Overview

This lab describes match-action tables and how to define them in a P4 program. It then explains the different types of matching that can be performed on keys. The lab further shows how to track the misses/hits of a table key while a packet is received on the switch.

Objectives

By the end of this lab, students should be able to:

1. Understand what match-action tables are used for.
2. Describe the basic syntax of a match-action table.
3. Implement a simple table in a P4.
4. Trace a table's misses/hits when a packet enters to the switch.

Lab settings

The information in Table 1 provides the credentials of the machine containing Mininet.

Table 1. Credentials to access Client machine.

Device	Account	Password
Client	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Defining a table with LPM matching.
4. Section 4: Loading the P4 program.
5. Section 5: Configuring switch s1.
6. Section 6: Testing and verifying the P4 program.

1 Introduction

1.1 Longest prefix match (LPM)

Table 2 is an example of a match-action table that uses LPM. Assume that the key is formed with the destination IP address. If an incoming packet has the destination IP address 172.168.3.5, two entries match. The first entry matches because the first 29 bits in the entry are the same as the first 29 bits of the destination IP. The second entry also matches because the first 16 bits in the entry are the same as the first 16 bits of the destination IP. The LPM algorithm will select 172.168.3.0/29 because of the longest prefix preference.

Table 2. Match-action table using LPM as the lookup algorithm.

Key	Action	Action data
172.168.3.0/29	forward	port 1, macAddr=00:00:00:00:00:01
172.168.0.0/16	forward	port 2, macAddr=00:00:00:00:00:02
default	drop	

Figure 1 shows the ingress control block portion of a P4 program. Two actions are defined, `drop` and `forward`. The `drop` action (lines 5 - 7) invokes the `mark_to_drop` primitive, causing the packet to be dropped at the end of the ingress processing. The `forward` action (lines 8 - 11) accepts as input (action data) the port and the destination MAC address. These parameters are inserted by the control plane and updated in the packet during the ingress processing.

In line 9, the P4 program assigns the new egress port to the `standard_metadata` egress port field (i.e., the field that the traffic manager looks at to determine which port the packet must be sent to). Line 10 assigns the destination MAC address passed as parameter to the packet's new destination address.

Lines 12-22 implement a table named `ipv4_lpm`. The table is matching against the destination IP address using the LPM type. The actions associated with the table are `forward` and `drop`. The default action is invoked when there is a miss. The maximum number of entries is defined by the programmer (i.e., 1024 entries, see line 20).

The control block starts executing from the apply statement (see lines 23-27) which contains the control logic. In this program, the `ipv4_lpm` table is activated in case the incoming packet has a valid IPv4 header.

```
/**********************INGRESS PROCESSING**********************/
control MyIngress(inout headers hdr,
                        inout metadata meta,
                        inout standard_metadata_t standard_metadata){
    action drop(){
        mark_to_drop(standard_metadata);
    }
    action forward(egressSpec_t port, macAddr_t dstAddr) {
        standard_metadata.egressSpec = port;
        hdr.ethernet.dstAddr = dstAddr;
    }
    table ipv4_lpm {
        key = {
            hdr.ipv4.dstAddr:lpm;
        }
        actions = {
            forward;
            drop;
        }
        size = 1024;
        default_action = drop();
    }
    apply {
        if (hdr.ipv4.isValid()){
            ipv4_lpm.apply();
        }
    }
}
```

Figure 1. Ingress control block portion of a P4 program. The code implements a match-action table with LPM lookup.

2 Lab topology

Let's get started by opening a simple Mininet topology using MiniEdit. The topology comprises three end hosts and one P4 programmable switch.

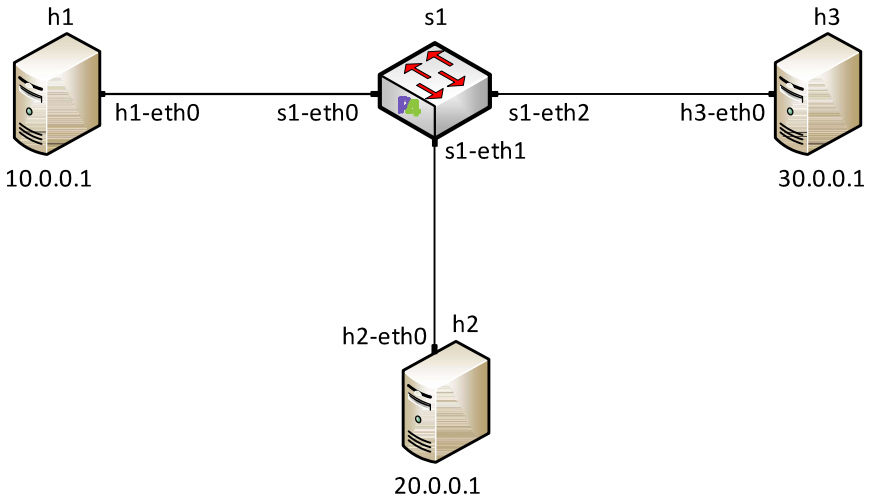

Figure 2. Lab topology.

Step 1. A shortcut to MiniEdit is located on the machine's desktop. Start MiniEdit by double-clicking on MiniEdit's shortcut. When prompted for a password, type `password`.

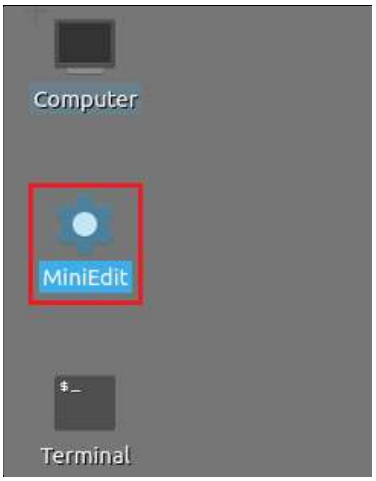

Figure 3. MiniEdit shortcut.

Step 2. In the MiniEdit application, load the topology by clicking on *File* then *Open*. Navigate to the *lab6* folder and search for the topology file called *lab6.mn* and click on Open. A new topology will be loaded to MiniEdit.

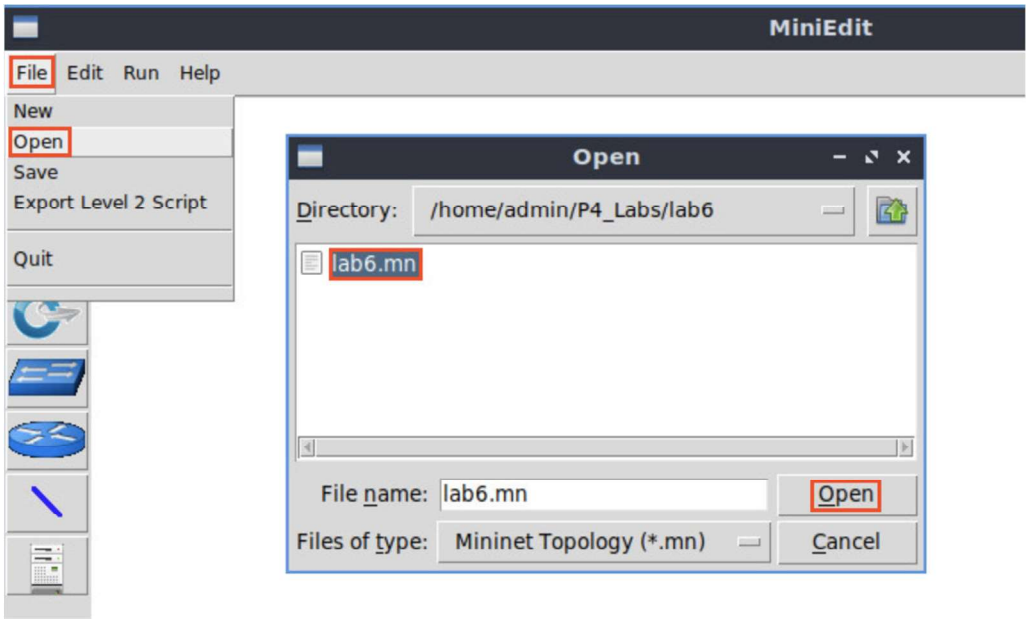

Figure 4. MiniEdit's *Open* dialog.

Step 3. The network must be started. Click on the *Run* button located at the bottom left of MiniEdit's window to start the emulation.

Figure 5. Running the emulation.

2.1 Starting end hosts

Step 1. Right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

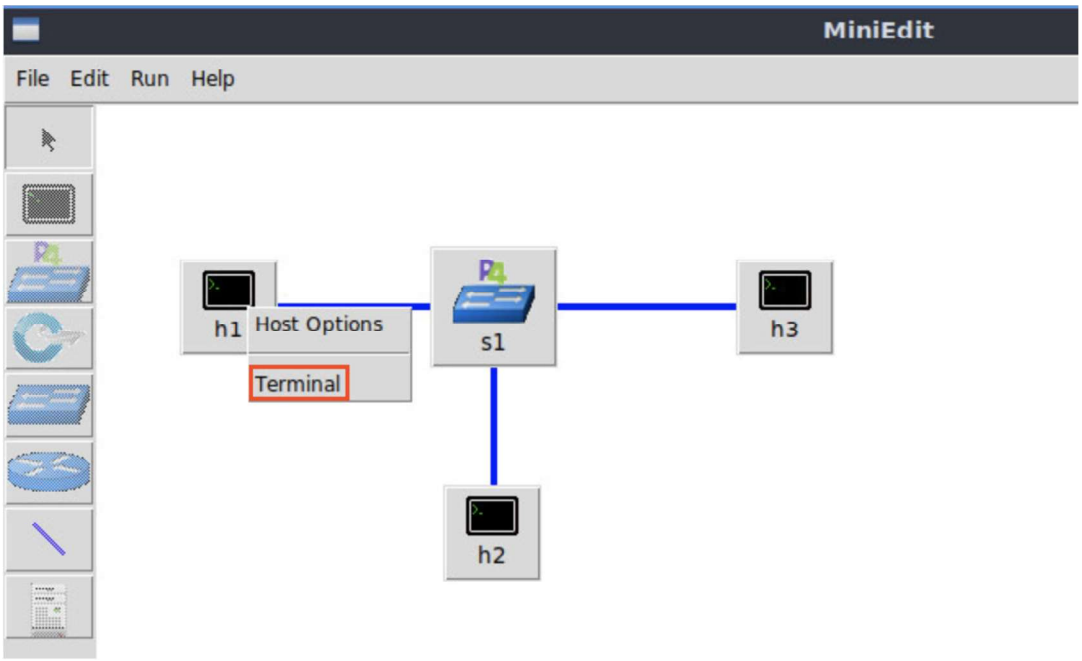

Figure 6. Opening a terminal on host h1.

Step 2. Verify the interfaces' configuration by issuing the following command.

```
ifconfig
```

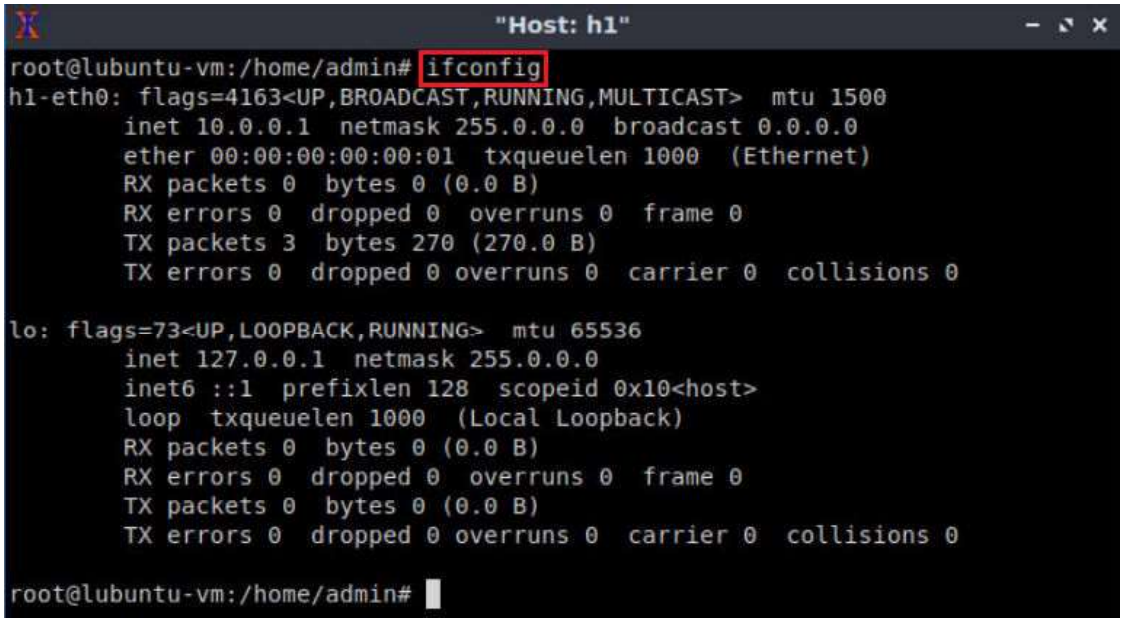

Figure 7. Verifying the configuration host h1 interfaces.

3 Defining a table with LPM matching

This section demonstrates how to implement a simple table in P4 that uses LPM matching on the packet's destination IP address. When there is a match, the switch forwards the packet from a certain port. Otherwise, the switch drops the packet.

3.1 Loading the programming environment

Step 1. Launch a Linux terminal by double-clicking on the icon located on the desktop.

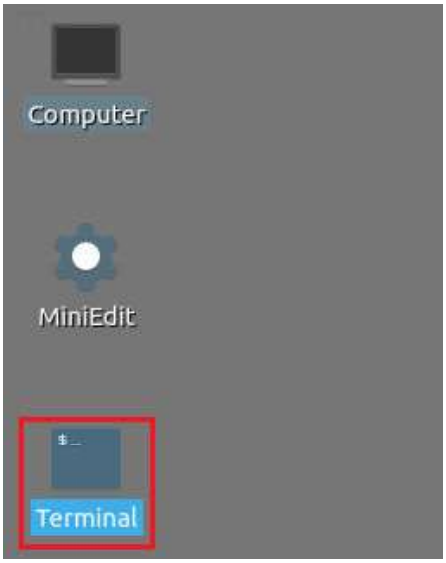

Figure 8. Shortcut to open a Linux terminal.

Step 2. In the terminal, type the command below. This command launches the Visual Studio Code (VS Code) and opens the directory where the P4 program for this lab is located.

```
code ~/P4_Labs/lab6
```

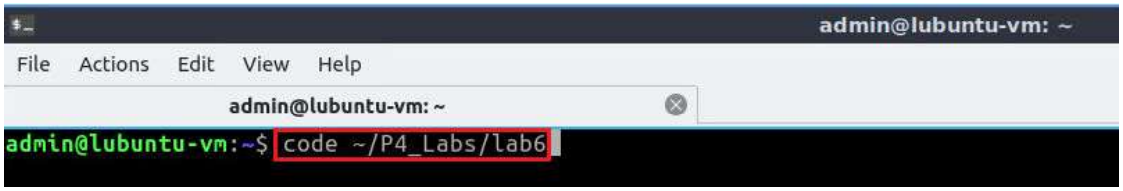

Figure 9. Launching the editor and opening the lab6 directory.

3.2 Programming the ingress block

Step 1. Click on the *ingress.p4* file to display the contents of the file. Use the file explorer on the left-hand side of the screen to locate the file.

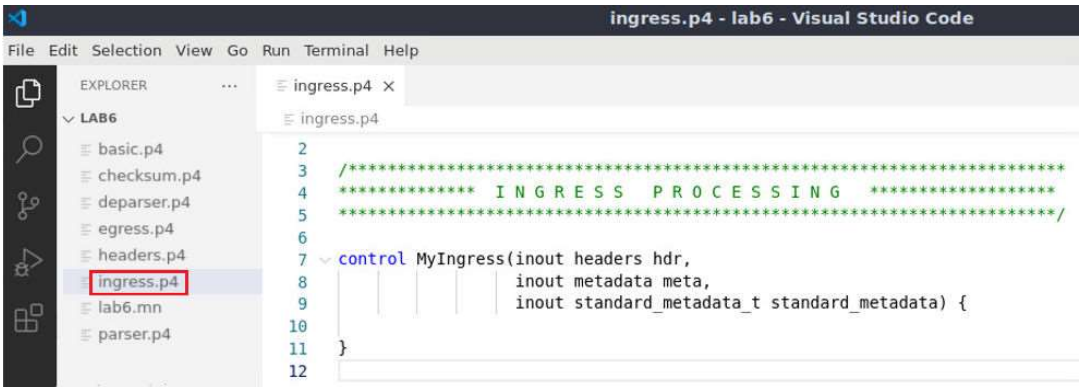

Figure 10. Opening the ingress processing block.

We can see that the *ingress.p4* declares a control block named `MyIngress`. Inside the block, we will define a table `ipv4_host` that is used to match on the destination IP address and forward/drop the packet. There are two actions that will be invoked in this table: `forward` and `drop`.

- `forward`: This action defines a set of basic operations on a packet header. Such operations are defined as follows: 1) Updating the egress port so the packet is forwarded to its destination through the correct port. 2) Updating the source MAC address with the packet's previous destination MAC address. 3) Changing the destination MAC address of the packet with the one corresponding to the next hop. 4) Decrementing the time-to-live (TTL) field in the IPv4 header.
- `drop`: this action will be used to drop the packet.

Step 2. The following code fragment describes the behavior of the `forward` action. Insert the code below inside the *MyIngress* control block.

```
action forward(macAddr_t dstAddr, egressSpec_t port){
    standard_metadata.egress_spec = port;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}
```

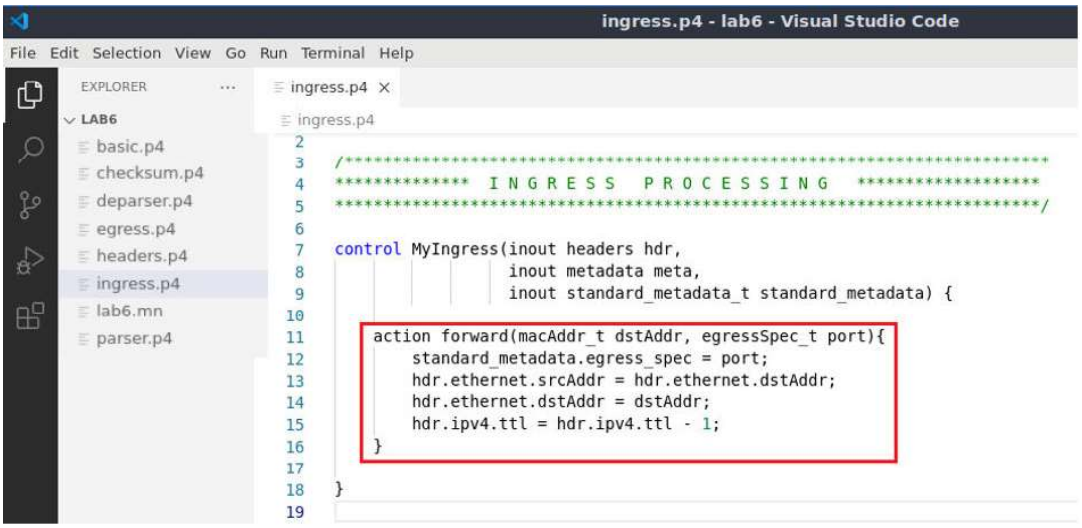

Figure 11. Defining the `forward` action.

The action `forward` accepts as parameters the next hop's MAC address (i.e., `macAddr_t dstAddr`) and the port number (i.e., `egressSpec_t port`) to be used by the switch to forward the packet. Note that `egressSpec_t` is just a typedef that corresponds to `bit<9>` and `macAddr_t` is a typedef that corresponds to `bit<48>`. These types are defined in the *headers.p4* file.

The `standard_metadata` is an instance of the `standard_metadata_t` struct provided by the V1Model[1]. This struct contains intrinsic metadata used in packet processing and in more advanced features. For example, to determine the port on which a packet arrives, we can use the `ingress_port` field in the `standard_metadata` (i.e., `standard_metadata.ingress_port`). Similarly, the egress port `egress_spec` field of the `standard_metadata` defines the egress port. Line 12 shows how to assign the egress port to forward an incoming packet to its destination.

To modify header fields inside the packet, we refer to the field name based on where it exists inside the headers. Recall that the names of the headers and the fields are defined by the programmer. The file *headers.p4* defines the program's headers. Line 13 shows how we are assigning the destination MAC address of the packet (i.e., `hdr.ethernet.dstAddr`) to be the new source MAC of the packet (i.e., `hdr.ethernet.srcAddr`). Line 14 shows how we are assigning the destination MAC address which is provided as a parameter (assigned later in the control plane) to be the new destination MAC of the packet.

It is possible in P4 to perform basic arithmetic operations on header fields and other variables. In line 15, we are decrementing the TTL value of the header field.

Step 3. Define the drop action by appending the following code into the *MyIngress* control block.

```
action drop() {
    mark_to_drop(standard_metadata);
}
```

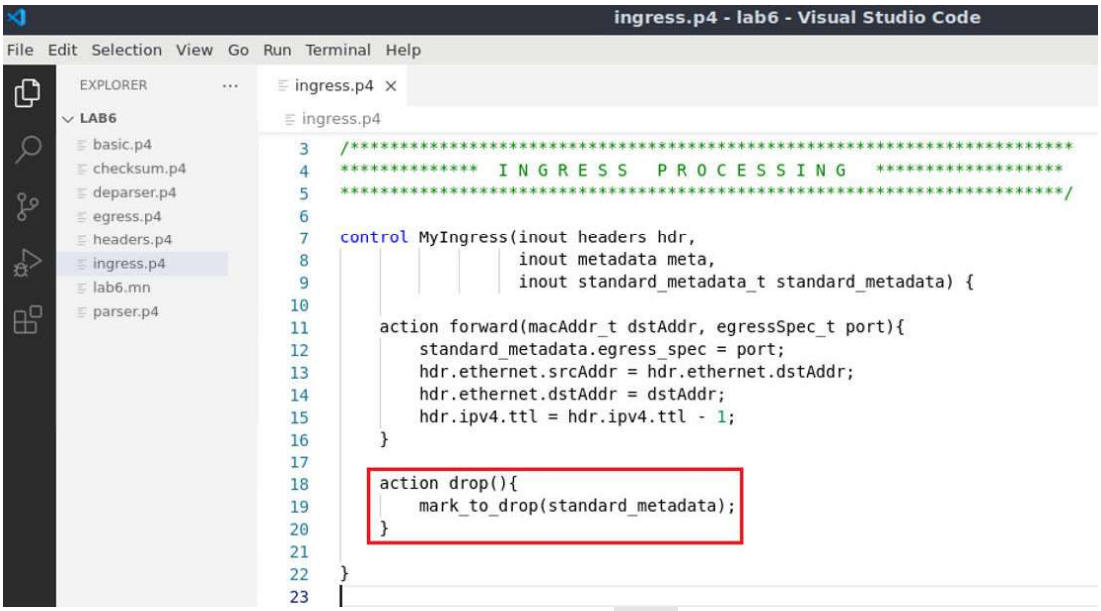

Figure 12. Defining the `drop` action.

Step 4. Define an exact match table by appending the following piece of code.

```
table ipv4_exact {
    key = {
        hdr.ipv4.dstAddr: exact;
    }
    actions = {
        forward;
        drop;
    }
    size = 1024;
    default_action = drop();
}
```

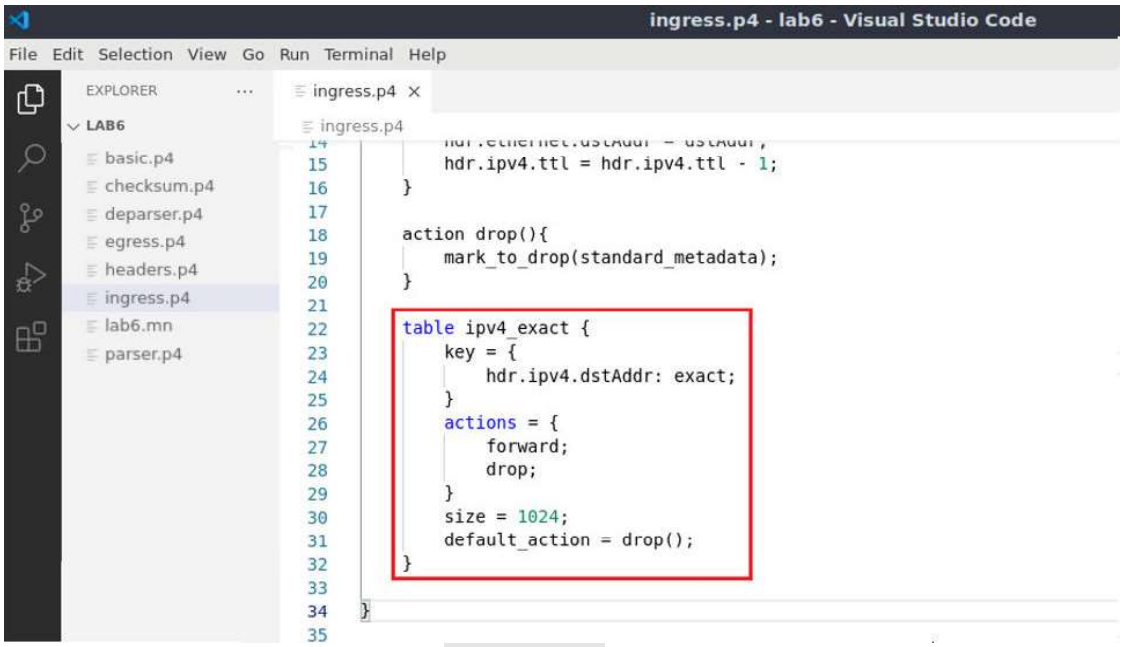

Figure 13. Defining the table `ipv4_exact` implementing exact match lookup.

Step 5. Now we will define a table that performs a LPM on the destination IP address of the packet. The table will be invoking the forward and the drop actions, and hence, those actions will be listed inside the table definition.

```
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        forward;
        drop;
    }
    size = 1024;
    default_action = drop();
}
```

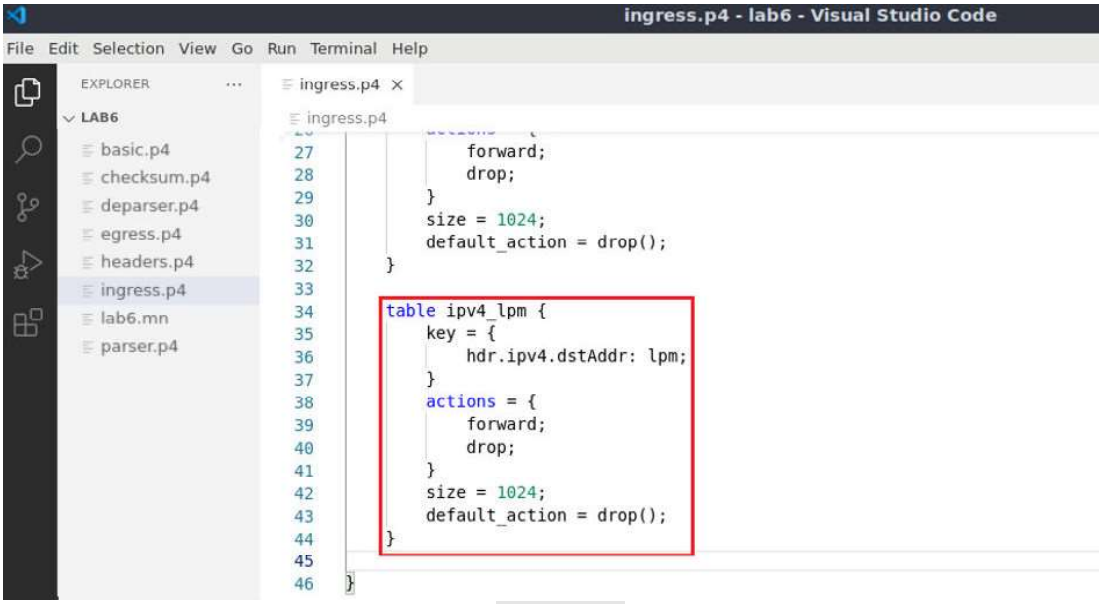

Figure 14. Defining the table `ipv4_lpm` implementing LPM lookup.

Step 6. Add the following code at the end of the *MyIngress* block. The `apply` block defines the sequential flow of packet processing. It is required in every control block, otherwise the program will not compile. It describes the sequence of tables to be invoked, in addition to other packet processing instructions.

```
apply {
    if(hdr.ipv4.isValid()) {
        if(ipv4_exact.apply().miss) {
            ipv4_lpm.apply();
        }
    }
}
```

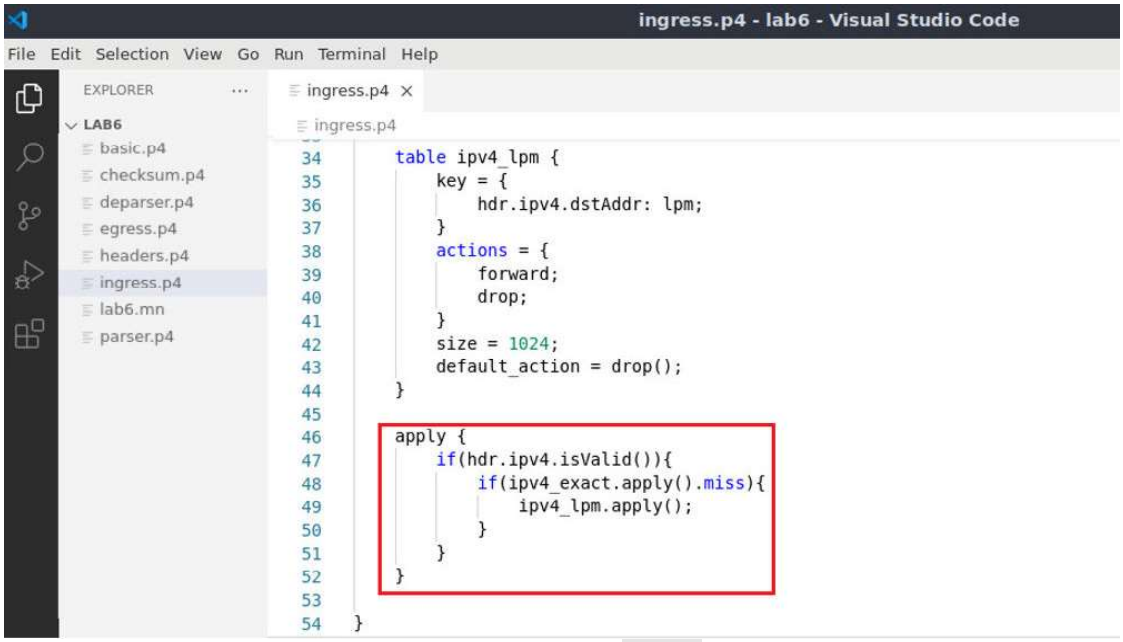

Figure 15. Defining the `apply` block.

The logic of the code above is as follows: if the packet has an IPv4 header, apply the `ipv4_exact` table which performs an exact match lookup on the destination IP address. If there is no *hit* (i.e., the table does not contain a rule that corresponds to this IPv4 address, denoted by the *miss* keyword), apply the `ipv4_lpm` table, which matches the destination IP address of the packet against a network address.

Step 7. Save the changes to the file by pressing `Ctrl + s`.

4 Loading the P4 program

4.1 Compiling and loading the P4 program to switch s1

Step 1. Issue the following command in the terminal panel inside the VS Code to compile the program.

```
p4c basic.p4
```

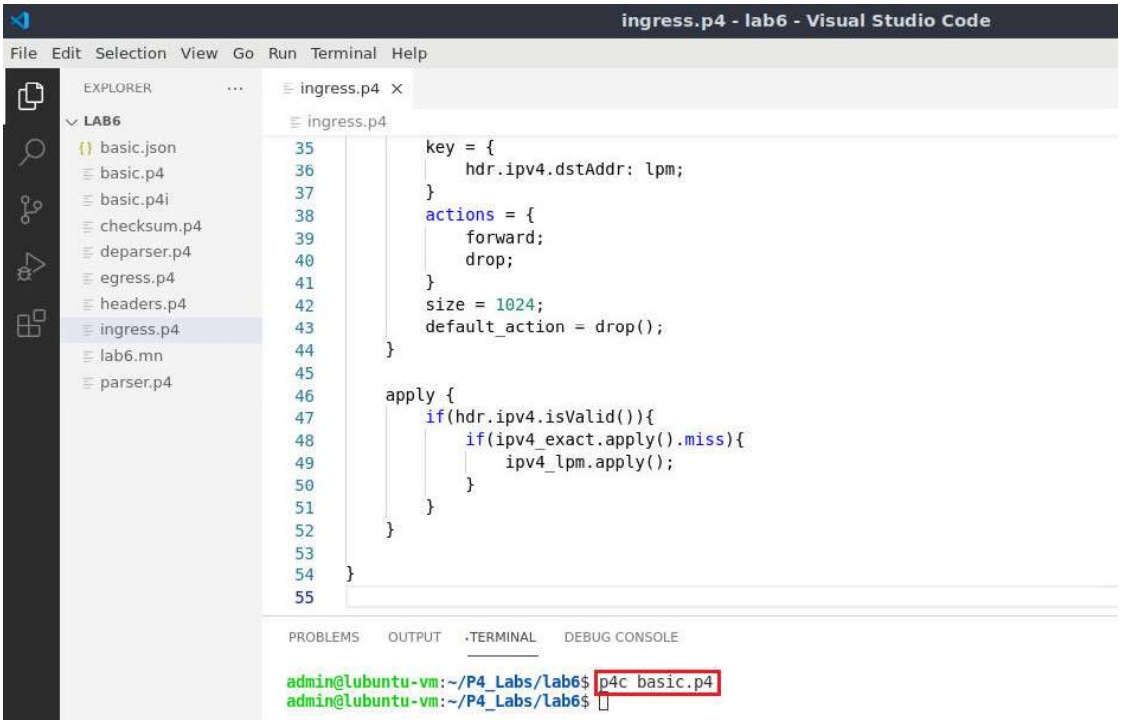

Figure 16. Compiling a P4 program.

Step 2. Type the command below in the terminal panel to push the *basic.json* file to the switch s1's filesystem. The script accepts as input the JSON output of the p4c compiler, and the target switch name. If asked for a password, type the password `password`.

```
push_to_switch basic.json s1
```

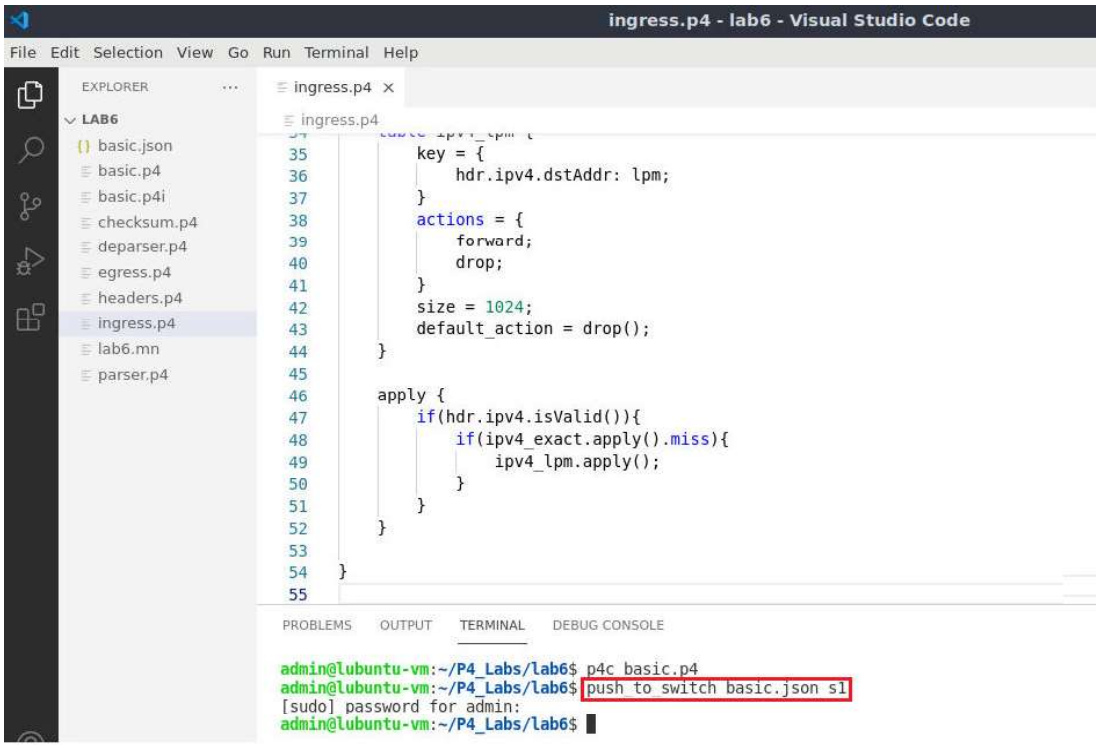

Figure 17. Pushing the *basic.json* file to switch s1.

4.2 Verifying the configuration

Step 1. Click on the MinEdit tab in the start bar to maximize the window.

Figure 18. Maximizing the MiniEdit window.

Step 2. Right-click on the P4 switch icon in MiniEdit and select *Terminal*.

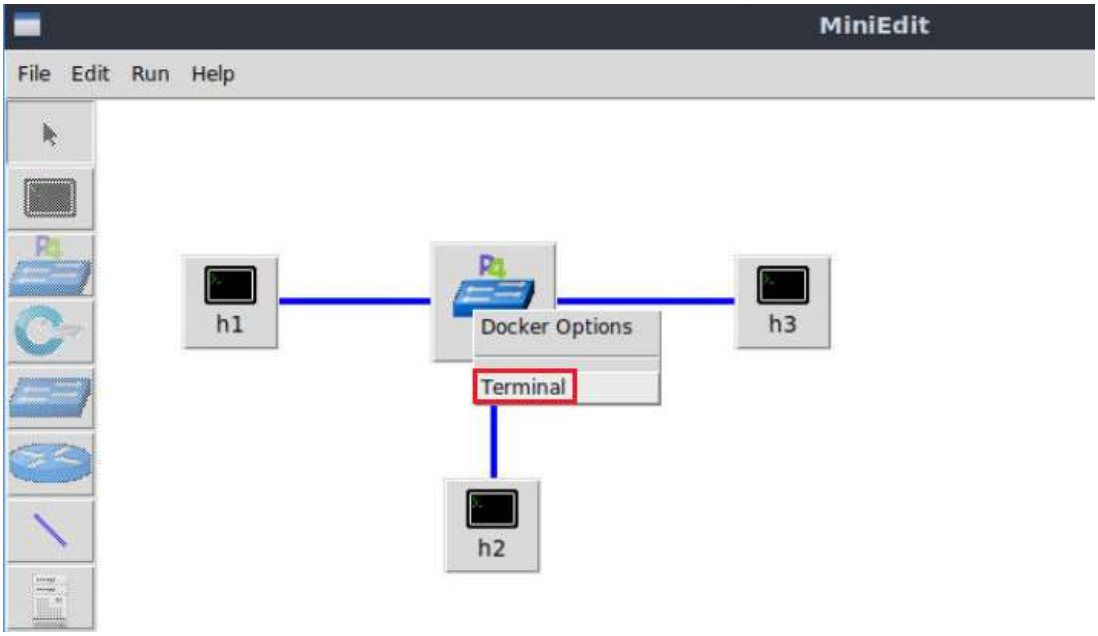

Figure 19. Opening switch s1 terminal.

Note that the switch is running on an Ubuntu image started on a Docker container. Thus, you will be able to execute any Linux command on the switch terminal.

Step 3. Issue the following command on switch s1 terminal to inspect the content of the current folder.

```
ls
```

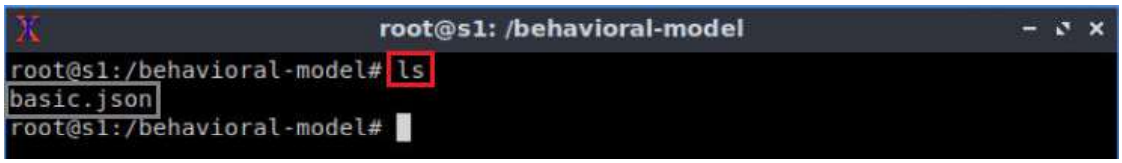

Figure 20. Displaying the content of the current directory in the switch s1.

We can see that the switch contains the *basic.json* file that was pushed previously after compiling the P4 program.

5 Configuring switch s1

5.1 Mapping P4 program's ports

Step 1. Issue the following command on switch s1.

```
ifconfig
```

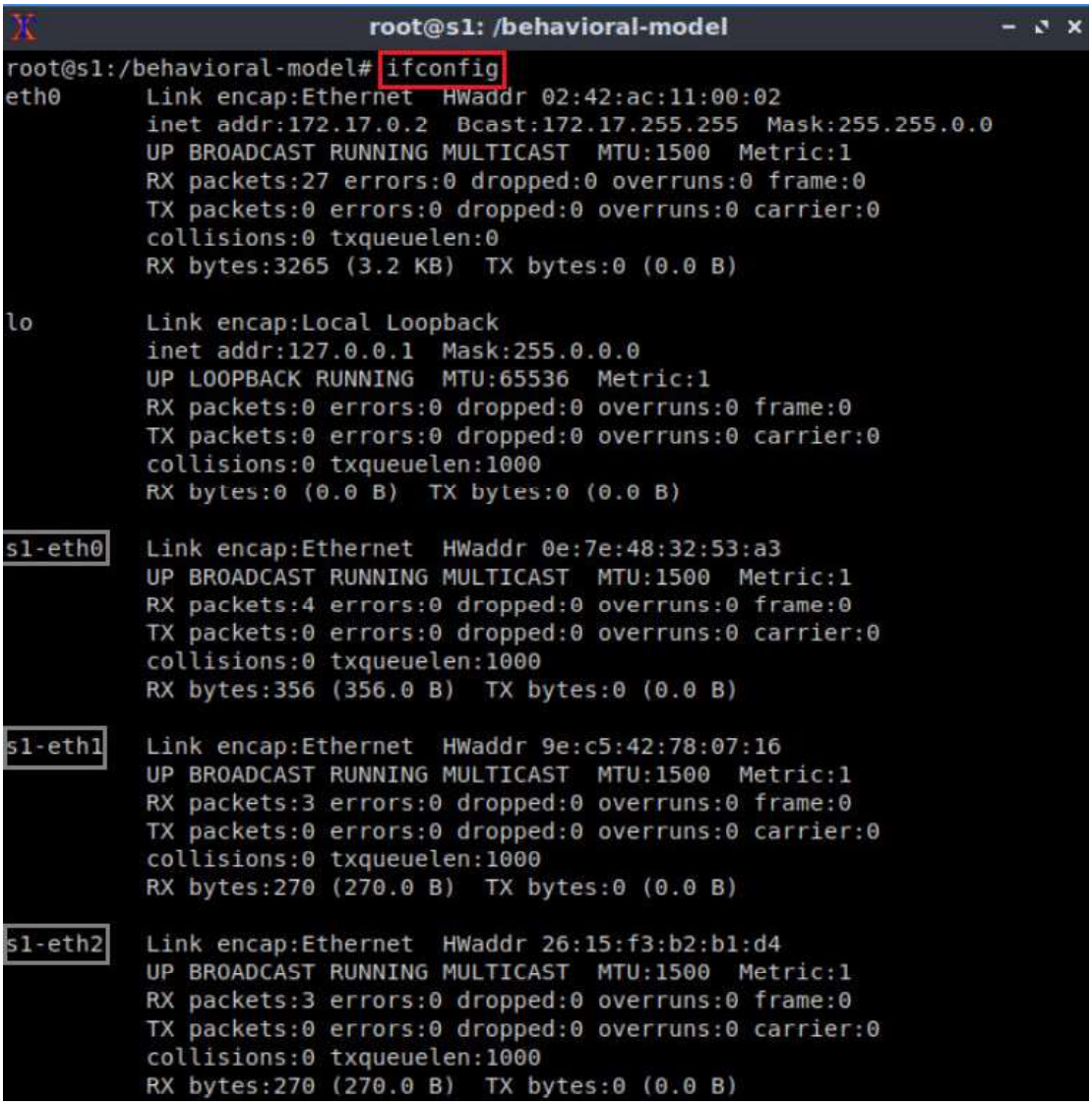

Figure 21. Displaying switch s1 interfaces.

The output displays switch s1 interfaces (i.e., *s1-eth0*, *s1-eth1* and *s1-eth2)*. The interface *s1-eth0* on the switch s1 connects to the host h1. The interface *s1-eth1* on the switch s1 connects to the host h2 and *s2-eth2* is connected to host h3.

Step 2. Start the switch daemon and map the logical interfaces (i.e., ports) to the switch's interfaces by issuing the following command. The `--nanolog` parameter is used to instruct the switch daemon to provide the switch's logs.

```
simple_switch -i 0@s1-eth0 -i 1@s1-eth1 -i 2@s1-eth2 --nanolog ipc:///tmp/bm-log.ipc  basic.json &
```

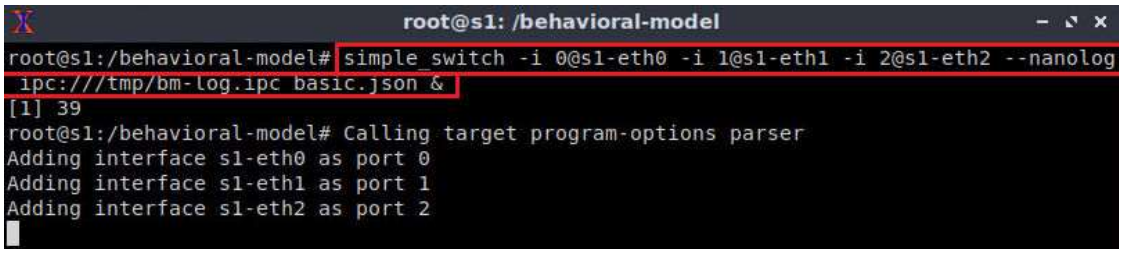

Figure 22. Starting the switch daemon and mapping the logical interfaces to Linux interfaces.

5.2 Loading the rules to the switch

Step 1. In switch s1 terminal, press *Enter* to return the CLI.

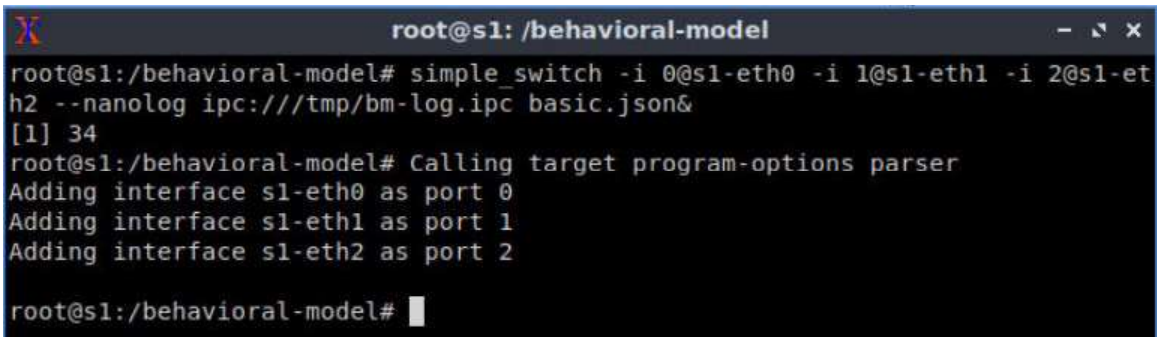

Figure 23. Returning to switch s1 CLI.

Step 2. Push the table entries to the switch by typing the following command.

```
simple_switch_CLI < ~/lab6/rules.cmd
```

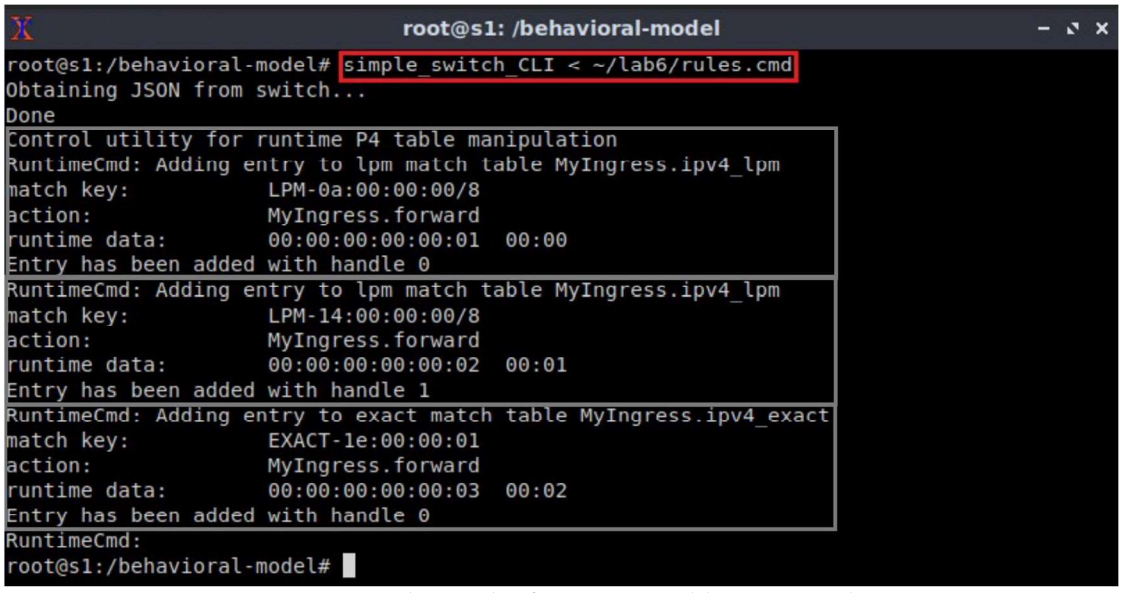

Figure 24. Populating the forwarding table into switch s1.

The script above pushes the rules to the switch daemon. We can see that we added three entries to the `ipv4_exact` and `ipv4_lpm` tables.

- The key of the first entry is 10.0.0.0/8 (which translates to 0a:00:00:00 in hexadecimal as shown in the figure above, next to match key) and its action is forward. This entry is added to the `ipv4_lpm` table. The action parameters or runtime data are 00:00:00:00:00:01 for the destination MAC (i.e., host h1's MAC address) and 0 for the output port (i.e., the port facing host h1).
- The key of the second entry is 20.0.0.0/8 (which translates to 14:00:00:00 in hexadecimal as shown in the figure above, next to match key) and its action is forward. This entry is added to the `ipv4_lpm` table. The action parameter or runtime data are 00:00:00:00:00:02 for the destination MAC (i.e., host h2's MAC address) and 1 for the output port (i.e., the port facing host h2).
- The key of the third entry is 30.0.0.1 (which translates to 1e:00:00:01 in hexadecimal as shown in the figure above, next to match key) and its action is forward. This entry is added to the `ipv4_host` table. The action values are 00:00:00:00:00:03 for the destination MAC (i.e., host h3's MAC address) and 2 for the output port (i.e., the port facing host h3).

6 Testing and verifying the P4 program

Step 1. Type the following command to display the switch logs.

```
nanomsg_client.py
```

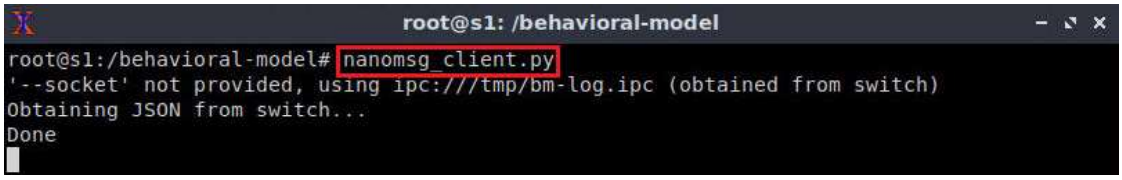

Figure 25. Displaying switch s1 logs.

Step 2. On host h2's terminal, type the command the command below so that the host starts listening for packets.

```
./recv.py
```

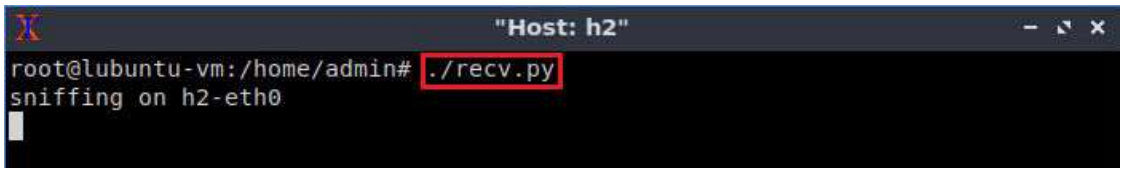

Figure 26. Listening for incoming packets in host h2.

Step 3. On host h1's terminal, type the following command to send a message to host h2. The output will show the Ethernet, IP and TCP header fields and their values. The payload is `HelloWorld`.

```
./send.py 20.0.0.1 HelloWorld
```

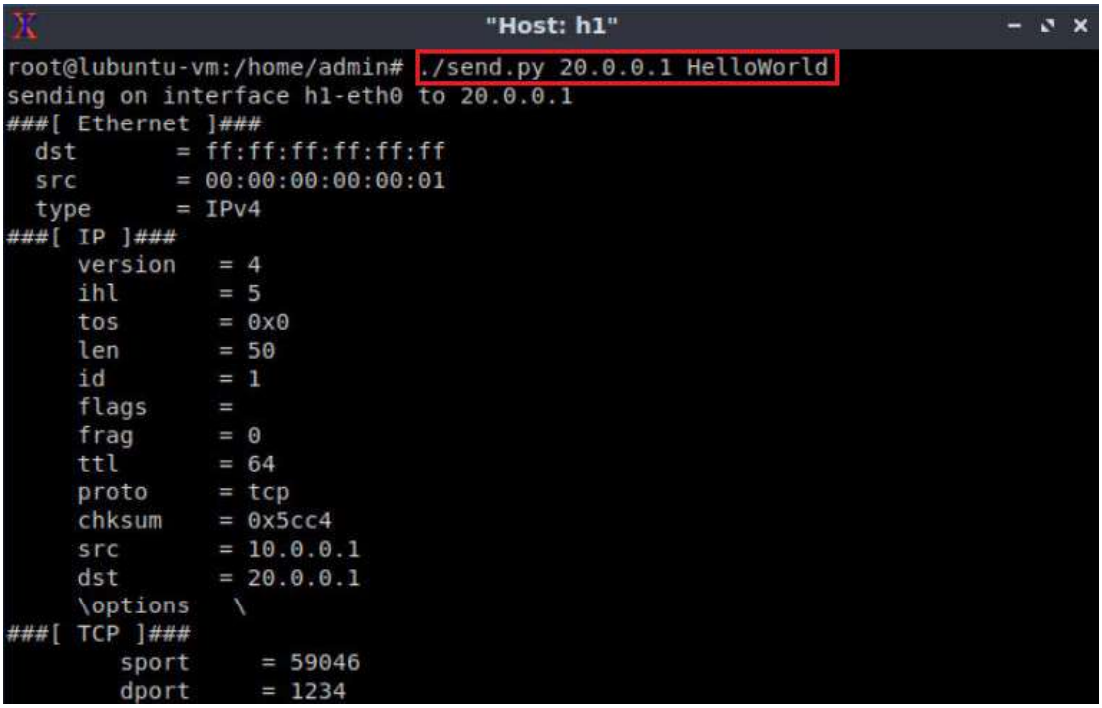

Figure 27. Sending a test packet from host h1 to host h2.

Step 4. Inspect the logs on switch s1 terminal.

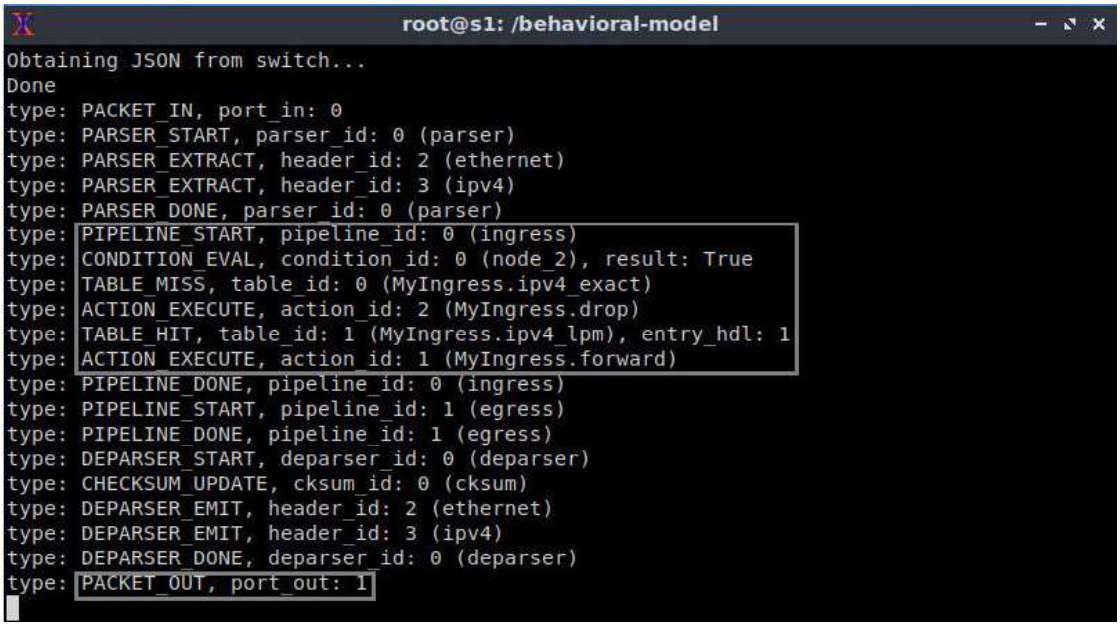

Figure 28. Inspecting the logs in switch s1.

Results show that there is a miss in the `ipv4_exact` table, but there is a hit on the `ipv4_lpm` table. Then, the packet is forwarded through port 1, which is connected to host h2. This behavior corresponds to the logic described by the `apply` block in the ingress processing.

Step 5. Verify that the packet was received on host h2. Notice that the TTL was decremented.

Step 6. On host h1's terminal, type the following command to send a message to host h3.

```
./send.py 30.0.0.1 HelloWorld
```

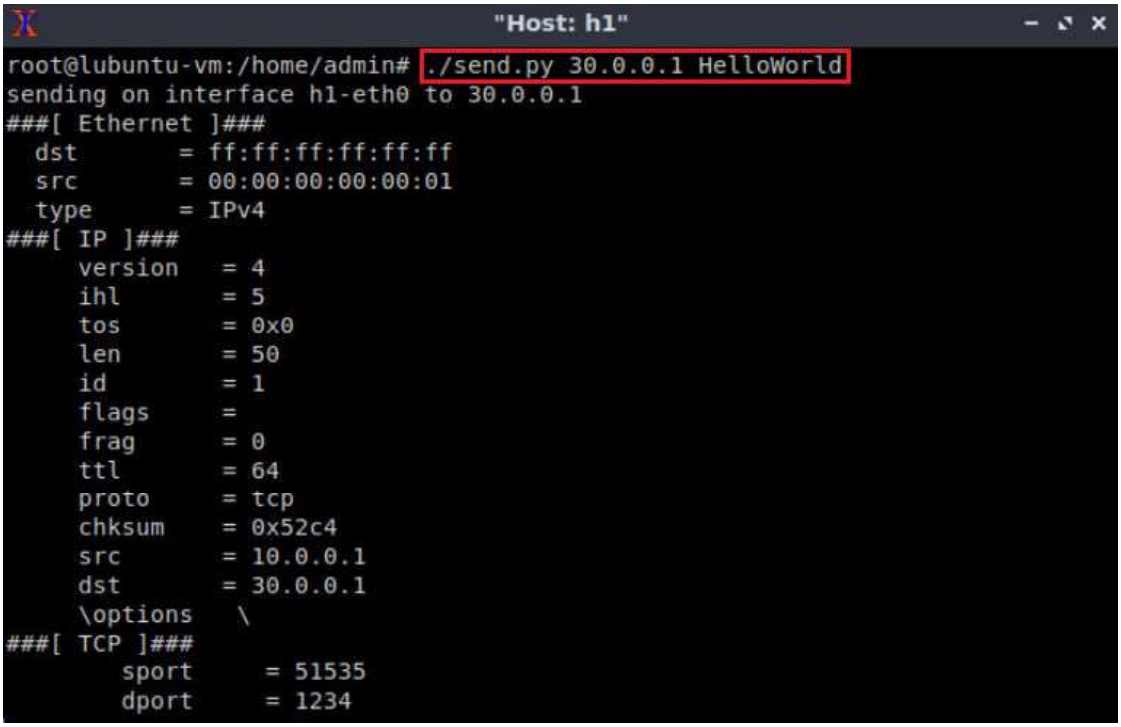

Figure 29. Sending a test packet from host h1 to host h3.

Step 7. Inspect the logs on switch s1 terminal.

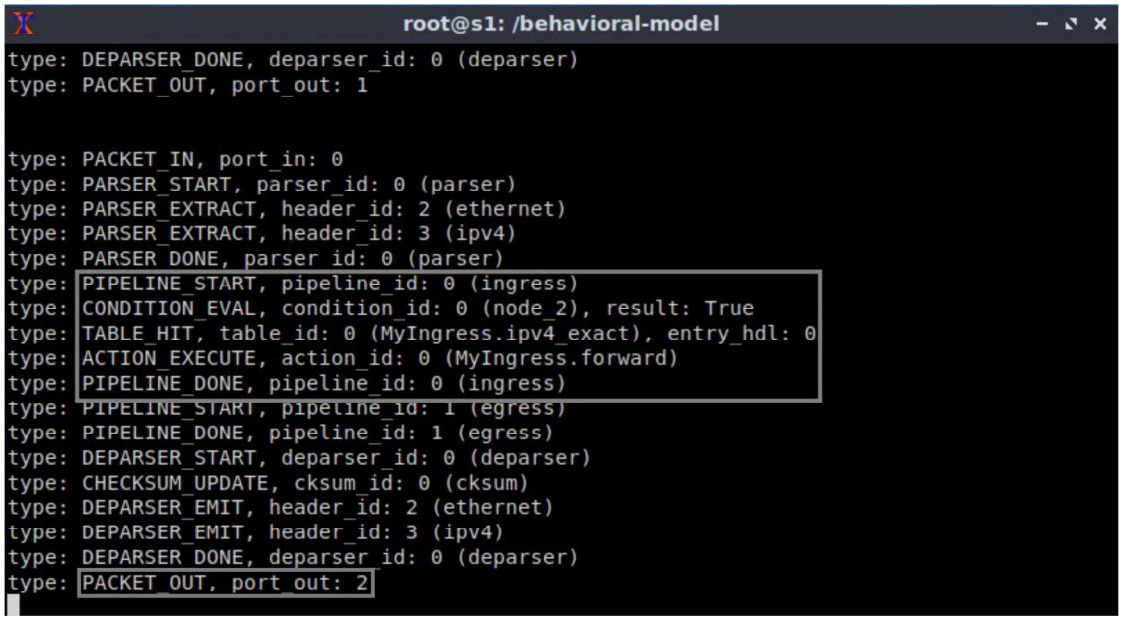

Figure 30. Inspecting the logs in switch s1.

The figure above shows that there is a hit in the `ipv4_exact` table. Then, the packet is forwarded through port 2, which is connected to host h3.

This concludes lab 6. Stop the emulation and then exit out of MiniEdit.

References

1. P4 Language Tutorial. [Online]. Available: https://tinyurl.com/2p9cen9e.
2. Mininet walkthrough. [Online]. Available: http://Mininet.org.
3. M. Peuster, J. Kampmeyer, H. Karl. "*Containernet 2.0: A rapid prototyping platform for hybrid service function chains.*" 4th IEEE Conference on Network Softwarization and Workshops (NetSoft). 2018.
4. R. Cziva. "*ESnet tutorial - P4 deep dive, slide 28*." [Online]. Available: https://tinyurl.com/rruscv3.
5. P4lang/behavioral-model github repository*. "The BMv2 simple switch target*." [Online]. Available: https://tinyurl.com/vrasamm.