2023 SDN-NFV Project2

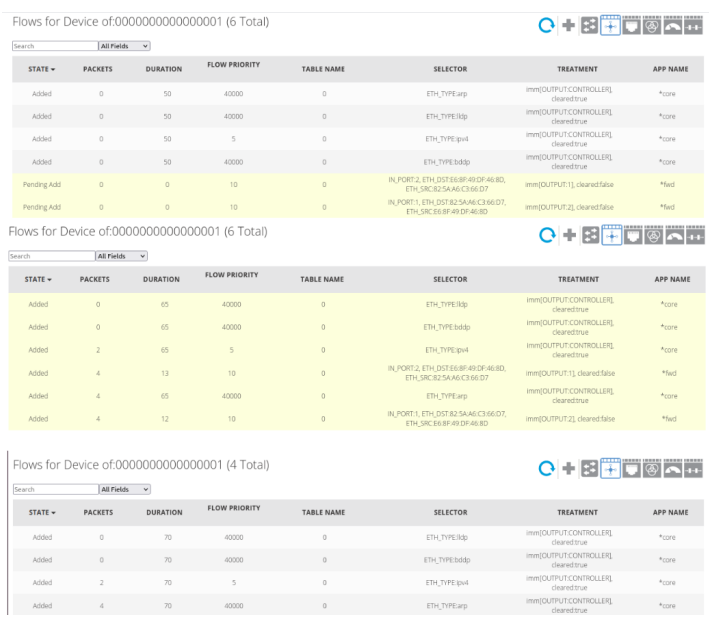
312552017 董家鳴

Part1 : Answer Questions

1. How many OpenFlow headers with the type "OFPT_FLOW_MOD" and command "OFPFC_ADD" are there among all the packets? **Ans: 6**, and the **figure 4** answers **2.** match fields and the corresponding actions in each "OFPT_FLOW_MOD" message and **3.** Idle Timeout values for all flow rules on s1 in GUI.

Steps about part1 experiment:

- 1. After executing the command "h1 ping h2 -c 5", the GUI shows the newly pending added flow rules. Not long after the ping ended, the forwarding rules also disappeared. Figure 1 shows the process.
- 2. While pinging I observed the idle timeout values for all flow rules on GUI, figure 2 shows an example.
- 3. I collected some packets with headers of type "OFPT_FLOW_MOD" and command "OFPFC_ADD" on wireshark, Figure 3 is one of them.
- 4. Finally, I have organized the table in Figure 4. Note that the Unknown type can be found in `utils\misc\src\main\java\org\onlab\packet\EthType.java`



Flows for Device of:0000000000000001 (6 Total)

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	50	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	50	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	50	5	0	ETH_TYPE:ipv4	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	50	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Pending Add	0	0	10	0	IN_PORT:2, ETH_DST:E6:8F:49:DF:46:8D, ETH_SRC:82:5A:A6:C3:66:D7	imm[OUTPUT:1], cleared:false	*fwd
Pending Add	0	0	10	0	IN_PORT:1, ETH_DST:82:5A:A6:C3:66:D7, ETH_SRC:E6:8F:49:DF:46:8D	imm[OUTPUT:2], cleared:false	*fwd

Flows for Device of:0000000000000001 (6 Total)

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	65	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	65	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	2	65	5	0	ETH_TYPE:ipv4	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	4	13	10	0	IN_PORT:2, ETH_DST:E6:8F:49:DF:46:8D, ETH_SRC:82:5A:A6:C3:66:D7	imm[OUTPUT:1], cleared:false	*fwd
Added	4	65	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	4	12	10	0	IN_PORT:1, ETH_DST:82:5A:A6:C3:66:D7, ETH_SRC:E6:8F:49:DF:46:8D	imm[OUTPUT:2], cleared:false	*fwd

Flows for Device of:0000000000000001 (4 Total)

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	70	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	70	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	2	70	5	0	ETH_TYPE:ipv4	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	4	70	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core

Figure 1: Ping

Figure 2: An example of idle timeout

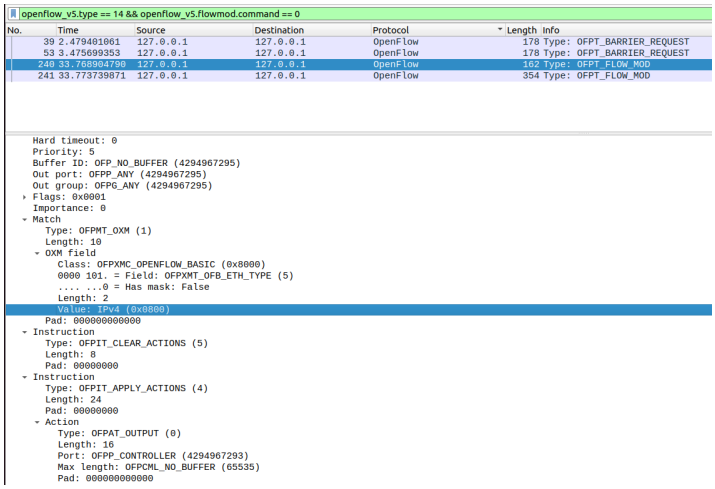


Figure 3: An example of packets

Match fields	Actions	Timeout values
IN_PORT=2	OUTPUT=1	10
IN_PORT=1	OUTPUT=2	10
ETH_TYPE=IPv4	OUTPUT=OFPP_CONTROLLER (4294967293)	0
ETH_TYPE=Unknown (0x8942) (BDDP)	OUTPUT=OFPP_CONTROLLER (4294967293)	0
ETH_TYPE=802.1 Link Layer Discovery Protocol (LLDP) (0x88cc)	OUTPUT=OFPP_CONTROLLER (4294967293)	0
ETH_TYPE=ARP (0x0806)	OUTPUT=OFPP_CONTROLLER (4294967293)	0

Figure 4: Table of OpenFlow packets

Part2 : Install Flow Rules

Write a JSON file of flow rules.

Steps about part2:

- 1. By using *curl*, I first upload the JSON describing the arp flow rule to the switch. And then the *arping* was successfully performed after installing my rule as shown in Figure 5.
- 2. By using *curl*, I first upload the JSON describing the ipv4 flow rule to the switch. And then the *ping* was successfully performed after installing my rule as shown in Figure 6.

Figure 5: arping

Figure 6: Host ping

Part3 : Create Topology with Broadcast Storm

Send packets to a topology that would cause the broadcast storm and observe the CPU's utilization.

Steps about part3:

- 1. Create a topology that may cause a broadcast storm [1]. Figure 7 shows my topology and figure 8 shows the source code. Notice that the switch 1 is connected to switch 2, and the switch 2 is connected to the switch 1.

- 2. However, step one is not enough to cause a broadcast storm. How the switch forward packets is also the key. Figure 9 shows the flow rule that would be installed in two switches.
- 3. By sending the ARP packets from one host to another, we can observe that the VM's CPU utilization is occupied a lot by the mininet in Figure 10.

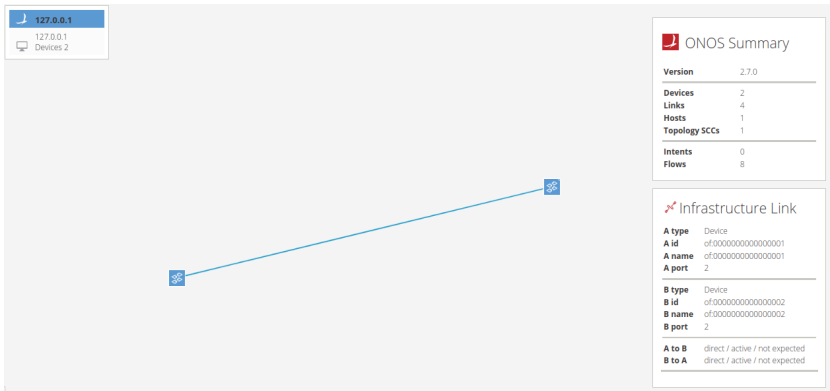


Figure 7: topology

```
from mininet.topo import Topo

class Project2_Topo_312552017( Topo ):
    def __init__( self ):
        Topo.__init__( self )
        # Add hosts
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )

        # Add switches
        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )

        # Add links, host to switch
        self.addLink( h1, s1 )
        self.addLink( h2, s2 )

        # Add links, switch to switch
        self.addLink( s1, s2 )
        self.addLink( s2, s1 )

topos = { 'topo_part3_312552017': Project2_Topo_312552017 }
```

Figure 8: topology source code

Figure 9: flow rule causes broadcast storm

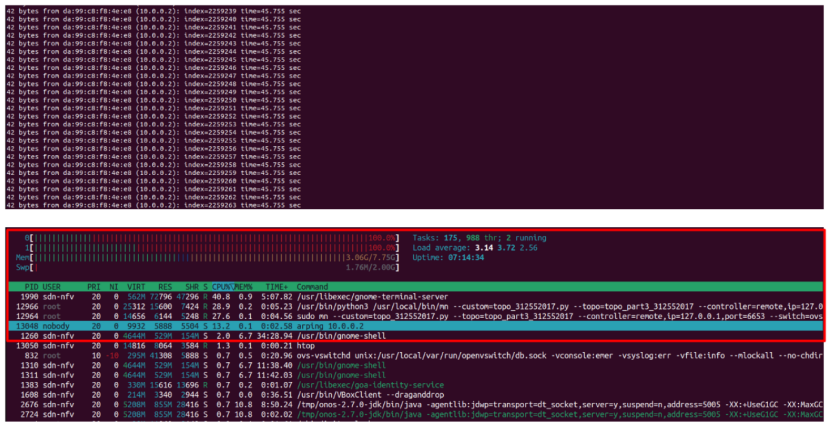


Figure 10: Broadcast storm

Part 4: Trace ReactiveForwarding

1. observe h1 pings h2, ARP and ICMP (Echo) request/response. Figure 11 shows the packets cached in the Wireshark to do the observation.

- 1. h1 send the arp request with type *packet_in* to the controller.
- 2. The controller receives the arp request from h1 and sends a packet with type *packet_out* by flooding (broadcasting) the arp request.
- 3. h2 receives the arp request and sends the arp reply with type *packet_in* to the controller.
- 4. The controller receives the arp reply and sends an arp reply with type *packet_out* by flooding (broadcasting) the arp reply.
- 5. h1 send the ICMP request with type *packet_in* to the controller.
- 6. The controller receives the ICMP request from h1 and sends a packet with type *packet_out* by flooding (broadcasting) the ICMP request.
- 7. h2 receives the ICMP request and sends the ICMP reply with type *packet_in* to the controller.
- 8. The controller receives the ICMP reply and sends an ICMP reply with type *packet_out* by flooding (broadcasting) the ICMP reply.

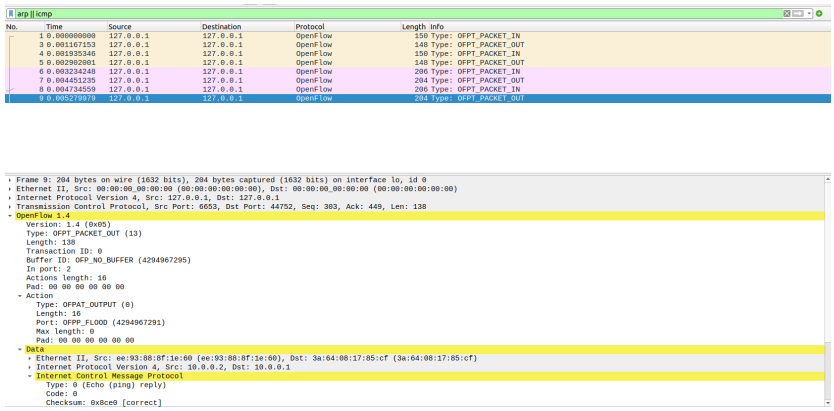


Figure 11: Ping flow - reactive forwarding

2. ONOS ReactiveForwarding application

1. The function *processor* in the class *ReactivePacketProcessor* is mainly responsible for forwarding packets along their paths. The process will first check if some conditions match(e.g., it is deemed a control packet, the host's destination is null or not).
2. Next, in the beginning, the controller didn't know the MAC address of the host it needed to send, so it used the function *flood* to send out the request.
3. Finally, the host will return its MAC address to the controller, and thus it now knows its MAC address so the controller starts to install the flow rule by using the function *installRule*. Therefore, the next time a packet is transmitted, if there is a rule that matches the rule, flooding will not be performed, but the packet will be sent out based on a specific port.

What you' ve learned or solved

1. The format and meaning of OpenFlow packets. For example, match fields indicate which packet the controller received or what protocol type it is. And action is what action needs to be performed.

2. Install Flow Rules, proactive forwarding. Instead of using reactive forwarding, how do we install forwarding directly on the switch (proactive forwarding).

3. Broadcast Storm. If there is a cycle on the switch, some ports must be unused (determined by using the spanning tree protocol) so that packets will not continue to circulate around the cycle.

4. Network Virtualization. The OpenFlow message encapsulates the entire original packet (be it ICMP, TCP, UDP, etc.), which allows the switch to forward the packet to the controller without being treated as a normal network packet by other devices in the physical network, and thus creating multiple virtual networks on top of a single physical network.

References

[1] 胡凱智. 從 *STP* 到 *RSTP* 認識多重生成樹協定. 2018. URL: https://www.netadmin.com.tw/netadmin/zh-tw/technology/4A78F63A33AE44938897A79F625548A8?page=1.