Delta University Scientific Journal
Journal home page: https://dusj.journals.ekb.eg

Performance Analysis in Software Defined Network (SDN) Multi-Controllers

Mohamed M. Elmoslemany [1]*, Adly S. Tag Eldien [2], Mazen M. Selim [3]

[1]Department of Computer Science Faculty of Computer & Artificial Intelligence, Benha University, Cairo, Egypt, m_elmoslemany@mans.edu.eg.
[2]Department of Electrical Engineering Faculty of Engineering at Shoubra, Benha University, Cairo, Egypt, adlytag@feng.bu.edu.eg, Benha University.
[3]Department of Computer Science. Faculty of Computer & Artificial Intelligence, Benha University, Cairo, Egypt, Selimm@bu.bu.edu.eg, Selimm@deltauniv.edu.eg.

ABSTRACT
Software-defined networking grants many advantages for networking via separating the control plane from the data plane. The Controller is the central and the single point of managing the whole network and responsible for decision-making in the SDN controller. However, using a single controller in the SDN model has a problem in reliability, scalability, and security which causes the SDN controller to be a single point of failure. This paper presents a structure which resolved this problem via building high availability controller (HAC) architecture for the control plane. OpenDayLight (ODL) controller was used as a point of study for making the clustering. D-ITG (Distributed Internet Traffic Generator) was used for benchmarking. It can perform and generate traffic at different layers and measure the important parameters that influence the network traffic and performance: Average delay, Average jitter, Average bitrate (Throughput), Packets dropped and others. Finally, we make a full comparison of the effects of these parameters on performance when using a single controller and a multi-controller.

Keywords: Software Defined Networks; High Availability; D-ITG; ODL; performance

1. Introduction

Internet traffic grows in a hurried way, due to different factors such as Higher Internet Users, the Proliferation of Devices and Connections, Video Services, social, cloud, big data, Mobility, The growth of data traffic on the web, the virtualization of services and others. This has resulted in the increase of large data centers with the purpose of treating all this information. However, traditional methods based on manual configuration of proprietary devices are cumbersome and error-prone, and they cannot fully utilize the capability of physical network infrastructure. Toward acknowledgment of the problem of complicated managed networks, a collection of industries such as researchers and appliance vendor's work together to develop the weaknesses in the current network. SDN architectures separate network control from forwarding functions, so it's simplified the network structure and solved interruption in a traditional network. SDN allows making network configuration programmatically to improve network management and meet our needs. The SDN architecture principal characteristics are, the control plane and the data plane are separated, forwarding decisions are flow-based, a flow means a sequence of packets from one.

point to another, the logic control is moved to an external entity This external entity is called SDN controller which makes forward decisions and install instructions on switches or routers, SDN networks are highly programmable through applications which run on top of the control plane. So SDN architecture centrally managed, agile, scalable, and directly programmable. OpenFlow is a widely used communication protocol. It's responsible for the communication process between the control plane and data plane. It's one of the first (SDNs) that defines the standard interface. It's managed by the Open Networking Foundation (ONF). OpenFlow enables

the SDN Controller to directly interact with the forwarding plane of network devices such as switches and routers both in physical and virtual devices. SDN centralizes network intelligence in one network part called a controller, it's the brain of the SDN network, which has a global view of the network. In SDN we have a single controller for managing the whole network which causes a single point of failure and decreases the availability of the network. These failures are due to several reasons: failure of the server where a controller is running, the server operating system failure, power outage, damaged termination of the controller process, network application failure, network attacks on the controller and many others. The controller becomes an obstacle to the smooth provision of service. If the controller itself fails, the switch that it is managed cannot be controlled. Moreover, the forwarding decisions are dependent directly on the controller. Once the SDN controller or the switches-to-controller links fail, the entire network may collapse. As a result, a single controller may become the bottleneck of an SDN. Therefore, to avoid such a single point of controller congestion/failure, the control plane is typically implemented as a distributed system with a cluster of controllers. Therefore, if one controller failed, the other controller in the cluster can take control and manage the whole network without any downtime. Clustering is a mechanism that allows multiple programs and processes to work simultaneously as one object. For example, when browsing any web site like yahoo.com and browse for something, you may think your search request is processed by a single web server, in fact the search request is handled and processed by a lot of web servers that are connected in a cluster. There are many objectives for creating a cluster such as High Availability, which avoids a single point of failure and ensures. reliability and doesn't miss any data when multiple controllers are running and one of them crashed. In this paper, we will choose OpenDayLight Controller for setting up clustering by using multiple controllers and testing some parameters that affect the performance. We also use D-ITG (Distributed Internet Traffic Generator) which is an application that able to perform and generate traffic at different layers such as the Network layer, Transport layer, and Application layer. Finally, we test the performance in the case of single and multi-controller and make full comparison about it.

The layout of this paper is organized as follows. Section 2, the previous works on comparison of SDN controllers. Section 3 describes the clustering in OpenDayLight controllers. Section 4, Research Methodology. Lastly, Section 5, concludes the paper.

2. Literature review

In this section, there are several works which have been done. We introduce and review notes of existing works relating to the evaluation of controller performance. D. Levin, A. Wundsam, analyzes the performance of network applications that operate on a distributed control platform. Network applications that are aware of the physical decentralization showed better performance than applications that assume a single network-wide controller. B. Heller, R. Sherwood, discuss the effect of latency for control communication in OpenFlow-based networks to satisfy these latency restrictions, they propose several required controllers with their position in the network. A. Tootoonchian and Y. Ganjali, the authors propose a control plane that is based on the NOX OpenFlow controller. They consider the administration of network applications and the consistent view of the framework in detail. such as, when a connection breaks, one controller determines the failure, still another controller may not be informed of the connection failure. The HyperFlow architecture ensures in such cases that network applications operate in a consistent state of the network, even though control elements may not share identical knowledge about the network. S. H. Yeganeh and Y. Ganjali, Kandoo is proposed in, which manages the controllers in a layer structure as there are some lower and higher layers are built. Controllers in lower layers process local network events and program the local portions of the network under their control. Controllers on higher layers sort network-wide decisions. They direct and query the local controllers at lower layers. Another proposed in was implementing multiple controllers, for installing replicated controllers to eliminate the single point of failure. Although, this process has many problems such as, using passive controllers that will be active, only in case the main controller fails. While a logically centralized architecture, all the controllers have the same responsibilities, and they split the charge equally. They are always aware of every change in the network, and they share the same information instantly.

3. Clustering in OpenDayLight Controllers

3.1. OpenDayLight

is a collaborative open-source project which develops quickly because of the immense contributions of the open-source community. It is hosted by The Linux Foundation. It began in April 2013, but it was published in February 2014. It has overcome "vendor locking" and supported more protocols than OpenFlow. ODL is a Java-based controller which is based on Beacon Controller design. The main objective of ODL is to provide centralized

management to have a programmable and intelligent network by using API frameworks. Many vendors worked together in this project to make a reliable and efficient controller for large scale networks of any size and scale and to support network services across a combination of hardware in multivendor environments. Fig 1 shows, It has 3 different separated layers: The top layer provides controller services and common REST APIs, The Middle layer is the Controller platform that communicates with the underlying network by using the southbound plug-ins, the bottom layer is the southbound interface which includes all the protocols for management and controls the networking infrastructure such as OpenFlow, NETCONF, SNMP, and Others. OpenDayLight uses software tools in design and configurations such as: Java interfaces, Maven, OSGi, and Karaf.

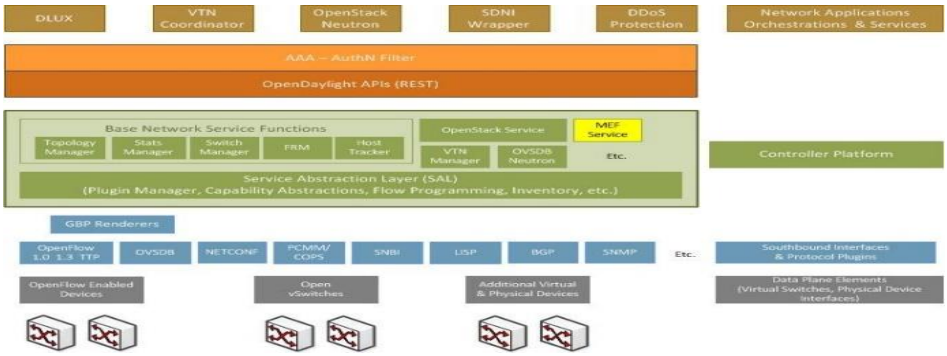

Figure 1: OpenDayLight Architecture

3.2 Clustering in OpenDayLight

Our goal is to have a set of nodes to provide a fault-tolerant with no single point of failure.
The cluster implementation architecture has 2 subsystems:

I. Data Store: It is used to allow high availability and scalability where all the members are talking to each other to distribute the data. It contains 2 modules: Config data store and operational data store.
II. Rpc connector: It is responsible for routing RPC requests to their implementations and routing.

3.4 Cluster Startup

When the cluster starts up, it creates an instant method of the distributed data store and waits until it gets ready. The waiting time to get ready is usually 90 seconds in which it tries to find the leader. At the same time, it tries to start another node to select a leader. If this step is done within 90 seconds, it will move to the next step. If not, it will block for 90 seconds.
When it creates a distributed data store, it creates two classes:

I. ACTOR CONTEXT: Which allows communicating with actors which is necessary to distribute data store.
II. SHARED MANAGER: which is the parent of all the shares. The shares are created at a configuration file called (module - shared.conf), so we needed to locate the leader within the 90 seconds. When shares are created, they first read and recover the information from the disk from each journal or the snapshot. First, it reconstructs the data tree, then sets its behavior to the follower. After that, it says "I am ready for communications". Finally, it continues with the elections process to choose the leader. Once the leader is selected, the countdown time starts and waits until it gets ready.

4. Research Methodology

This section explains the configuration, implementation, Tools used, test environment, parameters, and the output results to deploy a cluster architecture with the OpenDayLight controller.

4.1. configuration

We have multiple instances of the controller working together as a single entity, to perform high availability. There are some considerations which must be taken before working with the cluster:

- To build the cluster we must have an odd number of controllers because OpenDayLight is using the Raft algorithm and needs most of the members to be available. So, the minimum cluster size to maintain the HA feature is 3 nodes according to use the following equation: Tolerates: nodes=2f+1,

where (f) is number of failures. For example: when we use (3) nodes => we have (1) failure, for use (5) we have (2) failure, etc. If we use (2) nodes we never have high availability.

- The role for every node in the cluster.
- The data needed for the system which allocated among shares such as (Inventory, Topology, Toaster, and a default). If a node is unreachable, it remains down for configuration for some time (10 seconds, by default). Once the node goes down, it requires a restart it to rejoin the cluster. Once a node restarted, it joins the cluster and will automatically synchronize with the leader node.

4.2. Tools Used

I. Mininet: It is an open-source project. It's possibly more accurately a network emulation orchestration system. It simulates a collection of routers, end-hosts, switches, and links on a single Linux kernel. It uses virtual interfaces linked with virtual cables, to make a single system look like a complete network. It uses lightweight virtualization which is the capability of an operating system to be installed directly on the server hardware and provides the functionality to create Virtual Servers. We used it to make a complete network. It is commonly used in simulation, verification, testing tools, and resources [14].

II. D-ITG: D-ITG (Distributed Internet Traffic Generator) is an application that able to perform and generate traffic at different layers such as the Network layer, Transport layer, and Application layer. It supports a large-scale number of protocols like TCP, UDP, ICMP, SCTP, DCCP, ICMP, Telnet, and VoIP. This tool is used as Network Measurement Tools to measures and captures the well-known performance metrics such as round-trip-time (RTT), delay, jitters, throughput, one-way delay (OWD), and packet loss. It is also able to generate different packet streams and collect statistics with a logging server. This is done by using the two core features of D-ITG named as ITG-Send and ITG-Receive. D-ITG is designed to run on both Windows and Linux platform. Figure 5.11 describes all the major modules of the D-ITG are described as ITGSend, ITGRecv, ITGLog, and ITGDec. ITGSend is responsible for generating traffic to ITGRecv, using a multithreaded design. ITGSend able to send Single-flow, multiple, Daemon parallel traffic flows to multiple ITGRecv instances. ITGRecv able to receive multiple parallel traffic flows from multiple ITGSend instances, so there is a generated signaling channel between each pair of ITGSend and ITGRecv components to control the generation of all the traffic flows between them. The ITGLog is a storage to collect all log files, it can receive and storing detailed log information that may be sent by ITGSend or ITGRecv. This Log information is received over TCP or UDP protocols on port numbers from range [9003–10003]. Finally, the ITGDec is responsible for analyses the results collected by the ITGLog module. It is decoding and investigating the log files which is stored during the investigations that carried by using D-ITG. ITGDec parses the log files generated by ITGSend and ITGRecv and measures the average values.

4.3. Tools Used

The test environment is done on a virtual environment by using (VMWare). The test is done by using 4 virtual machines: (3) For the controller and (1) for the emulator tool (Mininet). The operating system used for Controllers virtual machines is Ubuntu 16.04 LTS-64 bit and for Mininet Ubuntu 14.04.4 LTS-64 bit all the machined connected across Gigabit Ethernet Interface speed (1GB). We use the following specifications hardware for virtual machine in our implementation, with the following specs in table 1.

Table 1
Hardware Specifications

Node Name	**CPU**	**Memory**
Controller (1)	(Intel(R) Xeon 2.19 GHZ), 8Cores	8GB RAM
Controller (2)	(Intel(R) Xeon 2.19 GHZ), 8Cores	8GB RAM
Controller (3)	(Intel(R) Xeon 2.19 GHZ), 8Cores	8GB RAM
Mininet	(Intel(R) Core(TM)i7-4790CPU, 3.6GHZ)	8GB RAM

4. Cluster monitoring and testing

It is necessary to verify that the cluster setup is correct after we make configuration. This validation is to confirm that the cluster is running correctly. We can verify that there is a cluster leader for each shard and the shard replication working correctly and committing actions. We can do that by using the "Postman" application for making HTTP requests, to the controller asking for (Get) information about a specific shard. Also, in the OpenDayLight, we can also show the shard information through MBeans, which can be investigated by VisualVM, or JConsole, or different JMX clients, or REST API using Jolokia. Jolokia can be implemented by installing the odl-jolokia Karaf feature in ODL controller. In our research, we used “Postman" application and retrieve information in 3 controllers.


```
{"request":{"mbean":"org.opendaylight.controller:Category=Shards,name=member-1-shard-inventory-config,type
=DistributedConfigDatastore","type":"read"},"value":{"ReadWriteTransactionCount":0,"SnapshotIndex":30
,"InMemoryJournalLogSize":1,"ReplicatedToAllIndex":30,"Leader":"member-3-shard-inventory-config"
,"LastIndex":31,"RaftState":"Follower","LastApplied":31,"LastCommittedTransactionTime":"1970-01-01 02:00
:00.000","PeerAddresses":"member-2-shard-inventory-config: akka.tcp:\/\/opendaylight-cluster-data@192.168
.40.144:2550\/user\/shardmanager-config\/member-2-shard-inventory-config, member-3-shard-inventory-config
: akka.tcp:\/\/opendaylight-cluster-data@192.168.40.146:2550\/user\/shardmanager-config\/member-3-shard
-inventory-config","LastLeadershipChangeTime":"2020-05-17 12:41:10.616","LastLogIndex":31
,"FollowerInitialSyncStatus":true,"WriteOnlyTransactionCount":0,"FollowerInfo":[]
,"FailedReadTransactionsCount":0,"Voting":true,"StatRetrievalTime":"432.3 µs","CurrentTerm":1260
,"LastTerm":1000,"FailedTransactionsCount":0,"PendingTxCommitQueueSize":0,"VotedFor":"member-3-shard
-inventory-config","SnapshotCaptureInitiated":false,"CommittedTransactionsCount":0,"TxCohortCacheSize":0
,"PeerVotingStates":"member-2-shard-inventory-config: true, member-3-shard-inventory-config: true"
,"LastLogTerm":1000,"StatRetrievalError":null,"CommitIndex":31,"SnapshotTerm":1000
,"AbortTransactionsCount":0,"ReadOnlyTransactionCount":0,"ShardName":"member-1-shard-inventory-config"
,"LeadershipChangeCount":3,"InMemoryJournalDataSize":37},"timestamp":1589712192,"status":200}
```


• *In Member (1) we use the following command:*

GET

http://192.168.40.145:8181/jolokia/read/org.opendaylight.controller:Category=Shards,name=member-1-shard-inventory-config,type=DistributedConfigDatastore.

Here, the request returns information about the state of the cluster as shown in Figure. ٢

Figure 2: Member 1 json output of shard’s data after make Http get request.

• In Member (2) we use the following command:

GET http://192.168.40.144:8181/jolokia/read/org.opendaylight.co-ntroller:Category=Shards,name=member-2-shard-inventory-config,type=DistributedConfigDatastore.

Here, the request returns information about the state of the cluster as shown in figure. 3


```
{"request":{"mbean":"org.opendaylight.controller:Category=Shards,name=member-3-shard-inventory-config,type
=DistributedConfigDatastore","type":"read"},"value":{"ReadWriteTransactionCount":0,"SnapshotIndex":30
,"InMemoryJournalLogSize":1,"ReplicatedToAllIndex":30,"Leader":"member-3-shard-inventory-config"
,"LastIndex":31,"RaftState":"Leader","LastCommittedTransactionTime":"1970-01-01 02:00:00.000"
,"LastApplied":31,"LastLogIndex":31,"LastLeadershipChangeTime":"2020-05-17 12:41:10.547","PeerAddresses"
:"member-1-shard-inventory-config: akka.tcp://opendaylight-cluster-data@192.168.40.145:2550/user
/shardmanager-config/member-1-shard-inventory-config, member-2-shard-inventory-config: akka.tcp
://opendaylight-cluster-data@192.168.40.144:2550/user/shardmanager-config/member-2-shard-inventory
-config","WriteOnlyTransactionCount":0,"FollowerInitialSyncStatus":false,"FollowerInfo"
:[{"timeSinceLastActivity":"00:00:06.408","active":true,"matchIndex":31,"voting":true,"id":"member-1
-shard-inventory-config","nextIndex":32},{"timeSinceLastActivity":"00:00:00.109","active":true
,"matchIndex":31,"voting":true,"id":"member-2-shard-inventory-config","nextIndex":32}]
,"FailedReadTransactionsCount":0,"StatRetrievalTime":"30.48 ms","Voting":true,"CurrentTerm":1260
,"LastTerm":1000,"FailedTransactionsCount":0,"PendingTxCommitQueueSize":0,"VotedFor":"member-3-shard
-inventory-config","SnapshotCaptureInitiated":false,"CommittedTransactionsCount":0,"TxCohortCacheSize":0
,"PeerVotingStates":"member-1-shard-inventory-config: true, member-2-shard-inventory-config: true"
,"LastLogTerm":1000,"StatRetrievalError":null,"CommitIndex":31,"SnapshotTerm":1000
,"AbortTransactionsCount":0,"ReadOnlyTransactionCount":0,"ShardName":"member-3-shard-inventory-config"
,"LeadershipChangeCount":1,"InMemoryJournalDataSize":37},"timestamp":1589712149,"status":200}
```


Figure 3: Member 2 json output of shard’s data after make Http get request.

• In Member (3) we use the following command:

GET

http://192.168.40.146:8181/jolokia/read/org.openda ylight. controller: Category=Shards, name=member-3-shard-inventory

Here, the request returns information about the state of the cluster as shown in figure. 4

{"request":{"mbean":"org.opendaylight.controller:Category=Shards,name=member-3-shard-inventory-config,type
=DistributedConfigDatastore","type":"read"},"value":{"ReadWriteTransactionCount":0,"SnapshotIndex":30
,"InMemoryJournalLogSize":1,"ReplicatedToAllIndex":30,"Leader":"member-3-shard-inventory-config"
,"LastIndex":31,"RaftState":"Leader","LastCommittedTransactionTime":"1970-01-01 02:00:00.000"
,"LastApplied":31,"LastLogIndex":31,"LastLeadershipChangeTime":"2020-05-17 12:41:10.547","PeerAddresses"
:"member-1-shard-inventory-config: akka.tcp://opendaylight-cluster-data@192.168.40.145:2550/user
/shardmanager-config/member-1-shard-inventory-config, member-2-shard-inventory-config: akka.tcp
://opendaylight-cluster-data@192.168.40.144:2550/user/shardmanager-config/member-2-shard-inventory
-config","WriteOnlyTransactionCount":0,"FollowerInitialSyncStatus":false,"FollowerInfo"
:[{"timeSinceLastActivity":"00:00:06.408","active":true,"matchIndex":31,"voting":true,"id":"member-1
-shard-inventory-config","nextIndex":32},{"timeSinceLastActivity":"00:00:00.109","active":true
,"matchIndex":31,"voting":true,"id":"member-2-shard-inventory-config","nextIndex":32}]
,"FailedReadTransactionsCount":0,"StatRetrievalTime":"30.48 ms","Voting":true,"CurrentTerm":1260
,"LastTerm":1000,"FailedTransactionsCount":0,"PendingTxCommitQueueSize":0,"VotedFor":"member-3-shard
-inventory-config","SnapshotCaptureInitiated":false,"CommittedTransactionsCount":0,"TxCohortCacheSize":0
,"PeerVotingStates":"member-1-shard-inventory-config: true, member-2-shard-inventory-config: true"
,"LastLogTerm":1000,"StatRetrievalError":null,"CommitIndex":31,"SnapshotTerm":1000
,"AbortTransactionsCount":0,"ReadOnlyTransactionCount":0,"ShardName":"member-3-shard-inventory-config"
,"LeadershipChangeCount":1,"InMemoryJournalDataSize":37},"timestamp":1589712149,"status":200}

Figure 4: Member 3 json output of shard's data after make Http get request.

5. Network Scenario

- The Controller used in this experiment single or multi-controller is ODL Controller. The Network topology is used in Mininet as in fig. 5 The default controller in Mininet is Pox, so we must change to ODL by using this command, where n is the switch number and must run this command for all switches: Sudo ovs-vsctl set-controller s1 tcp:192.168.40.144:6633.
- The D-ITG was used as generation tools: the host (10.0.0.1) as the source and (10.0.0.40) as the destination host.
- 2 flows were generated with different number of packets as follow: (٤000, ٨000, ١٢000, ١٦000, ٢٠000, ٢٤000, and ٢٨000) Packets, Using UDP protocol. The duration inflows generated is (10000 milliseconds). The size of each packet used is equal to 1024. TABLE II shows the parameters that used for evaluation.

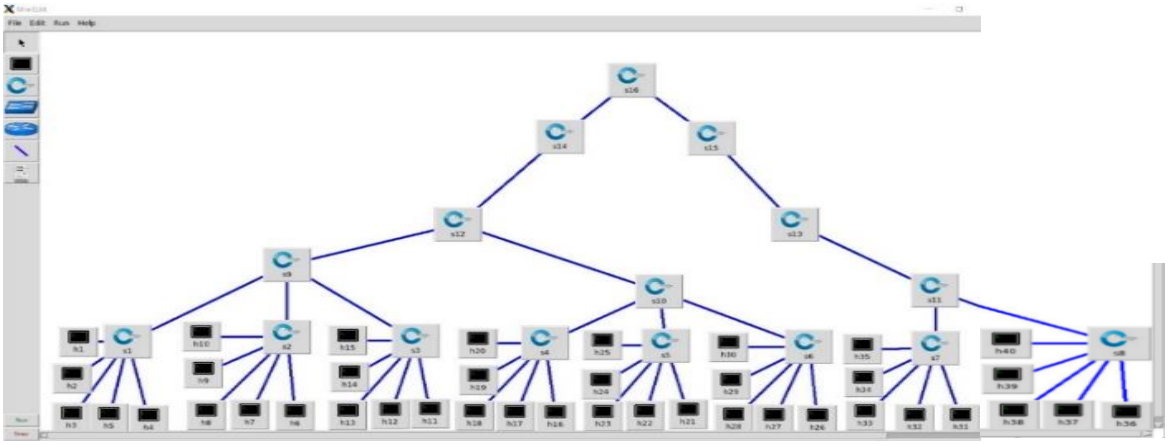

Figure 5: Our Topology

6. Result and Discission

I Minimum delay

II Average delay: The delay is a crucial factor for managing the incoming traffic in any networking scenario, as we cannot afford any delay.

III Average jitter: It's used to handle the load effectively. The load in any network can be changeable and can reach the peak at any time.

IV Bytes received.

V Average bitrate (Throughput): Measure the amount of data transferred in the investigated network in kbit/sec.

VI Packets dropped.

Table 2

Parameters used for Evaluation.

Items	Value
Controller Type	OpendayLight (ODL)
Number of Cluster Node	3
Controller IP	192.168.40.144,192.168.40.145 and 192.168.40.146
OpenFlow Version	1.3
Packet Type	UDP
Connection port	6633
Number of connected switch	16 OpenFlow Switch
Number of connected host	40
Chanel Type	TLS
Packet Size (KB)	1500
Packets sent	4000, 8000, 12000, 16000, 20000, 24000, 28000

7. Performance analysis result in single and multi-controllers and our observation.

The performance of these topologies was compared and analyzed. We measured the results and drew graphs to know which topology is more dependable. Finally, the result proved that the multi-controller technology can manage and handle the load in a more reliable way than single controller.

7.1 Minimum delay

The minimum delay for the single controller topology is extremely too high when the load is increased with sending more packets/sec. On sending 28000 packets/sec, it reached 0.00107 sec, on the other hand it reached 0.00086 sec on using multi-controller. So, this graph proves that the minimum delay is extremely high in single than in working with multi-controller as shown in Table 3, and figure. 6.

Table 3

Minimum delay for single and multi-cont.

No. of Packet	4000	8000	12000	16000	20000	24000	28000
Single Cont.	0.508	0.638	0.712	0.732	0.817	0.733	1.07
Multi-Cont.	0.318	0.388	0.538	0.608	0.721	0.533	0.86

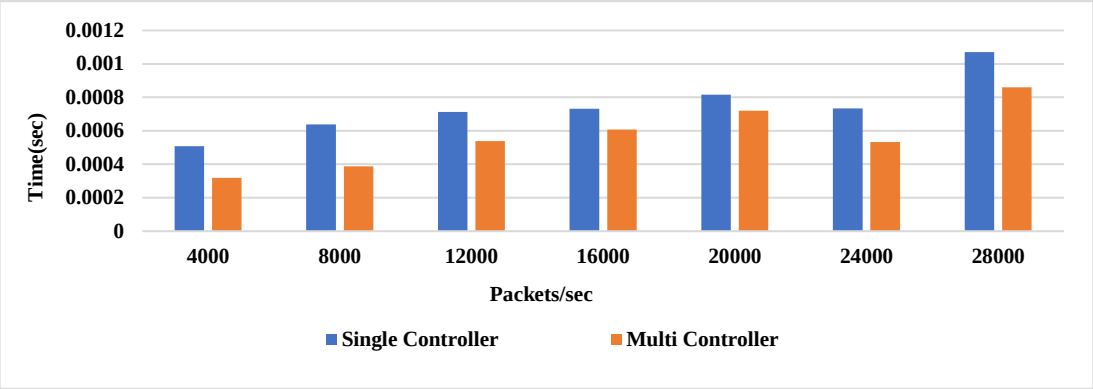

Figure 6: Minimum delay for Single Controller and Multi-Controller

7.2 Average delay

The delay in networking is the crucial parameter and affects the network reliability. We must have a minimum delay to ensure the network works in a stable way, overcomes any packet delay, and have the lowest latency. In the figure 7 and table 4, on sending simultaneous packets flows, we observed the Avg. delay/sec is too high in all single controllers compared with multi-controller. When we select 28000 packets/sec from the figure below, we noticed that the Avg. delay is 0.02745 in single controller compared to 0.010274 in multi-controller. We also observed the Avg. delay is increased and higher than expected in single controller. On the other hand, the delay isn't affected, and the change is not too much in multi-controller. This change s within the acceptable range and be in the accepted range: between: (0.004257 to 0.010274) sec, and in a single controller between :(0.010859 to 0.02745).

7.3 Average jitter

Jitter has an impact on network performance. We calculated the rate of average jitter. The results indicate and confirm that the topology runs-in multi-controller architecture can manage the load in a more reliable way compared to a single one-controller. We proved that using multi-controllers can manage the load with the minimum issue. Table 5 and Fig. 8, show that when sending 20000 packet the average jitter is 0.000838 for single compared to 0.000441 for multi-controller.

Table 4

Average delay for Single and Multi-Cont

No. of Packet	4000	8000	12000	16000	20000	24000	28000
Single Cont.	10.859	15.81	20.05	22.631	24.378	21.019	27.45
Multi-Cont.	4.257	5.278	4.823	5.887	6.571	6.575	10.274

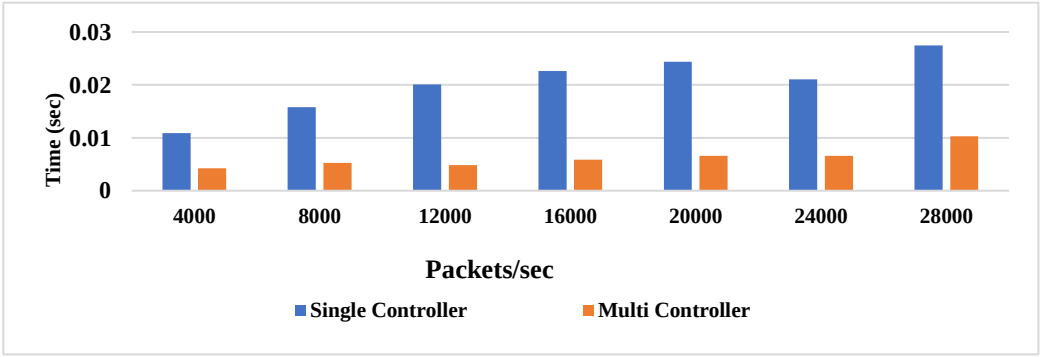

Figure 7: Average delay for Single Controller and Multi-Controller

Table 5

Average jitter for single and multi-cont.

No. of Packet	4000	8000	12000	16000	20000	24000	28000
Single Cont.	0.479	0.812	0.795	0.793	0.838	0.751	0.681
Multi-Cont.	0.283	0.389	0.437	0.495	0.441	0.498	0.586

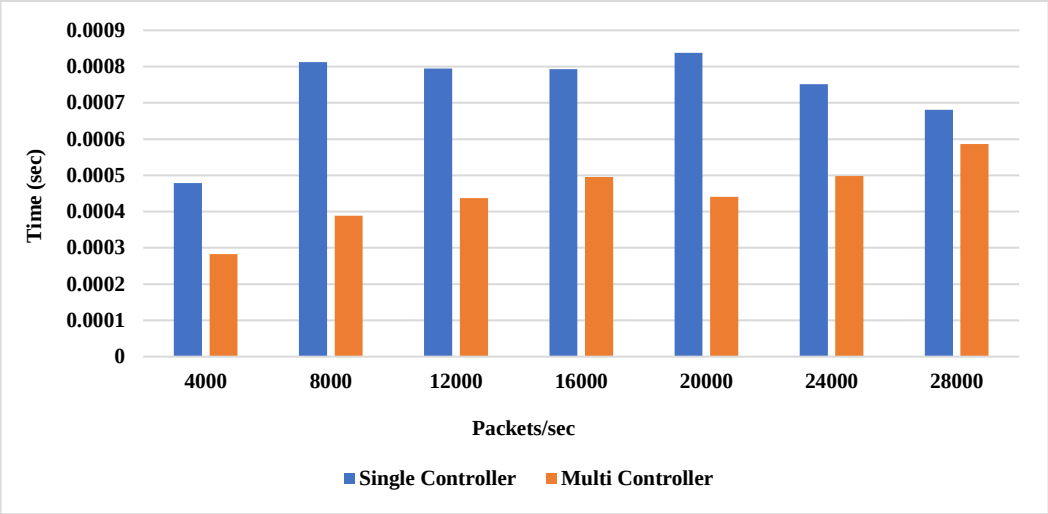

Figure 8: Average jitter for Single Cont. and Multi-Cont.

7.4 Bytes received.

Figure 9 and table 6, shows that the in multi-controller can received more packets compared to using single controller.

7.5 Average bitrate

This metric is also known as throughput, and it is an important parameter that influences network performance. We must obtain a high throughput. Fig. 10 shows that using multi-controller topology is more dependable and better than using the single one. When using multi-controller, the OpenFlow switches have connected to two or more controllers to communicate and push the rules, so the load is divided between these two or more controllers. According to our figure, we deduced that as the network grows, working with SDN architecture by multiple controllers is regularly more reliable and better than using a single controller. Fig. 10 and table 7, shows that when sending 28000 packets/sec the throughput in multi-controller is 128861.3049 Kbit/s compared to 97311.18874 Kbit/s in a single one.

Table 6

Bytes received for Single and Multi-Cont**.**

No. of Packet	4000	8000	12000	16000	20000	24000	28000
Single Cont.	692613	100824	105511	102602	102160	112374	122280
Multi-Cont.	970086	122331	134460	145287	135611	142529	153812

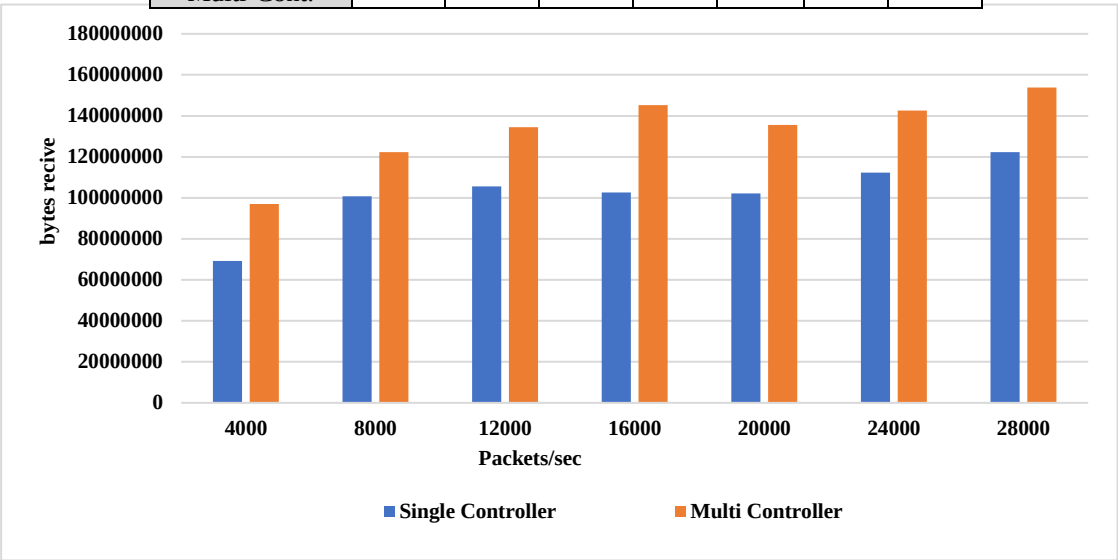

Figure 9: Bytes received for Single Cont. and Multi-Cont.

7.5 Packets dropped.

Fig. 11 and table 8, show that when sending continuous packets in a single controller, we have more packet loss due to the huge load in this single controller. On using a multi-controller, the load is divided between them. Fig. 11 shows that when sending a continuous flow of packets 20000, we have packet loss of 1581 in multi- controller compared to 6628 in single controller. This indicates that the number of packets loss in multi- controller compared to 6628 in single controller. This indicates that the number of packets loss in multi-controller is very small compared to the packet loss in a single controller.

Table 7

Average bitrate for single and multi-cont.

No. of Packet	4000	8000	12000	16000	20000	24000	28000
Single Cont.	55408	80215	83254	81592	72110	88540	97311
Multi-Cont.	77625	97888	107615	116890	108464	114090	128861

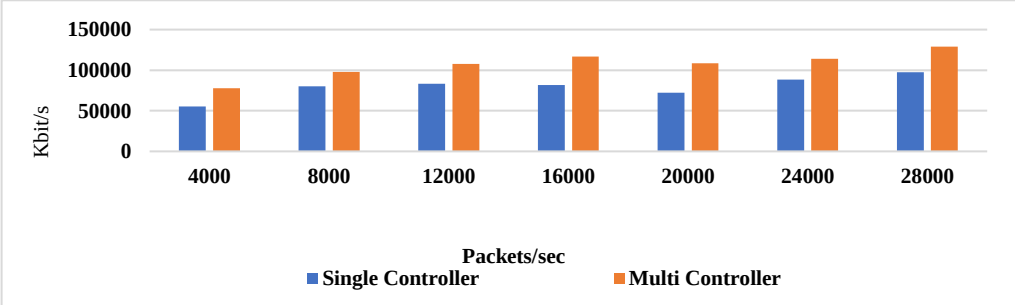

Figure 10: Average bitrates received for Single Cont. and Multi-Cont.

TABLE 8

Pactets dropped for single and Multi-cont.

No. of Packet	4000	8000	12000	16000	20000	24000	28000
Single Cont.	2027	3021	3869	4846	6628	5427	6966
Multi-Cont.	341	434	608	820	1581	1377	1274

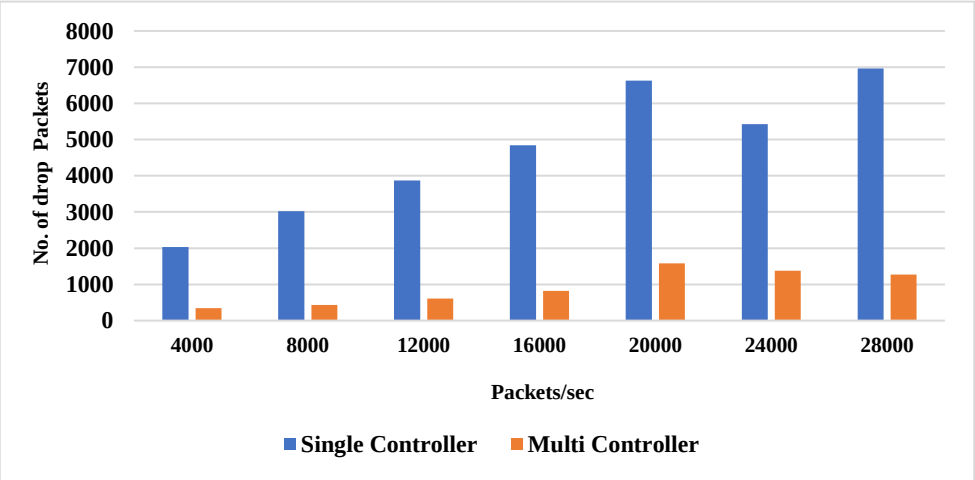

Figure 11: Packets dropped for Single Cont. and Multi-Cont.

Conclusion

Using a single controller may lead to a single point of failure. Therefore, the controllers need to be configured in distributed mode rather than a centralized one where the single failure affects the entire network performance. Moreover, load balancing and standby mechanisms can also be applied. Furthermore, we explained the effects of having a distributed SDN network and how to avoid the single point of failure problem. The distributed system achieves scalability, persistency of the information, and high availability. We focused on analyzing the behavior of the Open Daylight controller under the cluster architecture schema, how the controller and the cluster run, how it's communicating with other controllers and how the messages are transferred among the controllers. D-ITG was used as a generation tool to generate many different packets with different size, to measure the parameters that have affected the network performance such as, delay, throughput, jitter, packet dropped and others. Finally, a full comparison was made between using a single controller and multi-controller. Using multi-controller achieves high availability and makes the network more reliable..

References

Delta Journal reference style follows the uniform requirements for manuscripts which is based largely on an ANSI standard style adapted by the National Library of Medicine (NLM) for its databases https://dusj.journals.ekb.eg/ for a standard journal article:

A. Tootoonchian and Y. Ganjali, “Hyperflow: a distributed control plane for openflow,” Proc. 2010 internet Netw., pp. 1–6, 2010.

B. Heller, R. Sherwood, and N. Mckeown, “The Controller Placement Problem,” Proc. Second ACM SIGCOMM Work. Hot Top. Softw. Defin. networking., pp. 7–12, 2012.

C. Technical Report, "Forecast and Methodology, 2014-2019 White Paper.," Cisco Visual Networking Index, 2015.

D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized? State distribution trade-offs in software defined networks," HotSDN, pp. 1–6, 2012.

D. Erickson, "The Beacon OpenFlow controller," in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13), pp. 13–18, ACM, New York, NY, USA, 2013.

Esteban Hernandez, Maier Guido Alberto,"Implementation and Performance of a SDN Cluster-Controller Based on the OpenDayLight Framework", April 2016

E. M. V. R. P. E. V. C. E. R. S. A. S. U. Diego Kreutz, "Software-defined networking: A comprehensive survey," *IEEE*, 2015

https://en.wikipedia.org/wiki/Software-defined_networking#cite_note-23

https://nexus.opendaylight.org/content/sites/site/org.opendaylight.docs/master/userguide/manuals/userguide/bk-user-guide/content/_clustering_overview.html

https://github.com/mininet/mininet/wiki/Documentation

M. Raja, «MD-SAL Clustering Internals, Linux Foundation,» 2015. [Enlínea].Available:http://events.linuxfoundation.org/sites/events/files/slides/MD-SAL%20Clustering%20Internals.pdf

S. Avallone, S. Guadagno, D. Emma, A. Pescape, and G.Ventre, "D-ITG distributed Internet traffic generator", First International Conference on the Quantitative Evaluation of Systems, 2004.

S. H. Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," Proceeding HotSDN '12 Proc. first Work. Hot Top. Softw. Defin. networks, pp. 19–24, 2012.

"OpenDaylight: A Linux Foundation Collaborative Project." [Online].Available: https://www.opendaylight.org/

Yazici V, Sunay MO, Ercan AO. Controlling a software-defined network via distributed controllers. arXiv preprint arXiv:14017651. 2014