Software Defined Networking

Hend Abdelgader Eissa
Department of Computing and Informatics
Faculty of Electronic Technology
Tripoli, Libya
namarek2010@gmail.com

Kenz A. Bozed
Benghazi University , Faculty of Information Technology, Department of Computer System Design
Benghazi, Libya
kenz.bozed@uob.edu.ly

Hadil younis
Department of Computing and Informatics
Faculty of Electronic Technology
Tripoli, Libya
hadilyounes19@gmail.com

***Abstract*— This paper investigates the notion of software-defined networking (SDN), whose southbound interface can be applied through the OpenFlow protocol. The aim of this study is to discover the SDN architecture and the OpenFlow standard in some details. In addition, to look at implementing their tools on the oil company network. The simulation is done using mixing between a network emulator Mininet and Ryu controller with using python scripts.**

Keywords*— *Software-defined Networks; OpenFlow; Ryu Controller; Mininet

I. Introduction

Traditional networks architecture design depends on distribution control and transport network protocols running in the routers and switches that forward packets, allowing them to travel. This integration between the control plane and the data plane makes it difficult to manage the networks. [1]

SDN is a model framework that is dynamic, cost-effective, and adaptable, producing it good solution for the high-bandwidth, nature of implementation today's . This architecture separate the network control and forwarding functions, where it enables the network control to become programming's and the underlying infrastructure to be abstracted for network services. Open Flow protocol follows SDN and gives programmable control of flows to the network administrators allowing the controller to define the path the flow will take from source to destination whatever network topology, and utilizes flow based on processing for forwarding packets. Open Flow is a standard interface designed for SDN, and it gained great interest among developers and industrialist of network gear. The separation of the control and data planes turns the forwarding devices into dump switches, where the control logic is in a separate centralized controller or Network operating system. In SDN, one or more controller machines do a general-purpose program responding to events like the change in the network topology, connections establishment by end users, change in traffic load, by gathering a collection of packet-forwarding rules. The controllers pushes the rules to the switches through protocol such as Open Flow. The switches then implement their function efficiently using packet-processing hardware [1, 2].

The key contributions of this paper are:

- Study software-defined networking architecture and the principles reside behind its design.
- Study OpenFlow specification in details together with the Ryu controller and the specific exchanged messages between them.
- Use Ryu controller software environment to build a network application running with a network Mininet. The simulation demonstrates the advantages of using SDN architecture

II. SOFTWARE – DEFINED NETWORKING

Software-Defined Networking (SDN) is the physical separation of the network control plane from the forwarding plane, where a single control plane controls several devices. The SDN technique is for managing of traffic flows to be separated from the implicit infrastructure and systems that forward traffic.

SDN separates the control plane from the data plane, then SDN integrate the control plane, so that one control program controls more data plane elements [3], as laid out in Fig. 1.

The separation of the control plane and the data plane is defined as application programming interface (API) between the network device and the SDN controller OpenFlow protocol [4] is an example for an API. A switch with programmable interface enables the controller to communicate and set rules on the switch. OpenFlow switch can behave like a router, switch, firewall, network address translator [3].

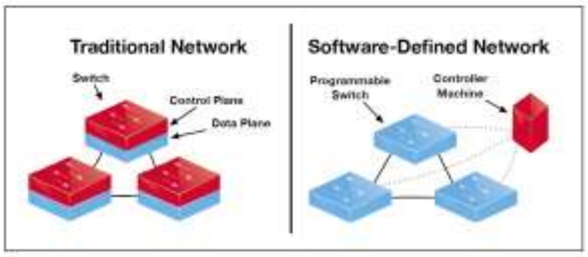


Fig. 1 Traditional and software-defined architectures

B. *The abstractions of SDN*

Abstractions are used to define the pertinent interfaces to form a modular scalable system. A modular system that allows to reuse of code. The implementation can be modified, but if the interface remains the same, it does not affect other parts of the software system. Abstractions have great benefits for building a scalable software system, modularity based on abstraction is required [5]. The SDN abstraction is similar to the traditional computer systems, which was integrated with proprietary operating system, Hardware and software into layered model with the ability to choose the suitable feature in each layer. SDN comes with the proper abstraction for the control plane. Three main columns are introduced to disconnect, the control plane:

- *Abstraction of the forwarding plane*: Abstracting the forwarding plane is hiding the complexity of its implementation from the control decisions. An open interface is used to rule the network devices. This means that there is no need to have concern about a specific vendor.

- *Abstraction of the network state*: The cause behind the complexity of managing and controlling the current networks is the complicated distribution algorithms such as OSPF. The idea is abstracting the complicated algorithms and come with a general network view for the controller to simplify the application functions. Instead of letting the network devices to communicate with each other, The SDN Controller uses specific protocol (e.g., OpenFlow) to communicate with network devices with information about the network to form the "view" or topology map. Configurations sent to the routers and switches to the forwarding depending.

- *Abstraction of the control plane*: The SDN controller provides application-programming interfaces that are accessed by applications. External applications can manipulate the network by the APIs through the controller using Java or REST. Developers can configure and control the network without having to write software to support multiple vendor hardware and software.

After implementing abstractions, the controller will work as a Network Operating System (NOS) and talk to the switches through API known as Southbound API. Where the applications are codes written in the controller, using API's provided by the NOS called Northbound API.

C. SDN Layers

The SDN architecture can be defined as a seven-layer mixture as shown in Fig.2 . Each layer has its own purposes. Some are provided continuously in the SDN architecture, such as the Southern API, network operating systems, the Northern Network API, and the network application. Others can only be introduced in certain arrangements, such as hypervisor - or programming languages [6, 7]. The following figure illustrates these layers.

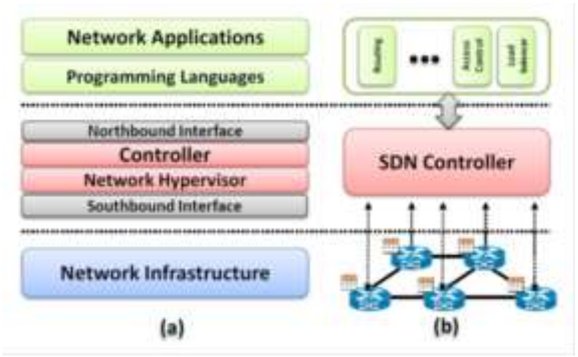


Fig. 2 Software-Defined Networks in (a) layers and (b) system design architecture

- *Infrastructure*: The traditional physical tools is became simple advancing fundamentals without insert control or for making decisions. The new networks are constructed on top of open and standard boundaries to guarantees compatibility and interoperability across different vendors. Moreover, the open interfaces enable controller entities to program heterogeneous forwarding devices, which is difficult in traditional networks [3].

- *Southbound Interfaces*: Southbound boundaries (SI) are the linking connections between control and network tools, the SI explain the communication procedure between advancing devices and control plane. This protocol establishes the method the control and information plane elements interact. On the other hand, these APIs are still secured to the advancing elements of the substructure physical or virtual infrastructure [6].

- Network Hypervisors: Network virtualization represents abstraction of a network that is separated from the underlying physical equipment. This to allow multiple virtual networks to run on top of shared infrastructure, where each virtual network can have a its topology, rather than the implicit physical network. Flow Visor was the initial effort to virtualize SDN. As revealed in Fig. 3, Flow Visor is a stage that acts as a substitution between the controller and the network tools to offer an concept layer that shares the OpenFlow data plane, permitting numerous controllers to control its own part. Flow Visor's main responsibilities are to decide who control the packets advanced by the switch, and check and establish the rules to be set by the controllers [6,8].

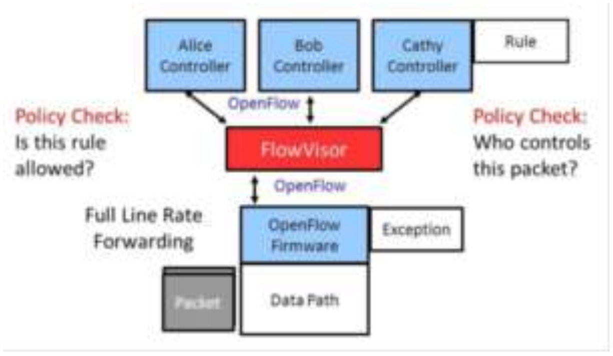


Fig. 3 Flow Visor message handling

- *Network Operating Systems*: NOS is the core element in SDN architecture. Similar to a current operating system, the controller abstracts the details of the SDN controller-to-device protocol where the applications above are able to communicate with those SDN devices without knowing the difference. This centralized control by NOS has to ease network management and simplify the overburden of solving networking problems. Controller core features include:

1) End-user device discovery: Discovery of end user devices such as laptops, desktops, mobile devices, and so on.

2) Network device discovery: Discovery of network devices that include the infrastructure of the network, such as switches, routers, and wireless access points.

3) Topology Management: Maintain information about the interconnection details of the network devices to each other and to the end-user.

4) Flow management: Maintain a database of the flows the managed by the controller and perform all necessary adjustment with the devices to ensure synchronization of the device flow entries with that database.

- *Northbound Interfaces:* The Northbound Interfaces are an abstraction that allow network applications not to depend on particular implementations to simplify programming the network. In reverse to the southbound interface, the northbound interface is mostly a software system, where applications like routing are built programmatically by programming languages such as Python or Java, this enables faster development, lower investment costs and easier troubleshoot compared to the Southbound API [9]. The controller notifies the application of proceedings that happen in the network. Events may concern an separate packet that has been established by the controller or state alteration in the topology, such as a connection going down. Applications implore different approaches in response to the event. This may comprise drop modify or forward the packet in case of an received packet event.
- *Programming languages*: Programming languages are high-level APIs that stretch an concept of the network itself, so that the developer need not be concerned with separate devices but slightly with the network as a whole [9]. Pyretic, Python and Frenetic amongst numerous other programming languages, are planned for SDN.
- *Network Applications*: Network applications implement the control-logic, which translated into, commends installed in the network devices, and determines their behavior. Network application is considered as the "Network brain". It registers as a listener for certain events as specified previously, then controller invokes the application's callback method whenever such an event occurs as well as apply them to External inputs such as performing security methods. [9]. A modest application, i.e., routing, the logic of this application is to describe the track through which packets will run from a point A to a point B. To accomplish this objective a routing application has to, based on the topology contribution, decide on the path to use and instruct the controller to connect the relevant advancing rules in all forwarding devices on the chosen path, from A to B [10]. Despite the wide diversity of usage cases, most SDN applications can be gathered in one of five classes: traffic engineering, mobility and wireless, measurement and monitoring, security and dependability and data center networking [6].

III. OPEN FLOW

Open Flow is the most widely establish open southbound API standard for SDN. It supply a common specification to implement Open Flow-enabled forwarding devices, and for the communication channel between data and control plane devices (e.g., switches and controllers). The OpenFlow protocol provides three information sources for network operating systems. First, event-based messages are sent by forwarding devices to the controller when a link or port change is produce. Second, flow statistics are generated by the forwarding devices and collected by the controller. Third, incoming packet messages are sent by forwarding devices to the controller when they do not know what to do with a new incoming flow or because there is an explicit "send to controller" action in the matched entry of the flow table. These information channels are the essential means to provide flow level information to the network operating system [9].

B. The OpenFlow Switch

An Open Flow Switch contains at least three parts: (1) A Flow Table, with an act related with each flow entry, to tell the shift how to practice the flow, (2) A Secure Channel that links the switch to a remote control process (called the controller), allowing instructions and packets to be sent between a controller and the switch using (3) The Open Flow Protocol, which offers an open and normal way for a controller to connect with a switch. Fig.4 shows an instance of an Open Flow switch [10].

The basic notion is simple: most contemporary Ethernet switches and routers comprise flow-tables that run at line-rate to device firewalls, NAT, QoS, and to gather figures. While each vendor's flow-table is different, there is an stimulating common set of purposes that run in many switches and routers. OpenFlow exploits this common set of purposes.

The basic idea is simple: most modern Ethernet switches and routers comprise flow-tables that run at line-rate to implement firewalls, NAT, QoS, and to gather figures. While each vendor's flow-table is different, there is an exciting common set of purposes that run in numerous switches and routers.

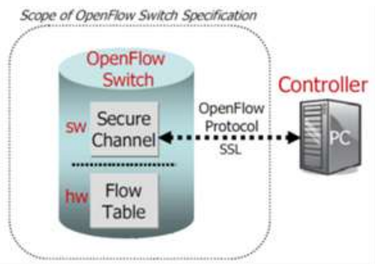


Fig. 4 Open Flow Switch

Open Flow quarries this public set of purposes. Open flow switches come in two diversities: pure (OpenFlow-only) switches, which have no legacy features or on-board control, and completely rely on a controller for forwarding decisions. And hybrid (OpenFlow-enabled) switches, which support OpenFlow in addition to outdated process and procedures. Most profitable changes available today are hybrids .

C. The OpenFlow Tables

An OpenFlow switch contains of a flow table, which achieves packet lookup and forwarding. Each flow table in the change holds a set of flow entries.

- *Flow entry*: A flow table contains of flow entries. Each flow access consists of :

1) Match Fields: to match against packets. These contain the entrance port and packet headers.

2) Priority: corresponding precedence of the flow entry.

3) Counters: to update for matching packets.

4) Instructions: to modify the action set or pipeline processing

5) Timeouts: maximum sum of time or idle time before flow is deceased by the switch. Switch

6) Cookie: mark used to filter exact entries when demanding figures, not used when processing packets.

- *Matching*: On receipt of a packet, an OpenFlow Switch achieves purposes starts by performing a table lookup in the first flow table, and based on a pipeline processing, may achieve table lookups in other flow tables. Packet match fields are removed from the packet. A packet matches a flow table entry if the standards in the packet match fields used for the lookup match those well-defined in the flow table entry. If a flow table entry field has a importance of ANY field omitted), it counterparts all potential standards in the header. If the switch supports arbitrary bitmasks on precise match fields, these covers can more exactly stipulate matches.

- *Counters*: Counters are used to gather the figures of processes and processes held by OpenFlow switch. They are preserved for each flow table, flow entry, and port.

- *Actions* : To process the packets, flow entries are followed with an action or a list of actions to be executed. A switch is not essential to support all action kinds, just those marked "Required Action" below.

1) Output/Drop (Required). The Output action forwards a packet to a specified OpenFlow port. OpenFlow switches must support advancing to physical ports, switch-defined rational ports and the required reserved ports. Drop is used to Drop the packet.

2) Set-Queue (Optional). The set-queue action sets the line id for a packet. When the packet is advanced to a port using the output action, the queue id controls which queue attached to this port is used for scheduling and forwarding the packet. This action is used to offer basic Quality-of-Service (QoS) support.

3) Group (Required). Procedure the packet over the specified group.

4) Push-Tag/Pop-Tag (Optional). Switches may support the capability to push/pop tags. To aid combination with current networks, the capability to push/pop VLAN tags is suggested to be supported.

5) Set-Field (Optional). The numerous Set-Field actions are recognized by their field type and adjust the values of particular header fields in the packet.

- *Instructions*: Each flow entry comprises a set of instructions that are performed when a packet matches the entry. These instructions is consequence in changes to the packet, action set and/or pipeline processing.

IV. SDN DESIGN

SDN architecture requires a controller connected to all the forwarding devices as far as they have open and programmable interfaces. OpenFlow is the Southbound API we used here. So, all the switches reside in the access and distribution layer are OpenFlow enabled switches. The router which interconnect the LAN to the backbone can be a legacy router. Ryu platform has been chosen to be the operating system and the controller of the hole network.

D. SDN Load Sharing

To utilize the redundancy links and share the loads between the switches, protocols such as PVST is used. But the configuration is done in a static manner. Using SDN concepts, by implementing an Algorithm to have a dynamic load sharing for the VLANs over the two distribution switches depending on their traffic loads. The algorithm is used to build a network application that runs by the Ryu controller[11]. OpenFlow switches have the benefit of using their counters to monitor the traffic and count the number of flowing packets and flow entries installed in the switch. The messages exchanged between Ryu and the switches is to be utilized here to request the information of the counters. The controller will use this information about the traffic to reconfigure the switches and rebalance the traffic inside the network depending on current loads of the VLANs. Fig.5 demonstrates the proposed implementation redundancy in a simplified topology of the company [12].

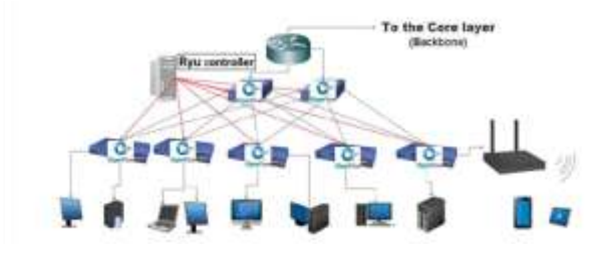


Fig. 5 The Simplified Network Topology

B. Simulation

In this simulation, a mini network representing the company network is created using Mininet software. The network will be composed of two distribution and six access switches. Mininet hosts will generate traffic between themselves, where each host belongs to one of the eight VLANs that will be configured. The network application is designed and applied upon a specific algorithm. The application is to be tested and the results will demonstrate the solution provided for the load sharing.

C. *Network Application*

The network Application has been designed to deploy the following algorithm:

- Initially, VLANs are divided among the two distribution switches, one switch will act as a root bridge for group of VLANs, and the second switch will take on the remaining VLANs. This step is equal to the PVST protocol.

- After a specific period of time, the controller sends a Read-State message to the root bridges querying about the traffic loads for each VLAN. The two switches reply immediately with consecutive reply messages.
- The controller uses the information about the traffic sent by the root switches to reallocate the VLANs among them as equally as.
- To apply the new Allocations of the Root switches, the controller sends Modify-State messages to the switches. This will include deleting, adding or modifying some of the flow table entries.
- The updating process is applied periodically. The time period shouldn't be too short to avoid processes overwhelming in the controller, and also shouldn't be too long to keep up with the VLANs imponderable traffic.

The Network Application is created as a Python scripts in a file named "Dynamic Utilize" saved within the Ryu applications folder .

D. *Network description*

The mini network Topology is designed to represent an approximation of the faculty of engineering network devices, users and VLANs. The topology of the network is shown in Fig. 6.

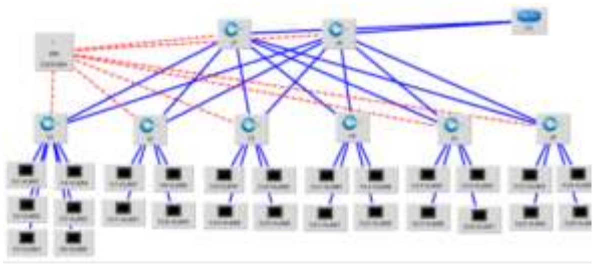


Fig. 6 Topology of the simulated network

There are 8 VLAN interfaces configured for the 26 hosts connected to the access switches. VLANs are fairly distributed among the hosts to represent the user groups whom targeted in the test. The following table contains the user groups, their VLAN ID's, the hosts and the subnet IP Address. Table 1 shows the user groups, their VLANs and addresses.

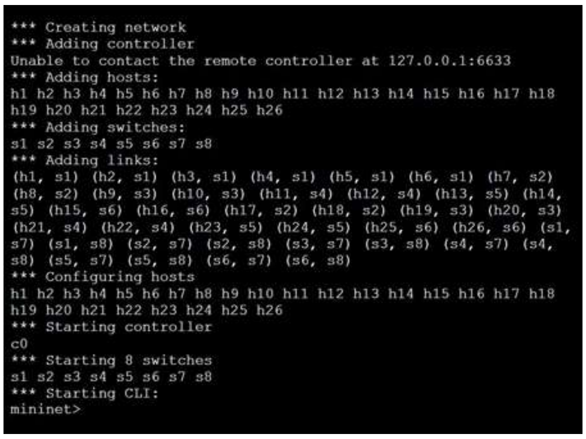
TABLE I. *USER GROUPS AND VLANS*

User Group	VLAN ID	Hots	Network IP
Employees	1	h1-h7-h13-h25	10.0.1.0
Technical staff	2	h2-h8-h15	10.0.2.0
Web services	3	h3-h9	10.0.3.0
Events Management	4	h4-h10-h14-h26	10.0.4.0
IT and Networking	5	h5-h11-h16-h18	10.0.5.0
IT and Networking	6	h6-h13-h20	10.0.6.0
Treasury	7	H17-h21-h24	10.0.7.0
Visitors	8	H190h220h23	10.0.8.0

E. *Design configurations*

Build an environment on Mininet. The command to be entered is the one shown below, which executes a Python script of the customized topology. After executing the command, a fully network with OpenFlow-enabled virtual switches, hosts and links are built as specified in the topology. As shown in Fig. 7.

VLAN ID is set to the interface of each host. This includes deleting the IP address that is assigned automatically on each host and set a new IP address.



```
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
h19 h20 h21 h22 h23 h24 h25 h26
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1) (h7, s2)
(h8, s2) (h9, s3) (h10, s3) (h11, s4) (h12, s4) (h13, s5) (h14,
s5) (h15, s6) (h16, s6) (h17, s2) (h18, s2) (h19, s3) (h20, s3)
(h21, s4) (h22, s4) (h23, s5) (h24, s5) (h25, s6) (h26, s6) (s1,
s7) (s1, s8) (s2, s7) (s2, s8) (s3, s7) (s3, s8) (s4, s7) (s4,
s8) (s5, s7) (s5, s8) (s6, s7) (s6, s8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
h19 h20 h21 h22 h23 h24 h25 h26
*** Starting controller
c0
*** Starting 8 switches
s1 s2 s3 s4 s5 s6 s7 s8
*** Starting CLI:
mininet>
```

Fig. 7 *Network environment establishment Command*

F. *Load Sharing Scenario*

Multiple different traffics are randomly generated between the hosts. Ping utility is used to generate four Traffics: A, B, C and D. Each Traffic is consecutively generated two times, one before the update take place and the other is after. This helps to demonstrate how can the reallocation of the VLANs to the root switches helps to improve the utilization of the redundancy links. The traffic is generated during eight minutes of simulation.

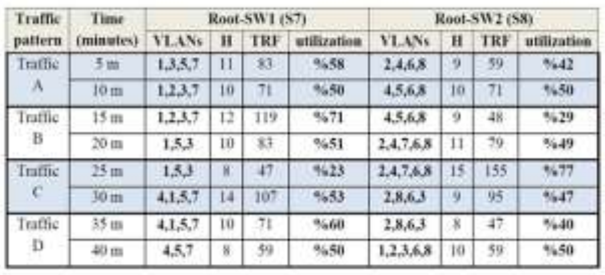
- VLANs: this column specifies the allocations of VLANs for each Root switch during a periodic time in which the traffic is generated.
- Number of Hosts (H): this column indicates the number of Hosts contribute to the traffic generation. Note that a higher number of hosts for a Root Switch doesn't require a higher traffic value.
- Time: the periodic time is chosen to be 5 minutes; the traffic is generated during this time. At the end of the time, the network application code instructs the controller to ask for the traffic statistics of the root switches function of Ryu to show the Traffic to the observer and calculate the utilization of the Root Switches.
- Traffic value (TRF): This value indicates how much of a traffic is passed through the switch and equal to the forwarded packets by the switch. We took advantage of Open Flow switch counters, which count the number of matched packets passed through the switch and send it to the controller as a reply for a using aggregate_stats_request request message.
- Utilization: it specifies each switch share from the overall traffic.

$$UT(S1) = \frac{TRF(S1)}{TRF(S1) + TRF(S2)} \times 100 \qquad (1)$$

$$UT(S2) = \frac{TRF(S2)}{TRF(S1) + TRF(S2)} \times 100 \qquad (2)$$

The optimum value of this entry is %50, which happens when each Root switch handles exactly the half of the traffic. An improvement in this column can be noticed after the update happens for each Traffic. As shown in Table 2

TABLE II. RESULTS: TRAFFIC AND UTILIZATION



Traffic pattern	Time (minutes)	Root-SW1 (S7)				Root-SW2 (S8)			
		VLANs	H	TRF	utilization	VLANs	H	TRF	utilization
Traffic A	5 m	1,3,5,7	11	83	%58	2,4,6,8	9	59	%42
	10 m	1,2,3,7	10	71	%50	4,5,6,8	10	71	%50
Traffic B	15 m	1,2,3,7	12	119	%71	4,5,6,8	9	48	%29
	20 m	1,5,3	10	83	%51	2,4,7,6,8	11	79	%49
Traffic C	25 m	1,5,3	8	47	%23	2,4,7,6,8	15	155	%77
	30 m	4,1,5,7	14	107	%53	2,8,6,3	9	95	%47
Traffic D	35 m	4,1,5,7	10	71	%60	2,8,6,3	8	47	%40
	40 m	4,5,7	8	59	%50	1,2,3,6,8	10	59	%50

As the simulation start, the controller configures the access switches according to an initial allocations using _init_ function in the code. Where VLANs: 1,3,5,7 traffic is sent to Root-Switch-1, and Root-Switch-2 takes the remaining VLANs: 2,4,6,8. The generated traffic results different loads on the Root Switches because the traffic of the VLANs is not equal. After 5 minutes, the controller collected the traffic stats on each Root Switch using the functions: send,_aggregate,_stats,_request and aggregate, stats, reply and handler. After the update took place, the utilization has been improved, and now every Root Switch take exactly the half of the traffic (50%) which represents the optimum load balancing.

V. CONCLUSION

This work presents a literature survey of SDN . A full description of SDN structure and its key components, including OpenFlow standard, Software-Defined Network eases the network administration, accelerates the innovation, reduce the costs, and enable programming in networks. A simulation of a Software-defined network was implemented using Mininet emulator. It adapted Ryu controller as the network operating system and the OpenFlow switches were established by Mininet, a network application is built to utilize the redundancy which produced a better load sharing configurations according to different traffic loads during time. The network application is a python script used the capabilities of the Ryu controller to collect the traffic statistics from the Open Flow switches. Results showed that an optimum or nearly load sharing can be given to the Root switches.

REFERENCES

[1] A.S. Tanenbaum, and D.J. Wetherall, *Computer Networks*. 5th Edition, Prentice Hall, *Inc., USA*, 2011.

[2] D. Hucaby, "CCNP SWITCH 642-813 Official Certification Guide" ,Cisco Press,2010.

[3] P. Oppenheimer, *"Top Down Network Design*", Cisco Press,2011.

[4] M. Casado, F. Foster, and A. Guha, " *Abstractions for software-defined networks. Commun*". ACM, Sep. 2014.

[5] N. MCKEOWN, T. ANDERSON, H. BALAKRISHNAN,G. PARULKAR, L PETERSON, J. REXFORD, SHENKER, S., and J. TURNER, "OpenFlow: Enabling innovation in campus networks", *ACM SIGCOMM Computer Communication Review*, V.38, 2(4) , 2008.

[6] D. Kreutz, J. Yu, P. Verissimo, F. Ramos, C. Magalhaes, The KISS principle in Software-Defined Networking: a framework for secure communications, IEEE Security & Privacy, Volume: 16 , Issue: 5 , September/October 2018.

[7] S. Azodolmolky, *Software Defined Networking with OpenFlow*, Packt Publishing, 1st ed., October 2013.

[8] F. Ramos, D. Kreutz, P. Verissimo, "Software-Defined Networks: On the Road to the Softwarization of Networking", Cutter IT Journal, May 2015.

[9] M. Casado, N. Foster, and A. Guha, "Abstractions for software-defined networks," CM Commun., vol. 57, no. 10, pp. 86–95, Sep. 2014.

[10] 9. Mininet. An Instant Virtual Network on your PC. Jun 13, 2018 from: http://www.mininet.org.

[11] Ryu Project. Ryu SDN Framework. Available at: https://osrg.github.io/ryu.

[12] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in Proceedings of the first workshop on Hot topics in software defined networks, ser. HotSDN '12. New York, NY, USA:ACM, 2012, pp. 7–12..