Equal-Cost Multipath Routing in Data Center Network Based on Software Defined Network

Fiqih Rhamdani*, Novian Anggis Suwastika†, Muhammad Arief Nugroho‡
School of Computing
Telkom University, Bandung, Indonesia
konzaho@student.telkomuniversity.ac.id*, anggis@telkomuniversity.ac.id†, arif.nugroho@telkomuniversity.ac.id‡

***Abstract*—In Data Center facilities, network performance is crucial. The system performance requirements are usually high throughput and packet loss sensitive. In addition, the data center also requires redundant links, in order to avoid the occurrence of failure in the network. Applying topologies such as as fat-tree can solve the redundant link needs problem. However, the use of redundant links that are only used as backups are considered ineffective. A routing mechanism that can exploit the potential of existing links sufficiently is required. Equal-cost multipath routing (ECMP) is a routing scheme that can be implemented to solve the problem. With the implementation of software defined network (SDN) on the network, the construction of this routing scheme will be more accessible. The ECMP scheme utilizes modified Dijkstra's algorithm to search for the shortest path, and uses the modulo-n hashing method in the selection of the delivery path. This paper implements ECMP scheme on fat-tree topology with SDN-based network. The results obtained in this study prove that the network implementing ECMP can provide a throughput difference of 60.14Mbps when compared with static routing in a 20Mbps bandwidth traffic test scenario. Also the ECMP network can provide 15.72% less packet loss compared to static routing networks.**

***Keywords*—Equal-Cost Multipath, Software Defined Network, Dijkstra**

I. Introduction

The development of the scale and complexity of computer networks, as well as the increasing need for network services, create increasingly high network requirements and makes the network control complicated and inflexible. This also applies to data centers. In the data center, network performance is crucial where there are many systems that require high throughput, packet loss sensitive and also data center networks must support fail-over mechanism [1].

One approach to support performance needed by the data center is to apply a fat-tree topology [2]. Where in this fat-tree topology there is a redundant link that serves as a fail-over mechanism. But applying the topology alone is not enough to meet the bandwidth requirements in the data center. The redundant link utility that is only used as a backup will make the use of network resources less efficient [3]. A routing mechanism is required to utilize resources sufficiently.

The routing technique with multipath is considered to provide better performance compared to single path routing [4]. By utilizing the shape of a fat-tree topology [5] that has the same redundant link weight, the equal cost multipath (ECMP) routing scheme can be implemented. ECMP is a method of multipath routing that looks for paths that have the same weight. The weights used in path search are the number of hops performed to send packets from one host to another. If there are multiple candidate paths, a scheme is needed to determine which path will be used to deliver the packet.

There are several ways to determine paths, among others, Random choose, Round Robin, and Modulo-N Hash. But there are disadvantages in the Random and Round Robin method, i.e. the path selection object on the Random or Round Robin method is the package to be sent itself. This can result in packets that have the same purpose going through different paths that may impact packets arriving out of sequence, or so-called packet re-ordering. This will certainly adversely affect the performance of the TCP protocol [6]. On that basis, the method chosen for use in this paper is the Modulo-N Hash method. There are problems arising from the process of implementing the scheme above, that is, network devices that lack the flexibility and scattered configuration makes the implementation and maintenance process difficult [7]. A solution to the problem is to perform centralized network control.

II. Dijkstra Algorithm

Dijkstra's algorithm is the shortest path search algorithm in a graph. The algorithm was proposed by Dutch computer scientist, Edsger W. Dijkstra, in 1956 and became popular three years later [8]. Dijkstra's algorithm is a greedy algorithm that solves the problem of the shortest distance from a directed graph with non-negative side weights [9].

In a simple application example, Dijkstra's algorithm is used to find a path between two points. For example, if points are represented as cities and paths are represented as roads, Dijkstra's algorithm can be used to find the shortest path between cities. In computer networks, Dijkstra's algorithm has been widely applied as a routing protocol. Examples of routing protocols that use Dijkstra's algorithm are IS-IS (Intermediate System to Intermediate System) and OSPF (Open Shortest Path First).

III. Proposed Method

A. Overall System

The system is built in 2 environments. The first environment is a virtual environment built using VirtualBox applications. This virtual environment will load the topology required in this

ISBN: 978-1-5386-4571-0 (c) 2018 IEEE

research. The topology in use is a fat-tree topology [5]. The second environment is the main computer used in this research, this computer aims to run the SDN controller POX, where the entire traffic management process and data processing in the virtual network is governed by this controller. Fig. 1. shows the illustration of the system.

B. Topology Design

The Topologi is designed on top of a virtual environment built using VirtualBox applications. Fig. 2 shows an overview of the topology used in this study.

The main reason to use this topology is, this topology is one of the topologies that is widely used in data centers [10]. The selection of data center topology is based on the background of the problem raised in this research, that is there is research which proves that Fat-tree topology is very suitable for environments that have heavy and diverse traffic [11]. In addition, the Fat-tree topology also has multiple redundant links that have the same hop length, making it very suitable to apply load balancing equal cost multipath routing scheme. Here are the specifications used in the topology:

- 16 Hosts
- 8 Edge Switch
- 8 Aggregation Switches
- 4 Core Switches
- 100Mb Bandwidth for each link

This topology is built with the Python programming language and is run by Mininet applications that reside in virtual environments. IP and MAC host on the network has been determined to avoid the ARP mapping initialization process, Table I describes the host configuration used in this study.

C. ECMP Routing

Fig. 4 shows how the ECMP scheme flow works [12]. The process begins with a packet coming on the first switch of the sending host. The switch then checks its flow-table, when the flow-table for the incoming packet exists, then the packet is immediately forwarded to the destination switch. But if not, then the packet will be encapsulated and then

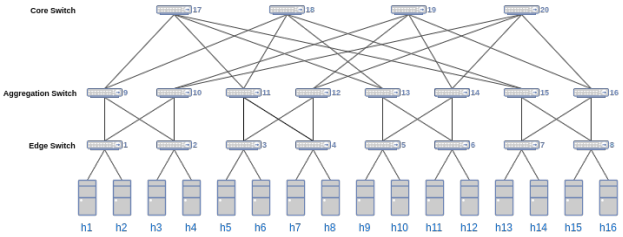

Fig. 2. Topology Design

Algorithm 1 Pseudocode algoritma Dijkstra

```
1:  function DIJKSTRA(Graph, source)
2:      create vertex set Q
3:      for each vertex v in Graph:
4:          dist[v] ← INFINITY
5:          prev[v] ← UNDEFINED
6:          add v to Q
7:      dist[source] ← 0
8:
9:      while Q is not empty do:
10:             u ← in Q with min dist[u]
11:             remove u from Q
12:             for each neighbour v of u:
13:                 alt ← dist[u] + length(u, v)
14:                 if alt < dist[v]
15:                     dist[v] ← alt
16:                     prev[v] ← u
17:     end while
18:     return dist[], prev[]
19: end function
```

Fig. 3. Dijkstra Algorithm Pseudocode

TABLE I. HOST CONFIGURATION

Host	IP Address	MAC Address
H1	10.0.0.2/8	00:00:00:00:00:01
H2	10.0.0.3/8	00:00:00:00:00:02
H3	10.0.1.2/8	00:00:00:00:00:03
H4	10.0.1.3/8	00:00:00:00:00:04
H5	10.1.0.2/8	00:00:00:00:00:05
H6	10.1.0.3/8	00:00:00:00:00:06
H7	10.1.1.2/8	00:00:00:00:00:07
H8	10.1.1.3/8	00:00:00:00:00:08
H9	10.2.0.2/8	00:00:00:00:00:09
H10	10.2.0.3/8	00:00:00:00:00:10
H11	10.2.1.2/8	00:00:00:00:00:11
H12	10.2.1.3/8	00:00:00:00:00:12
H13	10.3.0.2/8	00:00:00:00:00:13
H14	10.3.0.3/8	00:00:00:00:00:14
H15	10.3.1.2/8	00:00:00:00:00:15
H16	10.3.1.3/8	00:00:00:00:00:16

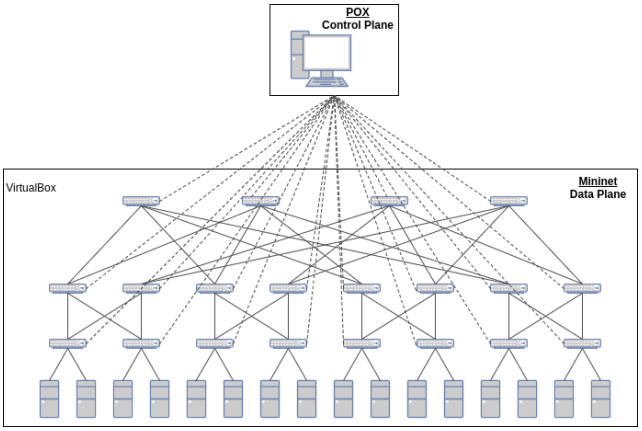

Fig. 1. Overall System

sent to the controller. packets that have been encapsulated are called PACKET_IN. PACKET_IN has additional headers namely SwitchID, ingressPort, etherType, etherSrc, etherDst, IPv4src, IPv4dst, and TCP / UDP port. PACKET_IN is what will be accepted by the controller for processing.

The controller will then check the incoming PACKET_IN. Checking is done to see if the destination MAC address is hosted on the MAC table. If not, then the controller will send PACKET_IN to all its sub-switches to find out the existence of the host in question. However, if the destination host MAC address is already in the MAC Table, then the path search process can be done.

The path finding process applies Dijkstra algorithm. But as already known, Dijkstra's algorithm provides only a result of a shortest path or path that has the smallest weights, the

Dijkstra pseudo code can be shown at Fig. 3. This certainly does not support the ECMP scheme to be implemented in this study. Therefore, in order to use the Dijkstra algorithm as a path search algorithm, the algorithm needs to be modified in order to provide multiple paths with identical least weights. Details on Dijkstra's modification algorithm will be discussed in subsequent chapters.

After obtaining the shortest paths, the next process is to do the PACKET_IN hashing, the result of the hash will then be processed using modulus N operation, where N is the number of paths selected by Dijkstra's previously executed algorithm. The result of the modulus operation will indicate which path will be chosen to send the packet. The next step is the controller creating a routing rule which is then forwarded to all switches associated with the routing and forwarding PACKET_IN back to the switch, which then the switch will forward the packet to the destination host.

D. Dijkstra Algorithm Modification

In the Dijkstra algorithm the path chosen as the best path is only one path. It is certainly not enough to apply the ECMP mechanism which requires not just one path. Selection of multiple paths or practically multipath Dijkstra actually has the same concept as Dijkstra algorithm in general. Fig. 5 is a pseudocode modification Dijkstra algorithm.

The algorithm is largely unchanged, but the difference is "alt $<$ dist [v]" in line 16, which is the condition when the shorter path is found then action empty "previous [v]" and add vertex u is needed. "alt == dist [v]" in line 21 explains what happens when it finds a path that has the same weight as the previous path.

E. Hashing Method

Once the shortest path is selected, the data is not transmitted automatically across these paths, but a mechanism is needed to determine which path to send the packet. The transmitting path selection mechanism used in this research is the modulus technique from the 5-tuple hashing result. 5-tuples refers to a set consisting of five different values consisting TCP/IP connections. The value consists of the sender's IP address, sender port, destination IP address, destination port, and the protocol used. Fig. 6 provides a hashing procedure pseudocode used in this research.

Fig. 6 explains the hashing process is in this research. This function begins by making 5 empty arrays and then receiving incoming packets. The variables contain 5-tuple information which will be included in the array with the requirement that the packet has an IPv4 address. If the packet is indicated using TCP or UDP protocol then the source port and destination port information will be inserted into the array. Once the information is complete, the CRC32 function is called to do hashing. The output result of this function is a 32-bit unsigned integer which is proceeded to the modulus <n> process <n> of the hash result, where <n> is the number of paths available. the result of the modulus indicates which path will be used to send the packet.

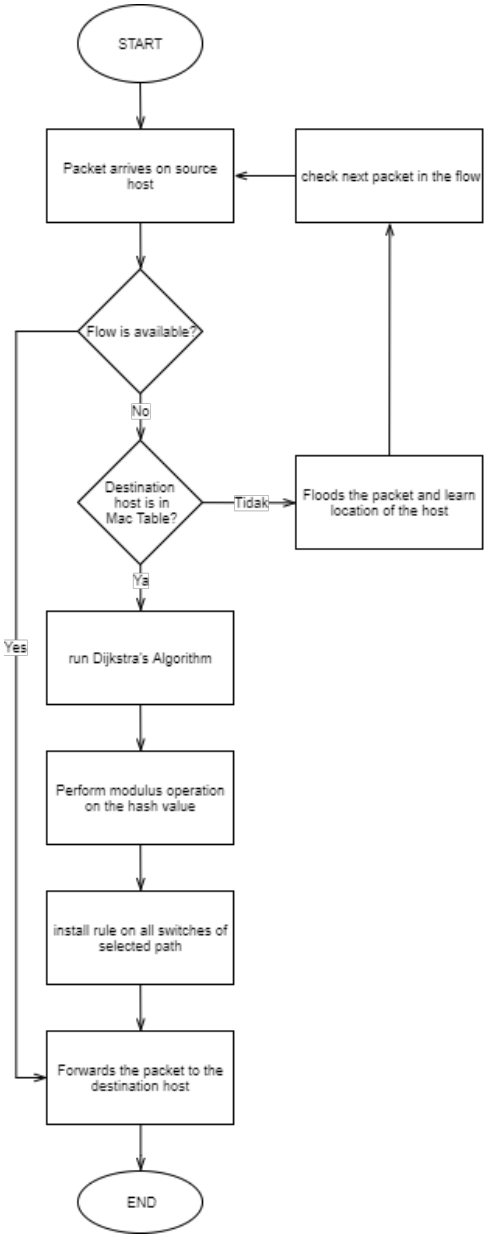

Fig. 4. ECMP System Flowchart

IV. Experimental Test and Result

A. Send TCP Data Test

This test generates data in the form of throughput comparison and delivery time of the ECMP network with static routing network can be seen in Table II. Testing is performed 3 times with 3 different scenarios, i.e. background traffic is set to 0Mbps, 20Mbps, and 30Mbps with packet size equal to 500MB.

Fig. 7 shows an increase in the required transmit time for 500 MB of data on a static routing network. There is a 76.52 seconds difference between scenario 0 Mbps and 20 Mbps background traffic and 693.62 seconds difference between scenario 0 Mbps and 30 Mbps. This comparison is shown

also in the throughput resulted by the network, the higher the amount of background traffic, the lower the performance of the throughput provided by the network using static routing. These numbers are shown in Fig. 8.

The performance of ECMP networks is moderately high compared to the performance of static routing networks. In the 20 Mbps background traffic scenario, there is a difference of 81.2 seconds in data transmission time and a difference of 58.14 Mbps in the resulting throughput. An even more dramatic comparison is shown in the 30 Mbps background traffic scenario, the delivery time difference is up to 774.98 seconds and the resulting throughput difference is 88.79Mbps.

Decreases that occur in static routing network is caused by the path utility on the network. Static routing networks use only one transmission path to transmit all data, resulting in network congestion that can degrade performance. In the ECMP network there are several paths used to transmit packets, so that if there is a path that has heavy traffic, the network can use other lines to transmit data resulting in network congestion avoidance.

TABLE II. TCP DATA TRANSMISSION TEST RESULT

Traffic	ECMP		Host	Static Routing	
	Time	Throughput		Time	Throughput
0 Mbps	47.94s	87.48 Mbps	H1-H16	49.54s	84.74 Mbps
20 Mbps	44.86s	93.42 Mbps		126.06s	33.28 Mbps
30 Mbps	44.7s	93.9 Mbps		819.68s	5.11 Mbps

B. Packet Loss and Jitter Test

The collection of packet loss and jitter data in this test is done simultaneously can be seen in Table III, by sending UDP packets for 10 seconds with maximum bandwidth capacity

Algorithm 2 *Pseudocode* algoritma Multipath Dijkstra

```
function MULTIPATHDIJKSTRA(Graph, source)
    for each vertex v in Graph:
        dist[v] ← INFINITY
        visited[v] ← FALSE
        previous[v] ← UNDEFINED
    end for
    dist[source] ← 0
    insert source into Q

    while Q is not empty do:
            u ← vertex in Q with smallest in dist[] and not yet visited;
            remove u from Q
            visited[u] ← true
            for each neighbour v of u:
                alt ← dist[u] + dist_between(u, v)
                if alt < dist[v]:
                    dist[v] ← alt
                    reset the previous[v]
                    add u into previous[v]
                end if
                else if alt == dist[v]:
                    add u into previous[v]
                if !visited[v] :
                    insert v into Q
                if
            end for

    end while
    return dist;
end function
```

Fig. 5. Multipath Dijkstra Pseudocode

Algorithm 3 *Pseudocode* fungsi *hashing*

```
function ECMP_HASH(packet)
    hash_input ← [0] * 5
    if ipv4 is in packet :
        ip ← packet
        hash_input[0] ← unsigned ip[srcIP]
        hash_input[1] ← unsigned ip[dstIP]
        hash_input[2] ← unsigned ip[protocol]
        if TCP is in ip or UDP is in ip:
            l4 ← ip
            hash_input[3] ← l4[srcPort]
            hash_input[4] ← l4[dstPort]
            return crc32(hash_input)
    return 0
end function
```

Fig. 6. Hashing Function Pseudocode

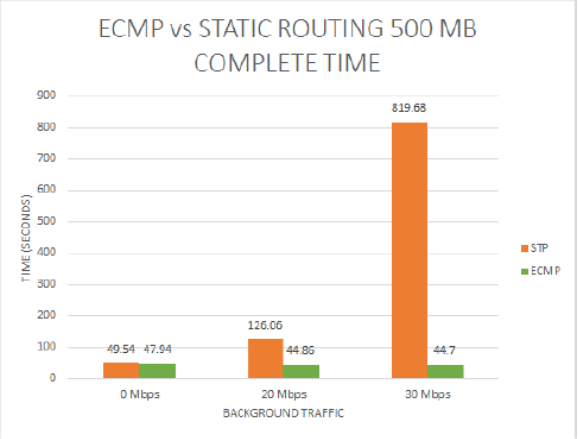

Fig. 7. TCP Packet Transmission Time Graphic

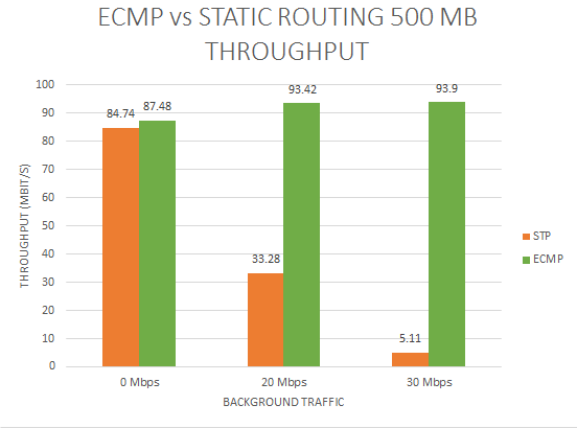

Fig. 8. TCP Packet Transmission Throughput Graphic

100 Mbps. Testing is done for 5 times on each scenario. The resulting value of each scenario is then averaged.

It can be seen in the Fig. 9 on both networks there is an increase in the percentage of packet loss along with the increase of background traffic. However, it can be seen that packet loss on ECMP networks is smaller when compared to static routing networks in any scenario. In the test scenario with no background traffic, packet loss difference between ECMP network and static routing network is 13.288% and with background traffic of 17.5 Mbps, static routing network packet loss increased 13.22% and ECMP network packet loss

TABLE III. PACKET LOSS AND JITTER TES RESULT

Traffic	ECMP		Host	Static Routing	
	Packet Loss	Jitter		Packet Loss	Jitter
0 Mbps	0.32%	1.91ms	H1-H16	13.61%	2.66ms
17.5 Mbps	11.11%	0.60ms		26.83%	3.50ms
22.5 Mbps	20.46%	0.96ms		36.63%	3.95ms

increased 10.786%. But despite an increase in packet loss on both networks, the overall results show that packet loss on ECMP networks is smaller when compared to static routing networks.

As seen in Fig. 10, the jitter difference between the two networks in the test scenario with no background traffic is 0.748ms. 17.5 Mbps background traffic increases jitter by 0.842ms on static routing network but decreases jitter on ECMP network by 1.307ms. With the increase of background traffic to 22.5 Mbps, static routing network jitter increases to 0.457ms and ECMP network jitter also increases, but not as steep as static routing, to 0.361ms.

Based on Fig. 10, the jitter value difference between the two networks in the test scenario without background traffic is 0.748ms. The use of background traffic with a throughput of 17.5 Mbps increases jitter by 0.842ms in static routing networks but causes a decrease in jitter on an ECMP network with a value of 1,307 ms. With increased the background traffic to 22.5 Mbps, it make jitter increase in static routing network is worth 0.457 ms and also increase in ECMP network with value 0.361 ms.

V. CONCLUSION

In this paper, we present the results of performance comparisons between ECMP networks and static routing networks, in which ECMP networks can provide better performance results with 15.72% fewer lost packets in each test scenario, as well as providing smaller delay and jitter when compared with static routing network. For further research, the tests used different metric routing as the path selector to analyze the performance of the ECMP.

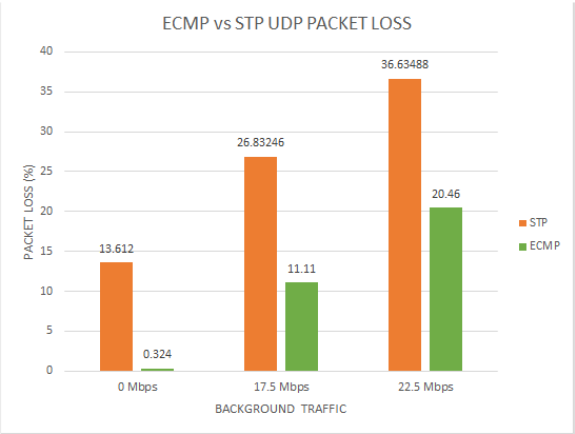

Fig. 9. Packet Loss Test Result Graphic

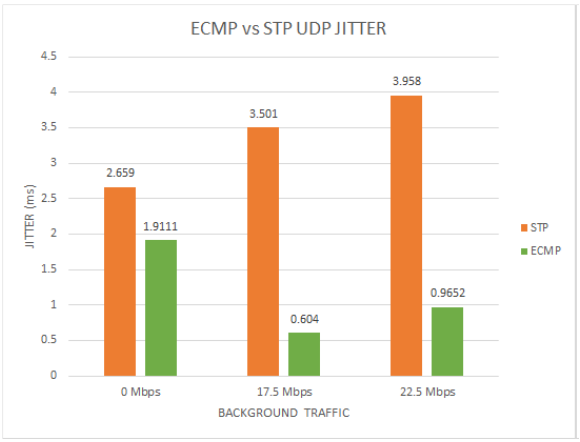

Fig. 10. Jitter Test Result Graphic

ACKNOWLEDGMENT

The author would like to thank Telkom University as sponsor for this research, and Radityo Putro Wibisono as a correctors.

REFERENCES

[1] H. Zhang, X. Guo, J. Yan, B. Liu, and Q. Shuai, "Sdn-based ecmp algorithm for data center networks," in *Computing, Communications and IT Applications Conference (ComComAp), 2014 IEEE*. IEEE, 2014, pp. 13–18.

[2] A. Ojo, N.-W. Ma, and I. Woungang, "Modified floyd-warshall algorithm for equal cost multipath in software-defined data center," in *Communication Workshop (ICCW), 2015 IEEE International Conference on*. IEEE, 2015, pp. 346–351.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[4] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE transactions on Computers*, vol. 100, no. 10, pp. 892–901, 1985.

[5] E. Jo, D. Pan, J. Liu, and L. Butler, "A simulation and emulation study of sdn-based multipath routing for fat-tree data center networks," in *Proceedings of the 2014 Winter Simulation Conference*. IEEE Press, 2014, pp. 3072–3083.

[6] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[7] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[8] S. Kaur, J. Singh, and N. S. Ghumman, "Network programmability using pox controller," in *ICCCS International Conference on Communication, Computing & Systems, IEEE*, no. s 134, 2014, p. 138.

[9] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[10] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 267–280.

[11] C. E. Hopps and D. Thaler, "Multipath issues in unicast and multicast next-hop selection," 2000.

[12] B. Heller *et al.*, "Openflow switch specification," *Version 0.8*, vol. 9, 2009.