CO515: Advances in Computer Networks: Selected Topics – Lab05 (Mininet Lab 02)

E/19/446: Wijerathna I.M.K.D.I.

02/05/2024

Lab Environment Setup

I used the same setup as the previous lab(Mininet Lab1) which is a lightweight ubuntu os setup which runs over vagrant+virtual box. The following is the lab environment structure.

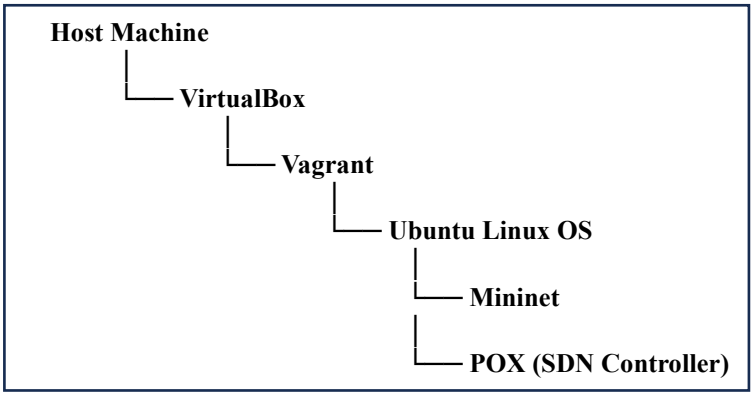


Figure 1: Lab Environment Setup

1. **Use the command line to setup the topology**
   $ sudo mn --topo single,3 --mac --switch ovsk --controller remote

```
vagrant@sdn-box:~$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0
c0
```

Then using a few verification commands, I checked the network configuration of the created topology.

- *IP configurations of the hosts*
    - *It can be observed that the last hexadecimal value of the mac addresses of hosts 1,2, and 3 are 1,2,3 respectively. This is because the –mac flag usage in the mininet command earlier.*

```
mininet> h1 ifconfig
h1-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:01
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::200:ff:fe00:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:648 (648.0 B)  TX bytes:738 (738.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet> h2 ifconfig
h2-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:02
          inet addr:10.0.0.2  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::200:ff:fe00:2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:648 (648.0 B)  TX bytes:738 (738.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet>
```

- *Network connections and nodes*

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0
c0
mininet> nodes
available nodes are:
c0 h1 h2 h3 s1
```

In here, it could be observed that the corresponding network interface cards of each link and how they are connected. The 'nodes' command outputs all the nodes in the network. Which are:

- *SDN controller – c0*
- *Openflow Switch – s1*
- *3 hosts – h1, h2, h3*

Based on the above results from verification commands, the following should be the topology created:

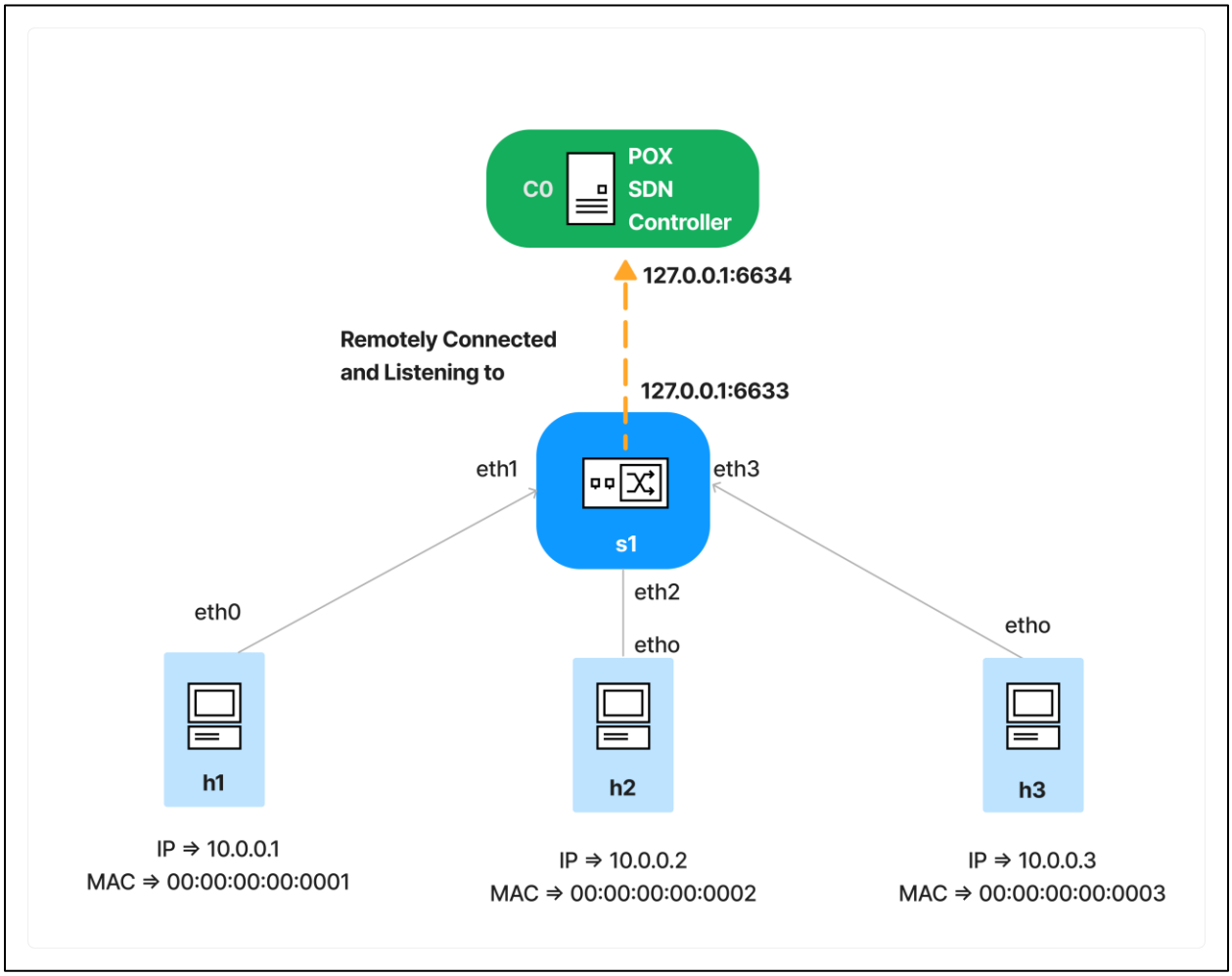


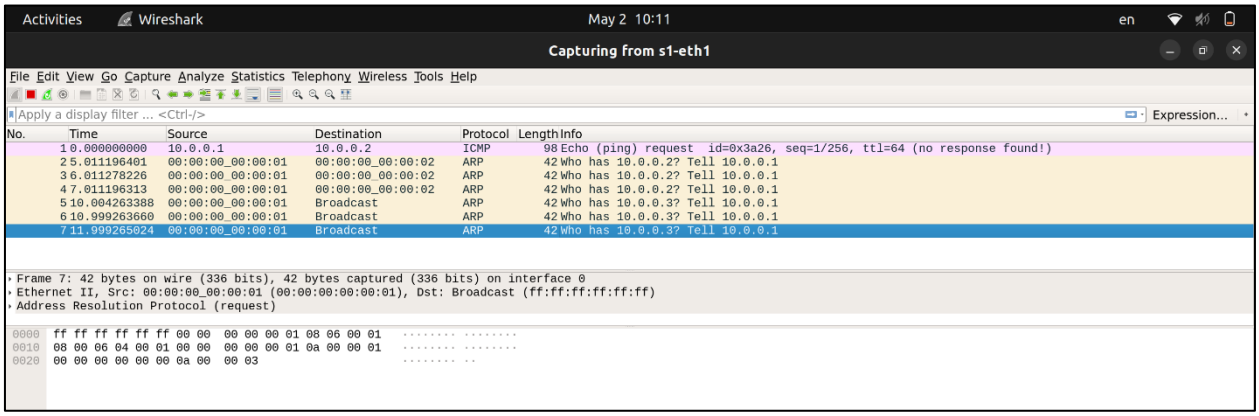
Figure 2: Assumed Topology Upon the Results of Verification Commands

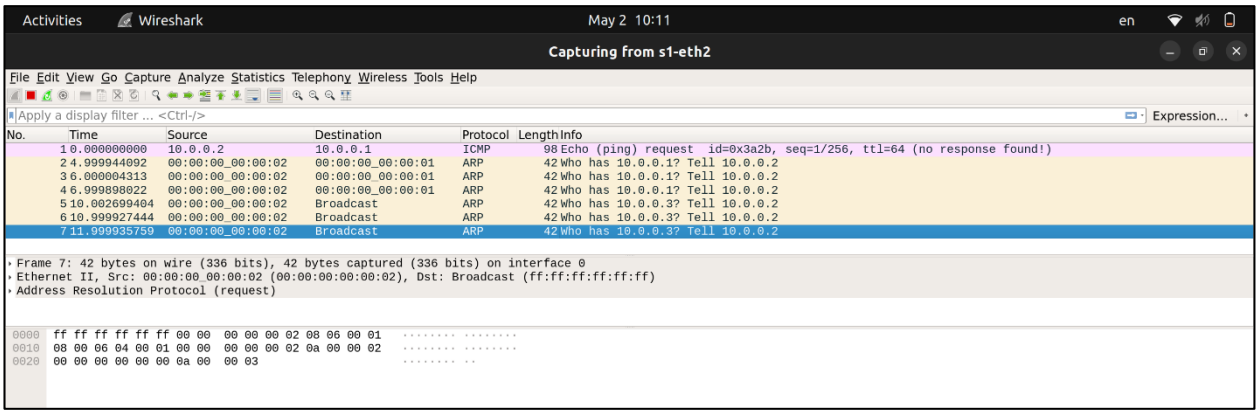
2. Do mininet > pingall

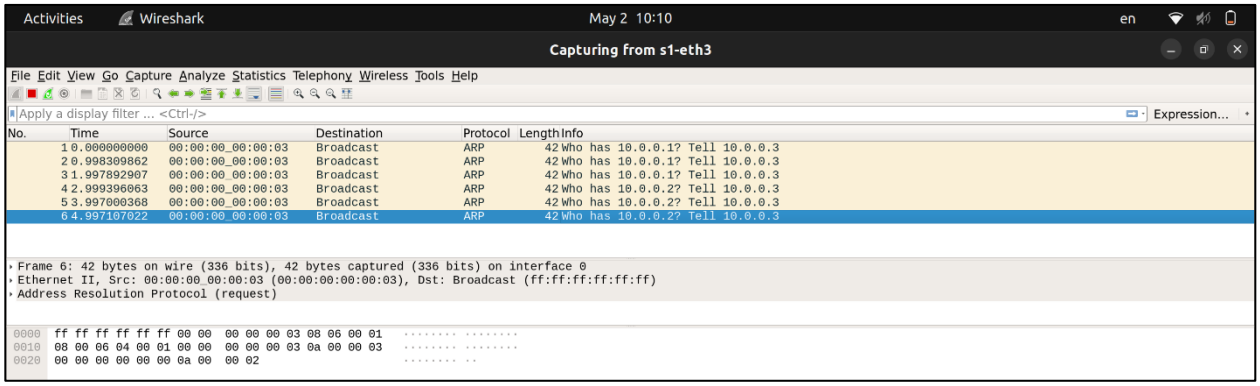
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet>
```

This failed because no Openflow rules at the moment in the Openflow tables.

The following were the wireshark outputs received at Openflow switch's interfaces (s1-eth1, s1-eth2, s1-eh3) while this 'pingall' command was in operation.

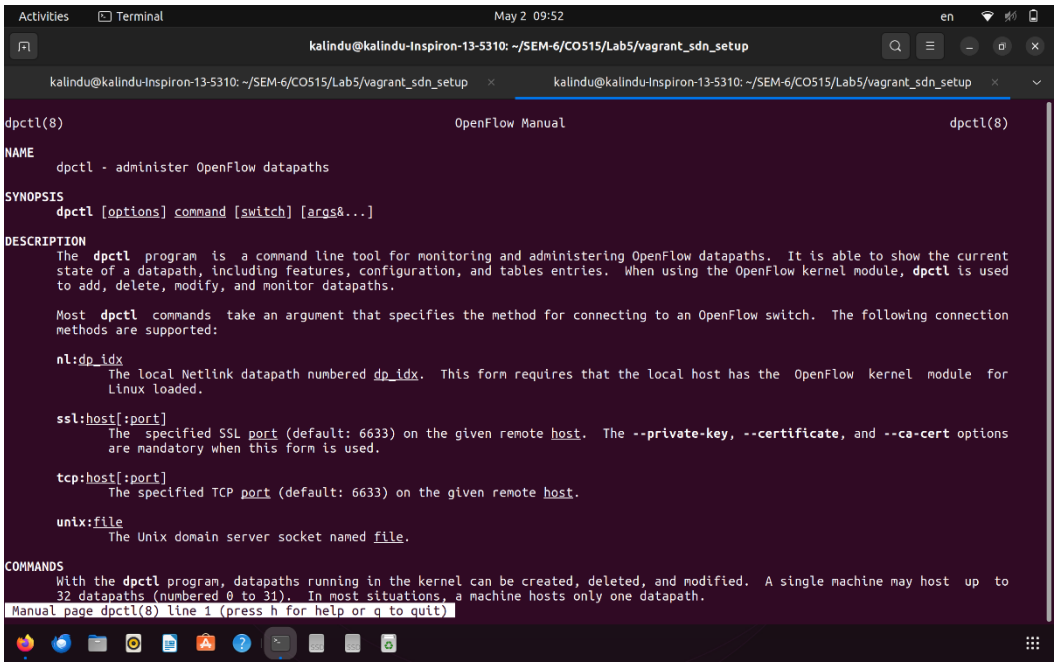




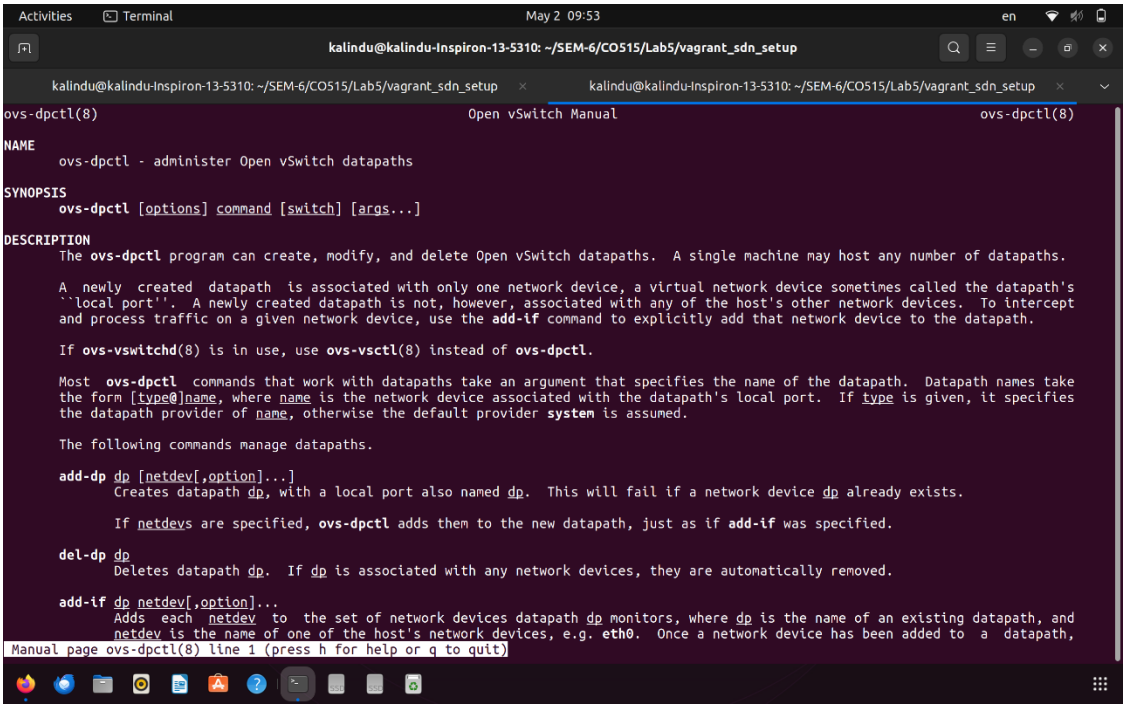


It can be clearly observed that, only a few echo requests have been received along with ARP requests but none of them were replied because connectivity hasn't been configured yet through Openflow rules

3. **Start another window do ‘man dpctl’ and ‘man ovs-dpctl’**



This command shows the instructions on how to monitor Openflow data paths using dpctl commands.



This command shows the instruction on how to configure Openflow rules using ovs-dpctl commands.

Basic functions like add, delete, methods and varies ways to perform new Openflow rules.

4. **$ dpctl show tcp:127.0.0.1:6634**

```
vagrant@sdn-box:~$ dpctl show tcp:127.0.0.1:6634
features_reply (xid=0x4ee0137a): ver:0x1, dpid:1
n_tables:254, n_buffers:256
features: capabilities:0xc7, actions:0xfff
 1(s1-eth1): addr:f2:13:58:7e:c6:79, config: 0, state:0
     current:    10GB-FD COPPER
 2(s1-eth2): addr:a2:62:e7:7d:50:18, config: 0, state:0
     current:    10GB-FD COPPER
 3(s1-eth3): addr:4a:a5:46:f6:8d:25, config: 0, state:0
     current:    10GB-FD COPPER
 LOCAL(s1): addr:1e:6e:bf:38:3a:48, config: 0x1, state:0x1
get_config_reply (xid=0xa11821d1): miss_send_len=0
vagrant@sdn-box:~$
```

This command shows the possible Openflow data paths in the current topology

5. **$dpctl dump-flows tcp:127.0.0.1:6634**

```
vagrant@sdn-box:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x219428eb): flags=none type=1(flow)
vagrant@sdn-box:~$
```

This command shows the existing rules in the current flow table. Since there are no rules in our current switch, the empty table was returned.

6. **Manually configure the OpenFlow switch with dptcl**
   - **$dpctl add-flow tcp:127.0.0.1:6634 in_port=1,idle_timeout=1000,actions=output:2**
   - **$dpctl add-flow tcp:127.0.0.1:6634 in_port=2,idle_timeout=1000,actions=output:1**

```
vagrant@sdn-box:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0xdc694917): flags=none type=1(flow)
vagrant@sdn-box:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=1,idle_timeout=1000,actions=output:2
vagrant@sdn-box:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=2,idle_timeout=1000,actions=output:1
vagrant@sdn-box:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x4dd5936): flags=none type=1(flow)
  cookie=0, duration_sec=12s, duration_nsec=530000000s, table_id=0, priority=32768, n_packets=0, n_bytes=0, idle_timeout=1000,hard_timeout=0,in_port=1,actions=output:2
  cookie=0, duration_sec=3s, duration_nsec=972000000s, table_id=0, priority=32768, n_packets=0, n_bytes=0, idle_timeout=1000,hard_timeout=0,in_port=2,actions=output:1
vagrant@sdn-box:~$
```

After adding the 2 rules, the flow table is now filled with those two new rules, as shown above.These 2 rules should successfully connect h1 and h2.

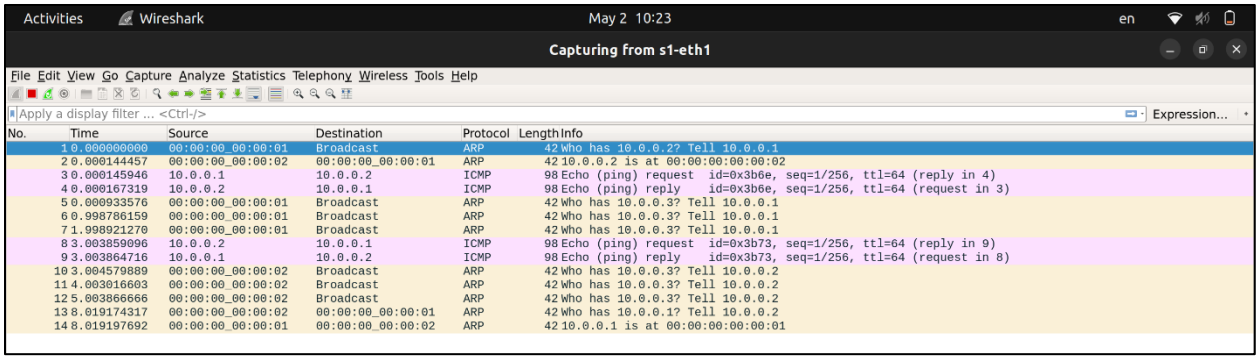
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 X
h3 -> X X
*** Results: 66% dropped (2/6 received)
mininet>
```

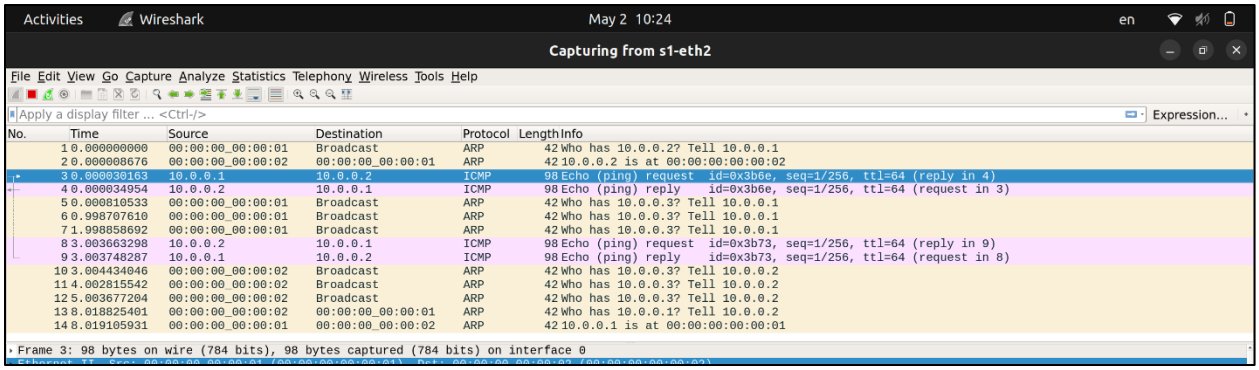
Since only h1 and h2 are connected 'pingall' was only successful between those two hosts.

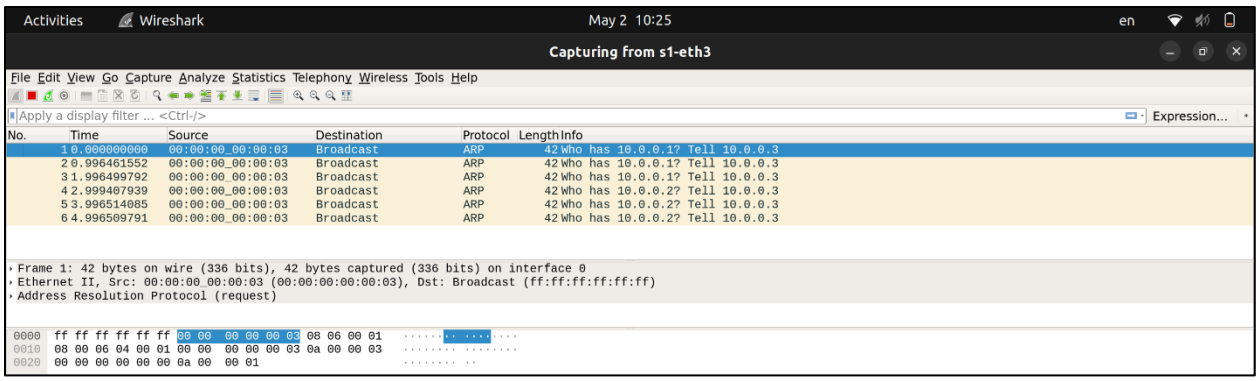
When the flow tables are rechecked after the above 'pingall', it can be observed that 7 packets have been matched those rule while the pingall process.

```
mininet> s1 dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x55c29868): flags=none type=1(flow)
  cookie=0, duration_sec=139s, duration_nsec=586000000s, table_id=0, priority=32768, n_packets=7, n_bytes=406, idle_timeout=1000,hard_timeout=0,in_port=1,actions=output:2
  cookie=0, duration_sec=131s, duration_nsec=28000000s, table_id=0, priority=32768, n_packets=7, n_bytes=406, idle_timeout=1000,hard_timeout=0,in_port=2,actions=output:1
mininet>
```

Wireshark outputs of the observed at the Openflow switch's interfaces







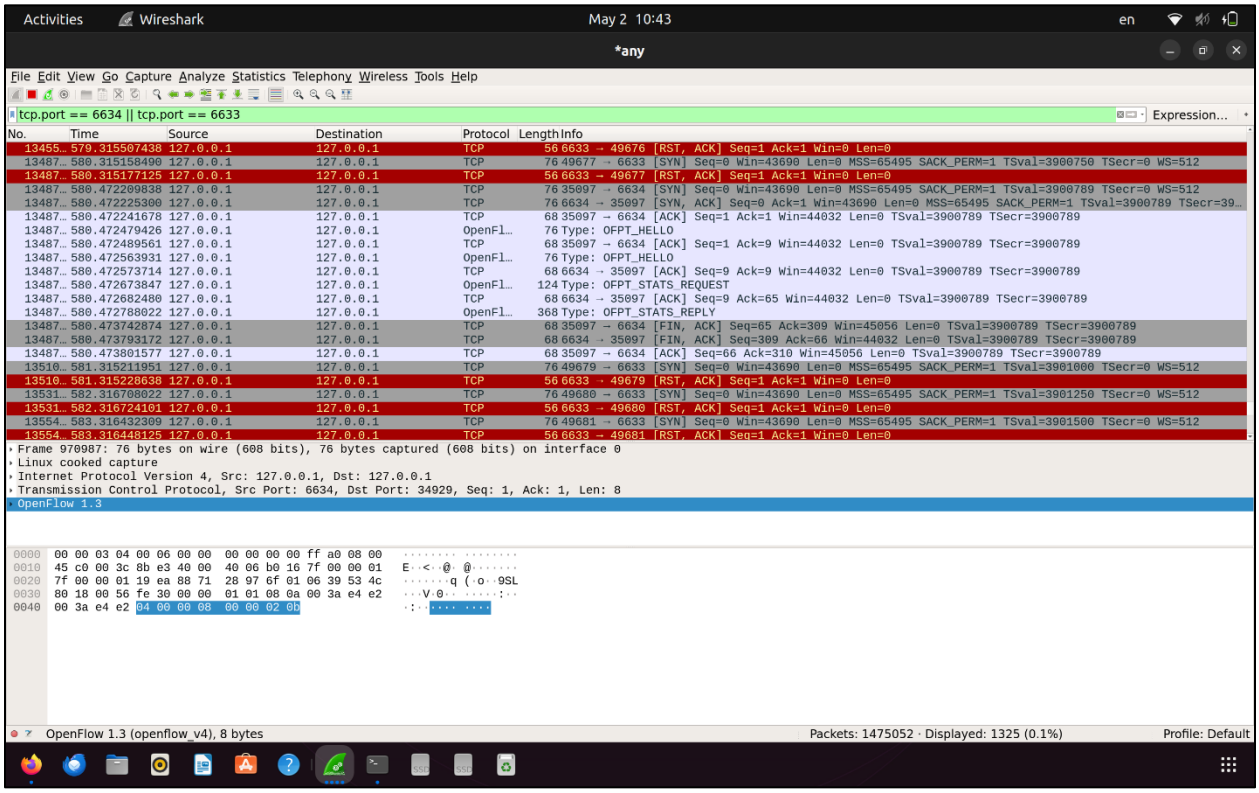
From these outputs, we can clearly observe that how echo request-responses have been exchanges between h1 and h2 while h3 haven't received any.

7. **Try the following:**
   - **$dpctl add-flow tcp:127.0.0.1:6634 dl_dst=0:0:0:0:0:1,idle_timeout=1000,actions=output:1**
   - **$dpctl add-flow tcp:127.0.0.1:6634 dl_dst=0:0:0:0:0:2,idle_timeout=1000,actions=output:2**
   - **$dpctl add-flow tcp:127.0.0.1:6634 dl_dst=0:0:0:0:0:3,idle_timeout=1000,actions=output:3**
   - **$dpctl dump-flows tcp:127.0.0.1:6634**
   - **Mininet> pingall**

Following is the execution of the above rules and verification through looking up the flow table

```
vagrant@sdn-box:~$ dpctl add-flow tcp:127.0.0.1:6634
dpctl: 'add-flow' command requires at least 2 arguments
vagrant@sdn-box:~$ dpctl add-flow tcp:127.0.0.1:6634 dl_dst=0:0:0:0:0:1,idle_timeout=1000,actions=output:1
vagrant@sdn-box:~$ dpctl add-flow tcp:127.0.0.1:6634 dl_dst=0:0:0:0:0:2,idle_timeout=1000,actions=output:2
vagrant@sdn-box:~$ dpctl add-flow tcp:127.0.0.1:6634 dl_dst=0:0:0:0:0:3,idle_timeout=1000,actions=output:3
vagrant@sdn-box:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x7683dd91): flags=none type=1(flow)
  cookie=0, duration_sec=38s, duration_nsec=396000000s, table_id=0, priority=32768, n_packets=0, n_bytes=0, idle_timeout=1000,hard_timeout=0,dl_dst=00:00:00:00:00:02,actions=output
:2
  cookie=0, duration_sec=26s, duration_nsec=980000000s, table_id=0, priority=32768, n_packets=0, n_bytes=0, idle_timeout=1000,hard_timeout=0,dl_dst=00:00:00:00:00:03,actions=output
:3
  cookie=0, duration_sec=164s, duration_nsec=425000000s, table_id=0, priority=32768, n_packets=0, n_bytes=0, idle_timeout=1000,hard_timeout=0,dl_dst=00:00:00:00:00:01,actions=outpu
t:1
vagrant@sdn-box:~$
```

While, executing the "dpctl dump-flows" was executed the following Openflow packets were captured at tcp:127.0.0.1:6633

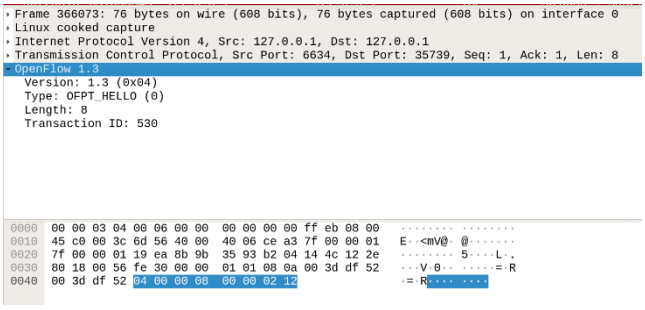


Types of Openflow packets observed:

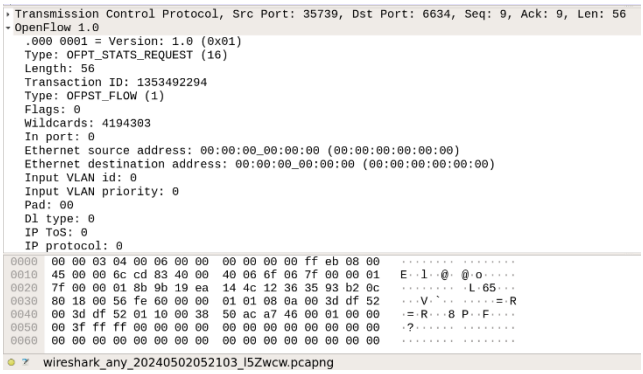
- *OFPT_HELLO : Openflow packet to initailze the communication with the Openflow switch (s1)*
- *OFPT_STATS_REQUEST :sent to sdn controller (c0) requesting the flow table statistics*
- *OFPT_STATS_REPLY : sent by the sdn controller with the current stats(flow rules)*

More in-dpeth view on above Openflow packets

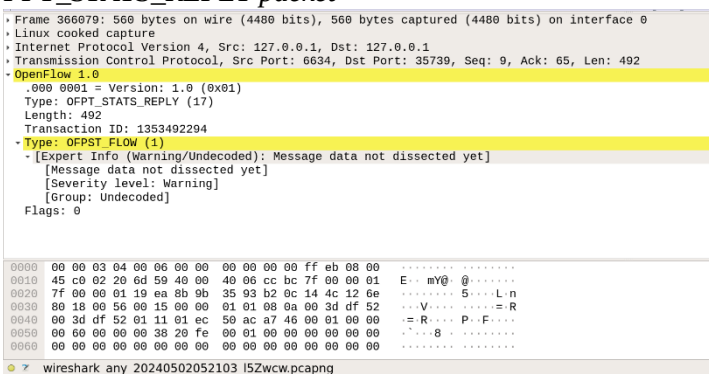
- *OFPT_HELLO packet*



- *OFPT_STATS_REQUEST packet*



- *OFPT_STATS_REPLY packet*



As the above configured 3 rules connect h2 with h3 now all the connections except for h1-h3 must piing

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 h3
h3 -> X h2
*** Results: 33% dropped (4/6 received)
mininet>
```

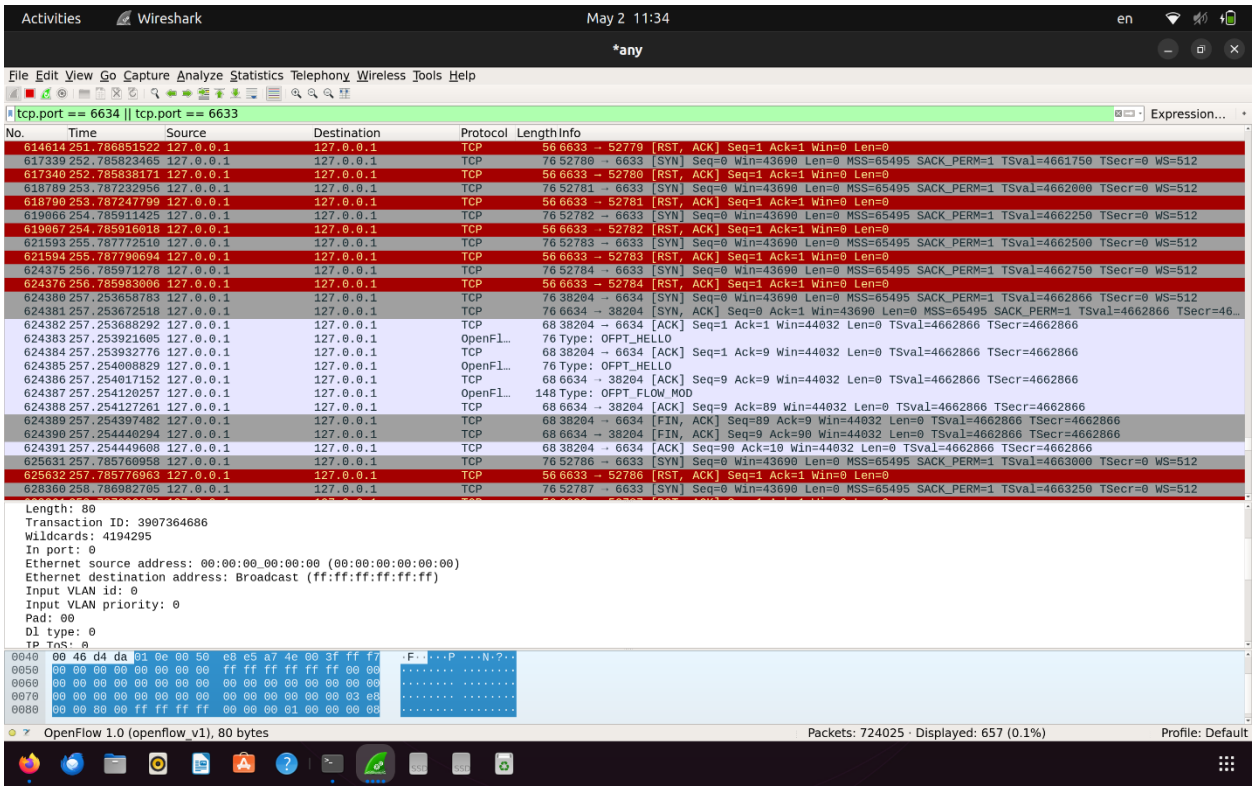
The expected result was achived.

8. $dpctl add-flow tcp:127.0.0.1:6634 dl_dst=ff:ff:ff:ff:ff:ff,idle_timeout=1000,actions=flood

```
vagrant@sdn-box:~$ dpctl add-flow tcp:127.0.0.1:6634 idle_timeout=1000,actions=flood
vagrant@sdn-box:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x4c2c0e93): flags=none type=1(flow)
  cookie=0, duration_sec=6s, duration_nsec=898000000s, table_id=0, priority=32768, n_packets=0, n_bytes=0, idle_timeout=1000,hard_timeout=0,actions=FLOOD
  cookie=0, duration_sec=350s, duration_nsec=195000000s, table_id=0, priority=32768, n_packets=4, n_bytes=168, idle_timeout=1000,hard_timeout=0,in_port=1,actions=output:2
  cookie=0, duration_sec=341s, duration_nsec=190000000s, table_id=0, priority=32768, n_packets=0, n_bytes=0, idle_timeout=1000,hard_timeout=0,in_port=2,actions=output:1
  cookie=0, duration_sec=939s, duration_nsec=700000000s, table_id=0, priority=32768, n_packets=11, n_bytes=798, idle_timeout=1000,hard_timeout=0,dl_dst=00:00:00:00:00:02,actions=ou
tput:2
  cookie=0, duration_sec=928s, duration_nsec=284000000s, table_id=0, priority=32768, n_packets=8, n_bytes=560, idle_timeout=1000,hard_timeout=0,dl_dst=00:00:00:00:00:03,actions=out
put:3
  cookie=0, duration_sec=1065s, duration_nsec=729000000s, table_id=0, priority=32768, n_packets=4, n_bytes=280, idle_timeout=1000,hard_timeout=0,dl_dst=00:00:00:00:00:01,actions=ou
tput:1
vagrant@sdn-box:~$
```

Adding this new Openflow rule.

This was the wireshark output received while adding this new rule:

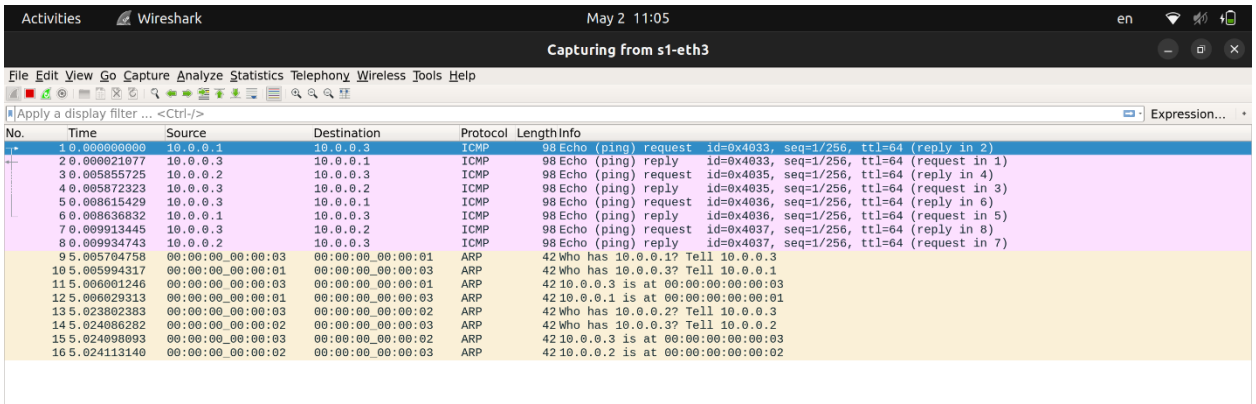


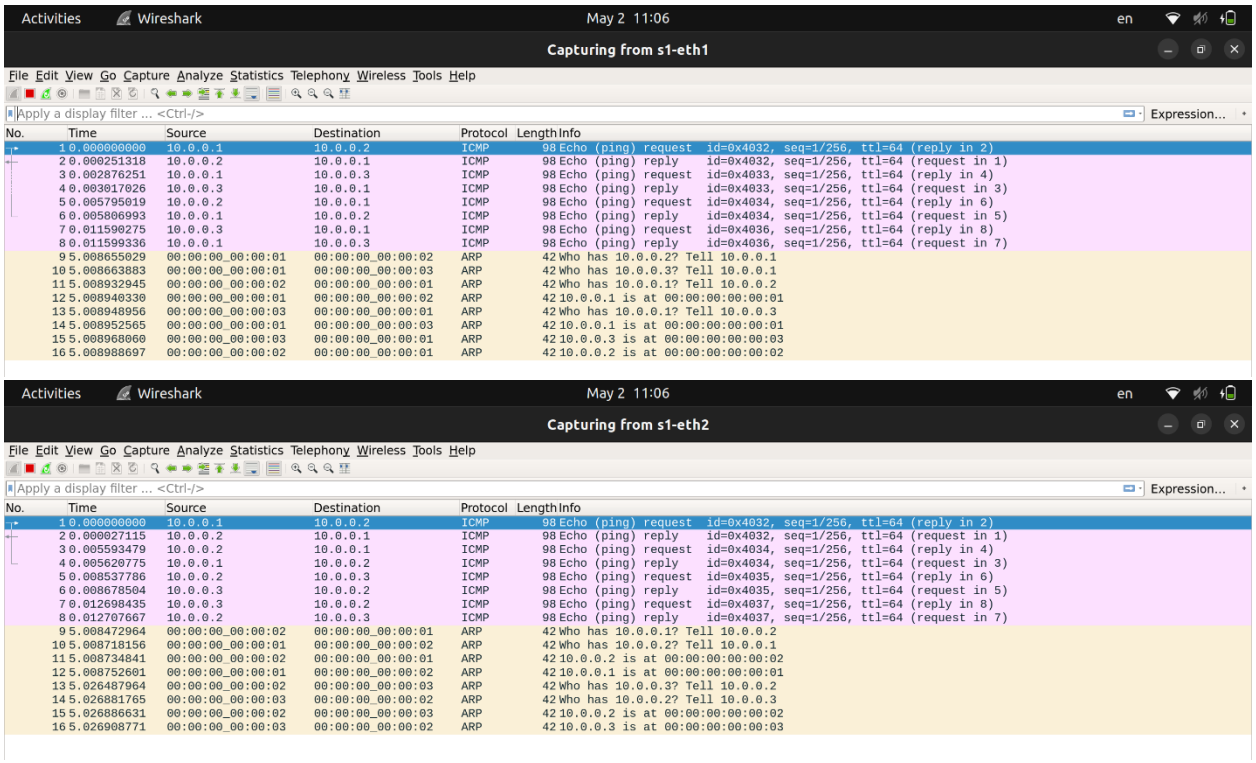
It can be observed that the 'OFPT_FLOW_MOD' packet contained the new rule information. Inside the packet we can see that the FLOOD rule has been sent as allowing from any MAC address to the broadcast mac address(ff:ff:ff:ff:ff:ff).

Now with this rule, the currently isolated h1-h3; must connect.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

As expected pingall was successful.





It can be clearly observed that all the echo requests and responses were successfully exchanged with all the 3 hosts.

9. How to make the ping successful for one pair of hosts

Lets make ping succefull only for the pair h1-h3

I have used the following commands for this task

- To add the flow path to host-3
  $ dpctl add-flow tcp:127.0.0.1:6634 dl_dst=0.0.0.3,idle_timeout=1000,actions=output:3
- To add flow path to host-1
  $ dpctl add-flow tcp:127.0.0.1:6634 dl_dst=0.0.0.1,idle_timeout=1000,actions=output:1

Flow table Output

```
vagrant@sdn-box:~$ dpctl add-flow tcp:127.0.0.1:6634 dl_dst=0:0:0:0:0:3,idle_timeout=1000,actions=output:3
vagrant@sdn-box:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0xb911e75e): flags=none type=1(flow)
  cookie=0, duration_sec=28s, duration_nsec=367000000s, table_id=0, priority=32768, n_packets=0, n_bytes=0, idle_timeout=1000,hard_timeout=0,dl_dst=00:00:00:00:00:03,actions=output
:3
  cookie=0, duration_sec=828s, duration_nsec=753000000s, table_id=0, priority=32768, n_packets=4, n_bytes=280, idle_timeout=1000,hard_timeout=0,dl_dst=00:00:00:00:00:01,actions=out
put:1
```

Succefully achived the connectivity between h1 and h3

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3
h2 -> X X
h3 -> h1 X
*** Results: 66% dropped (2/6 received)
mininet>
```

Analysis on OpenFlow

OpenFlow Packet Structure

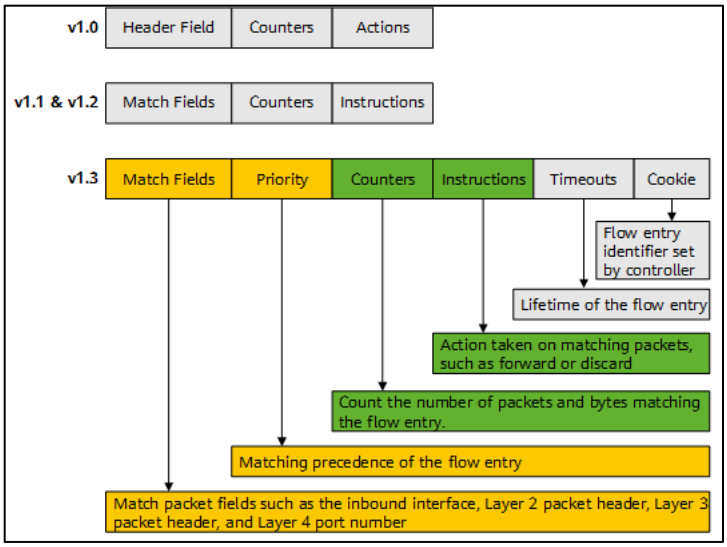


Figure 3: OpenFlow Packet Structure

Fields in an OpenFlow Packet

- ***Header**: Contains essential information about the packet, including the version number, type, length, and transaction ID.*

Depending on the Openflow header type the message body of that Openflow packet might vary. Common fields in the Openflow body are:

- ***Hello**: Used for establishing a connection between OpenFlow controllers and switches.*
- ***Features Request/Reply**: Requests or provides information about the capabilities of the switch, such as the number of ports and supported features.*
    - ***Eg**: STATS_REQUEST/REPLY (when executing dpctl dump commands)*
- ***Flow Mod**: Instructs the switch to add, modify, or delete flow entries in its flow table.*
- ***Packet Out**: Instructs the switch to forward a packet to a specified port.*
- ***Packet In**: Sent by the switch to the controller when it encounters a packet for which it has no matching flow entry.*
- ***Flow Removed**: Indicates that a flow entry has been removed from the switch's flow table.*
- ***Error**: Indicates an error condition encountered during the processing of a message.*

Different OpenFlow message types

a. *Hello (OFPT_HELLO): Used for establishing a connection between the controller and switch. It includes the version negotiation.*
b. *Echo Request/Reply (OFPT_ECHO_REQUEST/OFPT_ECHO_REPLY): Used for testing the responsiveness of the switch or controller.*
c. *Features Request/Reply (OFPT_FEATURES_REQUEST/OFPT_FEATURES_REPLY): Requests or provides information about the switch's capabilities and features.*
d. *Flow Mod (OFPT_FLOW_MOD): Instructs the switch to add, modify, or delete flow entries in its flow table.*
e. *Packet Out (OFPT_PACKET_OUT): Instructs the switch to forward a packet to a specified port.*
f. *Packet In (OFPT_PACKET_IN): Sent by the switch to the controller when it encounters a packet for which it has no matching flow entry.*
g. *Flow Removed (OFPT_FLOW_REMOVED): Indicates that a flow entry has been removed from the switch's flow table.*
h. *Error (OFPT_ERROR): Indicates an error condition encountered during the processing of a message.*