National College of Ireland

Detection and Re-routing of elephant flows in a Software Defined Networking to avoid traffic congestion

MSc Research Project
Cloud Computing

Sainath Bavugi
x17110319

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

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

Student Name:	Sainath Bavugi
Student ID:	x17110319
Programme:	Cloud Computing
Year:	2017
Module:	MSc Research Project
Lecturer:	Vikas Sahni
Submission Due Date:	13/08/2018
Project Title:	Detection and Re-routing of elephant flows in a Software Defined Networking to avoid traffic congestion
Word Count:	5548

I hereby certify that the information contained in this (Detection and Re-routing of elephant flows in a Software Defined Networking to avoid traffic congestion) 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):	

Detection and Re-routing of elephant flows in a Software Defined Networking to avoid traffic congestion

Sainath Bavugi
x17110319
MSc Research Project in Cloud Computing

16th September 2018

Abstract

Software Defined Networking (SDN) is widely used in datacenters because it makes the network topology flexible. It is very crucial to monitor and manage both incoming and outgoing traffic in order to utilize the resources that are available and avoid congestion. In a network, a large/elephant flow absorb more network bandwidth as well as takes time to get processed which impacts QoS. In this paper, a variety of approaches are discussed for detection of elephant flows and routing algorithms. The proposed routing algorithm scans the network to find all the paths available between source and destination and calculates the link bandwidth of different paths available. This research paper used mininet as network emulator, sFlow for traffic analysis and POX controller along with the OpenFlow protocol to identify the network paths. The algorithm has been tested on a fat tree topology and evaluated against different scenarios. In all the test scenarios, it was possible to detect elephant flows and the algorithm finds first and second best path out of available paths. Ultimately, the second best routing path is displayed which could be opted for the large flow.

1 Introduction

In today's world, datacenters play a key role in many business operations as the data is directly saved to cloud. This huge amount of data is frequently carried to datacenters via network lines that have immense bandwidth. This data is originated from different sources such as web applications, backups and many more. In order to traverse the data smoothly, along the network lines, there are many other components that need to be managed in between source and destination with complex architectures. The amount of time required to configure these devices and capital investment is reduced by using virtual devices which can be provisioned and configured quickly to optimize the network performance. Nowadays, network virtualization is a popular technology that offers easy interface to create topology, monitor the components and managing the resources.

One of the well-known techniques of network virtualization is Software Defined Network (SDN). According to Farhady et al. (2015), SDN is built using three key pillars: Control plane, Data plane, and Controller.

In traditional networking, there were same number of controllers and routers/switches in a network whereas this is not the scenario in the SDN. Before SDN, a switch was firmly connected to both control and data plane which seemed like a closed system. Hybrid SDN is a combination of both centralized and distributed controller which has the ability to serve the inquires within no time compared to traditional systems. Data plane in the SDN handles the actual packets where switches/routers resides depending on the configuration noted in the controller. The control plane where controller functions discovers the network graph, communicates with data plane and keep the switches/routers flow table updated about the network. Therefore, modern datacenters utilize these features and execute SDN in their datacenter for managing network.

As stated by Hakiri et al. (2014), most of the datacenters often observe huge traffic because many packets of large size get into the network in no time affecting mice flows. Eventually, this increases the latency and impacts the Quality of Service (QoS). As the elephant flows are huge, majority of the bandwidth is eaten up and mice flow starve due to this behaviour. As per Pettit et al. (2017), most of the mice flows end up in same queue waiting for the large flows to get processed. Due to this behaviour, mice flows reach late to the destination. These latency-sensitive mice flows face trouble due to elephant flows. Therefore, it is important to identify the elephant flow and handle the routing path in order to avoid traffic congestion.

The Research Problem: - *'Can the elephant flow be automatically detected and rerouted through less utilized paths in software-defined networks using a sampling based method? '*. This question is crucial to study because most of the mice flows are affected and majority of day-to-day transactions are taken care by the short flows. Therefore, the main objective of this research is to identify the elephant in a network and suggest new network path using routing algorithm.

Further, this paper is structured as follows. The **Related Work** sections 2 describes the reason for identification of large flows is a SDN network and provides a gist on previous works executed by other researchers for large flow detection along with routing schemes. Following section 3, the **Methodology** discusses the proposed design approach for achieving the research objectives along with detailed explanation on architecture and block diagrams. Next section 4, the **Implementation** provides a technique to build the algorithm using different software components and packages. Subsequently, the **Evaluation** 5 section validates the routing algorithm and large flow detection by executing different scenarios on the mininet network.

2 Related Work

The increase in the amount of data being processed these days needs more and more processing power and a network with higher and higher bandwidth to handle the transmission. As datacenters are huge, there are a variety of flows that enter the network among which elephant and mice flows are common. This section defines the elephant flows and discusses the importance of their detection in a network, different approaches used and various algorithms applied to route them through available network paths.

2.1 Definition of flows

McKeown et al. (2008) described the flow as a number of packets which are sent and received between source and target that have common properties like IP header and protocol. As per Afek et al. (2018), large flows can be classified and can be defined into three types listed below by observing different parameters like time, a number of packets and the size of the packet throughout its life cycle.

Additionally, Al-Fares et al. (2010) said that majority of flows (approximately 80%) that appear in the network are mice and they eat less bandwidth whereas most of the bandwidth is utilized by elephant flows event though they are less in number (20% consumption). Therefore, it is crucial to identify the flow and take appropriate action to avoid traffic congestion.

- **Heavy Flow**

  "Given a stream of packets S, a heavy flow is a flow which includes more than T percent of the packets since the beginning of the measurement." - Afek et al. (2018)

- **Bulky flow**

  "Given a stream of packets S, and a length of time m, a bulky flow is a flow that contains at least B packets in the previous m time units." - Afek et al. (2018)

- **Elephant flow/Large flow**

  "Given a stream of packets S, and a length of time m, an interval heavy flow is a flow that includes more than T percent of the packets seen in the previous m time units." - Afek et al. (2018)

2.2 Different approaches of Elephant flow detection

After looking at the definitions and detection importance, this section compares and contrasts various techniques used by different researchers for large flow detection.

Packet Sampling:

Afek et al. (2015) proposed and implemented an algorithm named as Sample and Hold. This approach works on the packet sampling technique where an agent or application is used to sample packets randomly using a switch that keeps the controller informed about the types of flows which takes further action. A similar approach was used by Mallesh (2018), where the packet analyzer reads the incoming packets and then pass it on to python script which displays the route information and the source and destination.

Statistics Gathering:

As the name suggests, this approach collects statistics like time duration, number of packets, size, etc. There are two different types of techniques named pull-based and push-based used in statistics gathering as per Curtis et al. (2011). DevoFlow was proposed by Curtis et al. (2011) based on this approach which scans all the incoming flows and finds the large flow.

Host or Application based detection:

In this approach, a sniffer is used for detection purpose. This is set up at the source and scans all the flows that are being sent to target. If any large flow is found then it is marked before sending to the destination. Benson et al. (2011) proposed Mahout where a buffer of TCP sockets are monitored and calls in the OS are cut off using the library to mark the flow as large.

Swtich Trigger:

As its name suggests, the flows are taken care at the data plane layer instead of a control plane. Yu et al. (2010) proposed a system named DIFFANE that uses the controller to update the policies and route tables of a switch on the regular basis so that most of the flows are managed without control plane involvement. Switches were capable of building wildcard rules by matching the packet destination IP with approaching controller for flow rule.

Counters based detection:

This approach uses software implementations in which the switch flow table entries are updated/created based on each flow that is looked by the peer depending on the counter set. Recently Basat et al. (2017) proposed this approach using two algorithms IM-SUM and DIM-SUM that maintain two different tables to calculate the traffic volume for a particular flow and detects if it is large or mice.

The table 1 summarizes various techniques used for elephant flow detection.

Approach	**Advantages**	**Disadvantages**	**Identification**	**Burden on the Network**
Packet Sampling	∗Scalable based on traffic volume ∗Low implementation cost	∗Time consuming process ∗Burden on the switch controller	High	Moderate
Statistics Gathering	∗No modifications of hosts and switches	∗Fixed threshold ∗High bandwidth utilization	Moderate	High
Host or Application based detection	∗Detection rate is very accurate ∗Better utilization of resources	∗Every agent need to be updated	High	Low
Switch trigger	∗Reduces communication leading to less bandwidth utilization	∗Need special application at switch level ∗Fixed threshold	Moderate	Moderate
Counters based detection	∗Detection rate is very accurate	∗Expensive implementation ∗Fixed threshold	High	High

Figure 1: Comparison of Elephant Flow Detection Schemes (Bavugi (2018))

2.3 Routing Elephant flows in a Network

Routing of large flows is equally important in a network as they might block mice flow which are in the queue to reach destination. This section discusses the significance of routing the large flows and various algorithms that were proposed by other researchers.

A routing algorithm was proposed by Cui et al. (2016) and named it as DiFS system. In this algorithm, a threshold of 100KB is configured and if the flow is discovered which exceeds the limit then it is declared as large flow. Upon detection, the system transmits the flow to the destination by switching from an existing path to the network path which has an ample amount of bandwidth. Another approach proposed by Dixit et al. (2013) uses ECMP routing called Random Packet Spraying. This algorithm divides the flow into a configured size and sends them via different shortest paths available. Wu and Yang (2012) built a system named DARD which works with available hosts in the network. All the hosts in the network are taught about the loads on the available links and hosts are configured in such a way that they are efficient to decide which path would be suitable for transmission of large flow. The decision is left on the host/sender. Additionally, this approach uses the same threshold limit as DiFS system i.e 100KB. Another algorithm explained by Chiesa et al. (2017) uses ECMP with algorithmic perspective. It uses a static hash function where every link mentioned on the packet headers are matched with the elements of the network and with the paths that are shorter to its destination.

The approaches such as Random Packet Spraying are not suitable in real-world because they can cause packet reordering. As the packets are split into equal size and are combined at the destination, there is a risk of data shuffling due to different reasons like the arrival of packets at the destination might get late due to network slowness on a particular path. Other proposals like DARD might also not work as they work with each host and it would be difficult to update about the network which consumes time and bandwidth to communicate. Also, there could be an issue with scalability due to replicating the logic on to the network devices.

2.4 Importance of SFlow Management Tool in Elephant Flow Detection

Most of the researchers have used flow management tools such as NetFlow, SFlow, IPFIX, etc. for analyzing flows, classify them as per the threshold and definitions configured.

Hong and Wey (2017) explained the working of flow management tools and compared well-known tools as part of the research. Unlike NetFlow, sFlow uses less network bandwidth and consumption of resources is also low because packets are sampled randomly for analysis. Peter (2011) said NetFlow works on layer 3 network connections which allow limited vision whereas Sflow focuses on layer2 which help to sample the packet quickly. When there is an incoming flow at the switch, the packets are sampled and data is collected using sflow agent. The collected data is compiled into UDP datagram and sent over to sFlow-RT for further analysis in every 1000 milliseconds.

sFlow-RT (*sFlow-RT* (2018)) is a real-time traffic analyzer that listens continuously to the sflow agents which monitors the network and transforms it into meaningful metrics to understanding the network. Whenever sFlow notices large flows, the controller is notified about it and a temporary flow rule is entered specifying the switch Datapath ID (DPID) and destination IP address as a packet. As Sflow is very quick in sampling and classifying the flows, this tool plays a key role in the detection of elephant flow.

2.5 Network Emulators

Network emulators play a key role in research projects as they help the researchers save time, space and money than to buy and set up hardware components like switches, routers etc. They are very easy to build and perform similarly to hardware components. Emulators assist the end user to deploy virtual network within minutes that consists of hosts, switches, controllers, and links. These emulators mostly run standard Linux OS and support a variety of switches and controller. One such emulator is Mininet which supports research, development, and learning. The Mininet can create SDN elements, customize them, share them with other networks and perform interactions.

In most of the networking research projects, Mininet emulator is opted due to interface provided for easy management of resources and ability to virtualize different networking elements such as hosts, switches, routers, and links. Researchers de Oliveira et al. (2014) and Keti and Askar (2015) had conducted various experiments such as scalability, creating large topologies, prototyping, etc and recommended mininet emulator than other emulators in the market. Another researcher Wang (2014) matched mininet with EstiNet and advised that mininet is suitable to execute small network whereas it starts using memory and generating strange results if used in large networks. As mininet performs well in smaller networks, this is suitable for the research project because the datacenters topologies can be replicated very easily and quickly giving the opportunity to test the algorithm. Therefore, this research project had used mininet as a network emulator to perform networking operations.

After looking at various types of large flow detection approaches and routing algorithms, subsequent section explains the methodology to implement the proposed approach.

3 Methodology

Many researchers had proposed numerous solutions in order to detect large flows and route them in a datacenter. This section provides a summary of the application which complies with the research objective. After looking at the detection, another important aspect of research is, to begin with routing the large flow via an available path to its destination.

Further, the figure 2 represents the overall architecture of the solution which displays crucial components that assist in identification and routing of large flow.

3.1 Large Flow Detection and Routing approach

The goal of the research is to discover the large flows in a datacenter network and send them to the destination via an efficient network path. The solution shown in figure 3 explains both identification and routing of flows from origin to destination. Let's look at each element and its objective.

Source & Target:

Use of computing devices that are effective for communicating with each other and interface in order to originate the traffic in the network.

Switch:

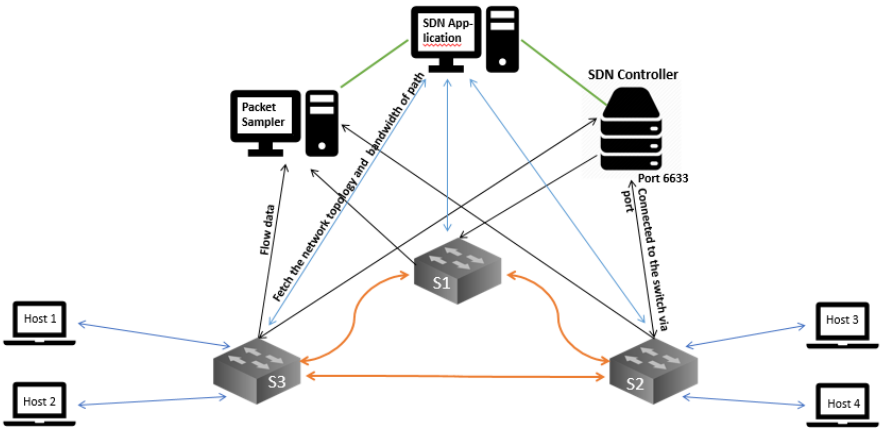

Figure 2: Architecture of Detection and Routing System. (Bavugi (2018))

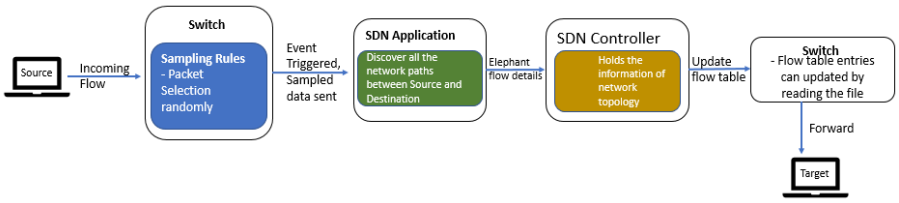

Figure 3: Large flow detection and Routing (Bavugi (2018))

Switches are the entrance for the flows to get into the network. They handle the incoming packets by matching the header with rules configured in flow tables. Once matched, they are routed to the next switch or to the destination. The incoming packets are sampled by a daemon based on the rules that are configured and it resides at the switch. When the threshold is exceeded then an event is triggered to the packet sampler i.e. sFlow tool along with the data that is gathered by sampling the packet.

SDN Application:

Once the flow is identified, the application helps in network path identification. It discovers all the network paths that are available in the topology with the controller from source and destination and calculates the efficient path.

SDN Controller:

All the switches are connected to this network component. Switch and controller are in constant touch and an update to the flow table of the switch can be performed from the controller. SDN application works with the controller to learn the topology that is built and particular links can be identified using LinkEvent.

3.2 Routing Algorithm

The goal of the routing algorithm is to discover all the network paths that are available in the topology and identify the second best path in order to route the large flow. Usually,

elephant flows are sent via shortest paths without looking at the bandwidth utilization of that particular link. This will slow down the transmission rate of the link impacting other flows that are in the queue. Instead, alternative network paths can be utilized for sending the large flows. By implementing this, mice flows are transmitted quickly and overall network performance improves.

Algorithm 1: Routing Elephant Flow (Bavugi (2018))

Condition: Flow should be a large flow
Input : srcip, dstip
Output : Second Best Path

$firstbestpath \leftarrow null$
$secondbestpath \leftarrow null$
$network \leftarrow getnetworktopology(srcip, dstip)$
`/* Paths are fetched using controller API and discovery module of POX */`
foreach *path in network* **do**
  $bandwidth_{path} \leftarrow iperf srcip, destip$
  `/* bandwidth calculation of each network path */`
  $T_{ptr} \leftarrow readtheavailablebandwidthfromoutputfile$
  if $bandwidth_{path}ofpath1 > bandwidth_{path}ofpath2$ **then**
    $secondbestpath \leftarrow firstbestpath$
    $firstbestpath \leftarrow path$
  else if $bandwidth_{path}ofpath1 > secondbestpath \;\; bandwidth_{path}ofpath1 \neq firstbestpath$ **then**
    $secondbestpath \leftarrow path$
  end

The algorithm 1 uses the bandwidth of a link in order to choose the path. Using iperf (bandwidth testing tool between source and destination B and R (2018)), it is possible to calculate the amount of data that can be transmitted through a particular network path. Once the network paths are identified from the topology, iperf is executed in order to decide the efficient path for the large flows. After execution the bandwidth of each link is saved to a output file. Post analyzing the bandwidth file, the application decides the path to be taken based on the available bandwidth and the suitable shortest path. The algorithm complexity is O(N).

Following section describes the modeling technique implemented to identify the elephant flow in a network and route them via efficient path towards the target.

4 Implementation

This particular section discusses the approaches used for implementation and development of routing algorithm in detail which helps in finding the efficient path. This elephant flow routing algorithm is built using Application Programming Interfaces (APIs) supplied by mininet, POX controller, OpenFlow, sFlow-RT and NetworkX is a software package.

Mininet as an emulator supports a wide range of network components including POX and OpenvSwitch (OVS) which are suitable for this research. Additionally, it provides a Python API for interacting with network elements. Therefore, it is important to understand how these components contribute in finding the efficient network path.

4.1 Communication between POX controller and Switch

According to Kaur et al. (2014), POX is an open source controller which is mainly used in building SDN networks and moreover it supports most popular communication protocol called as OpenFlow. As per *Open vSwitch Manual* (2018), OpenFlow switches are just a dumb forwarding element in a network. All the instructions are passed on to the switch from the controller in order to perform any action in the network. They are programmed by the controller for each and every action taken in the network.

When the controller and switches are created in the topology, there are many OpenFlow messages transmitted in between to establish the connection. As said by Soeurt and Hoogendoorn (2018), there are symmetrical message (SM), asymmetric message (CSM), Set config, Feature Request CSM, and Feature Reply messages that are exchanged between the controller and the switch. Once the SM packet is sent to switch, it identifies the controller and its compatible OpenFlow version number. After the establishment of the connection, other messages are sent to continue the communication. Additionally, the SM message is sent every 15 seconds in order to keep track of connection between controller and switches. In an OpenFlow protocol, all the flows have the idle_timeout configured to 60 seconds by default. The figure 4 explains the communication between controller and the switch.

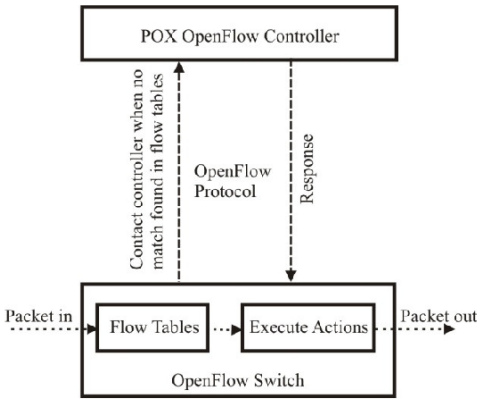

Figure 4: POX Mechanism with an OpenFlow Switch (Kaur et al. (2014))

For every incoming packet, the switch should have the rule to forward the flow to next switch or the destination. If the switch is not aware of flow rule for the incoming flow, it reads the packet headers and then passes them to the controller for further action.

4.2 Topology Specification:

A network topology is an arrangement of nodes and relating lines in order to communicate with each other achieving fault tolerance. The performance of the network and cabling

cost is directly dependent on the topology. Mininet is used for the creation of fat tree topology where the link bandwidth is specified and OVS is used as a switch.

Network links are emulated by the emulator along with the network elements. According to Requena et al. (2008), most of the datacenters are opting for fat tree as a network topology which helps achieve better network performance and fault tolerance. Additionally, this topology requires many switches, but flow can reach the destination via different efficient paths without worrying about latency.

The fat tree topology code is developed by Andreas (2016) which creates the topology as shown in figure 5 has the following components.

- Fat tree topology
- 20 switches
- 16 hosts
- POX controller

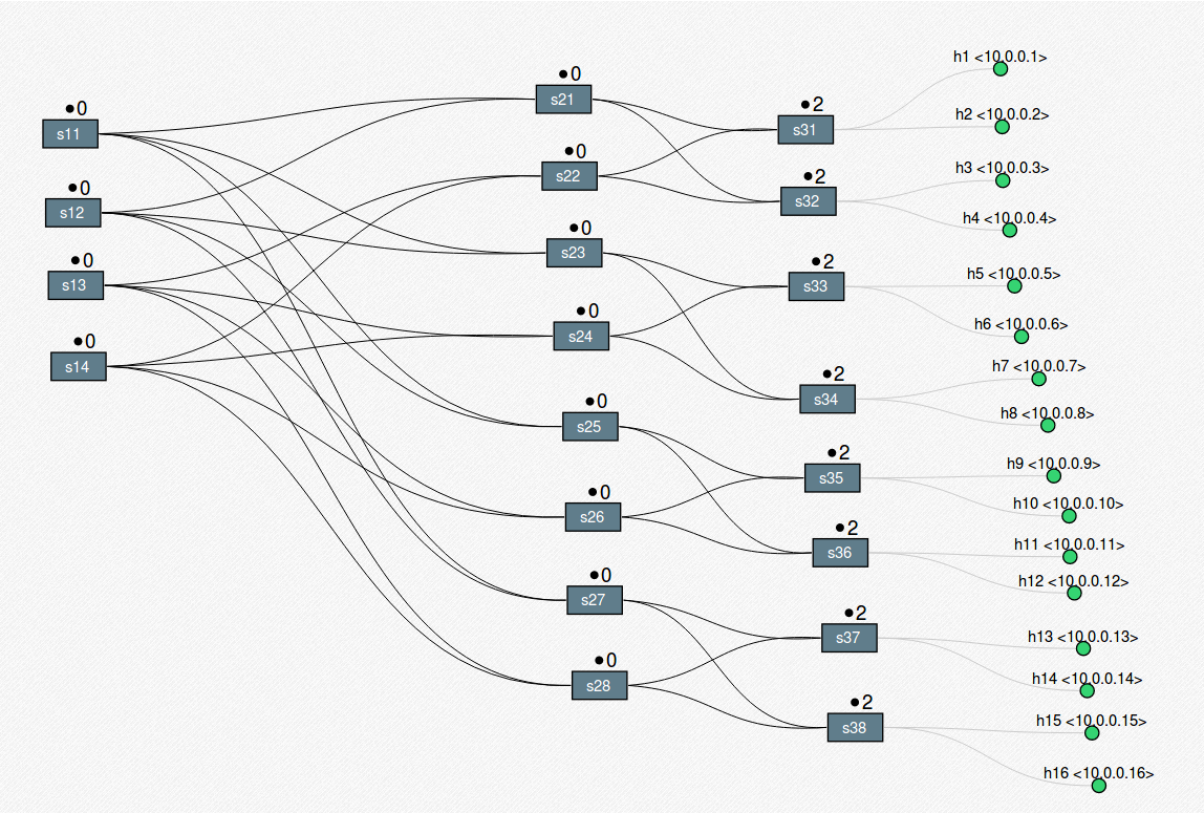

Figure 5: Network Topology

In fat tree topology, POX controller is created and connected to 20 OpenvSwitches along with 16 hosts connected to 8 edge switches. By default when the topology is connected to any remote controller in mininet, flow tables are empty. It needs to be manually added to communicate with each element in the network. The links between the switches and the hosts are configured to be 10 Mbits/sec to transmit any data. Also, the hardware address of the components is configured in number sequence instead of junk characters using -mac arguments.

Usually, the large flows create hindering in the network due to which smooth transactions are affected and cause mice flows to arrive late at the destination. One of the benefits of using mininet for creating network topology is that they are static which means it helps in testing and analyzing the flows without any change. Post creation of fat tree topology, sflow is notified and details are sent in order to replicate it on the mininet dashboard.

4.3 Configuring sFlow-RT and Mininet Dashboard

Mininet-dashboard (Phaal (2016)) is a real-time dashboard which displays the traffic flowing between the network elements. This helps in understanding the network path taken by the flow to reach the target. The topology is configured to handle ICMP, TCP, and UDP flow types and these are generated from a Linux environment.

As per Al-Fares et al. (2010), the size of large flow are generally 10% of the network bandwidth. As given by Phaal (2016), this range can be defined in the sFlow-rt which is capable of detecting elephant flow. If the flow enters into the network, the size of the flow is checked by the agent and reports it to sFlow-RT as a large flow.

The dashboard can be modified as per the user requirements. Adding the code written in figure 6 to the end of the metrics.js file under sFlow-rt will identify the elephant flow by the analyzer at switch and reports to sFlow-rt. This will display links that are involved in transmitting the large flow between hosts.

```
setFlow('pair',
  {'keys':'link:inputifindex,ipsource,ipdestination','value':'bytes'});

setThreshold('elephant',
  {'metric':'pair','value':1000000/8,'byFlow':true,'timeout':1});

setEventHandler(function(evt) {
  logInfo(evt.flowKey);
},['elephant']);
```

Figure 6: sFlow javascript code to detect large flows on dashboard (Phaal (2016))

4.4 Topology Discovery and Network Path Identification

In order to find the efficient network path, it is important to understand the network topology and find all the possible routes in between source and destination. There are different modules and packages used for finding the topology and network paths.

One of them is *openflow.discovery* which is responsible for topology discovery. It keeps the track of the change in topology and communicates via the listeners of network path program. According to Xu (2018), the module identifies the topology on the basis of Link Layer Discovery Protocol (LLDP) messages which are sent to OpenFlow switches for discovery. Additionally, *linkevents* are triggered when a link is up or down and an

attribute is added as *.addded* or *.removed* as per the link status. This link event is read at the listeners and update the network topology in its graph.

Another package used is *NetworkX* which is responsible to find all the network paths between source and destination. It is a python software package written by Hagberg et al. (2014). This particular package helps in studying the structure of the network topology, analyze the models, design new networks and draw networks. This package is used for listening to the events from openflow.discovery and analyze the network for further calculations. Once the graph of the network is analyzed, the controller calls the network path algorithm to compute all the paths between source and destination and load it to the output file.

Additionally, there are two more modules *l2_learning* and *spanning_tree* which are used for installing the flow rules in a network and avoid flooding on switch ports. The l2_learning reads the l2 address in the network and inserts flow rule that matches exactly to the flow header. The spanning_tree module is responsible to stop the flooding on ports of switches when they are connected and obstruct updating of flood before discovery module is executed.

The routing algorithm that is developed learns the output file which is loaded with the number of network paths and finds the path with fewer hops between the source and the destination. Post obtaining the second best shortest path, the bandwidth of the path is calculated using iperf command. It tests the link bandwidth between network elements by sending a large flow in the network and notes the bandwidth used for transmitting the flow. Based on this calculation, the path can be said to be the second best path for transferring the elephant flows leaving the shortest path for the mice flows.

After studying how to implement and find the second best shortest path in a network for routing elephant flows, let us look at the evaluation section which describes the testing approach.

5 Evaluation

In this section, the evaluation is performed by executing and capturing the test results to verify the behavior of the routing algorithm and other packages involved for network topology discovery. Since the routing algorithm needs to be compared against regular network paths, the research reveals an applicable evaluation method. Thus, the evaluation is performed using iperf which is a benchmark tool for finding the bandwidth between links and generate traffic in the network.

5.1 Experiment

In order to validate the shortest path routing algorithm, it is first important to note the route taken by the large flows so that comparison can be done. And before comparing the routes, it is crucial to identify the elephant flow. Therefore, the strategy chosen to transmit flows from hosts h1 to h16 and note the route traveled by the flow. Moreover, the experiment strategy can be repeated by generating a breakage between the switches by bringing down one of the links in the network to verify the algorithm working.

By utilizing sFlow-RT and mininet dashboard, the large flow or the elephant is detected and visualize the network path. Simultaneously, the routing algorithm is executed in order to find the second best network path that could be used by the flow instead of a first shortest path.

5.1.1 Experiment Setup:

To verify the algorithm, the POX controller is started which listens for the SM messages from the OVS switches. Also, the threshold is configured to the sflow which identifies the large flows based on the size. The source and target is set in the *topoDiscovery.py* file in order to find the paths in between. The shell script *./start.sh* is executed to start the sflow on localhost with 8008 port. This configured threshold is compared with the incoming flows and mark it as large flow. Once the listener is started, the custom fat tree topology is created using mininet and include *sflow-rt.py* in the topology so that the topology data is sent to the sFlow-RT (as shown in figure 7), dashboard and POX controller.

Also, the bandwidth link is created with 10 Mbits/sec and controller is mentioned as remote to connect to POX controller.

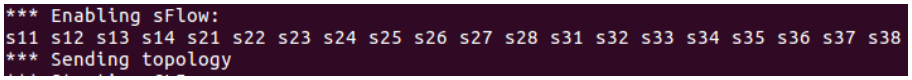

Figure 7: Mininet sends the topology information

5.2 Finding the Second Best Path

Post-execution of all the listeners and mininet topology, the traffic can be generated using iperf between h1 and h16 in order to find the normal route that is being taken by the large flow. The real-time traffic flow generated between h1 and h16 using iperf is displayed in the mininet dashboard. The path traversed by the elephant flow was h1->s31->s22->s14->s28->s38->h16 as presented in figure 8.

In order to reach the destination in this topology, the flow must traverse at least 5 hops and it is confirmed that elephant flow is taking the best shortest path available in the network. After looking at the path traversed by large flow, the routing algorithm is executed in order to calculate the second best path from *1360* network paths which can be opted by the elephant flow.

The figure 9 is the second best shortest path which is calculated by the routing algorithm after looking at the bandwidth of the path. This can be said to be second best because it has 7 hops and the first path has 5 hops.

5.3 Finding the Second Best Path When Link Between Switch is Down

The experiment is repeated by breaking a few links in between the switches to validate if the routing algorithm will work in this scenario. The links between switches s14 and s22, s32 and s24 are brought down to create a rupture scenario. Similar to the previous

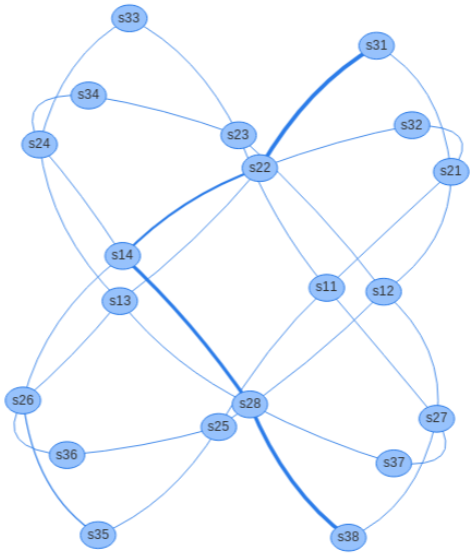

Figure 8: Mininet Dashboard real-time traffic - Test 1

```
Run CMDs.txt file: ran
The second best shortest path in this topology is:
['s31', 's21', 's32', 's22', 's14', 's28', 's38']
```

Figure 9: Second-Best shortest path - Test 1

experiment, the traffic is generated between hosts h1 and h16 and this time the shortest path that is taken is h1->s31->s22->s13->s28->s38->h16 as presented in figure 10.

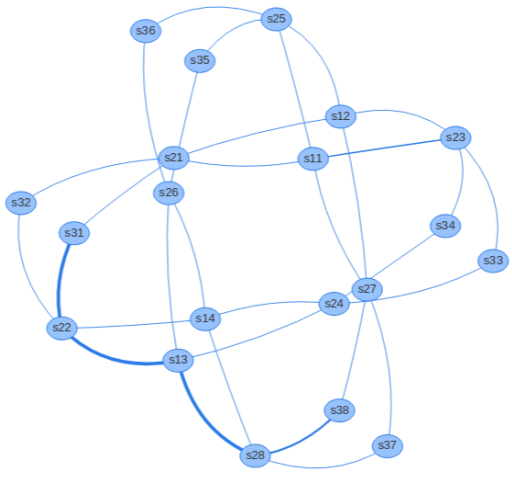

Figure 10: Mininet Dashboard real-time traffic - Test 2

The execution of the routing algorithm detected new paths from the topology and update the links removed from the topology. Once the topology is read, the shortest path (as shown in the figure 11) is calculated from the available paths after looking at the bandwidths.

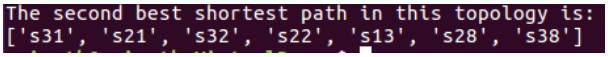

Figure 11: Second-Best Shortest path - Test 2

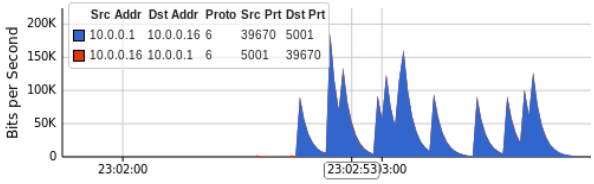

Figure 12: Mininet Dashboard Elephant flow Detection

The figure 12 is shown in the mininet dashboard when the elephant flow is in the network along with the source and destination details.

5.4 Discussion

This paper has taken two scenarios for validation purpose and verified the routing algorithm output. In both the scenarios, the routing algorithm had performed the calculations and displayed the second best shortest path that is available leaving the first one for the mice flows.

Looking at both the tests 5.2 and 5.3, the default routing path through which the large flow traversed is already the shortest path because there is no other path in the topology which has less than 5 hops. If the mice flows are sent via the same route, they get stuck behind large flow and wait for its turn for processing. Additionally, the topology discovery module is finding the new paths to destination when any of the link is brought down. By looking at graph 12, its evident that large flows carry huge data and they enter the network within no time. Therefore, the elephant flow can be sent via the resultant path to avoid traffic congestion and improve network performance.

6 Conclusion and Future Work

The aim of this research is to route the large flows via second best shortest path in order to allow mice flows to flow using first shortest path and optimize network performance. This improves the network performance as 80% of the flows are small such as insert, select and update queries to the databases. The routing algorithm had detected all the network paths that are available and opted the second best path in both the test scenarios for the large flow to traverse. Also, the detection was successful and the path opted by large flow was visualized in real-time to compare it with the routing algorithm results. There were many solutions already proposed by different researchers for detection of routing flows, but each have their own pros and cons. For instance, a popular routing algorithm such as *ECMP* routes the large flow via same path and overload the network which adversely affect mice flows. Majority of the research paper have concentrated on detection and routing of large flows but very few have looked at mice flow handling. As majority of

flows are small, it is important to investigate a better approach of handling mice flows in order to process them quickly and decrease the drop rate.

While the research outcome proved the feasibility of routing algorithm, the communication traffic between switch and controller is still an issue which enforces latency in the network such as find the routing for new mice flows. The controller is sometimes overloaded with requests from the switch in order to find the flow rule for mice flows. Due to frequent requests from switch to controller, performance degradation can be observed which might impact the network efficiency. In future, the routing path implementation can be investigated and suitable approach can be implemented to handle controller degradation.

References

Afek, Y., Bremler-Barr, A., Feibish, S. L. and Schiff, L. (2018). Detecting heavy flows in the SDN match and action model, *Computer Networks* **136**: 1 – 12. Core Rank A.
URL: *https://doi.org/10.1016/j.comnet.2018.02.018*

Afek, Y., Bremler-Barr, A., Landau Feibish, S. and Schiff, L. (2015). Sampling and large flow detection in SDN, *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, ACM, London, United Kingdom, pp. 345–346. Core Rank A*.
URL: *https://doi.org/10.1145/2829988.2790009*

Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N. and Vahdat, A. (2010). Hedera: Dynamic flow scheduling for data center networks, *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, USENIX Association, San Jose, California, pp. 19–19. Core Rank B.
URL: *http://ezproxy.ncirl.ie:3168/citation.cfm?id=1855711.1855730*

Andreas, P. (2016). panandr/mininet-fattree.
URL: *https://github.com/panandr/mininet-fattree*

B, C. and R, K. (2018). Rfc 7640 - traffic management benchmarking.
URL: *https://tools.ietf.org/html/rfc7640*

Basat, R. B., Einziger, G., Friedman, R. and Kassner, Y. (2017). Optimal elephant flow detection, *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, Atlanta, GA, USA, pp. 1–9. Core Rank A*.
URL: *https://doi.org/10.1109/INFOCOM.2017.8057216*

Bavugi, S. (2018). Routing elephant flows upon detection in a sdn to avoid traffic congestion.

Benson, T., Anand, A., Akella, A. and Zhang, M. (2011). MicroTE: Fine Grained Traffic Engineering for datacenters, *Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies*, CoNEXT '11, ACM, Tokyo, Japan, pp. 8:1–8:12. Core Rank A.
URL: *https://doi.org/10.1145/2079296.2079304*

Chiesa, M., Kindler, G. and Schapira, M. (2017). Traffic engineering with Equal-Cost-Multipath: An algorithmic perspective, *IEEE/ACM Transactions on Networking* **25**(2): 779–792. Core Rank A*.
URL: *https://doi.org/10.1109/TNET.2016.2614247*

Cui, W., Yu, Y. and Qian, C. (2016). DiFS: Distributed Flow Scheduling for adaptive switching in Fat-Tree data center networks, *Computer Networks* **105**: 166 – 179. Core Rank A.
URL: *https://doi.org/10.1016/j.comnet.2016.06.003*

Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P. and Banerjee, S. (2011). DevoFlow: Scaling flow management for high-performance networks, *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, ACM, New York, NY, USA, pp. 254–265. Core Rank A*.
URL: *http://doi.acm.org/10.1145/2018436.2018466*

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

Dixit, A., Prakash, P., Hu, Y. C. and Kompella, R. R. (2013). On the impact of packet spraying in data center networks, *2013 Proceedings IEEE INFOCOM*, Turin, Italy, pp. 2130–2138. Core Rank A*.
URL: *https://doi.org/10.1109/INFCOM.2013.6567015*

Farhady, H., Lee, H. and Nakao, A. (2015). Software-Defined Networking: A survey, *Computer Networks* **81**: 79 – 95. Core Rank A.
URL: *https://doi.org/10.1016/j.comnet.2015.02.014*

Hagberg, A., Schult, D. and Swart, P. (2014). Overview — networkx 1.9.1 documentation.
URL: *https://networkx.github.io/documentation/networkx-1.9.1/overview.html*

Hakiri, A., Gokhale, A., Berthou, P., Schmidt, D. C. and Gayraud, T. (2014). Software-Defined Networking: Challenges and research opportunities for future internet, *Computer Networks* **75**: 453 – 471. Core Rank A.
URL: *https://doi.org/10.1016/j.comnet.2014.10.015*

Hong, E. T. B. and Wey, C. Y. (2017). An optimized flow management mechanism in openflow network, *2017 International Conference on Information Networking (ICOIN)*, pp. 143–147.

Kaur, S., Singh, J. and Ghumman, N. S. (2014). Network programmability using pox controller.

Keti, F. and Askar, S. (2015). Emulation of software defined networks using mininet in different simulation environments, *2015 6th International Conference on Intelligent Systems, Modelling and Simulation*, pp. 205–210.

Mallesh, S. (2018). Automatic detection of elephant flows through openflow-based openvswitch - trap@nci.
URL: *http://trap.ncirl.ie/id/eprint/2873*

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*.
URL: *https://doi.org/10.1145/1355734.1355746*

Open vSwitch Manual (2018).
URL: *http://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.txt*

Peter (2011). Comparing sflow and netflow in a vswitch.
URL: *https://blog.sflow.com/2011/10/comparing-sflow-and-netflow-in-vswitch.html*

Pettit, J., Casado, M., Koponen, T., Davie, B. and Lambeth, W. A. (2017). Detecting an elephant flow based on the size of a packet. US Patent 9,548,924.
URL: *https://patents.google.com/patent/US9548924B2/en*

Phaal, P. (2016). Mininet dashboard.
URL: *https://blog.sflow.com/2016/05/mininet-dashboard.html*

Requena, C. G., Villamón, F. G., Requena, M. E. G., Rodríguez, P. J. L. and Marín, J. D. (2008). Ruft: Simplifying the fat-tree topology, *2008 14th IEEE International Conference on Parallel and Distributed Systems*, pp. 153–160.

sFlow-RT (2018).
URL: *https://inmon.com/products/sFlow-RT.php*

Soeurt, J. and Hoogendoorn, I. (2018). Shortest path forwarding using openflow acknowledgement.

Wang, S. Y. (2014). Comparison of sdn openflow network simulator and emulators: Estinet vs. mininet, *2014 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6.

Wu, X. and Yang, X. (2012). DARD: Distributed Adaptive Routing for Datacenter networks, *2012 IEEE 32nd International Conference on Distributed Computing Systems*, Macau, China, pp. 32–41. Core Rank A.
URL: *https://doi.org/10.1109/ICDCS.2012.69*

Xu, Y. (2018). Discovery topology in pox.
URL: *http://xuyansen.work/discovery-topology-in-mininet/*

Yu, M., Rexford, J., Freedman, M. J. and Wang, J. (2010). Scalable flow-based networking with DIFANE, *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, ACM, New Delhi, India, pp. 351–362. Core Rank A*.
URL: *https://doi.org/10.1145/1851182.1851224*