

SDN-NFV

lab2-report

111550120 吳盈庭

Part1: Answer the Questions

1. How many OpenFlow headers with type “OFPT_FLOW_MOD” and command “OFPPC_ADD” are there among all the packets?

As shown below, there are 2 distinct OpenFlow headers with type “OFPT_FLOW_MOD” (type==14) and command “OFPPC_ADD”.

<pre> 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: OFPPC_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: OFPXM_OPENFLOW_BASIC (0x8000) 0000 101. = Field: OFPXM_OFB_ETH_TYPE (5)0 = Has mask: False Length: 2 Value: IPv4 (0x0800) Pad: 0000000000000000 Instruction Instruction </pre>	<pre> OpenFlow 1.4 Version: 1.4 (0x05) Type: OFPT_FLOW_MOD (14) Length: 104 Transaction ID: 79 Cookie: 0x0051000033c8565a Cookie mask: 0x0000000000000000 Table ID: 0 Command: OFPPC_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: OFPXM_OPENFLOW_BASIC (0x8000) 0000 000. = Field: OFPXM_OFB_IN_PORT (0)0 = Has mask: False Length: 4 Value: 2 OXM field Class: OFPXM_OPENFLOW_BASIC (0x8000) 0000 011. = Field: OFPXM_OFB_ETH_DST (3)0 = Has mask: False Length: 6 Value: 0e:1e:28:95:bc:66 (0e:1e:28:95:bc:66) OXM field Class: OFPXM_OPENFLOW_BASIC (0x8000) </pre>
---	---

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?

Match fields	Actions	Timeout values
ETH_TYPE (5) = IPv4 OFPXM_OFB_ETH_TYPE (5) = IPv4	OUTPUT (0) port = OFPP_CONTROLLER (4294967293) OFPAT_OUTPUT (0) = OFPP_CONTROLLER (4294967293)	0

IN_PORT (0) = 1 ETH_DST (3) = f2:03:65:f3:1f ETH_SRC (4) = 0e:1e:28:95:bc:66 OFPXMT_OFB_IN_PORT (0) = 1 OFPXMT_OFB_ETH_DST (3) = f2:03:65:f3:1f OFPXMT_OFB_ETH_SRC (4) = 0e:1e:28:95:bc:66	OUTPUT (0) port = 1 OFPFT_OUTPUT (0) = 1	0
--	---	---

Part2: Install Flow Rules

```
ytw@ytw-ubuntu: ~/Desktop
mininet> h1 arping h2
ARPING 10.0.0.2
42 bytes from a2:c2:dd:c5:1a:51 (10.0.0.2): index=0 time=12.223 msec
42 bytes from a2:c2:dd:c5:1a:51 (10.0.0.2): index=1 time=60.000 usec
42 bytes from a2:c2:dd:c5:1a:51 (10.0.0.2): index=2 time=274.002 usec
42 bytes from a2:c2:dd:c5:1a:51 (10.0.0.2): index=3 time=200.002 usec
42 bytes from a2:c2:dd:c5:1a:51 (10.0.0.2): index=4 time=198.002 usec
42 bytes from a2:c2:dd:c5:1a:51 (10.0.0.2): index=5 time=107.001 usec
42 bytes from a2:c2:dd:c5:1a:51 (10.0.0.2): index=6 time=200.002 usec
42 bytes from a2:c2:dd:c5:1a:51 (10.0.0.2): index=7 time=194.001 usec
42 bytes from a2:c2:dd:c5:1a:51 (10.0.0.2): index=8 time=223.002 usec
42 bytes from a2:c2:dd:c5:1a:51 (10.0.0.2): index=9 time=230.002 usec
^C
--- 10.0.0.2 statistics ---
10 packets transmitted, 10 packets received, 0% unanswered (0 extra)
rtt min/avg/max/std-dev = 0.060/1.391/12.223/3.611 ms
mininet>
```

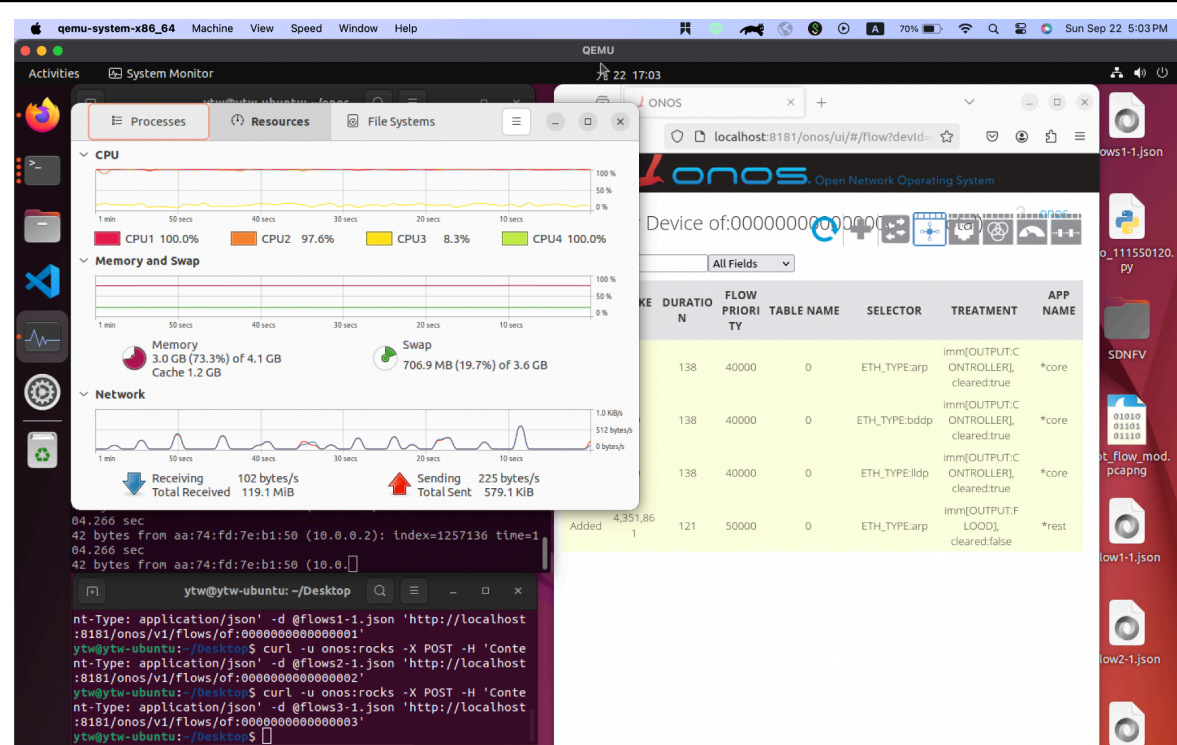
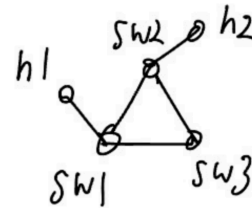
```
ytw@ytw-ubuntu: ~/Desktop
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=10.3 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.399 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.347 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.340 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=2.46 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=1.30 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.385 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.346 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=2.28 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.611 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=2.23 ms
^C
--- 10.0.0.2 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10069ms
rtt min/avg/max/mdev = 0.340/1.912/10.330/2.787 ms
mininet>
```

Part3: Create Topology with Broadcast Storm

I created the topology on the right and added flow rules below to the three switches:

When a switch receives a packet, it will FLOOD, and because the three switches create a loop, the switches continually “FLOOD” the packet, making the amount of packets grow exponentially and using up the bandwidth, causing a broadcast storm

As shown below, when “h1 arping h2”, a broadcast storm happens and the CPU usage hits 100%.



Part4: Trace ReactiveForwarding

control plane:

1. The ICMP packet (ping request) will reach the switch, received by the reactive forwarding application
2. The application finds that there are no existing flow rule to handle the request, so it sends a `PACKET_IN` message to the ONOS controller
3. The ONOS controller will decide the forwarding path and install the flow rule in the switch to handle further ICMP packets. This flow rule is sent from the controller to the switch via `FLOW_MOD` messages

data plane:

1. After the switch has the new flow rules, it can forward the subsequent ICMP packets directly between h1 and h2 without involving the controller.

step-by-step observation:

1. **ICMP Request** (h1 ping h2): The switch doesn't know how to handle, so it sends a packet-in message to the controller
2. **Flow Rule Installation:** ONOS responds by adding flow rules using `FLOW_MOD`
3. **ICMP Reply (from h2):** The switch forwards the reply based on the installed flow rule.
4. **Subsequent ICMP Requests or Replies:** After the flow rules are installed, subsequent pings are handled directly by the switch without contacting the controller (data plane forwarding).

Part5: What I Learned or Solved

I learned to install and delete flow rules via 'curl', what causes a broadcast storm, and how ovs communicates with the controller. I am also now more familiar with wireshark and openflow protocol.