National College of Ireland

Elastic SDN Based Switch Migration Algorithm to Dynamically Update Map Tables with Minimum Latency and Achieve High Throughput

MSc Research Project
Cloud Computing

Chaitanya Balakrishna Deshpande
x17102146

School of Computing
National College of Ireland

Supervisor: Mr. Vikas Sahni

National College of Ireland
Project Submission Sheet – 2017/2018
School of Computing

Student Name:	Chaitanya Balakrishna Deshpande
Student ID:	x17102146
Programme:	Cloud Computing
Year:	2018
Module:	MSc Research Project
Lecturer:	Mr. Vikas Sahni
Submission Due Date:	13/08/2018
Project Title:	Elastic SDN Based Switch Migration Algorithm to Dynamically Update Map Tables with Minimum Latency and Achieve High Throughput
Word Count:	5357

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature:	
Date:	16th September 2018

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
3. Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Elastic SDN Based Switch Migration Algorithm to Dynamically Update Map Tables with Minimum Latency and Achieve High Throughput

Chaitanya Balakrishna Deshpande
x17102146
MSc Research Project in Cloud Computing

16th September 2018

Abstract

Software Defined Networks is proving to be the future of networking world and is here to stay. The traditional networks are getting replaced by SDN which mainly improves the flexibility of routing the packets. SDN emerged with an idea of having a separate network control plane from its forwarding plane. The control plane is the one which controls the entire network with its centralized controller residing in it. Its main task is to monitor the network behaviour and implement the network policy. Coming to the forwarding plane, i.e, the data plane, it just forwards the packets as indicated by the controller. This paper is focused on building a switch migration algorithm which migrates the switches to the controllers which are not fully loaded and thus having a balanced traffic in the network when the traffic is at its peak. Having a single centralized SDN controller increases the problems with scalability as it takes more time to respond back to the switch. This problem gave way to have a logically distributed multiple controllers which allowed switches to have a better communication and response time with the controllers. But this approach faces problems with static mapping between the switch and a controller which is unable to adapt to traffic variations. Hence multiple controllers at times, are over loaded and eventually causes delay resulting in less throughput. Therefore, to solve all the above problems, this paper showcases the use of Elastic SDN controller which dynamically scales the controller pool depending on the network behaviour. With this approach, latency issue is minimized with much better improvement in the throughput giving 80% efficiency after migrating the switches.

1 Introduction

The growth of data has been increasing rapidly over the past few years with a demand for the expansion of the network which makes it more faster and widespread. The traditional IP networks are complex and hard to handle, which makes it both difficult to configure and reconfigure them to respond to any occurrence to changes, fault and load etc. The main reason to head towards a better network topology is because of the vertically integrated networks concept. In other words, the control plane and the data plane reside in a single bundle which makes the network inflexible and non-reliable. Hence there is a shift

from the conventional networks to a programmable platform called as Software Defined Networks which is more flexible, scalable and highly reliable. The words Software Defined refers to the fact that it uses virtual switches that can be configured programmatically instead of having a dedicated hardware which is inefficient to changing network conditions. The main feature of SDN is that the control plane and the data plane are decoupled from one another and form a separate entity. The Control Plane provides a centralized control by means of controllers and manages the whole network topology which involves the processing rules to be set up for each and every packet that enters the network whereas the Data Plane just forwards the packets to the required destination as defined by the control plane. A strong secure communication channel called the OpenFlow Protocol exists between the centralized SDN controller in the control plane and the switches or the routers in the data plane to communicate with one another efficiently without any delay (McKeown et al.; 2008).

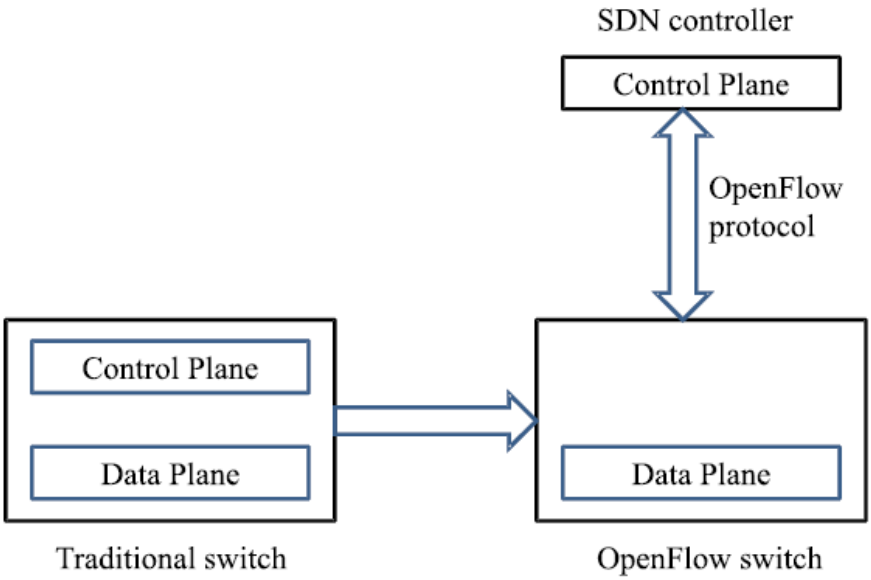


Figure 1: Traditional Network View Vs SDN View

Sezer et al. (2013) has stated that the conventional network style of approach had the control plane and data plane combined together in a single network node. Programming the paths and configuration of the node were the two main responsibilities of the control plane that were being used for data flows. Later on, the packets were forwarded to data plane once the path was known. It is based on this control information, the forwarding of the data takes place at the hardware level. In this approach, once the forwarding policy has been defined, the only way the policy can be balanced is through the changes to the device configuration. This can be a tedious job for the network operators to scale the networks whenever the traffic in the network demands. Therefore, as observed in Figure 1, the control plane is moved out as a separate centralized controller whose main responsibility is to manage the entire network. This in turn enables to have a Dynamic topology control by adjusting the usage of the switches or the routers depending on the traffic load in the network.

SDN is being promoted by the user driven non profitable association called, Open Network Foundation (ONF). Many SDN based architecture with OpenFlow based networks have been deployed (Ku et al.; 2014). The main feature of ONF is to promote and provide a networking environment that implements OpenFlow protocol so that the control plane can respond to the data plane. With enormous traffic in the network now a days, any kind of network drop or the delay has to be recorded and tested efficiently as it would have large volume of hosts, switches and controllers.

According to De Oliveira et al. (2014), this can be addressed by making use of prototypes and simulating them in virtual mode which can be tested through a software called as Mininet. Mininet is a network emulator which is used for deploying large networks on a simple single computer or virtual machine. Its main feature is to easily create, share, customize and test the SDN networks as seen in the Figure 2. Mininet contains number of different default typologies that emulates OpenFlow devices and the SDN controllers as well.

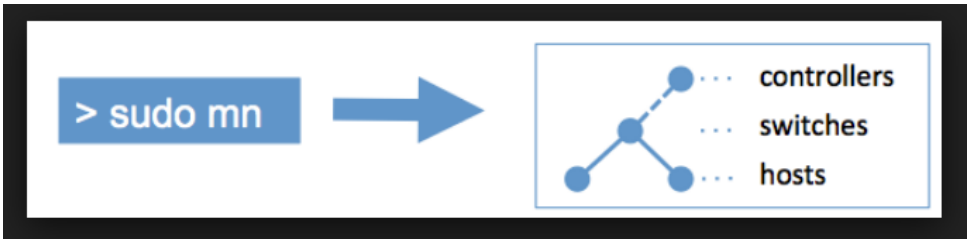


Figure 2: Emulating Real Networks in mininet

1.1 Research Question

Research Question: *"Can the packets be transferred from one switch to another by dynamically updating the map table without any delay and thus enhance the efficiency of throughput?"*

1.2 Research Objective

The research question will be addressed in detail by following the below specified objectives. Section 2 is focused on providing a brief over view on the literature in the SDN domain which describes its worth in the networking field and how it is different from the older networking features. Section 3 describes the methodology which involves the proposed idea of showing how the controllers scale dynamically. Section 4 describes about the implementation and design part, involving the proposed topology with existing and proposed design with facts and figures. Section 5 compares and evaluates the results with existing and proposed scenarios. Section 6 speaks about conclusion and the future work in this field.

2 Related Work

2.1 Three Layered Architecture of SDN

SDN has proved its worth by providing an all in one solution in the networking environment in terms of scalability, efficiency and experiencing an enhanced throughput. A high level overview of Software Defined Architecture is presented by the Open Network Foundation (ONF) for its development and standardization, dividing it into three main layers namely the Application Layer, Control Layer and the Infrastructure Layer. As mentioned by Kreutz et al. (2015), the network infrastructure has been changed drastically which tears the vertical integration by dividing the network's control plane which stores in the control logic of the entire network environment from the data plane which stores routers and switches that transfer the packets. A simplified view of the architecture is shown in Figure 3. Here, the controller can interact with the switches or routers with a well defined application programming interface (API), example OpenFlow. The Southbound Interface simply means, having a link between the control layer and the data layer (infrastructure layer) which should always remain open and secure for communication. The Northbound Interface means, having a link between the control layer and the application layer which would be having applications running on virtual or physical hosts that reserve resources like controllers as a backup by sharing policy information that makes easier for developer (Nunes et al.; 2014).

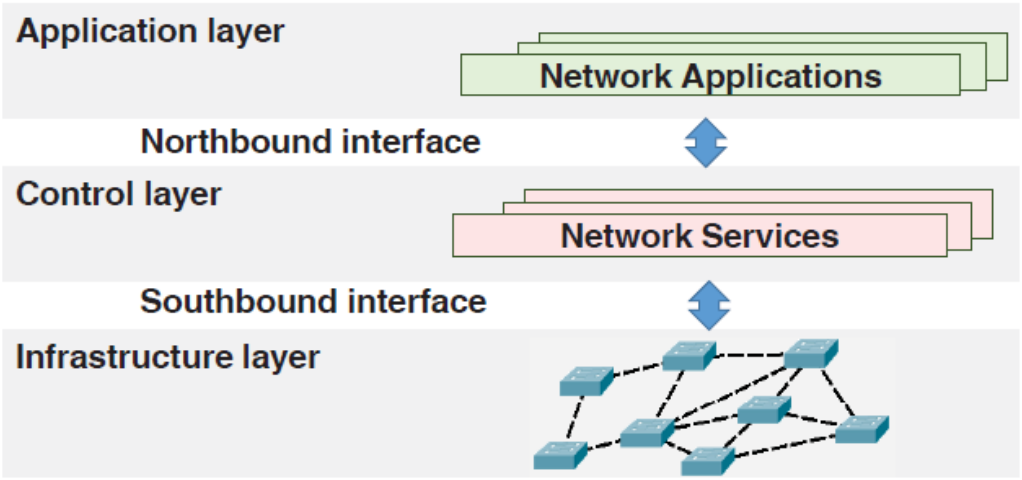


Figure 3: 3 Layered SDN Architecture

2.2 Controllers-Switch Interaction

The basic work-flow of how the packets are forwarded from one host to another is depicted Figure 4 with the interaction between a controller and a switch with the OpenFlow protocol. Here we see, packets P1, P2, P3 and P4 of a single flow F1 arrive at the switch one by one. The flow of packets is explained through the following steps in detail by Atli et al. (2017)

1. Consider the first packet P1 of Flow F1 enters the OpenFlow Switch.

2. Now the switch checks if any flow rule is installed in the flow table and finds out that no FlowTableEntry matching for packet P1 in table 0, so a table-miss entry matches and the basic action here to be carried out is to forward its packets to the controller.

3. The switch wraps packet p1 inside a packet-in message and forwards it to the controller using an OpenFlow API to generate a new flow. The packet-in message contains the header details such as the source_id, buffer_id, destination_id etc.

4. The controller picks up this packet P1 and sets up a new rule which includes the decision to route with a packet-out and flow modification message indicating that the packet P1 knows how it has to be routed and to which destination it has to be forwarded to.

5. Once the rule is known, the switch updates the flow table as per the flow_mod message.

6. Through the packet_out message the packet P1 would be forwarded to the next switch and the cycle repeats until the packet reaches the destination.

7. After P1, the next packets P2 arrive at the switch.

8. The switch checks if any rule is installed in the flow table about next packet P2 and eventually finds the rule associated with P2. This means that the P2 need not travel to the controller as the rule is known.

9. At last packet P2 would be forwarded to the next switch without the consent of the controller and so is for the packets P3 and P4.

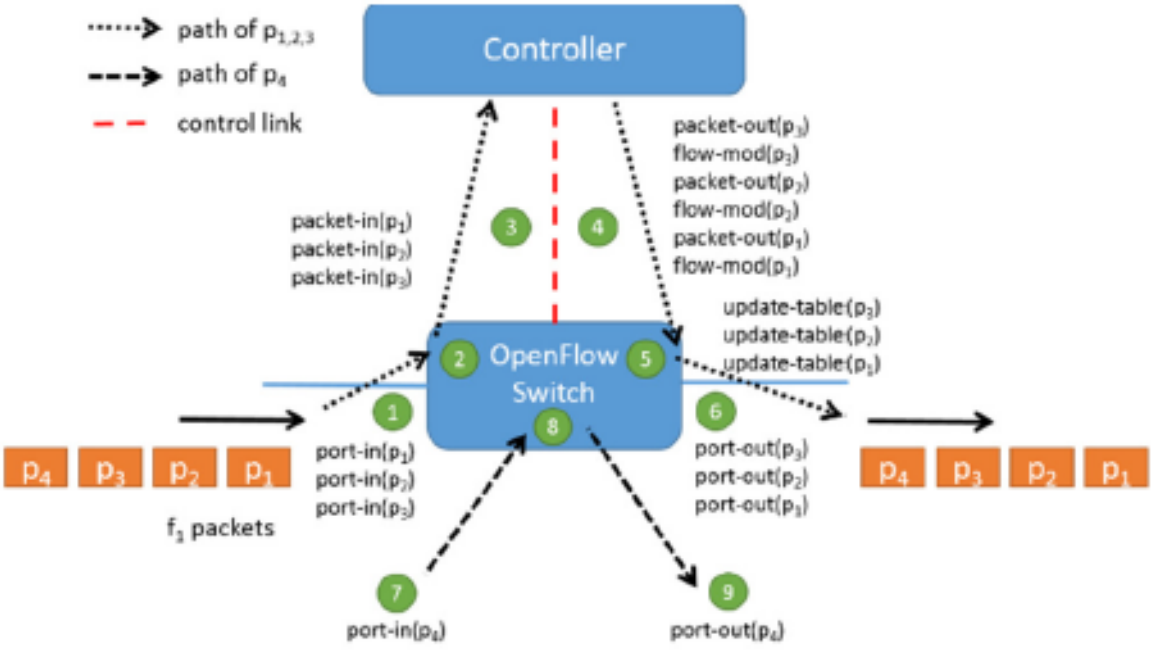


Figure 4: Packets Flow Scenario

2.3 The OpenFlow Switch

For an open software platform to exist, commercial routers and switches are typically not the ones which are adapted for the networking environment, as the external interfaces and standardization was narrow. This became a setback for researchers as they were running out of ideas to set up standard platform so that switch's internal flexibility is not hidden. In other words, flow-tables differed from vendor to vendor. To have a set of common functionalities, OpenFlow came into picture that can be run in many routers and switches. McKeown et al. (2008) said that, an open protocol is inherited to program the flow table in various routers and switches provided by the the OpenFlow. For every packet that enters through the switch, determines if any **Header Details** are matching which defines the flow. **Actions**, that describes as to how the packets must be traversed. **Statistics**, which keeps the count of number of packets which entered and left the switch and the time since the matching of the last packet.

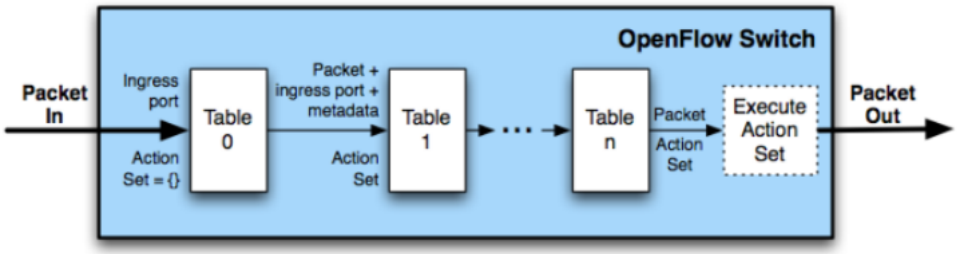


Figure 5: OpenFlow Switch

An OpenFlow switch is mainly composed of 3 different types of tables namely **FlowTable**, wherein for each specific flow, incoming packet is to be matched that in-turn specify the actions to be performed on each of the packets. There is something called as **Group Table** whose task is to activate actions that affect one or more flows. Also **Meter Table** can activate actions that are performance-related with respect to each of the flows. Stallings (2013) has said that " A flow is a sequence of packets traversing a network that share a set of header field values. For example, a flow could consist of all packets with the same source and destination IP addresses, or all packets with the same VLAN identifier". As seen in the Figure 5, packets are being matched against multiple tables one after another to find out for a highest priority matching flow entry. Specific instructions are added to modify the packet and for the updating the match field. The Ingress port is the one where the packets arrive in the switch with the action to be performed by carrying data from one table to another which is called as metadata.

2.4 Controllers in the Software Defined Networks

In SDN, controllers act as a brain of the entire network topology which contains all the policy information and hence is responsible for selecting the right path for the packet to traverse. Kreutz et al. (2013) has stated that the controller in the control plane is centralized with its overall view of the network environment, as a result of which development of more advanced network function is simplified. Also, any specious changes to the network state is automatically configured by the control program by maintaining

high-level policies. Depending upon the traffic in the network, controllers will be added or removed. Hence it can be said that, the controllers are centrally located but logically distributed to ensure better network capacity utilization. A single controller is used only when there is a single domain and multiple controllers are required when multiple domains are created (Yao et al.; 2015).

In a small scale network, **Single SDN Controller** was utilized to manage the traffic, as the load on the controller is enough to deal with the packets entering and leaving the switch. But, when the switch receives large number of flow requests, only one single controller is not able to serve and control the load on it. This proved out to be a serious concern with respect to network's scalability and reliability as it involved more delay and incurred high response time between the switch and the controller. To solve the above problem, **Multiple Controllers** emerged which were logically distributed over the network. By using this, load was uniformly distributed and latency was reduced with switches being allocated to controllers dynamically (Rath et al.; 2014).

According to Sridharan et al. (2017), flow setup requests were distributed to multiple controllers instead of it being mapped with only single controller. By doing this, the controller with least load on it, accepts the request and serves it. Hence, achieving more stability in dynamic traffic conditions with the availability of controller at any point of time to avoid single point of failure. If the controllers fail, it is obvious that any one of the controllers will overloaded. Strategy has to be made beforehand to have backup controllers in place all the time which will be in passive mode. A failure of a controller will automatically activate back up controller from passive mode to active mode and ensures there is no network drop.

Table 1: Comparison between different SDN controllers-(RIC proposal)

Technique	Advantages	Disadvantages	N/W Load
Single SDN Controllers	1. used mainly if packets input is low 2. easy to implement	1. not scalable 2. single point of failure 3 load on the controller would be more	High
Multiple SDN controllers	1. if one controller fails, another controller would carry the load of that controller. 2. no single point of failure. 3.enhanced control layer and better performance	static mapping between the switch an the controller	Moderate
Elastic controller	1.controllers would dynamically scale up and down. 2. load balancing performed periodically. 3. increase in throughput 4. minimum latency	resource utilization for shrinking and expanding the controllers might be a concern	low

The multiple controller approach had the limitation with the switch mapping technique, hence emerged with **Elastic SDN controllers** where in controllers dynamically scales depending on the traffic to balance the load.

When the request made to the controller is greater than the threshold it can handle, a new controller is dynamically added into the network pool to have a balanced traffic. If request to the controller is lesser than the threshold, then controller would be removed from the network pool. In other words, controllers can contract and relax depending upon the traffic conditions so that there is no congestion in the traffic. To sum it up in short, with three controllers and their respective roles, here is a Table 1 with its advantages and disadvantages as mentioned in the research proposal.

3 Methodology

In this project, routing of the packets is successfully carried out from source to destination without any traffic loss and thus avoiding congestion in the network. As depicted in Figure 6, a simple flowchart explaining the flow of packets is described. Seeing that no rule is installed at the switch, switch sends the packets to the controller and a new flow rule is designed. At the controller, the traffic burden is monitored by a monitor in the backend to know if the controller is loaded with more packets or not. If the controller crosses the load threshold, proposed algorithm works to migrate the switch from one controller pool to another with the controllers dynamically shrinking and expanding with respect to the traffic load in the network. Once the flow rule is found out for a particular packet, it is then delivered to the switch with packet_out message added onto it. With this approach latency will be reduced between the controllers and switches and thus improve overall scalability and reliability of the network.

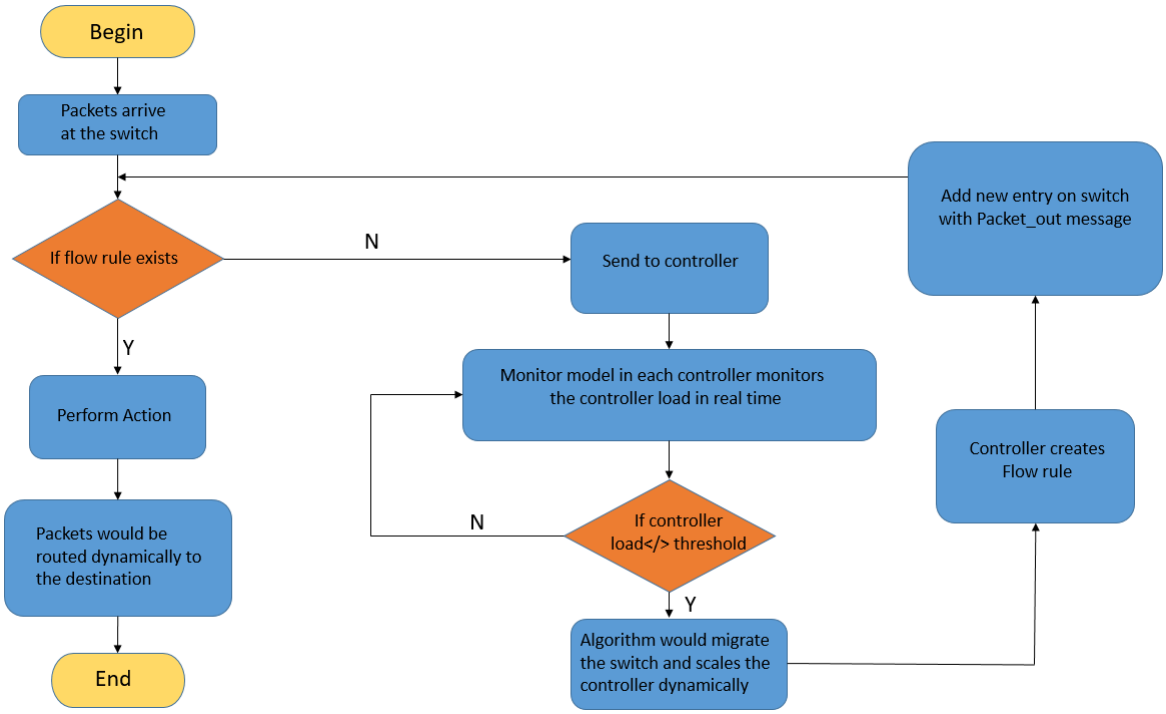


Figure 6: Proposed Flowchart

3.1 Migration Protocol with balancing the load among controllers

Balancing the traffic load amongst the controllers is the main factor to be considered to have a congestion free network and to have that, migration protocol is the one where the migration of switches takes place whenever there exists an unbalanced network. It is the key feature of Elastic SDN controller which helps in providing a better network environment.

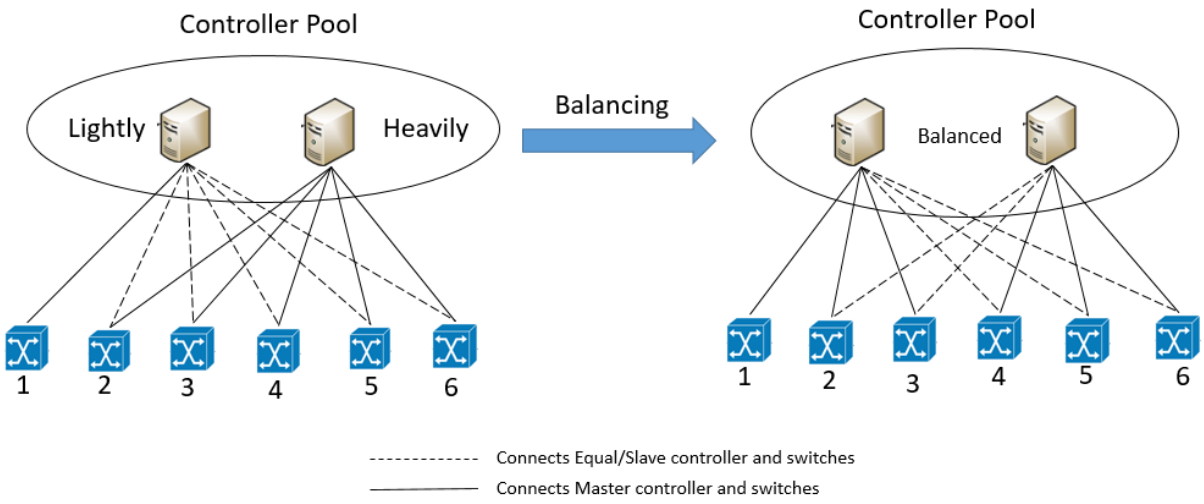


Figure 7: Load Balancing among controller

To improve network performance and to avoid controller crash, packets need to be evenly distributed among the controllers. Load balancing is achieved by migrating the switch from master controller to the slave controller. Two main concepts have to be kept in mind when migrating a switch: 1) when the slave controller is named to become a new master controller, it should not be overloaded. 2) there must be balanced load on the controller after migration. To decide how the switches can be migrated, we need to see if the controller is overloaded, and check is there any feasibility for each switch connected to that particular controller to migrate to other controller which is lightly loaded. If yes, shift that switch to the lightly loaded controllers so that no delay is encountered between the controller and the switch, then proceed the action of transferring of packets. The Figure 7 depicts a clear picture of the migration of switches. Here the left part specifies network before balancing the traffic load and right part of the network after balancing the load. Switch S1 is connected to a lightly loaded left controller and the other five switches, from S2 to S6 are connected the heavily loaded right controller which makes it imbalance. Hence to balance the network (after migration), two switches S2 and S3 from the heavily loaded controller is connected to the lightly loaded controller to have even distribution of traffic (Chen et al.; 2015).

3.1.1 Shrinking of Controller Pool

The controller pool is shrunk when the traffic at the controller is average and is less than the given capacity to handle the load. The switches that are connected to the master controller is migrated to the slave controller using the proposed algorithm. To make any controller to go in the inactive state, none of the switches should be connected to the that particular controller. Checking the controller that is in the inactive state is the important step to be considered. Here, we see the minimum loaded controller in the network which has the packets in it, and confirming that it has not yet crossed the average controller capacity. Next, feasibility is checked, if all switches connected to the master controller can be connected to the slave controller, if yes then all the switches are migrated and the other controller becomes idle. Hence increasing the efficiency and reliability to minimize the latency and thus improve the throughput.

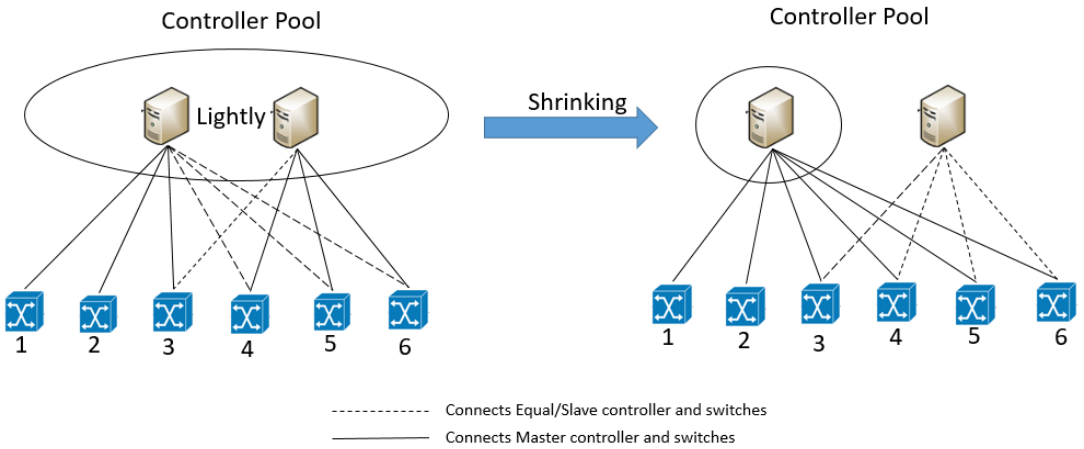


Figure 8: Shrinking of a controller pool

For example, as observed in the Figure 8, two scenarios exist.

- Before shrinking- where, the pool has two lightly active controller (left part).
- After Shrinking- where the controller has shrunk, with the Switch S4, S5, S6 migrated to the left controller and the right controller is made inactive (Right Part)

3.1.2 Expanding of Controller Pool

The controller pool is expanded when the traffic at the controller is overloaded and is greater than the given capacity to handle the load. More number of controllers are added in the network topology dynamically so that network works smoothly without any breakdown to make sure that no single point of failure occurs. When a controller is overloaded, proposed algorithm activates controllers and allows migration of switches to the newly added controllers. In other words, migration form a heavily loaded controller to the inactivated or the slave controller.

The controller which is overloaded needs to be found out and in turn check the feasibility of migrating the switches from that particular controller to the slave controller in the inactive state.

If yes, switches are migrated to the slave controller from the master controller. By doing this, traffic is balanced on both the controllers instead of one, which again increases the efficiency and reliability to minimize the latency and thus improve the throughput.

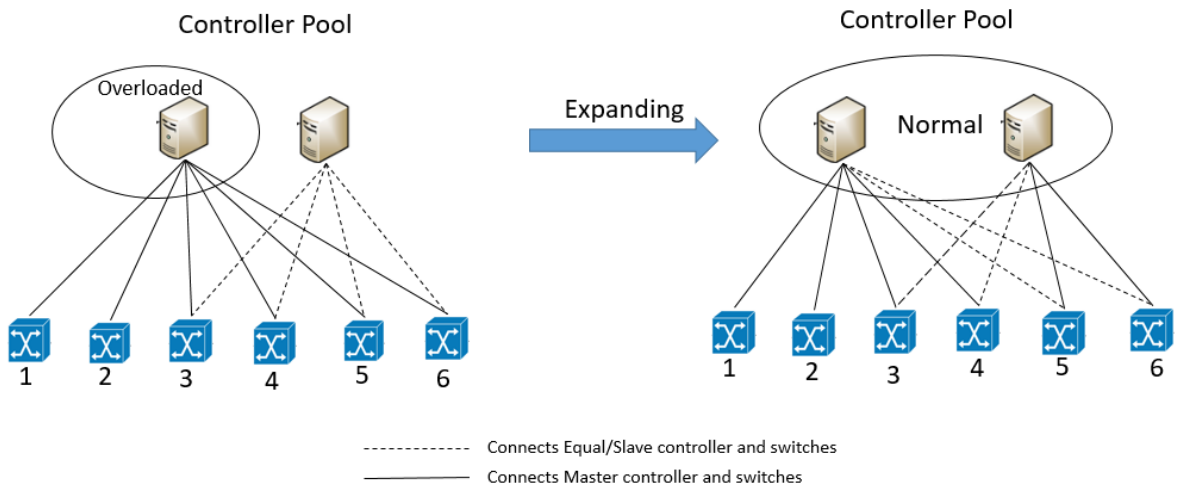
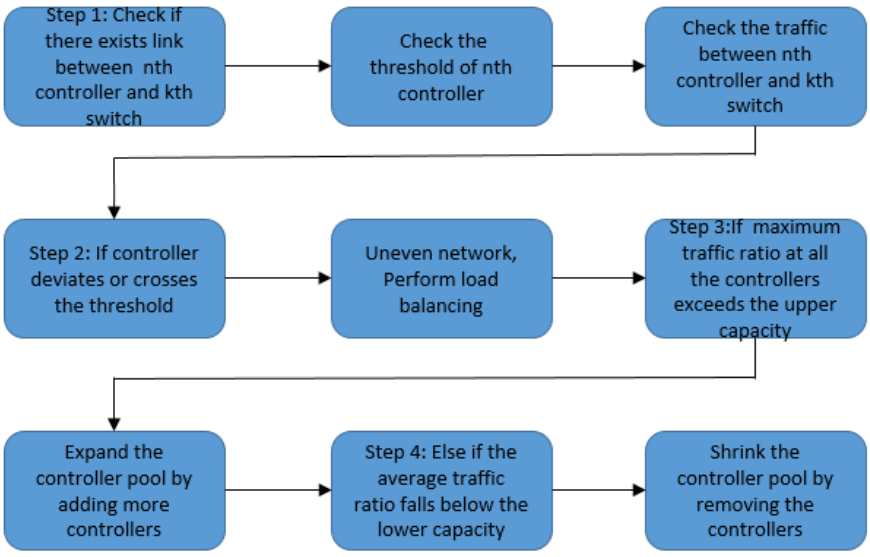


Figure 9: Expanding of a controller pool

For example, as observed in the Figure 9, two scenarios exist.

- Before expanding- where only one controller is in the active state which is overloaded and the other controller is in inactive state(left part).
- After expanding- where the last two switches namely S5 and S6 are migrated to the right controller by making it active (Right Part)

3.2 Flow Diagram for Shrinking and Expanding of Controllers



.

4 Implementation And Design

4.1 Technology

4.1.1 Mininet

To simulate large network of virtual hosts, links, switches, controllers in a single linux system, mininet is used, which acts as a network emulator to emulate the network environment using light weight virtualization. (Mininet; 2018) has stated that "Mininet hosts run standard Linux network software and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking. Mininet supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete experimental network on a laptop or other PC". In this project, we have installed Ubuntu 14.04 in the Virtual Box and in that mininet has been installed for simulation and to run various test scenarios. Figure 10, depicts the installation of mininet.

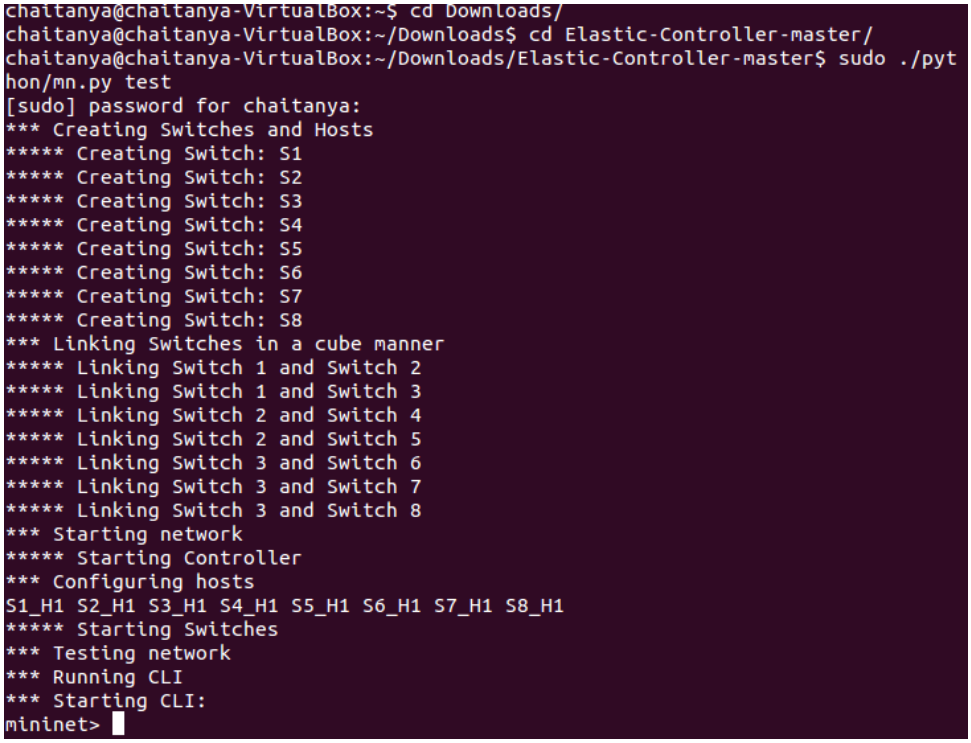


Figure 10: Connection Of Mininet

4.1.2 RYU Controller

Asadollahi et al. (2018) has stated that "RYU is an open source component based software defined networking framework under the Apache 2.0 license, written completely based on Python, supported and deployed by NTT cloud data centers". Main source code can be found in GitHub, provided and supported by Open Ryu community. It supports NETCONF and OF-config network management protocols, as well as OpenFlow.

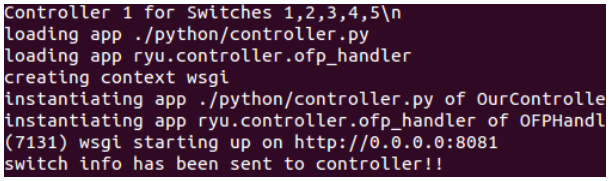


Figure 11: Switches connecting to Controller C1

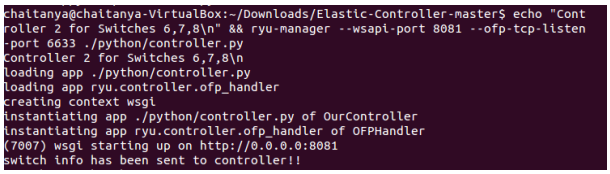


Figure 12: Switches connecting to Controller C2

RYU controller has plenty of libraries and packages, handles events related to the packets flow nd helps in processing the packets. The main executable is RYU Manager which has to be connected to the OpenFlow switch before which it will by default be connected to ip address 0.0.0.0 and port 8081 by default as shown in Figure 11 and Figure 12. RYU has several built in application such as tenant isolation, topology discovery, firewalls, routers, VLan etc.
In this project, wireshark analyzer tool has been used to analyze the traffic in the network as seen in the Figure 13.

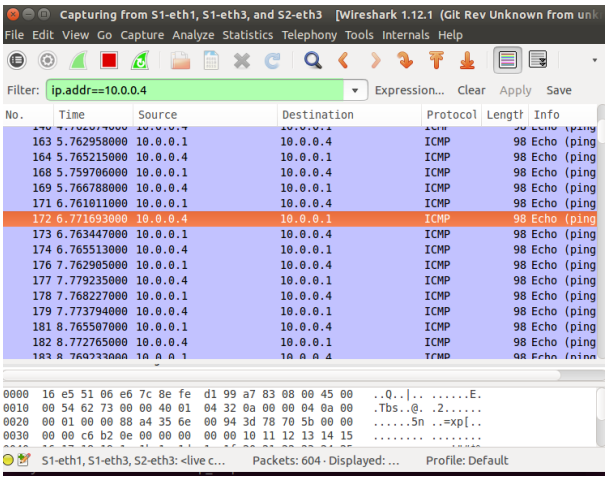


Figure 13: Traffic In Wireshark Analyzer

4.2 Topology Generation

As depicted below Figure 14, proposed topology consists of eight switches interconnected to one another with two controllers A and B connected to all the switches respectively. Each switch is associated with X hosts, where X can be configured according to our requirements. This topology is generated using Mininet Python Application Programming Interface. Here, monitor's main job is to monitor the controller's performance for all the switches. Monitor and controller communicate with each other through the HTTP requests. As there are series of information exchanging between them, monitor mainly concentrates on two important information sent by the controllers.

- The Package-In number of every controller in the network
- Once the migration is performed, controller sends a notification message.

Also monitor will send one type of information to the controller

- If the controller is overloaded, it advises the controller to begin the migration process.

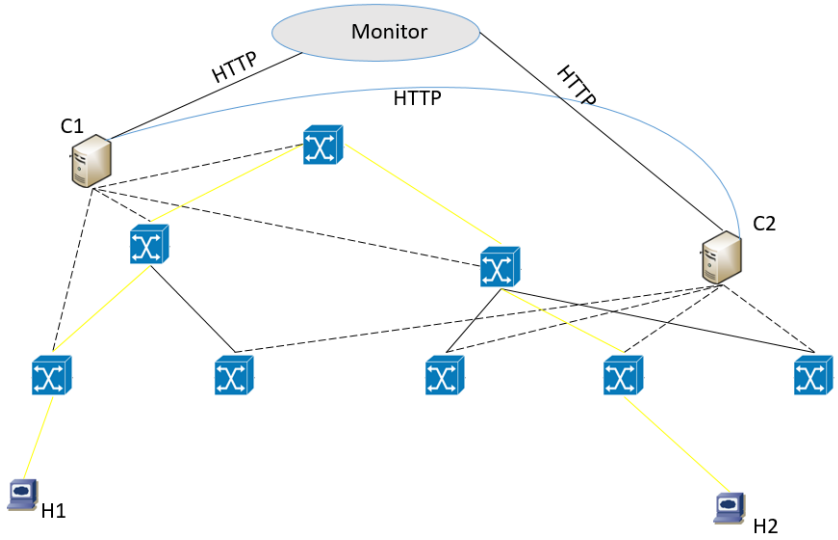


Figure 14: Proposed Topology

To decide if the switch has to be migrated, a command for the migration is to be issued which determines the recent collected traffic information for every 10 seconds. The reason behind using HTTP request is that, by default RYU controller framework provides HTTP server API for controller. As the controller delivers information on the traffic to the monitor for every 10 second, there will not be any performance related issues.
There are three main roles that the controller needs to do when the flood of packets has been transferred by the switch.

- Here the last 10 packets that has entered the controller is noted and in turn forward it to monitor and clear the cache.
- The basic forwarding of the Packet_messages from the switch to the controller is delivered which helps us in generating various test scenarios.
- Conduct migration

4.3 Architectural Design

4.3.1 Existing scenario for Routing of Packets

As explained in the research proposal by Deshpande (2018), "Consider the below Figure 15, where in there exists Hosts H1 to H5, Switches S1 to S8, Controllers C1 and C2. Let us see how SDN behaves in the following two scenarios. Here the Host H1 having a new flow F1 arrives at the switch S1. As it is a new flow, its obvious that the rule is not installed at the switch in the Flow Table. Hence the packet from Host H1 is forwarded

as a packet-in message to the controller C1. The controller C1 installs the flow rule at the switch S1 and S3. Next the flow arrives at S5, and sees that no rule is associated with this flow as the S5 is considered to be connected to controller C2 and not C1. Again the packet-in message is provided to C2 by S5 to install the rule. Therefore, rules on S5 and S7 gets installed. Considering large number of flows in the network with many hosts connected to different switches when traffic fluctuations arises.

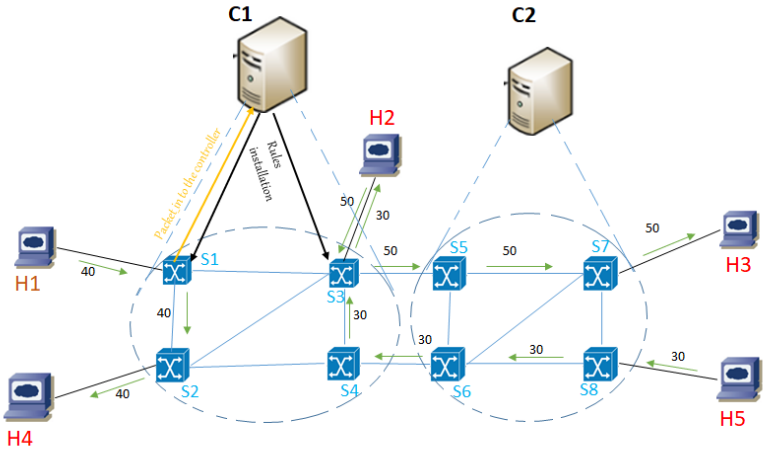


Figure 15: Existing Design

- say host H1 triggers 40 new packets to H4 and the flow is routed through S1-S2-H4.
- say host H2 triggers 50 new packets to H3 and the flow is routed through S3-S5-S7-H3.
- say host H5 triggers 30 new packets to H2 and the flow is routed through S8-S6-S4-S3-H2."

We need to determine the computational burdens on each of the controllers when ever new flows are instantiated. Let us consider 'x' unit of load at the controller for each and every single flow for path calculation whereas a single switch requires 'y' units of load at the controller for installation of rules of a single flow.
At Controller C1, when H1 triggers 40 packets, it generates 40x units for path calculation and for rules getting installed at S1 and S2 is (40+40)y units.
When H2 triggers 50 packets, it generates 50x units for path calculation and for rules getting installed at S3 is 50y units.
When H5 triggers 30 packets, it generates 30x units for path calculation and for rules getting installed at S4 and S3 is (30+30)y units.

At Controller C2, when H2 triggers 50 packets, it generates 50x units for path calculation and for rules getting installed at S5 and S7 is (50+50)y units.
When H5 triggers 30 packets, it generates 30x units for path calculation and for rules getting installed at S8 and S6 is (30+30)y units.

Assuming traffic Path Calculation > rule that is being installed, eg x=1 and y= 0.1[3]
Load Of C1 =(40+50+30)x + (80+50+60)y = **120.19 units per second**.
Load Of C2 =(50+30)x + (100+60)y = **80.16 units per second**
From the calculations, it an be seen that the controllers C1 and C2 are not rightly balanced because of its static mapping behaviour.

4.3.2 Proposed scenario for Routing of Packets

In the above figure, we see that S3 and S4 are not a part of C1 and its been connected to Controller C2. The computation burden for the flows from H2 and H5 is controlled by C2. Hence load traffic on controllers' would be :-
Load of C1= (40)x + (40+40)y = **40.08 units per second**.
Load of C2 =(30+50)x + (50+50+50+30+30+30+30)y = **80.27 unit per second**.

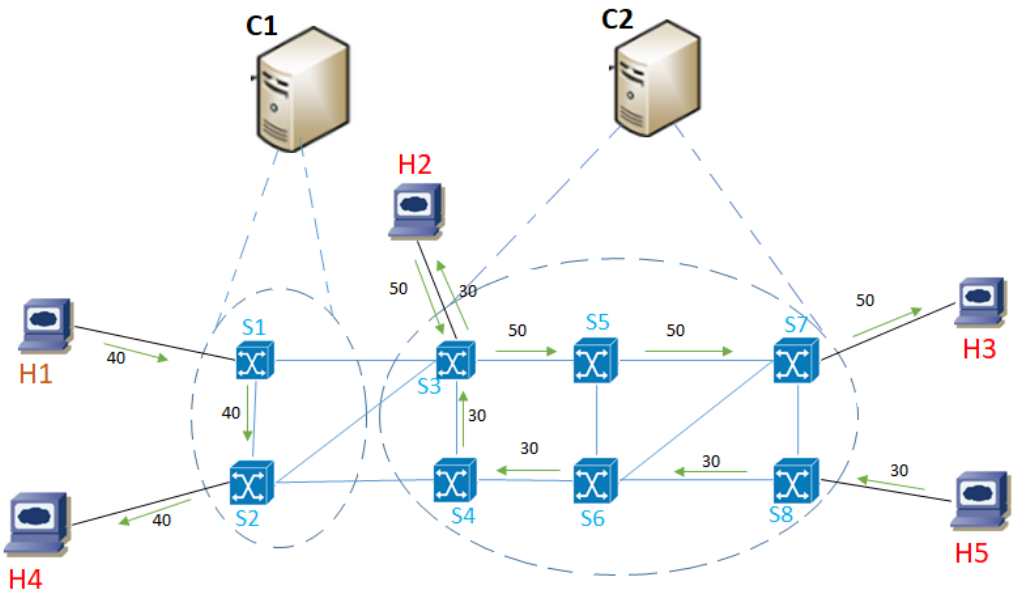


Figure 16: Proposed Design

Therefore, after the migration of switches S3 an S4, traffic (load) has been reduced to almost 80% at controller C1. and a small increase of 0.11% at controller C2 is seen as shown in Figure 16

5 Evaluation

Throughput of the controller and the time taken to respond back to the switch are the two main scenarios compared and evaluated before and after migration of the switch. Packets are sent from Host H1 to Host H2 which has been written in Python script as

depicted in the Figure 14. The path through which packets are being routed is from Host H1 To Host H2 indicated with the yellow lines. In this scenario, both the controllers C1 and C2 has four switches each, connected to one another. But in this scenario, the packets will be routed through all the four switches that is connected to controller C1. As the traffic is flooding more into Controller C1, there exists traffic imbalance in the network and eventually cause congestion. In other words, Controller C1 has four times traffic load than that of controller C2. Monitor observes this imbalance and migrates one switch from controller C1 to C2 which accounts to five switches with controller C2 and three switches with controller C1. Now, it can be said that, distribution of traffic and overall threshold of the controller are equal.

5.1 Before Migration

5.1.1 Throughput and Response Time

In the Figure 17, it has been clearly depicted that Controller C1 is loaded with heavy traffic and controller C2 is not, which in turn has more bandwidth which is not being used at all. The performance of Controller C1 has crossed the threshold capacity after sending at the rate of 1500 packets/sec because of which there is an imbalance in the traffic.

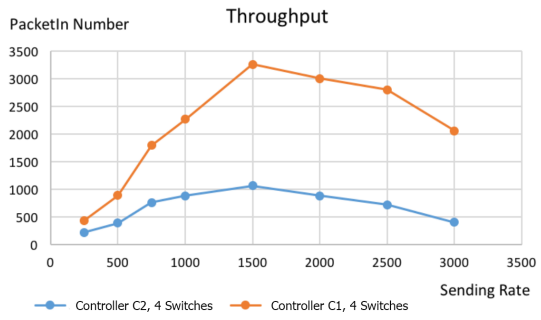


Figure 17: Throughput Of Controllers C1 and C2

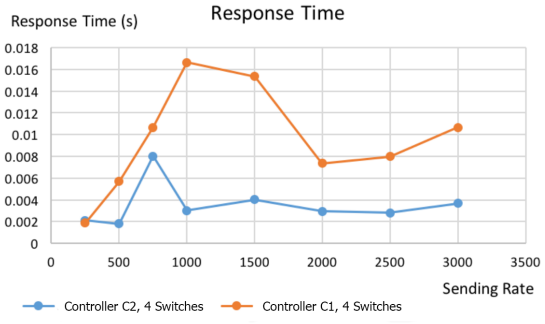


Figure 18: Response Time Of Controllers C1 and C2

Also in Figure 18, Controller C1 takes higher response time when compared to that of C2 as C1 is loaded with high traffic.

5.2 After Migration

5.2.1 Throughput and Response Time

In this Figure 19, there is significant improvement in throughput. The capacity of sending rate of the packets is more (around 2500 packets/sec) when compared to that of packets sent before migration.

Also in the Figure 20, to have a balanced traffic, the response time of Controller C2 is increased tremendously and Controller C1 reduces to have a balanced network.

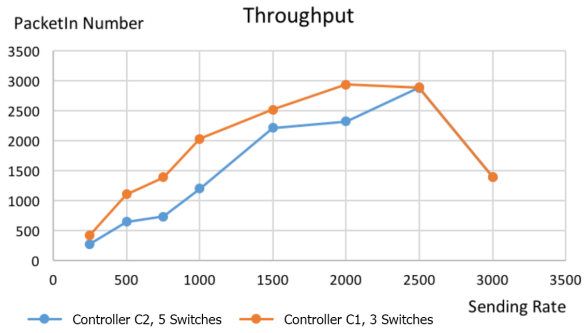


Figure 19: Throughput Of Controllers C1 and C2

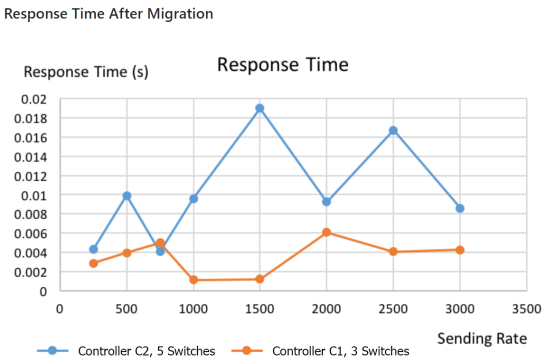


Figure 20: Response Time Of Controllers C1 and C2

6 Conclusion and Future Work

In this paper, based on SDN technology, an algorithm is proposed to migrate the switch whenever congestion exists in the network due the overload that controllers possess. That is the reason, the controllers shrink and expand dynamically depending on the network traffic. After the migration, the results shows the load being uniformly balanced between the controllers C1 and C2. The throughput increases drastically when the packets are flooded into the network and the response time the controllers take is a bit slow which might be because of the more traffic flow in the network as only a restricted bandwidth is allocated by the mininet. Therefore, by implementing the algorithm it is evident that throughput of the networks can be increased, which is the proposed objective in this project.

One of the future works could be deploying the system in a large scale network environment and evaluating the throughput and response time with real world network application being run on top of it. Secondly, the packets need to wait at the ingress switch, until the previous packets get processed by the controller even if the flow rule is known by that particular packet. Hence,the response time and the throughput could be more. An algorithm can be developed to transmit the the packets if the flow rule is known without depending on other packets to complete its transmission.

Acknowledgements

Finally, I must express my heartfelt gratitude to my supervisor **Mr.Vikas Sahni**, Professor in Computing at National College of Ireland who has been the driving force throughout the course of this research. Without his encouragement, guidelines and continuous support, this project would not have been possible. His valuable inputs and suggestions have always been extremely helpful to me in improvising the idea in completion of the the project.

I would also like to take this opportunity to thank **Dr.Horacio Gonzalez-Velez** who helped me giving continuous feedbacks in the initial stages of research .

Lastly, I would also like to thank my parents for their continuous support and encouragement.

References

Asadollahi, S., Goswami, B. and Sameer, M. (2018). Ryu controller's scalability experiment on software defined networks, *Current Trends in Advanced Computing (ICCTAC), 2018 IEEE International Conference on*, IEEE, pp. 1–5.

Atli, A. V., Uluderya, M. S., Tatlicioglu, S., Gorkemli, B. and Balci, A. M. (2017). Protecting sdn controller with per-flow buffering inside openflow switches, *2017 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, Istanbul, Turkey, pp. 1–5.

Chen, Y., Li, Q., Yang, Y., Li, Q., Jiang, Y. and Xiao, X. (2015). Towards adaptive elastic distributed software defined networking, *Computing and Communications Conference (IPCCC), 2015 IEEE 34th International Performance*, IEEE, Nanjing, China, pp. 1–8. Core Rank B.

De Oliveira, R. L. S., Shinoda, A. A., Schweitzer, C. M. and Prete, L. R. (2014). Using mininet for emulation and prototyping software-defined networks, *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, IEEE, Bogota, Colombia, pp. 1–6.

Deshpande, C. (2018). Elastic sdn based routing algorithm to dynamically update the map tables with minimum latency and achieve high throughput, National College Of Ireland.

Kreutz, D., Ramos, F. M. V., Verssimo, P. E., Rothenberg, C. E., Azodolmolky, S. and Uhlig, S. (2015). Software-defined networking: A comprehensive survey, *Proceedings of the IEEE* **103**(1): 14–76. Core Rank A*.

Kreutz, D., Ramos, F. and Verissimo, P. (2013). Towards secure and dependable software-defined networks, *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, ACM, Hong Kong, China, pp. 55–60. Core Rank A*.

Ku, I., Lu, Y. and Gerla, M. (2014). Software-defined mobile cloud: Architecture, services and use cases, *Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International*, IEEE, Nicosia, Cyprus, pp. 1–6. Core Rank B.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J. (2008). Openflow: enabling innovation in campus networks, *ACM SIGCOMM Computer Communication Review* **38**(2): 69–74. Core Rank A*.

Mininet, T. (2018). Mininet overview - mininet. [online] mininet.org. available at: http://mininet.org/overview/ [accessed 9 aug. 2018]., *Mininet Overview* **16**(1).

Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K. and Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks, *IEEE Communications Surveys Tutorials* **16**(3): 1617–1634.

Rath, H. K., Revoori, V., Nadaf, S. M. and Simha, A. (2014). Optimal controller placement in software defined networks (sdn) using a non-zero-sum game, *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, Sydney, NSW, Australia, pp. 1–6. Core Rank A.

Sezer, S., Scott-Hayward, S., Chouhan, P. K., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M. and Rao, N. (2013). Are we ready for sdn? implementation challenges for software-defined networks, *IEEE Communications Magazine* **51**(7): 36–43.

Sridharan, V., Gurusamy, M. and Truong-Huu, T. (2017). Multi-controller traffic engineering in software defined networks, *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, Singapore, Singapore, pp. 137–145. Core Rank A.

Stallings, W. (2013). Software-defined networks and openflow - the internet protocol journal, *The Internet Protocol Journal* **16**(1).

Yao, L., Hong, P., Zhang, W., Li, J. and Ni, D. (2015). Controller placement and flow based dynamic management problem towards sdn, *Communication Workshop (ICCW), 2015 IEEE International Conference on*, IEEE, pp. 363–368. Core Rank B.