JURNAL INFOTEL
Informatics - Telecommunication - Electronics

Website Jurnal : http://ejournal.st3telkom.ac.id/index.php/infotel
ISSN : 2085-3688; e-ISSN : 2460-0997

Improved Load Balancing on Software Defined Network-based Equal Cost Multipath Routing in Data Center Network

Ramadhika Dewanto[1*], Rendy Munadi[2], Ridha Muldina Negara[3]
[1,2,3]Universitas Telkom
[1,2,3]Jl. Telekomunikasi No. 1 Terusan Buah Batu, Bandung 40257, Indonesia
*Corresponding email: ramadhikadewanto@gmail.com

Received 15 July 2018, Revised 02 August 2018, Accepted 08 August 2018

Abstract — Equal Cost Multipath Routing (ECMP) is a routing application where all available paths between two nodes are utilized by statically mapping each path to possible traffics between the source and destination hosts in a network. This configuration can lead to congestion if there are two or more traffics being transmitted into paths with overlapping links, despite the availability of less busy paths. Software Defined Networking (SDN) has the ability to increase the dynamicity of ECMP by allowing the controller to monitor available bandwidths of all links in the network in real-time. The measured bandwidth is then implemented as the basis of the calculation to determine which path traffic will take. In this research, an SDN-based ECMP application that can prevent network congestion was made by measuring the available bandwidth of each available paths beforehand, thus making different traffics transmitted on non-overlapped paths as much as possible. The proposed scheme increased the throughput by 14.21% and decreased the delay by 99% in comparison to standard ECMP when congestion occurs and has 75.2% lower load standard deviation in comparison to round-robin load balancer.

Keywords – ECMP, SDN, Load Balancing

I. INTRODUCTION

With the rapid development of internet and its applications such as big data, cloud computing, and other large scale network services, enormous data centers which often take the form of multi-rooted switches or routers were built [1]. In order to efficiently use the resources of the topology, multipath transmission such as equal cost multipath routing (ECMP) was used to control which path of the network does each flow will take [2].

ECMP used a hash function to statically mapped each flow to a path, making every available path between two nodes utilized [3][4]. This static configuration, despite having more balanced load distribution than standard spanning tree protocol, still lacks a dynamic mechanism that accounts currently available bandwidth of each path that can contribute to an unbalanced distribution of loads in the network or even link congestion if the flows are large enough [5].

Software defined network is a concept where network architecture is divided into two parts, which are a controller (control plane) and switches that forward traffics based on controller's instruction (data plane) [6]. Having the ability to measure each link's bandwidth in real time [7], the controller can improve the dynamicity of path choosing algorithm by including currently available bandwidth in the calculation before sending the final path to the switches.

Various researches have been done to improve the dynamicity of ECMP in the data center such as [1], [5], [8] and [9]. [1] and [5] used a hash function to initialize a path like the standard ECMP and re-route the flow if it causes congestion along the way. [8] and [9] prevented congestion from occurring even more by measuring links' bandwidth beforehand and used the max-min remainder capacity (MMRCS) algorithm [13] to initialize a path. If congestion still occurs in the middle of transmission, the responsible flow(s) will be re-routed.

In this paper, we create a program as an alternative to the first half of the load balancing mechanism in [8] and [9] by measuring links' bandwidth before assigning a path to a traffic, only we use Dijkstra's widest path algorithm instead of max-min remainder capacity (MMRCS) and uses Ryu controller instead of NOX and Floodlight controller.

II. RESEARCH METHOD

A. Emulation Model

We choose Mininet to emulate a network in the form of fattree topology because it uses lightweight container-based virtualization that is capable of emulating a large scale network [10].

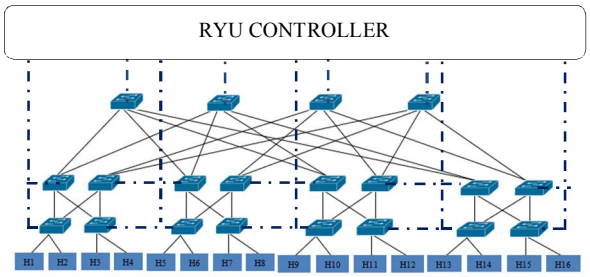

Fig.1. Network Simulation Topology

The network topology as illustrated in Fig.1 above is a fattree topology consists of 20 switches and 16 hosts, and we set the bandwidth of all links to be 8 Mbps. Each switch in the network is connected directly to the Ryu controller [14] and communicating via OpenFlow 1.3 protocol [15]. We ran the emulation on a laptop powered by quad-core 3.4 GHz CPU and 8 GB of RAM.

B. Load Balancing Algorithm

In software defined network architecture, switches forward every traffic based on its flow table, made by the controller. When a switch receives unknown traffic, it will send a packet-in to the controller as a request to update the flow table.

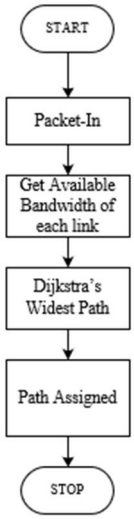

Fig.2. Program Flowchart

Figure 2 shows that our program started when the controller received a packet-in, as a request to find a path. The controller used flow_stats_request and its flow_stats_reply feature from OpenFlow to collect transmitted bytes and timestamps periodically to calculate the available bandwidth of each link. This measured bandwidth is used by Dijkstra's Widest Path algorithm along with topological information to determine which path has the biggest bottleneck available bandwidth. By doing this, the chosen path will be the least busy path thus making the load distribution in the network more balanced.

Furthermore, after the best path with the maximum available bandwidth was found, the controller informed every switch involved in the path to update its flow table. When the transmission process finished, the switches removed a specific traffic entry on its flow table in order to make sure to always send a packet-in to the controller so that the controller can choose the least busy path again.

C. Evaluation Scenario

There are two evaluation scenarios in this paper, with first scenario aimed to test the ability of the proposed scheme to choose the best path and its effect on the quality of service value and the second scenario to measure the load standard deviation to give a quantitative measure of load balancing process.

The first scenario was done by sending three 4.5 Mbps UDP data traffic from H4-H14, H5-H13, and H10-H15 as shown in Fig.1. If at least two out of three traffics transmitted on the same 8 Mbit link(s), congestion will occur thus making a decrease in QoS values. The proposed scheme is aimed to avoid this possibility. Throughput, delay, and packet loss are used as the parameter.

The second scenario was done by sending multiple G.711 VoIP traffics from H1 to H16, while two background traffics are running from H2-H13 and H4-H15 increasing from 0,4,6,8 to 10 Mbps. Throughput was measured to calculate load standard deviation using (1) and packet loss in additional.

$$LSD = \sqrt{\frac{\sum_{i=1}^{n}(load_i - \overline{load})^2}{n}} \quad (1)$$

We used a Distributed Internet Traffic Generator (D-ITG) [11] and iperf to measure the quality of service and to generate the UDP and VoIP traffic used in both scenarios.

III. RESULT

A. First Scenario

We tested the first scenario using spanning tree protocol (STP), the standard ECMP, and the SDN-Based ECMP with our proposed scheme.

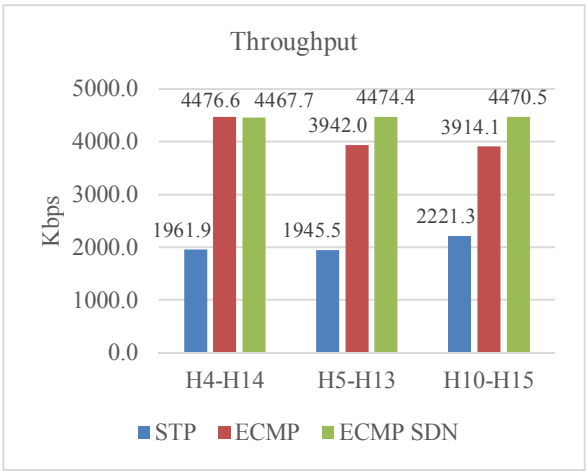

Fig.3. Throughput from First Scenario

Figure 3 shows that STP has the lowest throughput around 2 Mbit/s and the standard ECMP has also shown a decrease on H5-H13 and H10-H15 traffic from the original 4.5 Mbit/s to around 4 Mbit/s. Our proposed scheme, on the other hand, has shown a stable throughput from all of the three traffics to be around 4.5 Mbit/s. One thing to notice is that one traffic (H4-H14) has a 0.19% higher average throughput when using standard ECMP in comparison to our proposed scheme.

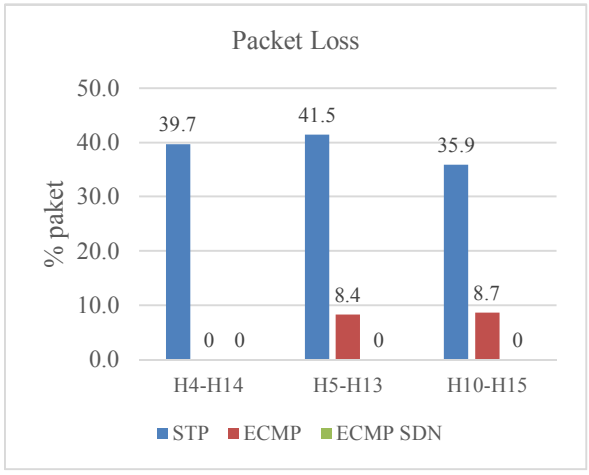

Fig.4. Packet Loss from First Scenario

Figure 4 shows that STP has the highest packet loss ranging from 35.9% to 41.5% at its peak while standard ECMP has also shown a good amount of packet loss on H5-H13 and H10-H15 traffic measuring 8.4 and 8.7%, despite having a zero packet loss on H4-H14 traffic. Our proposed scheme yields a zero packet loss either from H4-H14, H5-H13 or H10-H15 traffic.

With the same trend from throughput and packet loss, Fig.5. shows that STP has the worst delay value ranging from 691 ms to 730,2 ms. The standard ECMP has also shown a good amount of delay on H5-H13 and H10-H15 traffic which are 491 ms and 494 ms. Our proposed scheme has a stable low delay average value ranging from 13.4 ms to 13.6 ms. Just like throughput, standard ECMP has yielded a slightly lower delay (3.46%) than our proposed scheme.

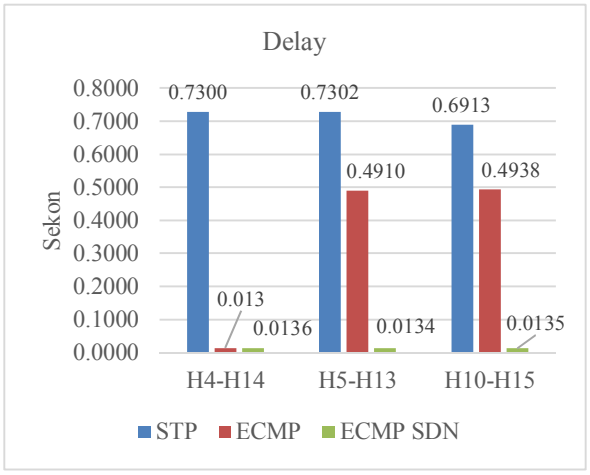

Fig.5. Delay from First Scenario

B. *Second Scenario*

We tested our scenario on the standard round-robin load balancer and our proposed scheme.

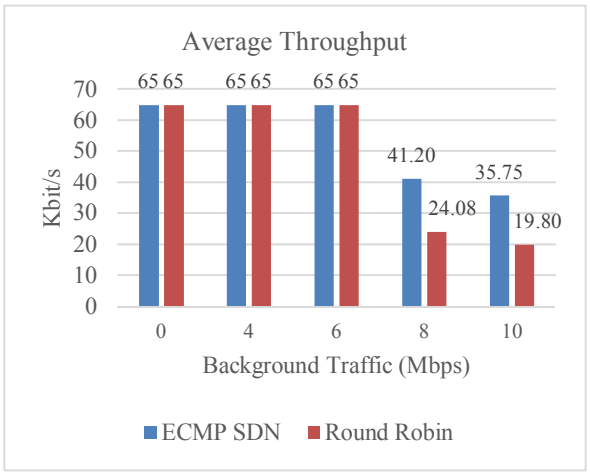

Fig.6. Throughput from the Second Scenario

Figure 6 shows that both round robin and our scheme has shown a stable 65 Kbps average throughput from 4 VoIP when the background traffics are 0, 4, and 6 Mbit/s and starting to decrease when the link is saturated which is when background traffics are 8 Mbit/s and 10 Mbit/s. Our proposed scheme has a better average throughput value than the standard round-robin, measuring 71.09% higher on 8 Mbit/s background traffics and 80.56% higher on 10 Mbit/s background traffics.

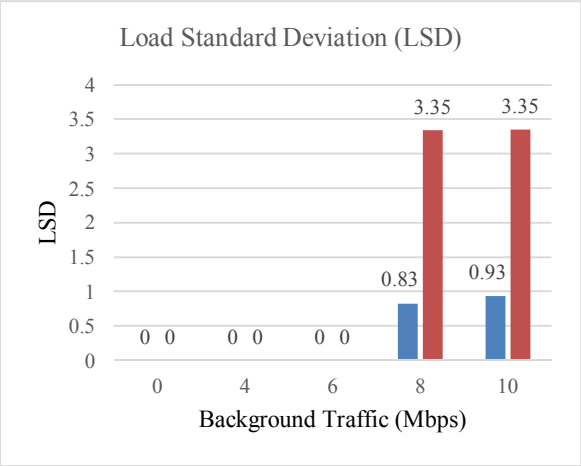

Fig.7. LSD from the Second Scenario

From the throughput, load standard deviation (LSD) is calculated using (1). Fig.7 shows that round robin and our scheme has shown a uniform zero LSD from 4 VoIPs when the background traffics are 0, 4, and 6 Mbit/s and starting to increase when the link is saturated on 8 Mbit. Our proposed scheme has a better LSD value than the standard round-robin, measuring 75.22% lower on 8 Mbit/s background traffics and 72.24% lower on 10 Mbit/s background traffics.

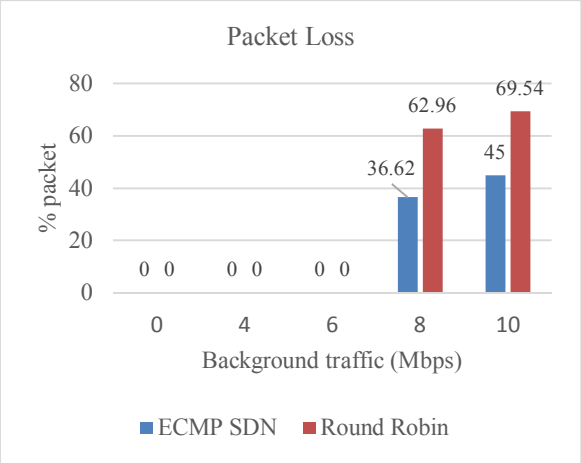

Fig. 8. Packet Loss from Second Scenario

Figure 8 shows that both round robin and our scheme has yielded zero packet loss from 4 VoIP when the background traffics are 0, 4, and 6 Mbit/s and starting to increase when the link is saturated. Our proposed scheme has shown a better quality in comparison to round robin, measuring 26.34% less packet loss on 8 Mbit/s background traffic and 24.54% less packet loss on 10 Mbit/s background traffic.

IV. DISCUSSION

A. First Scenario

Figure 3, 4, and 5 show that from the same scenario used to send three traffics, spanning tree protocol has the worst overall result. By using Wireshark, we were able to map which path that each traffic takes for transmitting the data as a result of the STP algorithm.

Fig. 9 shows that every traffic takes the same link between switch 1001 and switch 2007, making the total traffic adds up to 13.5 Mbit/s in 8 Mbit link triggering a sharp decrease in throughput and an increase in packet loss and delay values. This happened because the STP algorithm is aimed to break every loop in the network topology. As a result, there is only one core switch utilized (switch 1001), making the congestion occurred inevitably.

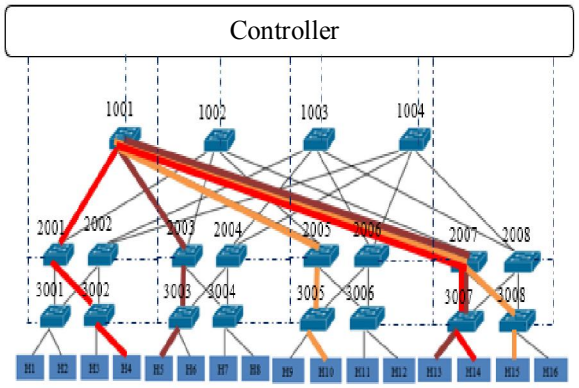

Fig. 9. STP Path Map

Figure 3, 4, and 5 show that the standard ECMP yielded a normal throughput, delay, and packet loss values on one traffic (H4-H14) and had a decrease in quality on two traffic which is H5-H13 traffic and H10-H15 traffic.

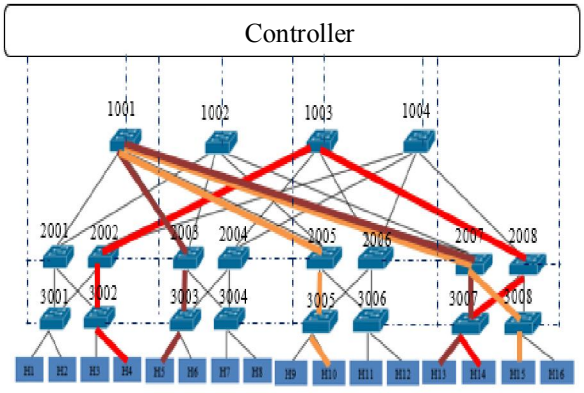

Fig.10. Standard ECMP Path Map

Figure 10 shows the mapped path of each traffic using Wireshark. It shows that the traffic between H5-H13 and H10-H15 was using the same link between switch 1001 and switches 2007 with the total traffic adds up to 9 Mbit/s on 8 Mbit/s, it explains the decrease in quality of service values on H5-H13 and H10-H15 traffic. The H4-H14, on the other hand, was using a completely different path which explains the normal quality of service values yielded earlier. This phenomenon occurred because the standard ECMP did not account a current bandwidth of the links and used a hash function instead to statically map a path to a traffic. Despite making a congestion in a link between switch 1001 and switch 2007, both H5-H13 and H10-H15 traffic was being transmitted in the same path because

the alternative paths had been assigned to another traffic, even though that specific traffic is not in the state of transmission at the moment. This problem could be solved by considering the currently available bandwidth of each link before choosing a path as our proposed scheme does.

Figure 3, 4, and 5 show that our proposed scheme had a stable near 4.5 Mbit/s throughput, zero packet loss, and near 13 ms delay values for all of the three traffic we have sent. This is explained by a completely three different paths utilized by the three traffics as shown in Fig.11.

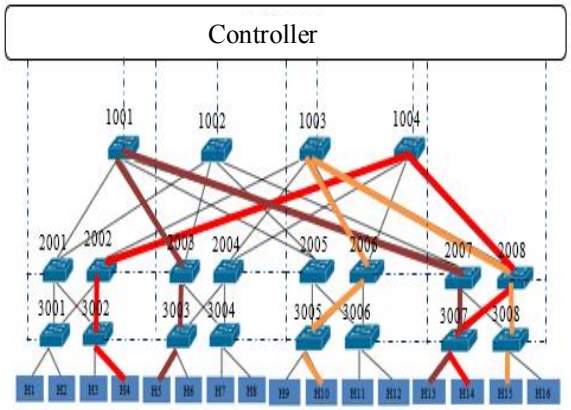

Fig. 11. Proposed Scheme Path Map

The paths that have been illustrated in Fig.11 is possible because the controller considered the available bandwidth of every link (which monitored periodically) when choosing which path traffic will take using Dijkstra's widest path. As a result, every traffic was assigned to the least busy path thus making the distribution of the load more balanced and the quality of service values are maintained.

One thing to notice is that on H4-H14 traffic as shown in figure 3, 4, and 5, our proposed scheme had a slight decrease in QoS values which are 3.46% higher delay and 0.19% lower throughput in comparison to the standard ECMP. This is caused by the additional process of Dijkstra's widest path algorithm to choose a path based on each path's bottleneck available bandwidth took more time in comparison to a statically mapped paths in the standard ECMP and there was no congestion occurred in the path of H4-H14 traffic (figure 10) that could decrease the quality of service values.

B. Second Scenario

Load standard deviation is a measurement to determine how even an algorithm can distribute traffic loads in the network. Based on formula (1), a load standard deviation value is better if its closer to zero. If traffic loads in the network are perfectly distributed thus making the throughput of each traffic has the same value, the average value of the throughput will match the same individual throughput values resulting in load standard deviation equal to zero.

Based on [12], a lower load standard deviation value indicating that specific load balancer is better than the others is only a valid statement if the average throughput of the transmitted loads is higher and the total of packet losses is lower.

Figure 7 shows that our proposed scheme had 75.22% lower LSD value than round robin on 8 Mbit/s background traffics and 72.24% lower LSD value than round robin on 10 Mbit/s background traffics. At the same time, figure 6 and figure 8 shows that at 8 Mbit/s background traffic our proposed scheme yielded 26.34% less packet loss and 71.09% higher throughput than round robin and yielded 24.54% less packet loss and 80.56% higher throughput on 10 Mbit/s background traffics, proving that our proposed scheme is a better load balancer. This improvement is the result of measuring the available bandwidth of each link before choosing a path instead of just alternately switching between available paths regardless of the bandwidth which was done on a round robin algorithm.

V. CONCLUSION

Our proposed scheme improves the ECMP load balancing capability by measuring available bandwidth of each link before choosing a path to transmit each traffic, those making each traffic assigned to the least busy path as much as possible. Our experiment proves that the proposed scheme works, yielding 14.21% higher throughput, 8.7% less packet loss and 97.27% lower delay in comparison to the standard ECMP when congestion occurs. When there is no congestion occurring, the additional Dijkstra's widest path process from our scheme causes a slight trade-off in 0.19% lower throughput and 3.46% higher delay in comparison to the standard ECMP. The proposed scheme is also proven to be a better load balancer when sending multiple traffics from the same source-destination than round robin with 75.2% lower LSD value, 24.54% less packet loss and 80.56% higher throughput.

The future work of this paper is to maximize the efficiency of Dijkstra's widest path algorithm (or trying another path finding algorithm) to minimize the trade-off as much as possible. We also want to complete the load balancing mechanism based on [8] and [9] so that our work can be reactive to a congestion occurred along the way in addition to being preventive.

REFERENCES

[1] Hailong Zhang, Xiao Guo, Jinyao Yan, Bo Liu, and Qianjun Shuai, "SDN-based ECMP algorithm for data center networks," *2014 IEEE Comput. Commun. IT Appl. Conf.*, pp. 13–18, 2014.

[2] M. Chiesa, G. Kindler, and M. Schapira, "Traffic Engineering with {ECMP}: An Algorithmic Perspective," *Proc. IEEE INFOCOM*, vol. 25, no. 2, pp. 1590–1598, 2014.

[3] C. Hopps, "Analysis of an Equal-Cost Multi-Path

Algorithm," *Doc. RFC 2992, IETF*, pp. 1–8, 2000.

[4] a. Iselt, A. Kirstadter, A. Pardigon, and T. Schwabe, "Resilient routing using MPLS and ECMP," *2004 Work. High Perform. Switch. Routing, 2004. HPSR.*, pp. 345–349, 2004.

[5] M. Al-Fares, S. Radhakrishnan, and B. Raghavan, "Hedera: Dynamic Flow Scheduling for Data Center Networks.," *Nsdi*, p. 19, 2010.

[6] F. Ieee *et al.*, "Software-Defined Networking : A Comprehensive Survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[7] A. L. Hf *et al.*, "Multipath Routing with Load Balancing and Admission Control in Software Defined Networking (SDN)," vol. 4, no. c, pp. 4–9, 2016.

[8] H. Long, Y. Shen, M. Guo, and F. Tang, "LABERIO: Dynamic load-balanced routing in OpenFlow-enabled networks," *Proc. - Int. Conf. Adv. Inf. Netw. Appl. AINA*, pp. 290–297, 2013.

[9] Y. L. Lan, K. Wang, and Y. H. Hsu, "Dynamic load-balanced path optimization in SDN-based data center networks," *2016 10th Int. Symp. Commun. Syst. Networks Digit. Signal Process. CSNDSP 2016*, pp. 0–5, 2016.

[10] J. Liu and L. Butler, "A Simulation and Emulation Study of SDN-Based Multipath Routing for Fat-tree Data Center Networks," pp. 3072–3083, 2014.

[11] A. Botta, A. Dainotti, and A. Pescapé, "A tool for the generation of realistic network workload for emerging networking scenarios," *Comput. Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.

[12] C. Wang, G. Zhang, H. Xu, and H. Chen, "An ACO-based Link Load-Balancing Algorithm in SDN," pp. 221–225, 2016.

[13] D. Bertsekas and R. Gallager, "Data Networks", Chapter 6, Prentice Hall, 1992.

[14] Ryu 4.26 Documentation. [Online]. Available: https://ryu.readthedocs.io/en/latest/index.html

[15] OpenFlow. [Online]. Available: https://www.opennetworking.org/technical-communities/areas/specification/open-datapath/