

STUDY OF P4 LANGUAGE

A Project Report

submitted by

ASHISH GONDIMALLA

*in partial fulfilment of the requirements
for the award of the degree of*

BACHELOR OF TECHNOLOGY



**DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.**

June 2016

THESIS CERTIFICATE

This is to certify that the thesis entitled **STUDY OF P4 LANGUAGE**, submitted by **Ashish Gondimalla**, to the Indian Institute of Technology Madras, for the award of the degree of **Bachelor of Technology**, is a bonafide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. KAMAKOTI V
Research Guide
Professor
Dept. of Computer Science and Engineering
IIT Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

I would like to thank Dr. V. Kamakoti without whose motivation this work would not have been possible. I would also like to give special thanks Dr. Vasanth and Dr. Shankar for suggesting and guiding through the tough times. The calm, encouraging support by the personnel at Netlab was really helpful to complete my tasks.

ABSTRACT

KEYWORDS: SDN, P4, Network Protocols, Docker, Quagga, FPGA platforms

In networking, many complex functions and protocols are already imprinted in the hardware switches. This makes the network operators lose the flexibility to control the software and also simulate newer methods in the networks. Networks are trying to get faster but not better. To tackle this problem software defined networks started to get deployed which separates control plane from the data flow plane. Openflow is one of the most widely used SDN control protocol. P4 Language tries to propose a different approach to configure switches.

This report aims to show how P4 programming language can be used to program switches. A few examples have been implemented on software switches along with a quagga supported P4 switch. Ongoing research to extend them onto hardware platforms are discussed at the end.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	i
ABSTRACT	ii
LIST OF TABLES	v
LIST OF FIGURES	vi
ABBREVIATIONS	vii
1 INTRODUCTION	1
2 P4 PROGRAMMING LANGUAGE	2
2.1 P4 Abstraction Model	2
2.2 The P4 programming Language	2
2.2.1 Headers	4
2.2.2 Parser	4
2.2.3 Tables	6
2.2.4 Actions	8
2.3 Comparison between P4 and Openflow	10
3 SAMPLE PROGRAMS	12
3.1 EasyRoute protocol	12
3.2 Flowlet Switching	14
4 Quagga application on a P4 based switch	18
4.0.1 switchapi	19
4.0.2 switchsai	19
4.0.3 switchlink	19

5	Current Hardware Implementations	23
5.1	FPGA	23
5.1.1	P4FPGA	23
5.1.2	Xilinx SDNet via PX	24
5.2	Network Processor	24
A	P4 installation	26
B	EasyRoute protocol code and results	28
C	Flowlet switching	37
D	Building docker image	51

LIST OF TABLES

A.1	Branch and commit id of the cloned modules (at time of downloading)	26
-----	---	----

LIST OF FIGURES

2.1	The abstraction model	3
2.2	State machine diagram for the parser	7
2.3	P4 vs Openflow	11
3.1	Topology used in mininet	13
3.2	Topology used for testing	15
3.3	Without Modification of P4 code	16
3.4	With Modification of P4 code	17
4.1	Architecture of the libraries	20
4.2	Topology with IP addresses	21
4.3	Host h1 packets in wireshark	21
4.4	Host h2 packets in wireshark	22
B.1	Easy Protocol testing in mininet	36

ABBREVIATIONS

API	Application Programming interface
ECMP	Equal-Cost Multi Path
FPGA	Field Programmable Gate Arrays
JSON	JavaScript Object Notation
NFP	Network Flow Processor
NVGRE	Network Virtualization using Generic Routing Encapsulation
P4	Programming Protocol independent Packet Processors
TCP	Transmission Control Protocol
TDG	Table Dependency Graph
SAI	Switch Abstraction Interface
SDN	Software Defined Networks
VXLAN	Virtual Extensible Local Area Network

CHAPTER 1

INTRODUCTION

Software Defined Networking is the key to achieve programmatic control over the network switches. It tries to refactor the relationship between network devices and the software that controls them. [Paraphrased from the HotSDN '12 Solicitation]. Openflow is the main forwarding table management protocol currently used. The background theory for SDN can be found in the tutorial by Heller (2012).

As newer packet encapsulation methods are getting applied (Ex: NVGRE, VXLAN) more header fields are created. However, Openflow explicitly specifies the headers on which it operates. This increases the complexity of programming along with not providing the flexibility to add new headers. Also Openflow needs to extend its specifications repeatedly to support the new protocols. For example, the header set has grown from 12 to 41 headers in about 5 years. This brings the need to develop a new approach to allow flexible mechanisms to parse packets, introduce matching header fields. The approach should try to tell the switch how to operate rather than getting constrained by a fixed switch design.

P4 (Programming Protocol independent Packet Processors) tries to balance this the need for expressiveness with the ease of implementation across a wide range of hardware and software switches. It targets three main goals:

1. Reconfigurability
2. Protocol Independence
3. Target Independence

The subsequent chapters explain how P4 achieves the above requirements.

CHAPTER 2

P4 PROGRAMMING LANGUAGE

2.1 P4 Abstraction Model

The P4 language assumes an abstraction model of the underlying switch. According to this abstraction, the switch consists of a programmable parser followed by multiple stages of match + action tables which can be arranged in series, parallel, or a combination of both. This looks very similar to that of openflow. But there are differences like presence of programmable parser and actions that are composed from protocol-independent primitives supported by the switch. This is shown in the figure 2.1

2.2 The P4 programming Language

Based on the above abstraction model, a language should be used to express how the switch is to be configured. The model requires flexible data processing on the headers. This motivates the language to allow a programmer to declare new headers and a control flow to describe header processing. Also the order of header processing by match + action depends on the dependencies. These dependencies are captured by a Table Dependency Graph (TDG). But programmers try to write program keeping the algorithm in view rather than a graph. Thus the language must be able to represent a control flow which can be converted into a TDG by

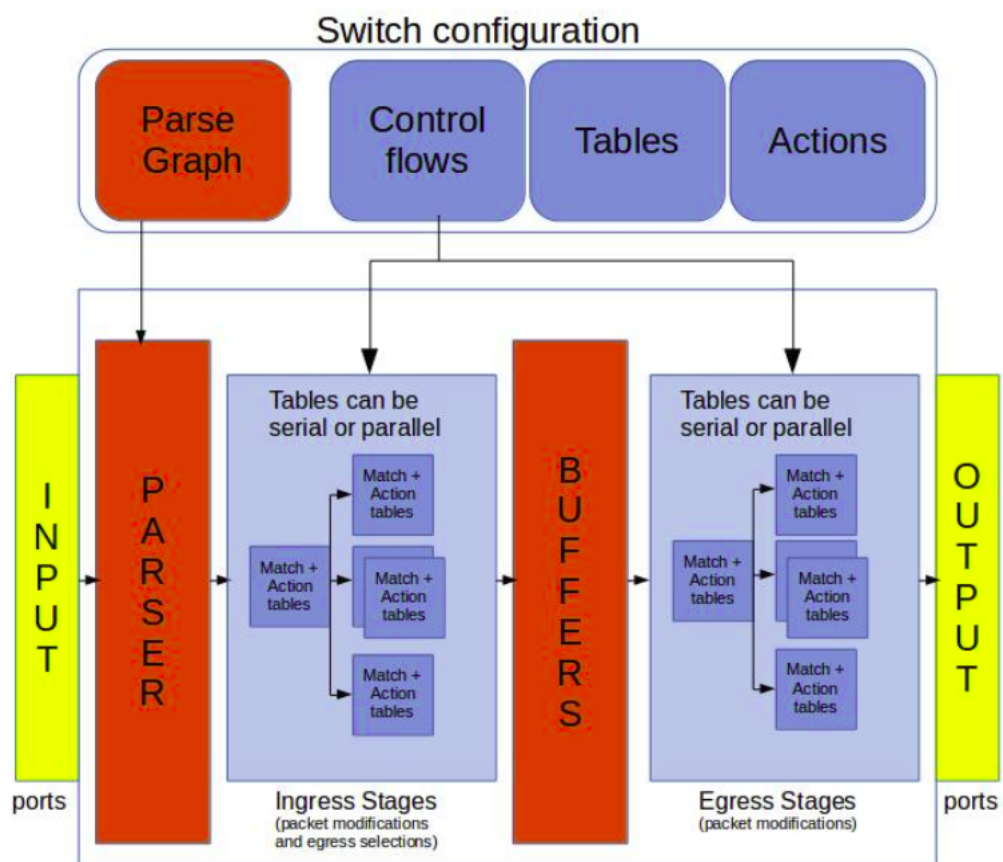


Figure 2.1: The abstraction model

the compiler. P4 achieves this by considering the following key components in its language.

2.2.1 Headers

A header definition describes the structure of various fields. It includes specifications like field widths and constraints on the field width. We can declare only the required headers or any new headers. This gives flexibility unlike OpenFlow where we are restricted to the fixed headers.

Example:

```
header ethernet {  
  fields {  
    dst_addr : 48; // width in bits  
    src_addr : 48;  
    ethertype : 16;  
  }  
}
```

2.2.2 Parser

A parser definition specifies how to identify headers and valid header sequences. Parsing in P4 has a start state and a stop state. Each parse declaration based upon the value of the header that is being parsed maps to another parser declaration or a control program (defined in later subsection). Thus the parser acts like a state machine with transitions based on header values.

Example:

```
parser start {  
    ethernet;  
}  
  
parser ethernet {  
    switch(ethertype) {  
        case 0x8100: vlan;  
        case 0x9100: vlan;  
        case 0x800: ipv4;  
        // Other cases  
    }  
}  
  
parser vlan {  
    switch(ethertype) {  
        case 0xaaaa: mTag;  
        case 0x800: ipv4;  
        // Other cases  
    }  
}  
  
parser mTag {  
    switch(ethertype) {  
        case 0x800: ipv4;  
        // Other cases  
    }  
}
```

The state machine diagram for the above parser code is as shown in figure 2.2. Based on the value *ethertype* the parser can expect a different fields during parsing unlike fixed set of headers.

2.2.3 Tables

Match + action tables are the mechanism to perform packet processing. A programmer defines what fields in a header can be operated and what actions to be used. For this there are attributes in a table like *reads* and *actions*. As the words suggest *reads* is used to know what fields the table have access to and *actions* is used to perform operation on those fields. There is also another attribute called *max_size* to describe how many entries the table should support.

Example:

```
table mTag_table {
  reads {
    ethernet.dst_addr : exact;
    vlan.vid : exact;
  }
  actions {
    // At runtime, entries are programmed with params
    // for the mTag action. See below.
    add_mTag;
  }
  max_size : 20000;
}
```

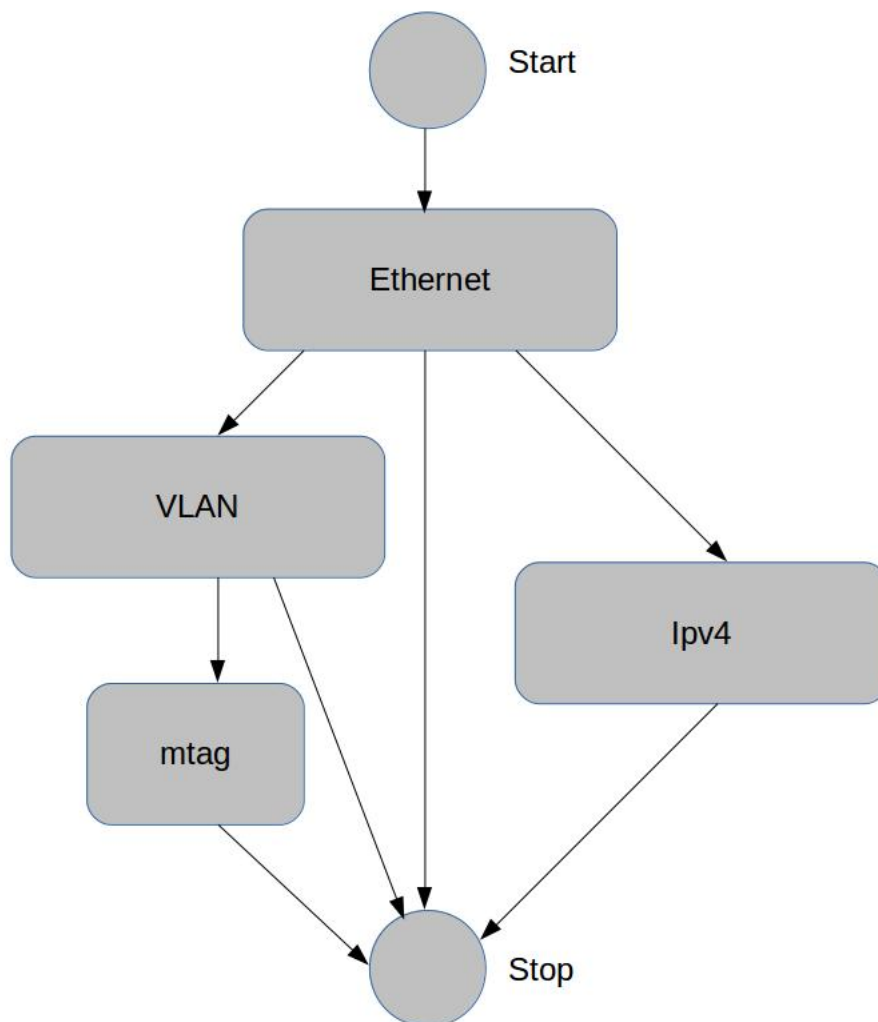


Figure 2.2: State machine diagram for the parser

Here, reads attribute tells only ethernet.dst_addr and vlan.vid are accessible by mTag_table. The action add_mTag works on the headers. Also the table can support a maximum of 20000 entries.

2.2.4 Actions

Actions are like functions in a general programming language. There are few primitive actions based on which complicated actions can be built. Some primitive actions include *set_field*, *copy_field*, *add_header*, *remove_header*, *increment*, *check_sum*. P4 assumes parallel execution of primitives within in an action function.

Example: A complex action add_mTag is made from primitive actions:

```
action add_mTag(up1, up2, down1, down2, egr_spec) {
  add_header(mTag);
  // Copy VLAN ethertype to mTag
  copy_field(mTag.ethertype, vlan.ethertype);
  // Set VLAN's ethertype to signal mTag
  set_field(vlan.ethertype, 0xaaaa);
  set_field(mTag.up1, up1);
  set_field(mTag.up2, up2);
  set_field(mTag.down1, down1);
  set_field(mTag.down2, down2);
  // Set the destination egress port as well
  set_field(metadata.egress_spec, egr_spec);
}
```

We also need to map the tables based on their dependencies. Thus a control

program is required to specify the flow from one table to another.

Example:

```
control ingress() {  
    // Verify mTag state and port are consistent  
    table(source_check);  
  
    // If no error from source_check, continue  
    if (!defined(metadata.ingress_error)) {  
        // Attempt to switch to end hosts  
        table(local_switching);  
        if (!defined(metadata.egress_spec)) {  
            // Not a known local host; try mtagging  
            table(mTag_table);  
        }  
        // Check for unknown egress state or  
        // bad retagging with mTag.  
        table(egress_check);  
    }  
}
```

ingress is the control program which says source_check should be performed first. Depending on the value of a particular field in the packet after getting processed, the next table is decided. The indepth details can be found in Consortium, 2015a

2.3 Comparison between P4 and Openflow

A rather improper but a suitable analogy to compare them is the following. Openflow is like machine dependent assembly language where P4 is at a higher level like C language. An assembly language is machine dependent and there is literally no load on the compiler. However it is extremely inflexible and varies based on the underlying architecture. This is exactly like Openflow protocol which sends queries (or instructions) to the openflow controller in the switch.

Whereas P4 is like C program which is quite independent of the platform on which it works. It becomes the compiler's job to compile the C code into required machines assembly code. Similarly for P4 the compiler takes the task of configuring the program into the switch. P4 language tries to find the sweet spot between expressiveness and implementation across different hardware and software switches. The difference is demonstrated in the figure 2.3

P4 requires a different compiler for different target switches. For this reason, the compilation of P4 is divided into frontend and backend. The frontend compilation converts the program into a Table Dependency Graph (TDG) which can be used by various backend compilers.

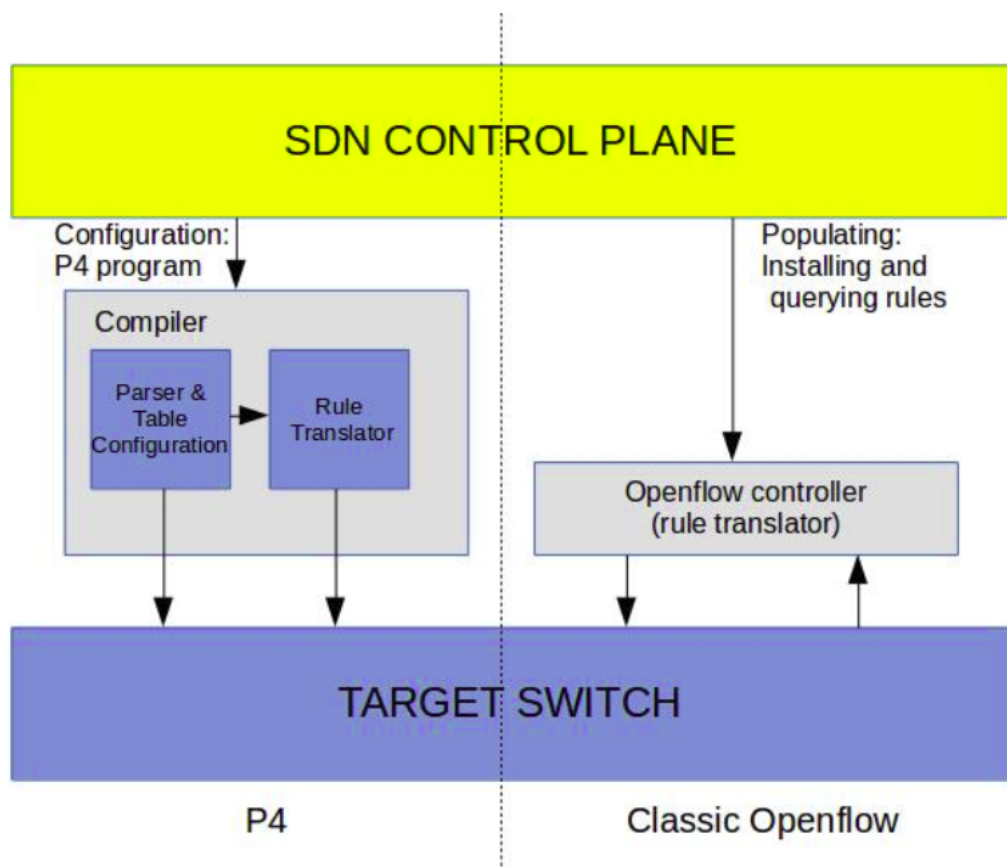


Figure 2.3: P4 vs Openflow

CHAPTER 3

SAMPLE PROGRAMS

Two sample programs were written in P4 and tested. The installation of P4 and the version used are mentioned in Appendix A. The programs are explained below:

3.1 EasyRoute protocol

A very simple source routing protocol is implemented using P4. The switch is completely written in P4 and tested using a mininet network. The EasyRoute protocol is a simple protocol in which packets are designed as follows:

```
preamble (8 bytes) | num_valid (4 bytes) | port_1 (1 byte) |  
port_2 (1 byte) | ... | port_n (1 byte) | payload
```

The *preamble* is taken as zero in this example. The *num_valid* field indicates the number of valid ports in the header. If your EasyRoute packet is to traverse 3 switches, *num_valid* will initially be set to 3, and the port list will be 3 byte long. When a switch receives an EasyRoute packet, the first port of the list is used to determine the outgoing port for the packet. *num_valid* is then decremented by 1 and the first port is removed from the list. Payload is the message that we will be sent. The topology of mininet network on which we are testing is shown in figure 3.1

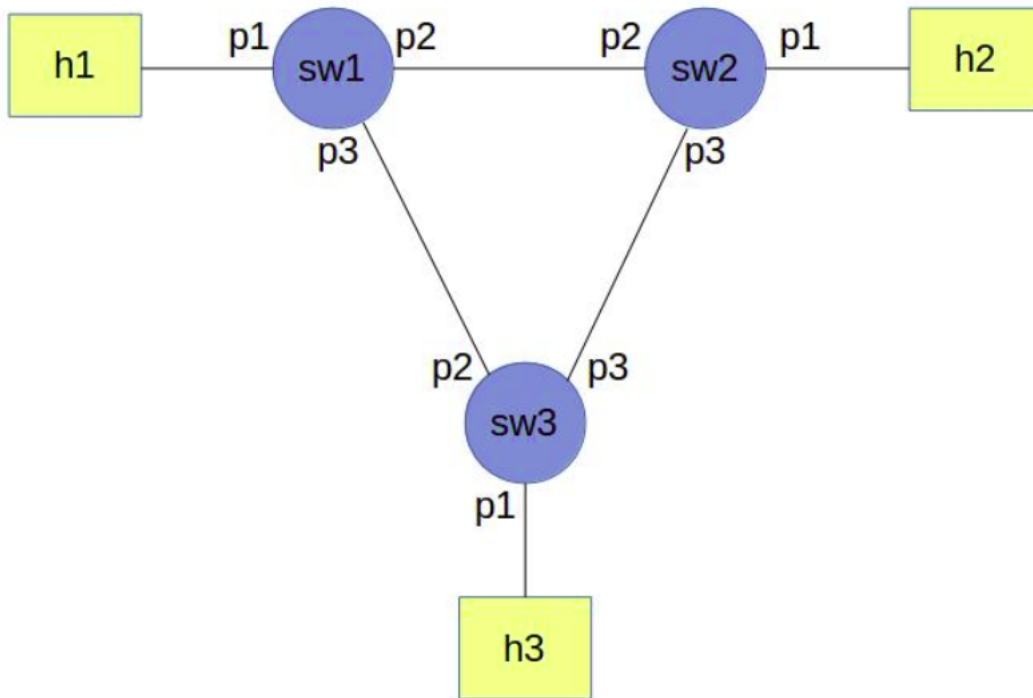


Figure 3.1: Topology used in mininet

It is also considered that non EasyRoute packets are dropped by the P4 switch. Also packets for which `num_valid` is 0 get dropped by the switches. The P4 code and implementation is shown in Appendix B. The P4 code has headers

- *easyroute_head* which has `preamble` and `num_valid` as the fields
- *easyroute_port* with `port` as field.

A parser to check if the packet is valid or not is written. Finally an action is designed to reduce the *num_valid* by 1 and remove the current port which is used by the table.

The target code is compiled using P4-hlir which generates a JSON output. The JSON output is loaded by the bmv2 switch model (which is based on the P4 abstraction model.) This is the configuration operation. We then populate the tables using CLI interface through a thrift server which runs in the bmv2 switch.

The hosts run a python program which sends the packets in the required format and also does a dijkstra's algorithm to find the shortest path. The results were as expected. If we try tweaking with the preamble value or the try send different packets they will not be received by the host at the end. Results can be found in Appendix B.

3.2 Flowlet Switching

Flowlet switching leverages the burstiness of TCP flows to achieve better load balancing of TCP traffic. The balance is done based on the layer 4 flows which is the classic ECMP. Flowlet switching is the hybrid between packet switching and flow based packet transmission. A TCP flow is considered as multiple small flows called flowlets which are distributed among different paths. To do this, we compute a hash over the 5-tuple and use this value to choose from a set of possible next hops. This means that all packets belonging to the same flow (i.e. with the same 5-tuple) will be routed to the same nexthop.

An addition of P4 code to a standard simple router (written in P4 language) gets the job done. To test the objective the following topology is taken as shown in figure 3.2 . Both nhop-0 and nhop-1 are assumed to be connected to the final destination giving rise to two paths via switch.

For each flow, we need to store the timestamp for last observed packet belonging to flow and a *flowlet_id*. Flowlet switching is very simple: for each packet which belongs to the flow, you need to update the timestamp. Then, if the time delta between the last observed packet and the current packet exceeds a certain timeout value then the *flowlet_id* will be incremented. With flowlet switching, packets

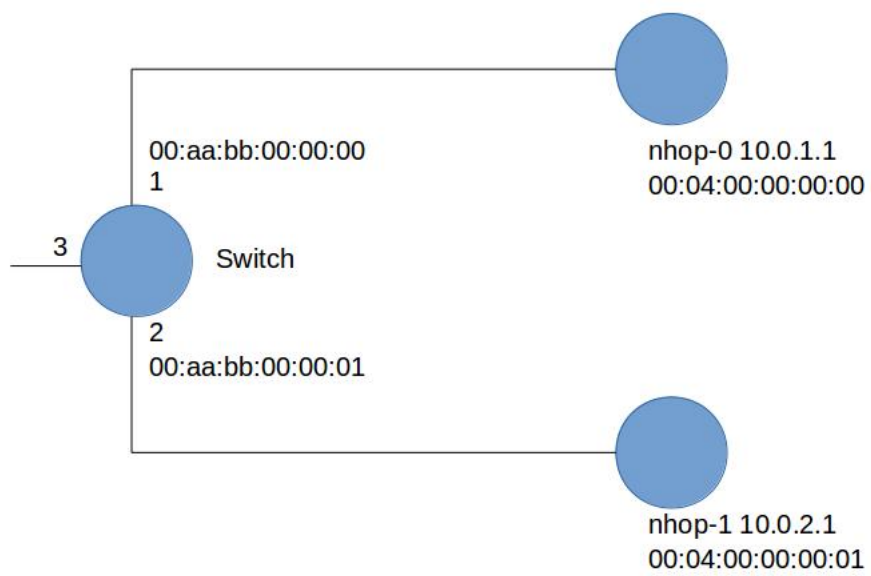


Figure 3.2: Topology used for testing


```
Terminal
ash@hpcomputer:~/temp/bmv2/targets/simple_switch$ sudo python run_test.py
1 0 0 0 1 0 0 1 0 1 1 1 1 1 0 1 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 1
1 0 1 0 0 0 1 1 0 1 1 1 1 1 1 0 0 1 1 0 0 0 1 1 0 1 1 1 0 0 0 0 1 0 1 1 1 0 1 0
1 1 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 1 0
0 1 0 0 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 1 1 1 0 0 1 0 1 1 1 1 0 1 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 1 0 1 0 1 0 0 0 0 1 0 1
0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 1 0 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0
1 1 1 1 0 0 1 0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 1 0
0 0 1 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1 0 1 0 1 0 1 0 0 0 1 1 0 0 1 0 1 0 1 1 1 1 1
0 1 0 1 0 0 0 1 1 0 1 1 0 0 0 0 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1
0 1 0 0 1 1 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 0 0 0 0 0 1 0 1 1 0 1 1 1 1 1 1 0 1 0
0 1 1 0 1 1 1 1 1 0 1 1 0 1 1 0 0 0 1 1 1 0 0 1 0 0 1 1 1 1 0 0 0 0 1 1 0 0 1 1
0 0 1 0 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1 0 0 1 0 1 1 1 0 0 1 0 0 0 0 1 1 0
1 1 1 1 1 0 1 1 0 0 1 1 0 1 0 0 0 0
ash@hpcomputer:~/temp/bmv2/targets/simple_switch$
```

Figure 3.4: With Modification of P4 code

distributed the flow among the different paths.

CHAPTER 4

Quagga application on a P4 based switch

Through this experiment we try to extend the capabilities of P4 switches and be able to run applications like quagga on them. Also we try to get closer to get an idea of how a hardware implementation can be done for this type of switches.

Quagga is a software for making hosts or switches capable of doing routing operations by performing protocols like BGP, OSPF etc.() To extend it to switches we need more resources to run applications. To test it on PC, a docker image containing P4, linux and essential libraries is created. Based on this image docker containers which act as P4 switches are connected in Mininet and the application is tested. The modules used in linking P4 with quagga and the method used to test is described in this chapter.

The P4 org has already provided a complete switch written in P4 with many supported features. This switch can be built with libraries like switchapi , switchsai or switchlink. These libraries are essential to form the bridge between APIs and the P4 switch. The soft switch is directly compiled from the P4 program. The lower level resource management API is autogenerated from the P4 code. On these the above libraries sit and communicate with external APIs. They are explained in detail below.

4.0.1 switchapi

switchapi lies on top of resource management api and supports features like Basic L2 switching, L2 multicast etc.

4.0.2 switchsai

SAI is an officially accepted API by OCP (Open Compute project). SAI allows software to program multiple switch chips without any changes, thus making the base router platform simple, consistent, and stable. It is analogous to a wrapper in a programming language but in strict sense it is a standardized API. switchsai exposes SAI on the switchapi.

4.0.3 switchlink

The switchlink library provides a netlink listener that listens to kernel notifications for network events (link, address, neighbor, route, etc.) and uses the switchsai library to program the data plane.

The stack arranged in order is as shown in Figure 4.1

This forms a complete switch which can support external applications.

But to test applications like quagga in a PC, we need multiple instances of the above switch which can be connected by mininet topology. We can't use containers used for mininet switches as they won't have sufficient resources to support the libraries. We take help of docker containers which help us in creating isolated package of software which can be easily used. The libraries used and how docker image was built can be found in Appendix D.

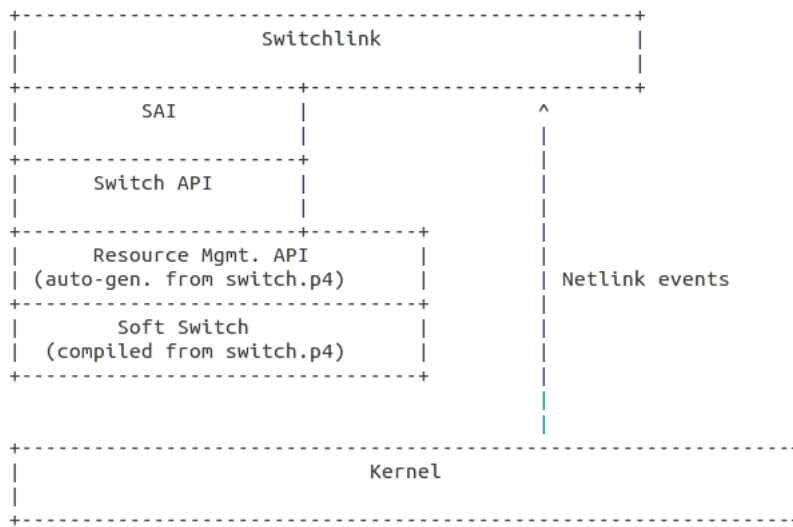


Figure 4.1: Architecture of the libraries

The topology used is the same as in figure 3.1. The topology along with IP addresses and subnets is in figure 4.2

The P4 switches can be configured using CLI. OSPF was the protocol that was implemented using quagga for routing. OSPF tries to update the routing tables with least cost paths. It took about 60 seconds for the Routing tables to get updated in the switches and then they begin to act as routers. Packets from a host h1 were as in figure 4.3 and from host h2 were as in figure 4.4

The figures show the movement of OSPF packets among the hosts (routers) and working of the quagga application. The final updation of the routing tables enabled the ICMP ping packets to reach from one host to another. This shows that quagga is working on the P4 switches.

A key point to be observed is that the P4 switch has its own resources. These resources are provided by suitable hardware (like NetFPGA or network processor) can support P4 switches along with applications.

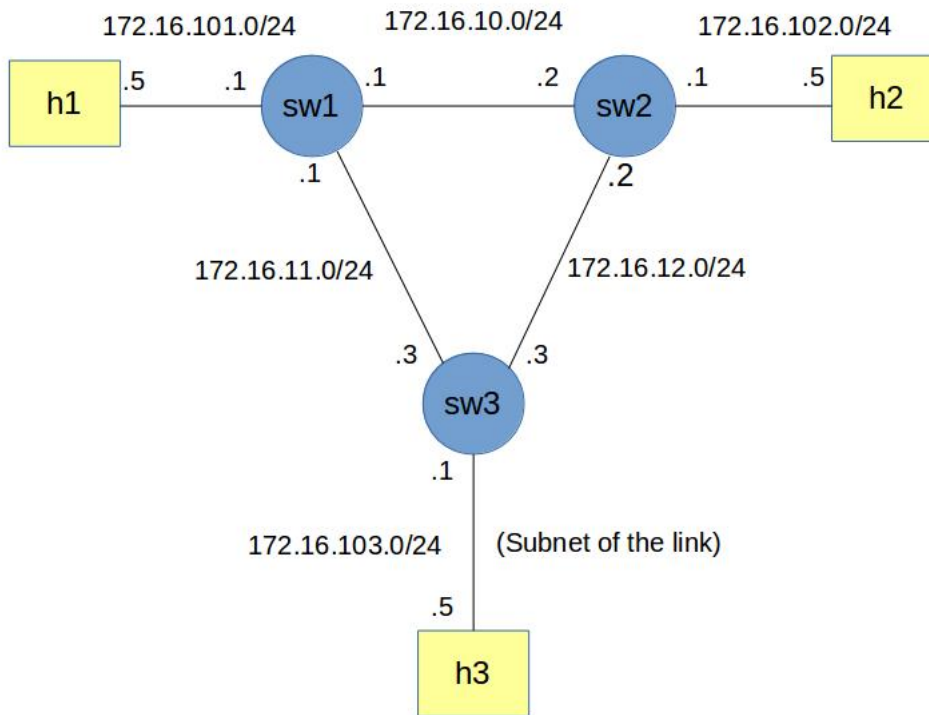


Figure 4.2: Topology with IP addresses

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fe80::204:ff:fe00:2	ff02::2	ICMPv6	70	Router Solicitation from 00:04:00:00:00:02
2	20.310547000	172.16.101.1	224.0.0.5	OSPF	78	Hello Packet
3	20.316240000	172.16.101.1	224.0.0.22	IGMPv3	54	Membership Report / Join group 224.0.0.5 for any sources
4	20.396283000	172.16.101.1	224.0.0.22	IGMPv3	54	Membership Report / Join group 224.0.0.5 for any sources
5	30.312561000	172.16.101.1	224.0.0.5	OSPF	78	Hello Packet
6	40.312611000	172.16.101.1	224.0.0.5	OSPF	78	Hello Packet
7	50.312638000	172.16.101.1	224.0.0.5	OSPF	78	Hello Packet
8	60.312820000	172.16.101.1	224.0.0.5	OSPF	78	Hello Packet
9	60.320426000	172.16.101.1	224.0.0.22	IGMPv3	54	Membership Report / Join group 224.0.0.6 for any sources
10	61.300394000	172.16.101.1	224.0.0.22	IGMPv3	54	Membership Report / Join group 224.0.0.6 for any sources
11	70.313703000	172.16.101.1	224.0.0.5	OSPF	78	Hello Packet
12	80.313817000	172.16.101.1	224.0.0.5	OSPF	78	Hello Packet
13	80.407038000	LexmarkI 00:00:02	Broadcast	ARP	42	Who has 172.16.101.1? Tell 172.16.101.5
14	80.407522000	EquipTra 00:00:01	LexmarkI 00:00:02	ARP	42	172.16.101.1 is at 00:01:00:00:00:01
15	80.407528000	172.16.101.5	172.16.102.5	ICMP	98	Echo (ping) request id=0x71a8, seq=1/256, ttl=64 (reply in 16)
16	80.409262000	172.16.102.5	172.16.101.5	ICMP	98	Echo (ping) reply id=0x71a8, seq=1/256, ttl=62 (request in 15)
17	80.411293000	172.16.101.5	172.16.103.5	ICMP	98	Echo (ping) request id=0x71a9, seq=1/256, ttl=64 (reply in 18)
18	80.412864000	172.16.103.5	172.16.101.5	ICMP	98	Echo (ping) reply id=0x71a9, seq=1/256, ttl=62 (request in 17)
19	80.414885000	172.16.102.5	172.16.101.5	ICMP	98	Echo (ping) request id=0x71aa, seq=1/256, ttl=62 (reply in 20)
20	80.414911000	172.16.101.5	172.16.102.5	ICMP	98	Echo (ping) reply id=0x71aa, seq=1/256, ttl=64 (request in 19)
21	80.419342000	172.16.103.5	172.16.101.5	ICMP	98	Echo (ping) request id=0x71ac, seq=1/256, ttl=62 (reply in 22)
22	80.419365000	172.16.101.5	172.16.103.5	ICMP	98	Echo (ping) reply id=0x71ac, seq=1/256, ttl=64 (request in 21)

Figure 4.3: Host h1 packets in wireshark

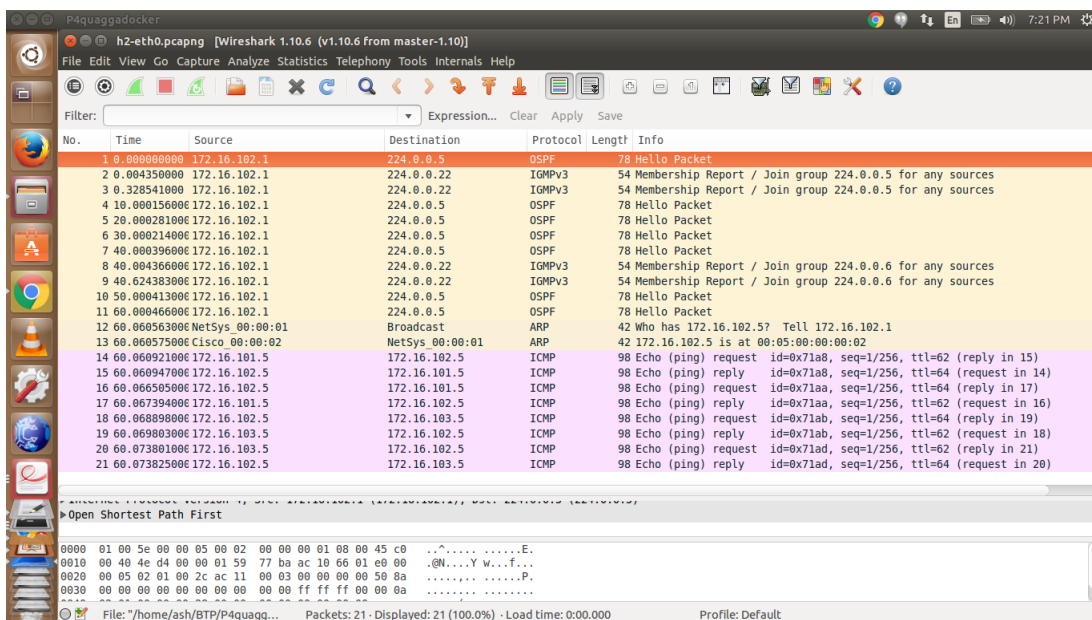


Figure 4.4: Host h2 packets in wireshark

CHAPTER 5

Current Hardware Implementations

There is no standard hardware platform for P4 switches yet. Efforts are being made to provide supporting hardware platform for P4.

5.1 FPGA

FPGA provides a robust platform. Now FPGAs can implement complex packet processing on single chip. Open platform used by researchers nowadays is NetFPGA. Third generation FPGAs with 100+ Gb/s line rates are being developed recently.

5.1.1 P4FPGA

Back end compiler which work on intermediate representation generated by p4-hlir is being developed. p4fpga.org is releasing a backend compiler which tries to map P4 programs to different FPGA platforms. The details can be found in. This compiler compiles the intermediate representation generated by p4-hlir to Bluespec systemverilog. Bluespec systemverilog is then to verilog which can used in FPGA firmware. Details can be found in Han Wang

5.1.2 Xilinx SDNet via PX

XilinxSDNet is coming up compilers which can convert a network program written in PX in to an effective FPGA implementation. PX is a network programming language but emphasis towards easy hardware very similar to P4 in many aspects (the language got its motivation from P4). They are also working on mappers to convert a P4 program into PX. This way FPGA implementation is being achieved. The details can be found in P4 to Fpga Brebner, 2015

5.2 Network Processor

Few companies are developing their own compilers to convert P4 to their respective App.firmware. They are building SDKs which can directly accept P4 Language input. One such example is seen in the white paper by netronome. It tries to take TDG generated by P4 compiler and generate C program for data path of NFP. White paper can be found in NETRONOME references.

Conclusions

1. P4 programming language tries to step towards more flexible switches whose functionality is specified and may be changed. It allows a programmers to work on how the forwarding plane processes packets without worrying about implementation details.
2. EasyRoute Protocol and Flowlet switching programs written in P4 has shown acceptable results on a software switch model.
3. To extend the capabilities of switches, libraries like switchsai, switchapi and switchlink are useful. They interact with external applications like quagga and improve the functionality of switches.
4. Using the above libraries a docker image was written in which OSPF was deployed using quagga on a P4 switch. The application ran successfully and P4 switches acted as routers.
5. Hardware Implementations on FPGA for P4 is still in development. Backend compilers and intermediate languages like PX are being created to support FPGA platform and network processors.
6. This above statement shows that P4's target independent programming ability is being recognised by organisations and it has a high scope of development in near future.

APPENDIX A

P4 installation

You can either download the p4 files from <https://github.com/p4lang/> or make git clone of the required repositories. (Consortium)

The steps followed during the installation are

- Clone the repositories p4factory for quagga related libraries and bmv2, p4c-bmv2 for easy protocol and flowlet switching
- Go to submodules folder and clone the submodules. This has to be checked for submodules of submodules too

Table A.1 shows the git branches (also commits which are not really important but for information) of the P4 I worked with

The procedure for installing and to test the installation is available in READ.md files. For information, the three main commands build are Using sudo is advised.

```
./autogen.sh
```

Repository or submodule	Branch	Commit ID
p4factory	Master	d91cee5ceb5cce77678c7dee29ec96e00fab3d94
p4factory/bm	Master	7decabc7cd8cb46d89bbe999883ffe11215db518
p4factory/oftest	Master	386180fea993f2be71fdef409a1b2131ec88a543
p4factory/p4c-behavioral	Master	ebe338d32593419437ae3e6c7cf43dff2ba1544c
p4factory/p4c-bm	Master	ebe338d32593419437ae3e6c7cf43dff2ba1544c
p4factory/p4ofagent	Master	63d7dafd211c9e405238492ed0ed499216871251
p4factory/ptf	Master	094882ee862e7f676f95ef031c9eeb27220e528c
p4factory/switch	Master	e3cadf2f56be38061a8cdf53357fe1a62917100
Bmv2 (behavioral-model)	Master	400b44865cf1ec2f30cb420d4ad699ff2751602b
p4c-bmv2	Master	e8e65c31df9de12cf54f2b1272df5e079aab91ce

Table A.1: Branch and commit id of the cloned modules (at time of downloading)

```
./configure
```

```
sudo make
```

APPENDIX B

EasyRoute protocol code and results

As explained in the section the EasyRoute P4 code is as below:

```
header_type easyroute_head_t {
    fields {
        preamble: 64;
        num_valid: 32;
    }
}

header easyroute_head_t easyroute_head;

header_type easyroute_port_t {
    fields {
        port: 8;
    }
}

header easyroute_port_t easyroute_port;

parser start {
    return select(current(0, 64)) {
        0: parse_head;
        default: ingress;
    }
}
```

```

parser parse_head {
    extract(easyroute_head);
    return select(latest.num_valid) {
        0: ingress;
        default: parse_port;
    }
}

parser parse_port {
    extract(easyroute_port);
    return ingress;
}

action _drop() {
    drop();
}

action route() {
    modify_field(standard_metadata.egress_spec, easyroute_port.port);
    add_to_field(easyroute_head.num_valid, -1);
    remove_header(easyroute_port);
}

table route_pkt {
    reads {
        easyroute_port: valid;
    }
    actions {
        _drop;
        route;
    }
}

```

```

    }

    size: 1;
}

control ingress {
    apply(route_pkt);
}

```

The topology is created by the following python code from p4.org

```

#!/usr/bin/python

# Copyright 2013-present Barefoot Networks, Inc.

#

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#

#    http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from mininet.net import Mininet
from mininet.topo import Topo

from mininet.log import setLogLevel, info

from mininet.cli import CLI

```

```

from mininet.link import TCLink

from p4_mininet import P4Switch, P4Host

import argparse

from time import sleep

import os

import subprocess

_THIS_DIR = os.path.dirname(os.path.realpath(__file__))

_THRIFT_BASE_PORT = 22222

parser = argparse.ArgumentParser(description='Mininet demo')

parser.add_argument('--behavioral-exe', help='Path to behavioral executable',
                    type=str, action="store", required=True)

parser.add_argument('--json', help='Path to JSON config file',
                    type=str, action="store", required=True)

parser.add_argument('--cli', help='Path to BM CLI',
                    type=str, action="store", required=True)

args = parser.parse_args()

class MyTopo(Topo):
    def __init__(self, sw_path, json_path, nb_hosts, nb_switches, links, **opts):
        # Initialize topology and default options
        Topo.__init__(self, **opts)
        for i in xrange(nb_switches):
            switch = self.addSwitch('s%d' % (i + 1),
                                    sw_path = sw_path,
                                    json_path = json_path,
                                    thrift_port = _THRIFT_BASE_PORT + i,
                                    pcap_dump = True,

```



```

device_id = i)

    for h in xrange(nb_hosts):
        host = self.addHost('h%d' % (h + 1))
        for a, b in links:
            self.addLink(a, b)
def read_topo():
    nb_hosts = 0
    nb_switches = 0
    links = []
    with open("topo.txt", "r") as f:
        line = f.readline()[:-1]
        w, nb_switches = line.split()
        assert(w == "switches")
        line = f.readline()[:-1]
        w, nb_hosts = line.split()
        assert(w == "hosts")
        for line in f:
            if not f: break
            a, b = line.split()
            links.append( (a, b) )
    return int(nb_hosts), int(nb_switches), links

def main():
    nb_hosts, nb_switches, links = read_topo()
    topo = MyTopo(args.behavioral_exe,

```

```

        args.json,
        nb_hosts, nb_switches, links)

net = Mininet(topo = topo,
              host = P4Host,
              switch = P4Switch,
              controller = None )

net.start()

for n in xrange(nb_hosts):
    h = net.get('h%d' % (n + 1))
    for off in ["rx", "tx", "sg"]:
        cmd = "/sbin/ethtool --offload eth0 %s off" % off
        print cmd
        h.cmd(cmd)

    print "disable ipv6"
    h.cmd("sysctl -w net.ipv6.conf.all.disable_ipv6=1")
    h.cmd("sysctl -w net.ipv6.conf.default.disable_ipv6=1")
    h.cmd("sysctl -w net.ipv6.conf.lo.disable_ipv6=1")
    h.cmd("sysctl -w net.ipv4.tcp_congestion_control=reno")
    h.cmd("iptables -I OUTPUT -p icmp --icmp-type destination-unreachable")

sleep(1)

for i in xrange(nb_switches):
    cmd = [args.cli, "--json", args.json,
          "--thrift-port", str(_THRIFT_BASE_PORT + i)]
    with open("commands.txt", "r") as f:
        print " ".join(cmd)
    try:

```

```

        output = subprocess.check_output(cmd, stdin = f)

        print output

    except subprocess.CalledProcessError as e:

        print e

        print e.output

    sleep(1)

    print "Ready !"

    CLI( net )

    net.stop()

if __name__ == '__main__':

    setLogLevel( 'info' )

    main()

```

The supporting text file “topo.txt”

```

switches 3

hosts 3

h1 s1

h2 s2

h3 s3

s1 s2

s1 s3

s2 s3

```

The script for running the topology

```

THIS_DIR=$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )

```

```

source $THIS_DIR/env.sh

P4C_BM_SCRIPT=$P4C_BM_PATH/p4c_bm/__main__.py

SWITCH_PATH=$BMV2_PATH/targets/simple_switch/simple_switch

CLI_PATH=$BMV2_PATH/tools/runtime_CLI.py

sudo PYTHONPATH=$PYTHONPATH:home/ash/p4lang/

$P4C_BM_SCRIPT p4src/source_routing.p4 --json source_routing.json

sudo PYTHONPATH=$PYTHONPATH:$BMV2_PATH/mininet/ python topo.py \
--behavioral-exe $BMV2_PATH/targets/simple_switch/simple_switch \
--json source_routing.json \
--cli $CLI_PATH

```

CLI commands used for populating the tables in switches. Stored in file “commands.txt”

```

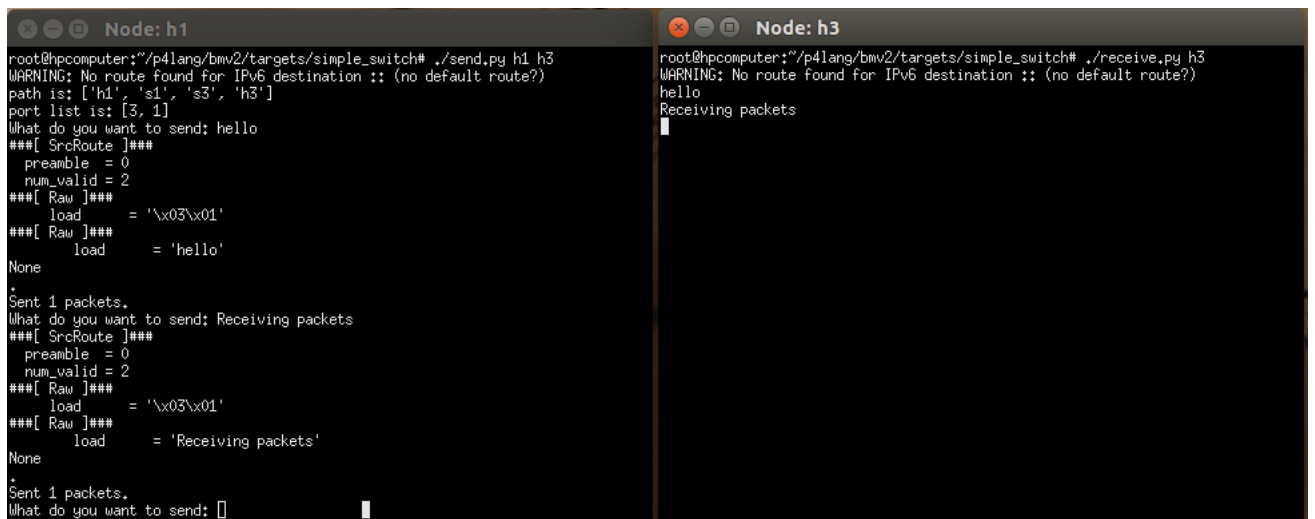
table_set_default route_pkt route

table_add route_pkt _drop 0 =>

```

The environmental variables should be updated properly in rundemo.py for the code to work. Also pythonpath should include all the necessary python modules. The results are shown in the following figure B.1

Messages sent from h1 are able to reach h2.



The image displays two terminal windows side-by-side, representing nodes in a Mininet network. The left window, titled 'Node: h1', shows the execution of a script that sends a packet to h3. The script outputs a warning about a missing IPv6 route, lists the path and port, and then displays the packet structure with a preamble and a 'hello' load. The right window, titled 'Node: h3', shows the execution of a script that receives a packet from h1. It also displays a warning about a missing IPv6 route and shows the received packet structure with a preamble and a 'Receiving packets' load.

```
Node: h1
root@hpccomputer:~/p4lang/bmv2/targets/simple_switch# ./send.py h1 h3
WARNING: No route found for IPv6 destination :: (no default route?)
path is: ['h1', 's1', 's3', 'h3']
port list is: [3, 1]
What do you want to send: hello
#### SrcRoute ####
  preamble = 0
  num_valid = 2
####[ Raw ]####
  load      = '\x03\x01'
####[ Raw ]####
  load      = 'hello'
None
Sent 1 packets.
What do you want to send: Receiving packets
#### SrcRoute ####
  preamble = 0
  num_valid = 2
####[ Raw ]####
  load      = '\x03\x01'
####[ Raw ]####
  load      = 'Receiving packets'
None
Sent 1 packets.
What do you want to send: []

Node: h3
root@hpccomputer:~/p4lang/bmv2/targets/simple_switch# ./receive.py h3
WARNING: No route found for IPv6 destination :: (no default route?)
hello
Receiving packets
```

Figure B.1: Easy Protocol testing in mininet

APPENDIX C

Flowlet switching

The modified P4 code for flowlet switching (modified from simple router code)

```
/*  
  
Copyright 2013-present Barefoot Networks, Inc.  
  
Licensed under the Apache License, Version 2.0 (the "License");  
you may not use this file except in compliance with the License.  
You may obtain a copy of the License at  
  
    http://www.apache.org/licenses/LICENSE-2.0  
  
Unless required by applicable law or agreed to in writing, software  
distributed under the License is distributed on an "AS IS" BASIS,  
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
See the License for the specific language governing permissions and  
limitations under the License.  
  
*/  
  
#include "includes/headers.p4"  
  
#include "includes/parser.p4"  
  
#include "includes/intrinsic.p4"  
  
#define FLOWLET_MAP_SIZE 13    // ADDED: 8K  
  
#define FLOWLET_INACTIVE_TOUT 50000 // ADDED: usec -> 50ms  
  
header_type ingress_metadata_t {  
    fields {
```

```

        flow_ipg : 32; // ADDED: inter-packet gap

        flowlet_map_index : FLOWLET_MAP_SIZE; // ADDED: flowlet map index

        flowlet_id : 16; // ADDED: flowlet id

        flowlet_lasttime : 32; // ADDED: flowlet's last reference time

        ecmp_offset : 14; //offset into the ecmp table

        nhop_ipv4 : 32;

    }

}

metadata ingress_metadata_t ingress_metadata;

action _drop() {

    drop();

}

//ADDED FIELDS (S)

field_list l3_hash_fields {

    ipv4.srcAddr;

    ipv4.dstAddr;

    ipv4.protocol;

    tcp.srcPort;

    tcp.dstPort;

}

field_list_calculation flowlet_map_hash {

    input {

        l3_hash_fields;

    }

    algorithm : crc16;

    output_width : FLOWLET_MAP_SIZE;

```

```

}

register flowlet_lasttime {
    width : 32;
    instance_count : 8192;
}

register flowlet_id {
    width : 16;
    instance_count : 8192;
}

// ADDED FIELDS (E)

action set_nhop(nhop_ipv4, port) {
    modify_field(ingress_metadata.nhop_ipv4, nhop_ipv4);
    modify_field(standard_metadata.egress_spec, port);
    add_to_field(ipv4.ttl, -1);
}

// ADDED ACTIONS and TABLES (S)

action lookup_flowlet_map() {
    modify_field_with_hash_based_offset(ingress_metadata.flowlet_map_index,
                                        flowlet_map_hash, FLOWLET_MAP_SIZE);
    register_read(ingress_metadata.flowlet_id,
                  flowlet_id, ingress_metadata.flowlet_map_index);
    modify_field(ingress_metadata.flow_ipg,
                  intrinsic_metadata.ingress_global_timestamp);
    register_read(ingress_metadata.flowlet_lasttime,
                  flowlet_lasttime, ingress_metadata.flowlet_map_index);
    subtract_from_field(ingress_metadata.flow_ipg,

```



```

        ingress_metadata.flowlet_lasttime);

    register_write(flowlet_lasttime, ingress_metadata.flowlet_map_index,
        intrinsic_metadata.ingress_global_timestamp);
}

table flowlet {
    actions { lookup_flowlet_map; }
}

action update_flowlet_id() {
    add_to_field(ingress_metadata.flowlet_id, 1);
    register_write(flowlet_id, ingress_metadata.flowlet_map_index,
        ingress_metadata.flowlet_id);
}

table new_flowlet {
    actions { update_flowlet_id; }
}

// ADDED ACTIONS and TABLES (E)

field_list flowlet_l3_hash_fields {
    ipv4.srcAddr;
    ipv4.dstAddr;
    ipv4.protocol;
    tcp.srcPort;
    tcp.dstPort;
    ingress_metadata.flowlet_id;
}

#define ECMP_BIT_WIDTH 10

#define ECMP_GROUP_TABLE_SIZE 1024

```

```

#define ECMP_NHOP_TABLE_SIZE 16384

field_list_calculation flowlet_ecmp_hash {
    input {
        flowlet_l3_hash_fields;
    }
    algorithm : crc16;
    output_width : ECMP_BIT_WIDTH;
}

action set_ecmp_select(ecmp_base, ecmp_count) {
    modify_field_with_hash_based_offset(ingress_metadata.ecmp_offset, ecmp_base,
                                        flowlet_ecmp_hash, ecmp_count);
}

table ecmp_group {
    reads {
        ipv4.dstAddr : lpm;
    }
    actions {
        _drop;
        set_ecmp_select;
    }
    size : ECMP_GROUP_TABLE_SIZE;
}

table ecmp_nhop {
    reads {
        ingress_metadata.ecmp_offset : exact;
    }
}

```

```

        actions {
            _drop;
            set_nhop;
        }
        size : ECMP_NHOP_TABLE_SIZE;
    }

    action set_dmac(dmac) {
        modify_field(ethernet.dstAddr, dmac);
    }

    table forward {
        reads {
            ingress_metadata.nhop_ipv4 : exact;
        }
        actions {
            set_dmac;
            _drop;
        }
        size: 512;
    }

    action rewrite_mac(smac) {
        modify_field(ethernet.srcAddr, smac);
    }

    table send_frame {
        reads {
            standard_metadata.egress_port: exact;
        }
    }

```

```

    actions {
        rewrite_mac;
        _drop;
    }
    size: 256;
}

control ingress {

    //ADDED FLOWLET TABLE

    apply(flowlet);
    if (ingress_metadata.flow_ipg > FLOWLET_INACTIVE_TOUT) {
        apply(new_flowlet);
    }

    apply(ecmp_group);
    apply(ecmp_nhop);
    apply(forward);
}

control egress {
    apply(send_frame);
}

```

Python code for creating the switch setup

```

#!/usr/bin/env python

# Copyright 2013-present Barefoot Networks, Inc.

#

# Licensed under the Apache License, Version 2.0 (the "License");

```

```

# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import time

NUM_PACKETS = 500

import random

import threading

from scapy.all import sniff

from scapy.all import Ether, IP, IPv6

from scapy.all import sendp

class PacketQueue:
    def __init__(self):
        self.pkts = []
        self.lock = threading.Lock()
        self.ifaces = set()

    def add_iface(self, iface):
        self.ifaces.add(iface)

    def get(self):
        self.lock.acquire()

```

```

        if not self.pkts:
            self.lock.release()
            return None, None
        pkt = self.pkts.pop(0)
        self.lock.release()
        return pkt

    def add(self, iface, pkt):
        if iface not in self.ifaces:
            return
        self.lock.acquire()
        self.pkts.append( (iface, pkt) )
        self.lock.release()

queue = PacketQueue()

def pkt_handler(pkt, iface):
    if IPv6 in pkt:
        return
    queue.add(iface, pkt)

class SnifferThread(threading.Thread):
    def __init__(self, iface, handler = pkt_handler):
        threading.Thread.__init__(self)
        self.iface = iface
        self.handler = handler

    def run(self):
        sniff(
            iface = self.iface,
            prn = lambda x: self.handler(x, self.iface)

```

```

    )

class PacketDelay:

    def __init__(self, bsize, bdelay, imin, imax, num_pkts = 100):

        self.bsize = bsize

        self.bdelay = bdelay

        self.imin = imin

        self.imax = imax

        self.num_pkts = num_pkts

        self.current = 1

    def __iter__(self):

        return self

    def next(self):

        if self.num_pkts <= 0:

            raise StopIteration

        self.num_pkts -= 1

        if self.current == self.bsize:

            self.current = 1

            return random.randint(self.imin, self.imax)

        else:

            self.current += 1

            return self.bdelay

pkt = Ether()/IP(dst='10.0.0.1', ttl=64)

port_map = {

    1: "veth3",

    2: "veth5",

    3: "veth7"

```

```

}

iface_map = {}

for p, i in port_map.items():
    iface_map[i] = p
queue.add_iface("veth3")
queue.add_iface("veth5")
for p, iface in port_map.items():
    t = SnifferThread(iface)
    t.daemon = True
    t.start()
import socket
send_socket = socket.socket(socket.AF_PACKET, socket.SOCK_RAW,
                             socket.htons(0x03))
send_socket.bind((port_map[3], 0))
delays = PacketDelay(10, 5, 25, 100, NUM_PACKETS)
ports = []
print "Sending", NUM_PACKETS, "packets ..."
for d in delays:
    # sendp is too slow...
    # sendp(pkt, iface=port_map[3], verbose=0)
    send_socket.send(str(pkt))
    time.sleep(d / 1000.)
time.sleep(1)
iface, pkt = queue.get()
while pkt:
    ports.append(iface_map[iface])

```



```

        iface, pkt = queue.get()

print ports

```

The script to initiate the topology is as follows

```

THIS_DIR=$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )

source $THIS_DIR/env.sh

P4C_BM_SCRIPT=$P4C_BM_PATH/p4c_bm/__main__.py

SWITCH_PATH=$BMV2_PATH/targets/flowlet_switching/simple_switch

CLI_PATH=$BMV2_PATH/tools/runtime_CLI.py

set -m

sudo PYTHONPATH=$PYTHONPATH:home/ash/p4lang/ $P4C_BM_SCRIPT p4src/simple_router.py

sudo echo "sudo" > /dev/null

sudo $BMV2_PATH/targets/flowlet_switching/simple_switch simple_router.json \
-i 0@veth0 -i 1@veth2 -i 2@veth4 -i 3@veth6 -i 4@veth8 \
--nanolog ipc:///tmp/bm-0-log.ipc \
--pcap &

sleep 2

$CLI_PATH --json simple_router.json < commands.txt

echo "READY!!!"

fg

```

Veth_setup script for creating interfaces

```

#!/bin/bash

noOfVeths=18

if [ $# -eq 1 ]; then

```

```

noOfVeths=$1

fi

echo "No of Veths is $noOfVeths"

idx=0

let "vethpairs=$noOfVeths/2"

while [ $idx -lt $vethpairs ]

do

intf0="veth$((idx*2))"

intf1="veth$((idx*2+1))"

idx=$((idx + 1))

if ! ip link show $intf0 &> /dev/null; then

ip link add name $intf0 type veth peer name $intf1

ip link set dev $intf0 up

ip link set dev $intf1 up

TOE_OPTIONS="rx tx sg tso ufo gso gro lro rxvlan txvlan rxhash"

for TOE_OPTION in $TOE_OPTIONS; do

/sbin/ethtool --offload $intf0 "$TOE_OPTION" off

/sbin/ethtool --offload $intf1 "$TOE_OPTION" off

done

fi

sysctl net.ipv6.conf.$intf0.disable_ipv6=1

sysctl net.ipv6.conf.$intf1.disable_ipv6=1

done

```

CLI commands used for populating the tables in switches. Stored in file “commands.txt”

```
table_set_default ecmp_group _drop
table_set_default ecmp_nhop _drop
table_set_default forward _drop
table_set_default send_frame _drop
table_set_default flowlet lookup_flowlet_map
table_set_default new_flowlet update_flowlet_id
table_add ecmp_group set_ecmp_select 10.0.0.1/32 => 0 2
table_add ecmp_nhop set_nhop 0 => 10.0.1.1 1
table_add ecmp_nhop set_nhop 1 => 10.0.2.1 2
table_add forward set_dmac 10.0.1.1 => 00:04:00:00:00:00
table_add forward set_dmac 10.0.2.1 => 00:04:00:00:00:01
table_add send_frame rewrite_mac 1 => 00:aa:bb:00:00:00
table_add send_frame rewrite_mac 2 => 00:aa:bb:00:00:01
```

The environmental variables should be updated properly in rundemo.py for the code to work. Sometimes pythonpath should be updated to include all necessary python modules

APPENDIX D

Building docker image

The docker setup file used is as follows. This can be found in p4factory/docker. Changes to libraries included in the docker image can be edited in this file.

```
FROM      ubuntu:14.04

MAINTAINER Antonin Bas <antonin@barefootnetworks.com>

RUN apt-get update

RUN apt-get install -y \
    automake \
    bridge-utils \
    build-essential \
    ethtool \
    git \
    libboost-dev \
    libboost-fs-dev \
    libboost-program-options-dev \
    libboost-system-dev \
    libboost-test-dev \
    libedit-dev \
    libevent-dev \
    libglib2.0-dev \
    libgmp-dev \
```

```

libhiredis-dev \
libjudy-dev \
libnl-route-3-dev \
libpcap0.8 \
libpcap0.8-dev \
libtool \
libssl-dev \
openssh-server \
packit \
pkg-config \
python-dev \
python-pygraph \
python-pygraphviz \
python-setuptools \
python-thrift \
python-yaml \
quagga \
redis-server \
redis-tools \
subversion \
tshark \
xterm

# install thrift
RUN mkdir -p /tmp/thrift ; \
    cd /tmp/thrift ; \
    wget -q http://archive.apache.org/dist/thrift/0.9.2/thrift-0.9.2.tar.gz

```

```

tar xvzf thrift-0.9.2.tar.gz; \
cd thrift-0.9.2; \
./configure ; cd test/cpp ; ln -s . .libs ; cd ../.. ; \
make -j 4 install; ldconfig ; \
rm -fr /tmp/thrift
# install scapy
RUN mkdir -p /tmp/scapy ; \
cd /tmp/scapy ; \
git clone https://github.com/p4lang/scapy-vxlan.git ; \
cd scapy-vxlan ; \
python setup.py install ; \
rm -fr /tmp/scapy
# install p4-hlir
RUN mkdir -p /tmp/p4-hlir ; \
cd /tmp/p4-hlir ; \
git clone https://github.com/p4lang/p4-hlir.git ; \
cd p4-hlir ; \
python setup.py install ; \
rm -fr /tmp/p4-hlir
# install mstpd
RUN mkdir -p /third-party/diffs
COPY diffs/mstpd.diff /third-party/diffs/mstpd.diff
RUN cd /third-party; \
    svn checkout svn://svn.code.sf.net/p/mstpd/code/trunk mstpd; \
    cd mstpd; patch -p0 -i /third-party/diffs/mstpd.diff; make install
# install ctypesgen

```

```

RUN mkdir -p /tmp/ctypesgen ; \
    cd /tmp/ctypesgen ; \
    git clone https://github.com/davidjamesca/ctypesgen.git ; \
    cd ctypesgen ; \
    python setup.py install ; \
    rm -fr /tmp/ctypesgen

#install nanomsg
RUN mkdir -p /tmp/nanomsg ; \
    cd /tmp/nanomsg ; \
    wget -q http://download.nanomsg.org/nanomsg-0.5-beta.tar.gz ; \
    tar xvzf nanomsg-0.5-beta.tar.gz ; \
    cd nanomsg-0.5-beta ; \
    ./configure ; \
    make && sudo make install ; \
    rm -fr /tmp/nanomsg

ADD p4factory /p4factory

ENV VTYSH_PAGER more

```

To build the image, change directory to targets/switch in terminal and enter the command

```
sudo make docker-image
```

The docker image is built if a similar output is observed when the first line is entered into terminal.

```
sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
p4dockerswitch	latest	d9d6b0a6ab79	7 weeks ago
ubuntu	14.04	07c86167cdc4	3 months ago

The python code for running the mininet network is

```

from mininet.net import Mininet, VERSION
from mininet.log import setLogLevel, info
from mininet.cli import CLI
from distutils.version import StrictVersion
from p4_mininet import P4DockerSwitch
from time import sleep

import sys

def main(cli = 0, ipv6 = 0):
    net = Mininet( controller = None )

    # add hosts
    h1 = net.addHost( 'h1', ip = '172.16.101.5/24', mac = '00:04:00:00:00:02' )
    h2 = net.addHost( 'h2', ip = '172.16.102.5/24', mac = '00:05:00:00:00:02' )
    h3 = net.addHost( 'h3', ip = '172.16.103.5/24', mac = '00:06:00:00:00:02' )

    # add switch 1
    sw1 = net.addSwitch( 'sw1', target_name = "p4dockerswitch",
        cls = P4DockerSwitch, config_fs = 'configs/sw1/l3_ospf',
        pcap_dump = True )

    # add switch 2
    sw2 = net.addSwitch( 'sw2', target_name = "p4dockerswitch",
        cls = P4DockerSwitch, config_fs = 'configs/sw2/l3_ospf',
        pcap_dump = True )

```



```

#add switch 3

sw3 = net.addSwitch( 'sw3', target_name = "p4dockerswitch",
cls = P4DockerSwitch, config_fs = 'configs/sw3/l3_ospf',
pcap_dump = True )

# add links

if StrictVersion(VERSION) <= StrictVersion('2.2.0'):
net.addLink( sw1, h1, port1 = 1 )
net.addLink( sw1, sw2, port1 = 2, port2 = 2 )
net.addLink( sw1, sw3, port1 = 3, port2 = 2 )
net.addLink( sw2, h2, port1 = 1 )
net.addLink( sw2, sw3, port1 = 3, port2 = 3 )
net.addLink( sw3, h3, port1 = 1 )
else:
net.addLink( sw1, h1, port1 = 1, fast=False )
net.addLink( sw1, sw2, port1 = 2, port2 = 2, fast=False )
net.addLink( sw1, sw3, port1 = 3, port2 = 2, fast=False )
net.addLink( sw3, h3, port1 = 1, fast=False )
net.addLink( sw2, h2, port1 = 1, fast=False )
net.addLink( sw2, sw3, port1 = 3, port2 = 3, fast=False )
net.start()

# configure hosts

h1.setDefaultRoute( 'via 172.16.101.1' )
h2.setDefaultRoute( 'via 172.16.102.1' )
h3.setDefaultRoute( 'via 172.16.103.1' )

if ipv6:

# hosts configuration - ipv6

```

```

h2.setIP6( '2ffe:0102::5', 64, 'h2-eth0' )
h1.setIP6( '2ffe:0101::5', 64, 'h1-eth0' )
h3.setIP6( '2ffe:0103::5', 64, 'h3-eth0' )
h1.setDefaultRoute( 'via 2ffe:0101::1', True )
h2.setDefaultRoute( 'via 2ffe:0102::1', True )
h3.setDefaultRoute( 'via 2ffe:0103::1', True )

#Opening wireshark for each of the hosts

# for host in net.hosts:

# host.cmd('wireshark &')

# sleep (10)

#TODO:tshark in switches

#

sw1.cmd( 'tshark -w /tmp/sw1.pcap -i swp1 -i swp2 -i swp3 ' )

#

sw2.cmd( 'tshark -w /tmp/sw2.pcap -i swp1 -i swp2 -i swp3 ' )

#

sw3.cmd( 'tshark -w /tmp/sw3.pcap -i swp1 -i swp2 -i swp3 ' )

# print "DONE pcap"

#Start Quagga in switches

sw1.cmd( 'service quagga start')
sw2.cmd( 'service quagga start')
sw3.cmd( 'service quagga start')

result = 0

if cli:

CLI( net )

```

```

else:
    sleep(60)
    node_values = net.values()
    print node_values
    hosts = net.hosts
    print hosts
    # ping hosts
    print "PING BETWEEN THE HOSTS"
    result = net.ping(hosts,30)
    if result != 0:
        print "PING FAILED BETWEEN HOSTS %s" % (hosts)
    else:
        print "PING SUCCESSFUL!!!"
    if ipv6:
        print "PING6 BETWEEN THE HOSTS"
        result = net.ping6(hosts, 30)
        if result != 0:
            print "PING6 FAILED BETWEEN HOSTS %s" % (hosts)
        else:
            print "PING6 SUCCESSFUL!!!"
    # print host arp table & routes
    for host in hosts:
        print "ARP ENTRIES ON HOST"
        print host.cmd('arp -n')
        print "HOST ROUTES"
        print host.cmd('route')

```

```

print "HOST INTERFACE LIST"

intfList = host.intfNames()

print intfList

#Something more to dump pcap files

net.stop()

return result

if __name__ == '__main__':

args = sys.argv

setLogLevel( 'info' )

cli = 0

ipv6 = 0

if "--cli" in args:

cli = 1

if "--ipv6" in args:

ipv6 = 1

main(cli, ipv6)

```

The startup_config script for building the switch is

```

#!/bin/bash

stty -echo; set +m

ip link set dev swp1 address 00:01:00:00:00:01

ip link set dev swp2 address 00:01:00:00:00:02

ip link set dev swp3 address 00:01:00:00:00:03

ip address add 172.16.101.1/24 broadcast + dev swp1

ip address add 172.16.10.1/24 broadcast + dev swp2

ip address add 172.16.11.1/24 broadcast + dev swp3

```

```
sysctl -q net.ipv6.conf.all.forwarding=1  
  
ip address add 2ffe:0101::1/64 dev swp1  
  
ip address add 2ffe:0010::1/64 dev swp2  
  
ip address add 2ffe:0011::1/64 dev swp3  
  
cp /configs/quagga/* /etc/quagga/  
  
chown quagga.quagga /etc/quagga/*
```

Switches need to be configured and the IP addresses of the neighbours are to be entered in the config file of switches. We are not using BGP to let the switches know all the IPs automatically so the config files needs to be changes. The config file for one switch is shown below:

ospf.conf

```
hostname ospfd  
  
password zebra  
  
enable password zebra  
  
router ospf  
  
network 172.16.10.0/24 area 0  
  
network 172.16.11.0/24 area 0  
  
network 172.16.101.0/24 area 0
```

REFERENCES

- Brebner, G.** (2015). P4 for an fpga target. URL http://schd.ws/hosted_files/p4workshop2015/33/GordonB-P4-Workshop-June-04-2015.pdf. 5.1.2
- Consortium, P. L.** (2015a). The p4 language specification. URL <http://p4.org/wp-content/uploads/2015/04/p4-latest.pdf>. 2.2.4
- Consortium, P. L.** (2015b). The p4 language web site. URL <http://www.p4.org>. A
- Han Wang, K. S. L. H. W., Vishal Shrivastav** (2015). P4fpga: Towards an open source p4 backend for fpga. URL <http://p4.org/wp-content/uploads/2015/12/Cornell-Demo-Poster.pdf>. 5.1.1
- Heller, B.** (2012). Sdn for engineers. URL <http://opennetsummit.org/archives/apr12/site/talks/heller-tutorial.pdf>. 1
- NETRONOME** (2015). Programming nfp with p4 and c. 5.2
- Paul Jakma, V. J. and G. Troxel** (v1.9, 2015). Quagga routing suite. URL <http://www.nongnu.org/quagga/>.
- Ross, J. F. K. . K. H.,** *Computer Networking A Top-Down Approach*. Pearson, 2013.
- Sivaraman, A.** (2015). P4 language evolution. URL <http://p4.org/p4/p4-language-evolution/>.
- Subramaniam, K.** (2015). Switch abstraction interface (sai) officially accepted by the open compute project (ocp). URL <https://azure.microsoft.com/en-in/blog/switch-abstraction-interface-sai-officially-accepted-by-the-open-compute-project-ocp/>.