111550139

郭芷安

SDN LAB2 report

Part1 answer questions

1. How many OpenFlow headers with type “OFPT_FLOW_MOD” and command “OFPFC_ADD” are there among all the packets?
2. What are the match fields and the corresponding actions in each “OFPT_FLOW_MOD” message?
3. What are the Idle Timeout values for all flow rules on s1 in GUI?

Ans:

There are 6 OpenFlow headers with type “OFPT_FLOW_MOD” and command “OFPFC_ADD” are there among all the packets. The following pictures is the screenshots of 6 headers, and the table of their match field, action and timeout values.

Observe the detail of each packet, and compare with the information shown in the ONOS GUI, we can find that headers 1-3 are related to the core application, headers 4 are related to the fwd application, and the last 2 headers are related to the ping command.

table:

Match fields	actions	timeout
ETH_TYPE:lldp	OUTPUT:CONTROLLER	0
ETH_TYPE:bddp	OUTPUT:CONTROLLER	0
ETH_TYPE:arp	OUTPUT:CONTROLLER	0
ETH_TYPE:ipv4	OUTPUT:CONTROLLER	0
IN_PORT:1, ETH_DST:46:81:F6:80:8F:52, ETH_SRC:BE:A0:C5:3A:EA:AF	OUTPUT:2	0
IN_PORT:2, ETH_DST:E:A0:C5:3A:EA:AF, ETH_SRC:46:81:F6:80:8F:52	OUTPUT:1	0

Screenshots:

a. header 1:

```
OpenFlow 1.4
  Version: 1.4 (0x05)
  Type: OFPT_FLOW_MOD (14)
  Length: 96
  Transaction ID: 53
  Cookie: 0x000100009465555a
  Cookie mask: 0x0000000000000000
  Table ID: 0
  Command: OFPFC_ADD (0)
  Idle timeout: 0
  Hard timeout: 0
  Priority: 40000
  Buffer ID: OFP_NO_BUFFER (4294967295)
  Out port: OFPP_ANY (4294967295)
  Out group: OFPG_ANY (4294967295)
▸ Flags: 0x0001
  Importance: 0
▾ Match
    Type: OFPMT_OXM (1)
    Length: 10
  ▾ OXM field
      Class: OFPXMC_OPENFLOW_BASIC (0x8000)
      0000 101. = Field: OFPXMT_OFB_ETH_TYPE (5)
      .... ...0 = Has mask: False
      Length: 2
      Value: 802.1 Link Layer Discovery Protocol (LLDP) (0x88cc)
    Pad: 000000000000
```

```
Action
  Type: OFPAT_OUTPUT (0)
  Length: 16
  Port: OFPP_CONTROLLER (4294967293)
  Max length: OFPCML_NO_BUFFER (65535)
  Pad: 000000000000
```

b. header 2:

```
OpenFlow 1.4
  Version: 1.4 (0x05)
  Type: OFPT_FLOW_MOD (14)
  Length: 96
  Transaction ID: 55
  Cookie: 0x000100007a585b6f
  Cookie mask: 0x0000000000000000
  Table ID: 0
  Command: OFPFC_ADD (0)
  Idle timeout: 0
  Hard timeout: 0
  Priority: 40000
  Buffer ID: OFP_NO_BUFFER (4294967295)
  Out port: OFPP_ANY (4294967295)
  Out group: OFPG_ANY (4294967295)
▸ Flags: 0x0001
  Importance: 0
▾ Match
    Type: OFPMT_OXM (1)
    Length: 10
  ▾ OXM field
      Class: OFPXMC_OPENFLOW_BASIC (0x8000)
      0000 101. = Field: OFPXMT_OFB_ETH_TYPE (5)
      .... ...0 = Has mask: False
      Length: 2
      Value: Unknown (0x8942)
    Pad: 000000000000
```

```
Action
  Type: OFPAT_OUTPUT (0)
  Length: 16
  Port: OFPP_CONTROLLER (4294967293)
  Max length: OFPCML_NO_BUFFER (65535)
  Pad: 000000000000
```

c. header 3:

```
OpenFlow 1.4
  Version: 1.4 (0x05)
  Type: OFPT_FLOW_MOD (14)
  Length: 96
  Transaction ID: 54
  Cookie: 0x00010000ea6f4b8e
  Cookie mask: 0x0000000000000000
  Table ID: 0
  Command: OFPFC_ADD (0)
  Idle timeout: 0
  Hard timeout: 0
  Priority: 40000
  Buffer ID: OFP_NO_BUFFER (4294967295)
  Out port: OFPP_ANY (4294967295)
  Out group: OFPG_ANY (4294967295)
▸ Flags: 0x0001
  Importance: 0
▾ Match
    Type: OFPMT_OXM (1)
    Length: 10
  ▾ OXM field
      Class: OFPXMC_OPENFLOW_BASIC (0x8000)
      0000 101. = Field: OFPXMT_OFB_ETH_TYPE (5)
      .... ...0 = Has mask: False
      Length: 2
      Value: ARP (0x0806)
    Pad: 000000000000
```

```
Action
  Type: OFPAT_OUTPUT (0)
  Length: 16
  Port: OFPP_CONTROLLER (4294967293)
  Max length: OFPCML_NO_BUFFER (65535)
  Pad: 000000000000
```

d. header 4:

```
OpenFlow 1.4
  Version: 1.4 (0x05)
  Type: OFPT_FLOW_MOD (14)
  Length: 96
  Transaction ID: 0
  Cookie: 0x00010000021b41dc
  Cookie mask: 0x0000000000000000
  Table ID: 0
  Command: OFPFC_ADD (0)
  Idle timeout: 0
  Hard timeout: 0
  Priority: 5
  Buffer ID: OFP_NO_BUFFER (4294967295)
  Out port: OFPP_ANY (4294967295)
  Out group: OFPG_ANY (4294967295)
▸ Flags: 0x0001
  Importance: 0
▾ Match
    Type: OFPMT_OXM (1)
    Length: 10
  ▾ OXM field
      Class: OFPXMC_OPENFLOW_BASIC (0x8000)
      0000 101. = Field: OFPXMT_OFB_ETH_TYPE (5)
      .... ...0 = Has mask: False
      Length: 2
      Value: IPv4 (0x0800)
    Pad: 000000000000
```

```
Action
  Type: OFPAT_OUTPUT (0)
  Length: 16
  Port: OFPP_CONTROLLER (4294967293)
  Max length: OFPCML_NO_BUFFER (65535)
  Pad: 000000000000
```

e. header 5:

```
Type: OFPT_FLOW_MOD (14)
Length: 104
Transaction ID: 59
Cookie: 0x007d00009798f80f
Cookie mask: 0x0000000000000000
Table ID: 0
Command: OFPFC_ADD (0)
Idle timeout: 0
Hard timeout: 0
Priority: 10
Buffer ID: OFP_NO_BUFFER (4294967295)
Out port: OFPP_ANY (4294967295)
Out group: OFPG_ANY (4294967295)
Flags: 0x0001
Importance: 0
Match
  Type: OFPMT_OXM (1)
  Length: 32
▾ OXM field
    Class: OFPXMC_OPENFLOW_BASIC (0x8000)
    0000 000. = Field: OFPXMT_OFB_IN_PORT (0)
    .... ...0 = Has mask: False
    Length: 4
    Value: 1
▾ OXM field
    Class: OFPXMC_OPENFLOW_BASIC (0x8000)
    0000 011. = Field: OFPXMT_OFB_ETH_DST (3)
    .... ...0 = Has mask: False
    Length: 6
    Value: 46:81:f6:80:8f:52 (46:81:f6:80:8f:52)
▾ OXM field
    Class: OFPXMC_OPENFLOW_BASIC (0x8000)
    0000 100. = Field: OFPXMT_OFB_ETH_SRC (4)
    .... ...0 = Has mask: False
    Length: 6
    Value: be:a0:c5:3a:ea:af (be:a0:c5:3a:ea:af)
```

```
Action
  Type: OFPAT_OUTPUT (0)
  Length: 16
  Port: 2
  Max length: 0
  Pad: 000000000000
```

f. header 6:

```
Type: OFPT_FLOW_MOD (14)
Length: 104
Transaction ID: 60
Cookie: 0x007d00002a5fa04f
Cookie mask: 0x0000000000000000
Table ID: 0
Command: OFPFC_ADD (0)
Idle timeout: 0
Hard timeout: 0
Priority: 10
Buffer ID: OFP_NO_BUFFER (4294967295)
Out port: OFPP_ANY (4294967295)
Out group: OFPG_ANY (4294967295)
Flags: 0x0001
Importance: 0
Match
  Type: OFPMT_OXM (1)
  Length: 32
▾ OXM field
    Class: OFPXMC_OPENFLOW_BASIC (0x8000)
    0000 000. = Field: OFPXMT_OFB_IN_PORT (0)
    .... ...0 = Has mask: False
    Length: 4
    Value: 2
▾ OXM field
    Class: OFPXMC_OPENFLOW_BASIC (0x8000)
    0000 011. = Field: OFPXMT_OFB_ETH_DST (3)
    .... ...0 = Has mask: False
    Length: 6
    Value: be:a0:c5:3a:ea:af (be:a0:c5:3a:ea:af)
▾ OXM field
    Class: OFPXMC_OPENFLOW_BASIC (0x8000)
    0000 100. = Field: OFPXMT_OFB_ETH_SRC (4)
    .... ...0 = Has mask: False
    Length: 6
    Value: 46:81:f6:80:8f:52 (46:81:f6:80:8f:52)
```

```
Action
  Type: OFPAT_OUTPUT (0)
  Length: 16
  Port: 1
  Max length: 0
  Pad: 000000000000
```

Part2 take screenshots of arping/ping result

Arping result:

```
mininet> h1 arping h2
ARPING 10.0.0.2
42 bytes from 22:db:86:f5:40:00 (10.0.0.2): index=0 time=1.490 msec
42 bytes from 22:db:86:f5:40:00 (10.0.0.2): index=1 time=7.312 usec
42 bytes from 22:db:86:f5:40:00 (10.0.0.2): index=2 time=6.550 usec
42 bytes from 22:db:86:f5:40:00 (10.0.0.2): index=3 time=6.656 usec
42 bytes from 22:db:86:f5:40:00 (10.0.0.2): index=4 time=6.738 usec
^C
--- 10.0.0.2 statistics ---
5 packets transmitted, 5 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 0.007/0.304/1.490/0.593 ms
```

Ping result:

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=27.3 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.052 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.059 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4065ms
rtt min/avg/max/mdev = 0.052/5.516/27.344/10.913 ms
mininet> h2 ping h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.063 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.061 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.063 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.047 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.061 ms
^C
--- 10.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4092ms
rtt min/avg/max/mdev = 0.047/0.059/0.063/0.006 ms
```

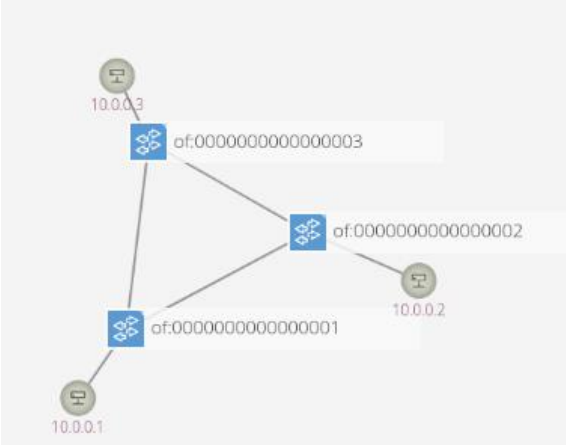
Part3 take screenshots and answer the questions

CPU utilization:

```
top - 22:40:37 up 4 days,  5:47,  2 users,  load average: 2.26, 1.67, 1.13
Tasks: 453 total,   8 running, 443 sleeping,   0 stopped,   2 zombie
%Cpu(s):  1.5 us,  1.2 sy,  0.0 ni, 27.8 id,  0.2 wa,  0.0 hi, 69.3 si,  0.0 st
MiB Mem :   3868.1 total,    230.3 free,   2879.7 used,    758.2 buff/cache
MiB Swap:   3220.0 total,   1212.0 free,   2008.0 used.    650.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
     24 root      20   0       0      0      0 R 100.0   0.0   0:21.94 ksoftirqd/1
     42 root      20   0       0      0      0 R 100.0   0.0   0:11.69 ksoftirqd/4
     36 root      20   0       0      0      0 R  99.7   0.0   0:14.75 ksoftirqd/3
     60 root      20   0       0      0      0 R  99.0   0.0   0:44.57 ksoftirqd/7
     30 root      20   0       0      0      0 R  98.4   0.0   0:08.62 ksoftirqd/2
     54 root      20   0       0      0      0 R  98.1   0.0   0:21.00 ksoftirqd/6
   8066 eji530    20   0 5389832 186236  72372 S   9.1   4.7  37:50.72 gnome-shell
  52118 eji530    20   0 8052780 775548  12288 S   5.5  19.6  94:45.81 java
  19748 eji530    20   0 3485088 306648 119588 S   4.9   7.7  45:43.87 firefox
   1307 root      10 -10  892884 107396   5888 S   2.9   2.7  12:57.27 ovs-vswitchd
    838 systemd+  20   0   14836   3712   3456 S   1.9   0.1   8:51.98 systemd-oomd
    932 root      20   0  745324  10940   9048 S   1.6   0.3   0:38.92 NetworkManag+
  43623 eji530    20   0  608136  47288  27628 S   1.6   1.2   1:34.89 gnome-termin+
    880 root      20   0  317832   5888   4736 S   1.0   0.1  17:34.79 vmtoolsd
   8348 eji530    20   0  216840  18992  12700 S   1.0   0.5  15:54.20 vmtoolsd
  20016 eji530    20   0 2666144 117016  52208 S   1.0   3.0  39:01.87 Isolated Web+
 111529 root      20   0  611076  37648  26112 S   1.0   1.0   1:59.51 noble
 139330 eji530    20   0   13776   4480   3456 R   1.0   0.1   0:00.41 top
   8228 eji530    20   0  315860   7424   5760 S   0.6   0.2   1:02.28 ibus-daemon
   8424 eji530    20   0  163920   6144   5760 S   0.6   0.2   0:19.10 ibus-engine-+
     16 root      20   0       0      0      0 S   0.3   0.0   0:01.13 ksoftirqd/0
```

To create broadcast storm, I add 3 switches to this topology, connect them as a circle, and add a host to each switch, which is shown in the below picture. And install flow rules to each switch to let them flood the packet to their neighbors.

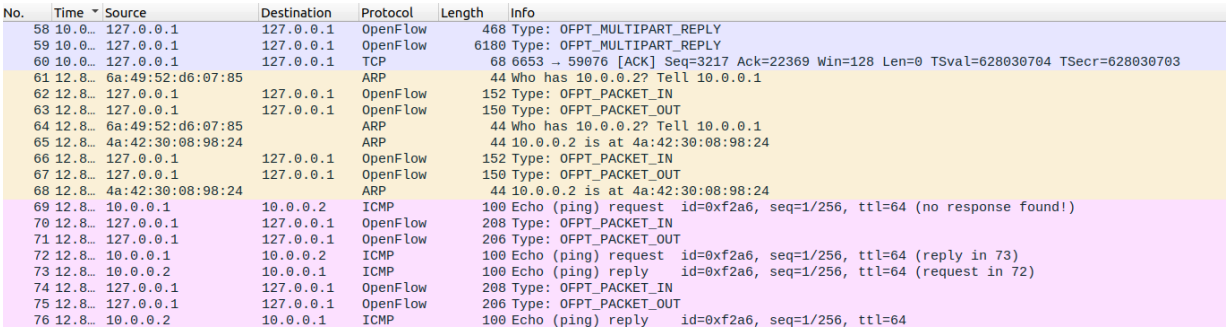


After setting up all the topology and flow rules, command h1 arping h2 in the CLI to observe the result. We can find that the number of packets increase rapidly overtime in the GUI, and the CPU utilization is merely 100%.

This is because that h1 will send an arp broadcast packet first, and after s1 receives the packet, it will broadcast the packet to s2 and s3. When s2 receives the packet, it will broadcast the packet to s1 and s3 too, etc. the packet will be broadcasted again and again, and leads to broadcast storm.

Part4 Write down what I have observed step by step

After all the preparations to start ONOS and mininet, command h1 ping h2 -c 5 in the mininet CLI, then capture the packets using wireshark “any” interface. The following are the packets I observed in order of the time.



No.	Time	Source	Destination	Protocol	Length	Info
58	10.0…	127.0.0.1	127.0.0.1	OpenFlow	468	Type: OFPT_MULTIPART_REPLY
59	10.0…	127.0.0.1	127.0.0.1	OpenFlow	6180	Type: OFPT_MULTIPART_REPLY
60	10.0…	127.0.0.1	127.0.0.1	TCP	68	6653 → 59076 [ACK] Seq=3217 Ack=22369 Win=128 Len=0 TSval=628030704 TSecr=628030703
61	12.8…	6a:49:52:d6:07:85		ARP	44	Who has 10.0.0.2? Tell 10.0.0.1
62	12.8…	127.0.0.1	127.0.0.1	OpenFlow	152	Type: OFPT_PACKET_IN
63	12.8…	127.0.0.1	127.0.0.1	OpenFlow	150	Type: OFPT_PACKET_OUT
64	12.8…	6a:49:52:d6:07:85		ARP	44	Who has 10.0.0.2? Tell 10.0.0.1
65	12.8…	4a:42:30:08:98:24		ARP	44	10.0.0.2 is at 4a:42:30:08:98:24
66	12.8…	127.0.0.1	127.0.0.1	OpenFlow	152	Type: OFPT_PACKET_IN
67	12.8…	127.0.0.1	127.0.0.1	OpenFlow	150	Type: OFPT_PACKET_OUT
68	12.8…	4a:42:30:08:98:24		ARP	44	10.0.0.2 is at 4a:42:30:08:98:24
69	12.8…	10.0.0.1	10.0.0.2	ICMP	100	Echo (ping) request id=0xf2a6, seq=1/256, ttl=64 (no response found!)
70	12.8…	127.0.0.1	127.0.0.1	OpenFlow	208	Type: OFPT_PACKET_IN
71	12.8…	127.0.0.1	127.0.0.1	OpenFlow	206	Type: OFPT_PACKET_OUT
72	12.8…	10.0.0.1	10.0.0.2	ICMP	100	Echo (ping) request id=0xf2a6, seq=1/256, ttl=64 (reply in 73)
73	12.8…	10.0.0.2	10.0.0.1	ICMP	100	Echo (ping) reply id=0xf2a6, seq=1/256, ttl=64 (request in 72)
74	12.8…	127.0.0.1	127.0.0.1	OpenFlow	208	Type: OFPT_PACKET_IN
75	12.8…	127.0.0.1	127.0.0.1	OpenFlow	206	Type: OFPT_PACKET_OUT
76	12.8…	10.0.0.2	10.0.0.1	ICMP	100	Echo (ping) reply id=0xf2a6, seq=1/256, ttl=64

1. In packet 61, h1 want to send echo ping request to h2, but h2’s address is not in its ARP table, so h1 broadcast an ARP request to ask who has the MAC address of h2.
2. Observe packet 62. After s1 receive h1’s ARP request, s1 doesn’t know the MAC address of h2 neither, so s1 send a PACKET_IN message to the control plane.
3. Observe packet 63. Control plane doesn’t have information neither, so the control plane FLOOD the PACKET_OUT message to data plane.
4. Observe packet 65. h2 receive the ARP request and tell h1 where it is. This packet will be received by s1 and be sent to h1.
5. In packet 66, s1 send this information to the control plane
6. In packet 67, the control plane installs the flow rule on s1. Now s1 knows the path between h1 and h2.
7. In packet 69. h1 send the echo ping request to h2, and this will be received by s1 first.
8. In packet 70. Although s1 knows how to forward this packet to h2, but the recent flow rule only tells s1 how to handle the packet when it is using the ARP protocol, so s1 had to send a PACKET_IN message to control plane first to ask for the specific rule to handle packet using ICMP protocol.
9. In packet 71. The control plane sent a PACKET_OUT message to s1 to install flow rule on it. Afterwards, h2 received the echo request sent from h1.
10. In packet 73. After h2 receive the echo request, it want to send an echo reply

to h1.

11. In packet 74. After s1 receive the echo reply, it had to send a PACKET_IN message to the control plane due to the same reason in point 8.
12. In packet 75. The control plane sent a PACKET_OUT message the s1 to install flow rule on it. After that, h1 receive the echo reply from h2 via s1.

- Some packets are ignored in this discussion because they are retransmitted packets.

Overall, the control plane operations are:

Application-fwd in ONOS receive the packet from a host -> examine whether there is a pre-existing flow rule for the specific protocol and destination -> if there are no pre-existing flow rules, the application will generate a rule to install on the switch.

And the data plane operations are:

After a switch receive a packet from a host -> the switch match the packets against the flow rule installed by the controller -> if there are no preinstalled rules, then send PACKET_IN message to the controller. And if the rule exist, forward the packets follow the rule.

Part5 what I've learned / solved

1. Learned how to install customized flow rule.
2. Learned how the broadcast storm is created, and the concept of it.
3. Learned how to understand the meaning of each packet captured using wireshark.
4. In part 4, it took me a lot of time to make through all the meanings of a packet, and why these packets exist, what are their character in the interaction between data plane and control plane. After I make through the concept, I feel that I am much clearer about the operations in data plane, control plane, and the interaction between them.