Modified Dijkstra Shortest Path Algorithm for SD Networks

Original Scientific Paper

Haitham M. Abdelghany
Electronics and Communication Engineering Department, Faculty of Engineering, Mansoura University, El-Mansoura, Egypt
habdelghany@ outlook.com

Fayez W. Zaki
Electronics and Communication Engineering Department, Faculty of Engineering, Mansoura University, El-Mansoura, Egypt
fwzaki2017@gmail.com

Mohammed M. Ashour
Electronics and Communication Engineering Department, Faculty of Engineering, Mansoura University, El-Mansoura, Egypt
mohmoh2@yahoo.com

Abstract *– This paper uses a modified Dijkstra shortest path method for considering cumulative delays rather than bandwidth in software-defined networks. To implement the proposed method, an open-source Ryu controller is used, and a Mininet tool is used to emulate the topology. The proposed method is compared with the traditional Dijkstra's algorithm to demonstrate its performance. This comparison shows that the modified Dijkstra's algorithm provides higher performance of the different cumulative delays. Several experiments were conducted to evaluate the performance of the proposed method using three parameters (bandwidth, transfer rate and jitter). In addition, the cumulative distribution function is calculated using the parameters to show its distribution through the experiment period.*

Keywords*: Dijkstra shortest path, Software-Defined Networking, Ryu, Mininet, Jitter, cumulative distribution function.*

1. INTRODUCTION

Software-defined networking (SDN), considered the next generation of networking [1-3], aims to separate the control plane and the data plane. The control plane is moved to the central unit, called the controller, while the other switches act as forwarders [4]. Separating the control plane makes it easy to develop novel techniques that give such networks the flexibility to respond to network changes.

SDN technology allows network programmers to deploy the controller in a flexible way through programming languages such as Python and Java. For example, programmers can apply load-balancing techniques and intrusion prevention through programming [5].

In this paper, the widest Dijkstra shortest path method [6] is modified to forward load based on link delay. Python is used to implement the modified method and evaluate its performance by comparing it with the traditional Dijkstra's method. The emulation is carried out using the Mininet tool. The modified method outperforms the original one.

As reported in [7], there are some problems in deploying Dijkstra's method [8] and the altered Floyed–Warshall shortest path method in OpenFlow. The altered Dijkstra's method in [7] is not the same as the method proposed here. The proposed method is therefore compared with the traditional Dijkstra's method.

The remaining sections of this paper are organized as follows. Section 2 introduces SDN and the Mininet emulator. Section 3 discusses the related work. Section 4 presents the modified widest Dijkstra's method and its deployment. Section 5 reports the emulation results. Section 6 describes the analytical model of SDN. Finally, Section 7 presents the conclusion of this work.

2. SOFTWARE-DEFINED NETWORKING AND MININET

In SDN, forwarding decisions are taken by the central unit, consisting of controllers [9], while the other network devices are forwarders. Fig. 1 shows the main concept of SDN. Communication from the controllers to the network device is called the southbound interface. The

most used protocol is the OpenFlow protocol [10,11] that may have a single or many flow tables and group tables. As shown in Fig. 2, the OpenFlow protocol allows the controllers to add or remove entries in the table.

When data reach the switch, the OpenFlow switch searches for a record in the flow table. If there is no record, the data are returned to the controller according to the routing policy [12].

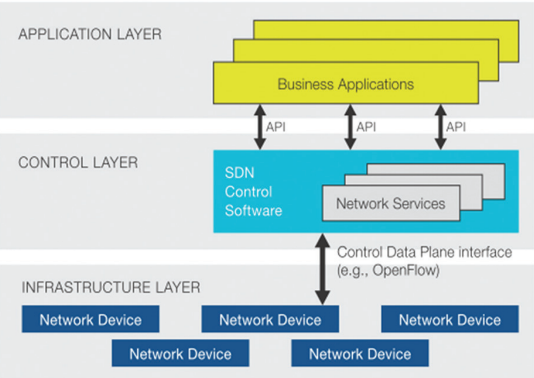

Fig. 1. The main concept of SDN [13]

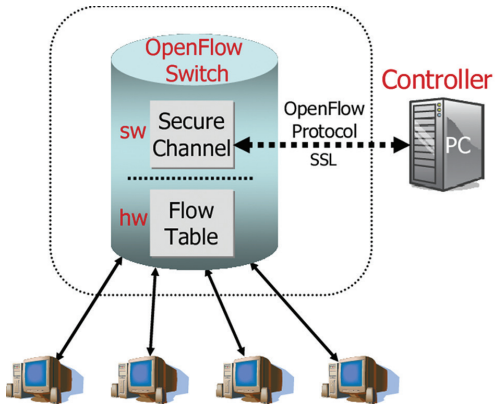

Fig. 2. The OpenFlow switch and the controller [14]

Mininet [6,15] is the network emulator that runs in Linux and is popular in SDN research. It is also widely used by researchers to emulate Open vSwitch and virtual hosts. As the complex topology depends on the specification of the server used, Mininet allows the researcher to create the topology using a Python script.

3. RELATED WORK

The Dutch computer researcher E. W. Dijkstra introduced the traditional Dijkstra algorithm in 1959. It has been used in many fields, such as mobile communication, computer networking, geographic information science and transportation. Dijkstra is a mathematical algorithm used to calculate the shortest path between two nodes in a system. The traditional algorithm was widely used in networking systems. Open Shortest Path First [16] mainly relies on Dijkstra's algorithm to calculate the best route from the source to a destination. The pseudocode for the traditional Dijkstra's algorithm is shown below.

Input: P, k **Output: dis[V], pre[V]**
1: for each v in P(V)
2: dis[v] ← ∞
3: pre[v] ← Ø
4: put distance at u node into Q set
5: while (Q!=Ø)
6: u← Min distance(Q)
7: for each v of u
8: if dis[v] > dis[u]+ew[u,v] then
9: dis[v]←dis[u]+ew[u,v]
10: pre[v]←dis[u]

In [15], J. R. Jiang and a group of researchers proposed an algorithm that extends the traditional Dijkstra's algorithm by adding predefined values of the node weights to prioritize the traffic [17]. Their extended Dijkstra's algorithm outperforms the traditional Dijkstra's algorithm. The weighted Dijkstra's algorithm was implemented using the Abilene topology [18]. This is because the extended Dijkstra's algorithm calculates the distance based on the weights of the nodes in the network and takes node weight into account, whereas the traditional Dijkstra's algorithms do not consider the edge weight or load balancing [19].

Laberio [20] and Lobus [21] proposed load-balancing algorithms for SDN that use the path and link employment to optimize the network throughput. The other algorithm is the service-based load-balancing algorithm [22]. In service-based algorithms, the flow is related to fixed services. Using the service-based load-balancing algorithm is specified to network devices as switches and routers with particular services to increase throughput of the network. Both methods consider the route as the important way of achieving the optimal throughput.

4. IMPLEMENTATION OF THE MODIFIED DIJKSTRA'S ALGORITHM

4.1 MODIFIED EXTENDED DIJKSTRA'S ALGORITHM

Given a single source S and weighted graph G = (V,E), the pseudocode is as shown below.

Input: G=(V, E), ed, nd, h **Output: delay[|V|], p[|V|]**
1: delay [h]←0; delay [t]←∞, for each t≠h, t€V
2: insert t with key delay [t] into the priority queue Q, for each t€V
3: while (Q≠Ø)
4: t←Extract-Min(Q)
5: for each v adjacent to t
6: if delay [v] > delay [t]+ed[t,v]+nd[t] then
7: delay [v]←delay [t]+ed[t,v]+nd[t]
8: p[v]←delay [t]

The difference between the modified extended Dijkstra and the extended Dijkstra is that in the former the route is selected based on distance instead of the delay. The importance of the proposed method is evident when the delays are different on equal bandwidth links. Thus, the selection is made based on delay. Table 1 shows a brief comparison of the traditional Dijkstra and modified Dijkstra 3.

Table 1. Comparison of traditional Dijkstra and modified Dijkstra 3

Algorithm	Traditional Dijkstra	Modified Dijkstra 3
Criteria for Path Selection	Distance	Delay
Initiation of Destination Node	+ve Infinity	-ve Infinity
Load Balancing	No	Yes
Weighted Nodes	No	Uses weights for nodes

5. ANALYTICAL MODEL

As shown in Fig. 3, in the OpenFlow SDN model, the controller is connected to a number of switches. It is assumed that packet arrival follows a Poisson distribution with an arrival rate λ_i. Packets that do not have matching entries are probably sent to the controller. Thus, the arrival rate is λ_i.*p, λ_i.*(1-p), and the processing time is exponential $1/\mu_i$ for the switches. The average service time for the controller is equal to $1/\mu_c$ where μ_i is the processing rate of the switches and μ_c is the processing time of the controller.

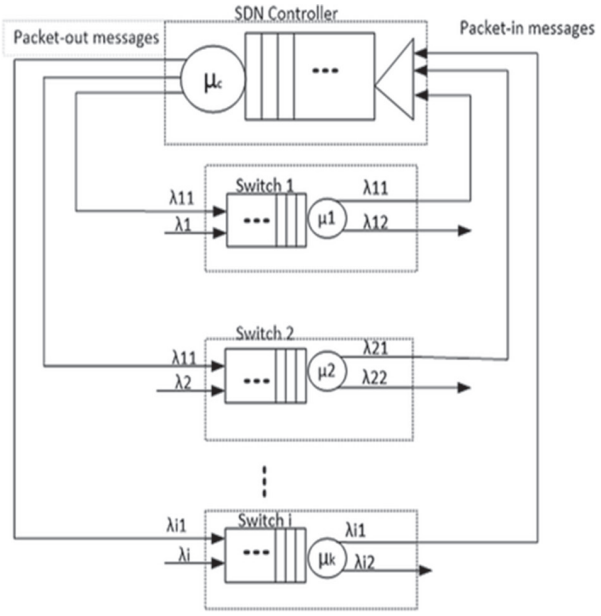

Fig. 3. OpenFlow SDN analytical model [23]

5.1. SWITCH PERFORMANCE

The flow tables are not always the same and can be changed based on link delay, and the processing time is assumed to follow an exponential distribution.

The OpenFlow switches and the controller can be modelled with a M/H_2/1 queue [14]. This means that the arrival of packets λ_i. and the serving rate are hyper-exponential with two-phases.

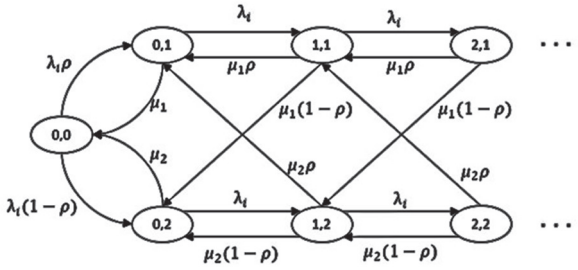

Fig. 4. State diagram of an M/H_2/1 queue [24]

In the state diagram in Fig. 4, p is the probability that data are processed at rate μ_1, and 1−p is the probability of receiving a service rate of μ_2.The stationary probability π(i) is a vector and can be represented as

$$\pi^i = (\pi_0{}^i, \pi_1{}^i, \pi_2{}^i, \ldots\ldots, \pi_k{}^i). \quad (1)$$

$$\rho = {}^{\lambda}/_{\mu} < 1. \quad (2)$$

$$\pi_0 = 1\text{-}\rho. \quad (3)$$

$$\pi_k = (1\text{-}p)\,\rho^k. \quad (4)$$

$\pi_k(1)$ is the *k* packet probability in the i_{th} switch.

The mean value of packets in the queueing system is

$$N_i = \sum_{k=o}^{\infty} k\pi_k^i \quad (5)$$

$$N_i = \sum_{k=o}^{\infty} k(1-\rho)\rho^k \quad (6)$$

For *k* = 0, the product is equal to 0, and so the summation can begin from *k* = 1:

$$N_i = (1-\rho)\sum_{k=1}^{\infty} k\rho^k = (1-\rho)\rho\sum_{k=1}^{\infty} k\rho^{k-1} \quad (7)$$

$$\text{With } k\rho^{k-1} = \frac{d\rho^k}{d\rho},$$

$$N_i = (1-\rho)\rho\sum_{k=1}^{\infty}\frac{d\rho^k}{d\rho}\rho^k \quad (8)$$

$$N_i = (1-\rho)\rho\frac{d\rho}{d\rho}\sum_{k=1}^{\infty}\rho^k \quad (9)$$

So,

$$\sum_{k=1}^{\infty}\rho^k = \sum_{k=0}^{\infty}\rho^k - 1 = \frac{1}{1-\rho} - 1 = \frac{\rho}{1-\rho} \quad (10)$$

$$N_i = \frac{\rho}{1-\rho} \text{ while } \rho < 1 \text{ and } \rho = {}^{\lambda}/_{\mu}. \quad (11)$$

From Little's formula, the mean processing delay in the i_{th} switch may be expressed as

$$W_{si} = \frac{1}{\lambda}\mu_i = \frac{1}{\mu-\lambda} \quad (12)$$

The average processing delay of packets by all switches can also be given as

$$W_s = \sum_{i=1}^{n}\frac{\lambda_i}{\sum_{i=1}^{n}\lambda_i} W_{si} \quad (13)$$

6. EMULATION

Mininet was used to implement the experiments of the virtual environment. The experiments were hosted on a custom-built server with Intel Core i5 CPU at 2.5 GHz, 8 GB RAM with a solid-state drive for storage and switching. The Ubuntu Server 20.04 LTS distribution was installed on the machine. Then the Open-V-Switch software was installed. One controller, three switches and three hosts were used, as shown in Fig. 5. The emulation parameters are shown in Table 2. Iperf was used to test the bandwidth, transfer rate and jitter. In the experiments, host 3 acted as a server, while hosts 1 and 2 acted as clients.

Table 2. Simulation parameters

Delay on Edges	1 ms
Delay Between Switches	1-3 ms
Number of Hosts	3
Number of Switches	3
Controller	Ryu 4.34
Testing Tool	Iperf

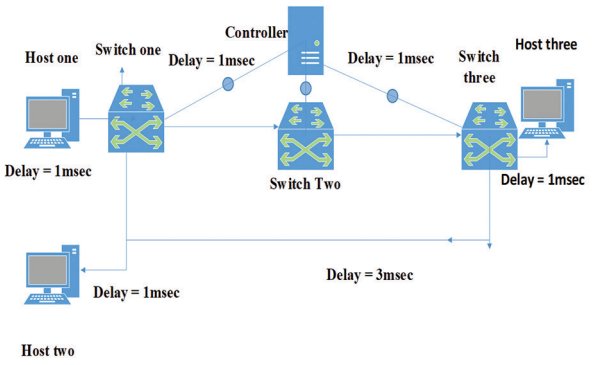

Fig. 5. The topology used in the experiment

6.1 EMULATION RESULTS

Fig. 5 shows that, when host 1 sent data to host 3, the link delay was 4 ms plus the processing time of every switch. Host 1 has two routes. The first route is the direct link from switch 1 to switch 3, which has a link delay of 4 ms plus the processing time of every switch. The other route, which is through switch 2, has link delay of 4 ms plus the processing time of every switch. The proposed algorithm load-balances the traffic between the two routes, whereas the traditional Dijkstra's algorithm does not. Iperf was used to measure the bandwidth, transfer rate and jitter. The results in Figs. 6, 7 and 8 show that the modified extended Dijkstra outperforms the extended Dijkstra when two hosts send data at the same time to the same host (host 3). In the preceding experiments, the cumulative distribution function (CDF) of the parameters was measured to describe the distribution of the parameters across the whole time.

As shown in Fig. 6, the CDF for the bandwidth from host 1 to host 3 was between 46 Mbps and 56 Mbps when the proposed method was used. In the traditional Dijkstra, the CDF for the bandwidth from host 1 to host 3 varied in a wide range from 20 Mbps to 43 Mbps.

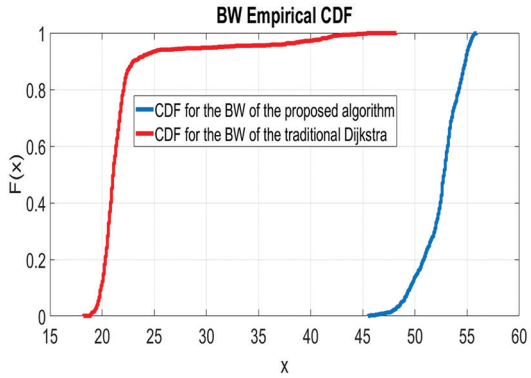

Fig. 6. The CDF for the bandwidth in Mbps for host 1 when the proposed method is applied vs traditional Dijkstra

As shown in Fig. 7, the CDF for the transfer rate lies between 5.5 Mbytes and 6.7 Mbytes using the proposed method. The CDF for the transfer rate varies widely from 2.3 Mbytes to 5 Mbytes using the traditional Dijkstra's algorithm.

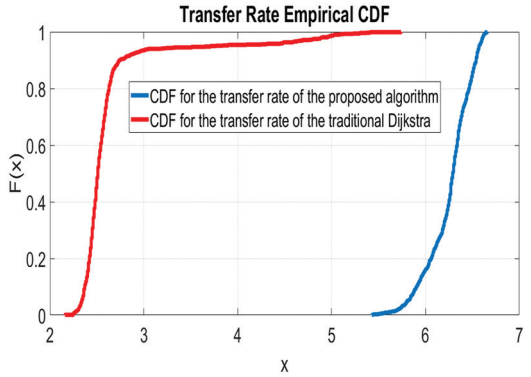

Fig. 7. The CDF for the transfer rate in MBytes for host 1 when the proposed method is applied vs traditional Dijkstra

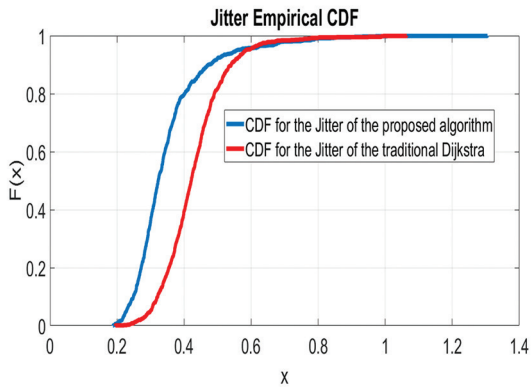

Fig. 8. The CDF for the jitter in ms for host 1 when the proposed method is applied vs traditional Dijkstra

Finally, as shown in Fig. 8, the CDF for the jitter is between 0.2 ms and 0.6 ms using the proposed method. The CDF for the jitter varies from 0.3 ms to 0.8 ms when traditional Dijkstra is used. Furthermore, Table 3 summarizes the average values of bandwidth, transfer rate and jitter for host 1 when traditional Dijkstra is used versus the modified Dijkstra.

Table 3. Average parameter values

Parameter	Traditional Dijkstra	Modified Dijkstra 3
Bandwidth (Mbps)	22.1213	52.4679
Transfer Rate (MB)	2.6367	6.2553
Jitter (ms)	0.4303	0.3501

7. CONCLUSION

This study proposes and implements a modified extended Dijkstra's algorithm. The main test showed that the proposed method outperforms the traditional Dijkstra's algorithm. In the previous experiments, the CDF of the selected parameters was used to evaluate the performance of the proposed method against the traditional Dijkstra's algorithm. It is important to mention that every experiment was repeated 10 times and held for about one hundred seconds. The proposed method uses a Ryu controller implemented in Python, and the Mininet tool was used to emulate the network topology. In the future, a more complex design will be proposed to further test the performance of the proposed method.

8. REFERENCES

[1] S. Rowshanrad, S. Namvarasl, V. Abdi, M. Hajizadeh, M. Keshtgary, "A survey on SDN, the future of networking", Journal of Advanced Computer Science & Technology, Vol. 3, No. 2, 2014, pp.232-248.

[2] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, T. Gayraud, "Soft- ware-defined networking: Challenges and research opportunities for future internet", Computer Networks, Vol. 75, No. 24, 2014, pp.453-471.

[3] J. Pan, S. Paul, R. Jain, "A survey of the research on future internet architectures", IEEE Communications Magazine, Vol. 49, No. 7, 2011, pp. 26-36.

[4] A. Lara, A. Kolasani, B. Ramamurthy, "Network innovation using open- flow: A survey", IEEE Communications Surveys & Tutorials, Vol. 16, No. 1, 2014, pp. 493-512.

[5] S. Ahmad, A. H. Mir, "Scalability, consistency, reliability and security in SDN controllers: A survey of diverse SDN controllers", Journal of Network and Systems Management, Vol. 29, No. 1, 2021, pp. 1-59.

[6] E. Dijkstra, "A note on two problemsecin connexion with graphs", Numerische Mathematik, Vol. 1, No.1, 1959, pp. 269-271.

[7] A. Furculita, M. Ulinic, A. Rus, V. Dobrota, "Implementation issues for Modified Dijkstra's and Floyd-Warshall algorithmsecin OpenFlow," Proceedings of the RoEduNet International Conference 12th Edition: Networking in Education and Research, 2013, pp. 141-146.

[8] A. Rus, V. Dobrota, A. Vedinas, G. Boanea, M. Barabas, "Modified Dijkstra's algorithm with cross-layer QoS", ACTA TECHNICA NAPOCENSIS, Electronics and Telecommunications, Vol. 51, No. 3, 2010, pp. 75-80.

[9] Open Network Foundation (ONF) Website (SDN whitepaper), https://www.opennetworking.org/sdn-resources/sdn-definition (accessed: 2021)

[10] A. Greenberg et al. "A clean slate 4D approach to network control and management", Computer Communication Review, Vol. 35, No. 5, 2005, pp. 41-54.

[11] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, "Taking control of the enterprise", Computer Communication Review, Vol. 37, No. 4, 2007, pp. 1-12.

[12] Open Networking Foundation, "OpenFlow Switch Specification version 1.4.0", October 14, 2013.

[13] Floodlight OpenFlow Controller—Project Floodlight, Big switch network, http://www.projectfloodlight.org/floodlight (accessed: 2019)

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, J. Turner, S. Shenker. "Openflow Enabling innovation in campus networks", Computer Communication Review, Vol. 38, No. 2, 2008, pp 69-74.

[15] J. Jiang, H. Huang, J. Liao, S. Chen, "Extending Dijkstra's Shortest Path Algorithm for Software Defined Networking", Proceedings of the 16th Asia-Pacific Network Operations and Management Symposium, Hsinchu, Taiwan, 2014, pp. 1-4.

[16] J. Moy, "OSPF: Anatomy of an Internet Routing Protocol", Addison-Wesley, 2000.

[17] P. Tantisarkhornkhet, W.Werapun, B. Paillassa "SDN experimental on the PSU network", Proceedings of the International Symposium on Intelligent Signal Processing and Communication Systems, Phuket, Thailand, 2016, pp. 1-6.

[18] Abilene Network, https://en.wikipedia.org/wiki/Abilene_Network (accessed: 2021)

[19] W. Yahya, A. Basuki, J. R. Jiang, "The extended Dijkstra's-based load balancing for openflow network", International Journal of Electrical and Computer Engineering, Vol. 5, No. 2, 2015, pp. 289-296.

[20] H. Long, Y. Shen, M. Guo, F. Tang. "LABERIO: Dynamic load-balanced routing in OpenFlow-enabled networks", Proceedings of the IEEE 27th International Conference on Advanced Information Networking and Applications, Barcelona, Spain, 2013, pp. 290-297.

[21] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, R. Johari, "Plug-n-Serve: Load-balancing web traffic using Open Flow", Demo at ACM SIGCOMM, August 2009.

[22] M. Koerner, O. Kao, "Multiple service load-balancing with OpenFlow", Proceedings of the 13th IEEE International Conference on High Performance Switching and Routing, Belgrade, Serbia, 2012, pp. 210-214.

[23] S. Muhizi, G. Shamshin, A. Muthanna, R. Kirichek, A. Vladyko, A. Koucheryavy, "Analysis and Performance Evaluation of SDN Queue Model", International Federation for Information Processing, 2017, pp. 26-37.

[24] Z. Shang, K. Wolter, "Delay evaluation of openflow network based on queueing model", http://arxiv.org/abs/1608.06491 (accessed: 2021)