumes data but also produces data. And at the network edge, devices not only request services and information from the cloud but also handle computing tasks including processing, storage, caching. Edge computing could yield many benefits. The most important benefit is reducing application latency. For instance, researchers built a proof-of-concept platform to run a face recognition application in [3], and the response time is reduced from 900 to 169 ms by moving computation from cloud to the edge. Researchers in [4] also have shown that using edge computing to offload computing tasks for wearable cognitive-assistance systems improves response times by between 80 and 200 ms and reduces energy consumption by 30 to 40 percent.

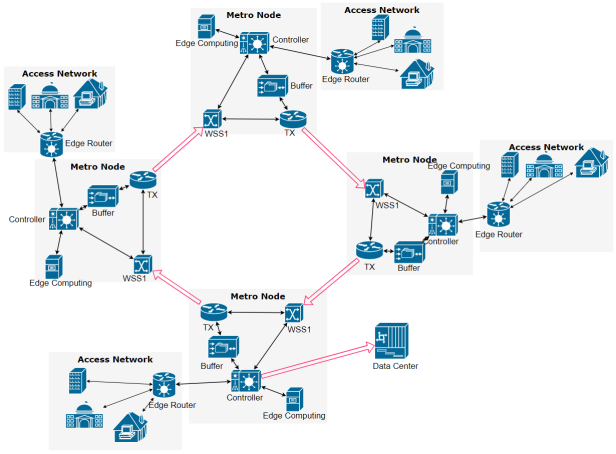

Fig. 1. Metro Ring Network

In the TU/e lab, there is an optical metro ring network that has already been set up as we can see from Figure 1, the ring optical metro network consists of 4 nodes connected by 10Gbps optical links. In a real-life scenario, the traffic that comes from the access network will be aggregated to the metro network. There are also optical links to the data center that have been set up in the lab.

Bitao *et. al.* in [5] demonstrated serving new applications in optical metro access ring network with edge computing resource using software defined networking (SDN) reconfiguration and network slicing (NS). The result shows that the latency for a latency-sensitive application would remain low and satisfy the application requirements. However, if the load on the link increase due to the bandwidth-intensitive applica-

tion, the latency of the latency-sensitive application could raise and not meet the application requirement. It also pointed out that, to solved the problem, the potential solution could be to reallocate the destination of the latency-insensitive application to save bandwidth for the latency-sensitive application. In the real-life scenario with multiple applications and different user patterns for each application and different applications' latency requirements, a resource optimization algorithm should be developed to automate the application deployment in the optical network.

The traditional simple traffic classification using TCP or UDP headers proved to be not capable of when dealing with the high complexity problem [6]. In recent years, Machine learning (ML) has been explored as a bright candidate to deal with the problem in the optical network including resource allocation to optimize the network by using Reinforcement Learning (RL) [7].

Based on it, this project aims to develop a reinforcement learning based algorithm to optimize the deployment of 5G and beyond applications in the optical metro network to satisfy applications' requirements. To achieve the thesis objective, the project has the following main tasks:

- Deploy real-life applications in the optical network.
- Develop user traffic generation model for each application.
- Set up an experiment to collect traffic data in the metro ring network.
- Develop network simulation model based on data collected in the experiment.
- Apply and train reinforcement learning model with simulation.
- Train the reinforcement learning model in the optical metro network.

In the first half part of the thesis, the real applications' deployment, user traffic pattern generation, set up an experiment to capture traffic from each application in the network and simulation software design are already achieved.

The rest of this report is structured as follows. The related work is presented in section II while the real application deployment, user traffic generation and traffic capture are presented in section III and section IV, respectively. The simulation design is represented in section V meanwhile, in the last section VIII, the plan for the second part of the thesis is discussed.

II. Literature review

According to Danish Rafique *et. al.* in [8], the main motivations to apply ML method in the optical network are heterogeneity, reliability, capacity, and complexity, which are not suitable with the static traditional design principle anymore. The author also argued that network self-configuration, including resource allocation and service (re)configurations—both for physical and virtual infrastructure are among the core application of RL, which is a branch of ML, in optical network. Applying Machine Learning, especially Reinforcement Learning in optical network is not new. Researchers have been using

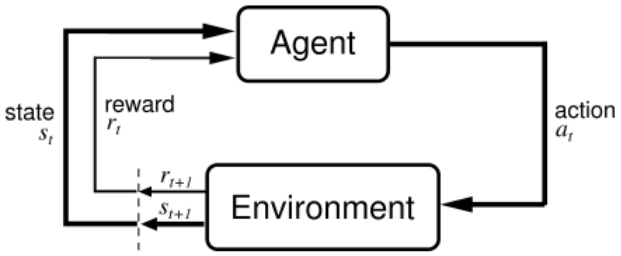

Fig. 2. Agent-environment interaction

RL for path selection and wavelength selection in optical burst switched networks since 2007 [9]. They proposed algorithms based on Q-learning to solve these problems near-optimally. In [10], Pham Tran Anh Quang *et. al.,* also using reinforcement learning to deal with the allocation of virtual network functions (VNFs) for realizing network services. The author proposed reinforcement learning with a neural network, which is called deep reinforcement learning (DRL), and "memory play" which allows having the advantages of supervised learning to allocate VNFs in the network.

Briefly speaking, reinforcement learning deals with learning via interaction and feedback, or in other words learning to solve a task by trial and error, or acting in an environment and receiving rewards for it. Basically, as we can see from Figure 2, the agent is the learner and the decision-maker from the environment, where the agent learns and decides what actions to perform. An action belongs to a set of actions that the agent can perform and the state is the state of the agent in the environment. For each action selected by the agent, the environment provides a reward which usually is a scalar value. The Policy is the decision-making function (control strategy) of the agent, which represents a mapping from situations to actions.

Reinforcement Learning has several more techniques to offer. To name a few, there are Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), Actor-Critic, and the most well-known reinforcement learning technique Double Deep Q Network (DDQN) [11]. In this work, DDQN is a bright candidate to use in this thesis not only because it has high efficiency but also based on the fact that the environment action space in this work is discrete. Besides, it is a well-built algorithm with high support from various tools and libraries.

III. Application Deployment

There are many usage cases of edge computing in real-life represented in various papers [1], [2], [12]. Among the applications that could be deployed in the optical metro network, web searching, web serving, and online shopping are chosen. They used the most popular platform on the Internet today: Nginx server, which has taken from Apache since April 2019 [13]. To deploy all these real application servers to the optical metro network, we have 2 options: container and virtual

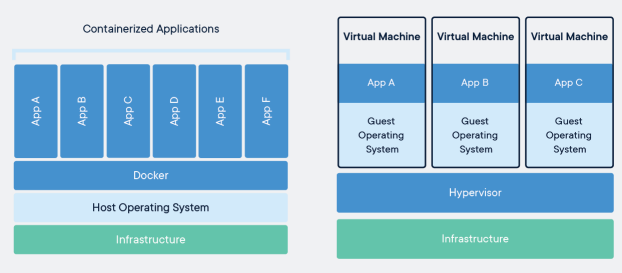

Fig. 3. Container and Virtual Machine

machine (VM). In this work, all the applications are deployed using docker container technology.

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings. Containers and virtual machines have similar resource isolation and allocation benefits but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient [14].

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in userspace. As in Figure 3, containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications, and require fewer VMs and Operating systems [15].

The application servers can be deployed in any node in the network. Because the access network is not available in the lab, simulated traffic is generated from the server in each node to the metro network as user simulator containers. Both application servers and users for each application server are containerized in docker containers as in Figure 4.

To build an effective algorithm that determines which node each application server should be, the bandwidth that each application server occupies based on real user requests is captured. To capture the traffic between each application and its user, node 3 and node 4 have been used to experiment as we can see in Figure 5. In the experiment, we captured each application at a time.

In the media streaming server, an HD video 1080p resolution length of 5.15 minutes is stored in the Nginx server and ready to be requested from the user simulator. Similarly, for web serving, web searching, and online shopping, there is a website page of Wikipedia, google result page and amazon frontpage HTML file is stored inside the Nginx server, respectively.

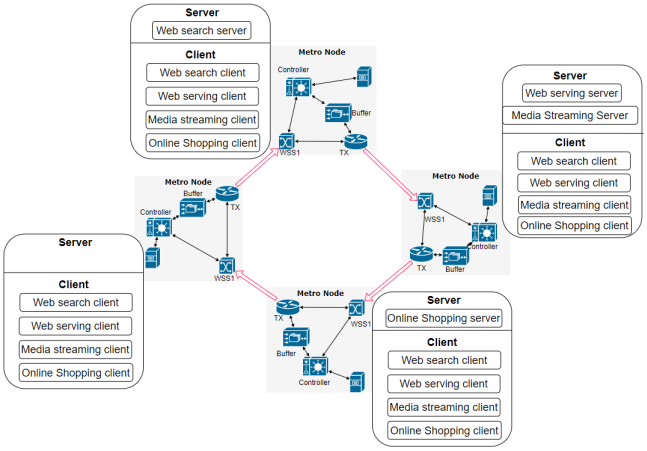

Fig. 4. Metro Ring Network

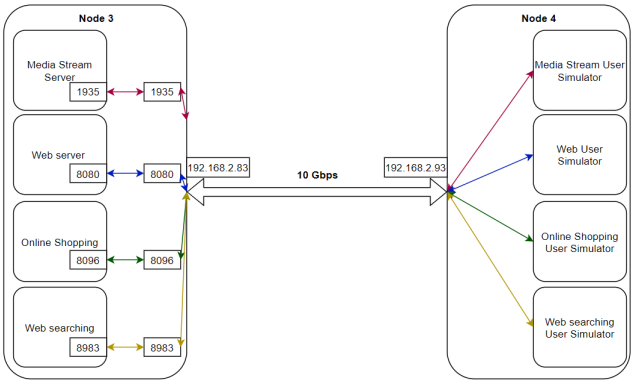

Fig. 5. Application and client Containers in node 3 and 4

IV. User Traffic Generation and Traffic Capture

To the most knowledge of the author, most of the user behavior can be assumed daily periodic based on research papers [16]–[19]. Based on the patterns in these researches, a software program python-based has been constructed to simulate the user. To simulate multiple concurrent users at the same time we used multi-threading in python. Each thread represents a user. The algorithm to generate users for each application is represented in Algorithm 1.

For each different server application, we used different tools to make user requests. For media streaming (Nginx server), RTMP protocol and FFmpeg has been used to send request from each user thread. For web searching, web serving, and online shopping, the user requests are HTTP protocol and python packages for HTTP requests have been used.

According to [20], around 70% traffic should be from video streaming applications, 30% for other applications. Here, in this work, 30% is divided equally among 3 other applications: web serving, web searching, and online shopping. The number of users should be used to stress the server for each application is calculated based on the traffic on the link that each application generated and based on the ratio above.

Algorithm 1: User request generator

Input:

- Number of concurrent active users hourly in 1 day.
- Scale ratio for each day.

Output: User threads hourly

User thread: **while** *True* **do**
    Send a request to the server.
    Sleep(Read/watch duration)
Initial: Create an initial number of user threads based on 1st day.
for *each day in our simulation time* **do**
    for *each hour in a day* **do**
        while *runtime less than an hour* **do**
            $UserNumber = UserNumberThisHour + (UserNumberNextHour - UserNumberThisHour) * runtime/hour$
            $UserNumber = UserNumber * scale$
            if $UserNumber > CurrentNumber$ **then**
                create $UserNumber - CurrentNumber$ threads.
            else
                stop and remove $CurrentNumber - UserNumber$ threads.
            Sleep(1 minute)

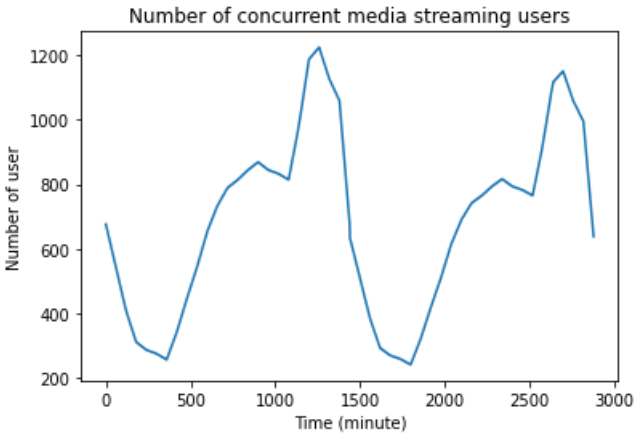

Fig. 6. Media user generated from user simulation

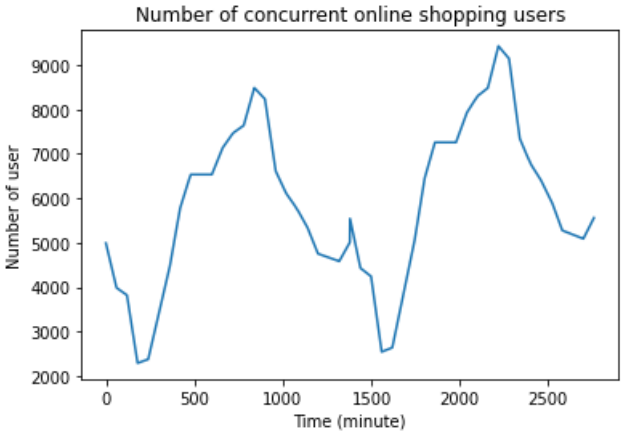

Fig. 7. Online Shopping user generated from user simulation

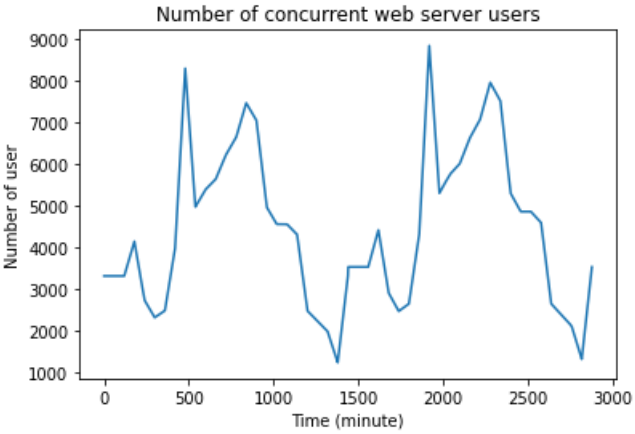

Fig. 8. Web server user generated from user simulation

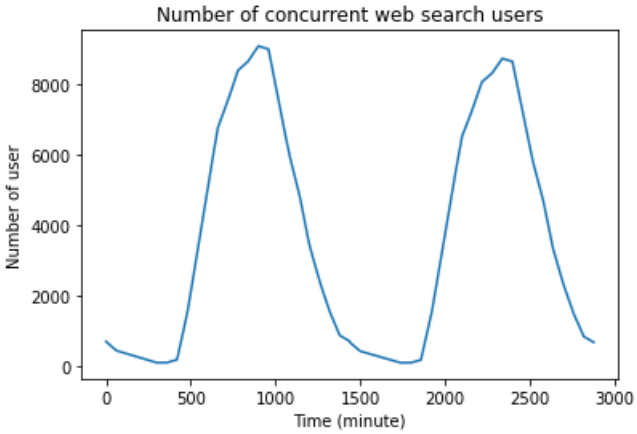

Fig. 9. Web searching user generated from user simulation

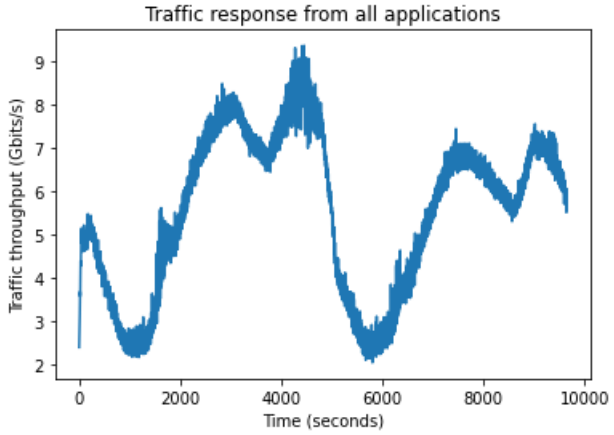

Fig. 10. Traffic response from all applications

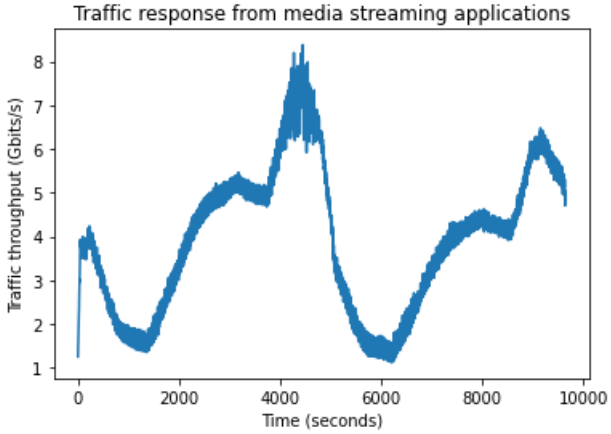

Fig. 11. Traffic response from media streaming application

NetData [21] has been used to capture the traffic on the link with a sampling rate of 1 second. All the application traffic traces has been captured as in Figure 10,11,12,13,14.

V. Simulation

To simulate the optical metro network with edge computing attached to each node and applications as well as client containers in each edge computing, the author proposes a simulation model which combines a network simulator based on OMNET++ and python programming. The optical communication links, buffers, TXs, and WSSs are modeled using the OMNET++ library. Meanwhile, the edge computing server which contains application servers and client containers is modeled using python.

A. *OMNET++ software simulator*

OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators [22].

Briefly speaking, in this OMNET++ software, during the initial stage, the control packets are sent to establish the time

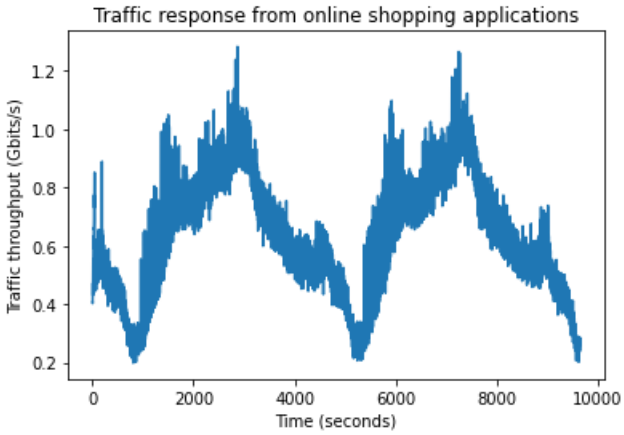

Fig. 12. Traffic response from online shopping application

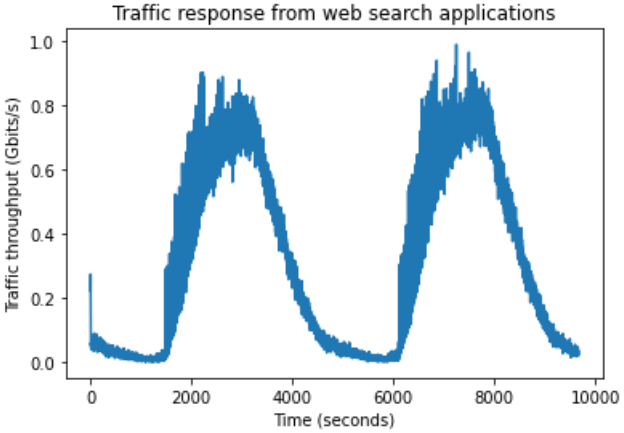

Fig. 13. Traffic response from web search application

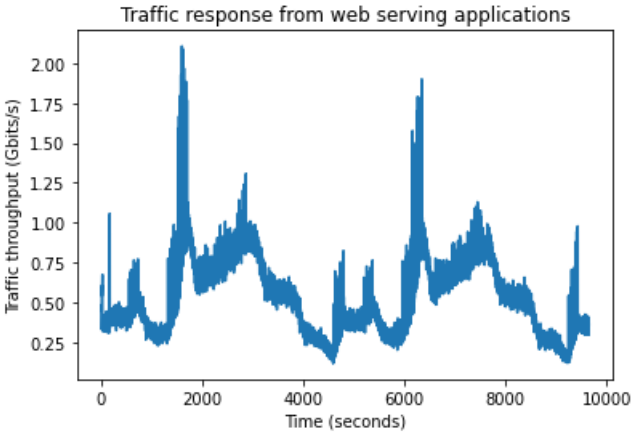

Fig. 14. Traffic response from web server application

slot in the whole network. The control packets contain a table of the destinations of packets in their time slot. In the running stage, the sources generate the data packets based on pre-defined load parameters to buffer with specify the destination of each packet. The buffers store the data packets from sources in their buffers and then check which wavelength is available and assign the data packet from the most loaded buffer to the available wavelength. In the meantime, the buffer also updates the control packet based on the destination of the data packet. If the buffers are full, the packet from the source would be discarded. The combiner is used to combine the packets from WSS1 and buffer together and put them to the link, which would introduce 100us latency since the optical link is 20km long. The WSS1 in each node checks the control packet if the data packet coming in that time slot is belongs to the node or not. If the packet does not belong to the node, then the WSS1 just forwards it to the combiner to send it to another node. Otherwise, WSS1 drops the packet to the node and calculates the latency of the packet.

B. Python

As we can see in Figure 15, the OMNET++ is running inside the python program. For each specific time, the simulation program will take an action set from the control algorithm. The action can be to deploy a new application to the network or reallocate an application in the network. Based on the action, the applications are deployed accordingly as traffic sources in each node in the OMNET++ simulation program. The load parameter for each traffic source is calculated based on the experiment result in section IV for each specific time. The users' request traffics are deployed as user traffic sources in every node for each application. The load traffic from each application to its client is split based on the load requests from each client to the server. After preparing all the traffic source loads for server and client, the simulation program will use the bash command to run OMNET++ software to do the simulation with specific source loads parameters. After each iteration, the latency, total packet arrival, and packet lost parameter are captured and sent to the control algorithm.

VI. Network Modeling

We have $G =< V, L >$ with $v \in V$ and $l \in L$

VII. Conclusion

In conclusion, for the first part of the thesis. The following main tasks have been achieved:

- Deploy real-life applications in the optical network.
- Develop user traffic generation model for each application.
- Set up an experiment to collect traffic data in the metro ring network.
- Develop network simulation model based on data collected in the experiment.

The real-life applications that have been deployed are media streaming, web serving, web searching, and online shopping. The user pattern generator for each application also has been

Algorithm 2: Simulation Software

Input: Deployment Action
Output: Network Performance metrics
Data: Traffic data

```
for each data in traffic data do
    for each application in deployment action do
        if Application is not created yet then
            Create an application with a particular index.
        if Application already belongs to the assigned node or Application does not belong to any node in the network or Assigned node resources is full then
            continue - skip the rest of this loop.
        if Application currently belongs to another node then
            Delete the application in that node.
        Deploy the application to the assigned node.
        Deploy users to other nodes.
    Update the source list in each node.
    Create a .ini file from the source list in each node.
    Run OMNET++ software program.
    if simulation runtime greater than 10 min then
        Return runtime error.
        Stop OMNET++ simulation
    else
        Return latency, packet arrival number, packet lost.
```

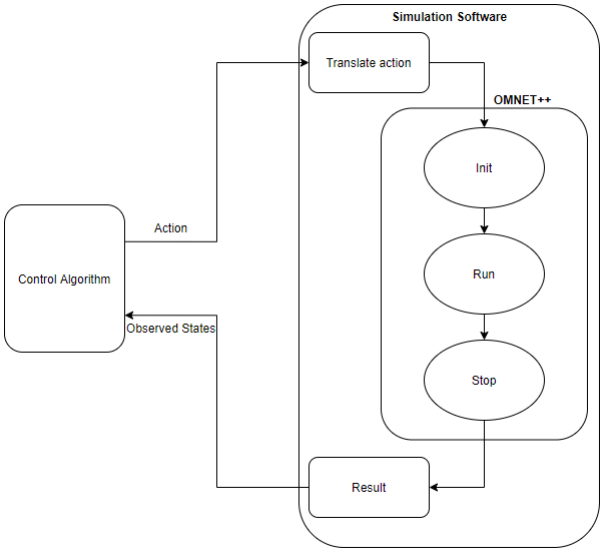

Fig. 15. Simulation model for each iteration

developed and tested with real-life applications. The result is shown in section IV. The traffic responses from server applications show that there is a similarity with the user request patterns. The simulation model is already developed based on the real experiment set up with 4 nodes and the source load parameters in each node are depend on the traffic capture in the experiment.

VIII. Plan

For the upcoming months, there are several tasks needed to be done.

- Apply reinforcement learning algorithm and train with the simulation model.
- Automate the network with docker swarm or K8s and monitoring latency.
- Train the reinforcement learning model in the optical metro network.

a) Algorithm: As we discussed in section II, the bright algorithm to try should be DDQN but the other algorithm such as Actor-Critic also should be considered because of the fast training speed and the main limitation of DDQN is long convergence time [23].

b) Application deploy automation and monitoring latency: To automate the deployment and manage all applications, an orchestrator tool for the container should be used. Kubernetes (K8s) and Docker swarm are the two biggest players in this field. In the next part of the work, one of them should be used. Since NetData can not give the metrics about the latency of each container, we need another tool to monitor the latency. Prometheus is a good choice to do this.

c) Online trainning: After finalizing the algorithm, test with simulation, and automate the application deployment, the algorithm should be trained with the real-life scenario in the optical network. Ok TEst!

References

[1] C. International, "Cisco Visual Networking Index: Forecast and Methodology Cisco Visual Networking Index: Cisco Visual Networking Index: Forecast and Methodology," *Forecast and Methodology*, pp. 6–17, 2017.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, 2015, pp. 73–78.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[5] B. Pan, E. Magalhaes, F. Wang, X. Xue, and N. Calabretta, "Experimental assessment of sdn controlled metro access network with network slicing and edge computing under 5g applications," in *2019 24th OptoElectronics and Communications Conference (OECC) and 2019 International Conference on Photonics in Switching and Computing (PSC)*, 2019, pp. 1–3.

[6] T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[7] R. Gu, Z. Yang, and Y. Ji, "Machine learning for intelligent optical networks: A comprehensive survey," *Journal of Network and Computer Applications*, vol. 157, no. December 2019, p. 102576, 2020. [Online]. Available: https://doi.org/10.1016/j.jnca.2020.102576

[8] D. Rafique and L. Velasco, "Machine learning for network automation: Overview, architecture, and applications," *J. Opt. Commun. Netw.*, vol. 10, no. 10, pp. D126–D143, Oct 2018. [Online]. Available: http://jocn.osa.org/abstract.cfm?URI=jocn-10-10-D126

[9] Y. V. Kiran, T. Venkatesh, and C. S. Ram Murthy, "A reinforcement learning framework for path selection and wavelength selection in optical burst switched networks," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, pp. 18–26, 2007.

[10] 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*, vol. 16, no. 4, pp. 1318–1331, 2019.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[12] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[13] "January 2021 web server survey," Jan 2021. [Online]. Available: https://news.netcraft.com/archives/2021/01/28/january-2021-web-server-survey.html

[14] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[15] "What is a container?" [Online]. Available: https://www.docker.com/resources/what-container

[16] F. Kooti, K. Lerman, L. M. Aiello, M. Grbovic, N. Djuric, and V. Radosavljevic, "Portrait of an online shopper: Understanding and predicting consumer behavior," *WSDM 2016 - Proceedings of the 9th ACM International Conference on Web Search and Data Mining*, pp. 205–214, 2016.

[17] I. Ullah, G. Doyen, G. Bonnet, and D. Gaïti, "A survey and synthesis of user behavior measurements in P2P streaming systems," *IEEE Communications Surveys and Tutorials*, vol. 14, no. 3, pp. 734–749, 2012.

[18] M. Taghavi, A. Patel, N. Schmidt, C. Wills, and Y. Tew, "An analysis of web proxy logs with query distribution pattern approach for search engines," *Computer Standards and Interfaces*, vol. 34, no. 1, pp. 162–170, 2012.

[19] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng, "Understanding user behavior in large-scale video-on-demand systems," *Proceedings of the 2006 EuroSys Conference*, pp. 333–344, 2006.

[20] 5GPPP, "View on 5G Architecture, Version 3.0, February 2020," *5G Architecture White Paper*, no. February, 2020. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2019/07/5G-PPP-5G-Architecture-White-Paper_v3.0_PublicConsultation.pdf

[21] Netdata, "netdata/netdata." [Online]. Available: https://github.com/netdata/netdata

[22] Andras and Rudolf, "Andras." [Online]. Available: https://omnetpp.org/

[23] D. Mehta, "State-of-the-art reinforcement learning algorithms," *International Journal of Engineering and Technical Research*, vol. V8, 01 2020.