Selected Topics on Software Defined Networking

EECE.7290

Instructor: *Prof. Yan Luo*

Lab 3: Programming Data Plane Using P4

Hand in Date: 27/03/17

Due Date: 27/03/17

By,

- *Naga Ganesh Kurapati*

ID:01592079

Table of contents:

a. Purpose

The main purpose this lab is to learn about development of data plane applications using a domain specific language called P4. We need to run a simple router applications written in P4 language. And observe the working of those behavioral models.

b. Lab Procedure

Initially we need to clone P4 repositories on VM as shown below.

Make a new directory in the home folder

```
mkdir p4lang
```

Clone following repositories in the that folder

```
cd p4lang
git clone https://github.com/p4lang/behavioral-model.git bmv2
git clone https://github.com/p4lang/p4c-bm.git p4c-bmv2
git clone https://github.com/p4lang/p4factory.git
```

Now we need to install the dependencies for p4factory and execute the behavioral model version 1 as shown below.

Go to p4factory in p4lang floder

```
cd p4factory
```

Update the submodule

```
git submodule update --init –recursive
```

To install Ubuntu dependencies

```
./install_deps.sh
```

creating veth interfaces that the simulator can connect to

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

Use autoconf tools to generate makefiles

```
./autogen.sh
./configure
```

To use one of the targets: Simple router

```
cd targets/simple_router
make bm
```

To run this behavioral model

```
./run_demo.bash
```

To install table entries

```
./run_add_demo_entries.bash
```

Now we see the screenshots of the behavioral model before and after adding table entries. The ping is reachable after adding table entries because there is no table generation mechanism exits. I mean no central controller exits.

```
Terminal
File  Edit  View  Terminal  Tabs  Help
Untitled                                                                 Untitled
11 packets transmitted, 11 received, 0% packet loss, time 10005ms
rtt min/avg/max/mdev = 0.581/0.946/1.450/0.292 ms
mininet> exit
*** Stopping 1 switches
s1 ..
*** Stopping 2 hosts
h1 h2
*** Stopping 0 controllers

*** Done
ubuntu@sdnhubvm:~/p4lang/p4factory/targets/simple_router[08:07] (master)$ ./run_demo.bash
*** Creating network
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1 Starting P4 switch s1
/home/ubuntu/p4lang/p4factory/targets/simple_router/behavioral-model --name s1 --dpid 0000000000000001 -i s1-eth1 -i s1-eth
switch has been started

**********
h1
default interface: eth0 10.0.0.10       00:04:00:00:00:00
**********
**********
h2
default interface: eth0 10.0.1.10       00:04:00:00:00:01
**********
Ready !
*** Starting CLI:
mininet> h1 ping h2
```

From above fig., we can see that h1 ping h2 is not reachable

```
ubuntu@sdnhubvm:~/p4lang/p4factory/targets/simple_router[22:11] (master)$ ./run_add_demo_entries.bash
Inserted entry with handle 0
Inserted entry with handle 1
Inserted entry with handle 0
Inserted entry with handle 1
Inserted entry with handle 0
Inserted entry with handle 1
ubuntu@sdnhubvm:~/p4lang/p4factory/targets/simple_router[08:09] (master)$
```

In above fig., we installed the table entries

```
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=39 ttl=63 time=2.68 ms
64 bytes from 10.0.1.10: icmp_seq=40 ttl=63 time=1.16 ms
64 bytes from 10.0.1.10: icmp_seq=41 ttl=63 time=1.31 ms
64 bytes from 10.0.1.10: icmp_seq=42 ttl=63 time=2.38 ms
64 bytes from 10.0.1.10: icmp_seq=43 ttl=63 time=1.77 ms
64 bytes from 10.0.1.10: icmp_seq=44 ttl=63 time=0.834 ms
64 bytes from 10.0.1.10: icmp_seq=45 ttl=63 time=0.631 ms
64 bytes from 10.0.1.10: icmp_seq=46 ttl=63 time=1.40 ms
64 bytes from 10.0.1.10: icmp_seq=47 ttl=63 time=0.544 ms
64 bytes from 10.0.1.10: icmp_seq=48 ttl=63 time=0.575 ms
64 bytes from 10.0.1.10: icmp_seq=49 ttl=63 time=1.17 ms
64 bytes from 10.0.1.10: icmp_seq=50 ttl=63 time=1.24 ms
64 bytes from 10.0.1.10: icmp_seq=51 ttl=63 time=0.844 ms
64 bytes from 10.0.1.10: icmp_seq=52 ttl=63 time=1.09 ms
64 bytes from 10.0.1.10: icmp_seq=53 ttl=63 time=1.20 ms
64 bytes from 10.0.1.10: icmp_seq=54 ttl=63 time=1.32 ms
64 bytes from 10.0.1.10: icmp_seq=55 ttl=63 time=1.44 ms
64 bytes from 10.0.1.10: icmp_seq=56 ttl=63 time=1.04 ms
64 bytes from 10.0.1.10: icmp_seq=57 ttl=63 time=1.40 ms
64 bytes from 10.0.1.10: icmp_seq=58 ttl=63 time=2.07 ms
64 bytes from 10.0.1.10: icmp_seq=59 ttl=63 time=1.08 ms
^C
--- 10.0.1.10 ping statistics ---
59 packets transmitted, 21 received, 64% packet loss, time 58162ms
rtt min/avg/max/mdev = 0.544/1.296/2.681/0.542 ms
mininet>
```

In above fig., we can see h1 ping h2 reachable after installing table entries

Now let us setup and run behavioral model version 2

To check the requirements and install

```
cd $HOME/p4lang/p4c-bmv2
sudo pip install -r requirements.txt
sudo pip install scapy thrift networkx
```

To build the code

```
cd $HOME/p4lang/bmv2
./autogen.sh
./configure
Make
```

To run the application

```
cd $HOME/p4lang/bmv2/mininet

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

To install table entries - Using the CLI to populate tables

```
cd $HOME/p4lang/bmv2/targets/simple_router
./runtime_CLI < commands.txt
```

Now we see the screenshots of the behavioral model before and after adding table entries. The ping is reachable after adding table entries because there is no table generation mechanism exits. I mean no central controller exits.

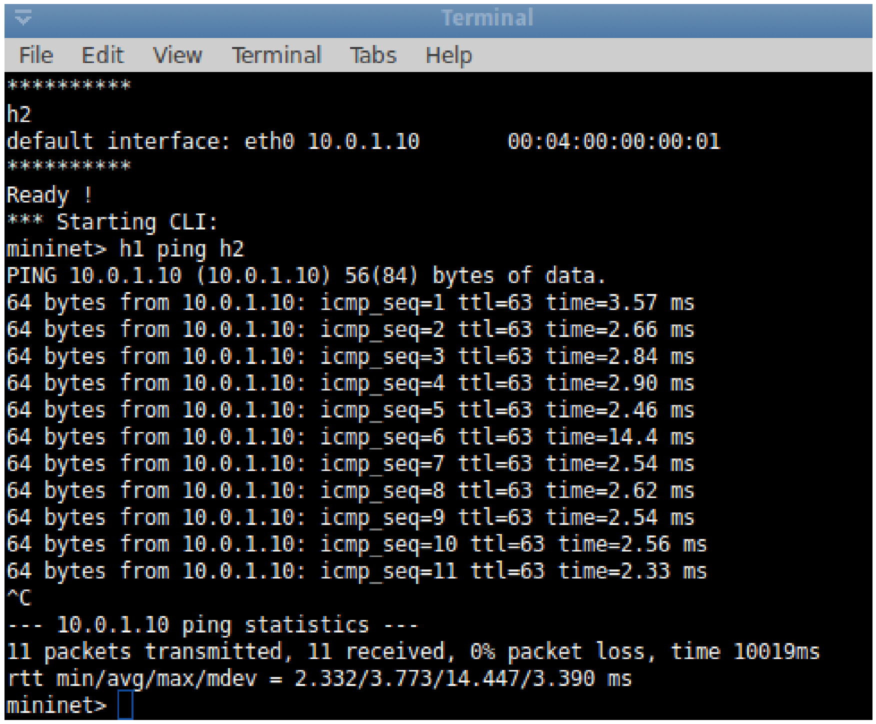
```
ubuntu@sdnhubvm:~/p4lang/bmv2/mininet[17:38] (master)$ sudo python 1sw_demo.py -
-behavioral-exe ../targets/simple_router/simple_router --
usage: 1sw_demo.py [-h] --behavioral-exe BEHAVIORAL_EXE
                   [--thrift-port THRIFT_PORT] [--num-hosts NUM_HOSTS]
                   [--mode {l2,l3}] --json JSON [--pcap-dump PCAP_DUMP]
1sw_demo.py: error: argument --json is required
ubuntu@sdnhubvm:~/p4lang/bmv2/mininet[17:38] (master)$ sudo python 1sw_demo.py -
-behavioral-exe ../targets/simple_router/simple_router --json ../targets/simple_
router/simple_router.json
*** Creating network
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1 Starting P4 switch s1.
../targets/simple_router/simple_router -i 1@s1-eth1 -i 2@s1-eth2 --thrift-port 9
090 --nanolog ipc:///tmp/bm-0-log.ipc --device-id 0 ../targets/simple_router/sim
ple_router.json
P4 switch s1 has been started.

**********
h1
default interface: eth0 10.0.0.10       00:04:00:00:00:00
**********
**********
h2
default interface: eth0 10.0.1.10       00:04:00:00:00:01
**********
Ready !
*** Starting CLI:
mininet> h1 ping h2
```

In above fig., we can see h1 ping h2 is unreachable before populating table entries

```
ubuntu@sdnhubvm:~/p4lang/bmv2/targets/simple_router[17:43] (master)$ ./runtime_C
LI < commands.txt
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: Setting default action of send_frame
action:              _drop
runtime data:
RuntimeCmd: Setting default action of forward
action:              _drop
runtime data:
RuntimeCmd: Setting default action of ipv4_lpm
action:              _drop
runtime data:
RuntimeCmd: Adding entry to exact match table send_frame
match key:           EXACT-00:01
action:              rewrite_mac
runtime data:        00:aa:bb:00:00:00
Entry has been added with handle 0
RuntimeCmd: Adding entry to exact match table send_frame
match key:           EXACT-00:02
action:              rewrite_mac
runtime data:        00:aa:bb:00:00:01
Entry has been added with handle 1
RuntimeCmd: Adding entry to exact match table forward
match key:           EXACT-0a:00:00:0a
action:              set_dmac
runtime data:        00:04:00:00:00:00
Entry has been added with handle 0
```

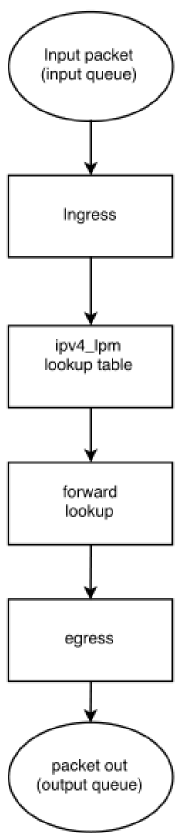
In above fig., table entries are installed



```
Terminal
File  Edit  View  Terminal  Tabs  Help
**********
h2
default interface: eth0 10.0.1.10       00:04:00:00:00:01
**********
Ready !
*** Starting CLI:
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=1 ttl=63 time=3.57 ms
64 bytes from 10.0.1.10: icmp_seq=2 ttl=63 time=2.66 ms
64 bytes from 10.0.1.10: icmp_seq=3 ttl=63 time=2.84 ms
64 bytes from 10.0.1.10: icmp_seq=4 ttl=63 time=2.90 ms
64 bytes from 10.0.1.10: icmp_seq=5 ttl=63 time=2.46 ms
64 bytes from 10.0.1.10: icmp_seq=6 ttl=63 time=14.4 ms
64 bytes from 10.0.1.10: icmp_seq=7 ttl=63 time=2.54 ms
64 bytes from 10.0.1.10: icmp_seq=8 ttl=63 time=2.62 ms
64 bytes from 10.0.1.10: icmp_seq=9 ttl=63 time=2.54 ms
64 bytes from 10.0.1.10: icmp_seq=10 ttl=63 time=2.56 ms
64 bytes from 10.0.1.10: icmp_seq=11 ttl=63 time=2.33 ms
^C
--- 10.0.1.10 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10019ms
rtt min/avg/max/mdev = 2.332/3.773/14.447/3.390 ms
mininet>
```

In above fig., we can see that h1 ping h2 is reachable.

c. <u>Software</u>



We can see the flowchart above. The mechanism for parsing the packets and lookup tables for the both behavioral models remains the same. But version 2 divides the switch logic and auto-generated PD API into different process.

d. Trouble shooting:

We observe the packets received when the h1 ping h2 happens. It is clearly shown in the lab procedure section. We can also use xterm to cross check the packets received.

e. Results:

Here we are trying to answer the questions posted in the handout.

In the bmv1 case, explain why the ping command did not go through until the command is executed in terminal 2?

The ping is reachable after adding table entries because there is no table generation mechanism exits. I mean no central controller exits.

Explain the difference between behavior model v1 and v2?

The main difference is version 2 divides the switch logic and auto-generated PD API into different process. Whereas in version 1 they are combined.

We can the screen shots in the lab procedure section

f. <u>Appendix:</u>

Version 1

```
#include "includes/headers.p4"
#include "includes/parser.p4"

action _drop() {
    drop();
}

header_type routing_metadata_t {
    fields {
        nhop_ipv4 : 32;
    }
}

metadata routing_metadata_t routing_metadata;

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

table ipv4_lpm {
    reads {
        ipv4.dstAddr : lpm;
    }
    actions {
        set_nhop;
        _drop;
    }
    size: 1024;
}

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

table forward {
    reads {
        routing_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 {
    apply(ipv4_lpm);
    apply(forward);
}

control egress {
    apply(send_frame);
}

/*----------------------------Headers.p4-------------------------*/
header_type ethernet_t {
    fields {
        dstAddr : 48;
        srcAddr : 48;
        etherType : 16;
    }
}

header_type ipv4_t {
    fields {
        version : 4;
        ihl : 4;
        diffserv : 8;
        totalLen : 16;
        identification : 16;
        flags : 3;
        fragOffset : 13;
        ttl : 8;
        protocol : 8;
        hdrChecksum : 16;
        srcAddr : 32;
        dstAddr: 32;
    }
}
/*----------------------------parser.p4-------------------------*/
parser start {
    return parse_ethernet;
}

#define ETHERTYPE_IPV4 0x0800

header ethernet_t ethernet;

parser parse_ethernet {
    extract(ethernet);
```

```
    return select(latest.etherType) {
        ETHERTYPE_IPV4 : parse_ipv4;
        default: ingress;
    }
}

header ipv4_t ipv4;

field_list ipv4_checksum_list {
        ipv4.version;
        ipv4.ihl;
        ipv4.diffserv;
        ipv4.totalLen;
        ipv4.identification;
        ipv4.flags;
        ipv4.fragOffset;
        ipv4.ttl;
        ipv4.protocol;
        ipv4.srcAddr;
        ipv4.dstAddr;
}

field_list_calculation ipv4_checksum {
    input {
        ipv4_checksum_list;
    }
    algorithm : csum16;
    output_width : 16;
}

calculated_field ipv4.hdrChecksum  {
    verify ipv4_checksum;
    update ipv4_checksum;
}

parser parse_ipv4 {
    extract(ipv4);
    return ingress;
}
```

Version 2

```
header_type ethernet_t {
    fields {
        dstAddr : 48;
        srcAddr : 48;
        etherType : 16;
    }
}

header_type ipv4_t {
    fields {
        version : 4;
        ihl : 4;
        diffserv : 8;
```

```
        totalLen : 16;
        identification : 16;
        flags : 3;
        fragOffset : 13;
        ttl : 8;
        protocol : 8;
        hdrChecksum : 16;
        srcAddr : 32;
        dstAddr: 32;
    }
}

parser start {
    return parse_ethernet;
}

#define ETHERTYPE_IPV4 0x0800

header ethernet_t ethernet;

parser parse_ethernet {
    extract(ethernet);
    return select(latest.etherType) {
        ETHERTYPE_IPV4 : parse_ipv4;
        default: ingress;
    }
}

header ipv4_t ipv4;

field_list ipv4_checksum_list {
        ipv4.version;
        ipv4.ihl;
        ipv4.diffserv;
        ipv4.totalLen;
        ipv4.identification;
        ipv4.flags;
        ipv4.fragOffset;
        ipv4.ttl;
        ipv4.protocol;
        ipv4.srcAddr;
        ipv4.dstAddr;
}

field_list_calculation ipv4_checksum {
    input {
        ipv4_checksum_list;
    }
    algorithm : csum16;
    output_width : 16;
}

calculated_field ipv4.hdrChecksum  {
    verify ipv4_checksum;
    update ipv4_checksum;
```

```
}

parser parse_ipv4 {
    extract(ipv4);
    return ingress;
}


action _drop() {
    drop();
}

header_type routing_metadata_t {
    fields {
        nhop_ipv4 : 32;
    }
}

metadata routing_metadata_t routing_metadata;

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

table ipv4_lpm {
    reads {
        ipv4.dstAddr : lpm;
    }
    actions {
        set_nhop;
        _drop;
    }
    size: 1024;
}

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

table forward {
    reads {
        routing_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 {
    if(valid(ipv4) and ipv4.ttl > 0) {
        apply(ipv4_lpm);
        apply(forward);
    }
}

control egress {
    apply(send_frame);
}
```