Source Routing
Using P4 match-action tables

Felipe Novais
Thiago Henrique S Rodrigues

Tables in Control Plane

```
table_set_default sroute_exact drop
table_set_default ipv4_lpm drop

table_add sroute_exact sroute_forward 10.0.0.10 10.0.2.10 => 0x000000000000020302
table_add sroute_exact sroute_forward 10.0.2.10 10.0.0.10 => 0x000000000000010101

table_add sroute_exact sroute_forward 10.0.1.10 10.0.0.10 => 0x000000000000010303
table_add sroute_exact sroute_forward 10.0.1.10 10.0.2.10 => 0x000000000000020301

table_add ipv4_lpm ipv4_forward 10.0.1.10/32 => 00:04:00:00:00:01 2
~
command_s1.txt                                                  2,1           All
table_set_default sroute_exact drop
table_set_default ipv4_lpm drop

table_add sroute_exact sroute_forward 10.0.0.10 10.0.2.10 => 0x000000000000020302
table_add sroute_exact sroute_forward 10.0.2.10 10.0.0.10 => 0x000000000000010101

table_add sroute_exact sroute_forward 10.0.1.10 10.0.0.10 => 0x000000000000010303
table_add sroute_exact sroute_forward 10.0.1.10 10.0.2.10 => 0x000000000000020301

table_add ipv4_lpm ipv4_forward 10.0.1.10/32 => 00:04:00:00:00:01 2
~
command_s2.txt                                                  4,1           All
table_set_default sroute_exact drop
table_set_default ipv4_lpm drop

table_add sroute_exact sroute_forward 10.0.0.10 10.0.2.10 => 0x000000000000020302
table_add sroute_exact sroute_forward 10.0.2.10 10.0.0.10 => 0x000000000000010101

table_add sroute_exact sroute_forward 10.0.1.10 10.0.0.10 => 0x000000000000010303
table_add sroute_exact sroute_forward 10.0.1.10 10.0.2.10 => 0x000000000000020301

table_add ipv4_lpm ipv4_forward 10.0.1.10/32 => 00:04:00:00:00:01 1
command_s3.txt                                                 10,1           All
```

Defines

```
 7
 8 #define MAX_HOPS 9
 9 // H1 - 10.0.0.10
10 #define H1 0x0a00000a
11 // H2 - 10.0.1.10
12 #define H2 0x0a00010a
13 // H3 - 10.0.2.10
14 #define H3 0x0a00020a
15
```

Parsers

```
74      state parse_ethernet {
75          packet.extract(hdr.ethernet);
76          transition select(hdr.ethernet.etherType) {
77                      TYPE_SRCROUTING: parse_srcRouting;
78                      TYPE_IPV4: parse_ipv4;
79                      default: accept;
80          }
81      }
82
83      state parse_srcRouting {
84          packet.extract(hdr.srcRoutes.next);
85          transition select(hdr.srcRoutes.last.bos) {
86                      1: parse_ipv4;
87                      default: parse_srcRouting;
88          }
89      }
90
91      state parse_ipv4 {
92          packet.extract(hdr.ipv4);
93          transition accept;
94      }
```

Action sroute_forward

```
133     action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
134         standard_metadata.egress_spec = port;
135         hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
136         hdr.ethernet.dstAddr = dstAddr;
137         hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
138     }
```

Tables in P4

```
177     table ipv4_lpm {
178         key = {
179             hdr.ipv4.dstAddr: lpm;
180         }
181         actions = {
182             ipv4_forward;
183             drop;
184             NoAction;
185         }
186         size = 1024;
187         default_action = NoAction();
188     }
189
190     table sroute_exact {
191         key = {
192             hdr.ipv4.srcAddr: exact;
193             hdr.ipv4.dstAddr: exact;
194         }
195         actions = {
196             sroute_forward;
197             drop;
198             NoAction;
199         }
200         size = 1024;
201         default_action = NoAction();
202     }
```

Apply

```
198     apply {
199             // If it's H2, uses destination routing | else if it's etherType is srcRouting use source routing
200             if (hdr.ipv4.dstAddr == H2) {
201                 ipv4_lpm.apply();
202
203             } else if (hdr.ethernet.etherType == TYPE_SRCROUTING) {
204                 // Check if it's an empty source routing header, then initialize using table values
205                 if (hdr.srcRoutes[0].port == 0 && hdr.srcRoutes[0].bos == 1) {
206                     hdr.srcRoutes.push_front(2);
207                     sroute_exact.apply();
208                 }
209
210                 // Usual flow for source routing used in task 3
211                 if (hdr.srcRoutes[0].isValid()){
212                     if (hdr.srcRoutes[0].bos == 1) {
213                         srcRoute_finish();
214                     }
215                     srcRoute_nhop();
216                     if (hdr.ipv4.isValid()){
217                         update_ttl();
218                     }
219                 }
220             }
221             else {
222                 drop();
223             }
224     }
```

Action sroute_forward

```
140     action sroute_forward(bit<72> sourceRoute) {
141         // Initializing bos with zeroes for all elements in srcRoutes
142         hdr.srcRoutes[0].bos = 0;
143         hdr.srcRoutes[1].bos = 0;
144         hdr.srcRoutes[2].bos = 0;
145         hdr.srcRoutes[3].bos = 0;
146         hdr.srcRoutes[4].bos = 0;
147         hdr.srcRoutes[5].bos = 0;
148         hdr.srcRoutes[6].bos = 0;
149         hdr.srcRoutes[7].bos = 0;
150         hdr.srcRoutes[8].bos = 1;
151 
152         // Decoding values from table using bitmask technique
153         hdr.srcRoutes[0].port = (bit<15>) ((sourceRoute & 0x00000000000000ff));
154         hdr.srcRoutes[1].port = (bit<15>) ((sourceRoute & 0x000000000000ff00) >> (8 * 1));
155         hdr.srcRoutes[2].port = (bit<15>) ((sourceRoute & 0x0000000000ff0000) >> (8 * 2));
156         hdr.srcRoutes[3].port = (bit<15>) ((sourceRoute & 0x00000000ff000000) >> (8 * 3));
157         hdr.srcRoutes[4].port = (bit<15>) ((sourceRoute & 0x000000ff00000000) >> (8 * 4));
158         hdr.srcRoutes[5].port = (bit<15>) ((sourceRoute & 0x0000ff0000000000) >> (8 * 5));
159         hdr.srcRoutes[6].port = (bit<15>) ((sourceRoute & 0x00ff000000000000) >> (8 * 6));
160         hdr.srcRoutes[7].port = (bit<15>) ((sourceRoute & 0xff00000000000000) >> (8 * 7));
161         hdr.srcRoutes[8].port = (bit<15>) ((sourceRoute & 0xff00000000000000) >> (8 * 8));
162 
163         // Setting the bos base on the next element port
164         if (hdr.srcRoutes[1].port == 0) hdr.srcRoutes[0].bos = 1;
165         if (hdr.srcRoutes[2].port == 0) hdr.srcRoutes[1].bos = 1;
166         if (hdr.srcRoutes[3].port == 0) hdr.srcRoutes[2].bos = 1;
167         if (hdr.srcRoutes[4].port == 0) hdr.srcRoutes[3].bos = 1;
168         if (hdr.srcRoutes[5].port == 0) hdr.srcRoutes[4].bos = 1;
169         if (hdr.srcRoutes[6].port == 0) hdr.srcRoutes[5].bos = 1;
170         if (hdr.srcRoutes[7].port == 0) hdr.srcRoutes[6].bos = 1;
171         if (hdr.srcRoutes[8].port == 0) hdr.srcRoutes[7].bos = 1;
172 
173         // The last element will always have bos 1
174         hdr.srcRoutes[8].bos = 1;
175     }
```

Bitmask explanation

```
Consider we have 0x0201:
0x0201 = 0000 0010 0000 0001 <- Binary representation
0x0201 ^ 0x00FF = 0000 0010 0000 0001 ^ 0000 0000 1111 1111
```

bitwise or

```
0000 0010 0000 0001
0000 0000 1111 1111
-------------------
0000 0000 0000 0001 = 0x0001

0x0201 ^ 0xFF00 = 0000 0010 0000 0001 ^ 1111 1111 0000 0000
```

bitwise or

```
0000 0010 0000 0001
1111 1111 0000 0000
-------------------
0000 0010 0000 0000 = 0x0200

Doing a shift right by 8 bits we isolate the second 2 bytes.
0x0200 >> 8 = 0000 0010 0000 0000 = 0000 00010 = 0x02
```

shift right

Thank you!