INF-556 Lab: P4

nov/2017

Hands-on using a pre-configured VM

Agenda

- ➔ Overview
- ➔ Environment Setup
- ➔ P4 Program Execution
- ➔ P4 Examples
  - ◆ L2 Switch
  - ◆ Simple Router
  - ◆ Simple Router + ACL
- ➔ P4 Dependency Graph Generator

Overview

- ➔ This tutorial aims to
  - ◆ show some P4 examples,
  - ◆ present types of software switch (behavioral models),
  - ◆ integration with Mininet
  - ◆ use P4-graphs tool to generate P4 dependency graphs.

- ➔ We provide a Virtual Machine with all previously installed packages that the software switch needs to run P4 programs.

Why behavioral model V2?

- P4 provides two types of behavioral model (BM). BM is a software switch which runs with the P4 defined datapath.
  - The bmv2 embodies various new features and will be the defacto switch for P4 from now on.

- Some interesting features of bmv2 are:
  - Second version of the P4 switch written in C++.
  - It is not necessary to auto-generate new code and recompile the switch every time the P4 code is modified (Static).
  - Debug logging feature is available.
  - Unit tests possible.

P4 Execution Process

How to execute a P4 program?

How to execute a simple P4 program?

Transform the P4 code into json representation:

```
p4c-bm2  --json  <path to JSON file>   <path to P4 file>
```

Then, execute the simple_switch target using the json file generated above:

```
sudo ./simple_switch  -i  0@<iface0>  -i  1@<iface1>  <path to JSON file>
```

➔ <iface0> and <iface1> are the interfaces which are bound to the switch (port0 and port1)

❖ **" simple_switch" target is the model target to be used with every P4 program.**

How to use P4 switch CLI ?

Using CLI to populate tables

The runtime_CLI tool connects to the Thrift RPC server through 9090 port. One CLI can only connect to one switch device.

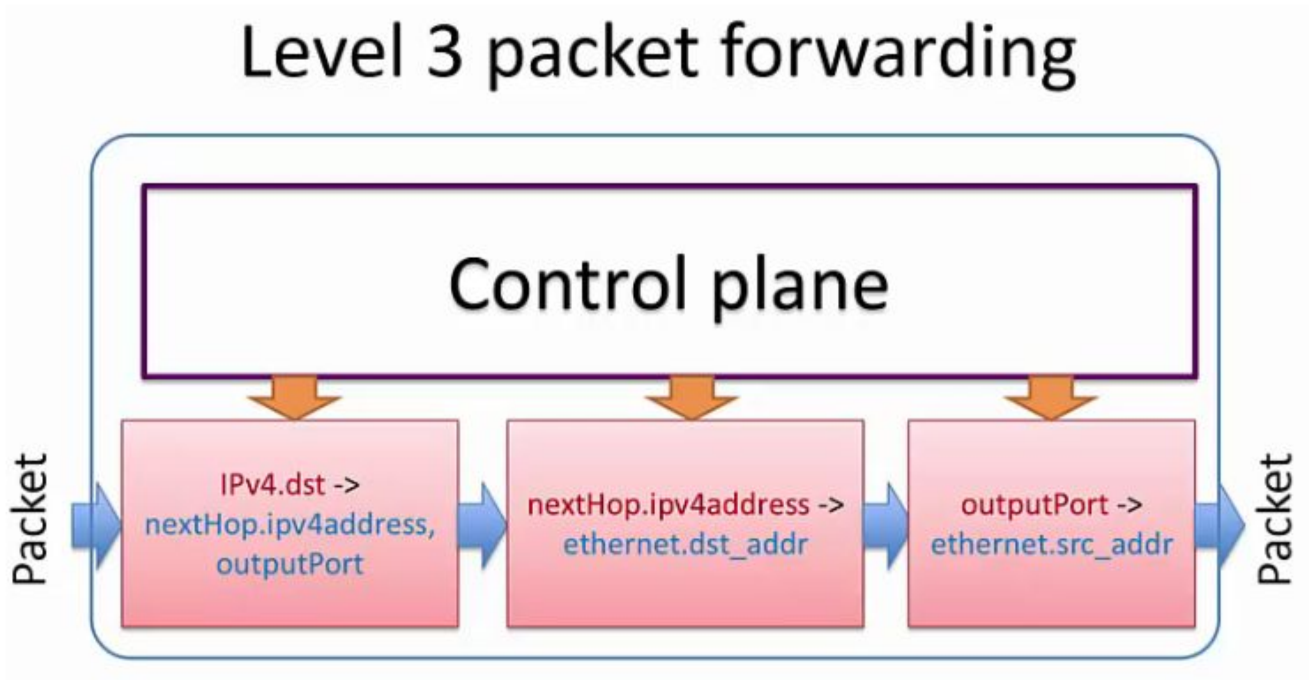
```
./runtime_CLI.py  --json  <path to JSON file>  --thrift-port 9090
```

The CLI is realized using the Python's cmd module and supports auto-completion.

```
table_set_default <table name> <action name> <action parameters>
table_add <table name> <action name> <match fields> => <action parameters> [priority]
table_delete <table name> <entry handle>
```

P4 Pipeline Example Overview

Illustrative representation of the “Simple Router” example:


Level 3 packet forwarding
Control plane
Packet
IPv4.dst -> nextHop.ipv4address, outputPort
nextHop.ipv4address -> ethernet.dst_addr
outputPort -> ethernet.src_addr
Packet

Hands On

Practical Activities

(DONE) P4 Environment Setup

1. Download the P4 tutorial VM image. OVA Source:

https://intrig.dca.fee.unicamp.br:8840/owncloud/index.php/s/cdv26HcrJTlgJTF

password: p4

2. Install the hypervisor. (**Virtualbox**)

3. Credentials for accessing:
    - **user:** ubuntu
    - **password:** ubuntu

P4 Environment Setup

➔ The behavioral model v2 code is already downloaded at:

```
$cd ~/behavioral-model/
```

➔ Create necessary veth interfaces required by P4 Switch. (**Need to execute this script each time the VM starts**)

```
$sudo ./tools/veth_setup.sh
```

The P4 code base is already downloaded and compiled in the VM. The next slides will explain how to install the “behavioral-model” and “p4c-bm” code for informational purpose. You can directly move to the P4 examples section.

(DONE) P4 Environment Setup

a) Compile and build bmv2

To build the behavioral-model code:

```
$ ./autogen.sh
$ ./configure
$ make
$ make install
```

You can build ‘bmv2’ with different configuration options:

- ❖ Disable logging macros for higher performance:
  `./configure  --disable-logging-macros`
- ❖ Enable debug mode: `./configure  --enable-debugger`
- ❖ In debug, you may disable compiler optimizations and enable symbols in the binary: `./configure 'CXXFLAGS=-O0 -g'`

(DONE) P4 Environment Setup

b) Compile and build p4c-bm

It generates Json configuration file required for the bmv2. Usage of the json file is explained later.

To install p4c-bm on your machine:

```
$ git clone https://github.com/p4lang/p4c-bm
$ cd p4c-bm
$ sudo pip install -r requirements.txt
$ sudo pip install -r requirements_v1_1.txt  (only P4 v1.1 programs)
$ sudo python setup.py install
```

Try :

```
$ p4c-bmv2 -h
```

--json to generate a JSON representation of the P4 program
--pd to generate the PD C++ code
--p4-v1.1 if your input program is a P4 v1.1 program

P4 Example Applications

P4 Examples and Documentation

- Find the P4 examples at:

  $ ~/behavioral-model/targets

  - l2_switch
  - simple_router
  - simple_router + ACL

- P4 documentation:
- P4_14 specification: https://p4lang.github.io/p4-spec/
- Code repository: https://github.com/p4lang/

L2 Switch

1. L2 switch

Compiling your first .p4 program

1. Open a terminal and generate json file for ‘***l2_switch***’ target as below:

```
cd ~/behavioral-model/targets/l2_switch
```

```
sudo p4c-bmv2 --json l2_switch.json l2_switch.p4
```

Q1: Open the the code ***l2_switch.p4*** and answer to the following questions:
- How many and which packet headers are defined in the p4 program?

- How many tables are defined?

- Which are the contents (packet header fields) used for lookup in each table?
  (Tip: In addition to inspecting the P4 code, analyze the image outputs of: `p4-graphs l2_switch.p4`)

1. L2 switch

2. Start the P4 data plane

```
sudo ./l2_switch -i 0@veth1 -i 1@veth3 -i 2@veth5 -i 3@veth7 l2_switch.json
  Thrift port was not specified, will use 9090
  Adding interface veth1 as port 0
  Adding interface veth3 as port 1
  Adding interface veth5 as port 2
  Adding interface veth7 as port 3
  Thrift server was started
```

1. Use CLI with L2 switch

3. In another terminal, we launch the CLI tool to populate the tables using the runtime_CLI script.

```
cd ~/behavioral-model/targets/l2_switch
```

```
 ./runtime_CLI --json l2_switch.json --thrift-port 9090
```

1. Use CLI with L2 switch

4. CLI examples for runtime manipulation of P4 tables

```
$:    show_tables
```

```
dmac                [implementation=None, mk=ethernet.dstAddr(exact, 48)]
smac                [implementation=None, mk=ethernet.srcAddr(exact, 48)]
```

```
$:    show_ports
```

```
 port #        iface name     status  extra info
=================================================
      0        veth1          UP      out_pcap=veth1.pcap; in_pcap=veth1.pcap
      1        veth3          UP      out_pcap=veth3.pcap; in_pcap=veth3.pcap
      2        veth5          UP      out_pcap=veth5.pcap; in_pcap=veth5.pcap
      3        veth7          UP      out_pcap=veth7.pcap; in_pcap=veth7.pcap
```

```
$:    show_actions
```

```
_nop                 []
broadcast            []
forward              [port(9)]
mac_learn            []
```

1. Use CLI with L2 switch

5. Run the below commands to update the tables:

```
RuntimeCmd: table_set_default smac mac_learn
Setting default action of smac
action:         mac_learn
runtime data:

RuntimeCmd: table_set_default dmac forward 1
Setting default action of dmac
action:         forward
runtime data:       00:01
```

Q2: What do these two commands do?

1. Test L2 switch

Testing

Once the tables are updated with the default entries, close the CLI and ping any IP address. From another terminal you may capture packets from the interface veth3 of the P4 dataplane.

Terminal #1: ***ping -I veth1 <any IP address>***

```
From 10.0.2.15 icmp_seq=1 Destination Host Unreachable
From 10.0.2.15 icmp_seq=2 Destination Host Unreachable
```

Terminal #2: ***sudo tshark -i veth3***

```
Capturing on 'veth3'
  1   0.000000 6a:b6:4c:1a:4a:f4 -> Broadcast        ARP 42 Who has <any IP address>?  Tell 10.0.2.15
```

Simple Router

2. simple_router

1. **Be sure to end the process** of the l2_switch by running ctrl+c on the terminal it was running it

2. Generate json file for ‘***simple_router***’ target as below:

```
cd ~/behavioral-model/targets/simple_router/
```

```
sudo p4c-bmv2 --json simple_router.json simple_router.p4
```

3. Run simple_router in Mininet (analogous to the L2 switch example).

```
cd ~/behavioral-model/mininet/
```

```
sudo python 1sw_demo.py --behavioral-exe ../targets/simple_router/simple_router --json
../targets/simple_router/simple_router.json
```

2. Use CLI with simple_router

4. In another terminal, we launch the CLI tool to populate the tables using the runtime_CLI script.

```
cd ~/behavioral-model/targets/simple_router/
./runtime_CLI < commands.txt
```

The file commands.txt have the following tables entries.

```
table_set_default send_frame _drop
table_set_default forward _drop
table_set_default ipv4_lpm _drop
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
table_add forward set_dmac 10.0.0.10 => 00:04:00:00:00:00
table_add forward set_dmac 10.0.1.10 => 00:04:00:00:00:01
table_add ipv4_lpm set_nhop 10.0.0.10/32 => 10.0.0.10 1
table_add ipv4_lpm set_nhop 10.0.1.10/32 => 10.0.1.10 2
```

Q3: For each table of the datapath pipeline, describe the inserted entries (Key:Match fields + Value:Actions).

2. Test simple_router with mininet

5. Testing:

Once the tables are updated with necessary entries, ping will be successful in mininet topology.

```
mininet > h1 ping h2
PING 10.0.1.10 (10.0.1.10) 56(84) bytes of data.
64 bytes from 10.0.1.10: icmp_seq=7 ttl=63 time=4.71 ms
64 bytes from 10.0.1.10: icmp_seq=8 ttl=63 time=4.48 ms
64 bytes from 10.0.1.10: icmp_seq=9 ttl=63 time=0.960 ms
```

6. Finally, stop “mininet”:

```
mininet > exit
```

Simple Router + ACL

3. simple_router + ACL

1. Generate json file for '***simple_router***' target as below:

```
cd ~/p4lang/bmv2/targets/simple_router_acl
sudo p4c-bmv2 --json simple_router_acl.json simple_router_acl.p4
```

2. Run simple_router_acl in Mininet

```
sudo mn -c
cd ~/p4lang/bmv2/mininet/
sudo python 1sw_demo.py --behavioral-exe ../targets/simple_router/simple_router --json
../targets/simple_router_acl/simple_router_acl.json
```

3. Use CLI with simple_router + ACL

3. In another terminal, we launch the CLI tool to populate the tables using the runtime_CLI script.

```
cd ~/p4lang/bmv2/targets/simple_router_acl
./runtime_CLI < commands.txt
```

The commands.txt has the following rules:

```
table_set_default send_frame _drop
table_set_default forward _drop
table_set_default ipv4_lpm _drop
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
table_add forward set_dmac 10.0.0.10 => 00:04:00:00:00:00
table_add forward set_dmac 10.0.1.10 => 00:04:00:00:00:01
table_add ipv4_lpm set_nhop 10.0.0.10/32 => 10.0.0.10 1
table_add ipv4_lpm set_nhop 10.0.1.10/32 => 10.0.1.10 2
```

Q4: Compare the P4 code of simple router with simple router + ACL. Explain the main observed differences.

3. Test simple_router with mininet + ACL

4. Testing:

Once the tables are updated with necessary entries, ping will be successful in mininet topology.

```
mininet > h1 ping h2
PING 10.0.1.10 (10.0.1.10) 56(84) bytes of data.
64 bytes from 10.0.1.10: icmp_seq=7 ttl=63 time=4.71 ms
64 bytes from 10.0.1.10: icmp_seq=8 ttl=63 time=4.48 ms
64 bytes from 10.0.1.10: icmp_seq=9 ttl=63 time=0.960 ms
```

5. Enter hosts “h1” and “h2” with the ***xterm*** command:

```
mininet > xterm h1 h2
```

3. Test simple_router with mininet + ACL

6. By default, h1 and h2 does not have any filter configured in their systems.

   Add the following rule to filter the traffic:

   ```
   cd ~/p4lang/bmv2/targets/simple_router_acl
   ./runtime_CLI
   ```

   # Filter all traffic

   ```
   RuntimeCmd: table_set_default acl _drop
   Setting default action of acl
   action:           _drop
   runtime data:
   ```

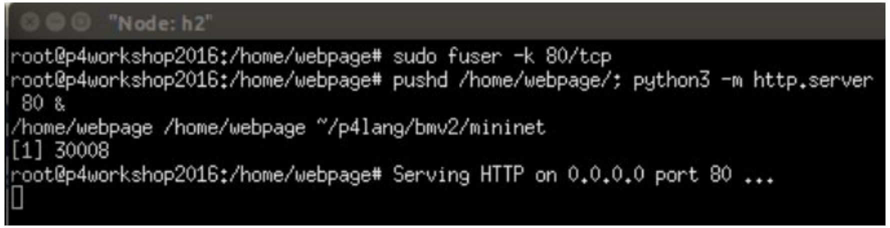
3. Test simple_router with mininet + ACL

7. Testing ***all traffic filtered*** using the ***curl*** command:

   In h2:

```
sudo fuser -k 80/tcp
```

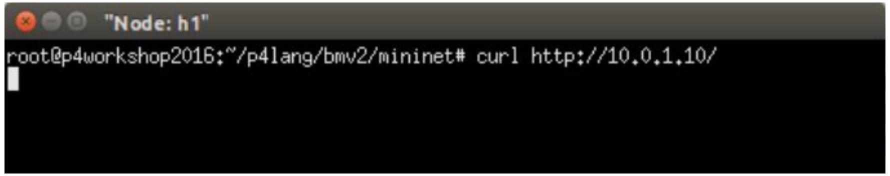
```
pushd /home/webpage/; python3 -m http.server 80 &
```



```
root@p4workshop2016:/home/webpage# sudo fuser -k 80/tcp
root@p4workshop2016:/home/webpage# pushd /home/webpage/; python3 -m http.server
 80 &
/home/webpage /home/webpage ~/p4lang/bmv2/mininet
[1] 30008
root@p4workshop2016:/home/webpage# Serving HTTP on 0.0.0.0 port 80 ...
```

   In h1

```
curl http://10.0.1.10
```



```
root@p4workshop2016:~/p4lang/bmv2/mininet# curl http://10.0.1.10/
```

3. Test simple_router with mininet + ACL

8. Add the following rule to allow the traffic:

   # Allow all traffic

   RuntimeCmd: ***table_set_default acl _nop***

   Setting default action of acl

   action: _nop

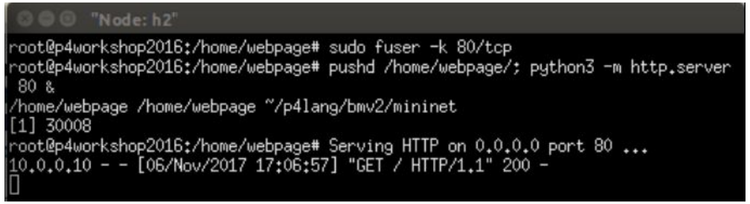
9. Testing ***all traffic allowed*** using the ***curl*** command:

   In h2:

   ```
   sudo fuser -k 80/tcp
   pushd /home/webpage/; python3 -m http.server 80 &
   ```

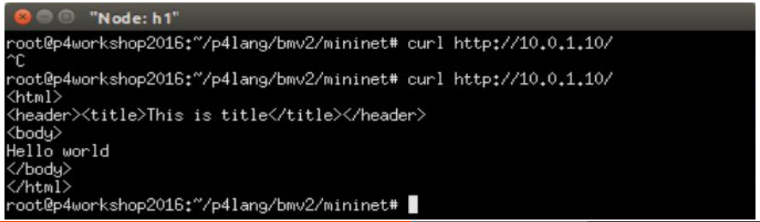
   In h1

   ```
   curl http://10.0.1.10
   ```



"Node: h2"

```
root@p4workshop2016:/home/webpage# sudo fuser -k 80/tcp
root@p4workshop2016:/home/webpage# pushd /home/webpage/; python3 -m http.server
 80 &
/home/webpage /home/webpage ~/p4lang/bmv2/mininet
[1] 30008
root@p4workshop2016:/home/webpage# Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.10 - - [06/Nov/2017 17:06:57] "GET / HTTP/1.1" 200 -
```



"Node: h1"

```
root@p4workshop2016:~/p4lang/bmv2/mininet# curl http://10.0.1.10/
^C
root@p4workshop2016:~/p4lang/bmv2/mininet# curl http://10.0.1.10/
<html>
<header><title>This is title</title></header>
<body>
Hello world
</body>
</html>
root@p4workshop2016:~/p4lang/bmv2/mininet#
```

P4 Dependency Graph Generator

P4 Dependency Graph Generator

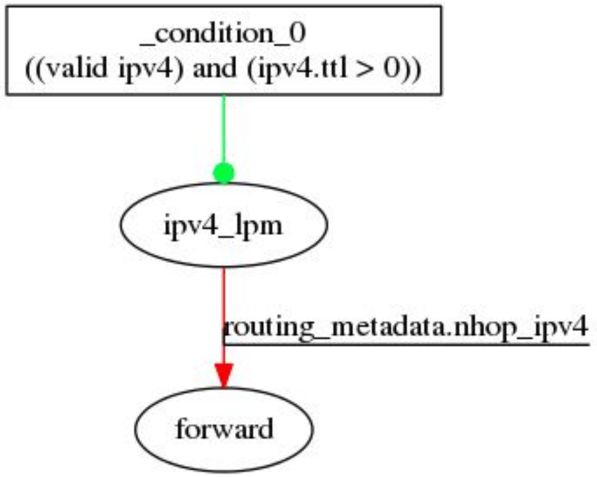
➔ In complex P4 programs it is difficult to comprehend the table relationships. The p4-graphs utility generates dependency graphs for a P4 program using graphviz.

```
cd ~/p4lang/bmv2/targets/simple_router/
p4-graphs simple_router.p4
```

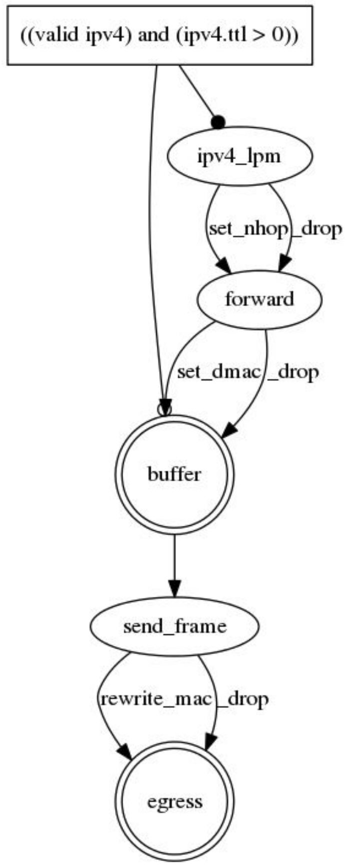
➔ P4-graphs generates different images for different P4 abstractions as mention below:

- ➔ **simple_router.ingress.tables_dep.png**
- ➔ **simple_router.tables.png**
- ➔ **simple_router.parser.png**
- ➔ simple_router.ingress.tables_dep.dot
- ➔ simple_router.tables.dot
- ➔ simple_router.parser.dot

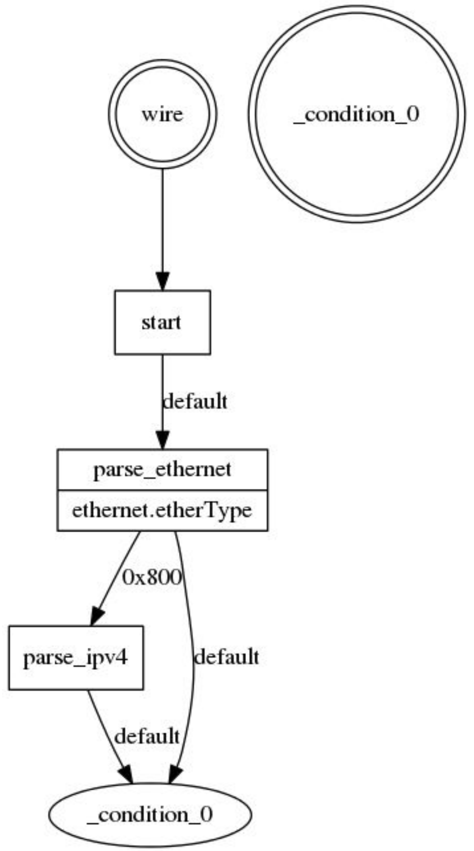
P4 Dependency Graph Generator



Ingress Table



Table



Parser

P4 Dependency Graph Generator

➔ Run the P4 Dependency Graph Generator on the Simple Router

```
cd ~/p4lang/bmv2/targets/simple_router/
p4-graphs simple_router.p4
```

➔ Now run the P4 Dependency Graph Generator on Simple Router + ACL

```
cd ~/p4lang/bmv2/targets/simple_router_acl/

p4-graphs simple_router_acl.p4
```

Q5: Explain the differences between the graphs generated by simple_router.p4 and simple_router_acl.p4:

simple_router.ingress.tables_dep.png <-> simple_router_acl.ingress.tables_dep.png
simple_router.tables.png <-> simple_router_acl.tables.png
simple_router.parser.png <-> simple_router_acl.parser.png

(OPTIONAL) Router with ACL+

Further experimenting with the ACL

simple_router + ACL

Launch the CLI tool to populate the tables using the runtime_CLI script.

```
cd ~/p4lang/bmv2/targets/simple_router_acl
sudo p4c-bmv2 --json simple_router_acl.json simple_router_acl.p4
sudo ./simple_router -i 1@veth1 -i 2@veth2 simple_router_acl.json
```

In another terminal, run the below commands to install state into the tables.

```
cd ~/p4lang/bmv2/targets/simple_router_acl
```

```
./runtime_CLI < commands.txt
```

```
table_set_default send_frame _drop
table_set_default forward _drop
table_set_default ipv4_lpm _drop
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
table_add forward set_dmac 10.0.0.10 => 00:04:00:00:00:00
table_add forward set_dmac 10.0.1.10 => 00:04:00:00:00:01
table_add ipv4_lpm set_nhop 10.0.0.10/32 => 10.0.0.10 1
table_add ipv4_lpm set_nhop 10.0.1.10/32 => 10.0.1.10 2
```

simple_router_acl

Optional Q6: Which command(s) would you need to run to block traffic based on:
(i) a specific TCP destination port? (ii) a specific destination IP subnet? (iii) a specific source MAC address?

Optional Q7: Do you need to modify the P4 program of simple_router_acl?

Tips:

Use nmap to test connectivity to a specific port

```
nmap --source-port <# port>  <target IP>   -p  <target port>
```

Use it to add a rule to drop traffic coming from a specific port:

```
runtime_CLI> table_add acl _drop <# port>  <# port>  =>
```

Use nc to open a specific port :

```
nc -l <# port>
```

References

[1] https://github.com/p4lang/

[2] https://github.com/p4lang/tutorials/

[3] http://p4.org

Credits

P4.org

INTRIG / Prof. Christian Rothenberg

Fabricio Rodriguez

Javier Richard Quinto Ancieta

P Gyanesh Patra

Celso Henrique Cesila

Daniel Feferman