위치 인식 다중 경로 네트워크 프로비저닝

홍남곽, 염성웅, 김경백*

전남대학교 인공지능융합학과

quachhongnam1995@gmail.com, yeomsw0421@gmail.com, kyungbaekkim@jnu.ac.kr

Location-aware Multi-Path Network Provisioning

Hong-Nam Quach, Sungwoong Yeom, Kyungbaek Kim*

Dept of Artificial Intelligence Convergence, Chonnam National University

Abstract

As the number of IoT and mobile devices grows and the concept of 5G technology becomes more mainstream, various network infrastructures are expanding, and demand for customized network services is becoming a hot trend. Also, the demands of request user-specific network services are increasing along with the still restricted network infrastructure. Providing a guaranteed QoS level to users' requirements necessitates taking into account the variety of factors that influence network service efficiency, as well as complex network provisioning. In this paper, we suggest a novel dynamic network provisioning scheme with multi-path planning. In this system, we design a user request processor system that understands user-specific QoS specifications in order to optimize network resource utilization.

Ⅰ. INTRODUCTION

Nowadays, along with network infrastructure and technology development, the demand for personalized networks from users is increasing [1]. Besides, the Covid-19 epidemic broke out, making it impossible for people not to go out, leading to an ever-increasing demand for internet use. The dynamic network provisioning concept is a way to provide flexibility and service customization in existing networks by sharing network resources, which include both physical and logical resources (e.g., computing, storage, etc.) among different service providers. Moreover, location-aware network provisioning allows dynamically generating a subnetwork based on users' requested locations to provide services that match particular needs [2]; users can also input some service and network parameters into their request [3]. However, when many users want to use the network services with the exact location and resources. The ISP needs to calculate for leveraging their resource to provide the several different paths. This helps them ensure available QoS and save the operating expenses.

Recently, pathfinding in a location-aware network has been the subject of many studies in recent years. For example, a method of best pathfinding using Location-aware AODV (Ad-hoc On-demand Distance Vector) for MANET (Mobile Ad-hoc NETwork) is suggested in [4]. This paper used multiple parameters such as node-ID, timestamp, GPS, bandwidth, RTT, packet loss ratio, and others to modify an existing protocol called AODV based on location to find the best path among multi-path routing protocols for MANET. On the other hand, the authors in [5], [6] proposed online algorithms with an auxiliary graph for unicast and multicast requests, including a bandwidth constraint and maximized network throughput. However, these studies do not consider the location-specific information of user requests. For example, when the students send requests to access the online classes in university in their house, the ISPs should determine the requested locations that students would use to make the network their university location, instead of his house's location. This paper suggested a dynamically estimating network switch multi-path for location-aware network provisioning to overcome this problem. This supports the network provider could leverage their resources and save cost operation.

Ⅱ. METHODOLOGY

This section suggested a method for estimating network switch multiple-path based on the user has requested locations. We use our proposed device architecture based on [2] in this paper. After that, we suggest two extensive new features. The web user interface, which is designed to allow several users to

submit their requests simultaneously. Second, we add the Network Path Calculator module, which calculates suitable switch multi-paths to accept as many user requests as possible.

A. Web application interface for obtaining the location requests

This section explains the web application, enabling users to send location requests and sending them to a central server for processing, as shown in Fig. 1. A map-based selector-region Web UI is shown in the diagram below. A user may select locations and determine the appropriate QoS level using network parameters such as bandwidth and delay. Then, the location of selected regions, which are expressed with the ID and the parameter of coordinates. These pieces of information are stored in the database and import into the CSV file.

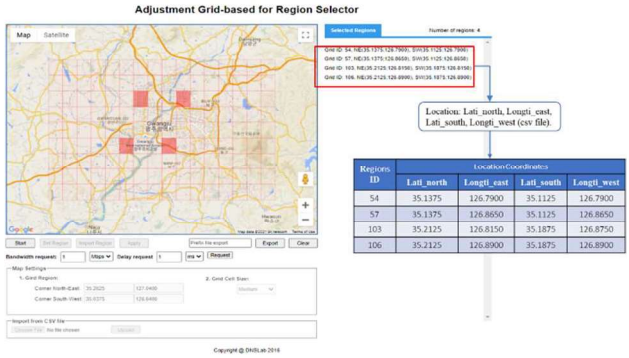

Figure 1: Web-UI with several selected location by a user

B. Multi-path calculation

This paper uses an algorithm to identify all network devices that cover the requested locations to create a subnetwork based on the requested locations. The algorithm analyses each area to see if it belongs to a switch's cover field. Finally, the algorithm returns a list of devices to build the subnetwork.

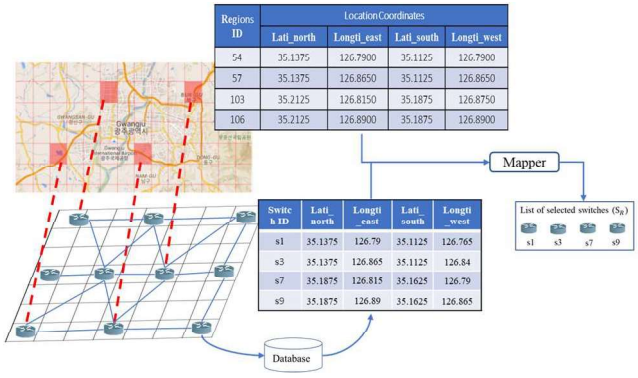

Figure 2: Model of Location-awareness

Users can select locations through a map interface that corresponding with switches connections in the below layer. Two tables contain the ID and necessary coordinates from the user-requested location and covered regions of selected switches.

Algorithm1: Mapping and find the corresponding switches cover selected regions

INPUT: The list of coordinates of selected regions

OUTPUT: List of switches cover selected regions S_{L_R}

Step 1: Get the and extract out the location coordinates

The List location of request: $L_R = \{l_1, l_2, \ldots l_n\}$

l_1 = (lat.ne, lng.ne, lat.sw, lng.sw)

Step 2: Get the information of resource switches in the DB

The resource Switches: $S = \{s_1, s_2, \ldots s_n\}$

s_1 = (reg.lat.ne, reg.lng.ne, reg.lat.sw, reg.lng.sw)

Step 3: Compare:

For each $s \in S$ do

For each $l \in L_R$ do

If:

l.lat.ne $\leq$ s.reg.lat.ne &&

l.lng.ne $\leq$ s.reg.lng.ne &&

l.lat.sw $\leq$ s.reg.lat.sw &&

l.lng.sw $\leq$ s.reg.lng.sw then

S_{L_R}.add(s)

End if

End for

End for

→List of cover switches: S_{L_R}

We created a location-awareness model in which a module uses an algorithm to identify switches with a cover region that contains at least one requested usage location. After that, the Network Path Calculator creates the sub-network topology using the list of switches identified by the location-awareness model and algorithm 2. The procedure "Mapping and finding the corresponding switches cover selected regions" in Algorithm 1 illustrates our deployment for the model of location-awareness. In algorithm 1, two processes: Get and extract locations which return a list of selected locations (Step 1), and get the list switches that return a list of resources switches (Step 2). Next, the algorithm checks and compares if at least one requested position (Area) into the cover region of that switch (Step3) exists for each switch. Finally, the algorithm returns a list of the corresponding switches covering selected regions. This list of cover switches forwarded to the path calculator to find the routes to build a sub-network topology to provide the customers' network services.

We assume that all link connect switches are of the same quality. Many paths to connect all switches in the selected list and make a subnetwork that covers the user has requested locations. The Network Path Calculator deployed the Dijkstra algorithm to find effective routes to create a subnetwork that uses the shortest paths between each intermediate device pair. In other words, it is the active path in the subnetwork. Then, we bring all of the shortest paths together to create a dynamic route used by default.

In the above part, we have designed a system for connecting all of the devices selected. However,

several scenarios in reality that cause lead to the path may not operate effectively, such as one or more connections in this subnetwork are overloaded or malfunctioning connections occur. For example, when so many users request a set of the same switches with different bandwidth, users simultaneously transfer their data in the same subnetwork. This leads links in the subnetwork to become overloading.

This paper deploys the algorithm to find other paths to overcome this issue, which are the alternative routes for the shortest path with similar constraints. We assume the traffic load of the network is set up and define the following:

The traffic load information of each link:

$$T_{(used)} = \sum_{i}^{k} BW_{i_{(required)}}$$

The remaining traffic load information of a link:

$$T_{(remain)} = T_{(original)} - T_{(used)}$$

Where:

$T_{(used)}$: the traffic load was used by users in a link

$BW_{i_{(required)}}$: the bandwidth requirement of the user i^{th}in the all k users send requests

$T_{(original)}$: the traffic load is set up initially on each link

Algorithm 2: Finding the alternative paths

INPUT: Set of the selected switches S_R

OUTPUT: Set of the paths to connect the switches in S_R

Procedure: S_R = {P_{S_R}} pair of switches in S_R

Step 1: Check the current traffic information in network

Step 2: Choose any P_{S_R} in S_R and find the route to connect

Step 3: In P_{S_R}, from the source switch, select the neighbor device.

Step 4: If the link between source and neighbor device have remained traffic.

Compare the remain traffic with bandwidth require.

If $BW_{(required)} < T_{(remain)}$

Add it to expected path

If the link between source and neighbor device have remained traffic.

Compare the remain traffic with bandwidth require.

If $BW_{(required)} < T_{(original)}$

Add it to expected path

Step 5: Repeat the step 4 until to reach the destination.

If not, return no path is found

Step 6: Perform similarly with next other pair in P_{S_R}

Finally, we calculate a set of alternative paths, and after combining these routes, we make a table, which contains the shortest paths for the user.

III. EVALUATED

A. Setting Environment

To conduct the test, we built a virtual machine on VMware with the Operating system Ubuntu 20.04.0.2 LTS. Also, we deploy the framework by using Python2.7, MySQL. To support the framework, we designed and implemented the Web-UI by using socket.io, node.js, and HMTL. We use the editor Visualcode to build the code of the framework and Mininet to build the environment.

Environment	Configuration
Operating system	Ubuntu-20.04.2.0 LTS
Memory	RAM 8GB
Programming language	Python 2.7, NodeJS, HTML
Libary	Python 2.7 Libary
Database	MySQL server
Simulator	Mininet 2.2.0
Protocol	OpenFlow1.3

Table 1: Experimental environment configuration.

In this paper, the experiment is conducted several experiments with various network settings using the mininet to evaluate the proposed method. Then, we measure the number of accepted requests and accumulate bandwidth. In these experiments, we set up switch network size in 13 nodes, and each network link has a bandwidth capacity in the range of 100 ~ 1000 Mbps and 2~5ms delays. Also, we randomly pick 20% of devices in a request and randomly choose the start time and usage duration. The QoS parameter of a request includes bandwidth constraint (1~10Mbps) and delay constraint in range (40ms ~ 200ms).

Performing experiments with the Web-UI helps the customers give the requirements about location, QoS constraints as bandwidth, and delay. The experiments assume the number of requests from users will become in our framework frequently and continuously. We generated requests with the experiments, such as 10, 20, 30, 50, and up to 500 requests to conduct this assume. Based on the location parameters of these will be handled to find the corresponding switches resources. We then used the QoS constraint of these requests to find the shortest path fitting each request. To evaluate our proposed method's performance, we offer the Dijkstra algorithm to compare.

B. Analysis, Evaluate Results

Our proposed method is evaluated with the network with topology size 13 switches, and the number of requests increases up to 100 while other parameters are fixed. Fig.3 plot the performance curves of two different methods from which it can be seen that the proposed method outperforms Dijkstra in all cases.

Specifically, at the very first simulation time, when the number of requests is relatively small (less than 50), the number of accepted requests seems to be the

same

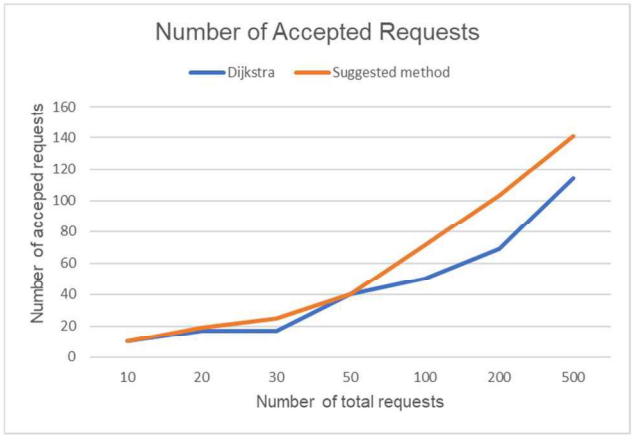

Figure 3: The number of accepted requests

between the two methods. However, as the number of requests grows, our proposed approach accepts a slightly higher number of requests than the other, up to 1.5 times at the end of the simulation. In other words, when more requests come into the system, more requests can be accepted more than the other, as Fig.3. With Dijkstra, when the network resource serves a request, Dijkstra cannot use it for other requests until its duration finishes. However, with the proposed method, the network resource can be shared by several requests within an expected certain amount of time them. Furthermore, the suggested method also finds alternative paths that can fit the user request's QoS constraints

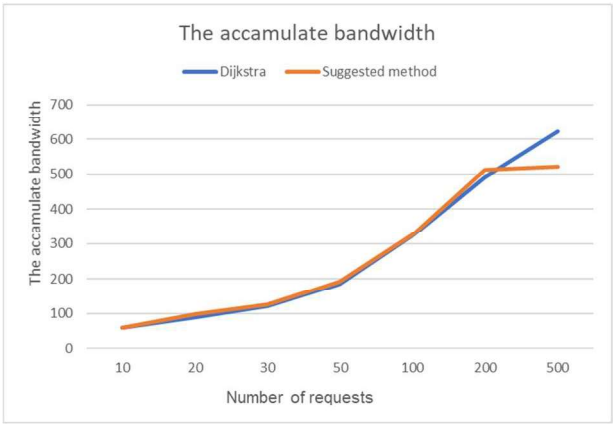

Figure 4: The accamulate bandwidth

Besides, Fig. 4 shows that our proposed approach provides more cumulative bandwidth than the Dijkstra method, despite the difference being very minor due to our simulation network resource constraints. However, we recognize that when the number of requests increases from 200 to 500 at the end of the simulation. The proposed method's cumulative bandwidth decreases more diminutive than the Dijkstra method. The explanation for this is that while the Dijkstra method resource is saturating, the proposed method can accept more requests whose QoS requirements are negligible.

IV. CONCLUSION

In this paper, an approach is proposed that is a framework that includes a web-UI to gather users' requests. The server responsible listens, obtains, and handles the request, which is continuously maintained. This server supports network resources' deployment, finds the routes that respond to user's requests, and creates subnetwork topologies that match the offers' requirements. The server also monitors the network operation to gather, analyze, anticipate, avoid, and respond to network incidents. We are trying to enhance network provisioning performance in the number of accepted requests and QoS constraints and extend the experiment to measure the average bandwidth and delay to evaluate the proposed method's performance.

ACKNOWLEDGMENT

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2021-2016-0-00314) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation). This results was supported by "Regional Innovation Strategy (RIS)" through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(MOE).

References

[1] Wellman, Barry. "Physical place and cyberplace: The rise of personalized networking." International journal of urban and regional research 25.2 (2001): 227-252.

[2] Van-Quyet Nguyen, Sinh-Ngoc Nguyen, Deokjai Choi, Kyungbaek Kim. "Location-aware Dynamic Network Provisioning," In Proceedings of the 19th APNOMS, 2017.

[3] Gde Dharma N., Van-Quyet Nguyen, Tiep Vu Duc, Ngoc NguyenSinh, Alvin Prayuda J.D., Kyungbaek Kim, Deokjai Choi "Design of Service Abstraction Model for Enhancing Network Provision in Future Network," In Proceedings of the 18th APNOMS, 2016.

[4] Anagha Raich, Amarsinh Vidhae. "Best Path Finding using Location-aware AODV for MANET," International Journal of Advanced Computer Research Volume 3, Num3, pp.336-340, Sep. 2013.

[5] M.Huang et al., "Dynamic routing for network throughput Maximization in Software-defined networks," in Proceeding of IEEE INFOCOM, USA, 2016, pp. 1-9

[6] Mike Jia, W.Liang, M.Huang, Z.Xu, and Yu Ma, "Routing Cost Minimization and Throughput Maximization of NFV-enabled Unicasting in Software-defined networks," IEEE Transaction and Network service Management, vol. 15, no. 2, pp. 732-745, June 2018.