NTNU – Trondheim
Norwegian University of
Science and Technology

Stochastic Switching Using OpenFlow

Komail Shahmir
Shourmasti

Master of Telematics - Communication Networks and Networked Services (2
Submission date: July 2013
Supervisor: Bjarne Emil Helvik, ITEM
Co-supervisor: Otto Wittner, UNINETT

Norwegian University of Science and Technology
Department of Telematics

NTNU

Department of Telematics

Stochastic Switching Using OpenFlow

Komail Shahmir Shourmasti

07/01/2013

Thesis Description

Stochastic routing is an approach in which choosing the egress port for traffic towards a specific destination among possible outgoing ports follows a probability distribution. This probabilities might change based on the current network behavior. Stochastic routing has properties that make its interesting for interesting for distributed, adaptive, autonomous routing systems, similar to the CEAS developed at the Department of Telematics.

Software defined network is an approach to networking in which control plane is decoupled from data plane and tasks related to control plane are offloaded to a piece of software called controller. SDN is claimed to have more flexibility than the legacy networks and provides abilities to innovate faster. Typical software defined network consists of dumb forwarding boxes, controller(s) and a communication protocol between these two components. The controller act as the brain of a network e.g. providing forwarding information to forwarding boxes (switches).

Although the main forwarding information gathering tasks take place in controller, yet, having a standardized communication protocol (e.g. OpenFlow) between controller and switch helps developers to extend functionality of an SDN environment even more.

Thesis Objectives:

The purpose of this work is to investigate the feasibility of stochastic routing in an SDN environment with focus on OpenFlow capabilities and constraints and, if relevant, to suggest modifications enable OpenFlow support this type of routing techniques.

The expected outcome of the thesis, is to demonstrate the feasibility, with or without extensions to OpenFlow, a design that enables OpenFlow to support stochastic routing will be provided. As far as time allows a proof of concept demonstration will be pursued.

Methodology:

In order to propose a stochastic routing design, we need to understand the requirement of such a routing system as well as features that OpenFlow protocol as a well-known SDN southbound protocol provides. Finding a relevant match between requirements and features will let us to implement the system on a SDN controller. The following questions will be addressed:

- What is the requirment of a stochastic routing design ?
- What routing techniques OpenFlow supports?
- Is OpenFlow able to support stochastic routing based on existing specifications ?

- If needed, what extensions are required for OpenFlow in orther to support stochastic routing ?

The work is divided in study, design, development and demonstration steps (step 3 and 4 depends on the remaining time). The output of each phase will be the input for the next step. The following task will be done in each step:

Step 1:

- Study the Software defined networking concepts
- Study the specifications and functionalities of OpenFlow protocol and SDN candidate elements such as Floodlight controller and Open vSwitch
- Investigate potential OF features to support stochastic routing
- If required functionalities are not supported in current version of OpenFlow, suggested modifications will be proposed

Step 2:

- Design a solution to support stochastic routing using OpenFlow and review the requirements (based on OF's existing features or proposed modifications to the existing functionalities)
- Getting familiar with Mininet as a virtual test environment

Step 3:

- Applying/appending suggested changes to an OpenFlow-enabled switch or presenting desired changes in pseudo code format.

Step 4:

- Run the system and examine some scenarios (on Mininet or other test environment)
- Verify the results of examined scenarios

Deliverable:

A design for OpenFlow to support stochastic routing. The design will be based on OpenFlow existing features or possibly with suggested changes to existing capabilities. A validation/proof of concept of the design as far as time allows, tentatively as an experimental setup.

Assignment given: January 2013
Supervisor: Bjarne Emil Helvik, ITEM
Co-supervisor: Otto J Winttner, UNINETT AS

Abstract

In this thesis, feasibility of performing stochastic switching using OpenFlow in SDN environments was investigated. In this work, stochastic switching is defined as forwarding incoming packets from ingress ports to one of possible egress ports according to the predefined output probabilities associated with each egress port.

Different scenarios to perform stochastic switching using OpenFlow were examined and advantages and drawbacks of each scenario was outlined. Through the investigation, SELECT method of OpenFlow group feature was found useful to execute load sharing algorithms. Since there is not any predefined SELECT function in OpenFlow specifications, a SELECT function was defined to execute stochastic output port selection according to the predefined egress port probabilities assigned to each egress port for that specific packet or flow. The defined SELECT function was implemented in an OpenFlow 1.3 enabled virtual switch (OF13SoftSwitch) in Mininet software emulator. The results of conducted tests revealed that the defined SELECT method works properly.

The results of this thesis might contribute to future research on developing stochastic routing module in OpenFlow controller in the SDN architecture.

Acknowledgment

This master's thesis has been written as the final part of a Master of Science degree in Telematics at the Norwegian University of Science and Technology (NTNU).

I would like to express my deepest appreciation to Professor Bjorne E. Helvik and Otto J Wittner for their immense and tireless help, constant encouragement and for the conceptual ideas throughout the course of this work.

Komail Shahmir Shourmasti

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Stochastic routing is a routing approach in which packets are forwarded to the specified egress port of a forwarding box according to the predefined probabilities of each output port for that specific flow or packet. Today's network related demands of businesses especially in cloud environments are rapidly changing and network operators need to respond to these demands as quickly as possible and try to meet requirements when needed. Stochastic routing is an option when we are facing non deterministic parameters in a network such as unknown demand size, delay requirements, businesses special needs, failure or even for security reasons.

Software defined networking (SDN) is a new networking paradigm which separates and abstracts data plain and forwarding plain in a network. Network intelligence is logically centralized in controller layer and abstracted from underlying physical network. SDN results in more efficient network management, more flexibility in response to demands and faster innovation. OpenFlow is an open communication protocol which connects the controller layer and the infrastructure layer of the SDN architecture. OpenFlow introduces new features that enables us to create and manage networks which are not possible to create and manage with IP and Ethernet protocols.

This work could be considered as a part of stochastic routing in a SDN environment. In this report stochastic routing is divided into two parts as shown in figure 1.1.

With regard to the previous division of stochastic routing in a SDN environment, in this report following terms are defined as:

- **Routing/forwarding:** Providing path information between sources and direct packet between source and destination.
- **Switching:** Receiving incoming packet and forward it to output port by a switch. In this report stochastic switching means directing incoming packet to an output port among set of posible output ports according to predefined output port probability associated with each output port.

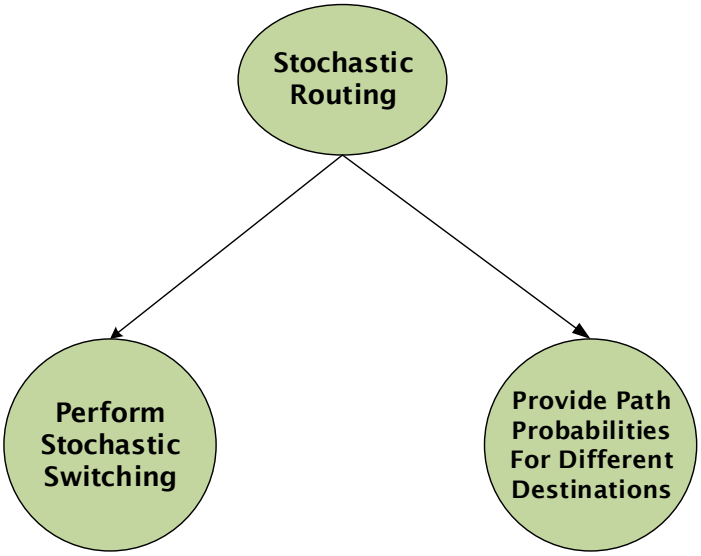

Figure 1-1 Stochastic routing division in two steps

- ✓ **Note:** In this report virtual switch and software switch terms are used interchangeably.

Research question

Studies of this report is done to answer the following question:

- Is it feasible to perform stochastic switching using OpenFlow protocol in a SDN environment?

1.1 Related work

Many routing protocols have been implemented using OpenFlow protocol in a software defined network (SDN) environment. MPLS (Sharafat et al., 2011) and shortest path forwarding (Soeurt and Hoogendoorn, 2012) are of well known protocols that have been implemented using OpenFlow.

In this report in order to focus more on similar topics to the topic of this thesis stochastic routing is categorized in non-equal cost load balancing category. Although, the work on this thesis is not considered as a load balancing approach, results of this study can contribute to further researches when stochastic routing is the subject of research. Since at the time of writing this

report, no specific work related to stochastic routing or switching using OpenFlow has been found and also considering that Load balancing mainly is done in per flow basis while stochastic routing requires deciding an output port in per packet basis, some references are made to other works that have been done on load balancing using OpenFlow.

The approach introduced by (Wang et al., 2011) proposes an algorithm to divide sets of flows over server replicas in a data center. OpenFlow wildcard rule was used to aggregate sets of microflow as a larger flow and direct each larger flows to one of server replicas. Using wildcard rules leads to better scalability and less flow rules in flow table of a switch. (Handigol et al.) Proposed a load balancing system to manage response time of web servers by adding or removing computing resources in a network. OpenFlow protocol is used to gather statistics and manage flow rules. (Uppal and Brandon, 2010) Added a load balancing module to the NOX OpenFlow controller to dictate load balancing policy to the OpenFlow switch. added load balancing module supports random, round robin and load-based policies to divide traffic between server replicas. In (Koerner and Odej, 2012) the consentration is on removing dedicated load balancing hardware which are expensive from networks and integrate the load balancing functions with forwarding boxes in a network. Furthurmore, using this approach and distribute load sharing functionality among forwarding boxes decreas the probability of accuring single point of failure in a data center. In this scenario different OpenFlow controllers are in charge of program forwarding boxes to direct each traffic type to the responsible servers. Flowvisor (Sherwood et al., 2009) is used to direct messages between forwarding boxes and OpenFlow controllers in a network. Zoltan Lajos Kis, research fellow at Ericsson in Hungary, developed an OpenFlow 1.1 compatible user space software switch (Lajos Kis, 2011) and implemented weighted round robin policy using SELECT method of OpenFlow group feature.

1.2 Methodology

During working on this thesis a looped series of steps have been considered. The steps are study, experiment, evaluation until the final scenario has been achieved.

Below is the approach we took while working on this thesis:

- Study stochastic routing to get familiar with stochastic routing concept (CEAS, a stochastic routing system developed at department of telematics at NTNU was the subject of this study part).

- Study OpenFlow 1.3 specifications to gain knowledge about latest OF version features and capabilities. Because of lack of documentation, in order to realize how some features work we needed to study products that implemented and support OpenFlow version 1.1 such as Open vSwitch (OVS) (virtual switch developed by NICIRA) and OF13SoftSwitch (a virtual switch developed by CPqD and based on Ericsson TrafficLab SoftSwitch).
- Build OpenFlow test environment. Mininet was chosen as our test bed. Run test scenarios to gain experience and get familiar with Mininet commands, virtual switch configurations and commands, DPCTL management tool and also work with OpenFlow features that considered helpful to our research (such as Group feature).
- Propose and evaluate scenarios to explore the capability of scenarios to perform stochastic switching and test them in Mininet.
- Propose our SELECT function that makes OpenFlow Group feature able to support stochastic switching and program it in OF13SoftSwitch.
- Test the proposal functionality and draw conclusions.

1.3 Thesis Organization

The rest of this thesis is organized as follow:

Chapter 2:

- A brief introduction to SDN and the role of OpenFlow in a SDN architecture.
- Introduce OpenFlow and describe main functionalities of OpenFlow

Chapter 3:

- Possible scenarios to perform stochastic switching using OpenFlow were investigated

Chapter 4:

- Introduces our test environment (Mininet and OpenFlow enabled software switch)
- A pretest has been done to test the functionality of proposed SELECT function to support stochastic switching

- Main test scenario has been explained

Chapter 5:

- Results of pretest and main test scenarios is presented and discussed

Chapter 6:

- Consists of conclusion and future work

Chapter 2: Introduction to OpenFlow

In this chapter, OpenFlow basic features are described. Since OpenFlow plays a key role in Software Defined Networking (SDN) architecture, In order to explain OpenFlow it is helpful to first take a look at software defined networking (SDN) concepts and architecture.

2.1 Software Defined Networking (SDN)

Software defined networking (SDN) is a new approach to networking and some experts believe it is a revolution in networking field. SDN advocates claim that if we continue to create and maintain networks in the way that we have been doing it in the past two decades and creating new mechanisms to meet our growing, facing challenges in networking, we will end up with having complex networks with too many efforts to have them working, regardless of the fact that the network completely meets our demands or not (Shenker, 2011).

The main characteristic of the SDN architecture is that control plain and data plane are decoupled and abstracted from each other. In this architecture network intelligence is moved to the control layer of the SDN architecture and abstracted from underlying network infrastructure which operates at control/infrastructure layer and connected through proper APIs. This separation makes developers and researchers able to better focus on each layer without considering the complexities of other layer. Programmability is a key feature of the SDN architecture that enables enterprises and carriers to adopt to rapidly changing business demands in more flexible and automated manner (ONF, 2012) (NICIRA, 2012). Figure 2.1 depicts SDN architecture.

The main characteristics of SDN include (NICIRA, 2012):

- Control and data planes are decoupled and abstracted from each other
- Intelligence and state are logically centralized, result in having a global view of network and changing demands

- Underlying network infrastructure abstracted from applications, which makes it possible to develop different applications according to the needs
- Programmable data plane brings automation and flexibility to networks
- Faster innovation

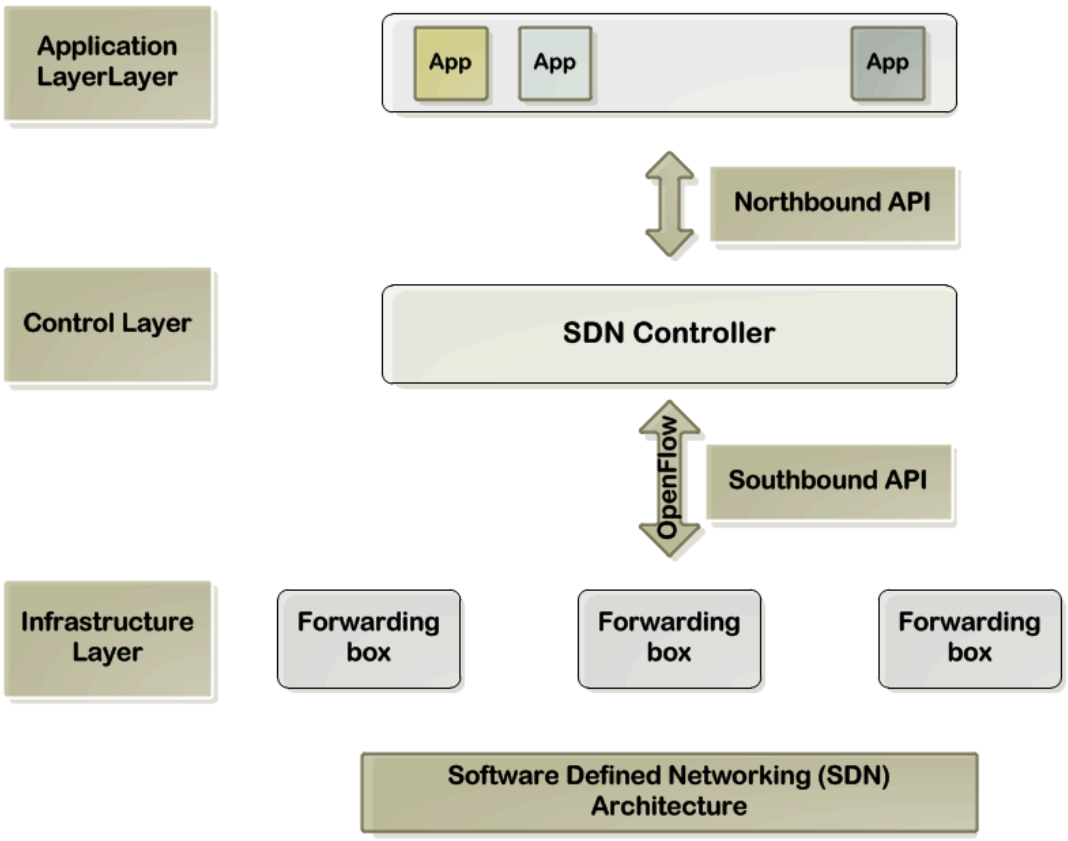

Figure 2-1 SDN Architecture

The essence of software defined networking (SDN) is to change the way that we create and manage networks. OpenFlow plays a key role as so called southbound API between control layer and infrastructure/switching layer i.e. it enables communication between SDN controller and OpenFlow-enabled switch. For interested readers there is a presentation by Scott Shenker professor at university of California in Berkeley and co-founder of NICIRA networks discussing about motivations toward software SDN an OpenFlow available on YouTube

2.2 OpenFlow

OpenFlow is an open communication protocol that enables SDN controller to program flow table of forwarding boxes in a network. Primary aim of OpenFlow is to make researchers able to experiment with new networking protocols on both research and production networks (McKeown et al., 2008).

2.2.1 OpenFlow Architecture

OpenFlow architecture consists of OpenFlow controller(s) that provide flow entries to the flow table(s) of an OpenFlow-enabled switch. A brief overview of OpenFlow architecture is demonstrated in figure 2.2.

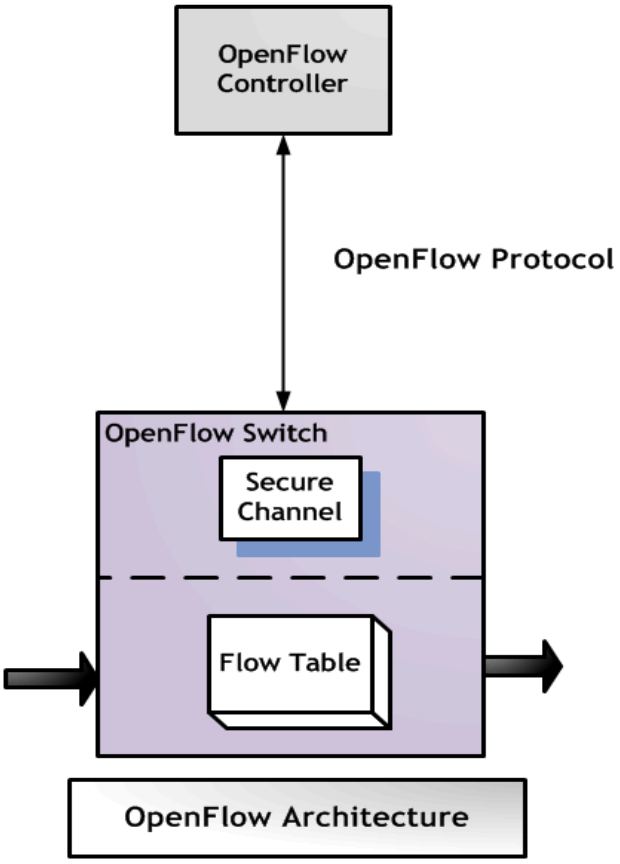

Figure 2-2 OpenFlow Architecture (OF 1.3 Specifications)

Some useful terms and definitions:

- **OpenFlow Switch:** a switch or router that supports OpenFlow protocol.

- **OpenFlow Controller:** a controller discovers network topology and run routing algorithms to find route(s) between source and destination. Then, programs switch(s) to forward frames to right output port(s). A controller uses Link Layer Discovery Protocol (LLDP) to discover topology of a network. (Has a global view of the network or the area under control of the controller).

- **Packet_in Message:** an unknown packet is encapsulated in packet_in message by switch and forwarded to the controller to provide the information to treat the packet.

- **Packet_out Message:** information in response to the packet_in message is encapsulated in packet_out message by the controller to inform the switch about how to treat the packet.

- **Flow Mode:** this message is used by a controller to inject flow rules into flow table of a switch. Flow mode message provides match criteria(s) and action for a switch to treat a flow or flows that meet match criteria(s).

- **Flow Miss Entry:** The flow entry that wildcards all fields (all fields omitted) and has priority equal to 0. Every flow table must contain a flow miss entry to define how to process a packet that does not match to all other flow entries.

- **Exact Match:** A match field is exact match when the matching value is exactly defined. An exact match is a binary match, it either matches or does not.

- **Wildcard Match:** When the match value is not important in making decisions, the match is said to be wildcarded.

- **Metadata:** A maskable register value that is used to carry information from one table to the next. Metadata field (64 bits) can be set and match in tables.

An OpenFlow switch is in charge of capturing incoming packets and matching them against flow table entries and executes the action associated with that specific entry match. An OpenFlow controller (OFC) executes routing algorithms and provides route(s) between each source and destination. OFC injects flow entries to the flow table of an OpenFlow switch. In this way controller programs switches in the zone under its supervision (Ichino, 2011).

2.2.2 Flow Table

An OpenFlow switch consists of one or many flow tables and a group table. Each flow tables contains many flow entries. A switch is in charge of the match and forwarding operations. In case of having more than one flow tables in an OpenFlow switch which is called OpenFlow pipeline, OpenFlow pipeline processing defines the way a packet interacts with these flow tables. A packet might visits all or some of flow tables in an OpenFlow pipeline depending on the outcome of table match and action operations of previous table (Pfaff, 2012). In other word, an OpenFlow switch consists of one or a chain of flow tables.

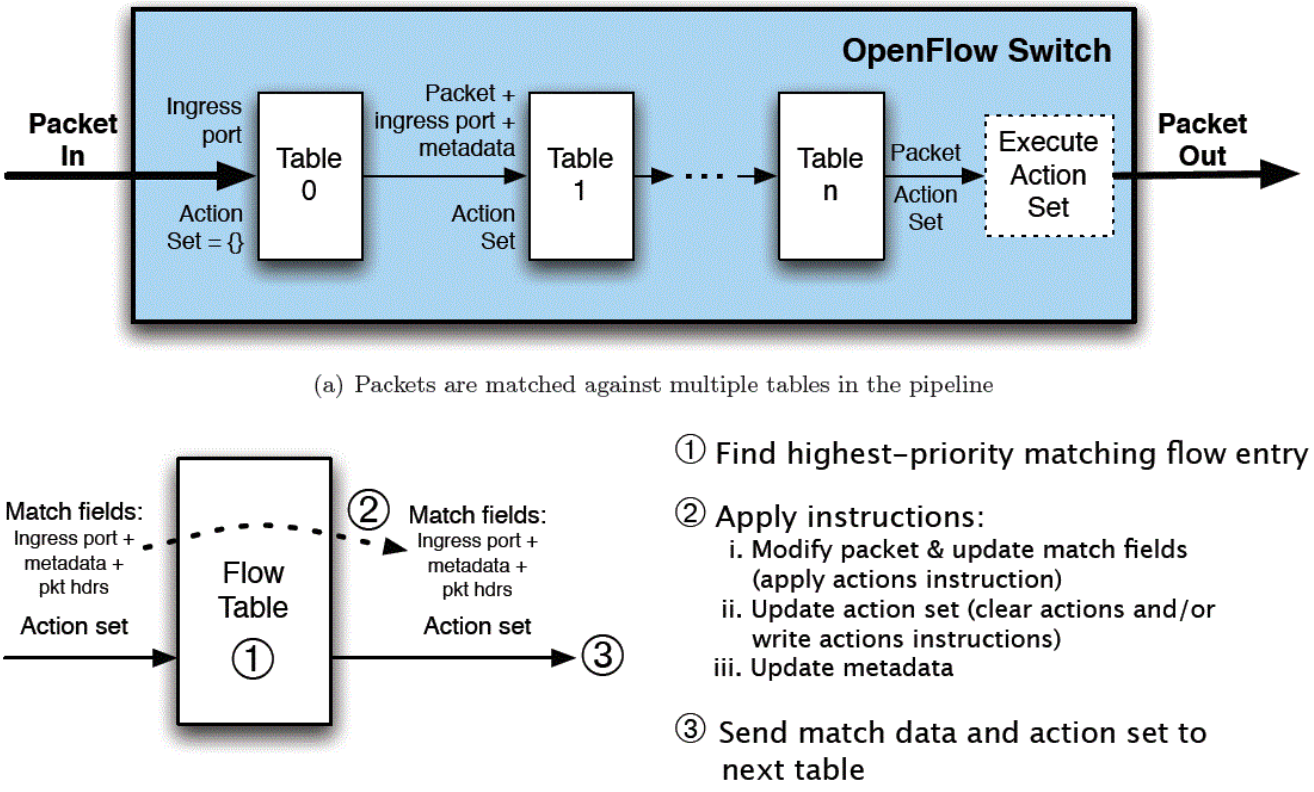

Figure 2-3 OpenFlow pipeline processing (Source: OF1.3 spec)

Any flow entries in flow table consist of two main parts. The first part is known as match that specifies conditions that a packet or a flow can match to that specific entry and the second part is action which defines the instructions to be executed. Various match fields enable OpenFlow to define flexible forwarding in a network. Components of a flow entry are shown in table 2.1. A flow table entry is identified by its match fields and priority. The match fields and priority taken together identify a unique flow entry in the flow table.

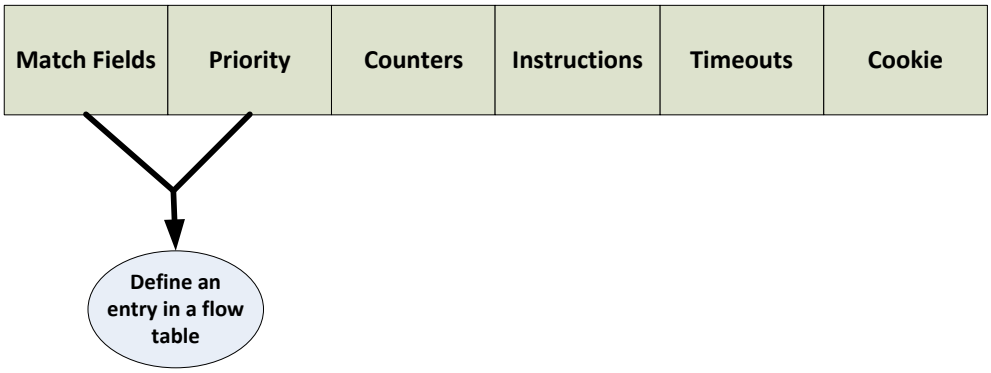

Figure 2-4 Main components of a flow entry in a flow table

Below is a brief description of each field:

- **Match fields:** To match against packets. These consist of the ingress port and packet headers, and optionally metadata specified by a previous table.

- **Priority:** Matching precedence of the flow entry.

- **Counters:** Updated when packets are matched.
- **Instructions:** To modify the action set or pipeline processing.

- **Timeouts:** Maximum amount of time or idle time before flow is expired by the switch.

- **Cookie:** Unique data value chosen by the controller. May be used by the controller to filter flow Statistics, flow modification and flow deletion. Not used when processing packets.

2.2.3 Match field structure

OF1.3 specification defines 13 required match fields that an OpenFlow enabled switch is required to support. Table 2.2 lists required match fields. More information about implementation of match fields is available on section 7.2.3.7 of OF1.3 specifications.

Fields	Description
IN_PORT	Ingress port. This may be a physical or switch-defined logical port.
ETH_DST	Ethernet destination address. Can use arbitrary bitmask
ETH_SRC	Ethernet source address. Can use arbitrary bitmask
ETH_TYPE	Ethernet type of the OpenFlow packet payload, after VLAN tags.
IP_PROTO	IPv4 or IPv6 protocol number
IPV4_SRC	IPv4 source address. Can use subnet mask or arbitrary bitmask
IPV4_DST	IPv4 destination address. Can use subnet mask or arbitrary bitmask
IPV6_SRC	IPv6 source address. Can use subnet mask or arbitrary bitmask
IPV6_DST	IPv6 destination address. Can use subnet mask or arbitrary bitmask
TCP_SRC	TCP source port
TCP_DST	TCP destination port
UDP_SRC	UDP source port
UDP_DST	UDP destination port

Table 2.1 Required match fields and descriptions.

Source (OF1.3 Spec)

Match fields are a combination of layer 2 to layer 4 match fields. Compared to switches which operate at layer 2, routers at layer 3 and firewalls at layer 4 [Network Static], This means having

more match options which results in more flexibility to define wider range of rules to a flow table of an OpenFlow-enabled switch. Value of a match field can be either Wildcarded or exact.

2.2.4 OpenFlow actions

OpenFlow 1.3 specification defines 2 types of actions for a switch. Table 2.2 describes required and optional actions. A switch must support required actions. An OpenFlow controller can query the switch about optional actions a switch supports.

Action	Description
Output (required)	Forwards a packet to a port
Set-Queue (required)	Sets queue ID for a packet
Drop (required)	Drops a packet
Group (required)	Process the packet through the specified group
Push-Tag/Pop-Tag (optional)	Push and pop VLAN, MPLS, PBB tags
Set-Field (optional)	Modifies value of a packet header field
Change-TTL (optional)	Modifies value of TTL

Table 2.2 OpenFlow actions

2.2.5 Instructions

When a packet matches to a flow entry of flow table, it goes through a set of instructions that are associated with that flow entry. Instructions make changes to the packet, action set or pipeline processing. It is only allowed to have maximum one type of instructions in each instruction set of a flow entry. If two instruction of the same type needs to be executed, *apply-action* instruction needs to be executed before adding the second instruction of the same type. For more details see section 5.11 of [OF1.3 Specifications]. Instructions are listed in table 2.3.

Instruction	Description
Write-Actions (required)	Add action into action set of a packet being processed
Goto-Table (required)	Specify next table in pipeline
Meter (optional)	Direct packet to the specified meter
Apply-Actions (optional)	Apply the specific action(s) immediately, without any change to the Action Set.
Clear-Actions (optional)	Removes actions of an action list
Write-Metadata (optional)	Writes the masked metadata value into the metadata field.

Table 2.3 OpenFlow instructions

2.2.6 Action set

An action list associated with each packet when the packet enters the pipeline. Instructions of a flow entry can add, remove or execute actions of an action set. When instruction of a flow entry does not contain *GOTO-Table* instruction, pipeline processing stops and the actions in the action set are executed.

A simple relation between instructions, actions and action set is depicted in figure 2.4.

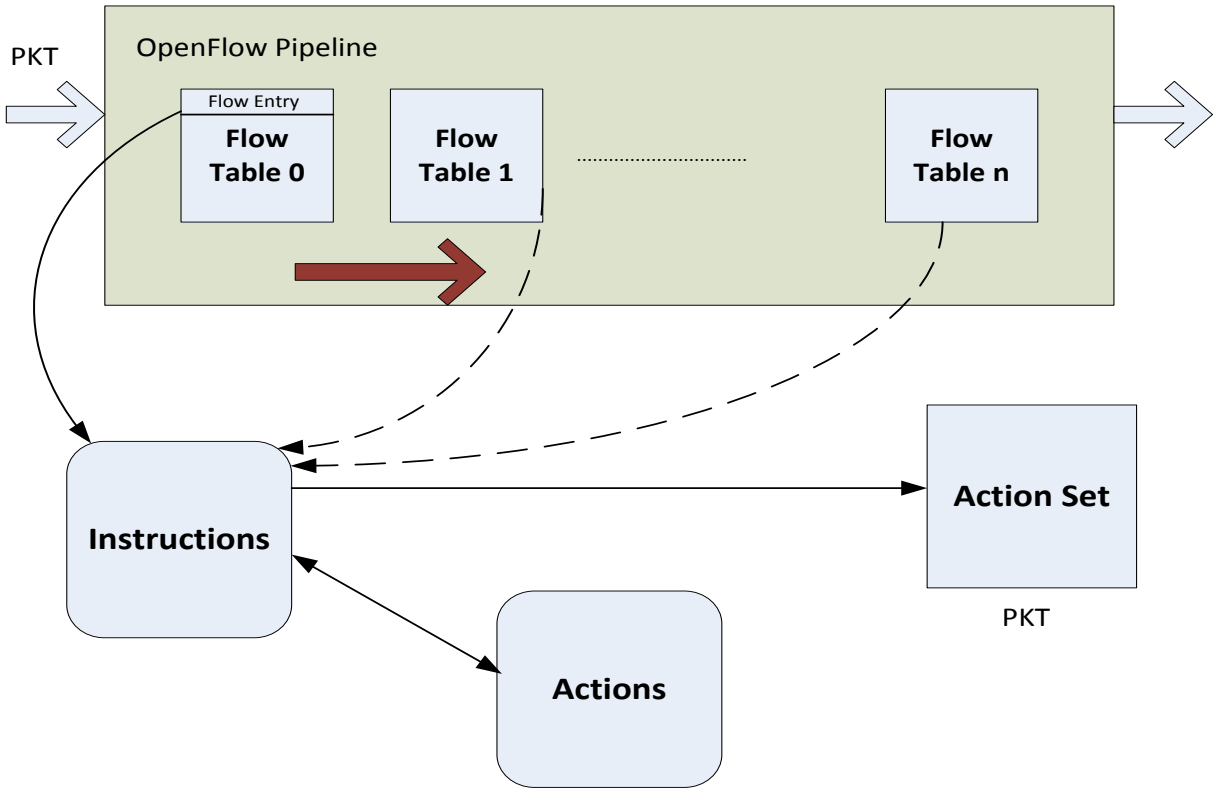

Figure 2-5 : Simple overview of relation between actions, instructions and action set

2.2.7 OpenFlow Group Feature

Group feature is introduced in OpenFlow 1.1. The aim is to make specific forwarding like flooding, multicast, failover and load balancing simpler. Group feature enables forwarding to all ports in a group (flooding), select among a series of ports (load balancing) and etc.

2.2.7.1 Group Table

A flow entry in a flow table can point to a group table consisting of group entries. The goal is to extend forwarding behavior to support different forwarding methods (e.g. Select and All) (Pfaff, 2012).

A group entry is identified by a 32 bits group identifier, each group entry contains:

- **Group identifier:** A 32 bits unique, unsigned integer used to identify a group
- **Group Type:** Explains the group semantic (e.g. All, Select, Failover)
- **Counters:** To provide number of processed packets by a group

- **Action Bucket(s):** An ordered list of action buckets each containing action and associated parameters to be executed

2.2.7.2 Group Types

A group type indicates the semantic of a group. There are required types as well as optional. A switch must support group types marked as required. OF controller can query switches about group types that they support (Pfaff, 2012).

Required group types are:

- All: Execute all buckets in the group
- Indirect: Execute the one defined bucket in the group

Optional group types are:

- Select: Execute one bucket in the group
- Fast Failover: Execute the first live bucket

2.2.7.3 How SELECT method of Group Feature Works

Select method is a function that introduced to ease performing load sharing in a network. Select function uses bucket weights assigned to the buckets to select one of buckets to execute the actions associated with that bucket.

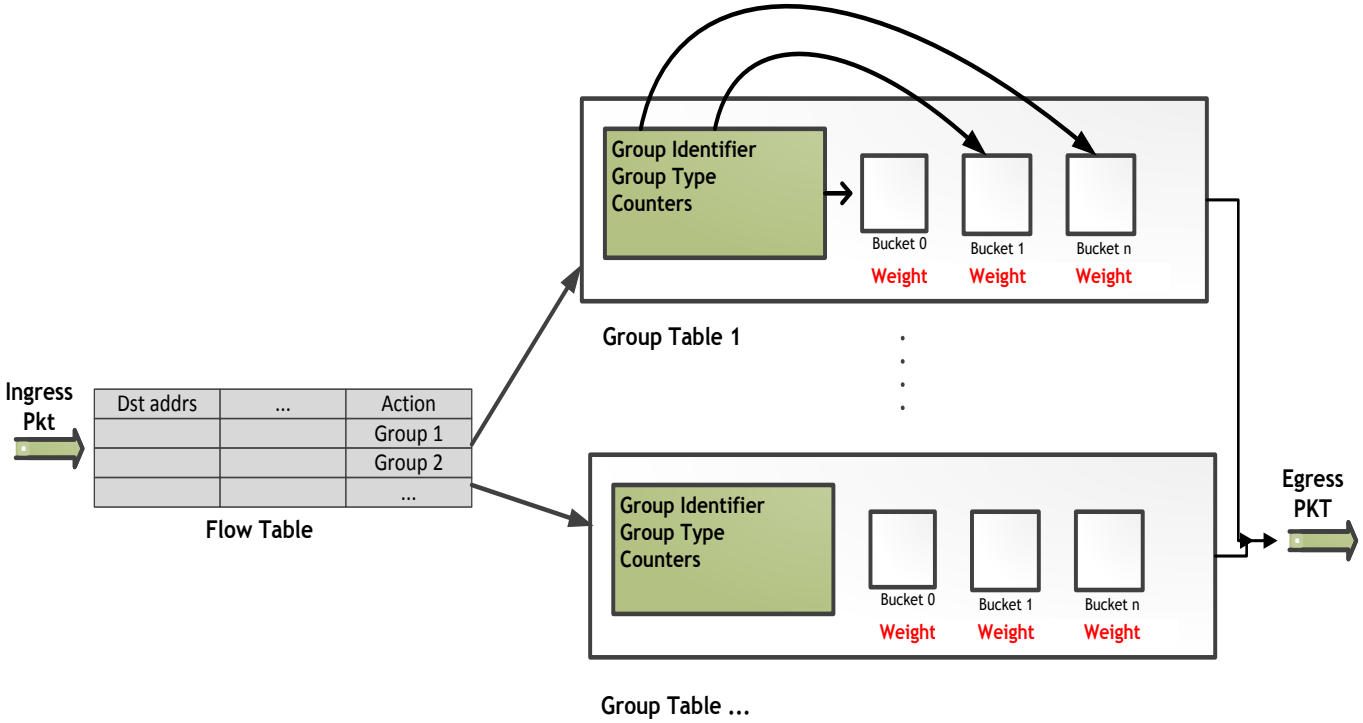

Figure 2-6 Group Table

2.3 OpenFlow proactive and reactive flow rule setup

There are two ways that a controller is able to program a switch i.e. insert flow rules to the flow table of a switch. Below is a description of these two methods. A third method can be considered by the combination of these two main methods.

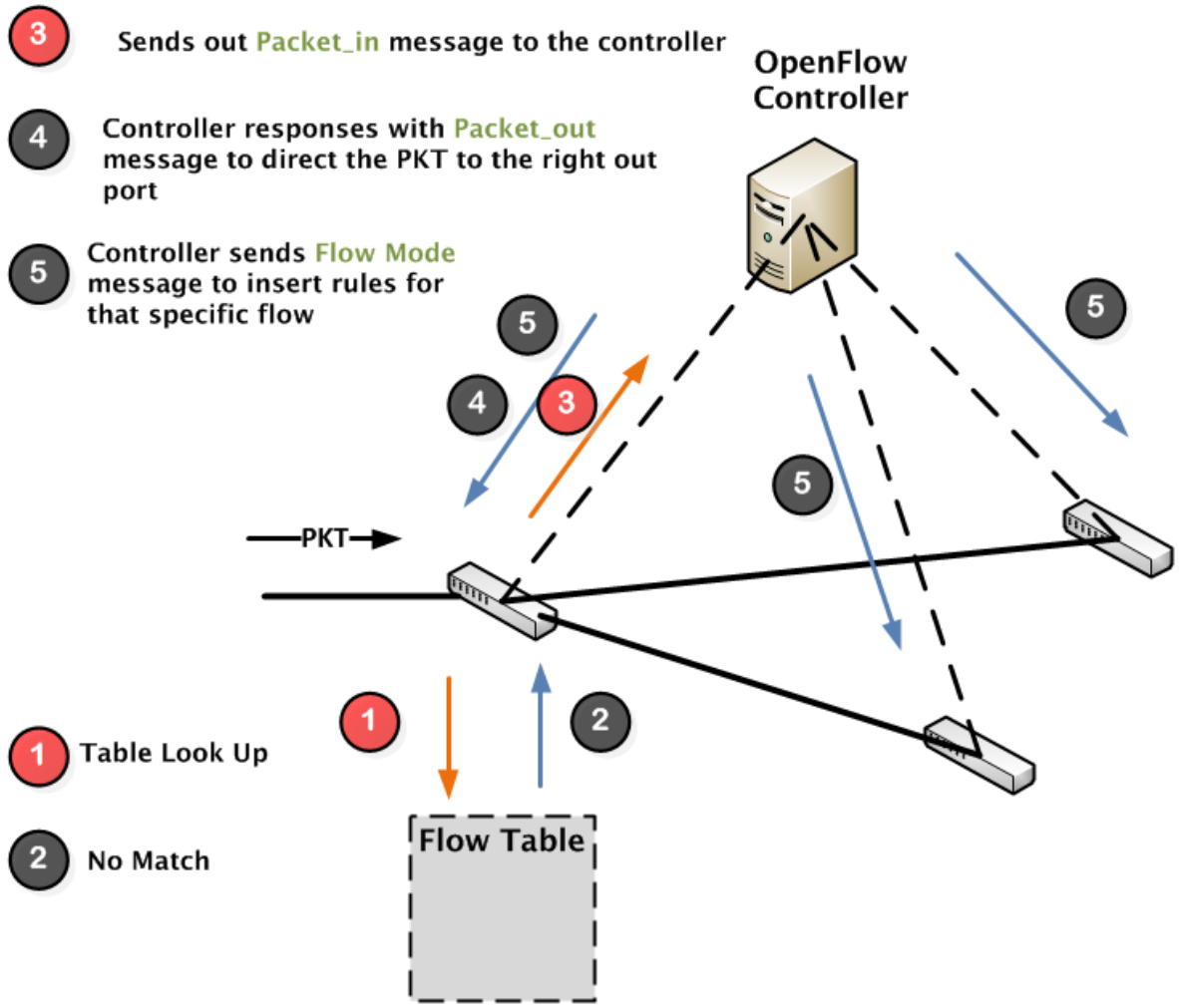

Figure 2-7 OpenFlow Reactive Model

Reactive setup is considered to be more flexible than proactive model, since controller is able to dictates logics than cannot be implemented by the switch. On the other hand, reactive model adds extra latency on the first packet of flow (OpenDaylight-Wiki, 2013).

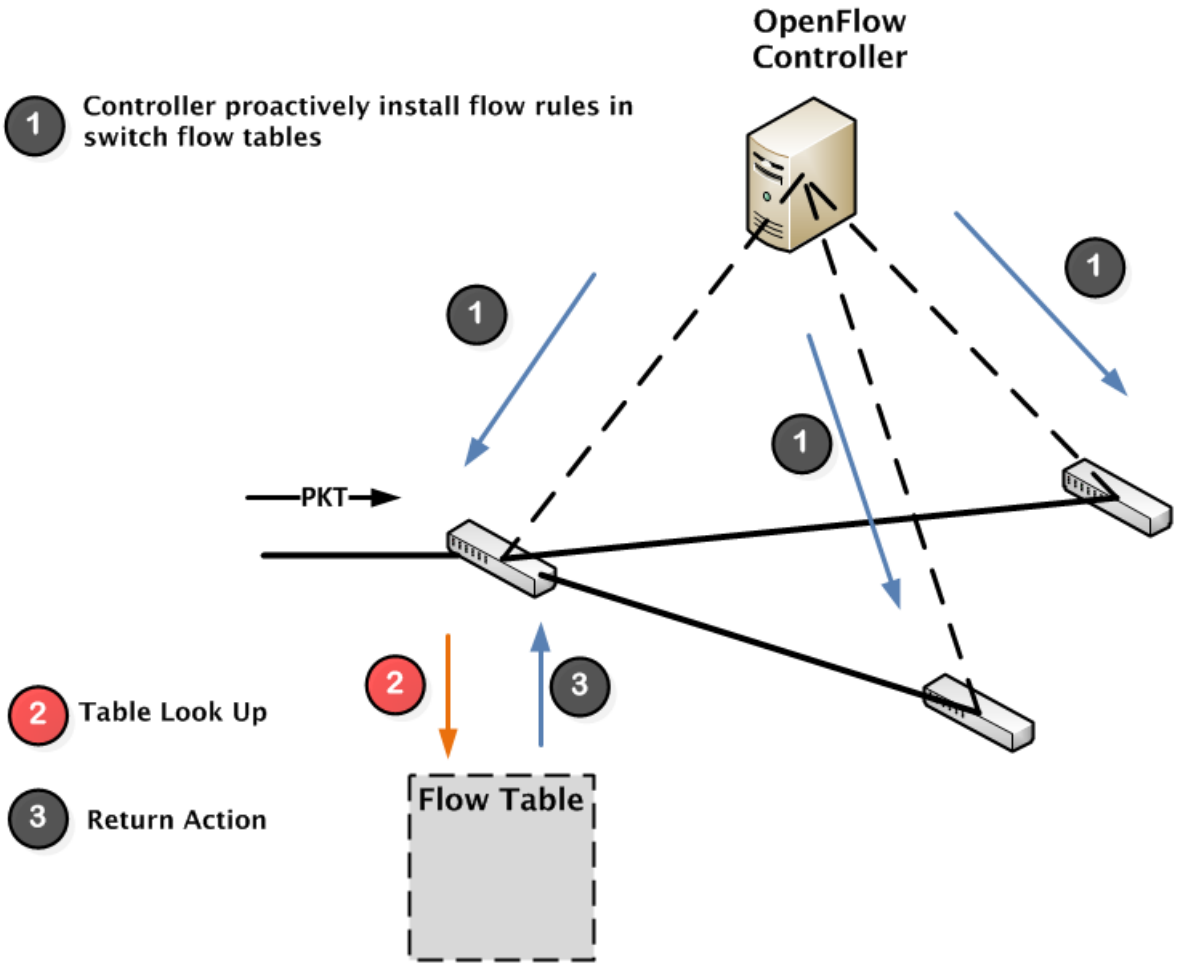

Figure 2-8 OpenFlow Proactive Model

As mentioned earlier in this chapter, OpenFlow introduces new features such as OF pipeline, Group feature, various match fields and actions that make it possible to configure more flexible networks than what are already in operation. Flexibility is the key advantages of OpenFlow compared to existing protocols such as IP and Ethernet. Generally, using OpenFlow results in the following advantages (NEC, 2010):

- **Network virtualization:** Creating and management of multiple virtual networks on the shared network infrastructure to meet the requirements and better utilization of network resources, responding faster and with less efforts to the network virtualization requirement such as VM creation and removal and movement, are some of key advantages of using OpenFlow to network virtualization (NEC, 2011).

- **Route distribution:** OpenFlow is able to handle flows inside a network more efficiently. Having a global view of network makes controller able to distribute routes according to demands and makes a balance between available resources and whats is required to handle flows inside a network.

- **Network visualization:** Control layer in the SDN architecture provides a global view of a network. OpenFlow controller collect information about statistics, devices, flows, routes between nodes and etc. This results in a better management of the network.

Considering the features and benefits of OpenFlow in the next chapter we are going to investigate feasibility of stochastic switching using OpenFlow.

Chapter 3: Stochastic Switching Using OpenFlow

In this chapter, possibility of performing stochastic switching using existing OpenFlow features was investigated. Required changes to the specification were proposed if necessary.

3.1 Stochastic Switching Using OpenFlow

Load balancing mainly is done in per flow basis while stochastic routing requires deciding an output port in per packet basis. Having multiple possible output port with different probabilities for each destination is the interest of this section.

Pardon, what was the question?

Referring to research question in chapter 1, In order to examine the feasibility of having probabilistic routing behavior in an SDN environment stochastic routing mechanism was divided into two phases. First step is to provide the output probability for each output port and for each desired destination address and the second step is to enable switching boxes to perform stochastic switching i.e. showing random behavior based on provided output port probabilities for that specific destination address.

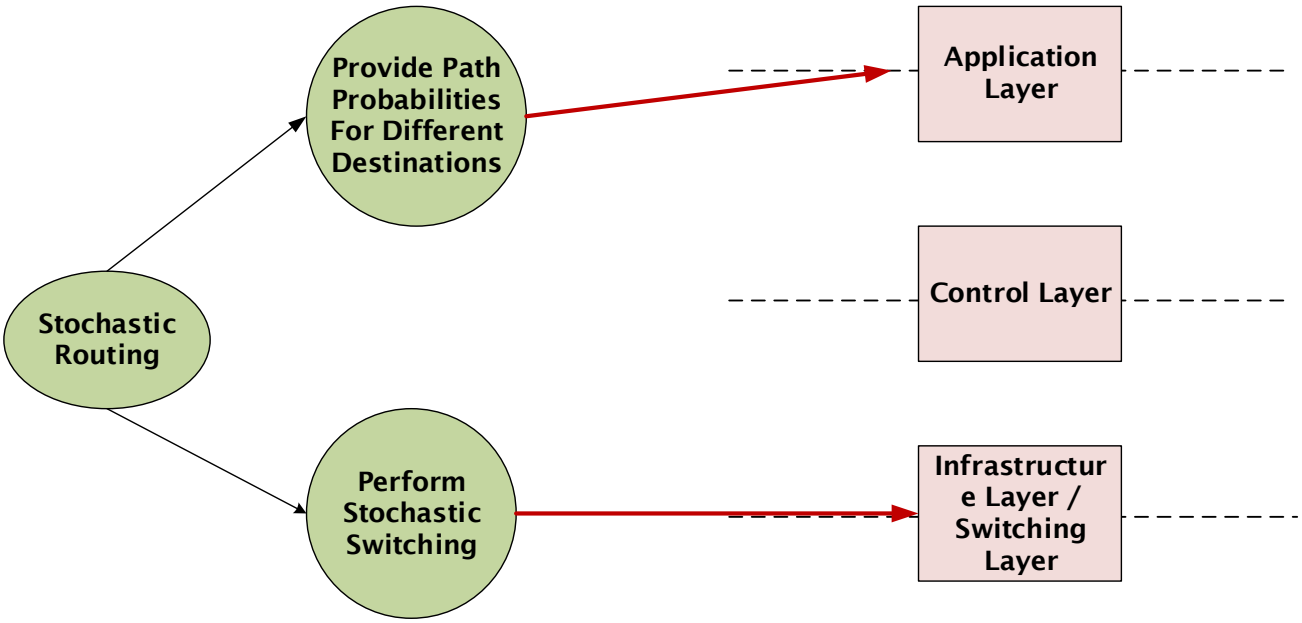

Figure 3-1 Stochastic Routing Map to SDN Architecture

OF controller takes part the main efforts to calculate output port probabilities for each destination address in a network. In this report we omit the first step and concentrate on the second part. The aim is to investigate the feasibility of performing stochastic switching according to the latest OpenFlow specification (OF 1.3) and to propose required change(s) to the current OF specification if necessary.

3.1.1 Stochastic Switching

In this section, different scenarios were investigated also advantages and disadvantages of each and configuration considerations for switch and controller mentioned.

3.1.1.1 Scenario One: Direct All Packets to the Controller

The simplest way of performing stochastic switching for a switch is to direct each packet to the controller, then stochastic routing engine/module of the controller inform the switch to which port forward the packet.

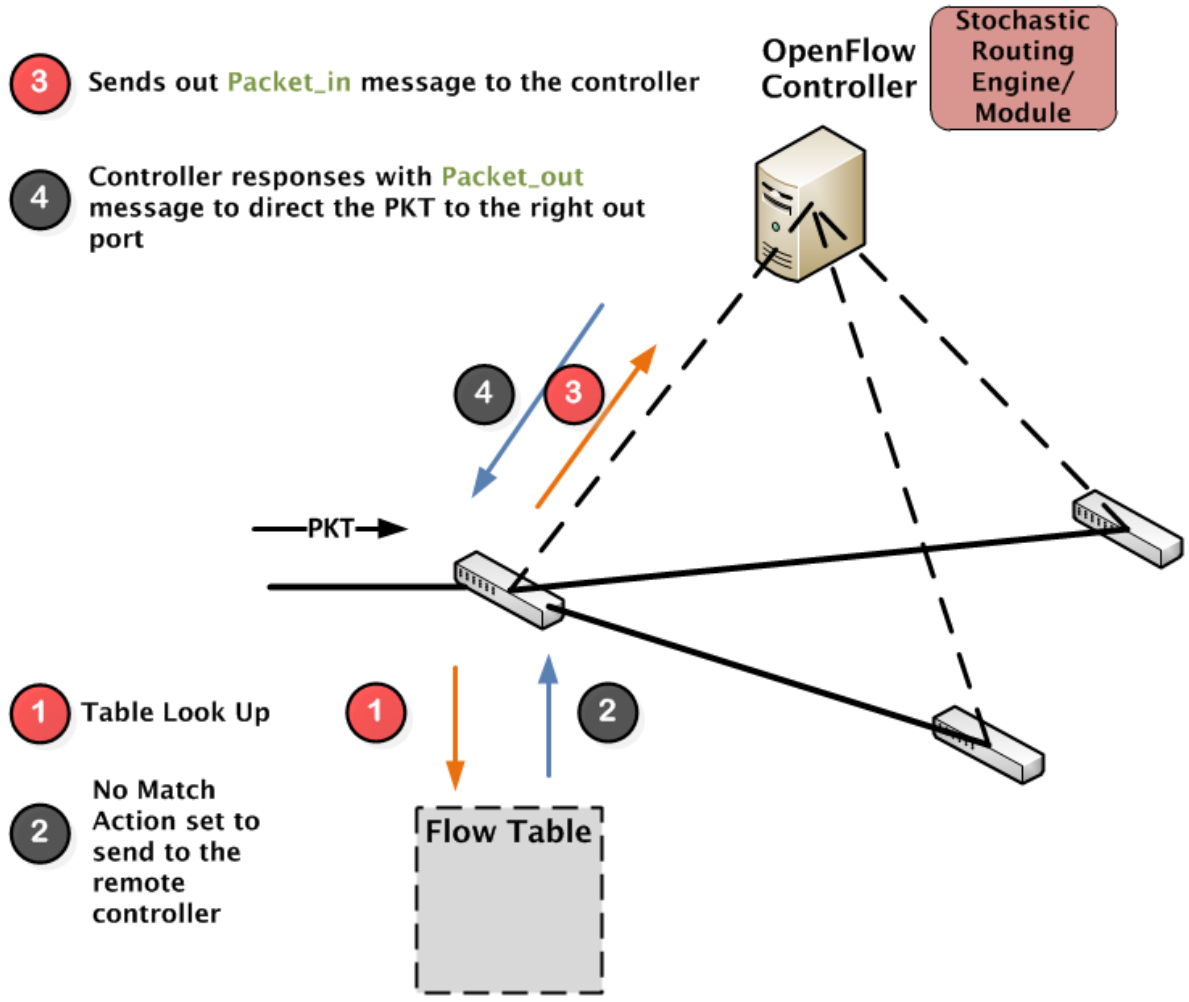

Figure 3-2 Scenario #1

Procedure: Switch receive incoming packet and match it against flow table entries, since switch is proactively configured to send each packet to the controller. Switch encapsulates the packet or part of it into the PACKET_IN message and forwards the packet to the controller. Then, controller pass to the stochastic routing module to decide which port the packet should be send

out. Then, controller informs the switch about the output port in form of PACKET_OUT message. Switch receives PACKET_OUT message and direct the packet toward output port.

Configuration requires:

- ✓ **For the switch:**
  - To be configured to direct each incoming packet to the controller.
- ✓ **For the controller:**
  - To receive and calculate the outgoing port for every packet forwarded to the controller and inform the switch about the outgoing port for that specific packet.
  - Switch does not need to install any flow rule for that specific destination address in a switch flow table.

Advantages and Limitations:

- ✓ **Advantages:**

- The switch does not need to maintain flow rules for different destinations and hence flow table size is small and does not require huge size of memory to contain flow rules.
- No changes to the current OF specification is required

- ✓ **Limitations:**

- The controller needs to calculate outgoing port for every incoming packet to the controller; hence the required amount of processing power should be provisioned.
- Since lots of packets are forwarded to the controller and send back to the switch, the link(s) between switches and controller(s) could be a potential bottleneck to the network in case of either link failure or number of packets.

- Security considerations need to be taken into account to prevent controller flooding attacks (DDOS etc.) and minimizing security threats.

- Forwarding every packet to the controller adds a round-trip delay time (RTD) to each packet.

3.1.1.2 Scenario Two: Updating Flow Table(s) In Short Time Intervals

In this scenario, controller updates switch routing table regularly in short intervals. This setup helps to prevent forwarding each and every packet to the controller. While, in a long time, switch shows stochastic-like packet forwarding, but in each interval time i.e. time between flow table updates, packet forwarding is deterministic.

Configuration requires:

For the switch:

- To forward packets as flow table dictates

For the controller:

- Updates forwarding table in short time intervals
- Hard time and soft time

Advantages and Limitations:

- ✓ **Advantages:**

- Omit RTT packet delay between switch and controller which has been introduced in the first scenario
- Remove the risk of the link between switch and controller to become a potential bottleneck
- Having stochastic-like packet forwarding in long time
- No changes to the current OF specification is required

✓ **Limitations:**

- Controller needs to update forwarding table of every switches regularly in short intervals.
- During time between forwarding table updates, packet forwarding is deterministic.

The chapter was started by designing simple scenarios such as scenarios one and two. While further studying OpenFlow features, it has come to our attention that the main purpose of introducing SELECT method of OpenFlow group feature is to ease performing load sharing. Since the functionality of this method is not elaborated in the specifications, we started by using SELECT method concept and proposing our solutions based on the concept.

3.1.1.3 Scenario Three, using group feature

This is the first design using OpenFlow group feature introduced in OpenFlow 1.1. Group feature gives the ability to assign flows that have same destination or common routes toward their destination to a group.

In this scenario, flows with the same best egress port among possible egress ports are categorize in same group. For instance, all flows for which port number 3 is the best egress port i.e. having higher egress port probability will be assigned to group 3. Number of group tables are equal to the number of physical ports of a switch. An example of this scenario is shown in figure 3.3.

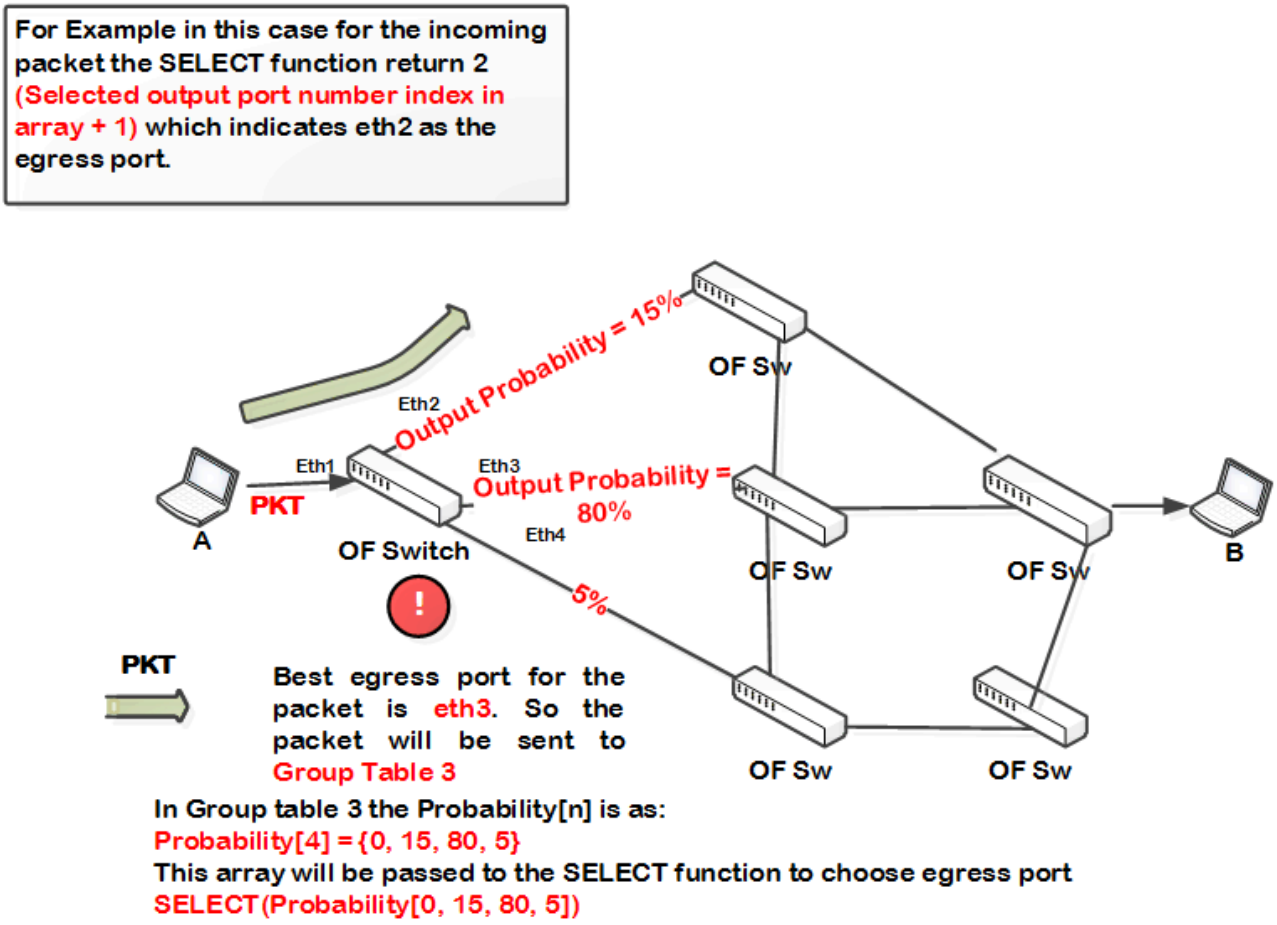

Figure 3-3 an example using scenario 3

Design requires:

- Number of groups is at least equal to the number of physical ports of a switch.
- Each group holds an array containing probabilities for all possible output ports beside the best one
- Groups are chosen based on the best egress port for that specific flow (Each physical port has a group table).
- The stochastic routing module added in OF controller will provide the probabilities for all possible out ports toward the destination of that packet or flow)
- The SELECT algorithm generate random number and according to the provided probabilities decides which output port is chosen for the packet

An overview of the scenario is shown in figure 3.4.

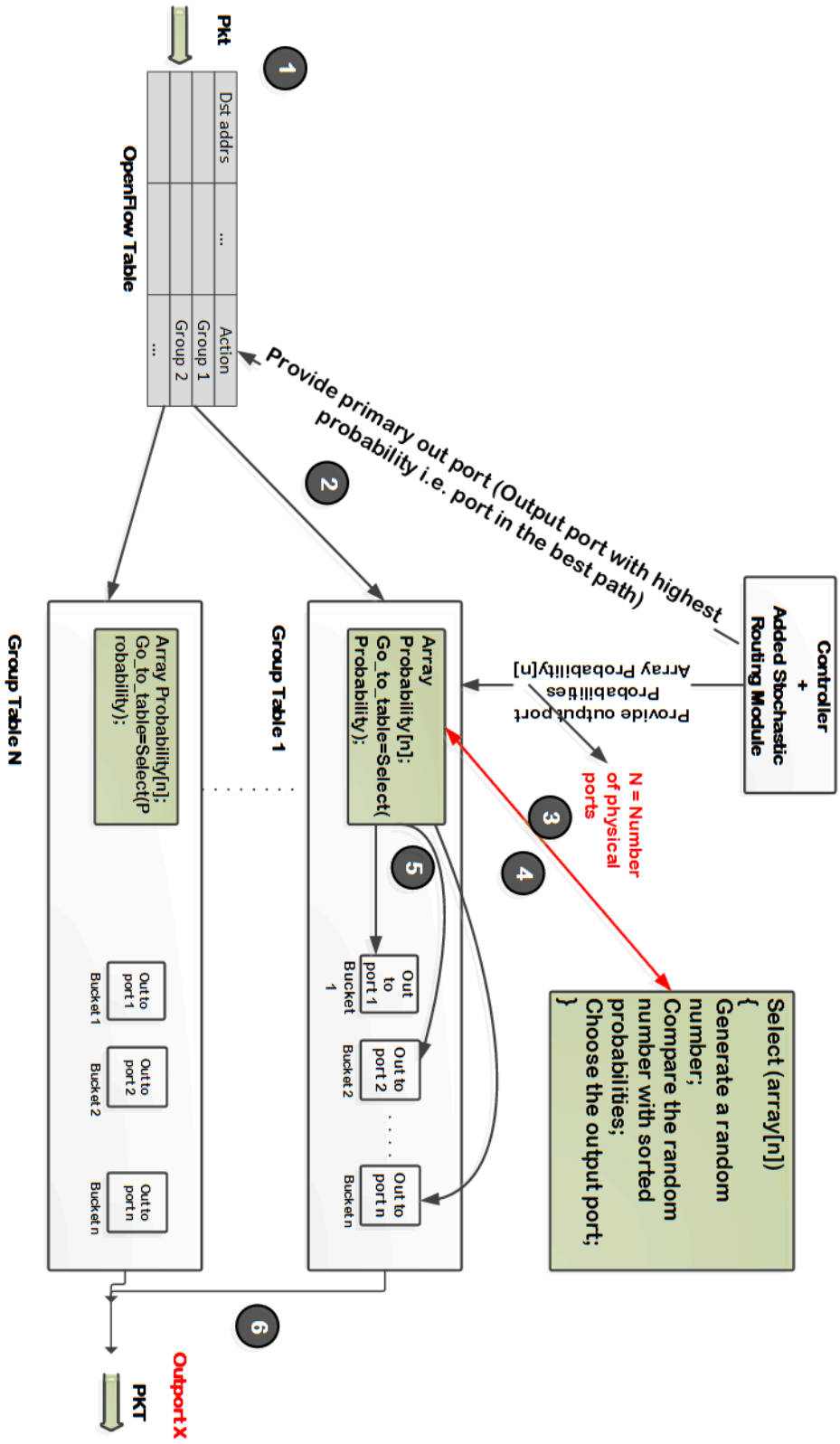

Figure 3-4 Overview of design #3

Configuration requires:

- ✓ **For the switch:**
  - To direct packets to groups
- ✓ **For the controller:**
  - To calculate route between sources and destinations
  - Create a unique group associated with each physical port of a switch
  - Program the switch to direct packets to available groups.

Advantages and Limitations:

- ✓ **Advantages:**

- Remove RTT packet delay between switch and controller which has been introduced in the first scenario
- Remove the risk of the link between switch and controller to become a potential bottleneck

- ✓ **Limitations:**

- Requires change to the OF specification. An array to hold the probability set of each group needs to be defined. Also, select method needs to accept the array as an argument.
- Another limitation is although, all flows that are assigned to a group have the same best egress port but, not all other possible egress ports for 2 different flows are not necessarily the same.
- Finding proper values of egress port probabilities is another challenging task. Having many flows with the same best egress port and different value for other ports, makes it difficult to calculate a common set of egress port probabilities for each group.

Considering the limitations mentioned above, this model does not work properly.

3.1.1.4 Scenario Four, enhancing scenario three

Concentration of this design is on removing limitations of previous design as much as possible. In previous design having a common best port was the requirement to place different flows into a common group. That criteria, causes the limitations mentioned in previous section.

Procedure:

Controller calculate routes and perform address aggregation to categorize as much as flows into a flow entry of a flow table. Then, controller create a group for that specific set of flows. Since, these flows reside in the same group they have the same set of egress port and probabilities for each egress port. Controller provide an array of egress port probabilities for each group. A proper select function needs to be defined and accept the probabilities array as an input. Then select function chooses one egress port among the set of possible egress ports according to provided probabilities by the controller. An overview of the design is shown in figure 3.5.s

Design requires:

- Controller to create a group for each aggregated set of addresses
- Each group holds an array containing probabilities for all possible output ports
- The stochastic routing module added in OF controller will provide the probabilities for all possible out ports toward the destination of that packet or flow
- The SELECT algorithm generate random number and according to the provided probabilities decides which output port is chosen for the packet

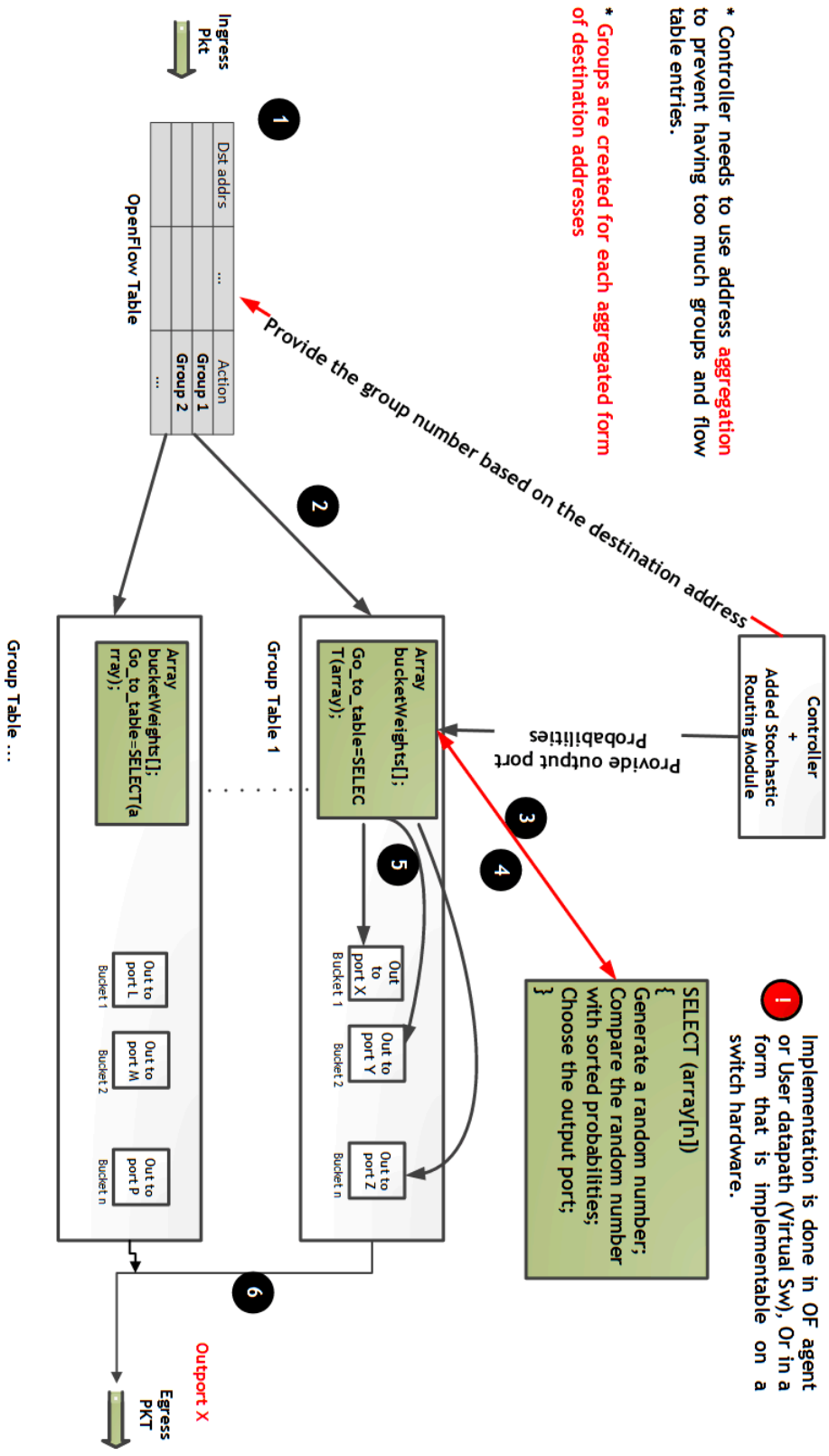

Figure 3-5 Overview of scenario number Four

Advantages and Limitations:

✓ Advantages:

- Remove RTT packet delay between switch and controller which has been introduced in the first scenario
- Remove the risk of the link between switch and controller to become a potential bottleneck
- Implementable in virtual switch and OF agent of a hardware switch

✓ Limitations:

- Requires change to the OF specification. An array to hold the probability set of each group needs to be defined. Also, select method needs to accept the array as an argument.
- Number of groups should be less than or equal to the maximum number of groups that a switch supports

3.1.1.5 Final model: Using Bucket Weight and Defining SELECT Function to Support Stochastic Switching

In order to learn the functionality of SELECT method OFSoftSwitch, developed at Ericsson lab, (implemented weighted round robin algorithm as a load sharing mechanism using SELECT method in their open source virtual switch) was studied.

In a private talk with Zoltan Lajos Kis, the developer of OFSoftSwitch and a fellow researcher at Ericsson in Hungary, his opinion about design number 4 was asked. He suggested to use bucket weights instead of defining new array to hold port probabilities similar to their implementation of round robin algorithm in OFSoftSwitch. By following his advice the final design was created.

Procedure:

Procedure is the same as previous design except in this model the need to create an array for each group to hold port probabilities are omitted. The OpenFlow specification introduces weight for each action bucket in a group table. The controller assigns egress port probabilities to this weight variable of each bucket. Each bucket contains the *out_to_port* action which forwards packets to an egress port. An overview of the design is shown in figure 3.6.

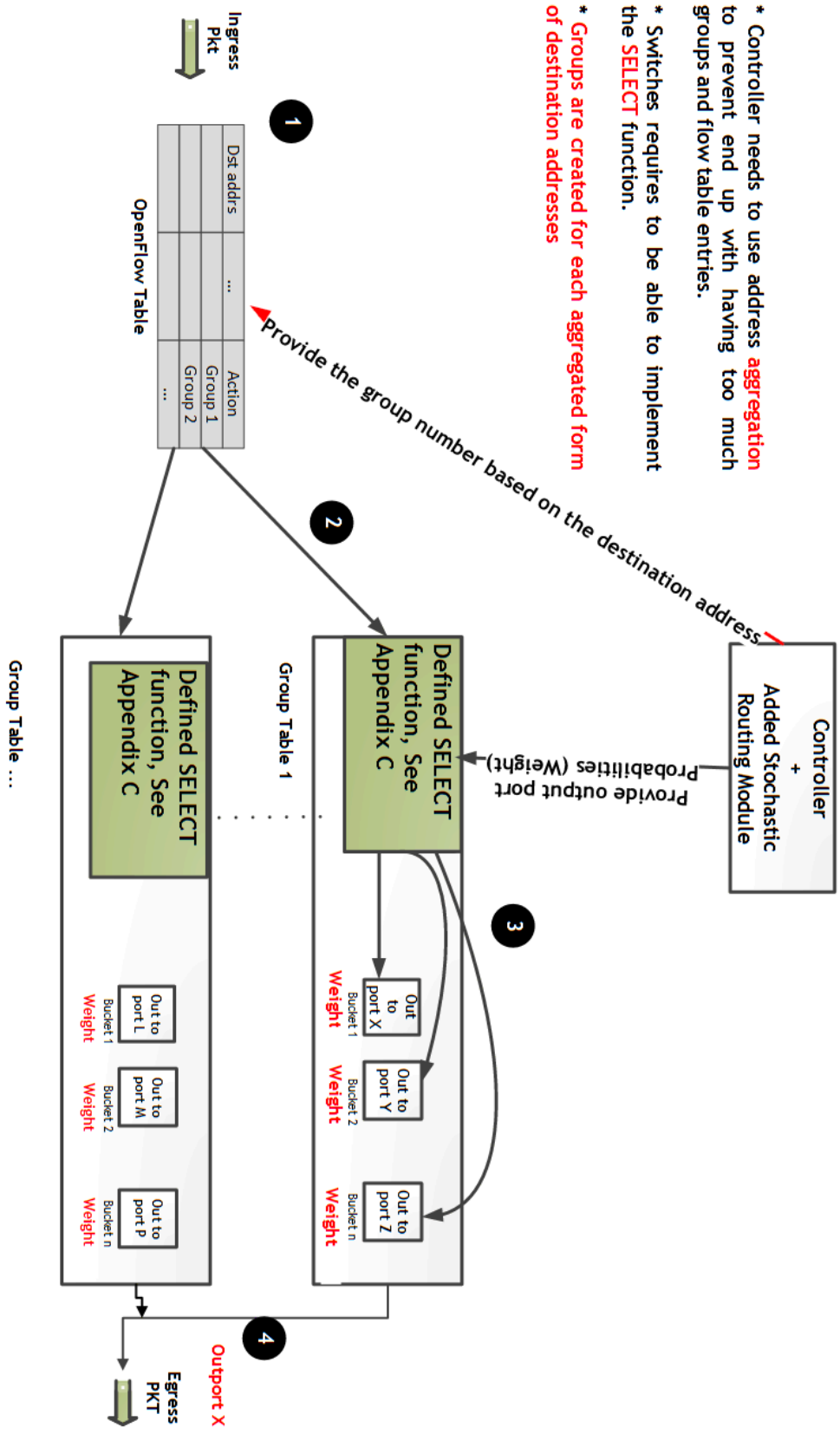

Figure 3-6 Overview of final design

Design requires:

- Controller to create a group for each aggregated set of addresses
- The stochastic routing module added in OF controller will provide the probabilities for all possible out ports toward the destination of that packet or flow
- The SELECT algorithm generate random number and according to the provided probabilities decides which output port is chosen for the packet

Advantages and Limitations:

✓ Advantages:

- Remove RTT packet delay between switch and controller which has been introduced in the first scenario
- Remove the risk of the link between switch and controller to become a potential bottleneck
- No changes in OpenFlow specifications are required

✓ Limitations:

- Number of groups should be less than or equal to the maximum number of groups that a switch supports

3.1.1.5.1 Further considerations for final design

Further studies seems to be required to answer the following considerations:

- An approach to deal with link or port failure need to be introduced. In other word, the approach should answer these questions: What happens to the weights (egress port probability) of other ports when a link or port failure occurs.
- Scalability of this design needs to be examined and analyzed in a network with reasonably large topology. Average number of required groups in a large operational network should be studied to investigate if the design is able to support stochastic switching in larger topologies.

Chapter 4: Implementation and Test

In this chapter test scenarios and implementation tools are introduced and described. The experiment starts by a pretest examining the functionality of the implemented SELECT function with one switch followed by the main test scenario consists of five OF13SoftSwitch and four end nodes. results are discussed in chapter 5. Source code of defined SELECT function is avalable in appendix C.

4.1 What is the aim of the test?

The aim of this test is to examine the virtual switch (OF13SoftSwitch) operation after changing the build in SELECT function which supported weighted round robin load sharing to the defined SELECT function which support stochastic switching.

4.2 What will be looking at?

To examine if our implemented function works we look at the following:

- For main test: Switch port counter, to explore if switch is forwarding packets to output ports according to the specified output port probabilities and as expected.
- For pre-test: Result of ping test lost packets.

4.3 Test Environment

4.3.1 Mininet

"*Mininet provides an easy way to get correct system behavior and experiment with topologies. The code you develop and test on Mininet, for an OpenFlow controller, modified switch, or host, can move to a real system with no changes, for real-world testing, performance evaluation, and deployment*" (Mininet).

Mininet (Mininet) (Lantz et al., 2010):

- provides a simple and inexpensive network testbed for developing OpenFlow applications
- enables multiple concurrent developers to work independently on the same topology

- supports system-level regression tests, which are repeatable and easily packaged
- enables complex topology testing, without the need to wire up a physical network
- includes a CLI that is topology-aware and OpenFlow-aware, for debugging or running network-wide tests
- supports arbitrary custom topologies, and includes a basic set of parameterized topologies
- is usable out of the box without programming
- also provides a straightforward and extensible Python API for network creation and experimentation

4.3.2 OF13SoftSwitch

OF13SofSwitch is a user-space compatible software switch developed at CPqD. The implementation is based on Ericsson TrafficLab 1.1 SoftSwitch which supports OF 1.1. Changes have been made to the control plane of Ericsson SoftSwitch to make OFSoftSwitch13 able to support OF 1.3.

Why OF13SoftSwitch was chosen?

At the time of working on this thesis the only shipped OpenFlow 1.1 supported physical switch we have known is NoviFlow. UNINETT made contact with the company to buy the switch. Unfortunately we did not get any response back from the company, so we decided to continue our experiment with an OF 1.1 supported virtual switch.

OF13SS supports what we need and also weighted round robin is implemented as a load sharing algorithm using SELECT function of Group feature. This implementation made us able to learn Group feature functionality better.

4.3.3 DPCTL

Dpctl is a management tool that enables control over OpenFlow switch. Using Dpctl it is possible to configure switch i.e. add, remove or modify flow entries of a flow table and also to query switch features and status such as port counter etc. List of Dpctl commands and features is accessible in CPqD Github page.

- ✓ **Note**: Other way to configure the switch instead of using DPTCL is to implement a static flow pusher module to insert the flow rules into the flow table of a switch (At the time of writing this report RYU controller developed by NTT laboratories OSRG group is an officially claimed SDN controller that supports OpenFlow 1.3. RYU uses REST API for this purpose).

4.4 Test Scenario

Test section starts by a pretest followed by the main test scenario.

4.4.1 Pre-test 1

In order to test functionality of the implemented SELECT functionality, a simple test was done. This test scenario consists of an OpenFlow enabled software switch (OF13SoftSwitch) which is connected to four end nodes.

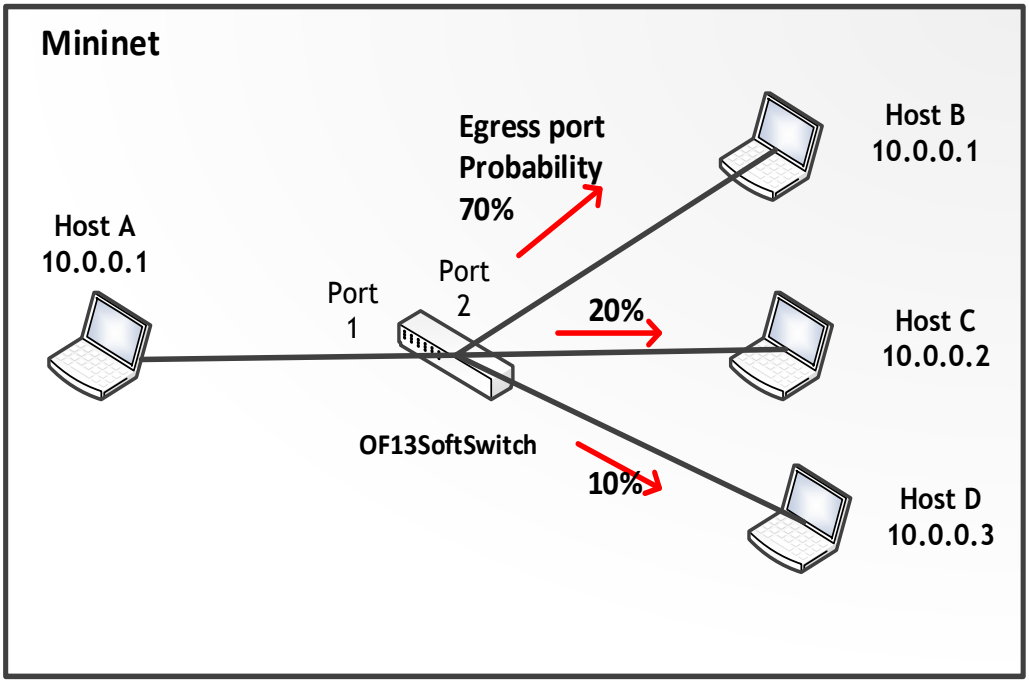

Figure 4-1 Pre-test 1

Test setup:

The switch and defined group configurations are presented at tables 4.1 and 4.2 respectively.

Action/Incoming port number	Port 1	Port 2	Port 3	Port 4
Forward to	Group 1	Port 1	Port 1	Port 1

Table 4.1 : Switch configuration

Group / Port number	**Port 2**	**Port 3**	**Port 4**
Group 1	Forward packets to port 1 with probability equal to 70%	Forward packets to port 2 with probability equal to 20%	Forward packets to port 3 with probability equal to 10%

Table 4.2 Group 1 configuration

Test description:

In each run of the simulation, host A sends 20 ping packets to one of the hosts B, C and D. Switch is configured to forward incoming packets from port 1 to one of ports 2, 3 and 4 with probabilities of 70%, 20% and 10% i.e. we roughly expect 70% of incoming packets from port 1 to be directed to port 2, 20% to port 3 and 10% of packets to port 4. Considering stochastic nature of this forwarding approach, mentioned numbers are not guaranteed and it is expected in long run that number of routed packets to each port to be close to the probabilities. Incoming packets from port 2, 3 and 4 are configured to be directed to port 1 to exit the switch.

To describe the test scenario we consider one of simulation runs. For instance host A starts sending 30 ping packets to host D. Running the simulation for ten times, we expect in average 70% of ping packets can successfully reach host B i.e. in average almost 30% of ping packets in each run are lost.

4.4.2 Pre-test 2

Host A sends ping packets to Hosts B, C and D. Output probabilities (bucket weights) for ports 2, 3 and 4 are 50%, 20% and 30% respectively. Procedure is the same as pre-test 1.

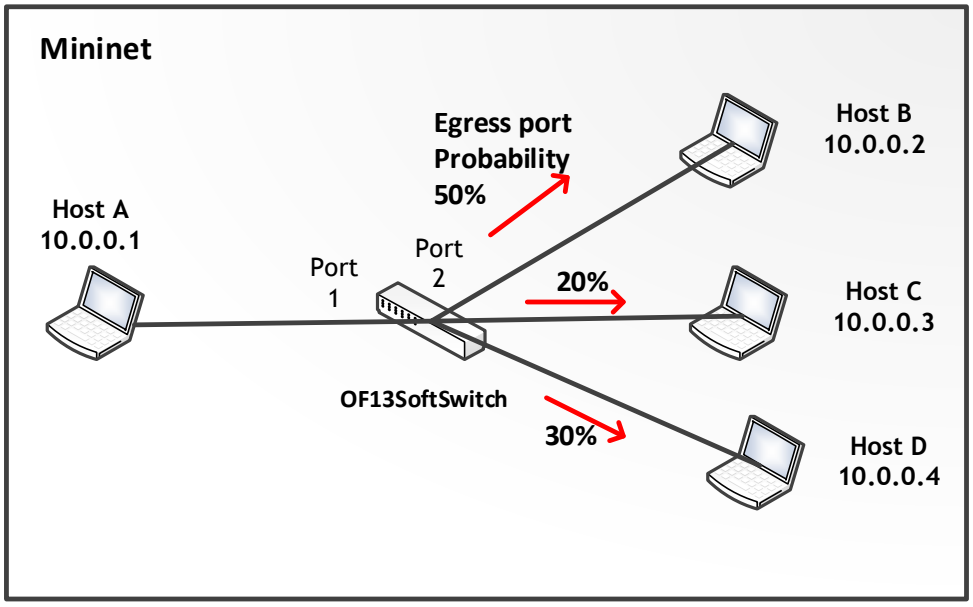

Figure 4-2 Pre-test 2 scenario

Group / Port number	Port 2	Port 3	Port 4
Group 1	Forward packets to port 1 with probability equal to 50%	Forward packets to port 2 with probability equal to 20%	Forward packets to port 3 with probability equal to 30%

Table 4.3 Group table definition

4.4.3 Main test scenario

A simple network scenario presented in figure 4 was considered to test the method in a more realistic network.

Test Setup:

IP addresses of nodes are shown in table 4.4. Mininet python API was used to create the topology. Python code is available in appendix D. Data-paths 0 to 4 of OF13SoftSwitch represent 5 OpenFlow enabled switches used in this scenario. OF13SoftSwitch enables to work with several data-paths at the same time without interfering with each other. Each of these data-paths listens to a unique port in Mininet. List of open port are shown in figure 4.3.

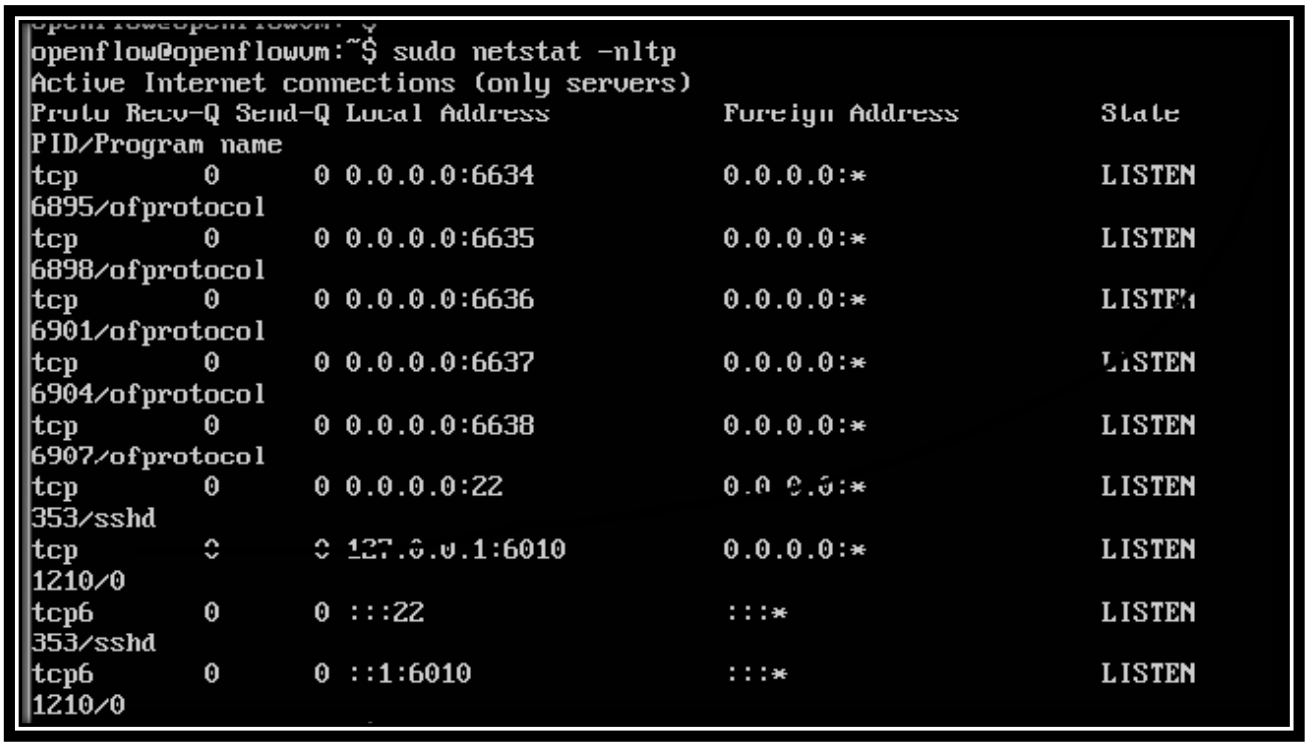

Figure 4-3 List of open ports that each data-path listens to.

As mentioned earlier, Mininet python API was used to create the topology. When executing the python script, Mininet creates the topology. Figure 4.4 is a screenshot of executed python script.

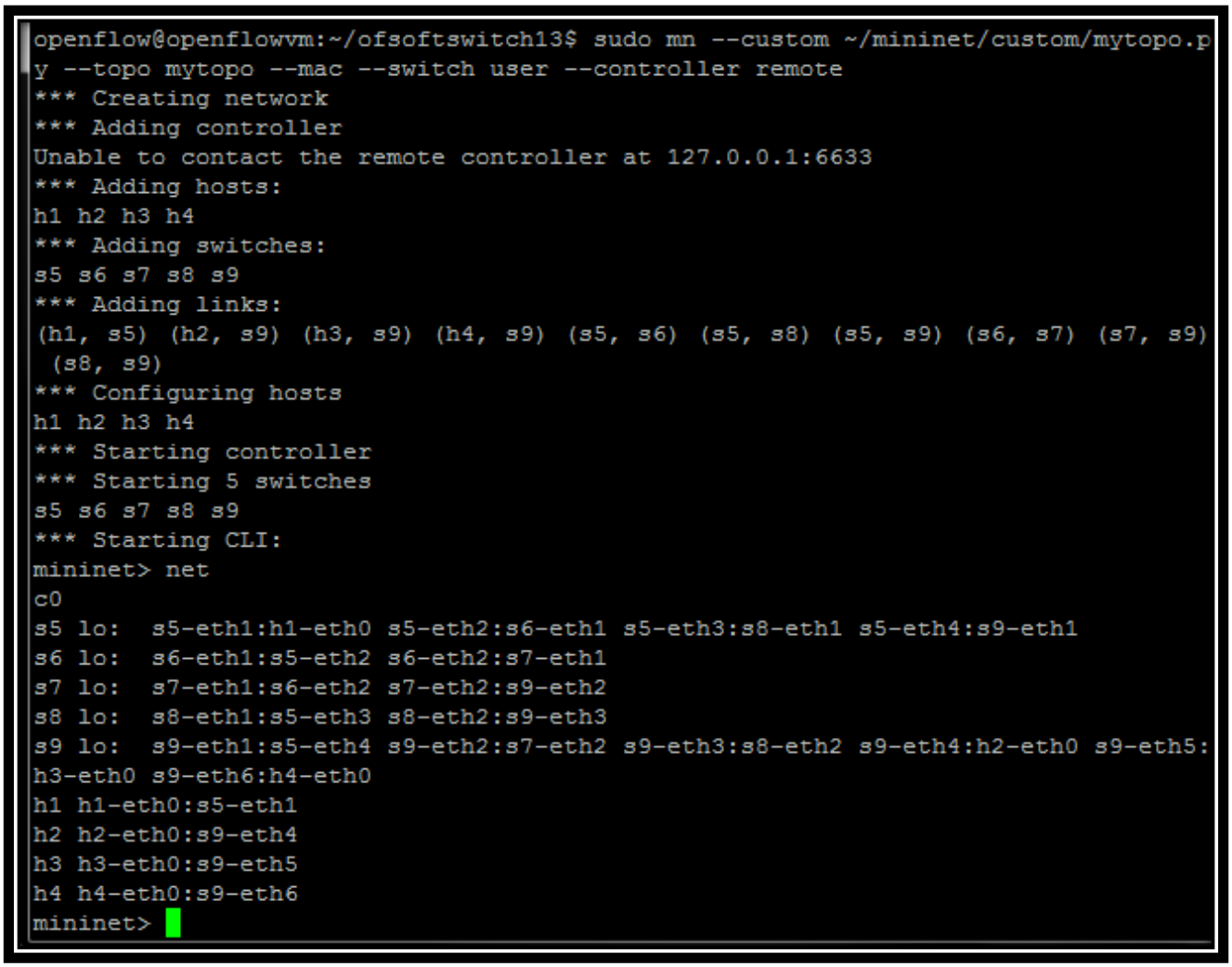

Figure 4-5 When executing python script, Mininet creates the topology

Device	IP Address	Operating System
Host A	10.0.0.1	Ubuntu 11.04
Host B	10.0.0.2	Ubuntu 11.04
Host C	10.0.0.3	Ubuntu 11.04
Host D	10.0.0.4	Ubuntu 11.04

Table 4.4 Hosts information

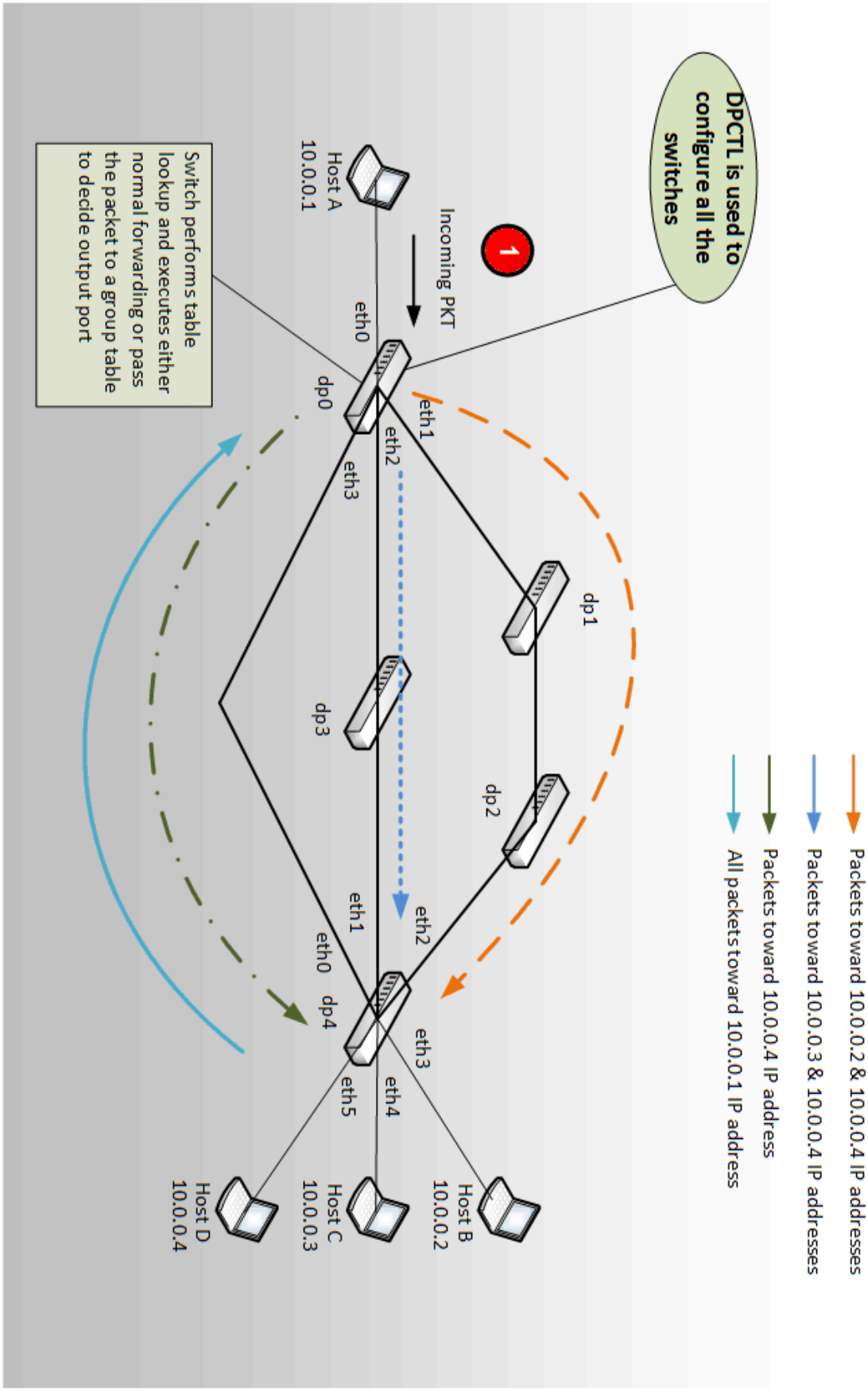

Figure 4-6 Main test topology

Configuration of switch 1 (dp0) and 5 (dp4) are shown at tables 4.5 and 4.6. Switch 2, 3 and 4 are configured to receive packets from port 1 and forward it to port 2. Configurations of all switches are available in appendix B.

Incoming port/IP Addresses	10.0.0.1	10.0.0.2	10.0.0.3	10.0.0.4
Port 1	_	Port 2	Port 3	Group 1
Port 2	_	_	_	_
Port 3	_	_	_	_
Port 4	Port 1	Port 1	Port 1	Port 1

Table 4.5 Switch 1 (dp0) configurations

Incoming port/ IP Addresses	10.0.0.1	10.0.0.2	10.0.0.3	10.0.0.4
Port 1	_	_	_	Forward to port 6
Port 2	_	_	Forward to port 5	Forward to port 6
Port 3	_	Forward to port 6	_	Forward to port 6
Port 4	Forward to port 1	_	_	_
Port 5	Forward to port 1	_	_	_
Port 6	Forward to port 1	_	_	_

Table 4.6 Switch 5 (dp4) configurations

Group /Port number	Port 2/eth1	Port 3/eth2	Port 4/eth3
Group 1	Forward packets to port 2 with probability equal to 20%	Forward packets to port 3 with probability equal to 10%	Forward packets to port 4 with probability equal to 70%

Table 4.7 Group table configuration

Procedure:

Host A, starts to send packet to host B, C and. Packets destined at host B are configured to pass switch 2 (dp1) and switch 3 (dp2), packets destined at host C pass switch 4 and packets toward host D will take one of three paths according to the group table configuration shown in table 4.7. After running the test and ping packet successfully received at host A, status of the switches was cleared to run the test again. In next run, host A, send ping packet to host D and a query to capture port statistics of switch 1 (dp0) was made. Results of pre-test and main test are described in the next chapter.

Chapter 5: Test Results

In this chapter, results of pre-test and main test are shown and described.

5.1 Pre-test results:

Figure 5.1 shows the result of pre-test 1. Switch was configured to forward 70% of packets to port 2. As an example, it is expected when Host A sends ping packets to host B, number of packets which receive at host B fluctuate around 70% percent of total ping packets. In other word, ping lost packet percentage fluctuate around 30%.

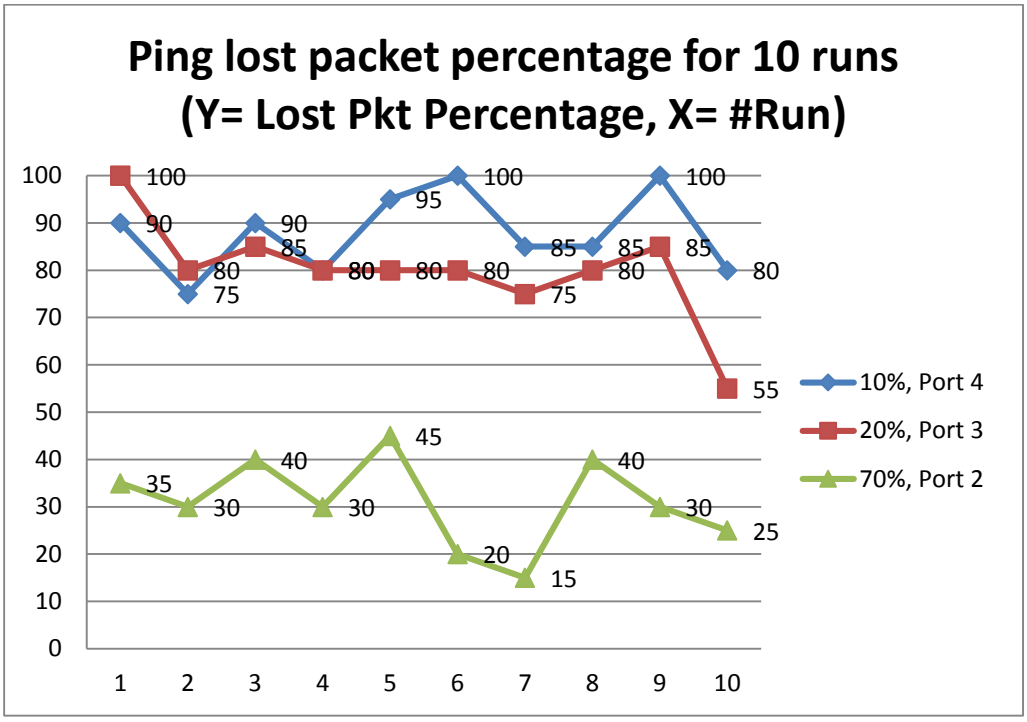

Figure 5-1 Pre-test 1 ping lost packets percentage

Group / Port number	Port 2	Port 3	Port 4
Group 1	Forward packets to port 1 with probability equal to 70%	Forward packets to port 2 with probability equal to 20%	Forward packets to port 3 with probability equal to 10%

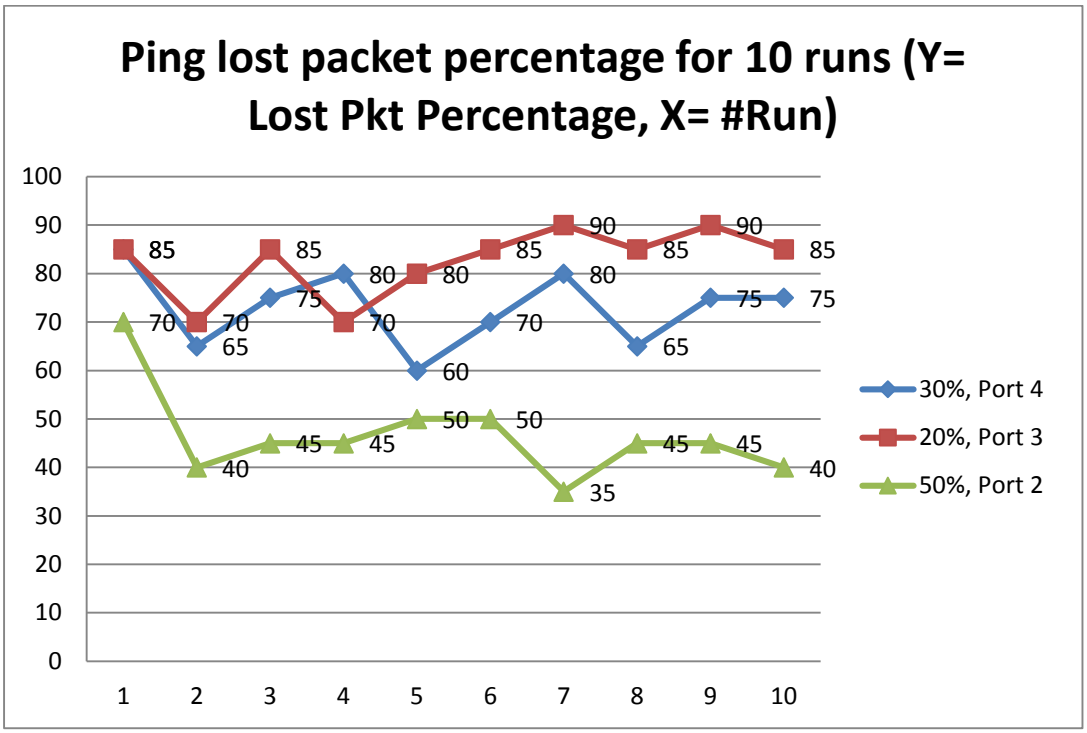

Figure 5-2 Pre-test 2 ping lost packets percentage

Group / Port number	Port 2	Port 3	Port 4
Group 1	Forward packets to port 1 with probability equal to 50%	Forward packets to port 2 with probability equal to 20%	Forward packets to port 3 with probability equal to 30%

Considering the stochastic behavior of the switch, the tests have shown that the average number of packets which reach their destinations are roughly close to what value of probabilities imply.

5.2 Main test results

Host A starts sending 40 ping packets to host D. Packets destined at host D are configured at switch 1 (dp0) to be forwarded stochastically through three available paths with egress port probabilities as presented in table 4.7. Figure 5.3 is a screenshot of host A pinging host B.

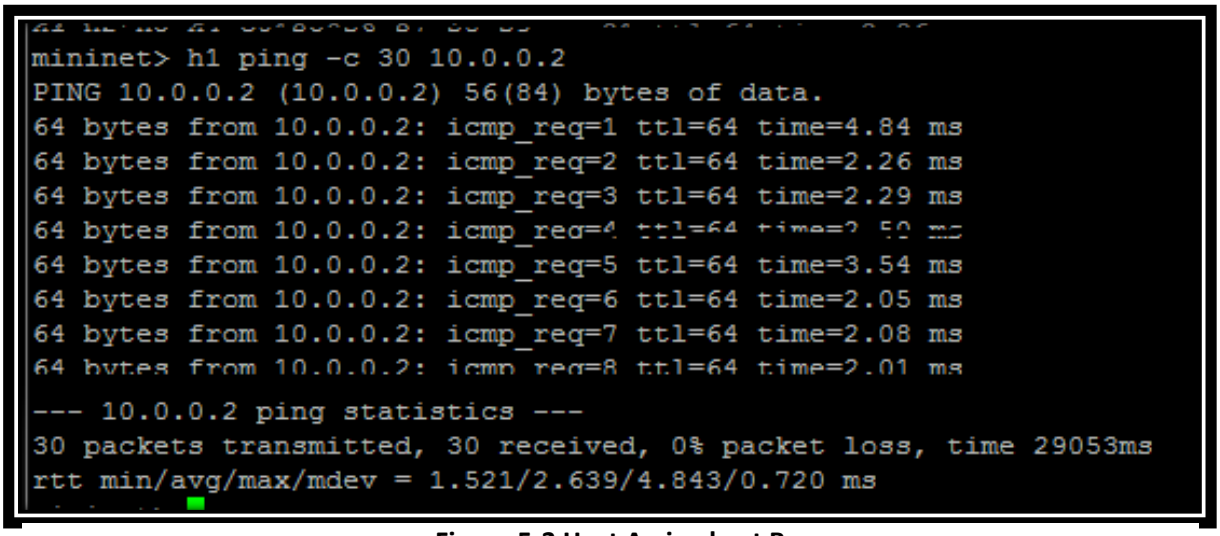

Figure 5-3 Host A ping host B

Table 5.1 represents transmitted packets by port 2, 3 and 4 of switch 1 (dp0).

Port number/Transmitted PKTs	First ping run	Second ping run	Third ping run
Port 2 (eth1)	**4**	**10**	**14**
Port 3 (eth2)	**11**	**20**	**25**
Port 4 (eth3)	**32**	**60**	**93**

Table 5.1 Transmitted packets on switch one ports

Considering the number of ping packets sent in each ping run (40 PKTs) number of transmitted packets by each port of switch 1 is close to the expected numbers according to the probabilities assigned to each port. The mismatch between number of ping packets forwarded by the switch and the port counter is mainly due to handling ARP packets.

In the box below, command used to query switch port counters and the responses are shown.

openflow@openflowvm:~/ofsoftswitch13$ sudo dpctl tcp:127.0.0.1:6634 stats-port 2

SENDING:

stat_req{type="port", flags="0x0", port="2"}

RECEIVED:

stat_repl{type="port", flags="0x0", stats=[{port="2", rx_pkt="3", tx_pkt="4", rx_byt es="230", tx_bytes="392", rx_drops="0", tx_drops="0", rx_errs="0", tx_errs="0", rx_f rm="0", rx_over="0", rx_crc="0", coll="0"}]}

openflow@openflowvm:~/ofsoftswitch13$ sudo dpctl tcp:127.0.0.1:6634 stats-port 3

SENDING:

stat_req{type="port", flags="0x0", port="3"}

RECEIVED:

stat_repl{type="port", flags="0x0", stats=[{port="3", rx_pkt="3", tx_pkt="11", rx_by tes="230", tx_bytes="1040", rx_drops="0", tx_drops="0", rx_errs="0", tx_errs="0", rx _frm="0", rx_over="0", rx_crc="0", coll="0"}]}

openflow@openflowvm:~/ofsoftswitch13$ sudo dpctl tcp:127.0.0.1:6634 stats-port 4

SENDING:

stat_req{type="port", flags="0x0", port="4"}

RECEIVED:

stat_repl{type="port", flags="0x0", stats=[{port="4", rx_pkt="50", tx_pkt="32", rx_b ytes="4722", tx_bytes="3060", rx_drops="0", tx_drops="0", rx_errs="0", tx_errs="0", rx_frm="0", rx_over="0", rx_crc="0", coll="0"}]}

5.3 Discussion

Results of the tests have shown that the implemented algorithm works properly as was expected.

SELECT method has been introduced as a way to support performing load sharing with more flexibility but there is not any defined load sharing algorithm in the OpenFlow specification. So, it provides the flexibility to define the selection methods according to the needs. It might be useful to define a range of most used load sharing algorithms such as round robin, weighted round robin, etc. as optional built in policies that one can choose among them.

Group feature makes it possible to have different type of load balancing algorithms for different flows at the same time e.g. it is possible to have stochastic forwarding for flow A and round robin for flow B or having stochastic forwarding behavior for a flow in one forwarding box in a network and different type of load sharing for the same flow in other forwarding boxes.

Chapter 6: Conclusion

In This thesis feasibility of performing stochastic switching using OpenFlow in a SDN environment was investigated. Different possible scenarios have been studied through working on this thesis.

Referring to the research question:

- Is it possible to perform stochastic switching using OpenFlow in a software defined network?

This study revealed that by defining proper function it is feasible to implement stochastic switching using SELECT method of OpenFlow group feature that has been introduced in OpenFlow 1.1 specifications. Test results have shown that implemented SELECT function performs stochastic switching properly.

It seems by defining desired SELECT method and developing route balancing applications on top of SDN controllers to provide required information to the load balancing methods implemented in a switch, different load sharing approaches could be implemented in a network.

Further studies regarding the proposed solution in larger topologies seems required to examine scalability aspect of stochastic routing using proposed method and Also, to answer the following question:

- What happens in case of link or port failure? How to deal with these cases?

Although the answer to this question might not be difficult but prototyping, implementation and evaluation of the answer to the question could be an extension to this thesis.

6.1 Future work

Considering that stochastic switching is implementable in an OpenFlow enabled virtual switch or in OpenFlow agent of a hardware switch, development of stochastic routing application on top of an OpenFlow controller is an open research topic.

The author have been granted a summer internship to work on HP OpenFlow 1.3 beta switch firmware testing on HP Procurve switches at UNINETT AS in Trondheim during summer 2013. The following tasks will be pursued:

- Test the functionality and performance of stochastic switching in an operational environment (A scenario with OpenFlow 1.3 enabled virtual switch and HP OpenFlow 1.3 enabled hardware switch is a topic of interest)
- Test HP OF 1.3 beta switch firmware features

Bibliography

HANDIGOL, N., SEETHARAMAN, S., FLAJSLIK, M., MCKEOWN, N. & JOHARI, R. Plug-n-Serve: Load-balancing web traffic using OpenFlow, Aug 2009. *Demo at ACM SIGCOMM*.

ICHINO, K. 2011. OpenFlow COMMUNICATION SYSTEM AND OpenFlow COMMUNICATION METHOD. Google Patents.

KOERNER, M. & ODEJ, K. Multiple service load-balancing with OpenFlow. High Performance Switching and Routing (HPSR), 2012 IEEE 13th International Conference on, 24-27 June 2012 2012. 210-214.

LAJOS KIS, Z. 2011. *OpenFlow 1.1 SoftSwitch* [Online]. Available: https://github.com/TrafficLab/of11softswitch.

LANTZ, B., HELLER, B. & MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, 2010. ACM, 19.

MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S. & TURNER, J. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review,* 38**,** 69-74.

MININET. *Mininet* [Online]. Available: www.mininet.org.

NEC. 2010. OpenFlow Feature Guide (Version 11.1 Compatible)

NEC. 2011. NEC ProgrammableFlow: Redefining Cloud Network Virtualization with OpenFlow.

NICIRA 2012. it's time to virtualize the network.

ONF 2012. Software-Defined Networking: The New Norm for Networks.

OPENDAYLIGHT-WIKI. 2013. *OpenDaylight SDN Controller Platform (OSCP)* [Online]. Available: https://wiki.opendaylight.org/view/OpenDaylight_SDN_Controller_Platform_%28OSCP%29:Overview#Reactive_and_Proactive_Flow_Setup.

PFAFF, B. 2012. Openflow switch specification v1. 3.0.

SHARAFAT, A. R., DAS, S., PARULKAR, G. & MCKEOWN, N. Mpls-te and mpls vpns with openflow. ACM SIGCOMM Computer Communication Review, 2011. ACM, 452-453.

SHENKER, S. 2011. The future of networking, and the past of protocols. *Open Networking Summit*.

SHERWOOD, R., GIBB, G., YAP, K.-K., APPENZELLER, G., CASADO, M., MCKEOWN, N. & PARULKAR, G. 2009. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*.

SOEURT, J. & HOOGENDOORN, I. 2012. Shortest path forwarding using OpenFlow.

UPPAL, H. & BRANDON, D. 2010. OpenFlow Based Load Balancing. *CSE561: Networking Project Report, University of Washington*.

WANG, R., BUTNARIU, D. & REXFORD, J. 2011. OpenFlow-based server load balancing gone wild. *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services.* Boston, MA: USENIX Association.

RYU SDN Controller [Online]. Available: http://osrg.github.io/ryu/.

Appendix A: Experiment Required Tools

OS Type	OS Version	Virtualization Software	X Server	Terminal
Windows	7	Oracle Virtual Box	Xming	PuTTY

Appendix B: Configurations

Data-paths 0 to 5 represents switches 1 to 6. DPCTL management tool was used to configure each data path.

Dp0/Switch 1 configurations:

Handle ARP messages:

sudo dpctl tcp:127.0.0.1:6634 flow-mod table=0,cmd=add eth_type=0x806,arp_tpa=10.0.0.1 apply:output=1

sudo dpctl tcp:127.0.0.1:6634 flow-mod table=0,cmd=add eth_type=0x806,arp_tpa=10.0.0.2 apply:output=2

sudo dpctl tcp:127.0.0.1:6634 flow-mod table=0,cmd=add eth_type=0x806,arp_tpa=10.0.0.3 apply:output=4

sudo dpctl tcp:127.0.0.1:6634 flow-mod table=0,cmd=add eth_type=0x806,arp_tpa=10.0.0.4 apply:group=1

Handle ICMP messages:

sudo dpctl tcp:127.0.0.1:6634 flow-mod table=0,cmd=add eth_type=0x800,ip_dst=10.0.0.2 apply:output=2

sudo dpctl tcp:127.0.0.1:6634 flow-mod table=0,cmd=add eth_type=0x800,ip_dst=10.0.0.3 apply:output=4

sudo dpctl tcp:127.0.0.1:6634 flow-mod cmd=add,table=0 eth_type=0x800,ip_dst=10.0.0.4 apply:group=1

Direct incoming packets of all type which arrive at port 4 to port 1:

sudo dpctl tcp:127.0.0.1:6634 flow-mod table=0,cmd=add in_port=4 apply:output=1

Group table creation and configuration:

sudo dpctl tcp:127.0.0.1:6634 group-mod cmd=add,type=1,group=1 weight=7,port=1,group=all output=4 weight=2,port=1,group=all output=3 weight=1,port=1,group=all output=2

Dp1/Switch 2 configurations:

Direct incoming packets of all type which arrive at port 1 to port 2:

sudo dpctl tcp:127.0.0.1:6635 flow-mod table=0,cmd=add in_port=1 apply:output=2

Dp2/Switch 3 configurations:

Direct incoming packets of all type which arrive at port 1 to port 2:

sudo dpctl tcp:127.0.0.1:6636 flow-mod table=0,cmd=add in_port=1 apply:output=2

Dp3/Switch 4 configurations:

Direct incoming packets of all type which arrive at port 1 to port 2:

sudo dpctl tcp:127.0.0.1:6637 flow-mod table=0,cmd=add in_port=1 apply:output=2

Dp4/Switch5 configurations

Handle ARP messages:

sudo dpctl tcp:127.0.0.1:6638 flow-mod table=0,cmd=add eth_type=0x806,arp_tpa=10.0.0.1 apply:output=1

sudo dpctl tcp:127.0.0.1:6638 flow-mod table=0,cmd=add eth_type=0x806,arp_tpa=10.0.0.2 apply:output=4

sudo dpctl tcp:127.0.0.1:6638 flow-mod table=0,cmd=add eth_type=0x806,arp_tpa=10.0.0.3 apply:output=5

sudo dpctl tcp:127.0.0.1:6638 flow-mod table=0,cmd=add eth_type=0x806,arp_tpa=10.0.0.4 apply:output=6

Handle ICMP messages:

sudo dpctl tcp:127.0.0.1:6638 flow-mod table=0,cmd=add eth_type=0x800,ip_dst=10.0.0.1 apply:output=1

sudo dpctl tcp:127.0.0.1:6638 flow-mod table=0,cmd=add eth_type=0x800,ip_dst=10.0.0.2 apply:output=4

sudo dpctl tcp:127.0.0.1:6638 flow-mod cmd=add,table=0 eth_type=0x800,ip_dst=10.0.0.3 apply:output=5

sudo dpctl tcp:127.0.0.1:6638 flow-mod cmd=add,table=0 eth_type=0x800,ip_dst=10.0.0.4 apply:output=6

Appendix C: Defined SELECT Function in C

```
#include <stdlib.h>
#include <time.h>
static size_t
select_from_select_group(struct group_entry *entry) {
    struct group_entry_wrr_data *data;
    //struct group_entry *data;
    size_t ran_num;
    size_t i;

    srand(time(NULL));
    ran_num = rand()%10;

    entry->data = xmalloc(sizeof(struct group_entry_wrr_data));
    data = (struct group_entry_wrr_data *)entry->data;

    data->curr_weight = ran_num + 1;

    if (entry->desc->buckets_num == 0) {
        return -1;
    }

    if ( data->curr_weight <=  entry->desc->buckets[0]->weight )
        return entry->desc->buckets[0]->weight;

    for (i=1; i< entry->desc->buckets_num; i++) {

        if ( entry->desc->buckets[i-1]->weight < data->curr_weight && data-
>curr_weight <= (entry->desc->buckets[i]->weight + entry->desc->buckets[i-
1]->weight)  )
            return entry->desc->buckets[i]->weight
    }

    VLOG_WARN_RL(LOG_MODULE, &rl, "Could not select from select group.");
    return -1;
    }
```

Appendix D: Python Code to Create Custom Topology in Mininet

This Python script creates the main test topology in Mininet.

```
from mininet.topo import Topo
class MyTopo( Topo ):
   "Topology."
 def __init__( self ):
       # Initialize topology
        Topo.__init__( self )

       # Add hosts and switches
        leftHostA = self.addHost( 'h1' )
        rightHostB = self.addHost( 'h2' )
        rightHostC = self.addHost( 'h3' )
        rightHostD = self.addHost( 'h4' )
        leftSwitch = self.addSwitch( 's5' )
        upLeftSwitch = self.addSwitch( 's6' )
        upRightSwitch = self.addSwitch( 's7' )
        midleSwitch = self.addSwitch( 's8' )
        rightSwitch = self.addSwitch( 's9' )

        # Add links
        self.addLink( leftHostA, leftSwitch )
        self.addLink( leftSwitch, upLeftSwitch )
        self.addLink( leftSwitch, midleSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( upLeftSwitch, upRightSwitch )
        self.addLink( upRightSwitch, rightSwitch )

        self.addLink( midleSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHostB )
        self.addLink( rightSwitch, rightHostC )
        self.addLink( rightSwitch, rightHostD )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```