

Project 2. OpenFlow

312551015 Sean 韋詠祥

Part 1: Answer Questions

Q1. How many OFPT_FLOW_MOD packets are there

When we enable `org.onosproject.openflow` and start the Mininet topology, the controller will send 3 OFPFC_ADD command within one packet, respectively for lldp, arp, bddp (broadcast domain discovery protocol).

After we enable `org.onosproject.fwd`, there will be 1 packet with OFPFC_ADD command to match IPv4 with priority 5

Then we use the command `h1 ping h2 -c 5` in Mininet, the controller will send two OFPFC_ADD command. The first one is the rule for h1 send packet to h2, the latter is for the opposite direction.

After 10 seconds timeout, two OFPFC_DELETE_STRICT command will be issued.

When we closed the Mininet, after four OFPT_FLOW_REMOVED command, it will be four OFPFC_ADD command for lldp, arp, bddp, ipv4 respectively.

2. What are the fields in each OFPT_FLOW_MOD messages?

There are 6 distinct OFPT_FLOW_MOD headers during the experiment.

Match fields	Action	Timeout value
ETH_TYPE = LLDP	Output = Controller	0
ETH_TYPE = ARP	Output = Controller	0
ETH_TYPE = BDDP	Output = Controller	0
ETH_TYPE = IPv4	Output = Controller	0
IN_PORT = 1, ETH_SRC = h1, ETH_DST = h2	Output = Port 2	10 seconds
IN_PORT = 2, ETH_SRC = h2, ETH_DST = h1	Output = Port 1	10 seconds

3. What are the Idle Timeout values for all flow rules?

In the Web GUI, we can see the default rules (lldp, arp, bddp, ipv4) have Idle Timeout value of 0, which means there have no timeout limit.

For the two rules added when we ping from h1 to h2, it have timeout value of 10 seconds.

Part 2: Install Flow Rules

1. ARP

Use criteria `ETH_TYPE = 0x0806 (ARP)` to match packets.

And the instructions is to `OUTPUT to ALL ports`.

The terminal session shows the creation of a flow rule named `flows_s1-1_312551015.json`. The flow rule is defined as follows:

```
+ part2 git:(master) x cat flows_s1-1_312551015.json
File: flows_s1-1_312551015.json
1 {
  "priority": 52000,
  "timeout": 0,
  "isPermanent": true,
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0806"
      }
    ]
  },
  "treatment": {
    "instructions": [
      {
        "type": "OUTPUT",
        "port": "ALL"
      }
    ]
  }
}
```

The flow rule is applied to interface `h1` and `h2`, matching ARP traffic (`0x0806`) and outputting to all ports (`ALL`). The command used to create the flow is:

```
+ part2 git:(master) x curl -vsu onos:rocks 127.1:8181/onos/v1/flows/of:0000000000000001/49539599127800013
001 -H 'Content-Type: application/json'
```

Output from the server indicates the flow was created successfully:

```
| 193 - org.onosproject.onos-core-net - 2.7.0 | Application org.onosprojec
t.fwd has been deactivated
| 11 - org.apache.karaf.features.core - 4.2.9 | Uninstalling bundles:
| 11 - org.apache.karaf.features.core - 4.2.9 | Refreshing bundles:
| 11 - org.apache.karaf.features.core - 4.2.9 | org.onosproject.onos-apps-fwd/2.7.
| 11 - org.apache.karaf.features.core - 4.2.9 | Done.
| 193 - org.onosproject.onos-core-net - 2.7.0 | Application org.onosprojec
```

2. IPv4 Ping

For second part we use `IPV4_SRC` and `IPV4_DST` to decide where the packet should go.

The terminal session shows the creation of a flow rule named `flows_s1-3_312551015.json`. The flow rule is defined as follows:

```
+ 1 flows_s1-3 312551015.json
+ 21 +- 2 lines: {
  20   "timeout": 0,
  19   "isPermanent": true,
  18   "selector": {
  17     "criteria": [
  16       {
  15         "type": "IN_PORT",
  14         "port": "1"
  13       },
  12       {
  11         "type": "ETH_TYPE",
  10         "ethType": "0x0800"
  9       },
  8       {
  7         "type": "IPV4_SRC",
  6         "ip": "10.0.0.1/0"
  5       },
  4       {
  3         "type": "IPV4_DST",
  2         "ip": "10.0.0.2/0"
  1       }
  23   ],
  2   "treatment": {
  3     "instructions": [
  4       {
  5         "type": "OUTPUT",
  6         "port": "2"
  7       }
  8     ]
  9   }
  10 }
```

The flow rule is applied to interface `h1` and `h2`, matching ICMP traffic (`0x0800`) and outputting to port `2`. The command used to create the flow is:

```
+ part2 git:(master) x curl -vsu onos:rocks 127.1:8181/onos/v1/flows/of:0000000000000001/48413700041701641
001 -H 'Content-Type: application/json'
```

Output from the server indicates the flow was created successfully:

```
| 201 - org.onosproject..onos-gui2-base-jar - 2.7.0 | GUI client connected -- user <none>
| 201 - org.onosproject..onos-gui2-base-jar - 2.7.0 | Session token authenticated
| 64 bytes from 10.0.0.2: icmp_seq=96 ttl=64 time=0.079 ms
| 64 bytes from 10.0.0.2: icmp_seq=97 ttl=64 time=0.075 ms
| 64 bytes from 10.0.0.2: icmp_seq=98 ttl=64 time=0.075 ms
| 64 bytes from 10.0.0.2: icmp_seq=99 ttl=64 time=0.072 ms
| 64 bytes from 10.0.0.2: icmp_seq=100 ttl=64 time=0.075 ms
| 64 bytes from 10.0.0.2: icmp_seq=101 ttl=64 time=0.095 ms
| 64 bytes from 10.0.0.2: icmp_seq=102 ttl=64 time=0.105 ms
| 64 bytes from 10.0.0.2: icmp_seq=103 ttl=64 time=0.077 ms
| 64 bytes from 10.0.0.2: icmp_seq=104 ttl=64 time=0.072 ms
| 64 bytes from 10.0.0.2: icmp_seq=105 ttl=64 time=0.064 ms
| 64 bytes from 10.0.0.2: icmp_seq=106 ttl=64 time=0.091 ms
| 64 bytes from 10.0.0.2: icmp_seq=107 ttl=64 time=0.071 ms
| 64 bytes from 10.0.0.2: icmp_seq=108 ttl=64 time=0.062 ms
| 64 bytes from 10.0.0.2: icmp_seq=109 ttl=64 time=0.074 ms
| 64 bytes from 10.0.0.2: icmp_seq=110 ttl=64 time=0.091 ms
| 64 bytes from 10.0.0.2: icmp_seq=111 ttl=64 time=0.068 ms
| 64 bytes from 10.0.0.2: icmp_seq=112 ttl=64 time=0.074 ms
* Server auth using Basic with user 'onos'
> HTTP /onos/v1/Flows/of:0000000000000001 HTTP/1.1
> Host: 127.0.0.1:8181
> Authorization: Basic b25vczpyb2Nrcw==
> User-Agent: curl/7.81.0
> Accept: */
> Content-Type: application/json
> Content-Length: 399
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 201 Created
< Location: http://127.0.0.1:8181/onos/v1/flows/of:0000000000000001/48413700041701641
< Content-Length: 0
< Server: Jetty(9.4.28.v20200408)
<
* Connection #0 to host 127.0.0.1 left intact
+ part2 git:(master) x curl -vsu onos:rocks 127.1:8181/onos/v1/flows/of:0000000000000001/48413700041701641
001 -H 'Content-Type: application/json'
```

Part 3: Broadcast Storm

Before the experiment, my average CPU load is about 40%.

```

2023-10-02T20:49:53,348 | WARN | pool-16-thread-1 | OFChannelHandler | 206 - org.onosproject.onos-protocols-openflow-ctl - 2.7.0 | Dropping messages for swi
tch NiciraSwitchHandshaker[session=127.0.0.1:44240, dpid=00:00:d2:43:6f:fe:89:4e] because channel is not connected: [OFMeterStatsRequestVer14(xid=0, flags=[], meterId=-1)]
2023-10-02T20:49:53,348 | WARN | pool-16-thread-1 | NiciraSwitchHandshaker
tch 00:00:d2:43:6f:fe:89:4e because channel is not connected: [OFMeterStatsRequestVer14(xid=0, flags=[], meterId=-1)]
```

In my experiment environment design, I use four switch fully mesh with others. And install the flow rule from part 2-1, which forward all the ARP packets to all ports.

When we use `h1 ping h2` command in Mininet, the average CPU load is over 500%.

```

2023-10-02T20:54:14,731 | INFO | onos-of-channel-handler-runtime-status-0 | OpenFlowControllerImpl$OpenFlowSwitchAgent | 205 - org.onosproject.onos-protocols-openflow-ctl -
2.7.0 | Purged pending stats 00:00:00:00:00:00:00:03
2023-10-02T20:54:14,731 | INFO | onos-of-channel-handler-runtime-status-0 | DeviceManager
cknowledged for device of:00000000000000000000000000000000
```

Part 4: Trace ReactiveForwarding

When we start the Mininet default topology and let h1 send a ICMP ping to h2, h1 will first lookup the MAC address of h2.

We can see a OFPT_PACKET_IN and OFPT_PACKET_OUT pair for ARP request, which is 10.0.0.1 asking who has 10.0.0.2.

Then h2 respond to the ARP request, along with its MAC address, generate another pair of OFPT_PACKET_IN and OFPT_PACKET_OUT packets.

Once the MAC address of h2 is known, h1 will send out an ICMP packet (with OFPT_PACKET_IN in Wireshark), and the packet will be forwarded to h2.

Inside the ReactiveForwarding program, when it received the ARP request from h1, since the destination is 00:00:00:00:00:00, it flood the packet to all ports.

It will add a rule (`IN_PORT = 1, ETH_SRC = h1, ETH_DST = h2`) → `OUT_PORT = 2` to flow table. And add the rule for opposite direction for packets send from h2 to h1.

Part 5: What we learned from this project

When I am doing Part 2-2 writing flow rule for IPv4, I was stuck because not fulfilled the “dependency”. I thought OpenFlow engine will warn me or reject the rule if I didn’t meet the requirement.

It take me a long time to realized it could be wrong, then follow the hint in the slide p.36 to add `ETH_TYPE` and other fields.