Halfway Evaluation Master thesis: Design an algorithm to optimize the metro ring network for real-life applications based on user demands

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

***Abstract*—This document is a halfway evaluation of the master thesis: Design an algorithm to optimize the metro ring network for real-life applications based on user demands. In this work, the real applications deployment, user traffic pattern generation software, traffic captured from each application as well as aggregated traffic and simulation software design are presented.**

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 they require low latency response. To address this issue, one solution that could think of is edge computing.

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 a powerful server as an 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 [2], and the response time is reduced from 900 to 169 ms by moving computation from cloud to the edge. Researchers in [3] 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.

In the TU/e lab, there is an experiment has been carried based on SDN reconfigurable and network slicing to enable optical metro access ring network with edge computing resources [4]. In this experimental set up, in Figure 1, B. Pan *et. al.* has been demonstrated that using SDN, the MCU-based OpenROADM agents can be controlled via NETCONF protocol, which in turn, gives the command to FPGA via PCIE data flow. The FPGA will drive SOAs to control the traffic inside the optical switch. The classification algorithm based on the utilization of network resources, if it is a slightly loaded application can be hosted in an edge computing node, if it is fully loaded the insensitive application is switched to centralized DC to save resources for the latency-sensitive application.

However, in the demonstration, there were only 2 applications and there was no real-life user pattern considered. Meanwhile in the future, with real traffic from various applications and rather complicated network status, such as link utilization, latency, and packet loss, we need an algorithm that can allocate the applications in different nodes in a more efficient way to serve users' requests. To solve this problem, this project aims to develop an algorithm, which is based on different application requirements and user demands, to optimize the deployment of applications in the optical metro network to satisfy service expectations.

II. Thesis Objective

The project's main objective is to develop a software algorithm to control where should the applications deployed in the network to satisfy the application requirements. The software will be developed, simulated, and tested. To achieve the thesis objective, the project has the following main tasks:

- Deploy real-life applications in the optical network.
- Develop user traffic generation software for each application.
- Set up an experiment to collect traffic data in the metro ring network.
- Develop network simulation software based on data collected in the experiment.
- Develop algorithm and test with simulation.
- Train the algorithm online with the real data from the experiment.

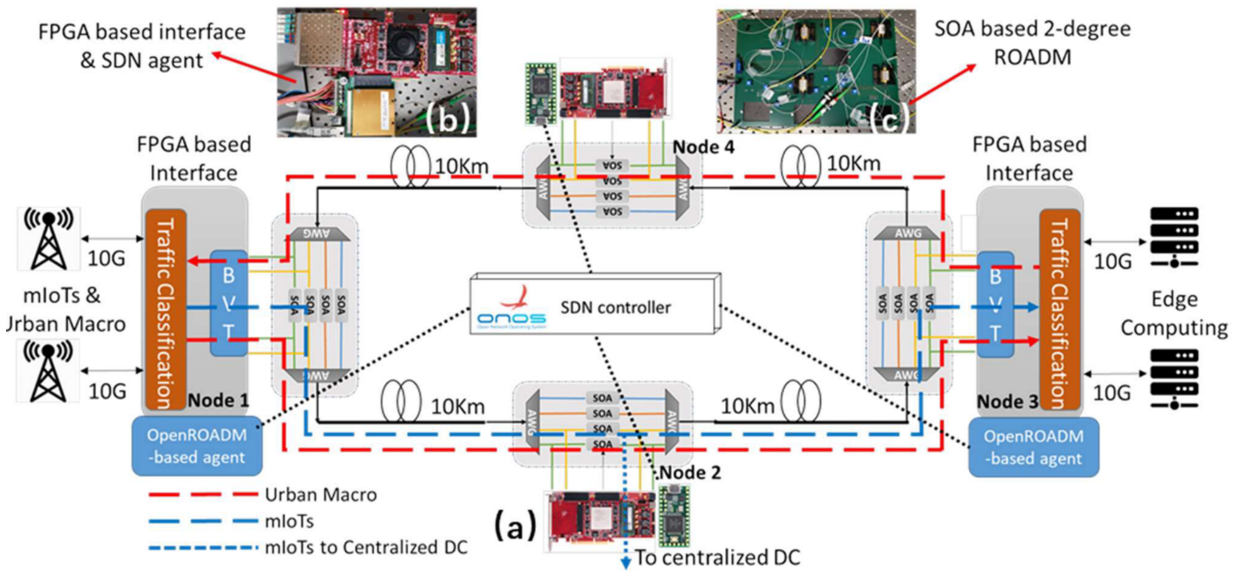

Fig. 1. Metro Ring Network

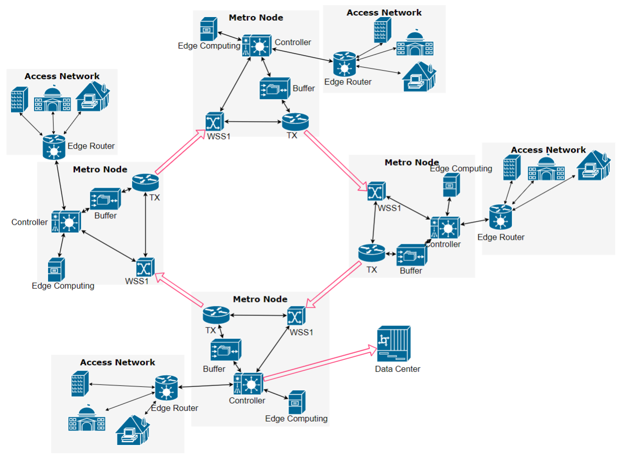

Fig. 2. Metro Ring Network

III. Experiment set up

A. Ring Metro Network set up

As we can see from the Figure 1 and Figure 2, the ring optical metro network consists of 4 nodes connect to each other by a 10Gbps unidirectional optical link (red lines). Each node consists of the optical part which is SOA-based 2-degree ROADM, FPGA controller with O/E/O interfaces, and a powerful server for computing. The traffic that comes from the access network will be aggregated to the metro network. There is also an optical link to the data center that has been set up in the lab.

B. Applications

There are many usage cases of edge computing in real-life represented in various papers [1], [5], [6]. 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 [7]. To deploy all these real application servers to the optical metro network, we have 2 options: container and virtual machine. 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 [8].

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 [9].

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.

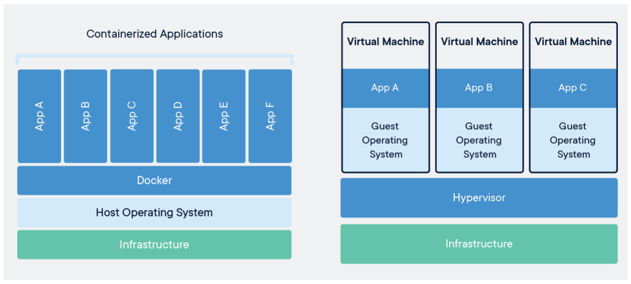

Fig. 3. Container and Virtual Machine

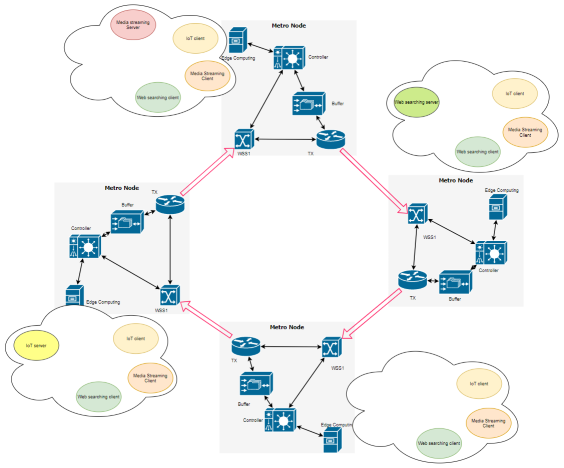

Fig. 4. Metro Ring Network

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.

C. User traffic generator

To the most knowledge of the author, most of the user behavior can be assumed daily periodic based on research papers [10]–[13]. 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),

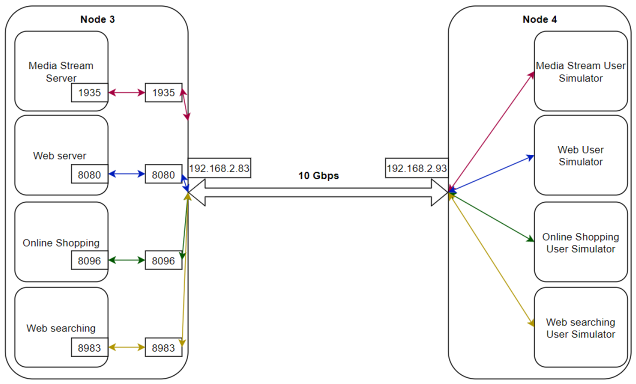

Fig. 5. Application and client Containers in node 3 and 4

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. Noted that FFmpeg needed to be configured to remove codec processing from the default setting to reduce the CPU consumption on the host server.

Algorithm 1: User request generator

```
Input: Number of concurrent active users hourly in 1 day
Output: User threads hourly
1  User thread: while True do
2      Send a request to the server.
3      Sleep(Read/watch duration)
4  Initial: Create an initial number of user threads based on 1st day.
5  for each day in our simulation time do
6      scale = uniform(0.9, 1.1) for each hour in a day do
7          while runtime less than an hour do
8              $UserNumber = UserNumberThisHour + (UserNumberNextHour - UserNumberThisHour) * runtime/hour$
               $UserNumber = UserNumber * scale$
9              if $UserNumber > CurrentNumber$ then
10                 create $UserNumber - CurrentNumber$ threads.
11             else
12                 stop and remove $CurrentNumber - UserNumber$ threads.
13             Sleep(1 minute)
```

According to [14], around 70% traffic should be from video streaming applications, 30% for other applications. Here, in

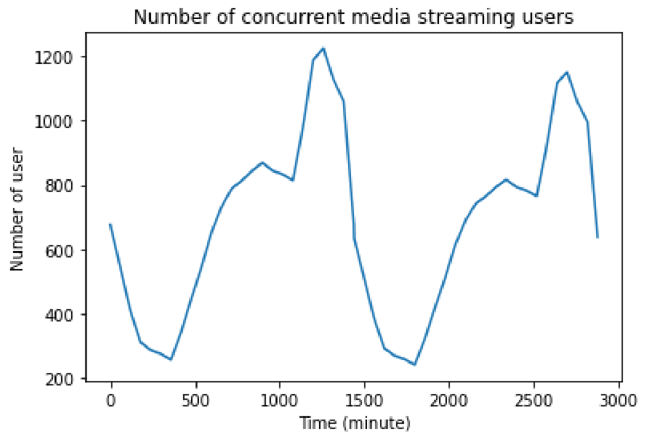

Fig. 6. Media user generated from user simulation

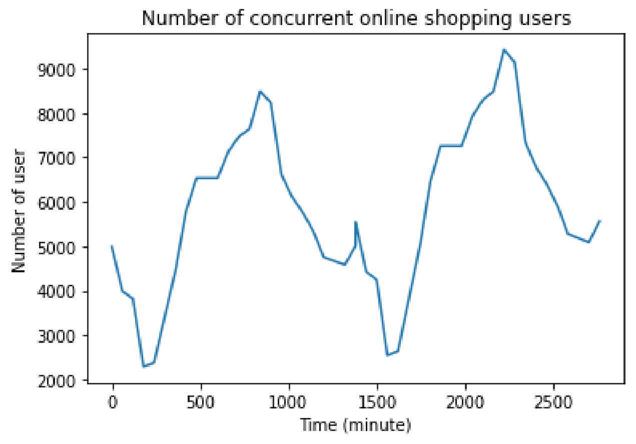

Fig. 7. Online Shopping user generated from user simulation

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.

D. Capturing data

NetData 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. Noted that to speed up, the user generator software scale down the real time to 18 times.

IV. 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

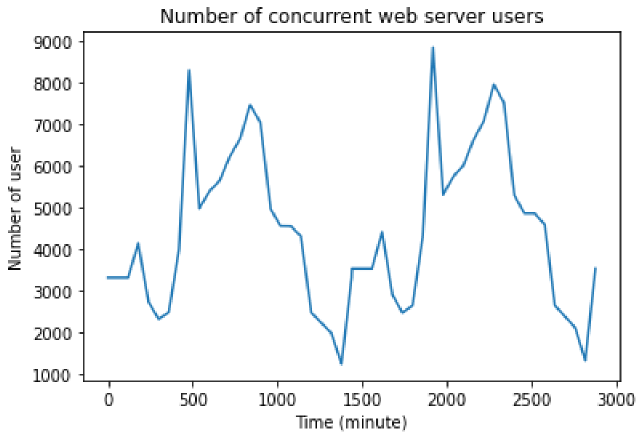

Fig. 8. Web server user generated from user simulation

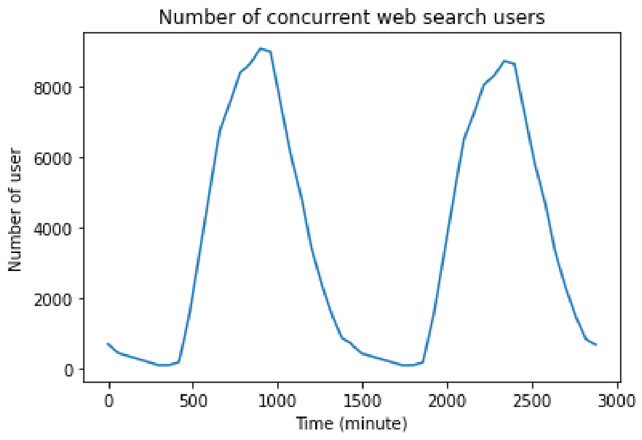

Fig. 9. Web searching user generated from user simulation

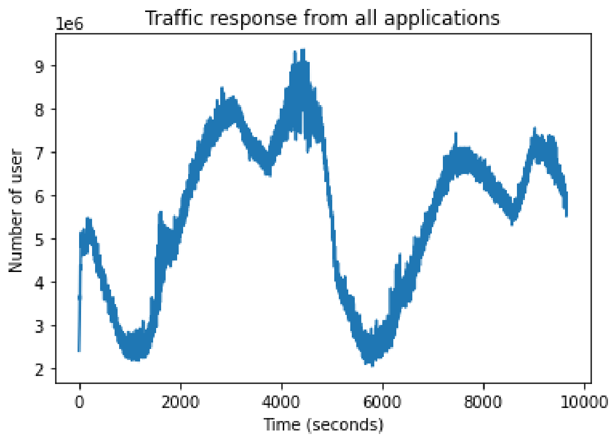

Fig. 10. Traffic response from all applications

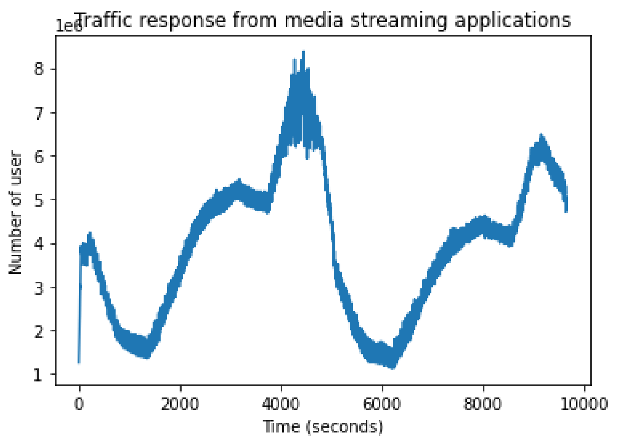

Fig. 11. Traffic response from media streaming application

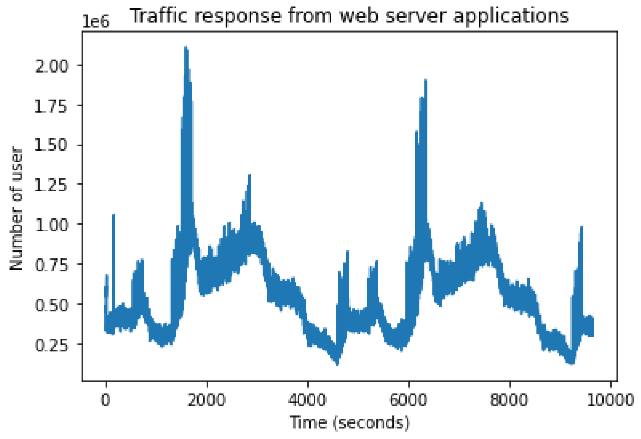

Fig. 12. Traffic response from online shopping application

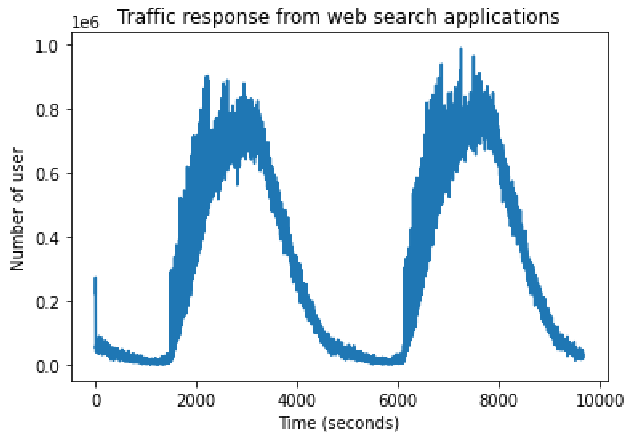

Fig. 13. Traffic response from web search application

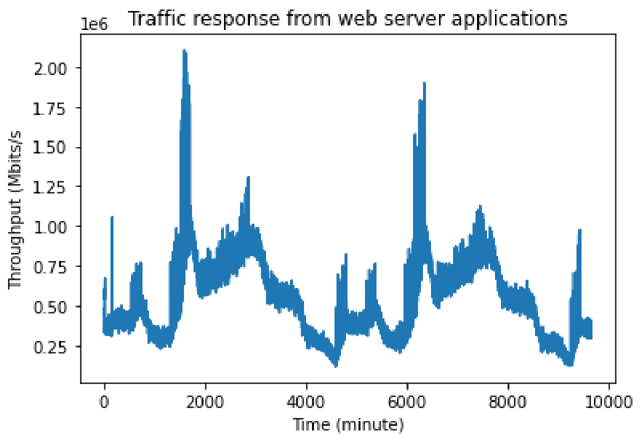

Fig. 14. Traffic response from web server application

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. In this work, the OMNET++ software simulation is already built by Bitao.

Briefly speaking, in this OMNET++ software, during the initial stage, the control packets are sent to establish the time 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 buffer store the data packets from sources in its 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 100ms 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 calculate the latency of the packet.

B. Python

For each specific time, the simulation software calculates the load based on the traffic captured for each application and prepares the initial file for OMNET++ to run. Therefore, each source in a node will represent an application server or a client depends on the deployment action. The action can be to deploy a new application to the network or reallocate an application in the network. In the software, the clients are deployed automatically in every node for each application. For example, with a media application, a media server can be deployed in node 1 and we have clients in all nodes. Therefore there should be 2 sources in node 1, 1 for the media server and the other for the media client and there should be 1 source in every other node for the media client. The load from the server to each client will be split depends on the load request from each client to the server. After the finish preparation stage, the software 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.

Algorithm 2: Simulation Software

```
Input: Deployment Action
Output: Network Performance metrics
Data: Traffic data
1  for each data in traffic data do
2      for each application in deployment action do
3          if Application is not created yet then
4              Create an application with a particular index.
5          if Application already belongs to the assigned node or Application application does not belong to any node in the network or Assigned node resources is full then
6              continue - skip the rest of this loop.
7          if Application currently belongs to another node then
8              Delete the application in that node.
9          Deploy the application to the assigned node.
10         Deploy users to other nodes.
11     Update the source list in each node.
12     Create a .ini file from the source list in each node.
13     Run OMNET++ software program.
14     if simulation runtime greater than 10 min then
15         Return runtime error.
16         Stop OMNET++ simulation
17     else
18         Return latency, packet arrival number, packet lost.
```

V. Plan

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

- Investigate the algorithm
- Train algorithm with the simulation software
- Automate the network with docker swarm or K8s
- Train algorithm with real network experiment

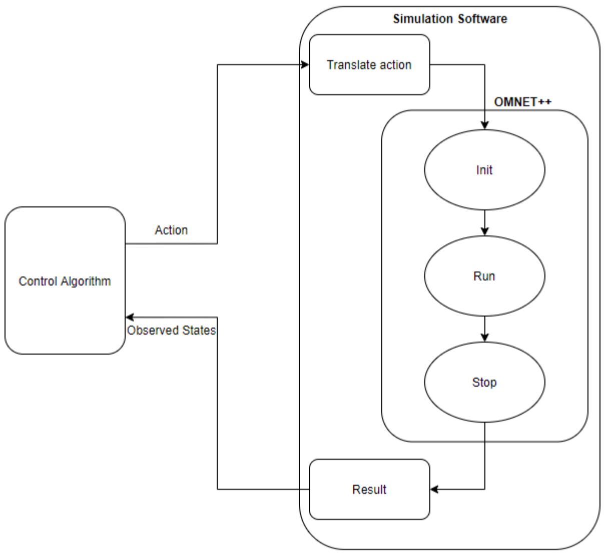

Fig. 15. Simulation Software

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] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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