Contents lists available at Science-Gate

International Journal of Advanced and Applied Sciences

Journal homepage: http://www.science-gate.com/IJAAS.html

The performance effect due to varying network topologies on a software-defined network employing the k-shortest path

R. Linsheng [1], M. N. Derahman [1], M. F. A. Kadir [2,*], M. A. Mohamed [2], S. Kamarudin [1]

[1]*Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Seri Kembangan, Malaysia*
[2]*Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, Kuala Terengganu, Malaysia*

ARTICLE INFO

Article history:
Received 2 December 2021
Received in revised form
10 March 2022
Accepted 11 April 2022

Keywords:
Software-defined network
Traffic forwarding
K-Shortest path
Dijkstra algorithm

ABSTRACT

Packet routing has always been an issue that affects network performance. The traditional protocol is known to work with local information and thus not able to produce a global optimal decision. In the software-defined network (SDN), its centralized controller obtains unified access to the entire network topology information and has the ability to partially solve traditional network packet forwarding problems. SDN controller uses the Dijkstra algorithm to find the shortest path calculated from the source node to the destination node. However, constraints such as the need to bypass the node which has a high rate of failure exist to prevent the Dijkstra algorithm from meeting this demand. In practice, we need not only consider the shortest path but also consider the second short path, the third short path, and so on. K-shortest paths algorithm discovers a set of paths ordered in the most optimal, optimal, and suboptimal, has a very wide application is integrated into SDN controller to handle routing functionality. In this study, instead of only the number of hops, bandwidth and delay are adhered to within the k-shortest path to select the final route. Under this circumstance, different topologies are examined in response to network bandwidth and packet delay. The experiment shows that tree topology is best suited for improving bandwidth while simple topology for reducing delay.

1. Introduction

In general terms, network forwarding is the basic function of communication, which completes the transmission of information in the network and realizes orderly data exchange. Through the centralized control of the Software-Defined Network (SDN) (Nisar et al., 2020; Benzekki et al., 2016), the basic forwarding algorithm and the shortest path algorithm based on the number of hops can be easily realized. However, the number of network hops is not the only state that determines the pros and cons of a path. In addition to the number of hops, there are also other parameters such as bandwidth and delay. This study develops the shortest path forwarding application based on traffic through the SDN controller Ryu using k-shortest paths (KSP) (Chen et al., 2020; Brander and Sinclair, 1996).

Nowadays, the computer network is developing rapidly. Since its basic architecture was proposed, there has not been an essential development. Moreover, with the rapid development of the Internet, the requirements for the network continue to increase. The problem that follows is that the traditional network architecture has been unable to meet today's technical requirements from many different aspects. As such, we come across a network such as a wireless sensor network (Shallahuddin et al., 2020; Muzakkari et al., 2018), a vehicular network (Elias et al., 2019; Karagiannis et al., 2011), and a wireless body area network (Latré et al., 2011). In another context, Software Defined Networking (SDN) was proposed and developed rapidly and has achieved excellent results (Rout et al., 2021; Akin and Korkmaz, 2019; Xia et al., 2014). The core idea of SDN (shown in Fig. 1) is to separate the control plane and data plane. The data plane is only responsible for data packet forwarding. Forwarding decisions are the responsibility of the control plane, which makes network configuration easy. The configuration of the network also solves many problems of the traditional network distributed architecture, such as uncontrollable networks and complicated configuration.

* Corresponding Author.
Email Address: fadzil@unisza.edu.my (M. F. A. Kadir)
https://doi.org/10.21833/ijaas.2022.06.018
Corresponding author's ORCID profile:
https://orcid.org/0000-0002-3647-5134

The overall framework of this project is to build network-aware services and shortest path applications based on network services. The main content includes Network Resource Awareness Module, Network Monitoring Module, and Network Packet Forwarding Module. Network resource awareness handles the real-time sensing of changes in the network resources including topology information such as switch and port, and host information that includes traffic statistics. In addition to the physical resource information, the network information also includes information such as logical links. Obtaining flow table information can obtain corresponding logical connection information.

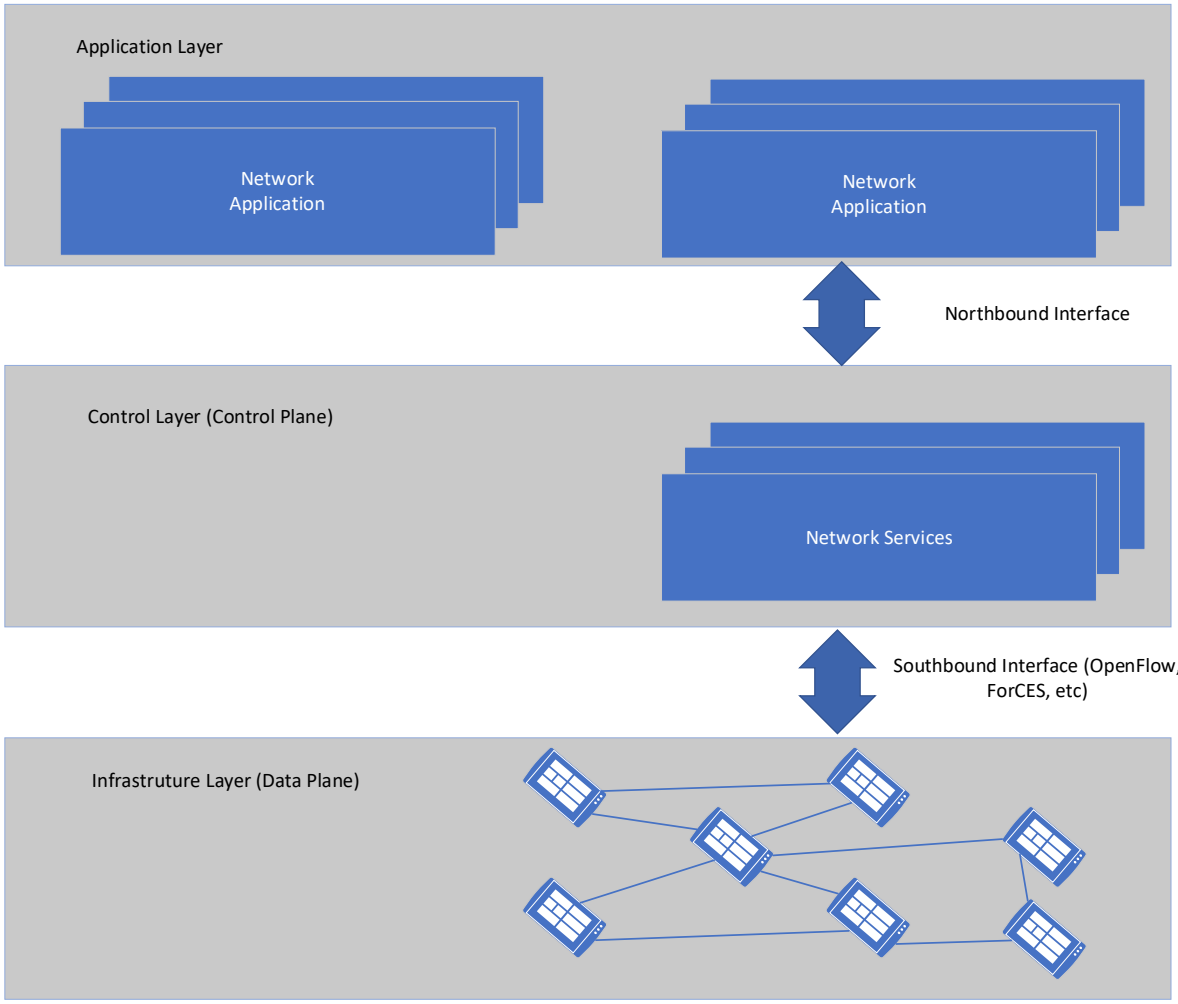


Fig. 1: SDN Architecture

At the same time, obtaining the statistics of the network's data traffic has played an important role in preventing network failures and optimizing the network reasonably. The network traffic monitoring module implements monitoring of port traffic and flows entry traffic. The application can periodically obtain the traffic information and output the display in the terminal. Finally, the routing algorithm calculates the optimal K paths based on the number of hops and then selects the path with the largest available bandwidth and minimal packet delay.

This paper is organized as follows: Section 1 introduces readers to the SDN network, defined the problem statement based on the literature review also explains the objectives and scope of the project. Section 2 reviews some literature related to network awareness, network monitoring, and forwarding services in SDN. Section 3 designs and develops the modules that are responsible for carrying out this research. Section 4 presents the findings from our research and provides some discussions. Section 5 offers a final conclusion to our works.

2. Literature review

A key activity in SDN is traffic engineering (TE), which measures and manages network traffic. Akyildiz et al. (2014) described that traffic engineering can improve the utilization of network resources by analyzing real-time traffic, predicting traffic, and designing appropriate routing mechanisms. In order to ensure the effective conduct of these activities, network monitoring is very important. The design of network parameters is a reflection of a good monitoring. The value of the current state of the network is represented by network parameters. One of the parameters used to measure network performance is bandwidth

utilization which refers to the efficiency of receiving and sending information in bandwidth per second. Bandwidth allocation is to ensure the transmission of real-time services within a limited bandwidth, so as not to cause network congestion due to too much traffic.

In recent years, many monitoring tools for measuring bandwidth utilization were developed. Luong et al. (2016) introduced two modules into the SDN controller responsible for monitoring link traffic throughput and forwarding packets. The data packet information between switches is determined by the throughput monitor module, and the packet forwarding module implements the forwarding algorithm. Consequently, any application can be monitored by the throughput monitoring module for its usage of bandwidth. Sinha et al. (2017) proposed an SDN controller that provides packet loss statistics using an online flow-by-flow and port-by-port monitoring and measurement which includes traffic monitoring between two pairs of source and target hosts. Through this process, the traffic information obtained will be more accurate. Bi et al. (2019) proposed an SDN controller that promotes the separation of the control plane and data plane as a form of centralized control and allocation of network resources and maintenance.

On the other hand, the research on the shortest path problem began in the late 1950s. Many problems in real life belong to the shortest path problem, and that includes the routing problem. However, in cases such that of packet communication, it is not enough to consider only the shortest distance. It is also necessary to consider the second shortest path, the next shortest path, that is, the k-shortest path. The shortest path algorithm based on the Dijkstra algorithm was proposed by Dijkstra (1959). Noto and Sato (2000) stated that the Dijkstra algorithm adopts the principle of the greedy algorithm and is currently the most widely used method to solve the shortest path problem. The algorithm is mainly to solve the problem of the shortest path between the source node and the destination node. Its main feature is that the point found in each iteration is the point closest to the source point except the point that has been found. The Floyd algorithm is mainly used to find the shortest path between multiple source points in the weighted graph. Tsitsiashvili and Losev (2008) showed that the points in the algorithm are gradually added, so the algorithm is also called the interpolation point method. Floyd algorithm is also called the recursive closure method from the principle of the algorithm. The essence of this algorithm is to multiply the adjacent matrix of the graph G n times, and its time complexity is $O(n^3)$. The Bellman-Ford algorithm is also an algorithm for finding the shortest distance, Cheng et al. (1989) stated that the difference between the Bellman-Ford algorithm and the previous two algorithms is that it allows the weights in the graph to be negative. Therefore, in the case of negative weights in the graph, The Bellman-Ford algorithm can be used, but the Dijkstra algorithm cannot be used in this case. The Bellman-Ford algorithm first initializes the distance between the point in the graph and the source point and performs relaxation operations on the edges in the graph, so that the distance between each point and the source point is gradually reduced, repeating n-1 times, If the relaxation can be performed, it means that there are negative weights in the graph, and the shortest path cannot be obtained. On the contrary, it means that the weights in the graph are all positive, and the shortest path can be solved. The time complexity of the Bellman-Ford algorithm is O(n*e) (where n is the number of vertices in the graph, and e is the number of edges in the graph).

In many cases, it is not enough to consider only the shortest distance. You also need to consider the next shortest path, the next shortest path, and so on, which is the K shortest path (KSP). The K shortest path algorithm has been applied in many aspects of our daily life and has great practical application value. The KSP algorithm, in its literal meaning, is K shortest path. KSP and the classic Dijkstra algorithm have a certain connection because the Dijkstra algorithm needs to be called frequently in the KSP calculation. The basic idea of the KSP algorithm in the SDN controller is to find K paths in the network topology between the source node and the destination node, and then select the paths that meet the requirements according to different business requirements. Among them, the number of alternative paths, that is, the K value can be changed according to specific needs. The advantage of using the KSP algorithm in SDN is that it can meet the needs of business diversity, reduce the failure rate of the network, and reduce the average service resistance rate of the network.

The balance of an algorithm can best be achieved through the two indicators of time complexity and space complexity. Trading time for space refers to reducing space complexity at the expense of time complexity. Conversely, sacrificing space complexity for reducing time complexity is to trade space for time. In addition, the performance of the algorithm is also affected by many practical situations such as the size of the data that needs to be processed, the frequency that needs to be used, and the environment in which the algorithm is running. The time complexity of the KSP algorithm is O(m+nlogn) while its space complexity is O(m+Kn).

3. Methodology

The idea of this experiment is to create a network awareness module, a network monitoring module, and a shortest route module based on network information. The network awareness module grasps the real-time dynamics of the network information including that of switches, ports, and hosts. The network monitoring module is responsible for monitoring flow-based traffic statistics and port-based traffic statistics. After mastering this key network information, the controller can make the

most correct routing decision while completing network communication.

3.1. Network awareness

The network awareness module is used to perceive real-time changes in network resources, including changes in topology information and host information. In any network application, accessibility is the most basic requirement. The centralized control of the SDN network enables the controller to make the best decision based on global information without using distributed routing algorithms on the switching nodes. Therefore, awareness of network resources is the most basic service for SDN applications.

3.2. Network monitor

In addition to physical resource information, network information also includes information such as logical links; obtaining flow table information can obtain corresponding logical connection information. In addition, obtaining statistics on-network data traffic plays an important role in preventing network failures and rationally optimizing the network. The network flow monitoring module realizes the monitoring of port flow and flows entry flow. The application can periodically obtain traffic information and display it in the terminal.

In this study, we use Mininet that runs on Linux Ubuntu 16.04 operating system to simulate a realistic virtual network that runs a real kernel, OpenFlow virtual switches and hosts and application code on a single machine (VM, cloud, or native), and RYU controller as an SDN controller responsible for hosting the forwarding algorithm as shown in Fig. 2. In addition, a python package called Networkx is used to store topology information and traffic information.

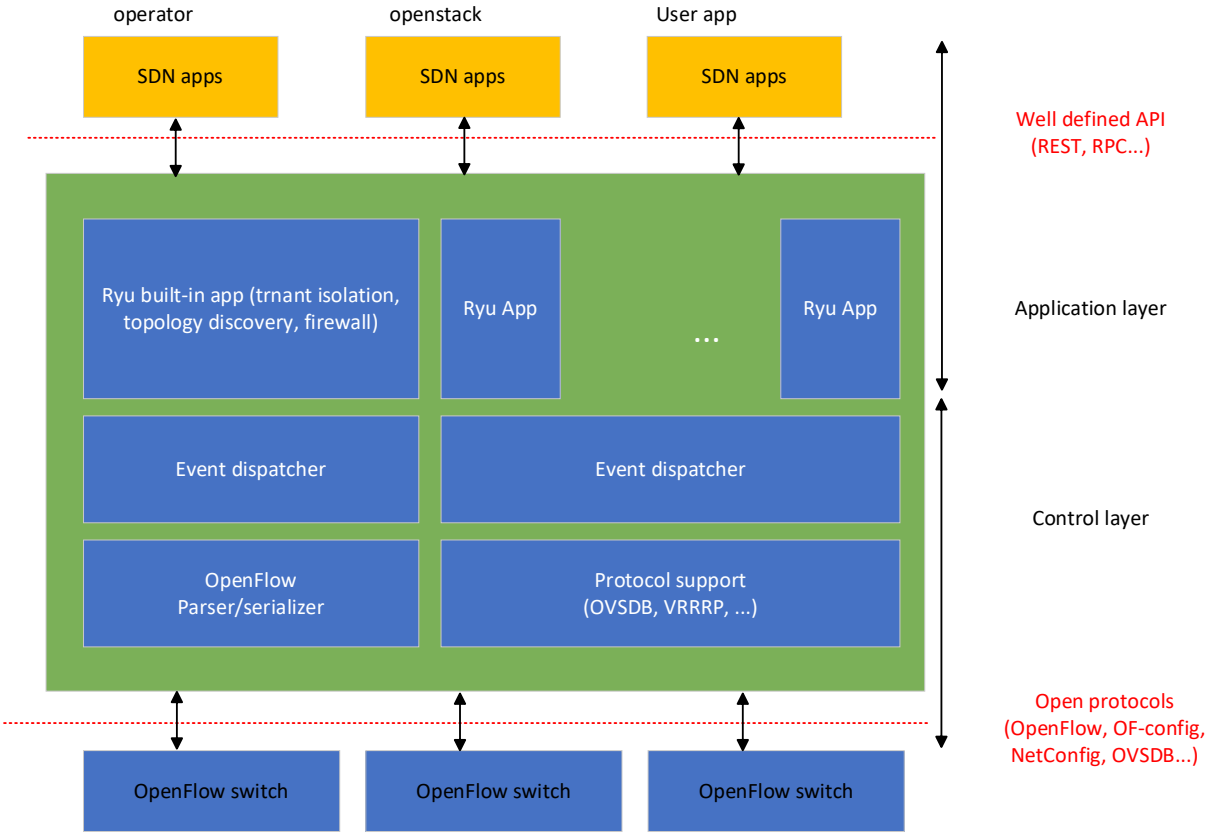


Fig. 2: Ryu SDN controller components

In this experiment, we examine three network topologies namely simple, ring and tree topologies having different environmental complexities, one is simple topology, one is medium complex topology, and the other is complex topology. The design principle is based on the following procedure:

1) Flow monitoring principle:

i) The controller periodically sends the acquisition statistics message to the switch, requesting the switch information:

(a) Port traffic statistics
(b) Request flow table entry statistics

ii) Calculate and calculate flow information based on switch statistics:

(a) Flow rate formula: speed=(s(t1)-s(t0))/(t1-t0)
(b) Remaining bandwidth formula: free_bw=capability−speed

2) Traffic-based optimal path forwarding principle:

i) Topology information discovery:

(a) Send LLDP packets periodically to discover link information
(b) Use Networkx to store topology information.

ii) Traffic information collection:

(a) Get traffic information from the Monitor module
(b) Use Networkx to store traffic information.

iii) Calculate optimal path forwarding based on traffic:

(a) Calculate K shortest paths based on hop count: Shortest_k_path

(b) Choose the path with the largest bandwidth available

3.3. Shortest path application flowchart

Based on the data provided by the network awareness module and network traffic monitoring module, we can do many things, such as load balancing, traffic scheduling, and secure access. This section introduces the shortest path application based on network resources. The reference system for measuring the shortest path is the number of hops. With a slight modification, it can be changed into a scheme of remaining bandwidth, delay, or weighting of multiple reference systems. Fig. 3 shows the shortest path application flow chart.

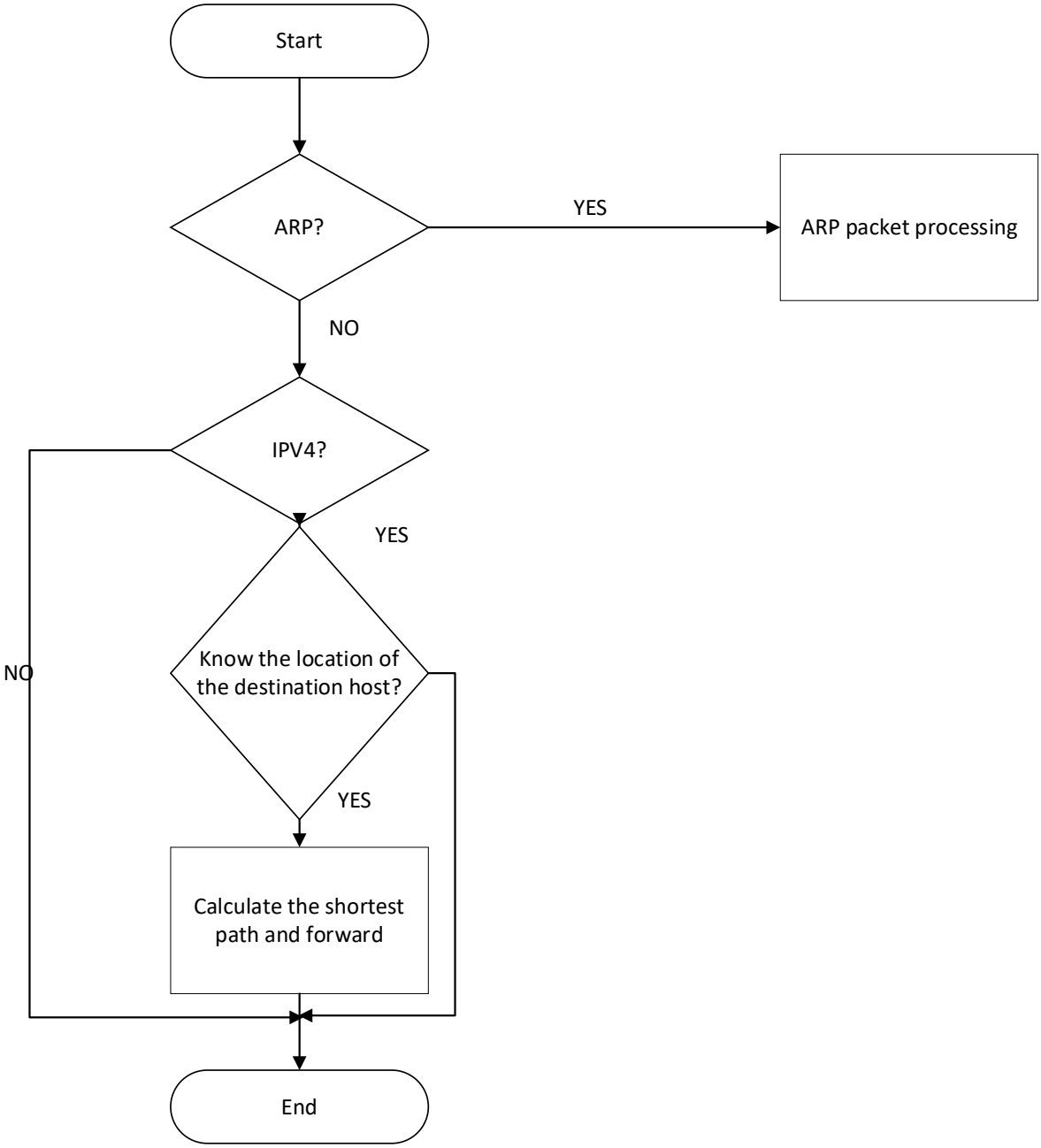


Fig. 3: Shortest path application flowchart

First, a query to the host table is executed, if the search is successful, then query the host location table, and then the controller directly sends the ARP data packet to the corresponding port. At this time,

the controller does not act as a proxy for ARP. When the target host replies to the ARP, it sends the data packet directly to the access port of the source host. This completes the ARP learning process. Since the host's access information and network information have been mastered at this time when an ICMP or other data packet initiates a packet_in event, the access switch can be queried according to the two IPs of the source and destination, and then the shortest path can be calculated based on the topology information, thereby completing the shortest route. If you want to use other reference standards to calculate the shortest path, you only need to modify the algorithm for calculating the shortest path.

When the network is initialized, the controller has no way to find the silent host. The reason is that we did not carry out the DHCP assignment, which caused the controller to not grasp the host's IP/MAC information. So the first step we need to deal with is ARP. The processing flow is shown in Fig. 4.

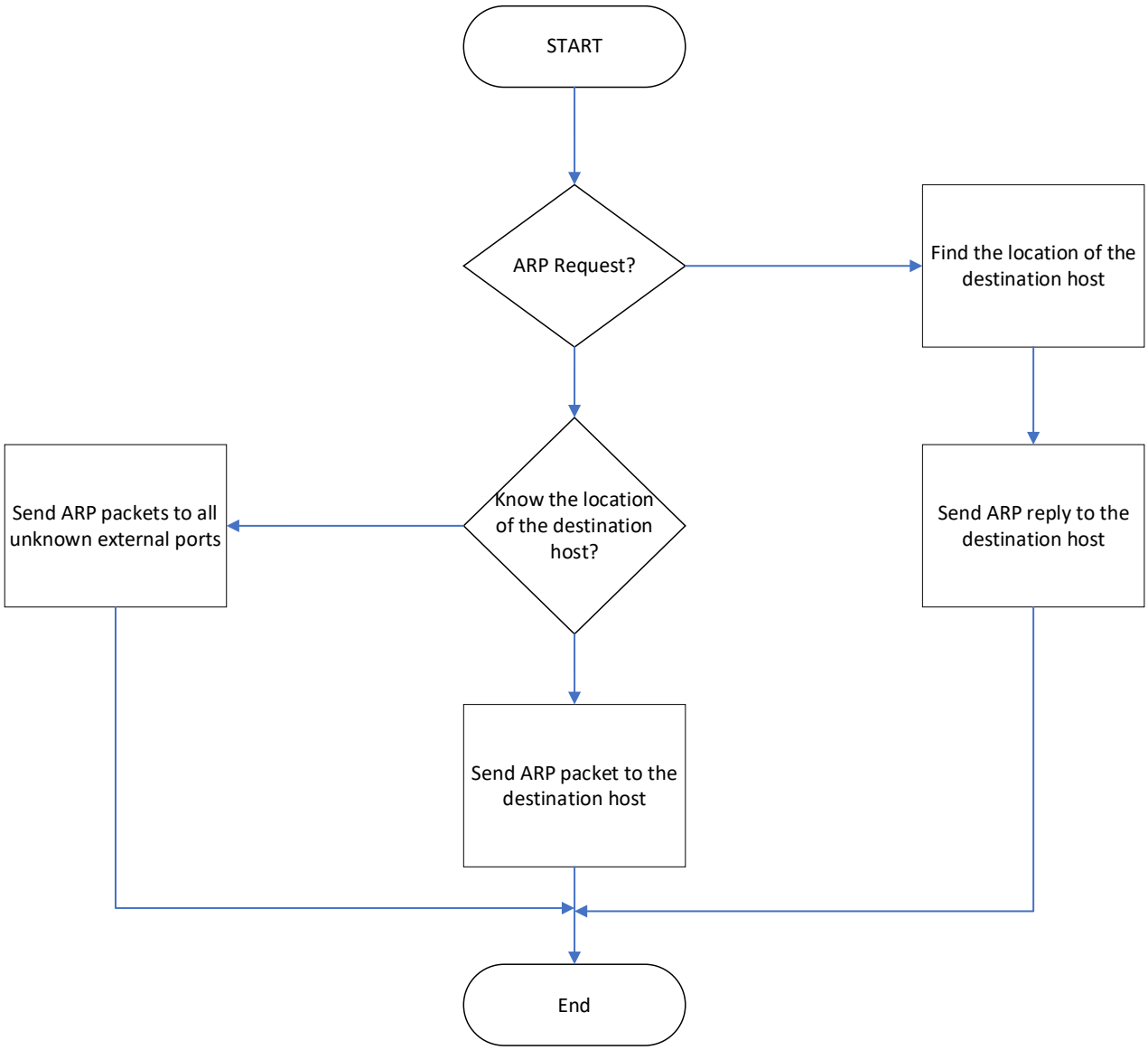


Fig. 4: ARP processing flowchart

This application assumes that when the host initiates communication, ARP must be initiated first, and ARP information cannot be obtained through other means. Otherwise, the controller cannot obtain the access information of the destination host, and the routing cannot be completed. For hosts outside the domain, they only need to send them to the egress gateway when the destination is not found. At this time, you need to use concepts such as subnet mask, network segment, and routing. This application is only for simple local area network calculation paths.

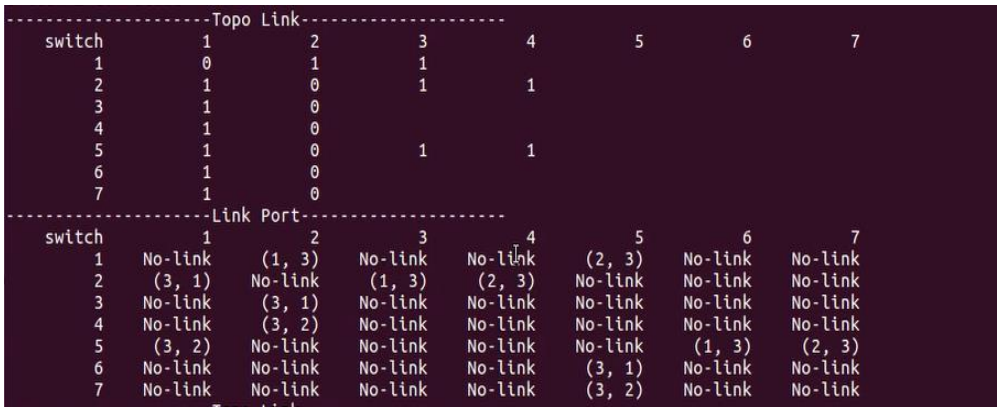
4. Result and discussion

Network Awareness is a set of Ryu applications for collecting the basic network information including the topology, link delay, and link-free bandwidth. The Shortest_forwarding.py application can achieve the shortest path forwarding based on Hop, Delay, and Bandwidth. This is achievable by adding a "weight" argument when starting Ryu. Moreover, we can set the "k-paths" argument to support the k-shortest paths computation. Fortunately, our application supports load balance

based on dynamic traffic information. In this version, we rely upon Networkx's data structure to store network topology. In addition, we also use Networkx's function to calculate the shortest path.

4.1. Network awareness application test

Fig. 5 shows the output from of Network awareness module, which includes topology link information and port information. It shows that there are 7 switches, divided into a number axis and a horizontal axis. The corresponding 0 means there is no link between the switches, and 1 means there is a link between the switches. In the port information part, if there is no link between the switches, naturally there is no port, and the output information is No-link. For example, switch 1 and switch 2 have a link, and are connected to port 3 in switch 2 through port 1 in switch 1, and the output information is (1, 3). Moreover, if the topology information in the Mininet changes, the output result of the terminal is updated in real-time.



----------Topo Link----------

switch	1	2	3	4	5	6	7
1	0	1	1				
2	1	0	1	1			
3	1	0					
4	1	0					
5	1	0	1	1			
6	1	0					
7	1	0					

----------Link Port----------

switch	1	2	3	4	5	6	7
1	No-link	(1, 3)	No-link	No-link	(2, 3)	No-link	No-link
2	(3, 1)	No-link	(1, 3)	(2, 3)	No-link	No-link	No-link
3	No-link	(3, 1)	No-link	No-link	No-link	No-link	No-link
4	No-link	(3, 2)	No-link	No-link	No-link	No-link	No-link
5	(3, 2)	No-link	No-link	No-link	No-link	(1, 3)	(2, 3)
6	No-link	No-link	No-link	No-link	(3, 1)	No-link	No-link
7	No-link	No-link	No-link	No-link	(3, 2)	No-link	No-link

Fig. 5: Network resource information

4.2. Network monitor application test

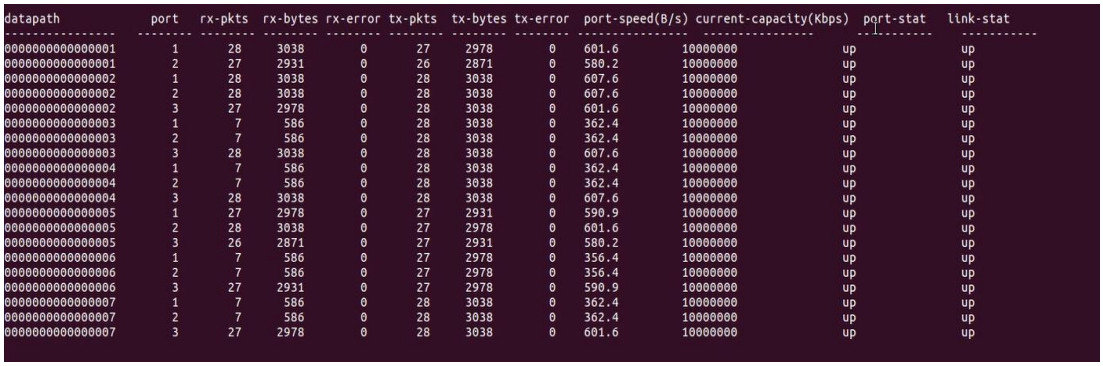
The results of Figs. 6 and 7 are from the Network monitoring module. The network information also includes information such as logical links. Obtaining flow table information can obtain corresponding logical connection information. In addition, obtaining the statistics of the network's data traffic has played an important role in preventing network failures and optimizing the network reasonably. The network traffic monitoring module implements monitoring of port traffic and flows entry traffic. The result shows the traffic dynamics that occur at each port on each switch, which is more intuitively displayed in the form of data.

4.3. Shortest path application test

Fig. 8 shows the resulting path obtained based on the data provided by the aforementioned network awareness module and network traffic monitoring module. For example, two results from h1 to h7 are [3, 2, 1, 5, 7], [5, 7].

4.4. Bandwidth analysis

We analyze the relationship between topology and bandwidth at different levels of complexity, and how the number of switches affects bandwidth. Fig. 9 shows a graph obtained by simple topology. The number of starting switches starts at 16, and the available bandwidth is 128 MB/s. The number of switches increases by 16 sequentially. The overall appearance first fell, then rose, then fell, then slowly rose, and finally slowly tends to a flat state. The data shows that the number of switches affects the available bandwidth, the more the number of switches, the less the available bandwidth.



datapath	port	rx-pkts	rx-bytes	rx-error	tx-pkts	tx-bytes	tx-error	port-speed(B/s)	current-capacity(Kbps)	port-stat	link-stat
0000000000000001	1	28	3038	0	27	2978	0	601.6	10000000	up	up
0000000000000001	2	27	2931	0	26	2871	0	580.2	10000000	up	up
0000000000000002	1	28	3038	0	28	3038	0	607.6	10000000	up	up
0000000000000002	2	28	3038	0	28	3038	0	607.6	10000000	up	up
0000000000000002	3	27	2978	0	28	3038	0	601.6	10000000	up	up
0000000000000003	1	7	586	0	28	3038	0	362.4	10000000	up	up
0000000000000003	2	7	586	0	28	3038	0	362.4	10000000	up	up
0000000000000003	3	28	3038	0	28	3038	0	607.6	10000000	up	up
0000000000000004	1	7	586	0	28	3038	0	362.4	10000000	up	up
0000000000000004	2	7	586	0	28	3038	0	362.4	10000000	up	up
0000000000000004	3	28	3038	0	28	3038	0	607.6	10000000	up	up
0000000000000005	1	27	2978	0	27	2931	0	590.9	10000000	up	up
0000000000000005	2	28	3038	0	27	2978	0	601.6	10000000	up	up
0000000000000005	3	26	2871	0	27	2931	0	580.2	10000000	up	up
0000000000000006	1	7	586	0	27	2978	0	356.4	10000000	up	up
0000000000000006	2	7	586	0	27	2978	0	356.4	10000000	up	up
0000000000000006	3	27	2931	0	27	2978	0	590.9	10000000	up	up
0000000000000007	1	7	586	0	28	3038	0	362.4	10000000	up	up
0000000000000007	2	7	586	0	28	3038	0	362.4	10000000	up	up
0000000000000007	3	27	2978	0	28	3038	0	601.6	10000000	up	up

Fig. 6: Port traffic statistics

datapath	in-port	ip-dst	out-port	packets	bytes	flow-speed(B/s)
0000000000000001	1	10.0.0.5	2	2	196	0.0
0000000000000001	1	10.0.0.5	2	2	196	0.0
0000000000000001	1	10.0.0.5	2	2	196	0.0
0000000000000001	1	10.0.0.5	2	2	196	0.0
0000000000000001	1	10.0.0.6	2	2	196	0.0
0000000000000001	1	10.0.0.6	2	2	196	0.0
0000000000000001	1	10.0.0.6	2	2	196	0.0
0000000000000001	1	10.0.0.6	2	2	196	0.0
0000000000000001	1	10.0.0.7	2	2	196	0.0
0000000000000001	1	10.0.0.7	2	2	196	0.0
0000000000000001	1	10.0.0.7	2	2	196	0.0
0000000000000001	1	10.0.0.7	2	2	196	0.0
0000000000000001	1	10.0.0.8	2	2	196	0.0
0000000000000001	1	10.0.0.8	2	2	196	0.0
0000000000000001	1	10.0.0.8	2	2	196	0.0
0000000000000001	1	10.0.0.8	2	2	196	0.0
0000000000000001	2	10.0.0.1	1	2	196	0.0
0000000000000001	2	10.0.0.1	1	2	196	0.0
0000000000000001	2	10.0.0.1	1	2	196	0.0
0000000000000001	2	10.0.0.1	1	2	196	0.0
0000000000000001	2	10.0.0.2	1	2	196	0.0
0000000000000001	2	10.0.0.2	1	2	196	0.0
0000000000000001	2	10.0.0.2	1	2	196	0.0
0000000000000001	2	10.0.0.2	1	2	196	0.0
0000000000000001	2	10.0.0.3	1	2	196	0.0
0000000000000001	2	10.0.0.3	1	2	196	0.0
0000000000000001	2	10.0.0.3	1	2	196	0.0
0000000000000001	2	10.0.0.3	1	2	196	0.0
0000000000000001	2	10.0.0.4	1	2	196	0.0
0000000000000001	2	10.0.0.4	1	2	196	0.0
0000000000000001	2	10.0.0.4	1	2	196	0.0
0000000000000001	2	10.0.0.4	1	2	196	0.0
0000000000000002	1	10.0.0.3	2	2	196	0.0
0000000000000002	1	10.0.0.3	2	2	196	0.0
0000000000000002	1	10.0.0.4	2	2	196	0.0
0000000000000002	1	10.0.0.4	2	2	196	0.0
0000000000000002	1	10.0.0.5	3	1	98	280.8
0000000000000002	1	10.0.0.5	3	2	196	280.8
0000000000000002	1	10.0.0.6	3	2	196	0.0
0000000000000002	1	10.0.0.6	3	2	196	0.0
0000000000000002	1	10.0.0.7	3	2	196	0.0
0000000000000002	1	10.0.0.7	3	2	196	0.0
0000000000000002	1	10.0.0.8	3	2	196	0.0
0000000000000002	1	10.0.0.8	3	2	196	0.0

Fig 7: Flow entry traffic statistics

```
[PATH]10.0.0.1<-->10.0.0.2: [3]
10.0.0.3 location is not found.
[PATH]10.0.0.1<-->10.0.0.3: [3, 2, 4]
10.0.0.4 location is not found.
[PATH]10.0.0.1<-->10.0.0.4: [3, 2, 4]
10.0.0.5 location is not found.
[PATH]10.0.0.1<-->10.0.0.5: [3, 2, 1, 5, 6]
[PATH]10.0.0.1<-->10.0.0.5: [2, 1, 5, 6]
10.0.0.6 location is not found.
[PATH]10.0.0.1<-->10.0.0.6: [3, 2, 1, 5, 6]
[PATH]10.0.0.1<-->10.0.0.6: [5, 6]
10.0.0.7 location is not found.
[PATH]10.0.0.1<-->10.0.0.7: [3, 2, 1, 5, 7]
[PATH]10.0.0.1<-->10.0.0.7: [5, 7]
10.0.0.8 location is not found.
[PATH]10.0.0.1<-->10.0.0.8: [3, 2, 1, 5, 7]
```

```
[PATH]10.0.0.2<-->10.0.0.6: [3, 2, 1, 5, 6]
[PATH]10.0.0.2<-->10.0.0.6: [5, 6]
[PATH]10.0.0.2<-->10.0.0.7: [3, 2, 1, 5, 7]
[PATH]10.0.0.2<-->10.0.0.7: [5, 7]
[PATH]10.0.0.2<-->10.0.0.8: [3, 2, 1, 5, 7]
[PATH]10.0.0.2<-->10.0.0.8: [5, 7]
[PATH]10.0.0.3<-->10.0.0.4: [4]
[PATH]10.0.0.3<-->10.0.0.5: [4, 2, 1, 5, 6]
[PATH]10.0.0.3<-->10.0.0.5: [2, 1, 5, 6]
[PATH]10.0.0.3<-->10.0.0.6: [4, 2, 1, 5, 6]
[PATH]10.0.0.3<-->10.0.0.6: [5, 6]
[PATH]10.0.0.3<-->10.0.0.7: [4, 2, 1, 5, 7]
[PATH]10.0.0.3<-->10.0.0.7: [5, 7]
[PATH]10.0.0.3<-->10.0.0.8: [4, 2, 1, 5, 7]
[PATH]10.0.0.3<-->10.0.0.8: [2, 1, 5, 7]
[PATH]10.0.0.4<-->10.0.0.5: [4, 2, 1, 5, 6]
[PATH]10.0.0.4<-->10.0.0.5: [5, 6]
[PATH]10.0.0.4<-->10.0.0.6: [4, 2, 1, 5, 6]
[PATH]10.0.0.4<-->10.0.0.6: [5, 6]
[PATH]10.0.0.4<-->10.0.0.7: [4, 2, 1, 5, 7]
[PATH]10.0.0.4<-->10.0.0.7: [5, 7]
[PATH]10.0.0.4<-->10.0.0.8: [4, 2, 1, 5, 7]
[PATH]10.0.0.4<-->10.0.0.8: [5, 7]
[PATH]10.0.0.5<-->10.0.0.6: [6]
[PATH]10.0.0.5<-->10.0.0.7: [6, 5, 7]
[PATH]10.0.0.5<-->10.0.0.8: [6, 5, 7]
[PATH]10.0.0.6<-->10.0.0.7: [6, 5, 7]
[PATH]10.0.0.6<-->10.0.0.8: [6, 5, 7]
[PATH]10.0.0.7<-->10.0.0.8: [7]
```

Fig. 8: Flow entry traffic statistics

Fig. 10 is based on a ring topology. We observed that, as the number of switches increases, the available bandwidth decreases. And when the number of switches is increased to 32, the available bandwidth decreases most obviously.

Fig. 11 is based on tree topology. Observably, the overall situation is declining. When the number of switches increases from 64 to 80, the available bandwidth rises slightly and then flattens out again.

For all three types of topologies, the more the number of switches, the slower the available bandwidth. In other words, more nodes can also affect the bandwidth. The larger the node, the smaller the available bandwidth.

4.5. Delay analysis

We analyzed the relationship between the delays in the number of switches. Fig. 12 is for a simple topology graph. The number of switches starts at 10 with a delay of 0.9 seconds. But as the number of switches increases, the delay also increases.

Fig. 13 is based on a more complicated ring topology. Likewise, the number of switches increases, and the delay also increases. Compared with the simple topology in Fig. 12, this topology is more complicated and consequently, the delay gets higher.

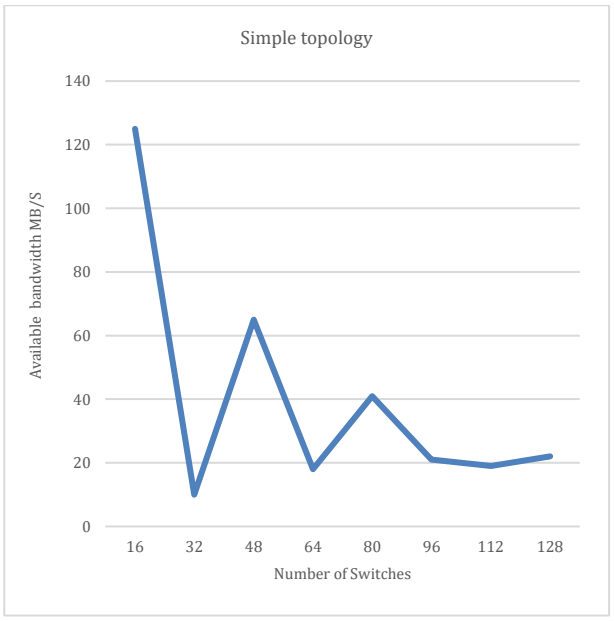


Fig. 9: Simple topology-bandwidth

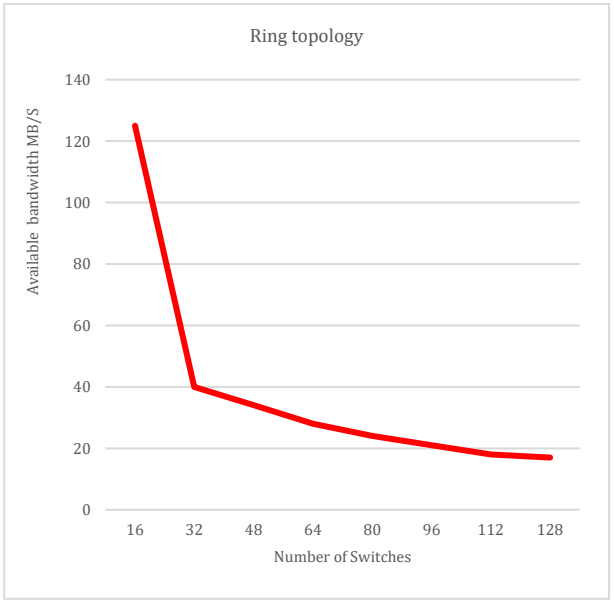


Fig. 10: Ring topology- bandwidth

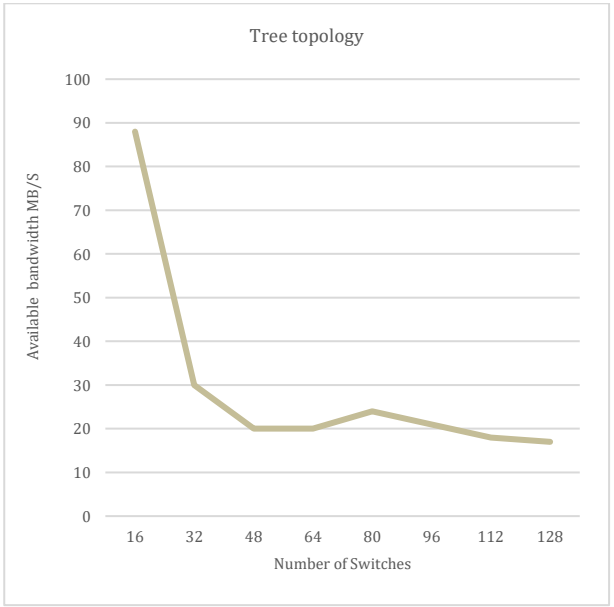


Fig. 11: Tree topology- bandwidth

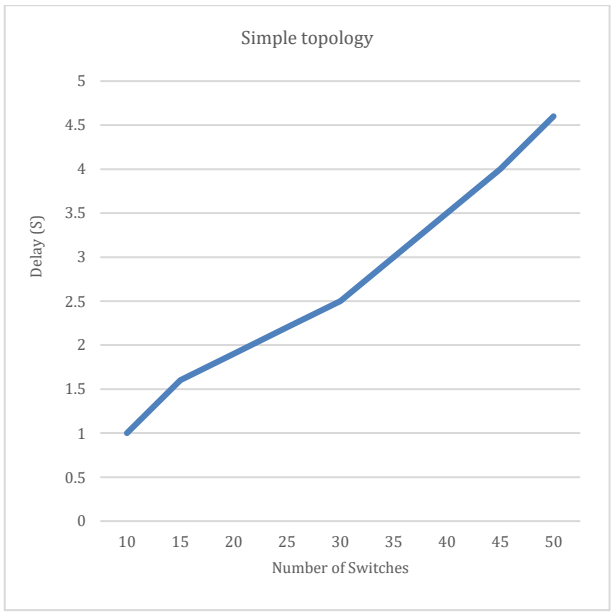


Fig. 12: Simple topology- Delay

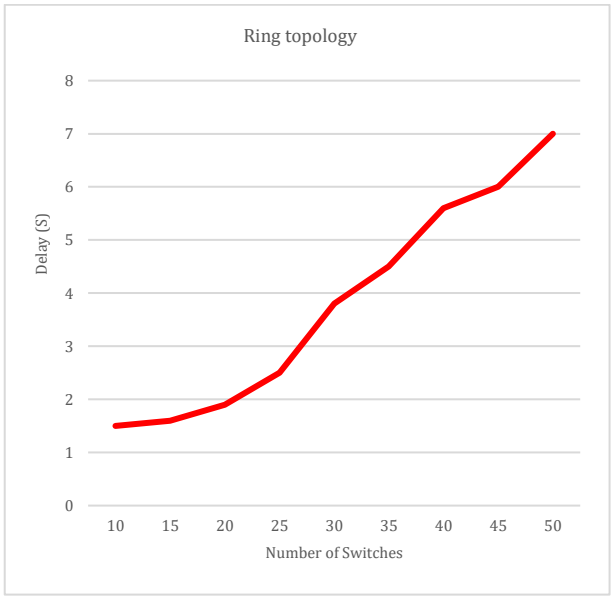


Fig. 13: Ring topology-delay

Fig. 14 is based on tree topology. Similar to Fig. 12 and Fig. 13, the delay increases as the number of switches increases. But as the tree topology is more complicated than the other two, thus the delay is much higher. When the number of switches is 50, the delay is 13.7 seconds. Approximately 3 times the simple topology and 2 times of ring topology.

Based on the three topologies, as the number of switches increases, the delay also increases. It was discovered that the delay is also related to the complexity of the topology. The more complex the network structure, the greater the number of switches and thus the greater the delay time increment.

5. Conclusions and future works

In this paper, we studied SDN network architecture and dynamic routing through Ryu controller and Mininet network simulation tools. The topology and resources of the entire network are

collected through the SDN controller to form a topology view and a resource view, which realizes the control function of the OpenFlow switch. The network topology and the volume mapping information of the switching equipment are saved and summarized, and the shortest path is calculated according to the Dijkstra algorithm at the same time, the resonance sensing function is added to obtain the bandwidth, delay, and packet loss rate of the network compound.

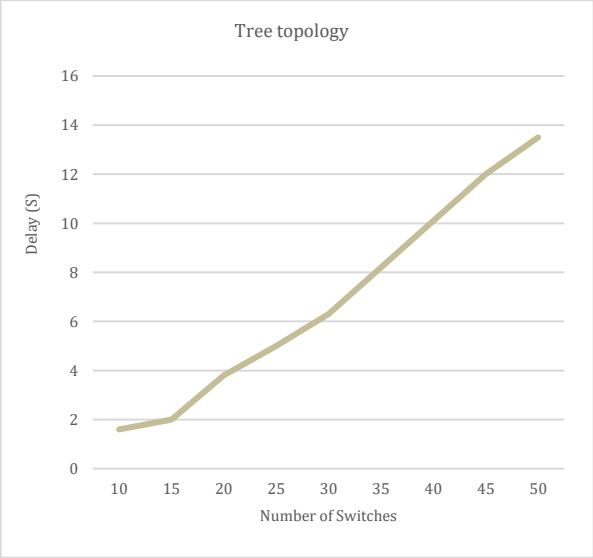


Fig. 14: Tree topology-delay

The simulation results show that the development of the Ryu controller application combined with the internal application can realize the dynamic routing function and complete the automatic flow table rule issuance. The application layer in the SDN architecture is feasible for developing functions such as network protocols, and the SDN controller can realize centralized management and automatic configuration of virtual network devices according to optimized routing algorithms.

Acknowledgment

This project is partially funded by the Center for Research Excellence, Incubation Management Center, Universiti Sultan Zainal Abidin, Malaysia.

Compliance with ethical standards

Conflict of interest

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

Akin E and Korkmaz T (2019). Comparison of routing algorithms with static and dynamic link cost in software defined networking (SDN). IEEE Access, 7: 148629-148644. https://doi.org/10.1109/ACCESS.2019.2946707

Akyildiz IF, Lee A, Wang P, Luo M, and Chou W (2014). A roadmap for traffic engineering in SDN-OpenFlow networks. Computer Networks, 71: 1-30. https://doi.org/10.1016/j.comnet.2014.06.002

Benzekki K, El Fergougui A, and Elbelrhiti Elalaoui A (2016). Software-defined networking (SDN): A survey. Security and Communication Networks, 9(18): 5803-5833. https://doi.org/10.1002/sec.1737

Bi Y, Han G, Lin C, Peng Y, Pu H, and Jia Y (2019). Intelligent quality of service aware traffic forwarding for software-defined networking/open shortest path first hybrid industrial internet. IEEE Transactions on Industrial Informatics, 16(2): 1395-1405. https://doi.org/10.1109/TII.2019.2946045

Brander AW and Sinclair MC (1996). A comparative study of k-shortest path algorithms. In: Merabti M, Carew M, and Ball F (Eds.), Performance engineering of computer and telecommunications systems: 370-379. Springer, London, UK. https://doi.org/10.1007/978-1-4471-1007-1_25

Chen BY, Chen XW, Chen HP, and Lam WH (2020). Efficient algorithm for finding k shortest paths based on re-optimization technique. Transportation Research Part E: Logistics and Transportation Review, 133: 101819. https://doi.org/10.1016/j.tre.2019.11.013

Cheng C, Riley R, Kumar SP, and Garcia-Luna-Aceves JJ (1989). A loop-free extended Bellman-Ford routing protocol without bouncing effect. ACM SIGCOMM Computer Communication Review, 19(4): 224-236. https://doi.org/10.1145/75247.75269

Dijkstra EW (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1(1): 269-271. https://doi.org/10.1007/BF01386390

Elias SJ, Hatim SM, Darus MY, Abdullah S, Jasmis J, Ahmad RB, and Khang AWY (2019). Congestion control in vehicular Adhoc network: A survey. Indonesian Journal of Electrical Engineering and Computer Science, 13(3): 1280-1285. https://doi.org/10.11591/ijeecs.v13.i3.pp1280-1285

Karagiannis G, Altintas O, Ekici E, Heijenk G, Jarupan B, Lin K, and Weil T (2011). Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions. IEEE Communications Surveys and Tutorials, 13(4): 584-616. https://doi.org/10.1109/SURV.2011.061411.00019

Latré B, Braem B, Moerman I, Blondia C, and Demeester P (2011). A survey on wireless body area networks. Wireless Networks, 17(1): 1-18. https://doi.org/10.1007/s11276-010-0252-4

Luong DH, Outtagarts A, and Hebbar A (2016). Traffic monitoring in software defined networks using open daylight controller. In the International Conference on Mobile, Secure, and Programmable Networking, Springer, Paris, France: 38-48. https://doi.org/10.1007/978-3-319-50463-6_4

Muzakkari BA, Mohamed MA, Kadir MF, Mohamad Z, and Jamil N (2018). Recent advances in energy efficient-QoS aware MAC protocols for wireless sensor networks. International Journal of Advanced Computer Research, 8(38): 212-228. https://doi.org/10.19101/IJACR.2018.837016

Nisar K, Jimson ER, Hijazi MHA, Welch I, Hassan R, Aman AHM, and Khan S (2020). A survey on the architecture, application, and security of software defined networking: Challenges and open issues. Internet of Things, 12: 100289. https://doi.org/10.1016/j.iot.2020.100289

Noto M and Sato H (2000). A method for the shortest path search by extended Dijkstra algorithm. In the SMC 2000 Conference Proceedings: 2000 IEEE International Conference on Systems, Man and Cybernetics. 'Cybernetics Evolving to Systems, Humans, Organizations, and Their Complex Interactions' (cat. no. 0), IEEE, Ashville, USA, 3: 2316-2320. https://doi.org/10.1109/ICSMC.2000.886462

Rout S, Sahoo KS, Patra SS, Sahoo B, and Puthal D (2021). Energy efficiency in software defined networking: A survey. SN

Computer Science, 2(4): 1-15. https://doi.org/10.1007/s42979-021-00659-9

Shallahuddin AA, Kadir MFA, Mohamed MA, Usop NSM, and Zakaria ZA (2020). An enhanced adaptive duty cycle scheme for optimum data transmission in wireless sensor network. In: Kim K and Kim HY (Eds.), Information science and applications: 33-40. Springer, Singapore, Singapore. https://doi.org/10.1007/978-981-15-1465-4_4

Sinha Y, Vashishth S, and Haribabu K (2017). Real time monitoring of packet loss in software defined networks. In the International Conference on Ubiquitous Communications and Network Computing, Springer, Bangalore, India: 154-164. https://doi.org/10.1007/978-3-319-73423-1_14

Tsitsiashvili GS and Losev AS (2008). Application of the Floyd algorithm to the asymptotic analysis of networks with unreliable ribs. Automation and Remote Control, 69(7): 1262-1265. https://doi.org/10.1134/S0005117908070175

Xia W, Wen Y, Foh CH, Niyato D, and Xie H (2014). A survey on software-defined networking. IEEE Communications Surveys and Tutorials, 17(1): 27-51. https://doi.org/10.1109/COMST.2014.2330903