A Proposed Linear Multi-Controller Architecture to Improve the Performance of Software Defined Networks

Haeeder Munther Noman1,2 and Dr. Mahdi Nsaif Jasim3
1 College of Information Technology, University of Babylon, Iraq
2 Technical Instructors Training Institute, Middle Technical University, Baghdad, Iraq
3 University of Information Technology and Communications, Baghdad, Iraq

hydermu79@gmial.com, mahdinsaif@uoitc.edu.iq

Abstract. In the last few years, there has been a growing interest in the rise of demand for moving from traditional networks towards software-defined networks, which has raised a lot of challenges. Software-defined networks are continually evolving, which include the need to address issues such as scalability, packet loss, transmission delay, and network congestion. Accordingly, researchers introduced the concept of multi-controller architectures, although it will not assist to balance the load between them, it will tackle the network congestion issue through the distribution of load between them. The present study proposes a linear multi-controller architecture to explore the impact of increasing the number of controllers connected in a linear style on network performance. The study was based on the generation of simultaneous multi-flows with different sizes using Distributed Internet Traffic Generator (D-ITG). From the outcomes of our investigation, it is possible to conclude that the performance-enhanced uniformly with the number of controllers while preserving the same number of Open Vswitches. As the number of controllers reached four, the Quad-controller architecture recorded the best results related to improving the reduction of average delay and average jitter to 62% and 64% as well as increasing the throughput, bytes received, and average packet rate to 32%, 32%, and 31.8%.

1. Introduction

In SDN architecture [1], there are two distinct interfaces namely northbound and southbound [2]. Northbound interface communicates within the controller through application programme interfaces (APIs). SDN has emerged as the latest, programmable, vendor-neutral which overcomes the shortcomings of long-established network architecture through the abstraction of the main intelligence of the network, i.e. the control plane from the forwarding data plane. Applications such as traffic engineering, network virtualization, Quality of Service monitoring, routing and any other application reside in the application layer. The control layer globally regulates the network states via network policies in either a centralized or a distributed manner. Due to the unrestricted access to global network elements and resources, such network policies can be timely updated to react to the current flow activity. Support is provided for a set of application programmable interface (such as northbound APIs) in order to launch a communication between the application and control layer to permit common network services as well as network management, in other words, these interfaces facilitate different business objectives for managing the network [3]. Data forwarding layer, on the other hand, can use a programmable OpenFlow switches via OpenFlow protocol and the switches communicate with the controller via northbound APIs, which is identified as the interface between the controller and the forwarding device. It allows automating the networking tasks by providing some enormous possibilities for network programming. OpenFlow protocol is the best example for the southbound interface; it shows the execution of the switch to controller communications and vice versa. However, in the last few years, there has been a growing interest in the SDN field, many researchers have been working to enhance the traditional single controller to move towards multi-controller SDN architecture. In this work, we are
analysing the performance of the controller considering two architectures, the central and distributed. The motivation is to support the opinion that distributed architecture utilizes the resources in a better way than central architecture.

2. Related work
So far, the Distributed Multi-controller architecture was introduced in order to mitigate the problem of a single point of failure [4]. However, many drawbacks were associated with this strategy like controller redundancy due to the utilization of passive controllers as active ones in order to ensure that the controllers can be switched on/off and add each controller that has the same degree of responsibility [2]. ONIX [6] was one of the most spoken about controller because of the distributed nature of the controller plane with the physical servers' cluster; ONIX proposed three approaches to increase network scalability. The first approach was the logical partitioning of the network to propagate the workload on several ONIX instants. In the Second approach, ONIX allowed multiple nodes to existing in the upper layer called the "aggregations" to appear as a single node. third approach ONIX permitted applications in data state to boost network stability and reliability. Beacon [7] and Nox [5] employed multithreading techniques to logically divide the single controller to improve the performance through the suggestion of a balanced load architecture for wide range OpenFlow networks named BalanceFlow [6]. BalanceFlow method used CONTROLLER X for switching and cross controller communication; select a controller to be a super controller that prevents the transmission delays. HybridFlow [7] was proposed to overcome the super-controller scalability issue through the integration of load distribution method and the centralized controller by the management of a cluster of controllers using a super-controller, depending on the load on cluster then super-controller divided the flow by a threshold between the internal and external cluster, among all the approaches the HybridFlow method was regarded as the best one but with one major drawback of being incapable to provide a solution for super-controller failure. HyperFlow [8] utilized a "publish/subscribe" messaging model to propagate the information inside the control plane which make sure that the same controller holds the ordering of events released, another advantage was the minimization of the traffic needed for inter-controllers to keep less overhead, the "publish/subscribe" program run at the top of wheels. ONOS [9] was considered as one of the distributed SDN controllers which contained two different prototypes, the first presented the global view of the network while on the other hand the second prototype was responsible for improving the prototype. A distributed multi-domain SDN controller called DISCO [10] has been introduced with inter-domain and intra-domain parts, the intra-domain part performed the network control and management of flow priority settings besides the inter-domain controller part provided the communications among multiple controllers. However, in the last few years, there has been a growing interest in the SDN field, many researchers have been working to enhance the traditional single controller to move towards multi-controller SDN architecture. In this work, we are analysing the performance of the two distinct architectures in order to support the conclusion that multi-controller utilize the resources in a better way than single controller architecture. The emphasize will be in providing research results comparison between multiple and single controller SDN architectures.

3. Problem Statement
In a centralized SDN, important parameters such as scalability, availability, and efficiency were not completely fulfilled [11]. In fact, the emergence of distributed multi-controller SDN architecture played an important role in addressing these issues. In addition to its contribution in avoiding the single point of failure issue. The aim of this study is the performance exploration of the distributed multi-controller architecture and to verify the motivations behind adopting this technique over the central controller architecture.

4. Proposed Scheme
4.1. Mininet
is a software simulator used for the development of simple SDN, consisting from OpenFlow controller [12], a flat Ethernet network of multiple OpenFlow switches, and hosts. Mininet contains a number of built-in functions that support the utilization of different types of switches and controllers. Moreover, Mininet python APIs could be used to construct complex customized scenarios. Several emulators such
as FIRE, GENI [13], VINI and Emulab are deployed with the important feature of being able to provide a global

4.2 POX
Is a software platform for open source applications based on Python [14] which developed faster than the NOX. The key benefit behind the use of POX controller is the ability to build network applications using python-based controllers, in addition, POX controller comes with several advantages like the ease to use, understand, and build applications. stock components bundled within the POX controller triggered the immediate employment of POX as a simple SDN controller. The core reason behind the selection of the POX controller among other controllers relied on the fact that various controllers included different needs, RYU [15] an SDN controller based on Python but is incapable of establishing inter-controller communications which in other words doesn’t support the design of multi-controllers and reduce the possibility of being efficient in the future. ONOS [16] is one of the controllers that support the distributed architecture but with a great deal of dependencies. Moreover, ONOS is still under the development phase and the processes of developing applications on the top of it found to be a difficult task. Floodlight one of the most popular SDN controllers with a Java-based API's that cannot be built within Python.

4.3. OpenFlow Switch
open-sourced, virtual switch program mainly developed for virtual servers and offers support to modern switching chipsets to high Fan-out switches which leads to offering the same flexibility of the physical infrastructure. Open switch is commonly utilized as the standard SDN switch that relies on the OpenFlow protocol as the keyway to manage the transmission process. OpenFlow Switch contains a user mode and kernel mode, the kernel-mode which is also known as the data path module receives the packets first and sends the packet to the user-mode ovs-vswitched for instructions on how to manage the switch has been managed with the POX controller, reconfigured the OpenFlow switch to the switch module.

4.4. Distributed Internet Traffic Generator (D-ITG)
A distributed internet traffic generating tool which supports both IPV4 and IPV6 traffic generation [17]. D-ITG can replicate a suitable stochastic process for both PS (packet size) and random variables IDT (inter-departure time). In addition, this tool is used for the creation of multiple flows which affects in the performance evaluation process for the network. In D-ITG Two flow modes were introduced first mode is the Single Flow Mode where the ITGSend generate one traffic flow according to the given command-line options in which the flow is handled by a dedicated thread, while a different thread is used by the ITGRecv component to set up and organize the generation process on a separate channel. On the other hand, in Multi-Flow Mode the ITGSend allows the multi-flow mode to create multiple flows simultaneously, a single thread manages each flow in addition a single separate thread act as a master and a coordinator to others in order to generate the n flows. The script file must contain n lines where each line specifies the characteristics of a single flow.

4.5. Linear distributed POX Controller Designs
The first design is a linear single POX controller topology. A remote POX controller named C0 is used to setup the SDN architecture through the assistant of Miniedit GUI emulator S1, S2 and S3 represented the OpenFlow switches. h1 through h12 referred to network hosts. The remarkable and important feature of the single POX controller topology is that all of the OpenFlow switches communicated with a single POX controller c0 through the default listening port 6633, this port is used to establish the connection between first POX controller C0 and OpenFlow switches S1, S2 and S3 as shown in figure 1.

Second design is based on a linear double POX controller topology which mainly employed two POX controllers named C0 and C1, three OpenFlow switches named S1, S2 and S3 and twelve hosts named h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, h11 and h12. The significant attribute for this topology is the utilization of two remote POX controllers through two listening ports, port 6633 to establish a connection between first POX controller C0 and the OpenFlow switches S1 and S2, and port 6634 to
establish a connection between second POX controller C1 and the open Flow switches S2 and S3 as illustrated in Figure 2.

The third design relied on linear triple POX controller topology, consisted from three POX controllers named C0, C1 and C2 beside to three OpenFlow switches named S1, S2 and S3 and twelve hosts named h1 to h12. This topology mainly relied on the utilization of three remote POX controllers through the listening ports 6633, 6634 and 6635. Port 6633 is utilized to establish a connection between first POX controller C0 and OpenFlow switch S1, similarly, a connection is established between second POX controller C1 with OpenFlow switch S2 and third POX controller C2 with open Flow switch S3 through ports 6634 and 6635 respectively as shown in figure 3.

Our fourth design is the linear quad POX controllers based on four remote POX controllers named C0, C1, C2 and C3 in with three OpenFlow switches named S1, S2, S3, S4 and hosts named h1 to h12 were utilized. This topology mainly relied on the utilization of four remote POX controllers through the listening ports 6633, 6634, 6635 and 6636. Ports are allocated in a similar way to the third design with one exception, port 6636 was used to establish a connection between the forth POX controller and OpenFlow switch S4 as was shown in Figure 4.
5. Experimental Setup and Evaluation

Through the use of Mininet network emulator, experimental “Testbed” was setup. POX controller was used to constructing the SDN topology and to model the SDN network. In order to evaluate the performance for the single as well as multi-controller SDN architectures, four separate topologies were established through the utilization of Miniedit GUI emulator (Miniedit is the graphical simulation platform of the Mininet) to build the GUI for those topologies. the OpenFlow protocol was setup through the use of OpenFlow switches. Furthermore, Network traffic was mainly generated through the use of D-ITG tool into which several network performance parameters such as throughput, average delay, average load jitter, average packet rate and bytes received were measured and analysed in order to present the influence of introducing additional POX controllers on these parameters in SDN environment. The D-ITG includes two flow models, the single flow model and the multi-flow model. In this research, the D-ITG multi-flow model was employed to generate traffic streams simultaneously. Multiple packets were sent from the sender h1 to the receiver h12 in a parallel form to verify the reaction of specific topology to traffic management in the network. Furthermore, D-ITG multi-flow model relied on a script file to flood the capacity of the OpenFlow switches. The script file relied on the configuration of multiple traffic flows; each line in this file represented the traffic flow as a set of command-line options similar to the single flow model. Furthermore, the script file mainly based on TCP as the communication protocol between the sender and the receiver. The script file termed script_file1 included nine flows, listed from flow1 up to flow9, each flow included a particular number of packets with packet
size which can be adjusted as needed, receiver listening port and finally the communication protocol responsible for establishing a connection between the sender h1 and the receiver h12, TCP protocol was our preferred choice as there will be no dropped packets or packet loss. Details and the operation mechanism of the script_file1 were explained further in Table-1.

| Flow No. | Function |
|----------|----------|
| Flow -1  | Send 1000 packets, each packet size is 512 byte using port 10001 based on TCP protocol |
| Flow -2  | Send 2000 packets, each packet size is 512 byte using port 10002 based on TCP protocol |
| Flow -3  | Send 3000 packets, each packet size is 512 byte using port 10003 based on TCP protocol |
| Flow -4  | Send 4000 packets, each packet size is 512 byte using port 10004 based on TCP protocol |
| Flow -5  | Send 5000 packets, each packet size is 512 byte using port 10005 based on TCP protocol |
| Flow -6  | Send 6000 packets, each packet size is 512 byte using port 10006 based on TCP protocol |
| Flow -7  | Send 7000 packets, each packet size is 512 byte using port 10007 based on TCP protocol |
| Flow -8  | Send 8000 packets, each packet size is 512 byte using port 10008 based on TCP protocol |
| Flow -9  | Send 9000 packets, each packet size is 512 byte using port 10009 based on TCP protocol |

6. Results

Figure 5. Average Delay Vs. Packets No.

Figure 6. Average Jitter Vs. Packets No.
After setting up and implementing the proposed network designs, critical parameters affecting the network traffic were measured. The average server delay was calculated in order to emphasize the behaviour and differentiate between the single, double, triple and quad POX controllers network performance. The average delay is considered as the most crucial factor when the incoming traffic is processed due to the fact that too much delay cannot be tolerated. Therefore, if the delay exceeds the tolerance limit then eventually the network will not be efficient anymore. Moreover, the more the delay the less reliable network can be attained. Consequently, the average delay was addressed by sending multiple packets simultaneously. It can be seen from Figure 5. that the average delay becomes less as the number of controllers in the design of SDN is increased. This event is due to the good load balancing approach of SDN. Additional controllers in the SDN model causes extra load to be distributed dynamically among controllers and results in low average delay with respect to transfer size. The average server delay of the quad POX controller recorded the best performance in terms of average server delay of 0.212 msec compared to 0.303, 0.4 and 0.0599 msec for single, double and triple POX controllers architecture. Average jitter was the other parameter that has been investigated and determined. As the load in our network can be inconsistent and may reach a maximum level at any time, then average jitter was computed in order to demonstrate the topology reaction. After running the network several times, the average value of jitter was calculated and recorded for a different number of packets. As was shown in figure 6., the average jitter for the quad POX controllers design reported the
lowest average jitter of 0.0127 msec, while single, double, triple and POX controller designs reported 0.309, 0.02891, 0.017533 and 0.0127 msec respectively. It was clear from figure 7., as the load was divided between the quad controllers functioning in the network which will communicate and push the necessary forwarding rules towards the OpenFlow switches, that Quad POX controllers design demonstrated a massive increase regarding average throughput 7664 (Kbit/sec), While the average throughput expressed by Single, double, triple POX controllers was 5808, 6057 and 6450 (Kbit/sec) respectively. As shown from Figures 8,9., an improvement in the average packet rate per second 1870 (Pkts/sec) as well as received bytes 9573 Kbyte was confirmed for the quad POX controllers architecture in comparison with other designs.

7. Conclusion
The performance evaluation of centralized and distributed controller architectures was carried out. Results demonstrated the impact of extra controllers on network performance, besides to which design demonstrated the preeminent results among the others. there was an improvement in the performance level correspondingly with the number controllers, the fourth design represented by the quad POX controller handled the load in a better way than others, scoring a reduction in The average delay and, average jitter to 62% and 64%. also there was an improvement regarding throughput, bytes received, average packet rate to 32%, 32% and 31.8%.

The key behind this achievement was due to fact that, while preserving the same number of OpenFlow switches and increasing the number of controllers then flow installation time (path provisioning) tend to decline, thereby more rules will be pushed from the controllers to the OpenFlow switches and eventually this leads to load being able to be divided among them.

8. References
[1] Shaghaghi A, Kaafar M A, Buyya R, and Jha S 2020 “Software-Defined Network (SDN) Data Plane Security: Issues, Solutions, and Future Directions” in Handbook of Computer Networks and Cyber Security Springer pp 341–387
[2] Ferguson G.P, Tung, Chiu S S, and Pederson D R 02-Jan-2020 “Methods and system for optimal multi-environment data synchronization/replication.” Google Patents
[3] Flauzac O, Robledo E M G, and Nolot F 2019 “Is Mininet the Right Solution for an SDN Testbed” IEEE Global Communications Conf.(GLOBECOM) pp 1–6
[4] Manuel, T and Goswami, B H 2019 “Experimenting with scalability of beacon controller in software defined network”. Int. Journal of Recent Technology and Engineering 7(5S2) pp 550-555.
[5] Paliwal M, Shrimankar M, and Tembhurne O 2018 “Controllers in SDN: A review report,” IEEE Access 6 pp. 36256–36270
[6] Fu Y, Bi J, Gao K, Chen Z, Wu J and Hao B 2014 “Orion: A hybrid hierarchical control plane of software-defined networking for large-scale networks,” IEEE 22nd International Conf. on Network Protocols pp 569–576.
[7] Huang S, Zhao J, and Wang X 2016 “HybridFlow: A lightweight control plane for hybrid SDN in enterprise networks” IEEE/ACM 24th International Sym. on Quality of Service (IWQoS) pp 1–2
[8] Xiao L, Xiang S, and Zhuy H 2018 “Modeling and verifying SDN with multiple controllers” in Proceedings of the 33rd Annual ACM Sym. on Applied Computing pp 419–422
[9] Sameer M and Goswami B 2018 “Experimenting with ONOS scalability on software defined network” J. Adv. Res. Dyn. Control Syst. 10 pp 1820–1830.
[10] Bannour F, Souihi S, Mellouk A 2018 “Distributed SDN control: Survey, taxonomy, and challenges” IEEE Commun. Surv. Tutorials 20 pp 333–354.
[11] Hohlfeld O, Kempf J, Reisslein M, Schmid S, and Shah N 2018 “Scalability Issues and Solutions for Software Defined Networks,” IEEE Journal on selected areas in communications 36
[12] Kumar D and Sood M 2020 “Analysis of Impact of Network Topologies on Network Performance in SDN” in International Conf. on Innovative Computing and Communications pp 357–369.

[13] Sydney A, Ochs D S, Scoglio C, Gruenbacher D, and Miller R 2014 “Using GENI for experimental evaluation of software defined networking in smart grids,” Comput. Networks 63 pp 5–16

[14] Shi X et al 2020 “An OpenFlow-Based Load Balancing Strategy in SDN,” C. Mater. Contin., 62 pp 38520–398.

[15] Asadollahi S, Goswami B, Sameer M 2018 “Ryu controller’s scalability experiment on software defined networks” IEEE International Conf. on Current Trends in Advanced Computing (ICCTAC) pp 1–5.

[16] Vizarreta P, Trivedi K, Helvik B, Heegaard P, Kellerer W, and Machuca C M 2017 “An empirical study of software reliability in SDN controllers” in 2017 13th International Conf. on Network and Service Management (CNSM) pp 1–9

[17] Botta A, de Donato W, Dainotti A, Avallone S, Pescapé A 2019 “D-ITG § VERSION § Manual”