Exercise for Lecture Software Defined Networking

Prof. Dr. David Hausheer, Julius Rückert
Christian Koch, Jeremias Blendin, Leonhard Nobach, Matthias Wichtlhuber

TECHNISCHE UNIVERSITÄT DARMSTADT

Winter Term 2016/17
Exercise No. 7
Published: 17.01.2017
Submission exclusively via Moodle, Deadline: 24.01.2017
Contact: Please use the Moodle forum to post questions and remarks on the exercise.
Web: `http://www.ps.tu-darmstadt.de/teaching/ws1617/sdn/`
Submission: `https://moodle.tu-darmstadt.de/course/view.php?id=8385`

Surname (Nachname):	
First name (Vorname):	
ID# (Matrikelnummer):	

Team member: Letian Feng(2255840), Zhen Chen(2665935), Chunyuan Yu(2587628)

Problem 7.1 - P4 Parse Graph

Given the following file `headers.p4`.

```

header_type ethernet_t {
    fields {
        bit<48> dstAddr;
        bit<48> srcAddr;
        bit<16> etherType;
    }
}

header_type vlan_t {
    fields {
        bit<16> vlanTag;
        bit<16> etherType;
    }
}

parser start {
    return parse_ethernet;
}

header ethernet_t ethernet;

#define ETHERTYPE_VLAN 0x8100

```

```
parser parse_ethernet {
    extract(ethernet);
    return select(latest.etherType) {
        ETHERTYPE_VLAN : parse_vlan;
        default: ingress;
    }
}

header vlan_t vlan;

parser parse_vlan {
    extract(vlan);
    return ingress;
}
```

a) Explain the general purpose of the P4 code above, and describe the result of the code applied to an incoming packet.

The general purpose of the code is to check if the incoming packet is a vlan-tagged packet, if yes then get the vlan-tag of the packet and do some work defined in control "ingress". For example, a packet vlan-tagged with vlan-5 will be first parsed in "parse_ethernet" and be selected as "ETHERTYPE_VLAN", then the paser "parse_vlan" will be executed and all the information of header_type "vlan_t" will be extracted and then saved in the header "vlan" waiting for the processing in the "ingress".

b) What does the statement `return ingress;` do?

Ingress will check if the header "vlan" is valid, if yes then do some other processing such as untag the packet and forward it to a port, or change the tag etc. Otherwise, the ingress could add a vlan-tag on the packet or do some other works.

Problem 7.2 - P4 Matchers and Actions

Given the following P4 code.

```
#include "headers.p4" //this is the file in the last problem
                                        // import program in problem 7.1
header_type ingress_md_t {              // define header type ingress_md_t, which has a field vlanid
    fields {
  bit<16> vlanid;
    }
}

metadata ingress_md_t ingress_md;       // declare a metadata named ingress_med in type of ingress_md_t

action _nop() { }                       // do nothing

// ————————————————————————————————————————

table map_vlan {  // table for vlanid, read ingress port & vlanid in full length and save in the table;
    reads {       // table can do actions: sendPushTag, sendChangeTag, sendPopTag, send or _drop
        standard_metadata.ingress_port : exact;
        ingress_md.vlanid : exact;
    }
    actions {
        sendPushTag;
        sendChangeTag;
        sendPopTag;
  send;
  _drop;
    }
}

action sendPushTag(in bit<16> tag, in bit<16> port) { // add parameter "tag" to the
  add_header(vlan);                                   // vlan header and send it out
  // Missing code here                                // via parameter "port"
}

action sendChangeTag(in bit<16> tag, in bit<16> port) { // change the tag of vlan
  vlan.vlanTag = tag;                                   // header to the parameter
  standard_metadata.egress_spec = port;                 // "tag" and send it out via
}                                                       // parameter "port"

action sendPopTag(in bit<16> port) {     // set the packet non-vlan-tagged & remove the vlan-tag
  ethernet.etherType = vlan.etherType;   // and send it out via parameter "port"
  remove_header(vlan);
  standard_metadata.egress_spec = port;
}

action send(in bit<16> port) {           // set egress data in the metadata as the parameter "port"
  standard_metadata.egress_spec = port;
}
```

```

action _drop() {                                                    // drop the packet
    drop();
}

// ————————————————————————————————————————————

action set_has_vlan() {                      // set the vlanid in the metadata as in vlan header
  ingress_md.vlanid = vlan.vlanTag;
}

action set_has_no_vlan() {                           // set vlanid in the the metadata as 0
  ingress_md.vlanid = 0;
}


table prepare {     // table for preparation, read the etherType in full length and save in the table,
  reads {           // the table can do actions: set_has_vlan, set_has_no_vlan
    ethernet.etherType : exact;
  }
  actions {
    set_has_vlan;
    set_has_no_vlan;
  }
}

control ingress {  // call the table "prepare" & "map_vlan" to read the packet and do relevant works
    apply(prepare);
    apply(map_vlan);
}

control egress { }                                                  // do nothing
```

a) Describe the semantics of the code above. Be precise.

As shown above, in green.

b) Complete the missing code in the `sendPushTag` action.

```
action sendPushTag ( in bit <16> tag , in bit <16> port ) {
        add_header(vlan);
        ethernet.etherType = ETHERTYPE_VLAN;
        standard_metadata.egress_spec = port ;
 }
```

c) P4 Pipelines and Flow Entries

Assume the P4 code is installed on a switch with 4 ports (Port 0-3). Port 0 is designated as a tagged trunk port, Port 1,2,3 are designated as untagged access ports. Packets coming in on any of the ports 1,2 or 3 shall be forwarded on Port 0, tagged with the respective VLAN 1,2, or 3. Likewise, packets coming in on Port 0 and being tagged with the VLAN 1,2 or 3 must be forwarded untagged on the respective port 1,2, or 3.

Create a set of flow entries for the tables `prepare` and `map_vlan` that ensure the switch behaves as desired.

Use an informal notation like "`match=x:1,y:2, action=foobar`"

match=ethernet.etherType:0x8100, standard_metadata.ingress_port:0, ingress_md.vlanid:1,
action=sendChangeTag(0,1);
match=ethernet.etherType:0x8100, standard_metadata.ingress_port:0, ingress_md.vlanid:2,
action=sendChangeTag(0,2);
match=ethernet.etherType:0x8100, standard_metadata.ingress_port:0, ingress_md.vlanid:3,
action=sendChangeTag(0,3);
match=ethernet.etherType:0x0800, standard_metadata.ingress_port:1, action=sendPushTag(1,0);
match=ethernet.etherType:0x0800, standard_metadata.ingress_port:2, action=sendPushTag(2,0);
match=ethernet.etherType:0x0800, standard_metadata.ingress_port:3, action=sendPushTag(3,0);