Indonesian Journal of Electrical Engineering and Computer Science
Vol. 22, No. 1, April 2021, pp. 307~314
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v22.i1.pp307-314

Generation and collection of data for normal and conflicting flows in software defined network flow table

Mutaz H.H.Khairi1, Sharifah H. S. Ariffin, N. M. Abdul Latiff, Kamaludin Mohamad Yusof
School of Electrical Engineering, Faculty of Engineering, Universiti Teknologi Malaysia, Johor, Malaysia

Article Info

Article history:

Received Nov 19, 2020
Revise Jan 9, 2021
Accepted Jan 29, 2021

Keywords:

Conflict policies
Forwarding policies
Software defined network
TCP and UDP
Throughput

ABSTRACT

In terms of network simplification and regulation, software defined networking (SDN) is a new form of infrastructure that offers greater adaptability and flexibility. SDN, however, is an invention that is logically centralized. In addition, the optimization of the control plane and data plane in SDN has become an area deserving of more attention. The flow in OpenFlow has been one of the essential parameters in the SDN standards, in which every individual flow includes packet matching fields, flow priority, separate counters, instructions for packet forwarding, flow timeouts and a cookie. This research work is conducted to generate and collect flows from the OpenFlow switch in two scenarios; normal flows and when conflict policy rules are enforced in the network. In this article, throughput is used as a performance metric to evaluate the impact of flow conflict on two protocols, transmission control protocol (TCP) and the user datagram protocol (UDP). During the simulation of the SDN OpenFlow network, the metrics are tested using MININET. The results reveals that the existence of SDN conflict rules forces TCP and UDP to have a significant average change in bandwidth that eventually affects the network and operations performance.

Corresponding Author:

Mutaz Hamed Hussien Khairi
Faculty of Engineering
Universiti Teknologi Malaysia
Johor, Malaysia
Email: hhkmutaz2@graduate.utm.my

1. INTRODUCTION

In traditional distributed networks, operation of choice-producing systems identified by the control plane and transfer of data traffic (data plane) are implemented across networking devices (e.g., hubs or routers). This allows network providers individually manage the flow rules (e.g. forwarding, routing, service quality) for each unit [1]. In software-defined networking (SDN), a network architecture which splits the control logic and forwarding functions into various layers [2, 3], the control functions are organized through the SDN controller which is a logical component that operates as a control application framework. This software provides standards that control the actions of the data plane equipment. The machines only maintain the basic functions of transmitting messages in compliance with the rules stored in their flow tables on how they are processed [4].

OpenFlow (OF) is the standard network protocol that is used in SDN. One of its core elements of the SDN structure is the controller, which facilitates application creation through the north-bound API by functioning as the network operating system. The action of the controller greatly affects SDN; thus, its effectiveness significantly determines the performance of the SDN [5, 6]. The OpenFlow structure in SDN is shown in Figure 1. OpenFlow switches are made up of a variety of flow tables that are connected via an

OpenFlow protocol to a controller. This protocol has been used a means of transport or link between the switches and the controller. Flow tables are configured to ensure that packets are correctly stored and transmitted [7, 8]. A flow table is made up of several flow entries that include: Match fields, being used to match flows; Priority, being used to match the priority address of the flow; Counters, to be updated to match the packets; Instructions, to adapt the action taken or to handle the flow; Timeouts, total time or idle time before the timeout ends and; Cookie, the implicit value of the data chosen by the controller. The controller may use it to filter, adjust and delete flow data, but is not used when processing packets [6, 9].

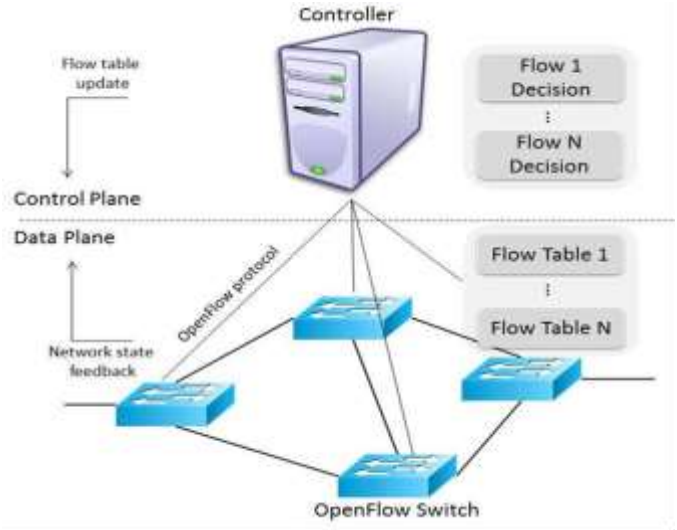


Figure 1. OpenFlow SDN architecture [6]

Both Traditional networks and SDN s are affected by several types of flow conflicts which limit network performance [10]. Flow conflicts can be classified based on their rules and effect on the network into two main types, namely: Intelligible and Interpretative Conflicts. Since packet counters and timeout values are not important in the management of flow conflicts, the flow entries considered in the remainder of this study are limited to priority, match fields, and action fields.

Conflicts that occur in the SDN depend on the impact and adjustment of features, such as priority and action. Several conflict forms appear in the controller and flow table when changes are made to flow rule policy or flow entry. In addition to being the major features that differentiate the traditional networks and SDNs, priority and actions also serve as the key components for developing the rule and flow entry in SDNs. To predict and detect flow conflicts in SDN, a labelled dataset must be readily prepared. This process requires all available referenced SDN datasets to be reviewed and evaluated. However, not all datasets that have been checked and reviewed include OpenFlow table features like priority and action. For this purpose, two topologies referenced by GitHub are used to generate and save flow entries [4]. These topologies have been used in previous research for flow conflict detection [9]. For the generation and collection of dataset in SDN, several studies and researches have been conducted using generation application to collect the generated flow from open flow switch as [1]. Also, several projects and researches exist that use GitHub as a reference for building and implementing the topology [11-13]. In this research, ryu controller is chosen to be used to enable implementation and updating of open flow table rules using python programming language. The log file in the switch captures and saves diverse file formats. Other research studies have also been conducted using Ryu controller in SDN [14-23].

2. PROPOSED METHOD

The proposed solution will be implemented and run in the ryu controller as shown in Figure 2. The solution is used to generate and collect conflicting and normal flows from OpenFlow table version 1.3. The proposed model consists of three main phases, which are: generation of normal flow from running topology; creation and implementation of conflict rules in open flow table; and generation of conflicting flow.

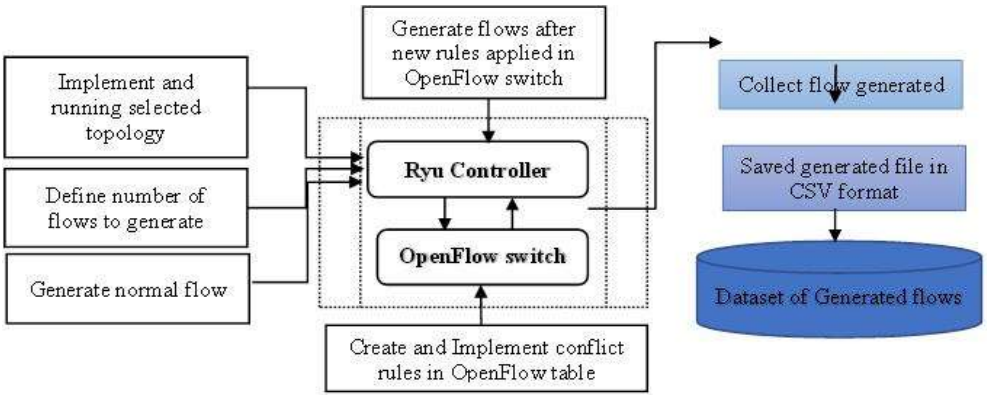


Figure 2. The proposed generation flows model

The algorithm to implement conflict policy rules with the generation and collection of OpenFlow switch flow entries is shown in algorithm 1.

Algorithm 1. Generation and Collection SDN flow with implementation of conflict policy rules in OpenFlow switch

Input: flow required L, host R, flow attack D, flow1 ð, flow2 ñ, priority P, Protocol T, action Á.
Output: conflict rules in OpenFlow switch, CSV file for normal and conflict flows.
Procedure: topo (), ConRule (), geneflow (), csv ()

1. Topology implementation has function of (L, D, R, x)
2. L=(L*20/100) +L
3. D=L/2
4. D= D/2, D/10+
5. For x in range (1, R+1)
6. Update 7 conflict rules
7. Conflict percentage = 10
8. **if** (Pð > Pñ, Tð = Tñ, Áð = Áñ, Ỹ ð, ð. *addr* ⊆ ñ. *addr* or ñ. *addr* ⊆ ð. *addr*) **then**
9. return redundancy conflict
10. **else if** (Pð < Pñ, Tð= Tñ, Áð ≠ Áñ, ð. *addr* ⊆ ñ. *addr*) **then**
11. **return** shadowing conflict
12. **else if** (Pð < Pñ, Tð = Tñ, Áð ≠ Áñ, ñ. *addr* ⊆ ð. *addr*) **then**
13. **return** Generalization conflict
14. **else if** (Pð < Pñ, Tð=Tñ, Áð ≠ Áñ, ð. *addr* ∩ ñ. *addr*) **then**
15. **return** correlation **A** conflict
16. **else if** (Pð = Pñ, Tð=Tñ, Áð ≠ Áñ, ð. *addr* ⊆ ñ. *addr* or ñ. *addr* ⊆ ð. *addr*) **then**
17. **return** correlation B conflict
18. **else if** (Pð < Pñ, Tð=Tñ, Áð = Áñ, ð. *addr* ∩ ñ. *addr*) **then**
19. **return** overlap conflict
20. **else**, (Pð = Pñ, Tð = 0, Áð = Áñ, ð. *addr* ∩ ñ. *addr* or ñ. *addr* = ☋) **then**
21. **return** imbrication conflict
22. Flow cookie = 0
23. start L4
24. **if** buffer Id **then**
25. mod = ofpflow, cookie = cookie id, buffer = buffer id, P= P, match =match, instruction =inst
26. **else** mod = ofpflow, cookie = cookie id, P=P, match =match, instruction =inst
27. ip = pkt.get
28. srcip = ip.src
29. dstip = ip.dst
30. Protocol = ip protocol
31. match of protocol ICM or TCP or UDP
32. Import CSV
33. Update CSV
34. f name = switch +str(dpid)
35. Write CSV (fp, delimiter)
36. Writer row (header)
37. Writ.writerow(row)

The following definitions have been used and applied in [10, 24]:

Definition 1.1 The subset of set of every possible hexadecimal numbers of 6-byte which also has OSI layer-2 (MAC) addresses is known as a frame space of a rule ***r***. This corresponds to a 2-tuple (ϵ_s, ϵ_d) where subscripts s and d denote source and destination addresses respectively.

Definition 1.2 The subset of the set of all possible 32-bit numbers which represents the addresses of OSI layer-3 (IPv4) is known as a packet space of rule r. This corresponds to a 2-tuple (ζ_s, ζ_d) where subscripts s and d denote source and destination addresses respectively.

Definition 1.3 A rule r segment space is defined as the subset of all possible set of 16-bit numbers that represents the layer-4 (TCP / UDP) addresses of the OSI. This corresponds to a 2-tuple (η_s, η_d) where subscripts s and d denote source and destination addresses respectively.

Definition 1.4 A 6-tuple of frame space, packet space and segment space is called the address space ***n*** of a rule ***r***. This is represented as (ϵ_s, ϵ_d, ζ_s, ζ_d, η_s, η_d), where subscripts s and d denote source and destination addresses respectively. Since ***N*** is the universal set of space addresses, we have:

Definition 1.5. A function $\boldsymbol{f: N \rightarrow N}$ that transforms ***n*** to $\boldsymbol{n'}$ is defined as a flow rule r; where (ϵ'_s, ϵ'_d, ζ'_s, ζ'_d, η'_s, η'_d) denotes $\boldsymbol{n'}$ alongside an associated action set a, that is capable of having any of the values. Hence,

$$r: = f\ (n) \qquad a$$

In the action fields of the rules, a set-field capability guarantees that all or none of the fields in ***n*** can be changed due the transformation function ***f***. The transformation function can add to the outcome of the initial transformation of n' , considering instances where the operation set is a reference to a particular flow table. Formally,

$$\text{if } r := f\ (n) \text{ ----- } a;\ f\ (n) = n' \text{ and } a := g(n') \text{----- } a' \text{ then, } r := g\ (f\ (n)) \text{ ---- } a'$$

3. METHODOLOGY

One of the research objectives of this study is to generate, collect and save normal and conflicting flows of a simulated SDN. Mininet platform [25] is used for this work in conjunction with ryu controller [26] and mininet simulator. Virtual Box [27], which is a virtual communication network on Mininet is also installed. Table 1 lists the simulation specifications of the simulation environment.

Table 1. Specifications of the framework and the setting for the implementation of MININET

Software /Hardware	Specifications
Processor	Intel Core i5
RAM	12 GB
System	Ubuntu 18.04
Controller	Ryu
Switch	OpenFlow Version 1.3

Network topologies are created in mininet and linked to the ryu controller. Both switches and hosts in the topologies are interconnected by a python programming language-based application named *Topo.py*. Figure 3 shows the two topologies created in this work (i.e., simple tree and fat tree topology) that generates the traffic automatically. The traffic generation consists of 1000 to 100000 flows, n flows. The 10 *iperf* servers starts on each host. Every server listens to different ports (i.e., 8081, 8082, 8089). Flow generation is carried out to produce between 1000 and 100000 flows. This process entails a simple switch 1.3 application, used as the base application as a L4 Match application is created.

The following is used to generate new flows: *src / dst ip, src/dst* port and protocol. Every packet is passed on to the controller after which the controller installs a new flow in the switch. In OpenFlow switch version 1.3, the switch training method is still the same as earlier versions, however, flows now depend on Layer 4 Match rather than Layer 2. In order to create flow conflict, all flows are initially gathered in the controller where 10% of the flows are selected. The policy rules in the OpenFlow table are then updated to get new conflict flow response based on changes in the match field. After normal and conflicting flow creation has been completed in all of the switches throughout the topology, data capture is carried out by executing the flowstat application; this application is used to capture and save all the flow entries in CSV file.

Python 2.7 programming language is used in a Linux environment to implement all the essential functions of the proposed method. The RYU controller is used in this experiment to ensure connection to OpenFlow switch version 1.3 to enable the two topologies to run and analyze the data. After all these phases

and procedures have been implemented and executed, and all the required data are generated and obtained from the OpenFlow switch. Figure 4 shows the samples of normal and conflicting flow entries collected and saved from OpenFlow switch.

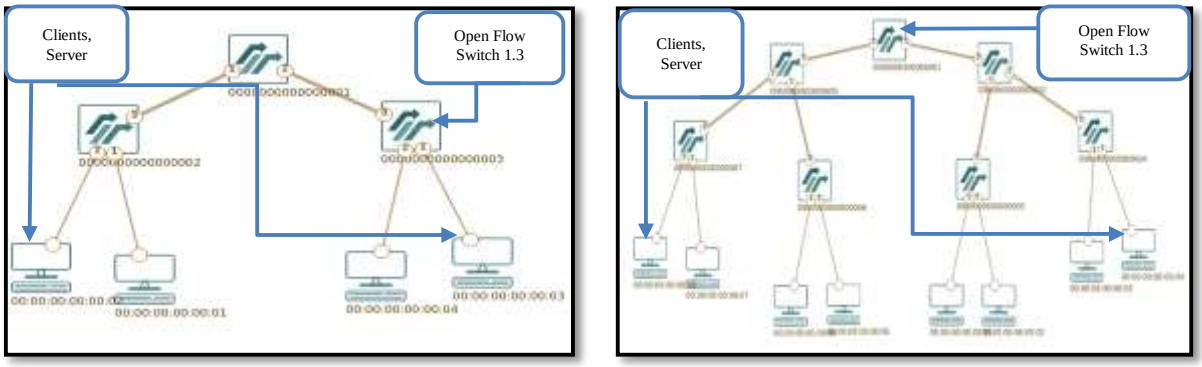


Figure 3. The network topologies used in MININET

cookie'	priority'	src_mac'	dst_mac'	src_ip'	dst_ip'	protocol'	src_port'	dst_port'	action'	byte_cour	packet_co	hard_time	idle_time	duration'
120853	100	00:00:00:0	00:00:00:0	10.5.50.0/	10.211.2.6	tcp	25129	8080	forward	162	3	0	0	193
120041	100	00:00:00:0	00:00:00:0	10.5.50.0/	10.211.1.6	tcp	5054	8085	forward	162	3	0	0	248
121451	100	00:00:00:0	00:00:00:0	10.211.2.0	10.5.50.5	tcp	8080	25171	forward	0	0	0	0	185
111414	100	00:00:00:0	00:00:00:0	10.211.2.0	10.5.50.5	tcp	8080	25152	forward	232	4	0	0	185
211974	100	00:00:00:0	00:00:00:0	10.211.2.0	10.5.50.5	tcp	8080	25066	forward	58	1	0	0	191
500058	103	00:00:00:0	00:00:00:0	10.5.50.5	10.211.1.6	tcp	5072	8083	drop	0	0	0	0	138
221731	100	00:00:00:0	00:00:00:0	10.5.50.0/	10.211.2.6	tcp	25043	8080	forward	162	3	0	0	195
111431	100	00:00:00:0	00:00:00:0	10.211.2.0	10.5.50.5	tcp	8080	25166	forward	0	0	0	0	185
111560	100	00:00:00:0	00:00:00:0	10.211.2.0	10.5.50.5	tcp	8080	25327	forward	0	0	0	0	183
500052	103	00:00:00:0	00:00:00:0	10.5.50.5	10.211.1.6	tcp	5071	8083	drop	0	0	0	0	138
141396	100	00:00:00:0	00:00:00:0	10.211.2.0	10.5.50.5	tcp	8080	25146	forward	0	0	0	0	185

Figure 4. Sample of normal and conflict flows collected from the OpenFlow switch

4. RESULTS AND DISCUSSION

In this work, experimental simulations have been carried out in two scenario topologies as in Figure 3. All testing and evaluation are executed according to the proposed method illustrated in Figure 2.

4.1. Result

Transmission control protocol (TCP) as well as user datagram protocol (UDP) have been leveraged for transmission packet across multiple connected devices in the network. (TCP) is a two - way data connection known to the source host from the destination host after all packets have been obtained by the destination. UDP is used to transfer smoother data flow between connected devices with monitoring and fail rate testing [6]. The quality of TCP and UDP protocols in terms of bandwidth and transfer rate have been compared and tested in pertinent relevant studies [28, 29]. In [30, 31], research simulations were performed to illustrate and evaluate the two protocols (TCP and UDP) in SDN networks. The algorithm presented in this research is evaluated on the two chosen topologies. TCP and UDP protocols are evaluated and checked for normal and conflicting flows over short and long periods according to the bandwidth.

4.1.1. TCP bandwidth forwarding throughput

TCP bandwidth forwarding throughput is simulated for both tree and fat tree topologies in Figure 5(a, b). The test was initially applied for short time between 0-120 seconds before increasing it to 3600 sec (1 hour) as in Figure 5(c, d). The result displays the throughput directly in Gbits per second for TCP traffic for one host connected to server within specific time interval.

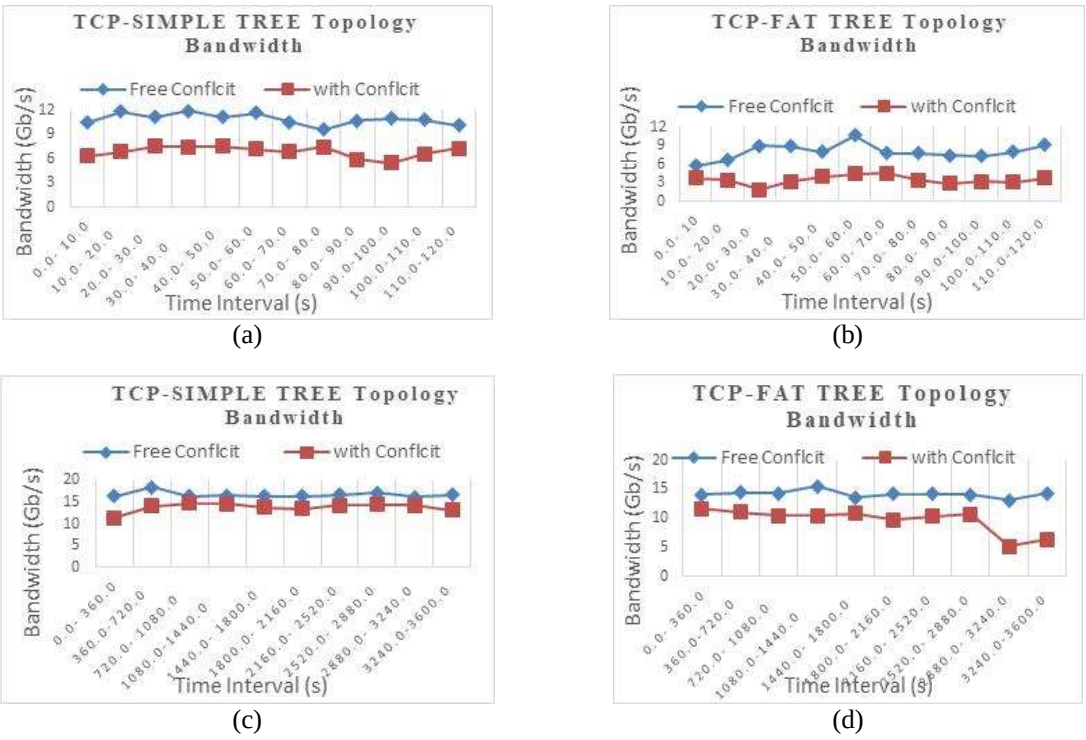


Figure 5(a, b). Comparison of TCP Bandwidth forwarding throughput for 120 sec, (c, d) Comparison of TCP Bandwidth forwarding throughput for 1 hour

4.1.2. UDP bandwidth forwarding throughput

UDP bandwidth forwarding throughput is simulated for both tree and fat tree topologies in Figure 6(a, b). The test was initially applied for short time between 0-120 seconds before increasing it to 3600 sec (1 hour) as in Figure 6 (c,d). The result shows the output in Mbits per second for host to server UDP traffic over specific time interval.

Figure 5 and Figure 6 show the result of the two protocols (TCP/UDP) based on the respective througputs of the two types of topologies implemented in this research. The blues line in the figures indicate the normal flows while the brown line depict flows when conflict rules are implemented and updated in OpenFlow table. The results show that the conflict rules effects the network bandwidth, such that the bandwidth of flows with conflicts are lower compared to bandwidth of normal flows across all test times.

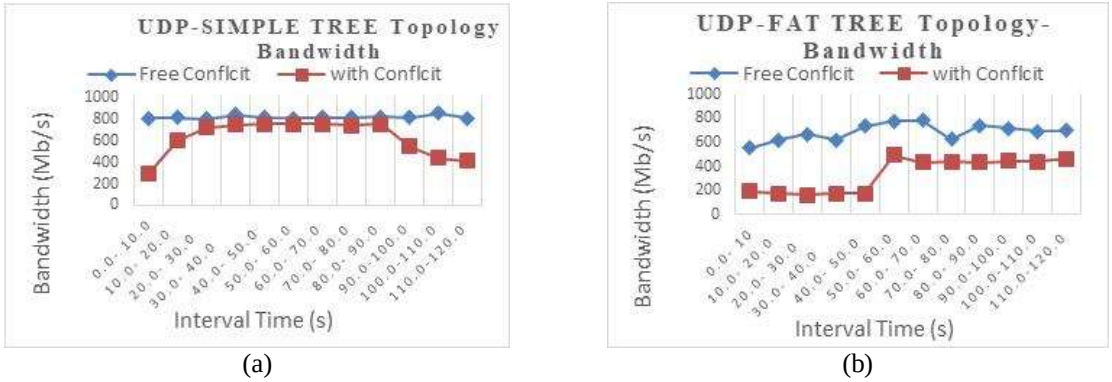
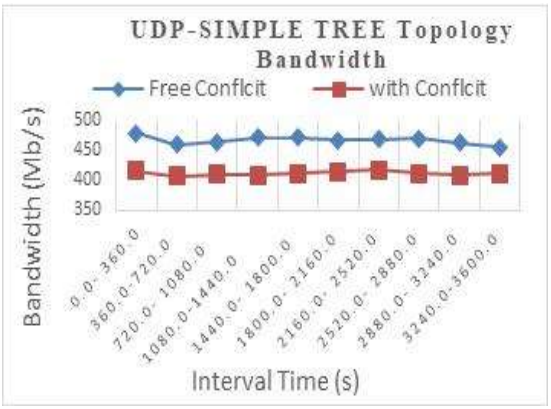
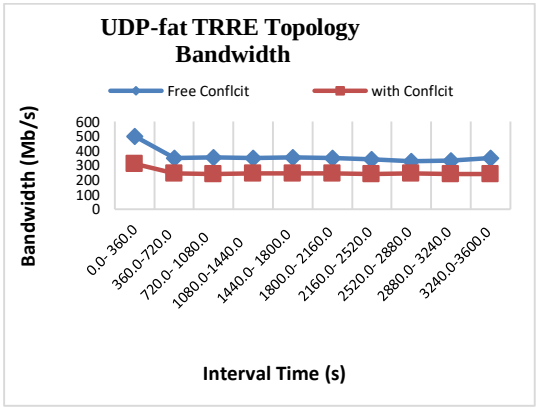


Figure 6(a,b). Comparison of UDP bandwidth forwarding throughput for 120 sec



(c)



(d)

Figure 6(c, d). Comparison of UDP bandwidth forwarding throughput for 1hour

4.2. Discussion

Two alternative topologies have been implemented and tested to measure and evaluate the bandwidth over specific times interval for TCP and UDP. Figure 5(a, b) provides a comparison of the short-term forwarding throughput for TCP (0-120 sec). This shows a significant 37 % and 56% average decrease in throughput for simple tree and fat tree topologies respectively; due to conflict rules introduced and applied in the OpenFlow table. This decline happens when load is added to determine the traffic in flow table. Also, as shown in Figure 5(c, d) for time interval between 0-3600sec, the change in simple tree topology was 17 % while 31% for the fat tree topology. Figure 6(a, b) shows the results of findings on evaluating the second protocol (UDP). In this setup, a respective drop of 23% and 52 % for simple tree and fat tree topologies was observed while the network operates for a short period of time (0-120 sec). The drop that occurs when measuring the UDP shows why the UDP has a signigicant affect on essential operations such as real-time and online communications applications. The comparsion of bandwidth in Figure 6(c, d) for longer time interval of to 3600 seconds shows 11 % and 30 % drop in bandwidth of simple tree and fat tree topologies respectively.

5. CONCLUSION

This paper presents the generation and collection of flows in SDN. The flows were generated in two different scenarios; normal flows generation with normal producers and conflict policy rules application and update in the OpenFlow table to generate conflicting flows. The generation was carried out in two scenarios using two variations of network topologies (i.e., simple Tree and Fat Tree). The number of flows considered in this study ranges from 1000 to 10000 flows. In normal forwarding SDN, traffic generation is evaluated and checked using the throughput parameter of TCP and UPD protocols with and without the conflict policy rules modules. The mininet software emulator was used to perform the simulation results. The work shows that the implementation of rules for different conflict has significant effect on the bandwidth. The mininet network simulation shows that the drop in throughput is due to the OpenFlow implementation of conflict rules. The findings also showed a clear problem of conflict in SDN and its impact. Therefore, this requires further investigation to effectively resolve this problem. One of the findings is that in order to eliminate the conflict in SDN, it is important to detect and classify the conflict according to its forms. Thus, the future research will be dedicated to the identification and classification of flow conflicts.

REFERENCES

[1] M. S. Elsayed, N.-A. Le-Khac, and A. D. Jurcut, "InSDN: A Novel SDN Intrusion Dataset," *IEEE Access*, vol. 8, pp. 165263-165284, 2020, doi: 10.1109/ACCESS.2020.3022633.

[2] T. A. Assegie and P. S. Nair, "A review on software defined network security risks and challenges," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 17, no. 6, pp. 3168-3174, 2019, doi: 10.12928/telkomnika.v17i6.13119.

[3] T. E. Ali, A. H. Morad, and M. A. Abdala, "Traffic management inside software-defined data centre networking," *Bulletin of Electrical Engineering and Informatics*, vol. 9, no. 5, pp. 2045-2054, 2020, doi: 10.11591/eei.v9i5.1928.

[4] V. Danciu and C. N. Tran, "Side-Effects Causing Hidden Conflicts in Software-Defined Networks," *SN Computer Science*, vol. 1, no. 5, pp. 1-16, 2020, doi: 10.1007/s42979-020-00282-0.

[5] M. Cheng, *et al.*, "Flow Setup Rate Test for OpenFlow Controller," June 25, 2017.
[6] H. Khairi, *et al.*, "The impact of firewall on TCP and UDP throughput in an openflow software defined network," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 20, no. 1, pp. 256-263, 2020, doi: 10.11591/ijeecs.v20.i1.pp256-263.
[7] B. A. A. Nunes, *et al.*, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617-1634, 2014, doi: 10.1109/SURV.2014.012214.00180.
[8] N. McKeown *et al.*, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, 2008.
[9] [online]http://networkstatic.net/wp-content/uploads/2013/02/openflow-spec-v1.3.0.pdf.
[10] S. Pisharody, J. Natarajan, A. Chowdhary, A. Alshalan, and D. Huang, "Brew: A security policy analysis framework for distributed sdn-based cloud environments," *IEEE transactions on dependable and secure computing*, 2017.
[11] [online]https://github.com/iist-sysnet/OpenSDNDataset.
[12] [online]https://github.com/parasgulati8/SDN-Simulation-with-OpenFlow.
[13] F. Hauser, M. Schmidt, M. Häberle, and M. Menth, "P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection With MACsec in P4-Based SDN," *IEEE Access*, vol. 8, pp. 58845-58858, 2020, doi: 10.1109/ACCESS.2020.2982859.
[14] S. Y. Mehr and B. Ramamurthy, "An SVM Based DDoS Attack Detection Method for Ryu SDN Controller," in *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies*, 2019, pp. 72-73, doi: 10.1145/3360468.3368183.
[15] R. K. Arbettu, R. Khondoker, K. Bayarou, and F. Weber, "Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers," in *2016 17th International telecommunications network strategy and planning symposium (Networks)*, 2016: IEEE, pp. 37-44, doi: 10.1109/NETWKS.2016.7751150.
[16] S. Asadollahi, B. Goswami, and M. Sameer, "Ryu controller's scalability experiment on software defined networks," in *2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC)*, IEEE, 2018, pp. 1-5, doi: 10.1109/ICCTAC.2018.8370397.
[17] P. Raghav and A. Dua, "Enhancing flow security in ryu controller through set operations," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, IEEE, 2017, pp. 1265-1269, doi: 10.1109/CompComm.2017.8322746.
[18] M. F. Ramdhani, S. N. Hertiana, and B. Dirgantara, "Multipath routing with load balancing and admission control in Software-Defined Networking (SDN)," in *2016 4th International Conference on Information and Communication Technology (ICoICT)*, IEEE, 2016, pp. 1-6, doi: 10.1109/ICoICT.2016.7571949.
[19] Y. Chen, W. Chen, Y. Hu, L. Zhang, and Y. Wei, "Dynamic load balancing for software-defined data center networks," in *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, Springer, 2016, pp. 286-301, doi: 10.1007/978-3-319-59288-6_26.
[20] S. Liu, T. A. Benson, and M. K. Reiter, "Efficient and safe network updates with suffix causal consistency," in *Proceedings of the Fourteenth EuroSys Conference*, 2019, pp. 1-15, doi: 10.1145/3302424.3303965.
[21] C. S. Khin, M. Z. Oo, and A. T. Kyaw, "Packet-in Messages Handling Scheme to Reduce Controller Bottlenecks in OpenFlow Networks," in *2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, IEEE, 2020, pp. 502-505, doi: 10.1109/ECTI-CON49241.2020.9158127.
[22] S. Usman, I. Winarno, and A. Sudarsono, "Implementation of SDN-based IDS to protect Virtualization Server against HTTP DoS attacks," in *2020 International Electronics Symposium (IES)*, IEEE, pp. 195-198, 2020, doi: 10.1109/IES50839.2020.9231699.
[23] R. C. Meena, M. Bundele, and M. Nawal, "RYU SDN Controller Testbed for Performance Testing of Source Address Validation Techniques," in *2020 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE)*, IEEE, 2020, pp. 1-6, doi: 10.1109/ICETCE48199.2020.9091748.
[24] S. Pisharody, "Policy conflict management in distributed SDN environments," Arizona State University, 2017, doi:
[25] [online]http://mininet.org/download/#option-1-mininet-vm-installation-easy-recommended.
[26] [online]https://readthedocs.org/projects/ryu/downloads/pdf/latest/.
[27] [online]https://www.virtualbox.org/wiki/Downloads.
[28] Y. Gu and R. L. Grossman, "UDT: UDP-based data transfer for high-speed wide area networks," *Computer Networks*, vol. 51, no. 7, pp. 1777-1799, 2007, doi: 10.1016/j.comnet.2006.11.009.
[29] P. Monika, R. M. Negara, and D. D. Sanjoyo, "Performance analysis of software defined network using intent monitor and reroute method on ONOS controller," *Bulletin of Electrical Engineering and Informatics*, vol. 9, no. 5, pp. 2065-2073, 2020.
[30] M.-H. Wang, L.-W. Chen, P.-W. Chi, and C.-L. Lei, "SDUDP: A reliable UDP-Based transmission protocol over SDN," *IEEE Access*, vol. 5, pp. 5904-5916, 2017, doi: 10.1109/ACCESS.2017.2693376.
[31] H. T. Zaw and A. H. Maw, "Traffic management with elephant flow detection in software defined networks (SDN)," *International Journal of Electrical & Computer Engineering (2088-8708)*, vol. 9, no. 4, p. 3203, 2019, doi: 10.11591/ijece.v9i4.pp3203-3211.