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):	

Problem 7.1 - P4 Parse Graph

Given the following file `headers.p4`.

```
1
2  header_type ethernet_t {
3      fields {
4          bit<48> dstAddr;
5          bit<48> srcAddr;
6          bit<16> etherType;
7      }
8  }
9
10 header_type vlan_t {
11     fields {
12         bit<16> vlanTag;
13         bit<16> etherType;
14     }
15 }
16
17 parser start {
18     return parse_ethernet;
19 }
20
21 header ethernet_t ethernet;
22
23 #define ETHERTYPE_VLAN 0x8100
24
```

```
25 parser parse_ethernet {
26     extract(ethernet);
27     return select(latest.etherType) {
28         ETHERTYPE_VLAN : parse_vlan;
29         default: ingress;
30     }
31 }
32
33 header vlan_t vlan;
34
35 parser parse_vlan {
36     extract(vlan);
37     return ingress;
38 }
```

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

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

Problem 7.2 - P4 Matchers and Actions

Given the following P4 code.

```
#include "headers.p4" //this is the file in the last problem

header_type ingress_md_t {
    fields {
  bit<16> vlanid;
    }
}

metadata ingress_md_t ingress_md;

action _nop() { }

// ------------------------------------------------

table map_vlan {
    reads {
        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_header(vlan);
  // Missing code here
}

action sendChangeTag(in bit<16> tag, in bit<16> port) {
  vlan.vlanTag = tag;
  standard_metadata.egress_spec = port;
}

action sendPopTag(in bit<16> port) {
  ethernet.etherType = vlan.etherType;
  remove_header(vlan);
  standard_metadata.egress_spec = port;
}

action send(in bit<16> port) {
  standard_metadata.egress_spec = port;
}
```

```
48
49 action _drop() {
50     drop();
51 }
52
53 // ------------------------------------------------
54
55 action set_has_vlan() {
56   ingress_md.vlanid = vlan.vlanTag;
57 }
58
59 action set_has_no_vlan() {
60   ingress_md.vlanid = 0;
61 }
62
63
64 table prepare {
65   reads {
66     ethernet.etherType : exact;
67   }
68   actions {
69     set_has_vlan;
70     set_has_no_vlan;
71   }
72 }
73
74 control ingress {
75     apply(prepare);
76     apply(map_vlan);
77 }
78
79 control egress { }
```

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

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

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`"