Advanced Networking 2018

Lab #7: P4
Total points: 10 pts

Assignment
Lab date: March 23, 2018
Submission date: 11:59PM March 25, 2018 CET

Author: Joseph Hill
Email: j.d.hill@uva.nl
University of Amsterdam

Introduction

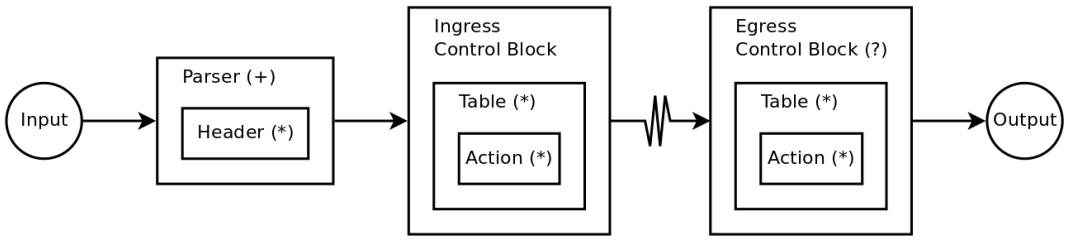
This lab will explorer the features of P4. P4 is a program language used to describe the behavior of the data plane of a forwarding device. It is protocol independent meaning that there is no predetermined definition of the structure of a packet or how it should be handled. With P4 the term packet is used generically and could refer to a Ethernet frame, IP packet or a unit of data in a user defined protocol. There are actually two releases of the P4 language. We will be using $P4_{14}$ which is the older release of the language. $P4_{16}$ is the current release of the language however it is not as well supported by devices and has not yet implemented all of the features of $P4_{14}$. To run the code we will be using the software switch Behavioral Model (BMv2) with the simple_switch architecture.

You will need to refer to $P4_{14}$ Language Specification throughout the lab.
https://p4.org/p4-spec/p4-14/v1.0.4/tex/p4.pdf

The Scapy documentation for crafting packets may also be useful.
http://scapy.readthedocs.io/en/stable/usage.html

The following is a brief description of the P4 code blocks we will be dealing with. See section 1.3 of the P4 specification for more information.

- Header (type definition and declaration): Defines the structure of a header, the order and size of the fields.
- Parser: Determines which headers are extracted from a packet and in what order.
- Action: Modifies a packet or metadata, using primitive actions.
- Table: Matches against fields or metadata and executes actions.
- Control: Determines which tables are applied and in what order.



(?) None or One, (+) One or More, (*) Zero or More

The flow of a P4 program begins with a parser block named start. This parser block is required and additional ones may be defined. Each parser may reference one or more headers before passing control on to another parser block or to the ingress control block. The ingress control block may reference one or more tables and each table may reference one or more actions. The optional egress control block, if it exists, will automatically follow the ingress control block. It is functionally equivalent to the ingress control block with a few exceptions. It exists to process a packet after and egress port has been decided.

Preparation

Accessing Virtual Machines

Virtual machines with the necessary P4 components have been provided. Using these VMs is the best supported method for completing this lab. Access your VM by connecting via SSH to the following:

IP: 145.100.132.177
Port: 40000 + Seat Number
Username: student
Password: boj2EeP˜ai

P4 Installation

This is not necessary if using one of the VMs provided. However, for those interested in how to install P4 the following guidance is provided. These steps have been tested on Ubuntu Server 16.04.

I found that this was need to avoid locale issues.

```
sudo locale-gen nl_NL.UTF-8
```

Install the software switch.

```
git clone https://github.com/p4lang/behavioral-model.git
cd behavioral-model/
./install_deps.sh
./autogen.sh
./configure
make
sudo make install
cd ..
```

Install the P4 compiler.

```
git clone https://github.com/p4lang/p4c-bm.git
cd p4c-bm/
sudo pip install -r requirements.txt
sudo python setup.py install
```

Sometimes necessary to avoid having to log out.

```
sudo ldconfig
```

Install Scapy.

```
sudo apt install python-scapy
sudo apt install python-ipaddr
```

Install the latest stable version of Wireshark.

```
sudo add-apt-repository ppa:wireshark-dev/stable
Sudo apt update
sudo apt install wireshark-gtk
Sudo apt install tshark
```

Warnings

Some things to watch out for while completing the lab. P4

- The parsing of trailers is not supported.
- Multicast/Broadcast is not implemented directly in P4.
- Do not attempt to create loops in the program flow. This is not allowed.

Simple Switch

- Use the –log-console option to get detailed information about the processing of packets.
- The internal numbering of the ports is determined by -i option.

The Simple Switch CLI

- The CLI will need to be restarted whenever the switch is restarted.
- Pressing enter on a blank line resends the last command.
- When using the table_add command '=>' is required even when not passing a parameter.
- Common address formats can be used (IP, MAC, etc), as well as decimal or hex.
- Some output may be in hex without the 0x prefix.

Compiler

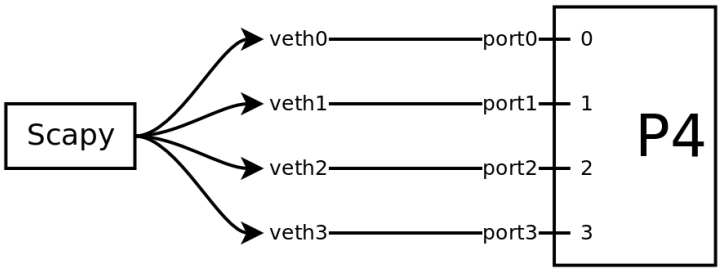
- Ignore the "Header type standard_metadata_t not byte-aligned, adding padding" warning.

Scapy

- Scapy will automatically populate some fields when it can, such as the Ethertype and IP protocol. It may also generate some warnings related to this.

Capturing

- An update version of Wireshark has been installed that supports the interface_name field. This is useful when capturing multiple interfaces at once.



Walkthrough

Several virtual ethernet pairs will be used. One end of each pair will be connected to the P4 switch. Scapy can attach to the other end to send packets. Wireshark can then be used to capture traffic on either end of the pair. The above diagram shows how the components will interact. The files needed for the lab are in the lab directory.

Use the script provided to create the virtual Ethernet interfaces.

```
cd lab
./veth_setup.sh {0..3}
```

Step 1 - A Trivial P4 Program

This step will ensure that everything is working correctly and walk you through the workflow. View the trival.p4 program. Notice that only the required blocks are defined. No headers are parsed and no tables or actions are defined.

Compile the P4 program to get the JSON config for the switch.

```
p4c-bmv2 --json trivial.json trivial.p4
```

Start the switch using the resulting JSON file. Notice how the virtual ethernet interfaces are associated with P4 internal ports (<P4_INTERNAL_PORT>@<INTERFACE>).

```
sudo simple_switch -i 0@port0 -i 1@port1 -i 2@port2 -i 3@port3 trivial.json
```

Start a traffic capture on all the veth interfaces. You can use any tool to do this so long as you can identify what interface a packet is being received on and the actual order of packets. For example:

```
tshark -n -i veth0 -i veth1 -i veth2 -i veth3 -T fields -e frame.time_relative \
-e frame.interface_name -e eth -e ip -e ipv6 -e udp -e tcp
```

Scapy will be used for packet generation. Start it as super user and use the following command to define a packet with some fields filled in randomly.

```
sudo scapy
p = Ether(src=RandMAC(),dst=RandMAC())/IP(src=RandIP(),dst=RandIP())/ \
UDP(sport=RandShort(),dport=RandShort())
```

Use the following command to send a packet. Change the interface used in the command to send some packets on each of the veth interfaces. Make note of which interfaces the packet is received on.

```
sendp(p, iface='veth0')
```

Pay special attention to the timestamp of the captured packets. The order they are displayed in may not be the order they actually appeared on the interfaces.

Discuss: Describe the observed behavior.

Step 2 - Layer 1 Switching (Repeater)

Review section 6, Standard Intrinsic Metadata, of the P4 specification. Examine the l1_switching.p4 program. Notice that a table and action have been added. The syntax of table declarations is in section 11 and the syntax of action definitions is in section 9.2.

Discuss: What field does a P4 program modify in order to influence the interface a packet is output on? (Reference: P414 Specification, Section 13)

Discuss: What is the function of the "`standard_metadata.ingress_port : exact;`" line in the table declaration?

Discuss: What is the function of the "`modify_field(standard_metadata.egress_spec, port);`" line in the action definition?

Compile the l1_switching.p4 program.

```
p4c-bmv2 --json l1_switching.json l1_switching.p4
```

Start the switch using the resulting JSON file.

```
sudo simple_switch -i 0@port0 -i 1@port1 -i 2@port2 -i 3@port3 l1_switching.json
```

Use scapy to send packets to each of the veth interfaces and make note of which interfaces the packet is seen on.

```
sendp(p, iface='veth0')
```

Discuss: What happens when a packet does not match any table entry?

Start the simple switch CLI.

```
simple_switch_CLI
```

Add entries to the table using the following commands.

```
table_add forwarding_table set_egress 0 => 1
table_add forwarding_table set_egress 1 => 2
table_add forwarding_table set_egress 2 => 3
table_add forwarding_table set_egress 3 => 0
```

Restart the capture if necessary. Then use Scapy to send packets to each of the veth interfaces and make note of which interfaces the packet is output on.

```
sendp(p, iface='veth0')
```

Step 3 - Layer 2 Switching

Examine l2_switching.p4. Notice that a header has been added and the start parser has been updated (sections 2.1, 2.2, and 4 of the specification). The forwarding table has also been modified to match the Ethernet destination field.

Discuss: What is the function of the "`extract(ether_hdr);`" line?

Compile the l2_switching.p4 program and start the switch.

```
p4c-bmv2 --json l2_switching.json l2_switching.p4
sudo simple_switch -i 0@port0 -i 1@port1 -i 2@port2 -i 3@port3 l2_switching.json
```

Start the simple switch CLI.

```
simple_switch_CLI
```

Add entries to the table using the following commands.

```
table_set_default forwarding_table drop_packet
table_add forwarding_table set_egress 02:00:00:00:00:0a => 1
table_add forwarding_table set_egress 02:00:00:00:00:0b => 2
```

Within Scapy use variations of the following command to send packets to various veth interfaces. Use some destination MAC addresses that there are entries for and some that will result in a table miss.

```
sendp(Ether(src=RandMAC(), dst='02:00:00:00:00:0a'), iface='veth0')
```

Discuss: Describe the observed behavior.

Discuss: Notice that the Scapy command used does not include an IP header. Would you expect a packet such as this to be forwarded by a standard layer 2 switch?

Discuss: Does the P4 Language support IPv6? What protocols does P4 support?

Discuss: Does the order of table entries matter?

Tasks

Task 1 - Layer 3 Switching (Routing) (4 pt)

Examine l3_switching.p4. This program implements IPv4 forwarding based on longest prefix matches. Note that MAC addresses are not changed.

Compile the l3_switching.p4 program and start the switch.

```
p4c-bmv2 --json l3_switching.json l3_switching.p4
sudo simple_switch -i 0@port0 -i 1@port1 -i 2@port2 -i 3@port3 l3_switching.json
```

Start the CLI and insert the following entries.

```
simple_switch_CLI
table_set_default ipv4_forwarding_table drop_packet
table_add ipv4_forwarding_table set_egress 10.0.0.0/24 => 1
table_add ipv4_forwarding_table set_egress 10.0.0.0/8 => 2
table_add ipv4_forwarding_table set_egress 10.0.0.0/16 => 3
```

Use scapy to send packets to each destination. Verify that routing is working as expected.

```
sendp(Ether()/IP(dst='10.0.1.1'), iface='veth0')
```

Modify the l3_switchting.p4 program to implement IPv6 while retaining IPv4 support.
This will require:

- Creating an additional header, parser, and table.
- Uncommenting the IPv6 entry in the start parser.
- Modifying the ingress block.

Create table entries to implement the following:

- 2001::/16 forwarded out port 1
- 2001:4860:4860::/48 forwarded out port 2
- 2001:610:158:960::/64 forwarded out port 3
- Drop IPv6 packets to unknown destinations.

The following command can be used to send IPv6 packets with Scapy.

```
sendp(Ether()/IPv6(src=RandIP6(),dst='2001:0:0:1::5'), iface='veth0')
```

Submit: Your P4 file (NAME-t1.p4) and the commands used to configure the tables (NAME-t1.txt).

Task 2 - Modifying a Packet (2 pt)

In a separate file extend your layer 3 switching program to modify the flow label of any IPv6 packet from a unique local address. Set the flow label to 0xABCDE. It may be useful to look at how the egress spec is modified in previous examples.

Submit: Your P4 file (NAME-t2.p4) and the commands used to configure the tables (NAME-t2.txt).

Task 3 - Filtering (4 pt max)

In a separate file extend your layer 3 switching program to do packet filtering. Create a control block called egress. This block is automatically executed after the ingress block. This is a good place to do filtering but do not attempt to change the egress port in this block.

Task 3.1 - Layer 2 Filtering

Create a table that will filter based on the destination mac address. Drop any packets going to the layer 2 broadcast address. (1 pt)

OR

Create a table that will filter any packets destined for a layer 2 multicast address, (hint: use a ternary match). (2 pt)

Task 3.2 - Higher Layer Filtering

Use another filtering table to drop traffic from the IP address 172.16.0.10 through 172.16.0.50. Do NOT create a separate entry for each IP in the range. IPv6 should not be affected by this filter. (1 pt)

OR

Implement UDP and TCP parsing (don't attempt to parse TCP options) and filter traffic destined to either TCP port 23 or UDP port 69. Assume that there are no IPv4 option fields or IPv6 extension headers. (2 pt)

Submit: Your P4 file (NAME-t3.p4) and the commands used to configure the tables (NAME-t3.txt).