LINK ERROR DETECTION AND FAILURE RECOVERY IN SOFTWARE DEFINED NETWORKING

by

SWETHA VS	**2010103078**
THILLAI RAJA STS	**2012103611**
SUNIL NK	**2012103602**

A project report submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANNA UNIVERSITY, CHENNAI – 25

MAY 2016

BONAFIDE CERTIFICATE

Certified that this project report titled **LINK ERROR DETECTION AND FAILURE RECOVERY IN SOFTWARE DEFINED NETWORKING** is the *bonafide* work of **SWETHA VS (2010103078)**, **THILLAI RAJA STS (2012103611)** and **SUNIL NK (2012103602)** who carried out the project work under my supervision, for the fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on these or any other candidates.

Place: Chennai
Date:

Dr.MARY ANITHA RAJAM
Professor
Department of Computer Science and Engineering
Anna University, Chennai – 25

COUNTERSIGNED

Head of the Department,
Department of Computer Science and Engineering,
Anna University Chennai,
Chennai – 600025

ACKNOWLEDGEMENT

We would like to thank the entire committee consisting of Dr. V. Vetriselvi, Dr. S. Sudha, Dr. V. Mary Anita Rajam, Dr. A. P. Shanthi, Dr. Rajeswari Sridhar and Dr. D. Shiloah Elizabeth for constantly motivating us and identifying the areas of improvement on the project. We would like to extend our immense gratitude to our project guide, Dr. Arul Siromoney for his perpetual support and able guidance which was instrumental in taking the project to its successful conclusion. Finally, we would like to thank the Head of the department, Dr. D. Manjula for providing a conducive environment and amenities to facilitate our project work.

SWETHA VS **THILLAI RAJA STS** **SUNIL NK**

ABSTRACT

The present networking system is decentralized and the functionality is distributed. While this admits more freedom to respond to a failure event, it ultimately means that each controller application must include its own recovery logic, which makes the code more difficult to write and potentially more error prone.

Software Defined Networks makes the network centralized and execute the task in a systematic fashion. It has two main parts. ie., Data plane and Control plane. The whole network is monitored using SDN Controller which is considered to be the brain of the network. Any modification in the network such as updations and deletions are only done via the Controller. Software defined networking is an approach to computer networks where the data plane resides in the network devices, and the control plane in a centralized controller. The data plane helps in forwarding the traffic and the control plane decides about where the traffic is sent. The communication between the controller and the network devices is through the Openflow protocol. Controller studies the network topology and updates the forwarding table on routers and switches. The switches perform the action mentioned in the rules in the forwarding table.

Our project is mainly focused on link error detection and recovery. A network is can be said reliable if there is no packet drop

and its communication is done without any delay. Both of these are affected if a link is failed and so the links in the network must be kept resilient.This project deals with an unique mechanism to overcome the link failures. Our project ensures that the packet is rerouted properly to the destination node even if the link fails and the restoration policy of the state is taken into consideration. This will make the network reliable and this is done in a virtualized environment called mininet in order to provide with better results.

ABSTRACT

TABLE OF CONTENTS

ABSTRACT – ENGLISH iii

ABSTRACT – TAMIL v

LIST OF FIGURES ix

LIST OF TABLES x

LIST OF ABBREVIATIONS xi

1 INTRODUCTION 1

1.1 GENERAL OVERVIEW 1

1.2 PROBLEM DOMAIN 2

1.3 PROBLEM STATEMENT 3

1.4 OBJECTIVE 4

1.5 PROJECT DESCRIPTION 4

1.6 CONTRIBUTION 5

1.7 ORGANISATION OF REPORT 6

2 RELATED WORK 7

2.1 SDN ARCHITECTURE 7

2.2 LITERATURE SURVEY 8

3 REQUIREMENTS ANALYSIS 15

3.1 COMPONENTS OF SDN 15

3.2 FUNCTIONAL REQUIREMENTS 16

3.3 NON FUNCTIONAL REQUIREMENTS 17

3.3.1 Simulator 17

3.3.2 Floodlight Controller 18

3.3.3 Scalability 19

4 SYSTEM DESIGN 20

4.1 DESIGN OF THE SYSTEM 20

4.2 LINK DETECTION MODULE 21

4.2.1 Handling the failure links 21

4.3 PATH CALCULATION MODULE 23

4.3.1 Shortest Path Algorithm 23

4.3.2 The Debugger 23

4.4 RECOVERY MODULE 24

4.4.1 Compression Algorithm 24

4.4.2 Storing of dropped packets in switch 25

4.5 RULE PLACEMENT MODULE 26

5 SYSTEM DEVELOPMENT 28

5.1 SIMULATOR DESCRIPTION 28

5.2 CONTROL FLOW 29

5.3 IMPLEMENTATION 31

5.3.1 Link Detection Module 31

5.3.2 MACTracker Function 32

5.3.3 Program Flow Function 33

5.3.4 Path Calculation Module 33

5.3.5 The Debugger 34

5.3.6 Recovery Module 35

5.3.7 Rule Placement Module 37

6 RESULTS AND DISCUSSION 38

6.1 Results 38

6.1.1 Assessment 38

6.1.2 Evaluation 39

6.2 TEST CASES 41

6.2.1 Rule Space Overhead 41

6.2.2 Link Failure 41

7 CONCLUSIONS 42

7.1 GOALS ACHIEVED 42

7.2 FUTURE WORK 43

A Screenshots and Snippets of SDN 44

A.1 SCREESHOTS 44

A.1.1 NETWORK CREATION SIMULATOR 44

A.1.2 UPDATION OF RULES IN SWITCH 45

A.1.3 LINK LOSS 46

A.1.4 MACTracker with Network Topology 46

A.1.5 Ping ICMP Packets 47

A.2 Sample snippets for sdn 47

A.2.1 Concurrentcircularbuffer.java 47

A.2.2 Forwarding.java 50

REFERENCES 67

LIST OF FIGURES

2.1 Basic architecture of SDN 7
3.1 Components of SDN 15
3.2 Usecase diagram 17
4.1 Block diagram of the proposed system 20
4.2 Sequence diagram - Detection Module 22
4.3 Sequence diagram - Recovery Module 26
5.1 File Structure of the system 28
5.2 Control flow of the system 30
6.1 Formula for packet drop and restoration 39
6.2 Packet Distribution 40
6.3 packetdrop and restoration 40
6.4 Rules updates during link failure 41
A.1 simulation of network creation 44
A.2 Rules updated in switch 45
A.3 Shortest path routing 46
A.4 MACTracker and Network Topology 46
A.5 Ping the packets betwween the hosts 47

LIST OF TABLES

6.1 Packet distribution 38

6.2 Packet route through ports 38

6.3 Path traffic 39

LIST OF ABBREVIATIONS

SDN	Software Defined Networking
CDPI	Control to Data-Plane Interface
SNMP	Simple Network Management Protocol
ICMP	Internet Control Message Protocol
MSP	Multi Service Ports
LAN	Local Area Network
MAC	Media Access Control

CHAPTER 1

INTRODUCTION

1.1 GENERAL OVERVIEW

The aim of SDN is to provide open interfaces enabling development of software that can control the connectivity provided by a set of network resources and the flow of network traffic through them, along with possible inspection and modification of traffic that may be performed in the network.With SDN, the applications can be network aware, as opposed to traditional networks where the network is application aware (or rather, application ambivalent)Traditional (i.e. non-SDN) applications only implicitly and indirectly describe their network requirements, typically involving several human processing steps, e.g., to negotiate if there are sufficient resources and policy controls to support the application. The control plane is (1) logically centralized and (2) decoupled from the data plane. The SDN Controller summarizes the network state for applications and translates application requirements to low-level rules.This does not imply that the controller is physically centralized. For performance, scalability, and/or reliability reasons, the logically centralized SDN Controller can be distributed so that several physical controller instances cooperate to control the network and serve the applications. Control decisions are made on an up-to-date global view of the network state, rather than distributed in isolated behavior at each network hop. With SDN, the control plane acts as a single, logically centralized network operating system in terms of both scheduling and resolving resource conflicts, as well as abstracting away low-level device details, e.g., electrical vs. optical transmission.

SDN CONTROLLER

The SDN Controller has complete control of the SDN Datapaths, subject to the limit of their capabilities, and thus does not have to compete/contend with other control plane elements, which simplifies scheduling and resource allocation. This allows networks to run with complex and precise policies with greater network resource utilization and quality of service guarantees. This occurs through a well-understood common information model (e.g. as the one defined by OpenFlow). It is a logically centralized entity in charge of (i) translating the requirements from the SDN Application layer down to the SDN Datapaths and (ii) providing the SDN Applications with an abstract view of the network (which may include statistics and events). An SDN Controller consists of one or more NBI Agents, the SDN Control Logic, and the Control to Data-Plane Interface (CDPI) driver. Definition as a logically centralized entity neither prescribes nor precludes implementation details such as the federation of multiple controllers, the hierarchical connection of controllers, communication interfaces between controllers, nor virtualization or slicing of network resources.

1.2 PROBLEM DOMAIN

One major Problem in openflow networks is the Cost and energy needed for all the routers in the network and its functionality requirements.Another problem that SDN presents is that existing monitoring and management tools will need major updates to capture and correctly interpret network routing and traffic data. With SDN, networks will be much more dynamic; SNMP tools will be inadequate when networks are constantly changing automatically or programmatically. In order to adapt, network monitoring and management tools will need to be part of the SDN infrastructure to collect, analyze and present in real time the data that administrators and engineers need to understand the dynamics and make solid decisions about the network. Looking even further, softwarifying wide area networking presents interesting dilemmas. SDN doesnt only change the network but also the job descriptions of network engineers and administrators. With SDN Controllers, entire networks will be reconfigured on the fly (reconfiguring routers, adding paths, etc.) based on traffic conditions, redundancy requirements, and even iterative development to find the most efficient way of ensuring end-to-end performance. As a result, network engineers and administrators will spend less time planning for network changes to accommodate new or changed workloads and re-routing being done , which is a big part of network administration today. In theory, the network will handle that itself. Thus reducing time and space overhead.

1.3 PROBLEM STATEMENT

Our system is based on the process of detecting the link failure in a network and providing a secure path for the packets. As in most cases, the re-routing path provided is pre-computed already and is stored in the table. Even if its being the common trite , the networks in existence today are blindly dependent on this framework. The existing system uses a working and a transient plane. Working plane is used till the link failure occurs and then the transient plane comes into play when a link is failed.The transient plane is like a tree in which the failure node is the root.In SDN, we separate the control plane (which tells data where to go) from the data plane (which forwards traffic to the next node).

We have physical routers and other infrastructure, but theyll become commodity forwarding devices with the control plan intelligence residing in a server the SDN controller. This enables us to create virtual network overlays and functions.We will highlight how SDN contributes to network innovation by various networking applications that were implemented and analyzed in recent years. These applications fall in the areas of network management and traffic engineering, security, network virtualization and routing optimization. Several surveys of SDN already exist; this contribution differs from them through an analysis of architectural design choices for OpenFlow-based SDN networks. We discuss performance issues of OpenFlow-based SDN and refer to studies regarding these aspects.

1.4 OBJECTIVE

The objective of the project is:

- Detection of link failures
- Recovery from failure of links
- Rule placement in switches
- Shortest path calculation within the LAN
- Managing rule space overhead

1.5 PROJECT DESCRIPTION

SDN play a major role in error detection and load balancing in network traffic between Multi Service Ports (MSPs). An MSP allows a number of network services, such as IP connections and light paths, to be set up on one physical port. If theres a failure in the normal MSP, network services remain available thanks to the secondary MSP.

Our system makes use of SDN to provide an optimal link failure detection and recovery .Error detection is traditionally arranged via link aggregation. This is an IEEE 802.1ax technology, which uses the Link Aggregation and Control Protocol (layer 2). To detect errors, link aggregation checks whether a connection is live by sending data packages from one end point to the other, and by checking whether they are returned by the other end point. Load balancing and error detection are employed to make optimal use of the primary and secondary MSPs. Load balancing distributes the traffic evenly across the two MSPs. In the event of a link break, for instance, error detection ensures that a network connection can quickly be rerouted via network connections that are still working. As a result, institutions always have access to their network services.
SDN calculates the shortest path between the source and the destination within the LAN using the shortest path algorithm and ports calculation algorithm. Then the rules are updated in the switches along the path. If any link along the shortest path calculated goes down, the controller is notified and the next optimal path is calculated to transmit the packet to the destination.Every time before updating rules in the switches, the number of rules already present is calculated. If it exceeds the threshold (maximum number of rules), few rules are flushed before installing new rules.

1.6 CONTRIBUTION

This project has its contribution in :

- Shortest path Algorithm: Takes the destination host as input and formulates the shortest path..
- Compression Algorithm: This algorithm consists of replacing

large sequences of repeating data(packets) with only one item of this data(packets) followed by a counter showing how many times this item is repeated and storing the packets in the switch

1.7 ORGANISATION OF REPORT

Chapter 2 of the report describes the basic architecture of SDN, the OpenFlow protocol and work related to the project. Chapter 3 explains the functional and nonfunctional requirements of the projects and the various components of the project. Chapter 4 is the system design which explains the various modules in the project and the algorithms used. Chapter 5 describes the structure of the simulator used and implementation details of the system. Chapter 6 explains the performance evaluation and various test cases that the system satisfies. Chapter 7 is the conclusion, future work and the limitations of the project.

CHAPTER 2

RELATED WORK

This chapter focuses on the architecture of SDN and the work that has been done in the field of Software Defined Networking. It discusses the various applications of SDN that has been deployed so far.

2.1 SDN ARCHITECTURE

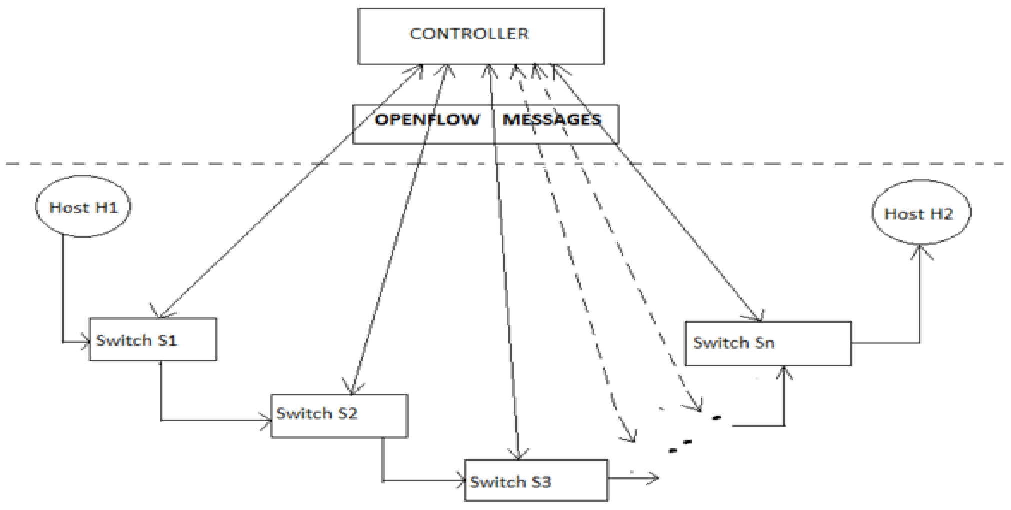


Figure 2.1 Basic architecture of SDN

Software Defined Networking helps in programming a centralized controller that performs functionalities similar to a control plane in a traditional network. The centralized controller in SDN communicates with the switches through the OpenFlow messages. OpenFlow messages provide access to the forwarding plane of a switch or router over the network. The OpenFlow protocol [8] sends out messages from the controller to place rules in the switch. Every switch in the network is connected to the controller in addition to being connected to the adjacent switch. There can be numerous switches connected between two hosts. The centralized controller is designed in such a way that it has information on the entire network. A new packet arriving at the switch is forwarded to the destination host according to the rules that the controller places in the switches. Every switch checks its connectivity with its neighbouring nodes and reports to the controller the network topology.

2.2 LITERATURE SURVEY

Carrier-grade Ethernet networks, industrial area networks and some local area networks (LANs) have to provide a resilient spring back in case of a network failure. Open Flow Architecture is generally based on Ethernet cables. Thus link failure in these networks are common. Thus in OpenFlow-Based Segment Protection in Ethernet Network, the Openflow architecture is enhanced to support segment based routing algorithm for efficient transfer of messages if there are link failures. This mechanism is efficient in choosing the backup path (secondary path) in case of an intermediate link failure. The chosen (secondary) path is calculated within few tens of milliseconds. The above method takes in loads of memory space and is also considered to be inefficient when it comes to memory handling. Also, the time taken to recover is approximately 1ms and that is quite high [2]

Moreover, switches are essential for forwarding the packets in a local area network and if a switch fails, then the packets are not able to reach their destination. The new trend in Software Defined Networking (SDN) has made the use of software switches quite popular. These Software switches are required to be resilient to failure. This Fault-Tolerant

OpenFlow-based Software Switch Architecture with LINC Switches for a Reliable Network Data Exchange explains one mechanism for fault tolerance of LINC (Link Is Not Closed), an open source OpenFlow switch, which is written in Erlang programming language. We leverage the built-in concurrency, and fault-tolerant features of Erlang to realize a fault-tolerant distributed LINC switch systemThe LINC switches took only a little more than half the time the hardware switches took to connect hosts to the fault-tolerant system. When failover happens, the controller modifies the flow entries in the LINC switches which causes the packets to be sent to the new switchs input port almost instantaneously. Erlang system shows some ease of programmability and faster deployment. The future Work is that by making efficient use of the Erlang Distributed System, the Fault Tolerant System can be improved further [6].

It provides alternatives for both the scalability and resilience issues in OpenFlow Networks. OpenFlow switches store their flow tables in expensive, limited Ternary Content-Addressable Memory (TCAM) due to which the stored tables cannot be large, thus forwarding packets in line speed. Most resilience mechanisms require additional entries thus the implementation in OpenFlow may quickly exceed the available TCAM. Loop-Free Alternates (LFAs) are a standardized mechanism for fast reroute in IP networks which do not require additional entries. However, some LFAs cause loops when some node or multiple nodes failures occur. This renders additional links unusable. But if we were to exclude such LFAs, it would reduce the protected destination coverage even further. To overcome this, a scheme is designed to detect the loops caused by LFAs. This maximizes the protection coverage because the LFAs can be used without creating loops. This paper describes how

LFAs and the loop detection scheme can be implemented in OpenFlow networks with only little packet overhead and a single additional entry per switch. They are simple and have no additional forwarding entries. And had no flow tables. The ternary content-addressable memory was used to store the flow tables. The TCAM is very expensive. Hence, here, its cost is also reduced thus giving maximized protection.Some problems not dealt here are that they mostly cannot protect all traffic and some of them cause micro-loops in case of node failures or multiple failures. Loop detection just helps to prevent loops for LFAs, but it cannot protect traffic for which no LFAs exist[10]

The three layers of the Orion model provide the hybrid hierarchical control layer for large scale networks. The Orion has the area controller which is responsible for collecting physical device information and link information, managing the intra-area topology and processing intra-area routing requests and updates. The domain controller synchronizes the global abstracted network view through a distributed protocol. The Domain Routing Management Sub-Module of the domain controller computes the global shortest path. When a PacketIn message reaches the area controller, the area routing management sub-module checks the source address and destination address of the message. If the destination address is in the area, the area controller employs Dijkstra algorithm to compute intra-area path and if is not present in the area, the area controller sends the source address and the destination address of the message to the domain controller, and stores the message to a waiting buffer with index. The domain controller computes the routing path for the flow and sends the routing result to the area controller The CPU utilization of the domain controller is 40%. The domain controller of Orion costs lower than an OpenFlow based SDN controller and the load of the area controller of Orion is between the load of the controller in the flat architecture and the lowest area controller in the abstracted hierarchical SDN architecture. One disadvantage is the memory overhead due to the maintenance of area and domain controllers[7] During the data transmission, there may occur many number of the data failures in the data path. Segment protection is the key feature used to reroute the data in the secondary route which may or may not be the best or optimal path. Independent Transient Plane (IPL) is designed in Europe-wide demonstration of fast network restoration with OpenFlow which reduces the path complexity and maintains the security of the data. This work results in the most efficient protection in the secondary path and optimal solution to the data failure problem. One advantage here is the designed mechanism deals with the data packets in all paths simultaneously. The data packets which are lost during the transmission in the malicious path are gained by this designed. In some cases those data are sent to the destination in the legitimate path in retransmission phase but the regained data packets are not considered as legitimate; therefore there is a possibility of malicious data present in that regained data [4]

Fast Recovery in Software-Defined Networks implements a failover scheme with per-link Bidirectional Forwarding Detection sessions and preconfigured primary and secondary paths computed by an OpenFlow controller It uses the usual way of detecting failures in Ethernet networks like if an acknowledgement is not received within 50-150ms , then the link is said to be broken. Uses two steps of process The first step involve a fast switch-initiated recovery based on preconfigured forwarding rules guaranteeing end-to-end connectivity. The second step involves the controller calculating and configuring new optimal paths.A lower detection time due to decreased session round trip time (RTT) with the removal of false positives. As each session spans a single link, false positives due to network congestion can be easily removed by prioritizing the small stream of control packets. But the problem is that the memory is not efficiently handled since the flow table should store two ways of possible communication and Redundant routing information is stored in the group tables[5, 13].

Detour Planning for Fast and Reliable Failure Recovery in SDN with OpenState provides a secure and reliable path if a particular node or a link fails. A protection scheme is given in this paper which calculates the backup paths in prior and the routing used here is MPLS which ensures zero packet loss after the link failure is detected. Also in this paper, the forwarding rules are done autonomously without the use of the controller always. The recovery is done with the help of tagged indexes making the routing simpler. The centralized controller is not frequently accessed since few forwarding rules are done independently and no packets are lost. However this mechanism cannot be expanded to multiple link failures. Also if the tag indices are lost, then the packet drop numbers will increase [8, 14]

Openflow Path Fast Failover Fast Convergence Mechanism deals with a fast and efficient failover mechanism for redirecting traffic to more optimal backup path when there is a link failure or congestion problem in SDN. It also proposes a local pre-calculated path dataset mechanism in OpenFlow controller to allow fast network convergence. The central OpenFlow Controller computes the main and the best backup path based on the current network topology. OpenFlow controller is said to have a local dataset of path information and in case of a link failure or congestion in a path, the affected switch sends port-status message to the controller and the controller checks the flow entries affected by the failure. The controller pushes the main and the backup path to the OpenFlow switches and will recalculate the less congested backup path after it is updated periodically by the network. Once the controller get the notification about a link failure, it will perform simple lookup in its local dataset to find whole flow entry that affected by the failure. Finally, the affected entries will be deleted from the flow table and the pre-computed less congestion backup path will be selected. The controller then updates the flow entries of all switches and incorporates the new backup plan. The single backup path for every main path in only one single switch flow table reduces the possibilities of flow table explosion and the network traffic is redirected to alternate optimal path. But one drawback is the memory overhead due to the recalculation of less congested path in the controller[9, 11].

One efficient re-routing is the Automatic Re-routing with Loss Detection architecture (ARLD) proposed in Efficient routing for traffic offloading in Software-defined Network paper works on the assumption that if a packet loss occurs at a link , it is mainly due to congestion. The controller treats this link as a bottleneck link. The Openflow protocol has a stats message which delineates the status of each node and each port to the SDN controller. The Re-routing module of the SDN controller computes an alternate path as a bypass route. The Re-routing module updates the virtual topology by eliminating the node at which the packet loss occurred and finds an alternative route without the switch that dropped the packet. After examining the availability of the path, the module returns the alternate route to controller and controller distributes new forwarding rules to each switch and updates the flow table. In this way the packet loss at the switches due to congestion can be reduced to yield better performance. With the help of an Openflow based SDN architecture , the controller detects the packet loss at switches in a shorter time and the model reduces the packet loss by providing a better performance. However when dropped packet is routed to bypass route the traffic on that route may cause congestion on that link. This effect would be a serious problem in larger sized network. Also, In the proposed architecture, the average loss rate is reduced and this might decrease average latency of each flow, although measuring delay is not conducted due to the low timing realism of mininet emulation [3].

IP Fast Rerouting here provides us the methodology employed to recover from link failures in a network with the help of Relaxed Multiple Routing Configuration (rMRC). The MRC and the rMRC guarantee link or node failure from biconnected topologies. Backup topologies can be constructed using different methods and the number of states required in a router will increase with multiple backup paths. In the rMRC the requirements of the network topologies are relaxed. The difference between rMRC and MRC is that in conventional MRC the instantaneous recovery from the node failure is done by isolating the affected nodes. Instead, rMRC computes the shortest path without the failed link in the backup topology where the detecting node itself is isolated. Using the backup topology where the detecting node is isolated ensures that the traffic cannot loop back to the detecting node but still enables the rMRC forwarding to reach the destination node. The presented algorithm can guarantee link and node fault tolerance with fewer backup topologies than MRC. As relaxed backup topologies do not isolate all links, there is more flexibility in rMRC than in MRC to decrease the number of backup topologies. However the problem here is rMRCs ability to spread traffic over more links can sometimes have a dramatic impact in a sparsely connected network topology[1, 12].

CHAPTER 3

REQUIREMENTS ANALYSIS

This chapter describes the various components of the system and the functional and nonfunctional requirements of the system.

3.1 COMPONENTS OF SDN

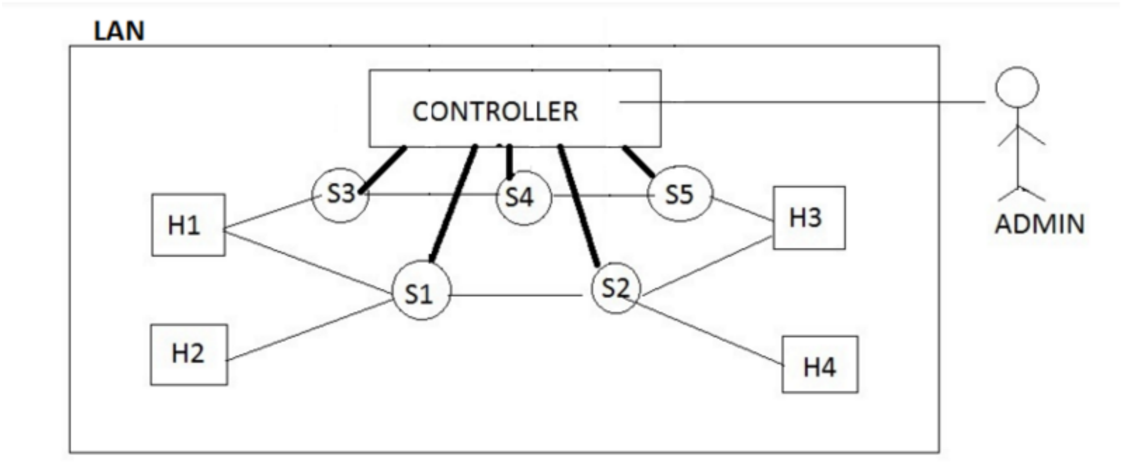


Figure 3.1 Components of SDN

The description of the components of SDN are given in detail as follows:

H1...H4 : Hosts within the LAN

S1...S5 : Switches within the LAN with OpenFlow support

Controller: The centralized controller that controls the switches within the network.

Admin: The one who configures the controller based on the requirements

The switches are connected to the adjacent switches and the hosts nearby. All the switches are monitored by the SDN controller(Floodlight) .The Controller is designed to work with the growing number of switches, routers, virtual switches, and access points that support the OpenFlow standard.

3.2 FUNCTIONAL REQUIREMENTS

The various functions performed by the system are,

1. Detecting the failure of links and handling the packet loss.
2. Storing the packets in switch to overcome drop of packets.
3. Calculating the shortest path if multiple paths are present.
4. Placing rules in the switches accordingly along the path calculated.
5.Flushing rules in the switches within the openflow table in case of overhead.

The use case diagram depicts the various functional requirements of the proposed system, as shown in Figure 3.2. The actors are user, controller and switch.
The user should provide the source and destination nodes to forward the packets between them.Based on the destination address, the request is classified at the first switch of the network. If all the links between the hop is not broken, then the packets can be sent successfully without loss.If any of the links between the hops are broken, then controller will have to find the next alternate path and transmit the packets without loss.Once the alternate path is found, rules are being placed accordingly in the switches, which is done after calculating the shortest path between the source and destination.The alternate path calculated by the controller intimates the controller that the link has broken so the packets which are started from the source has to be stored in the switch and packets which hasnt started from source are forwarded by alternate path to achieve the

destination. The packets which are stored in the switch is done by having the buffer limit in the switch to tell the controller that stored packets in the switch has to be transmitted to the destination by finding the next shortest path.The debugging module for calculating the shortest path is used to find another optimal path between the source and destination, if any link along the first optimal path is down. The cover and replace is used to remove rules from switches if they get overloaded and exceed the threshold.

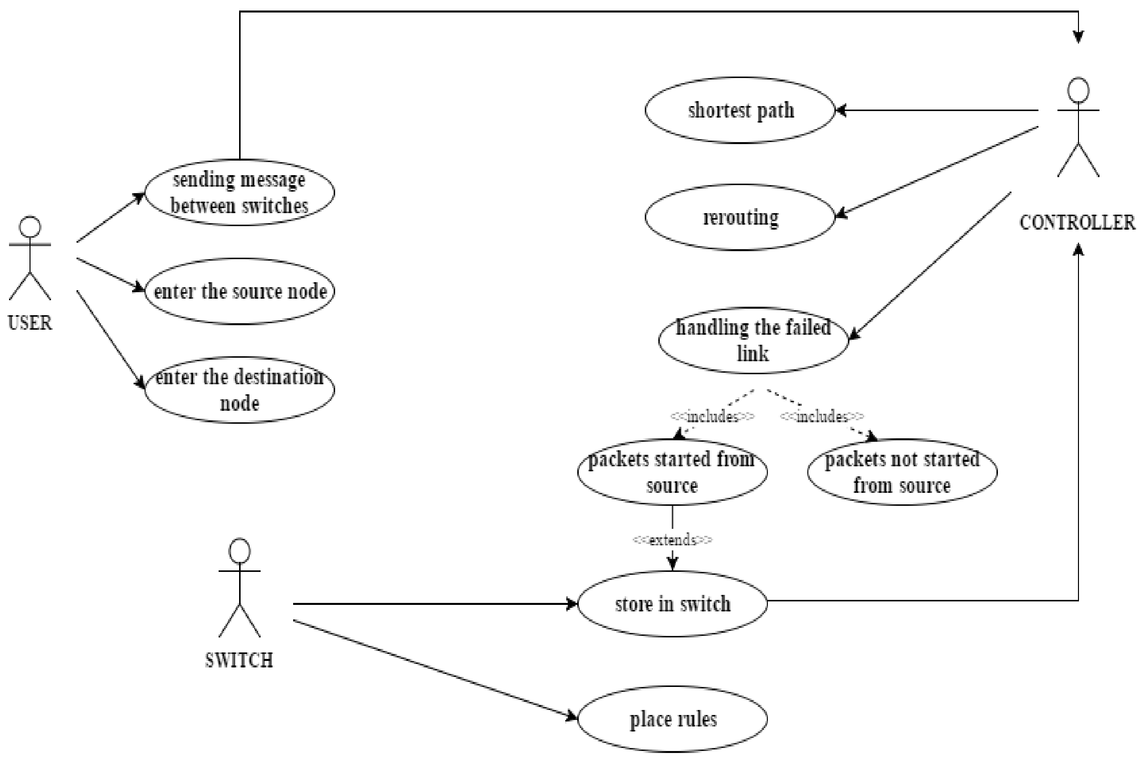


Figure 3.2 Usecase diagram

3.3 NON FUNCTIONAL REQUIREMENTS

3.3.1 Simulator

The system can either be deployed into a real time network or simulated with the help of a simulator. This project aims at simulating SDN and perform access control using it. A simulator that can be used should support a centralized controller program that communicates with the switches connected to it with the help of OpenFlow Protocol. The switches used should be openflow switches, to support openflow messages. One simulator that has these features is MININET and is used in this project.

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking. The mininet VM includes all OpenFlow binaries and tools pre-installed, and tweaks to the kernel configuration to support larger Mininet networks. The mininet VM is run using the VMware workstation. The VMware workstation enables users to set up one or more virtual machines on a single physical machine, and use them simultaneously along with the actual machine. Each virtual machine can execute its own operating system.

3.3.2 Floodlight Controller

The Floodlight Open SDN Controller is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller. It is supported by a community of developers including a number of engineers from Big Switch Networks. OpenFlow is a open standard managed by Open Networking Foundation. It specifies a protocol through switch a remote controller can modify the behavior of networking devices through a well-defined forwarding instruction set. Floodlight is designed to work with the growing number of switches, routers, virtual switches, and access points that support the OpenFlow standard.

3.3.3 Scalability

The system makes use of a controller, to control switches within a LAN. The simulator, Mininet supports 4096 switches. However the topology used has 5 hosts and 3 switches with all the nodes reporting to a single controller within a LAN.

CHAPTER 4

SYSTEM DESIGN

This chapter deals with the various modules used in the project and their functionalities. It gives an idea about the inputs and outputs of the modules and the usage of the components in the working of the system.

4.1 DESIGN OF THE SYSTEM

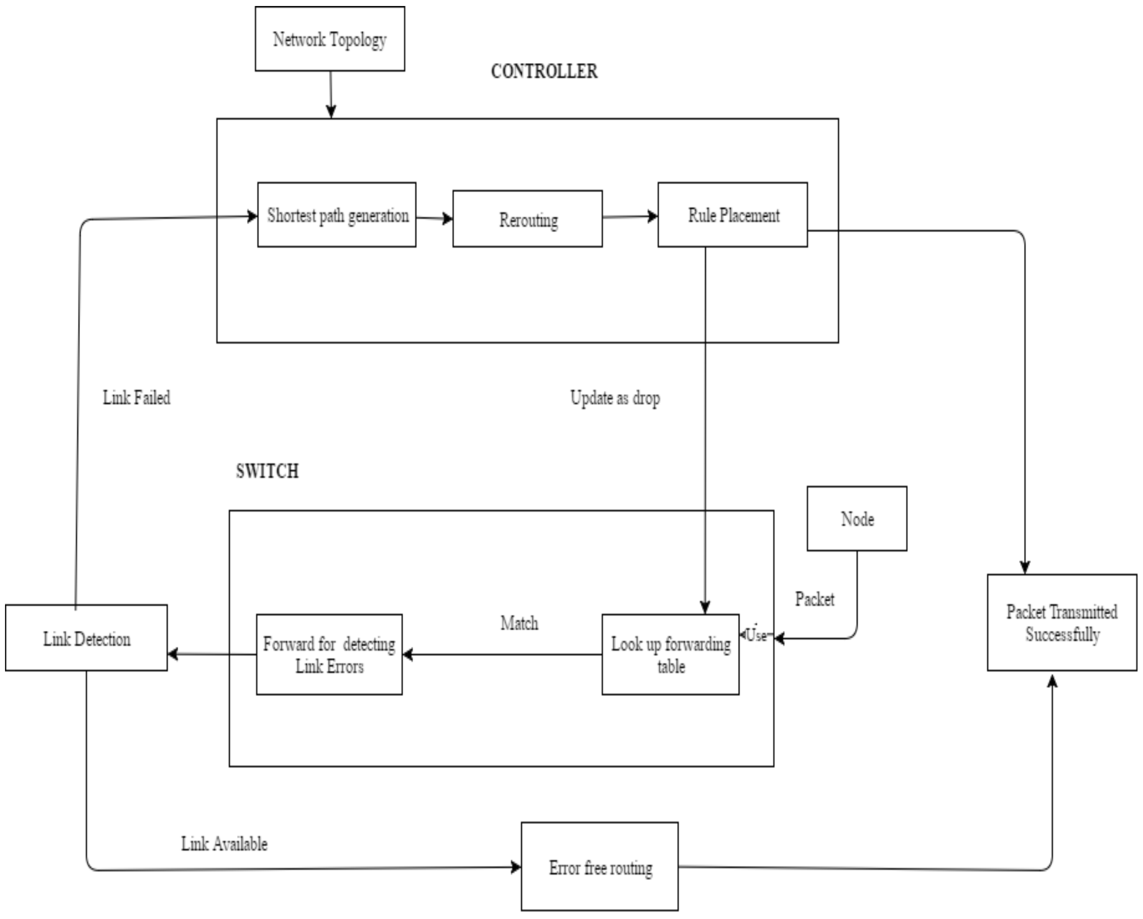


Figure 4.1 Block diagram of the proposed system

It depicts the various modules that are involved in the system. When a packet arrives at a switch, a look up of the forwarding table is done. In case of a match, the packet is forwarded to the Link Detection module. Otherwise the packet is send to the controller. In case of a there is no link failure between the links,packets can be transmitted without any loss. If there happens the link failure then controller has to find the optimal path using Path Calculation module to provide the optimal path between the destination and the source. The Rule Placement module helps in avoiding overloading of the switches.
The proposed system consists of 4 modules, namely,

- Link Detection Module.
- Path Calculation Module
- Failure recovery Module
- Rule-Placement Module.

4.2 LINK DETECTION MODULE

Link between nodes is considered to be the framework for any network and so preserving the link is cardinal. If the network link is error free ,it reroutes and finally transmits the packets without any loss. If a link failure is detected, the nodes between the failed link is identified. After the detection of link failure, input is then passed to the recovery module,and the openflow protocol intimates the SDN controller.

4.2.1 Handling the failure links

When a packets reaches a switch, it checks if there is a rule in its forwarding table to reach the destination. In case of a miss, the packet is forwarded to the controller. The controller then invokes the path calculation module and uses the rule placement module to update the rules in the switches. When the rules are updated in the switch, user can now send packets to the destination without packet loss.In case if there happens a link failure between the hop by finding the loss of packets is greater than the buffer limit, switch will intimate the controller that packets which are not started from source has to be stopped and controller will find the next optimal path for those packets.

INPUT: Source, destination, network topology

OUTPUT: Finding out the link which has broken and transmit the packets to the destination through alternate path which is robust.

PROCESS: Link failure discovery, Rerouting

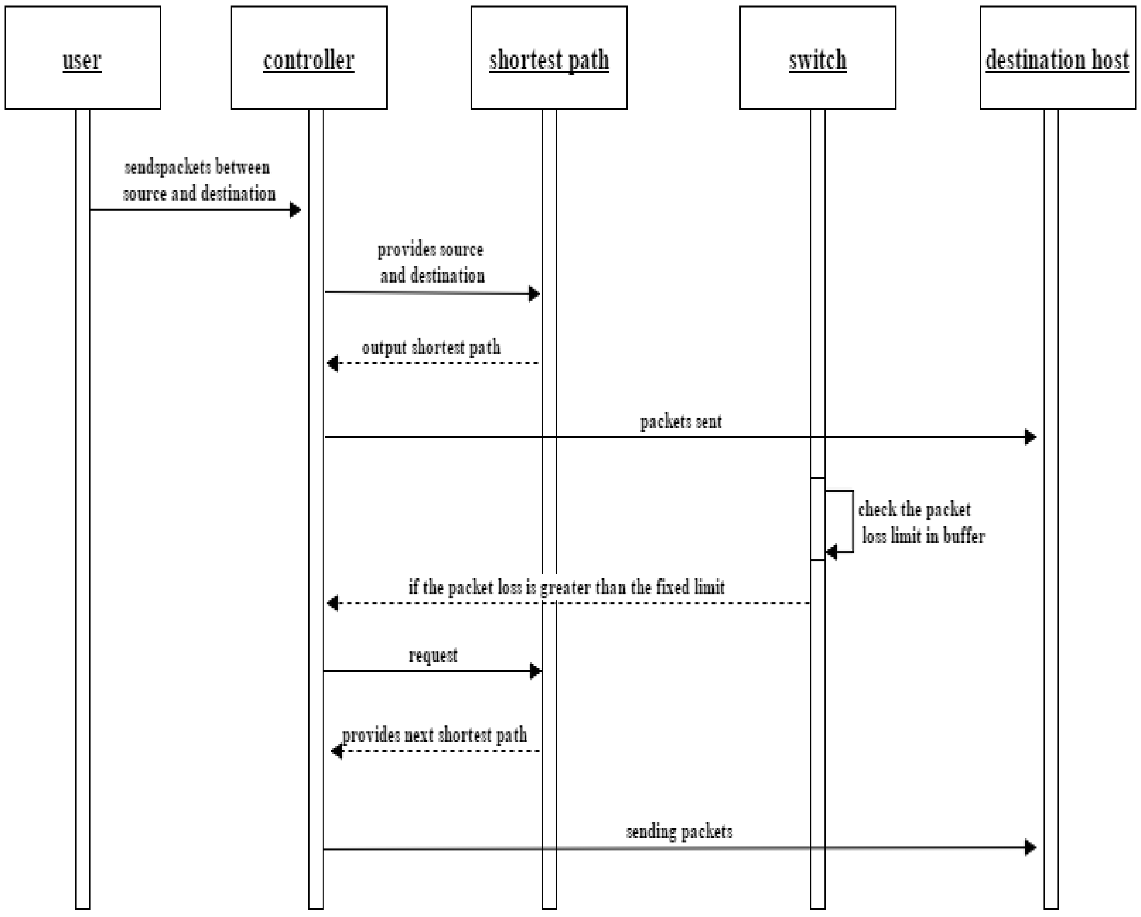

user
controller
shortest path
switch
destination host
sendspackets between source and destination
provides source and destination
output shortest path
packets sent
check the packet loss limit in buffer
if the packet loss is greater than the fixed limit
request
provides next shortest path
sending packets

Figure 4.2 Sequence diagram - Detection Module

4.3 PATH CALCULATION MODULE

The module helps in calculating the optimal path between two hosts within the same network. To implement this, two algorithms have been used, namely,

- The shortest path algorithm
- The debugger.

4.3.1 Shortest Path Algorithm

This algorithm computes the best path(shortest) to reach the destination host within the network, when provided with the source, destination and the graph topology as inputs

4.3.2 The Debugger

Whenever a link, which is being used by a user to connect to a destination host, is down, the access to the destination host is denied to the user . In order to avoid this, the debugger module checks if all the links along the path generated by shortest path algorithm are connected and only then runs the rule placement algorithm. If it finds that some link along the path is down, it immediately calculates the next optimal path within the network and updates the rules in the switches along the alternate path so that the user is always able to sent the packets to the destination.

INPUT: Source, destination, network topology
OUTPUT: Optimal path between source and destination hosts within network.
PROCESS: Shortest path calculation,debugging.

4.4 RECOVERY MODULE

Storing of dropped packets in compression state.Handling the failed link using the alternate back path (the packets which are not left from the source).Stored packets are sent using fast rerouting technique.Packets that are not left from the source are rerouted in pre-calculated alternate path.Packets that are transmitted before the detected failed link are stored in the switch in the compressed state.The stored packets are then decompressed and fast rerouted to the destined node.SDN controller controls all the rerouting paths using Dijkstras algorithm.

4.4.1 Compression Algorithm

Data compression reduces the size of data frames to be transmitted over a network link. Reducing the size of a frame reduces the time required to transmit the frame across the network. Data compression provides a coding scheme at each end of a transmission link that allows characters to be removed from the frames of data at the sending side of the link and then replaced correctly at the receiving side.The most commonly used compression algorithm is Run length encoding compression.

Run length encoding is probably the easiest compression algorithm there is. It replaces sequences of the same data values within a file by a count number and a single value. It consists of replacing large sequences of repeating data(packets) with only one item of this data(packets) followed by a counter showing how many times this item is repeated and storing the packets in the switch.

4.4.2 Storing of dropped packets in switch

When a switch receives a packet, it parses the packet header, which is matched against the flow table. If a flow table entry is found where the header field wildcard matches the header, the entry is considered. If several such entries are found, packets are matched based on prioritization, i.e., the most specific entry or the wildcard with the highest priority is selected. Then, the switch updates the counters of that flow table entry. Finally, the switch performs the actions specified by the flow table entry on the packet, e.g., the switch forwards the packet to a port. Otherwise, if no flow table entry matches the packet header,switch generally notifies its controller about the packet, which is buffered when the switch is capable of buffering.
To that end, it encapsulates either the unbuffered packet or the first bytes of the buffered packet using a PACKET-IN message and sends it to the controller; it is common to encapsulate the packet header and the number of bytes defaults to 128. The controller that receives the PACKET-IN notification identifies the correct action for the packet and installs one or more appropriate entries in the requesting switch. Buffered packets are then forwarded according to the rules; this is triggered by setting the buffer ID in the flow insertion message or in explicit PACKET-OUT messages. Most commonly, the controller sets up the whole path for the packet in the network by modifying the flow tables of all switches on the path.

INPUT : Buffered packets, Source , Network topology
OUTPUT: It will store the packets which have started from source and buffer the packets in the switch in the compressed state.
PROCESS: Storing the packets , finding of alternate path.

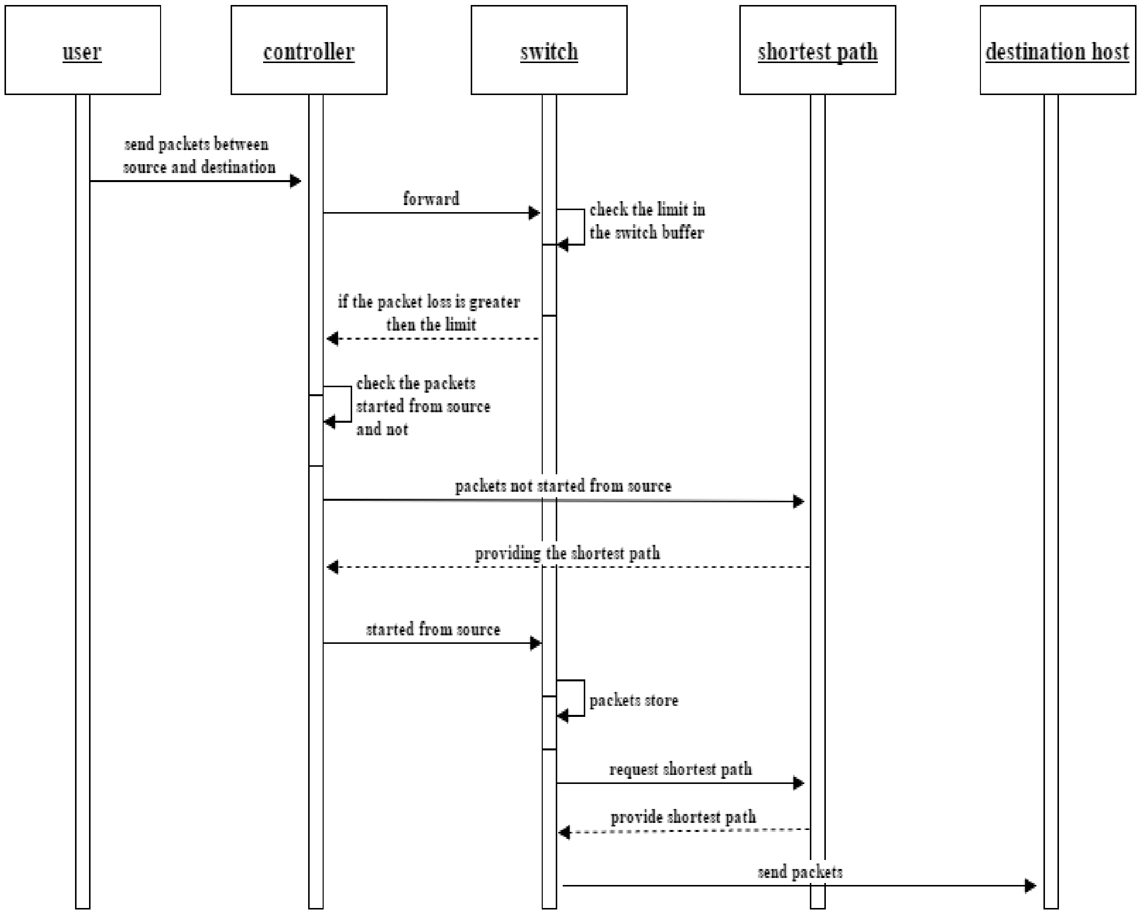

user
controller
switch
shortest path
destination host
send packets between source and destination
forward
check the limit in the switch buffer
if the packet loss is greater then the limit
check the packets started from source and not
packets not started from source
providing the shortest path
started from source
packets store
request shortest path
provide shortest path
send packets

Figure 4.3 Sequence diagram - Recovery Module

4.5 RULE PLACEMENT MODULE

The rule placement module is concerned about the placing the appropriate rules in corresponding switches along the path to the destination in an efficient manner such that the switch does not get overloaded by rules (beyond rule space) at any instant of time . To enable this, the module makes use of the cover and replace algorithm, routing policy and the rule placement algorithm. The routing policy and the rule placement algorithm together make sure that the packet reaches the destination through the path specified by the shortest path algorithm and thus ensuring that the network obeys the endpoint policy E , forwards the packets over the paths specified by R.

To further improve upon the efficiency and to ensure that the rules in the switch do not exceed the rule space, the cover and replace algorithm is used. This algorithm checks if the rules in any switch exceeds the rulespace. If any, then it tries to minimize the number of rules in the overloaded switch. It places only the recently used rules in the overloaded switch and transfers the rest to another idle switch along the path . A common rule is written in the overloaded switch to seek the idle switch(the one to which the rules are transferred) in case of need.

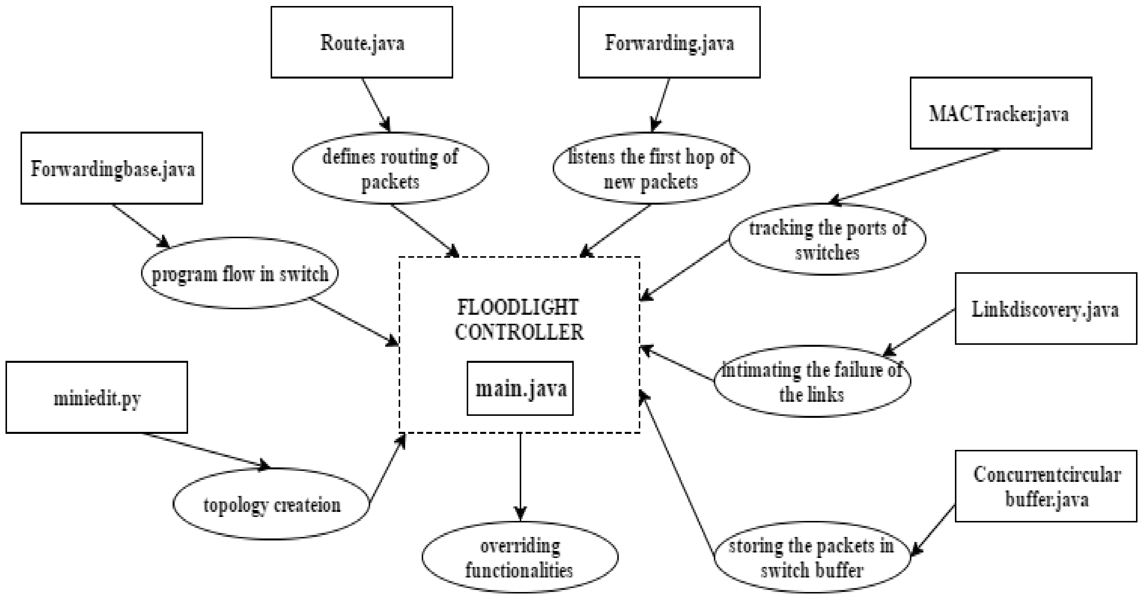
INPUT: Endpoint policy, Routing policy, Network topology and the maximum number of rules each physical switch can hold.
Endpoint policy (E) :It views the entire network as one big switch. The policy specifies which packets to drop or to forward to specific egress port as well as any modification of specific header fields.
Routing policy(R) :The routing policy specifies what paths traffic should follow between the ingress and egress ports.
OUTPUT: Efficient placement of rules in the switches.
PROCESS: Rule placement algorithm and Cover and replace Algorithm.

CHAPTER 5

SYSTEM DEVELOPMENT

The chapter describes the implementation of the project. It helps in understanding the structure of the project in detail with the help of simulator structure and code snippets.

5.1 SIMULATOR DESCRIPTION



Route.java
Forwarding.java
MACTracker.java
Forwardingbase.java
defines routing of packets
listens the first hop of new packets
tracking the ports of switches
program flow in switch
FLOODLIGHT CONTROLLER
main.java
Linkdiscovery.java
intimating the failure of the links
miniedit.py
topology createion
Concurrentcircular buffer.java
overriding functionalities
storing the packets in switch buffer

Figure 5.1 File Structure of the system

Software defined networking can be simulated through various simulators. Each of these simulators help in creating a virtual network and user defined controller. Thus mininet is one such network simulator that adheres to the requirements of the project and helps in having a better understanding of the network technology. It has pre-defined packages installed that helps it to simulate software defined networks

(SDN) . The floodlight controller behaves as a remote controller that helps in handling the functions of the control plane. The controller can be programmed according to the administrators need.
Figure 5.1 defines the structure of mininet. A network of hosts and switches can be created with the help of miniedit.py python file that is pre-installed in mininet.. The IP address and the MAC address of every host is retrieved from the MACTracker.java file that includes numerous functions describing the routine of various controllers and different types of switches. Every switch is converted to a learning switch that is used in SDN networks. The main.java file contains the functions of the floodlight controller. Thus main.java file imports all these files and overrides the functionalities of the controller according to the needs of the network administrator. The Forwarding.java file also contains commands to update rules to the switch.

5.2 CONTROL FLOW

Figure 5.2 depicts the control flow of the entire project. The network topology is the input to the overall proposed system. The simulator, Mininet, creates a network topology given the hosts and the switches. The simulator provides pre-defined functions for network creation. The switch in the path of the client and server, performs a look-up in its forwarding table. When there is no rule in the switch, the packet is sent to the controller. The controller performs the shortest path algorithm to determine the optimal path for packet transmission. If the link between the switches are not broken, then packets will send successfully without loss. In case of a failure link, the first switch in the path updates the rule. When the rules are updated in the switch, user can now send packets to the destination without packet loss.In case if there happens a link failure between the hop by finding the loss of packets is greater than the buffer limit, switch will intimate the controller that packets which are not started from source has to be stopped and controller will find the next optimal path for those packets. The packets which are started have to go through the path of the failed link and the buffer in the switch will compress all the packets and controller will find the next optimal path and send the packets to the destination .

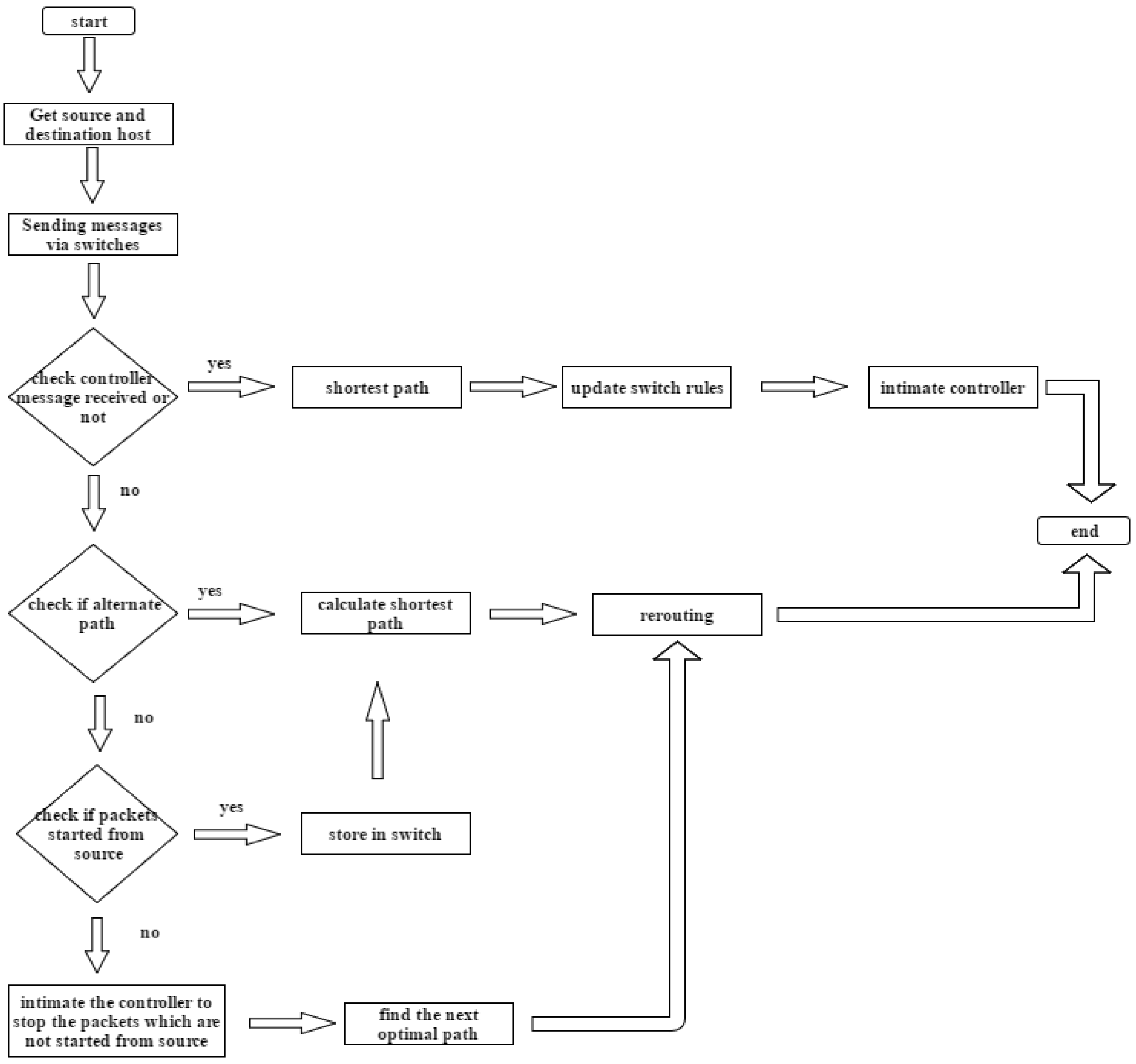

start
Get source and destination host
Sending messages via switches
check controller message received or not
yes
shortest path
update switch rules
intimate controller
end
no
check if alternate path
yes
calculate shortest path
rerouting
no
check if packets started from source
yes
store in switch
no
intimate the controller to stop the packets which are not started from source
find the next optimal path

Figure 5.2 Control flow of the system

The rule placement algorithm ensures that every switch is not overloaded with rules at any point in time. Even a single match makes the controller update the rule to all the switches along the path. However in case of no match, the controller updates the switches along the path with a permit rule. There are 4 modules that the project focuses on :

- Link Detection Module
- Path Calculation Module
- Recovery Module
- Rule-Placement module

5.3 IMPLEMENTATION

5.3.1 Link Detection Module

Error detection is traditionally arranged via link aggregation.Load balancing distributes the traffic evenly across the two MSPs. In the event of a link break, for instance, error detection ensures that a network connection can quickly be rerouted via network connections that are still working. As a result, institutions always have access to their network services.The packets which are started have to go through the path of the failed link and the buffer in the switch will compress all the packets and controller will find the next optimal path and send the packets to the destination .

pseudo code to detect and handle link failure

```
for each switch in a network
    sending & receiving ICMP packets to nearby switches
    IF (link failure detected)
        identify the failed link between switches
        link array[] = store the failed link
        initialize link flag bit =1
        IF (packet loss > buffer limit in switches)
            stop the packet transmission from the source.
            pass the packet flow to recovery module
```

```
      modify the routing table and initialize the flag bit
         =1.
      intimate the SDN Controller.
   ELSE
      re routing the packets without loss.
```

5.3.2 MACTracker Function

A network is created of mininet and the IP address and the MAC address of every host is retrieved from the MACTracker file.It includes numerous functions describing the routine of various controllers and different types of switches. Every switch is converted to a learning switch that is used in SDN networks.

Function to track the ports of the switches

```
public net.floodlightcontroller.core.IListener.Command receive(
           IOFSwitch sw, OFMessage msg, FloodlightContext cntx) {
        Ethernet eth =
                  IFloodlightProviderService.bcStore.get(cntx,
                     IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
          Long sourceMACHash =
             eth.getSourceMACAddress().getLong();
           if (!macAddresses.contains(sourceMACHash)) {
              macAddresses.add(sourceMACHash);
              logger.info("MAC Address: {} seen on switch: {}",
                     eth.getSourceMACAddress().toString(),
                     sw.getId().toString());
           }
           return Command.CONTINUE;
        }
```

5.3.3 Program Flow Function

In an OpenFlow SDN, when a switch receives a packet on a port, it will try to match the packet to a flow entry in the switch's default flow table. If the switch cannot locate a flow that matches the packet, it will by default* send the packet to the controller as a packet-in for closer inspection and processing.

Function to operate the first entry packets

```
public Command receive(IOFSwitch sw, OFMessage msg,
    FloodlightContext cntx) {
        switch (msg.getType()) {
        case PACKET_IN:
            IRoutingDecision decision = null;
            if (cntx != null) {
                decision = RoutingDecision.rtStore.get(cntx,
                    IRoutingDecision.CONTEXT_DECISION);
            }
        return this.processPacketInMessage(sw, (OFPacketIn) msg,
            decision, cntx);
        default:
            break;
        }
        return Command.CONTINUE;
    }
```

5.3.4 Path Calculation Module

The module helps in calculating the optimal path between two hosts. The path is calculated with the help of shortest path algorithm and ports calculation algorithm. A test case that is implemented in this module is Debugger function.

Inputs to the module are two variables, src and dest. The source host is stored in the variable src and the destination host is stored in the variable dest.

The shortest path algorithm

The variable graph contains the network topology defined by the network administrator. The variable src is stored in the variable start and the variable dest is stored in the variable end. The output is the shortest path between the start and the end nodes. The output is stored in the variable tmp-path.

Algorithm to find the shortest path in a network

```
def spath(graph,start,end,path=[]):
    temp-path=[start]
    add tmp-path to queue
    while q is not empty
        dequeue q to tmp-path
            last-node=tmp-path[len(tmp-path)-1]
            print tmp-path
        if last-node == end
            tm-path is a valid path
        for link node in graph[last -node]
            if link node not in tmp-path
                newpath = tmp-path+ link-node
                q.enqueue(new path)
return tmp-path
```

5.3.5 The Debugger

This algorithm is a test case to the project. The algorithm helps in calculating the next optimal path between the same pair of source and destination hosts when any of the links along the shortest path calculated is down. The source and the destination stored in the variables src and dest respectively are given as input to the shortest path algorithm. All possible path between the source and destination is calculated using the shortest path algorithm. When a link is down, traversed to get the next optimal path length. The corresponding path is traversed in the spath file. A pattern match is done and the list of switches The rules are updated in the switches .

Function to calculate the optimal path

```
def spath(pathlen):
    maxim=100
    a=0
    print pathlen
    for i in range(0,len(pathlen)):
        if pathlen[i]<maxim:
                a=i
                maxim=pathlen[i]
    pathlen[a]='100'
    path=re.findall('[a-z][0-9]',lines[a])
    print path
    path1=path
return path
```

5.3.6 Recovery Module

Storing of dropped packets in compression state.Handling the failed link using the alternate back path.Stored packets are sent using fast rerouting technique.Packets that are not left from the source are rerouted in pre-calculated alternate path.Packets that are transmitted before the detected failed link are stored in the switch in the compressed state.The stored packets are then decompressed and fast rerouted to the destined node.

Compression algorithm

Run length encoding is probably the easiest compression algorithm there is. It replaces sequences of the same data values within a file by a count number and a single value. It consists of replacing large sequences of repeating data(packets) with only one item of this data(packets) followed by a counter showing how many times this item is repeated and storing the packets in the switch.

Function to compress packets in switches

```
message = 'aaaaaaaaaabbbaxxxxyyyzyx';
function run_length_encode($msg)
{
   $i = $j = 0;
   $prev = '';
   $output = '';
   while ($msg[$i]) {
      if ($msg[$i] != $prev) {
         if ($i && $j > 1)
                 $output .= $j;
         $output .= $msg[$i];
         $prev = $msg[$i];
         $j = 0;
      }
      $j++;
      $i++;
   }
   if ($j > 1)
      $output .= $j;
   return $output;
}
// a10b3ax4y3zyx
echo run_length_encode($message);
```

5.3.7 Rule Placement Module

The rule placement module avoids overloading of switches with rules.A variable n helps in keeping count of the number of rules in the switch. Subprocess call function helps in implementing python commands. The rules in every switch is written to a file called cover. The number of rules is calculated and the value of n is incremented. If the value exceeds the threshold value ,8 ,the rules in the switch are flushed and new rules are added. Edst[] is the list of visited sites.

Function to update the rules in switch

```
def count(y):
    n=0
    fz=open('cover','w')
    subprocess.call(["ovs-ofctl","dump-flows","%s"%(y)],stdout=fz)
    fm=open('cover','r')
    line1 =fm.readlines()
    for k in line1:
          m=re.search('cookie',k)
          if m!=None:
               n=n+1
    return n
//call for count function
n=count(y)
               if n>=8:
                    edst=[]
                    subprocess.call(["ovs-ofctl","del-flows","%s"%(y)])
```

CHAPTER 6

RESULTS AND DISCUSSION

This chapter describes the performance evaluation measures and the different test cases along with screen shot

6.1 RESULTS

6.1.1 Assessment

S.No	PACKET SIZE(bytes)	PROPORTION OF TOTAL	DISTRIBUTION (packets)	DISTRIBUTION (bytes)
1	40	7 PARTS	58.3333%	6.856%
2	576	4 PARTS	33.333%	56.4155%
3	1500	1 PART	8.33%	36.7288%

Table 6.1 Packet distribution

S.No	switch	Input	Outports and distribution
1	s1	3	1 (40 %),2(60%)
2	s2	1	3(100%)
3	s2	2	3(100%)
4	s3	2	1(100%)
5	s3	3	1 (75 %),2(25%)
6	s4	2	1(100%)
7	s4	3	1 (62.5 %),2(37.5%)
8	s5	1	2 (35 %),3(65%)
9	s5	2	3(100%)
10	s6	1	2 (35 %),3(65%)

Table 6.2 Packet route through ports

S.No	Path	Fraction of total traffic(%)
1	s1 s3 s4 s5 s2	10
2	s1 s4 s5 s6 s2	10
3	s1 s4 s5 s2	10
3	s1 s4 s5 s2	10
4	s1 s4 s3 s6 s5 s2	5
5	s1 s3 s6 s2	30
6	s1 s3 s6 s5 s2	15
7	s1 s4 s3 s6 s2	10
8	s1 s3 s5 s6 s2	5

Table 6.3 Path traffic

6.1.2 Evaluation

$$R_T = T - t_1 + \lfloor \frac{t - t_0}{T} \rfloor \times T$$

Figure 6.1 Formula for packet drop and restoration

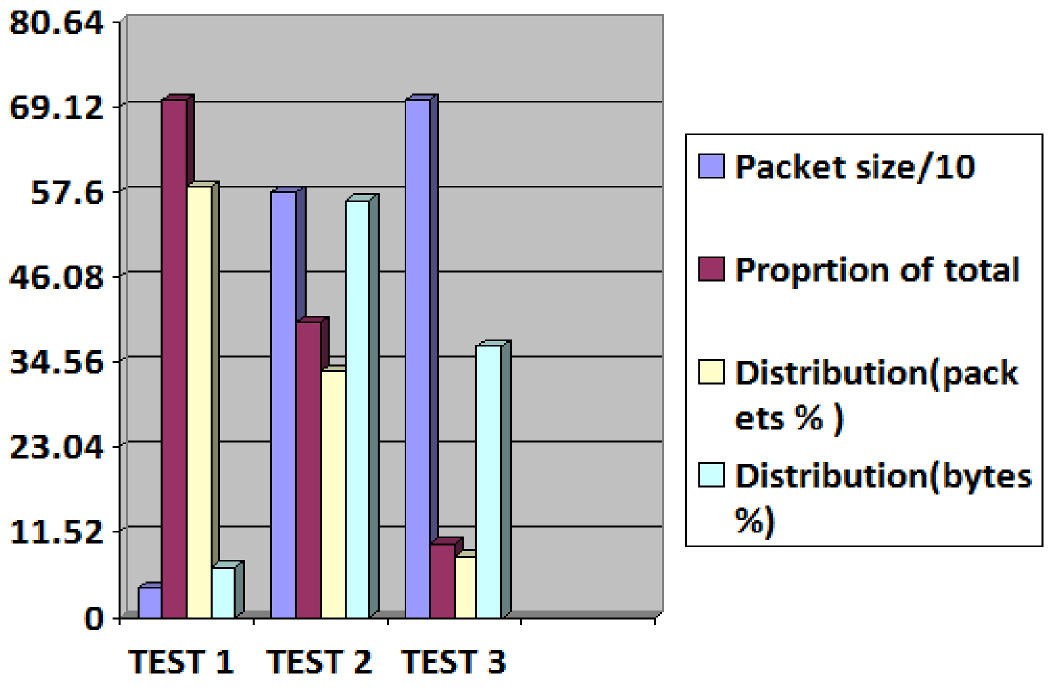


Figure 6.2 Packet Distribution

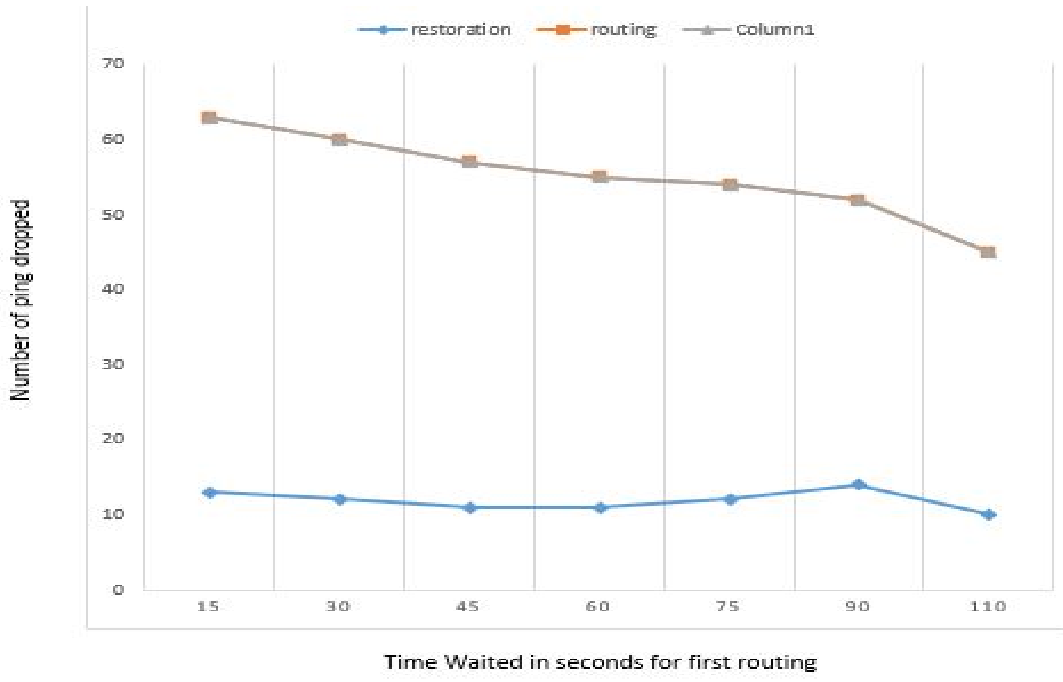


Figure 6.3 packetdrop and restoration

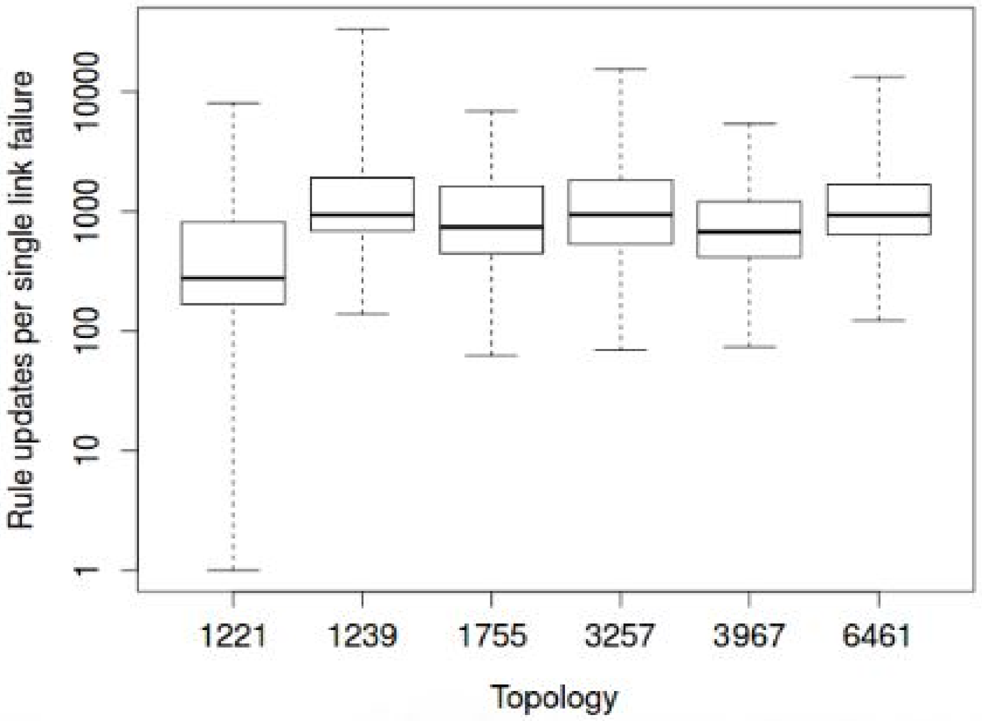


Figure 6.4 Rules updates during link failure

6.2 TEST CASES

6.2.1 Rule Space Overhead

A threshold, which specifies the maximum number of rules in each switch, is determined when the network is created. For each request, before the switch is loaded with new rules, the number of rules already present is compared against the threshold. If it exceeds the threshold, the rules are flushed before loading new rules for the current transmission.

6.2.2 Link Failure

Once the shortest path is calculated, the links along the path are checked if they are up. If not, the current calculated path is neglected and a second optimal path is calculated between the source and destination and the process is continued.

CHAPTER 7

CONCLUSIONS

This chapter explains the goals that are achieved through this project, the future works that can be done and the limitations found in this simulator.

7.1 GOALS ACHIEVED

Thus in this project, a network is simulated and the client host is allowed to connect with the remaining hosts in the network It emphasizes the rule space constraints of OpenFlow switch in energy-aware routing. In addition to capacity constraint, the rule space is also important as it can change the routing solution and affects QoS. Based on simulations with real traffic traces, Shortest Path Calculation and Re routing can achieve high energy efficiency for a backbone network while respecting both the capacity and the rule space constraints.

The various goals achieved are,

- Link Failure detected and packet loss is handled.
- The shortest path is calculated from multiple paths present and thus an alternate path is chosen, if any link along the path calculated goes down.
- Storing the packets in switch to overcome the reduction in packet dropping between the links.
- The rules in a switch are flushed before installing new rules, if the count exceeds the threshold decided.

7.2 FUTURE WORK

The system designed is concerned with hosts and switches within a LAN, and they are controlled by a single controller. This system can be extended in such a way that multiple LANs are involved and each LAN being controlled by multiple controllers, if there are large number of nodes.This project simulates the idea of Error detection and Recovery in SDN with the help of simulator called MININET and FLOODLIGHT controller. This can ipoalso be deployed in a real time scenario with the help of switches, host machines and other controllers.

APPENDIX A

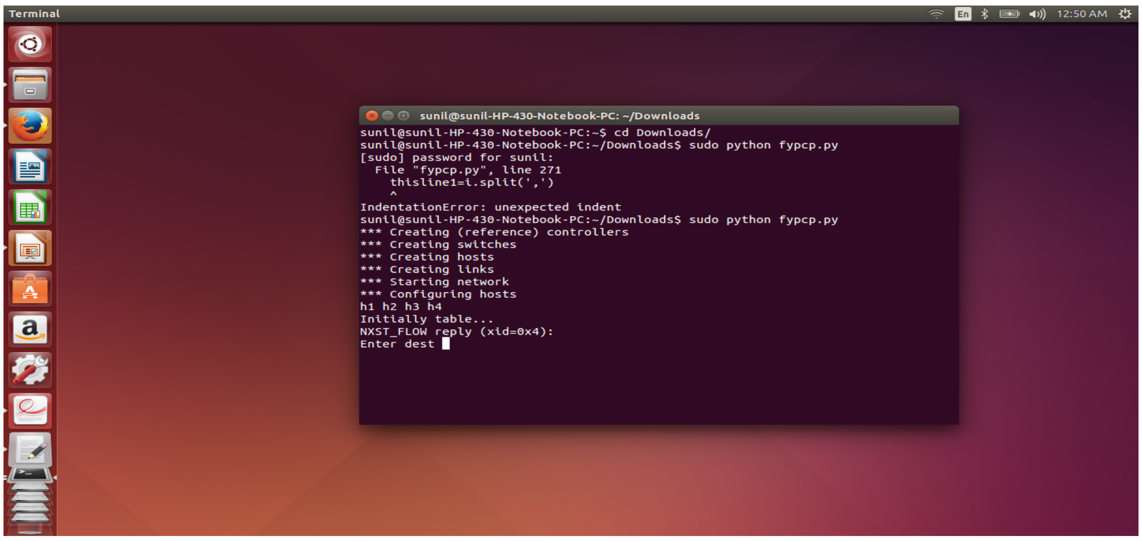
Screenshots and Snippets of SDN

Every rule updated in the switch has the following major fields to be filled:

- In_port : the incoming port number of the switch.
- dl_src : the MAC address of the source host
- dl_dst : the MAC address of the destination host
- nw_src : the IP address of the source host
- nw_dst : the IP address of the destination host

A.1 SCREESHOTS

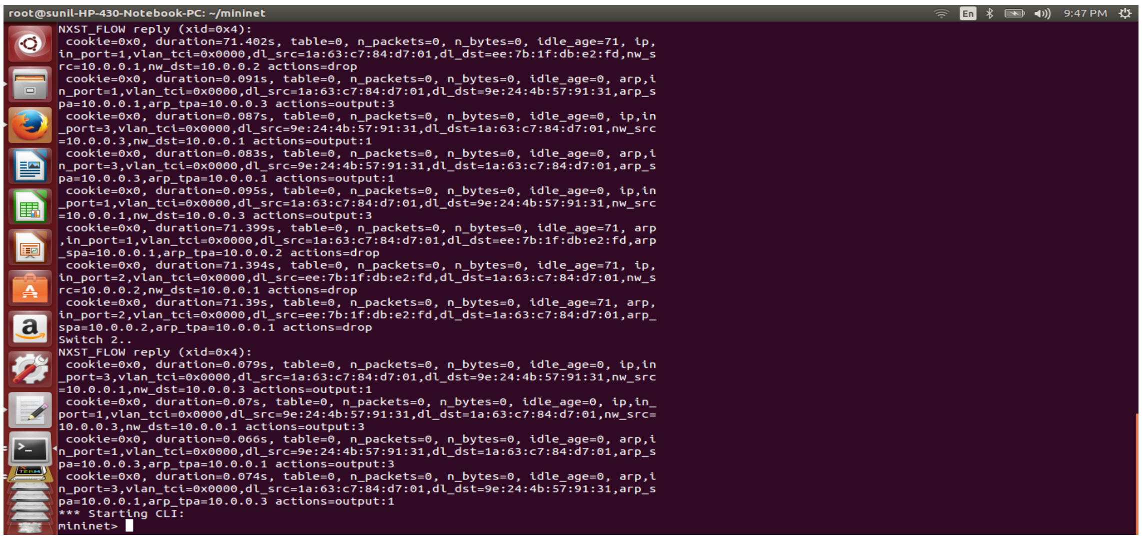
A.1.1 NETWORK CREATION SIMULATOR



```
sunil@sunil-HP-430-Notebook-PC:~$ cd Downloads/
sunil@sunil-HP-430-Notebook-PC:~/Downloads$ sudo python fypcp.py
[sudo] password for sunil:
  File "fypcp.py", line 271
    thisline1=i.split(',')
    ^
IndentationError: unexpected indent
sunil@sunil-HP-430-Notebook-PC:~/Downloads$ sudo python fypcp.py
*** Creating (reference) controllers
*** Creating switches
*** Creating hosts
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4
Initially table...
NXST_FLOW reply (xid=0x4):
Enter dest
```

Figure A.1 simulation of network creation

A.1.2 UPDATION OF RULES IN SWITCH

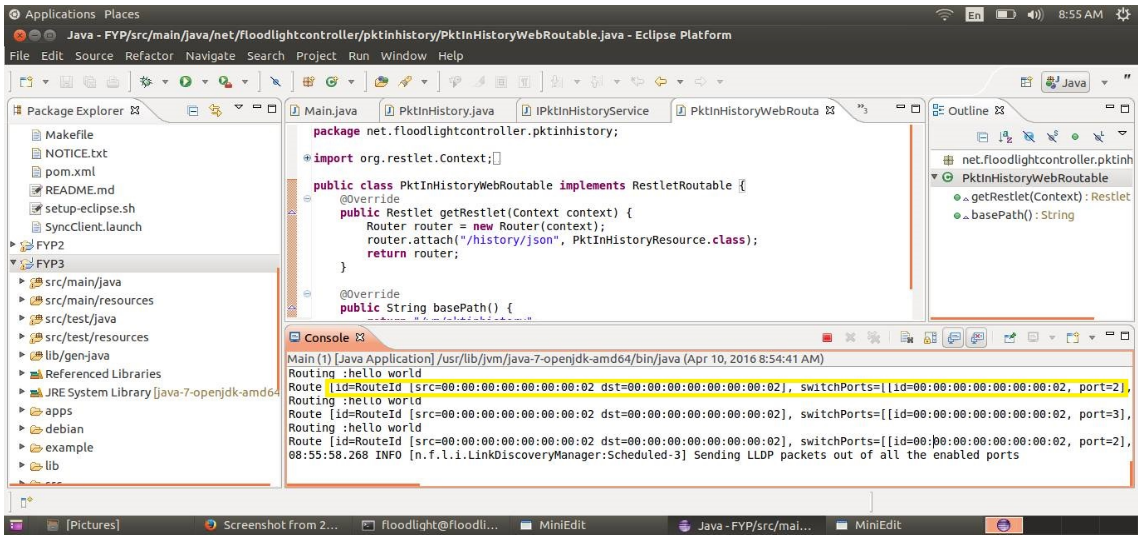


```
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=71.402s, table=0, n_packets=0, n_bytes=0, idle_age=71, ip,
in_port=1,vlan_tci=0x0000,dl_src=1a:63:c7:84:d7:01,dl_dst=ee:7b:1f:db:e2:fd,nw_s
rc=10.0.0.1,nw_dst=10.0.0.2 actions=drop
 cookie=0x0, duration=0.091s, table=0, n_packets=0, n_bytes=0, idle_age=0, arp,i
n_port=1,vlan_tci=0x0000,dl_src=1a:63:c7:84:d7:01,dl_dst=9e:24:4b:57:91:31,arp_s
pa=10.0.0.1,arp_tpa=10.0.0.3 actions=output:3
 cookie=0x0, duration=0.087s, table=0, n_packets=0, n_bytes=0, idle_age=0, ip,in
_port=3,vlan_tci=0x0000,dl_src=9e:24:4b:57:91:31,dl_dst=1a:63:c7:84:d7:01,nw_src
=10.0.0.3,nw_dst=10.0.0.1 actions=output:1
 cookie=0x0, duration=0.083s, table=0, n_packets=0, n_bytes=0, idle_age=0, arp,i
n_port=3,vlan_tci=0x0000,dl_src=9e:24:4b:57:91:31,dl_dst=1a:63:c7:84:d7:01,arp_s
pa=10.0.0.3,arp_tpa=10.0.0.1 actions=output:1
 cookie=0x0, duration=0.095s, table=0, n_packets=0, n_bytes=0, idle_age=0, ip,in
_port=1,vlan_tci=0x0000,dl_src=1a:63:c7:84:d7:01,dl_dst=9e:24:4b:57:91:31,nw_src
=10.0.0.1,nw_dst=10.0.0.3 actions=output:3
 cookie=0x0, duration=71.399s, table=0, n_packets=0, n_bytes=0, idle_age=71, arp
,in_port=1,vlan_tci=0x0000,dl_src=1a:63:c7:84:d7:01,dl_dst=ee:7b:1f:db:e2:fd,arp
_spa=10.0.0.1,arp_tpa=10.0.0.2 actions=drop
 cookie=0x0, duration=71.394s, table=0, n_packets=0, n_bytes=0, idle_age=71, ip,
in_port=2,vlan_tci=0x0000,dl_src=ee:7b:1f:db:e2:fd,dl_dst=1a:63:c7:84:d7:01,nw_s
rc=10.0.0.2,nw_dst=10.0.0.1 actions=drop
 cookie=0x0, duration=71.39s, table=0, n_packets=0, n_bytes=0, idle_age=71, arp,
in_port=2,vlan_tci=0x0000,dl_src=ee:7b:1f:db:e2:fd,dl_dst=1a:63:c7:84:d7:01,arp_
spa=10.0.0.2,arp_tpa=10.0.0.1 actions=drop
Switch 2..
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=0.079s, table=0, n_packets=0, n_bytes=0, idle_age=0, ip,in
_port=3,vlan_tci=0x0000,dl_src=1a:63:c7:84:d7:01,dl_dst=9e:24:4b:57:91:31,nw_src
=10.0.0.1,nw_dst=10.0.0.3 actions=output:1
 cookie=0x0, duration=0.07s, table=0, n_packets=0, n_bytes=0, idle_age=0, ip,in_
port=1,vlan_tci=0x0000,dl_src=9e:24:4b:57:91:31,dl_dst=1a:63:c7:84:d7:01,nw_src=
10.0.0.3,nw_dst=10.0.0.1 actions=output:3
 cookie=0x0, duration=0.066s, table=0, n_packets=0, n_bytes=0, idle_age=0, arp,i
n_port=1,vlan_tci=0x0000,dl_src=9e:24:4b:57:91:31,dl_dst=1a:63:c7:84:d7:01,arp_s
pa=10.0.0.3,arp_tpa=10.0.0.1 actions=output:3
 cookie=0x0, duration=0.074s, table=0, n_packets=0, n_bytes=0, idle_age=0, arp,i
n_port=3,vlan_tci=0x0000,dl_src=1a:63:c7:84:d7:01,dl_dst=9e:24:4b:57:91:31,arp_s
pa=10.0.0.1,arp_tpa=10.0.0.3 actions=output:1
*** Starting CLI:
mininet> 
```

Figure A.2 Rules updated in switch

As mentioned above, threshold is determined when the network is created. For each request,before the switch is loaded with new rules, the number of rules already present is compared against the threshold. If it exceeds the threshold, the rules are flushed before loading new rules for the current transmission.

A.1.3 LINK LOSS



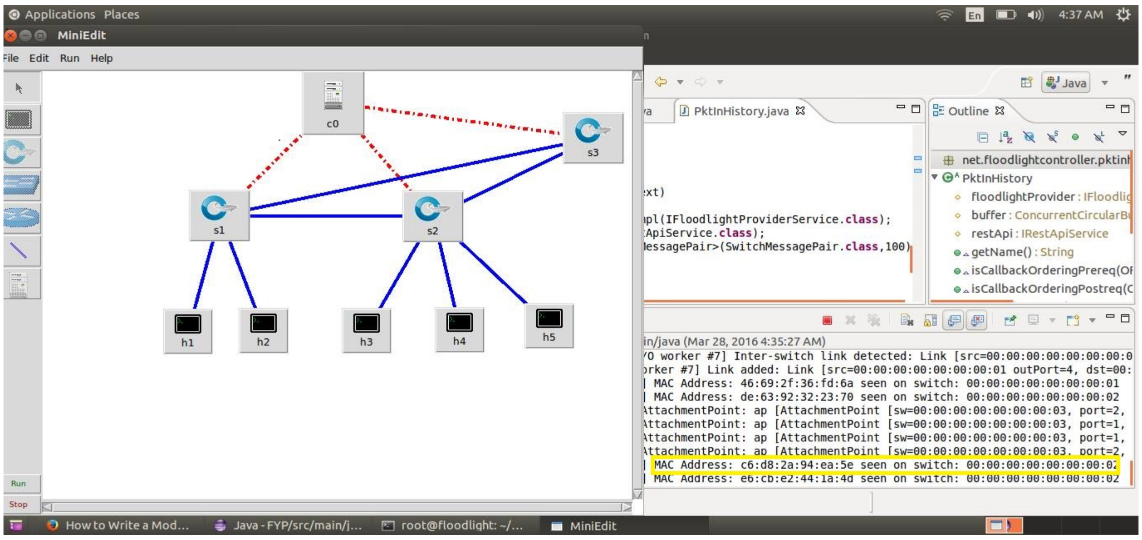
```
package net.floodlightcontroller.pktinhistory;

import org.restlet.Context;

public class PktInHistoryWebRoutable implements RestletRoutable {
    @Override
    public Restlet getRestlet(Context context) {
        Router router = new Router(context);
        router.attach("/history/json", PktInHistoryResource.class);
        return router;
    }

    @Override
    public String basePath() {
```

```
Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Apr 10, 2016 8:54:41 AM)
Routing :hello world
Route [id=RouteId [src=00:00:00:00:00:00:00:02 dst=00:00:00:00:00:00:00:02], switchPorts=[[id=00:00:00:00:00:00:00:02, port=2],
Routing :hello world
Route [id=RouteId [src=00:00:00:00:00:00:00:02 dst=00:00:00:00:00:00:00:02], switchPorts=[[id=00:00:00:00:00:00:00:02, port=3],
Routing :hello world
Route [id=RouteId [src=00:00:00:00:00:00:00:02 dst=00:00:00:00:00:00:00:02], switchPorts=[[id=00:00:00:00:00:00:00:02, port=2],
08:55:58.268 INFO [n.f.l.i.LinkDiscoveryManager:Scheduled-3] Sending LLDP packets out of all the enabled ports
```

Figure A.3 Shortest path routing

Once the shortest path is calculated, the links along the path are checked if they are up. If not, the current calculated path is neglected and a second optimal path is calculated between the source and destination and the process is continued.

A.1.4 MACTracker with Network Topology

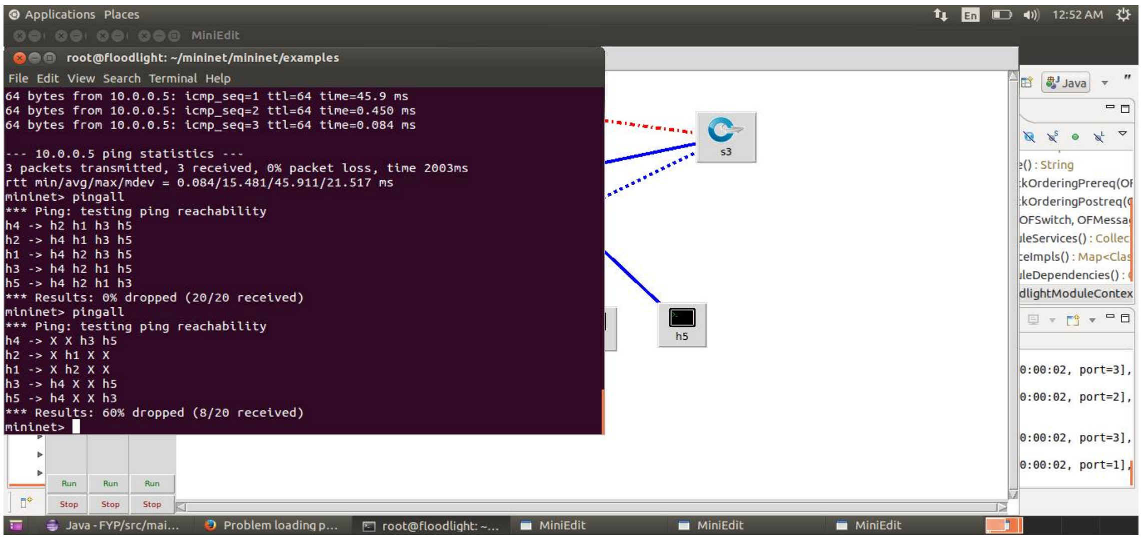


```
xt)

pl(IFloodlightProviderService.class);
ApiService.class);
essagePair>(SwitchMessagePair.class,100)
```

```
in/java (Mar 28, 2016 4:35:27 AM)
O worker #7] Inter-switch link detected: Link [src=00:00:00:00:00:00:00:0
rker #7] Link added: Link [src=00:00:00:00:00:00:00:01 outPort=4, dst=00:
| MAC Address: 46:69:2f:36:fd:6a seen on switch: 00:00:00:00:00:00:00:01
| MAC Address: de:63:92:32:23:70 seen on switch: 00:00:00:00:00:00:00:02
ttachmentPoint: ap [AttachmentPoint [sw=00:00:00:00:00:00:00:03, port=2,
ttachmentPoint: ap [AttachmentPoint [sw=00:00:00:00:00:00:00:03, port=1,
ttachmentPoint: ap [AttachmentPoint [sw=00:00:00:00:00:00:00:03, port=1,
ttachmentPoint: ap [AttachmentPoint [sw=00:00:00:00:00:00:00:03, port=2,
| MAC Address: c6:d8:2a:94:ea:5e seen on switch: 00:00:00:00:00:00:00:02
| MAC Address: e6:cb:e2:44:1a:4d seen on switch: 00:00:00:00:00:00:00:02
```

Figure A.4 MACTracker and Network Topology

A.1.5 Ping ICMP Packets



```
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=45.9 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.450 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.084 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.084/15.481/45.911/21.517 ms
mininet> pingall
*** Ping: testing ping reachability
h4 -> h2 h1 h3 h5
h2 -> h4 h1 h3 h5
h1 -> h4 h2 h3 h5
h3 -> h4 h2 h1 h5
h5 -> h4 h2 h1 h3
*** Results: 0% dropped (20/20 received)
mininet> pingall
*** Ping: testing ping reachability
h4 -> X X h3 h5
h2 -> X h1 X X
h1 -> X h2 X X
h3 -> h4 X X h5
h5 -> h4 X X h3
*** Results: 60% dropped (8/20 received)
mininet>
```

Figure A.5 Ping the packets betwween the hosts

A.2 SAMPLE SNIPPETS FOR SDN

A.2.1 Concurrentcircularbuffer.java

```
package net.floodlightcontroller.pktinhistory;
import java.util.concurrent.atomic.AtomicInteger;
import java.lang.reflect.Array;

public class ConcurrentCircularBuffer <T> {
   private final AtomicInteger cursor = new AtomicInteger();
   private final Object[]    buffer;
   private final Class<T>    type;

   public ConcurrentCircularBuffer (final Class <T> type,
                                     final int bufferSize)
   {
       if (bufferSize < 1) {
      throw new IllegalArgumentException(
                "Buffer size must be a positive value"
```

```
                    );
        }

        this.type  = type;
        this.buffer = new Object [ bufferSize ];
    }

    public void add (T sample) {
        buffer[ cursor.getAndIncrement() % buffer.length ] = sample;
    }

    @SuppressWarnings("unchecked")
    public T[] snapshot () {
        Object[] snapshots = new Object [ buffer.length ];

        /* Identify the start-position of the buffer. */
        int before = cursor.get();

        /* Terminate early for an empty buffer. */
        if (before == 0) {
            return (T[]) Array.newInstance(type, 0);
        }

        System.arraycopy(buffer, 0, snapshots, 0, buffer.length);

        int after          = cursor.get();
        int size           = buffer.length - (after - before);
        int snapshotCursor = before - 1;

        /* The entire buffer was replaced during the copy. */
        if (size <= 0) {
            return (T[]) Array.newInstance(type, 0);
        }

        int start = snapshotCursor - (size - 1);
```

```
int end  = snapshotCursor;

if (snapshotCursor < snapshots.length) {
    size  = snapshotCursor + 1;
    start = 0;
}

/* Copy the sample snapshot to a new array the size of our
   stable
 * snapshot area.
 */
T[] result = (T[]) Array.newInstance(type, size);

int startOfCopy = start % snapshots.length;
int endOfCopy = end  % snapshots.length;

/* If the buffer space wraps the physical end of the array,
   use two
 * copies to construct the new array.
 */
if (startOfCopy > endOfCopy) {
    System.arraycopy(snapshots, startOfCopy,
                     result, 0,
                     snapshots.length - startOfCopy);
    System.arraycopy(snapshots, 0,
                     result, (snapshots.length - startOfCopy),
                     endOfCopy + 1);
}
else {
    /* Otherwise it's a single continuous segment, copy the
       whole thing
     * into the result.
     */
    System.arraycopy(snapshots, startOfCopy, result, 0,
       size);
```

```
      }

      return (T[]) result;
   }
}
```

A.2.2 Forwarding.java

```
package net.floodlightcontroller.forwarding;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.List;
import java.util.Map;

import net.floodlightcontroller.core.FloodlightContext;
import net.floodlightcontroller.core.IFloodlightProviderService;
import net.floodlightcontroller.core.IOFSwitch;
import net.floodlightcontroller.devicemanager.IDevice;
import net.floodlightcontroller.devicemanager.IDeviceService;
import net.floodlightcontroller.devicemanager.SwitchPort;
import
   net.floodlightcontroller.core.annotations.LogMessageCategory;
import net.floodlightcontroller.core.annotations.LogMessageDoc;
import net.floodlightcontroller.core.annotations.LogMessageDocs;
import net.floodlightcontroller.core.internal.IOFSwitchService;
import
   net.floodlightcontroller.core.module.FloodlightModuleContext;
import
   net.floodlightcontroller.core.module.FloodlightModuleException;
import net.floodlightcontroller.core.module.IFloodlightModule;
import net.floodlightcontroller.core.module.IFloodlightService;
import net.floodlightcontroller.core.util.AppCookie;
import net.floodlightcontroller.debugcounter.IDebugCounterService;
import net.floodlightcontroller.packet.Ethernet;
import net.floodlightcontroller.packet.IPv4;
import net.floodlightcontroller.packet.TCP;
import net.floodlightcontroller.packet.UDP;
import net.floodlightcontroller.routing.ForwardingBase;
import net.floodlightcontroller.routing.IRoutingDecision;
import net.floodlightcontroller.routing.IRoutingService;
import net.floodlightcontroller.routing.Route;
import net.floodlightcontroller.topology.ITopologyService;

import org.projectfloodlight.openflow.protocol.OFFlowMod;
import org.projectfloodlight.openflow.protocol.match.Match;
import org.projectfloodlight.openflow.protocol.match.MatchField;
import org.projectfloodlight.openflow.protocol.OFFlowModCommand;
import org.projectfloodlight.openflow.protocol.OFPacketIn;
import org.projectfloodlight.openflow.protocol.OFPacketOut;
import org.projectfloodlight.openflow.protocol.OFVersion;
import org.projectfloodlight.openflow.types.DatapathId;
import org.projectfloodlight.openflow.types.EthType;
import org.projectfloodlight.openflow.types.IPv4Address;
import org.projectfloodlight.openflow.types.IpProtocol;
import org.projectfloodlight.openflow.types.MacAddress;
import org.projectfloodlight.openflow.types.OFBufferId;
import org.projectfloodlight.openflow.types.OFPort;
import org.projectfloodlight.openflow.types.OFVlanVidMatch;
import org.projectfloodlight.openflow.types.U64;
import org.projectfloodlight.openflow.types.VlanVid;
import org.projectfloodlight.openflow.protocol.action.OFAction;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@LogMessageCategory("Flow Programming")
public class Forwarding extends ForwardingBase implements
   IFloodlightModule {
```

```
protected static Logger log =
   LoggerFactory.getLogger(Forwarding.class);

@Override
@LogMessageDoc(level="ERROR",
message="Unexpected decision made for this packet-in={}",
explanation="An unsupported PacketIn decision has been " +
      "passed to the flow programming component",
      recommendation=LogMessageDoc.REPORT_CONTROLLER_BUG)
public Command processPacketInMessage(IOFSwitch sw, OFPacketIn
   pi, IRoutingDecision decision, FloodlightContext cntx) {
   Ethernet eth = IFloodlightProviderService.bcStore.get(cntx,
      IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
   // We found a routing decision (i.e. Firewall is enabled...
      it's the only thing that makes RoutingDecisions)
   if (decision != null) {
      if (log.isTraceEnabled()) {
         log.trace("Forwaring decision={} was made for
            PacketIn={}",
            decision.getRoutingAction().toString(), pi);
      }

      switch(decision.getRoutingAction()) {
      case NONE:
         // don't do anything
         return Command.CONTINUE;
      case FORWARD_OR_FLOOD:
      case FORWARD:
         doForwardFlow(sw, pi, cntx, false);
         return Command.CONTINUE;
      case MULTICAST:
         // treat as broadcast
         doFlood(sw, pi, cntx);
         return Command.CONTINUE;
      case DROP:
            doDropFlow(sw, pi, decision, cntx);
            return Command.CONTINUE;
         default:
            log.error("Unexpected decision made for this
                packet-in={}", pi, decision.getRoutingAction());
            return Command.CONTINUE;
         }
      } else { // No routing decision was found. Forward to
         destination or flood if bcast or mcast.
         if (log.isTraceEnabled()) {
            log.trace("No decision was made for PacketIn={},
                forwarding", pi);
         }

         if (eth.isBroadcast() || eth.isMulticast()) {
            doFlood(sw, pi, cntx);
         } else {
            doForwardFlow(sw, pi, cntx, false);
         }
      }

      return Command.CONTINUE;
   }

   @LogMessageDoc(level="ERROR",
         message="Failure writing drop flow mod",
         explanation="An I/O error occured while trying to write a
             " +
               "drop flow mod to a switch",
               recommendation=LogMessageDoc.CHECK_SWITCH)
   protected void doDropFlow(IOFSwitch sw, OFPacketIn pi,
       IRoutingDecision decision, FloodlightContext cntx) {
      OFPort inPort = (pi.getVersion().compareTo(OFVersion.OF_12)
         < 0 ? pi.getInPort() :
         pi.getMatch().get(MatchField.IN_PORT));
```

```
    Match m = createMatchFromPacket(sw, inPort, cntx);
    OFFlowMod.Builder fmb = sw.getOFFactory().buildFlowAdd(); //
       this will be a drop-flow; a flow that will not output to
       any ports
    List<OFAction> actions = new ArrayList<OFAction>(); // set
       no action to drop
    U64 cookie = AppCookie.makeCookie(FORWARDING_APP_ID, 0);

    fmb.setCookie(cookie)
    .setHardTimeout(FLOWMOD_DEFAULT_HARD_TIMEOUT)
    .setIdleTimeout(FLOWMOD_DEFAULT_IDLE_TIMEOUT)
    .setBufferId(OFBufferId.NO_BUFFER)
    .setMatch(m)
    .setActions(actions) // empty list
    .setPriority(FLOWMOD_DEFAULT_PRIORITY);

    try {
       if (log.isDebugEnabled()) {
          log.debug("write drop flow-mod sw={} match={}
             flow-mod={}",
               new Object[] { sw, m, fmb.build() });
       }
       boolean dampened = messageDamper.write(sw, fmb.build());
       log.debug("OFMessage dampened: {}", dampened);
    } catch (IOException e) {
       log.error("Failure writing drop flow mod", e);
    }
 }

 protected void doForwardFlow(IOFSwitch sw, OFPacketIn pi,
    FloodlightContext cntx, boolean requestFlowRemovedNotifn) {
    OFPort inPort = (pi.getVersion().compareTo(OFVersion.OF_12)
       < 0 ? pi.getInPort() :
       pi.getMatch().get(MatchField.IN_PORT));
    // Check if we have the location of the destination
```

```
IDevice dstDevice = IDeviceService.fcStore.get(cntx,
   IDeviceService.CONTEXT_DST_DEVICE);

if (dstDevice != null) {
   IDevice srcDevice = IDeviceService.fcStore.get(cntx,
      IDeviceService.CONTEXT_SRC_DEVICE);
   DatapathId srcIsland =
      topologyService.getL2DomainId(sw.getId());

   if (srcDevice == null) {
      log.debug("No device entry found for source device");
      return;
   }
   if (srcIsland == null) {
      log.debug("No openflow island found for source {}/{}",
            sw.getId().toString(), inPort);
      return;
   }

   // Validate that we have a destination known on the same
      island
   // Validate that the source and destination are not on
      the same switchport
   boolean on_same_island = false;
   boolean on_same_if = false;
   for (SwitchPort dstDap : dstDevice.getAttachmentPoints())
      {
      DatapathId dstSwDpid = dstDap.getSwitchDPID();
      DatapathId dstIsland =
         topologyService.getL2DomainId(dstSwDpid);
      if ((dstIsland != null) &&
         dstIsland.equals(srcIsland)) {
         on_same_island = true;
         if (sw.getId().equals(dstSwDpid) &&
            inPort.equals(dstDap.getPort())) {
```

```
            on_same_if = true;
         }
         break;
      }
   }

   if (!on_same_island) {
      // Flood since we don’t know the dst device
      if (log.isTraceEnabled()) {
         log.trace("No first hop island found for destination
            " +
              "device {}, Action = flooding", dstDevice);
      }
      doFlood(sw, pi, cntx);
      return;
   }

   if (on_same_if) {
      if (log.isTraceEnabled()) {
         log.trace("Both source and destination are on the
            same " +
              "switch/port {}/{}, Action = NOP",
              sw.toString(), inPort);
      }
      return;
   }

   // Install all the routes where both src and dst have
      attachment
   // points. Since the lists are stored in sorted order we
      can
   // traverse the attachment points in O(m+n) time
   SwitchPort[] srcDaps = srcDevice.getAttachmentPoints();
   Arrays.sort(srcDaps, clusterIdComparator);
   SwitchPort[] dstDaps = dstDevice.getAttachmentPoints();
```

```
Arrays.sort(dstDaps, clusterIdComparator);

int iSrcDaps = 0, iDstDaps = 0;

while ((iSrcDaps < srcDaps.length) && (iDstDaps <
    dstDaps.length)) {
   SwitchPort srcDap = srcDaps[iSrcDaps];
   SwitchPort dstDap = dstDaps[iDstDaps];

   // srcCluster and dstCluster here cannot be null as
   // every switch will be at least in its own L2 domain.
   DatapathId srcCluster =
      topologyService.getL2DomainId(srcDap.getSwitchDPID());
   DatapathId dstCluster =
      topologyService.getL2DomainId(dstDap.getSwitchDPID());

   int srcVsDest = srcCluster.compareTo(dstCluster);
   if (srcVsDest == 0) {
      if (!srcDap.equals(dstDap)) {
         Route route =
               routingEngineService.getRoute(srcDap.getSwitchDPID(),
                     srcDap.getPort(),
                     dstDap.getSwitchDPID(),
                     dstDap.getPort(), U64.of(0)); //cookie
                        = 0, i.e., default route
         if (route != null) {
            if (log.isTraceEnabled()) {
               log.trace("pushRoute inPort={} route={} " +
                     "destination={}:{}",
                     new Object[] { inPort, route,
                     dstDap.getSwitchDPID(),
                     dstDap.getPort()});
            }
            U64 cookie =
               AppCookie.makeCookie(FORWARDING_APP_ID, 0);

            Match m = createMatchFromPacket(sw, inPort,
               cntx);

            pushRoute(route, m, pi, sw.getId(), cookie,
                 cntx, requestFlowRemovedNotifn, false,
                 OFFlowModCommand.ADD);
          }
        }
        iSrcDaps++;
        iDstDaps++;
      } else if (srcVsDest < 0) {
        iSrcDaps++;
      } else {
        iDstDaps++;
      }
    }
  } else {
    // Flood since we don’t know the dst device
    doFlood(sw, pi, cntx);
  }
}


protected Match createMatchFromPacket(IOFSwitch sw, OFPort
   inPort, FloodlightContext cntx) {
  // The packet in match will only contain the port number.
  // We need to add in specifics for the hosts we’re routing
     between.
  Ethernet eth = IFloodlightProviderService.bcStore.get(cntx,
     IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
  VlanVid vlan = VlanVid.ofVlan(eth.getVlanID());
  MacAddress srcMac = eth.getSourceMACAddress();
MacAddress dstMac = eth.getDestinationMACAddress();

Match.Builder mb = sw.getOFFactory().buildMatch();
mb.setExact(MatchField.IN_PORT, inPort);

if (FLOWMOD_DEFAULT_MATCH_MAC) {
   mb.setExact(MatchField.ETH_SRC, srcMac)
   .setExact(MatchField.ETH_DST, dstMac);
}

if (FLOWMOD_DEFAULT_MATCH_VLAN) {
   if (!vlan.equals(VlanVid.ZERO)) {
      mb.setExact(MatchField.VLAN_VID,
         OFVlanVidMatch.ofVlanVid(vlan));
   }
}

// TODO Detect switch type and match to create
   hardware-implemented flow
// TODO Allow for IPv6 matches
if (eth.getEtherType() == EthType.IPv4) { /* shallow check
   for equality is okay for EthType */
   IPv4 ip = (IPv4) eth.getPayload();
   IPv4Address srcIp = ip.getSourceAddress();
   IPv4Address dstIp = ip.getDestinationAddress();

   if (FLOWMOD_DEFAULT_MATCH_IP_ADDR) {
      mb.setExact(MatchField.ETH_TYPE, EthType.IPv4)
      .setExact(MatchField.IPV4_SRC, srcIp)
      .setExact(MatchField.IPV4_DST, dstIp);
   }

   if (FLOWMOD_DEFAULT_MATCH_TRANSPORT) {
      /*
       * Take care of the ethertype if not included earlier,
         * since it’s a prerequisite for transport ports.
         */
        if (!FLOWMOD_DEFAULT_MATCH_IP_ADDR) {
           mb.setExact(MatchField.ETH_TYPE, EthType.IPv4);
        }

        if (ip.getProtocol().equals(IpProtocol.TCP)) {
           TCP tcp = (TCP) ip.getPayload();
           mb.setExact(MatchField.IP_PROTO, IpProtocol.TCP)
           .setExact(MatchField.TCP_SRC, tcp.getSourcePort())
           .setExact(MatchField.TCP_DST,
              tcp.getDestinationPort());
        } else if (ip.getProtocol().equals(IpProtocol.UDP)) {
           UDP udp = (UDP) ip.getPayload();
           mb.setExact(MatchField.IP_PROTO, IpProtocol.UDP)
           .setExact(MatchField.UDP_SRC, udp.getSourcePort())
           .setExact(MatchField.UDP_DST,
              udp.getDestinationPort());
        }
     }
  } else if (eth.getEtherType() == EthType.ARP) { /* shallow
      check for equality is okay for EthType */
     mb.setExact(MatchField.ETH_TYPE, EthType.ARP);
  }
  return mb.build();
}

/**
 * Creates a OFPacketOut with the OFPacketIn data that is
    flooded on all ports unless
 * the port is blocked, in which case the packet will be
    dropped.
 * @param sw The switch that receives the OFPacketIn
 * @param pi The OFPacketIn that came to the switch
 * @param cntx The FloodlightContext associated with this
    OFPacketIn
 */
@LogMessageDoc(level="ERROR",
      message="Failure writing PacketOut " +
            "switch={switch} packet-in={packet-in} " +
            "packet-out={packet-out}",
            explanation="An I/O error occured while writing a
               packet " +
                 "out message to the switch",
                 recommendation=LogMessageDoc.CHECK_SWITCH)
protected void doFlood(IOFSwitch sw, OFPacketIn pi,
    FloodlightContext cntx) {
   OFPort inPort = (pi.getVersion().compareTo(OFVersion.OF_12)
      < 0 ? pi.getInPort() :
      pi.getMatch().get(MatchField.IN_PORT));
   if (topologyService.isIncomingBroadcastAllowed(sw.getId(),
      inPort) == false) {
      if (log.isTraceEnabled()) {
         log.trace("doFlood, drop broadcast packet, pi={}, " +
               "from a blocked port, srcSwitch=[{},{}],
                  linkInfo={}",
               new Object[] {pi, sw.getId(), inPort});
      }
      return;
   }

   // Set Action to flood
   OFPacketOut.Builder pob = sw.getOFFactory().buildPacketOut();
   List<OFAction> actions = new ArrayList<OFAction>();
   if (sw.hasAttribute(IOFSwitch.PROP_SUPPORTS_OFPP_FLOOD)) {
      actions.add(sw.getOFFactory().actions().output(OFPort.FLOOD,
         Integer.MAX_VALUE)); // FLOOD is a more
         selective/efficient version of ALL
   } else {
        actions.add(sw.getOFFactory().actions().output(OFPort.ALL,
            Integer.MAX_VALUE));
    }
    pob.setActions(actions);

    // set buffer-id, in-port and packet-data based on packet-in
    pob.setBufferId(OFBufferId.NO_BUFFER);
    pob.setInPort(inPort);
    pob.setData(pi.getData());

    try {
        if (log.isTraceEnabled()) {
            log.trace("Writing flood PacketOut switch={}
                packet-in={} packet-out={}",
                    new Object[] {sw, pi, pob.build()});
        }
        messageDamper.write(sw, pob.build());
    } catch (IOException e) {
        log.error("Failure writing PacketOut switch={}
            packet-in={} packet-out={}",
                new Object[] {sw, pi, pob.build()}, e);
    }

    return;
}

// IFloodlightModule methods

@Override
public Collection<Class<? extends IFloodlightService>>
    getModuleServices() {
    // We don't export any services
    return null;
}
```

```
@Override
public Map<Class<? extends IFloodlightService>,
    IFloodlightService>
getServiceImpls() {
   // We don’t have any services
   return null;
}

@Override
public Collection<Class<? extends IFloodlightService>>
    getModuleDependencies() {
   Collection<Class<? extends IFloodlightService>> l =
         new ArrayList<Class<? extends IFloodlightService>>();
   l.add(IFloodlightProviderService.class);
   l.add(IDeviceService.class);
   l.add(IRoutingService.class);
   l.add(ITopologyService.class);
   l.add(IDebugCounterService.class);
   return l;
}

@Override
@LogMessageDocs({
   @LogMessageDoc(level="WARN",
         message="Error parsing flow idle timeout, " +
               "using default of {number} seconds",
               explanation="The properties file contains an
                  invalid " +
                    "flow idle timeout",
                    recommendation="Correct the idle timeout in
                       the " +
         "properties file."),
         @LogMessageDoc(level="WARN",
         message="Error parsing flow hard timeout, " +
               "using default of {number} seconds",
             explanation="The properties file contains an
                invalid " +
                  "flow hard timeout",
                  recommendation="Correct the hard timeout in
                     the " +
             "properties file.")
})
public void init(FloodlightModuleContext context) throws
   FloodlightModuleException {
   super.init();
   this.floodlightProviderService =
      context.getServiceImpl(IFloodlightProviderService.class);
   this.deviceManagerService =
      context.getServiceImpl(IDeviceService.class);
   this.routingEngineService =
      context.getServiceImpl(IRoutingService.class);
   this.topologyService =
      context.getServiceImpl(ITopologyService.class);
   this.debugCounterService =
      context.getServiceImpl(IDebugCounterService.class);
   this.switchService =
      context.getServiceImpl(IOFSwitchService.class);

   Map<String, String> configParameters =
      context.getConfigParams(this);
   String tmp = configParameters.get("hard-timeout");
   if (tmp != null) {
      FLOWMOD_DEFAULT_HARD_TIMEOUT = Integer.parseInt(tmp);
      log.info("Default hard timeout set to {}.",
         FLOWMOD_DEFAULT_HARD_TIMEOUT);
   } else {
      log.info("Default hard timeout not configured. Using
         {}.", FLOWMOD_DEFAULT_HARD_TIMEOUT);
   }
   tmp = configParameters.get("idle-timeout");
```

```
if (tmp != null) {
   FLOWMOD_DEFAULT_IDLE_TIMEOUT = Integer.parseInt(tmp);
   log.info("Default idle timeout set to {}.",
      FLOWMOD_DEFAULT_IDLE_TIMEOUT);
} else {
   log.info("Default idle timeout not configured. Using
      {}.", FLOWMOD_DEFAULT_IDLE_TIMEOUT);
}
tmp = configParameters.get("priority");
if (tmp != null) {
   FLOWMOD_DEFAULT_PRIORITY = Integer.parseInt(tmp);
   log.info("Default priority set to {}.",
      FLOWMOD_DEFAULT_PRIORITY);
} else {
   log.info("Default priority not configured. Using {}.",
      FLOWMOD_DEFAULT_PRIORITY);
}
tmp = configParameters.get("set-send-flow-rem-flag");
if (tmp != null) {
   FLOWMOD_DEFAULT_SET_SEND_FLOW_REM_FLAG =
      Boolean.parseBoolean(tmp);
   log.info("Default flags will be set to SEND_FLOW_REM.");
} else {
   log.info("Default flags will be empty.");
}
tmp = configParameters.get("match");
if (tmp != null) {
   tmp = tmp.toLowerCase();
   if (!tmp.contains("vlan") && !tmp.contains("mac") &&
      !tmp.contains("ip") && !tmp.contains("port")) {
      /* leave the default configuration -- blank or invalid
         'match' value */
   } else {
      FLOWMOD_DEFAULT_MATCH_VLAN = tmp.contains("vlan") ?
         true : false;
```

```
                FLOWMOD_DEFAULT_MATCH_MAC = tmp.contains("mac") ? true
                    : false;
                FLOWMOD_DEFAULT_MATCH_IP_ADDR = tmp.contains("ip") ?
                    true : false;
                FLOWMOD_DEFAULT_MATCH_TRANSPORT = tmp.contains("port")
                    ? true : false;

            }
        }
        log.info("Default flow matches set to: VLAN=" +
            FLOWMOD_DEFAULT_MATCH_VLAN
                + ", MAC=" + FLOWMOD_DEFAULT_MATCH_MAC
                + ", IP=" + FLOWMOD_DEFAULT_MATCH_IP_ADDR
                + ", TPPT=" + FLOWMOD_DEFAULT_MATCH_TRANSPORT);

    }

    @Override
    public void startUp(FloodlightModuleContext context) {
        super.startUp();
    }
}
```

REFERENCES

[1] "Ip fast reroute for single and correlated failures with rmrc: Relaxed multiple routing configurations:ip fast reroute for single and correlated failures", *IEEE Transactions on network and service management*, vol. 6, num. 1, March 2009.

[2] "Openflow-based segment protection in ethernet networks", *IEEE/OSA*, vol. 5, num. 1, September 2013.

[3] "Efficient routing for traffic offloading in software-defined network", *International Workshop on Software Defined Networks for a New Generation of Applications and Services*, 2014.

[4] "Europe-wide demonstration of fast network resto-ration with openflow , ieice commun. express", *IEEE 22nd International Conference on Network Protocols.*, vol. 3, pp. 275–280, 2014.

[5] "Fast recovery in software-defined networks, ieee software defined networks (ewsdn)", *Third European Workshop*, vol. 13, num. 5, September 2014.

[6] "Fault-tolerant openflow-based software switch architecture with linc switches for a reliable network data exchange,research and educational experiment workshop", *Third GENI*, pp. 19 – 20, March 2014.

[7] "Using orion:-a hybrid hierarchical control plane of software-defined networking for large-scale networks", *IEEE 22nd International Conference on Network Protocols.*, 2014.

[8] "Detour planning for fast and reliable failure recovery in sdn with openstate, design of reliable communication networks (drcn)", *11th International Conference*, March 2015.

[9] "Openflow path fast failover fast convergence mechanism", *Network Research Proceedings of the Asia-Pacific Advanced Network*, vol. 38, pp. 23–28, 2015.

[10] "Scalable resilience for software-defined networking using loop-free alternates with loop detection, network softwarization (netsoft)", *1st IEEE Conference.*, 2015.

[11] Frank Slyne David B. Payne Marco Ruffini Nattapong Kitsuwan, Seamos McGettrick, "Independent transient plane design for protection in openflow based networks", *IEEE/OSA*, vol. 7, March 2015.

[12] Benjamin J. van Asten Niels L. M. van Adrichem and Fernando A. Kuipers, "Fast recovery in software defined networks in ieee software defined networks", *Third European Workshop*, vol. 7, September 2015.

[13] Nor Masri Sahri and Koji Okamura, "Network research workshop proceedings of the asia pacific advanced network", *IEEE 22nd International Conference on Network Protocols.*, vol. 38, pp. 23–28, 2014.

[14] Kai Gao Ze Chen Jianping Wu Yonghong Fu, Jun Bi and Bin Hao, "Orion: A hybrid hierarchical control plane of softwaredefined networking for large scale networks", *IEEE 22nd International Conference on Network Protocols.*, 2014.