

Creating Network Applications without using Controller

Using reference/OVS Controller/Remote Controller

```
root@mininet-vm:~# mn --controller ref
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> _
```

Openflow switches are of two types, physical and virtual switches. By default, Mininet creates virtual switch in OpenFlow mode. We need OpenFlow controller to manage and control OpenFlow switch. Mininet supports many controllers, such as OpenFlow reference controller, OVS controller and less used NOX Classic. You can choose OpenFlow controller simply by using the mn command.

```
root@mininet-vm:~# ps -ax |grep controller
2031 tty2      S+   0:00 /usr/bin/python /usr/local/bin/mn --controller ref
2104 pts/2     S+   0:00 controller -v tcp:6653
```

mn --controller ref

Using reference/OVS Controller/Remote Controller

```
root@mininet-vm:~# mn --controller ovsc
```

```
*** Creating network
```

```
*** Adding controller
```

```
*** Adding hosts:
```

```
h1 h2
```

```
*** Adding switches:
```

```
s1
```

```
*** Adding links:
```

```
(h1, s1) (h2, s1)
```

```
*** Configuring hosts
```

```
h1 h2
```

```
*** Starting controller
```

```
c0
```

```
*** Starting 1 switches
```

```
s1 ...
```

```
*** Starting CLI:
```

```
mininet>
```

```
mininet> pingall
```

```
*** Ping: testing ping reachability
```

```
h1 -> h2
```

```
h2 -> h1
```

```
*** Results: 0% dropped (2/2 received)
```

```
mininet> _
```

```
# mn --controller ovsc
```

```
root@mininet-vm:~# ps -ax |grep controller
```

```
2309 tty2      S+   0:00 /usr/bin/python /usr/local/bin/mn --controller ovsc
```

```
2377 pts/2      S+   0:00 ovs-controller -v ptcp:6653
```

```
2440 tty1       S+   0:00 grep --color=auto controller
```

Default Controller

```
root@mininet-vm:~# mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

```
# mn
root@mininet-vm:~# ps -ax |grep controller
1793 pts/2    S+         0:00 controller -v ptcp:6653
```

```
# ovs-vsctl show
(gives information about the open vswitch s1)
```

```
root@mininet-vm:~# ovs-vsctl show
918037ec-b307-45d7-a75a-f1ac4337d135
    Bridge "s1"
        Controller "tcp:127.0.0.1:6653"
            is_connected: true
        Controller "ptcp:6654"
        fail_mode: secure
        Port "s1-eth2"
            Interface "s1-eth2"
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1"
            Interface "s1"
                type: internal
    ovs_version: "2.0.2"
root@mininet-vm:~#
```

Default Controller

```
root@mininet-vm:~# ovs-ofctl show s1
OFPT_FEATURES_REPLY (xid=0x2): dpid:000000000000000001
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_IP_SRC
SET_IP_DST ENQUEUE
1(s1-eth1): addr:fa:a4:90:bf:1e:f4
    config:      0
    state:       0
    current:     10GB-FD COPPER
    speed: 10000 Mbps now, 0 Mbps max
2(s1-eth2): addr:b6:98:29:d7:61:68
    config:      0
    state:       0
    current:     10GB-FD COPPER
    speed: 10000 Mbps now, 0 Mbps max
LOCAL(s1): addr:da:af:0b:30:e2:40
    config:      0
    state:       0
    speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_interval=0
root@mininet-vm:~#
root@mininet-vm:~# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
root@mininet-vm:~#
```

ovs-ofctl show s1
(shows information about flow tables and ports)

```
mininet> h1 ping -c2 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=2.00 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.67 ms
```

ovs-ofctl dump-flows s1
(gives information about flow table entries)

```
root@mininet-vm:~# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
    cookie=0x0, duration=1.416s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=1, priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=b2:f7:37:66:7b:18,dl_dst=56:e2:23:79:ae:db,arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op=1 actions=output:1
    cookie=0x0, duration=1.415s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=1, priority=65535,arp,in_port=1,vlan_tci=0x0000,dl_src=56:e2:23:79:ae:db,dl_dst=b2:f7:37:66:7b:18,arp_spa=10.0.0.1,arp_tpa=10.0.0.2,arp_op=2 actions=output:2
    cookie=0x0, duration=6.43s, table=0, n_packets=2, n_bytes=196, idle_timeout=60, idle_age=5, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=b2:f7:37:66:7b:18,dl_dst=56:e2:23:79:ae:db,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:1
    cookie=0x0, duration=5.417s, table=0, n_packets=1, n_bytes=98, idle_timeout=60, idle_age=5, priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_src=56:e2:23:79:ae:db,dl_dst=b2:f7:37:66:7b:18,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:2
root@mininet-vm:~#
```

Using Remote Controller

```
root@mininet-vm:~# mn --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)
mininet>
```

Since we are not running any controller, hosts will not be able to ping with each other.

```
root@mininet-vm:~# ps -ax |grep controller
2561 tty2    S+   0:00 /usr/bin/python /usr/local/bin/mn --controller remote
```

In Mininet network, switches can be connected to a remote controller. The syntax is

```
# mn --controller=remote, ip=[controller IP],
port=[controller listening port]
```

```
#mn --controller remote
```

(This command creates a topology that contains 2 hosts, single open vswitch & point to the remote controller running on localhost)

Without Controller

```
root@mininet-vm:~# mn --controller none
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)
mininet>
```

Here we are creating topology without specifying any controller, hosts will not be able to ping with each other.

```
root@mininet-vm:~# ps -ax |grep controller
3395 tty2      S+   0:00 /usr/bin/python /usr/local/bin/mn --controller none
```

Using ovs-ofctl

```
root@mininet-vm:~# ovs-ofctl dump-flows tcp:127.0.0.1:6634
ovs-ofctl: connecting to tcp:127.0.0.1:6634 (Connection refused)
root@mininet-vm:~#
root@mininet-vm:~# ovs-ofctl dump-flows tcp:127.0.0.1:6654
NXST_FLOW reply (xid=0x4):
root@mininet-vm:~#
```

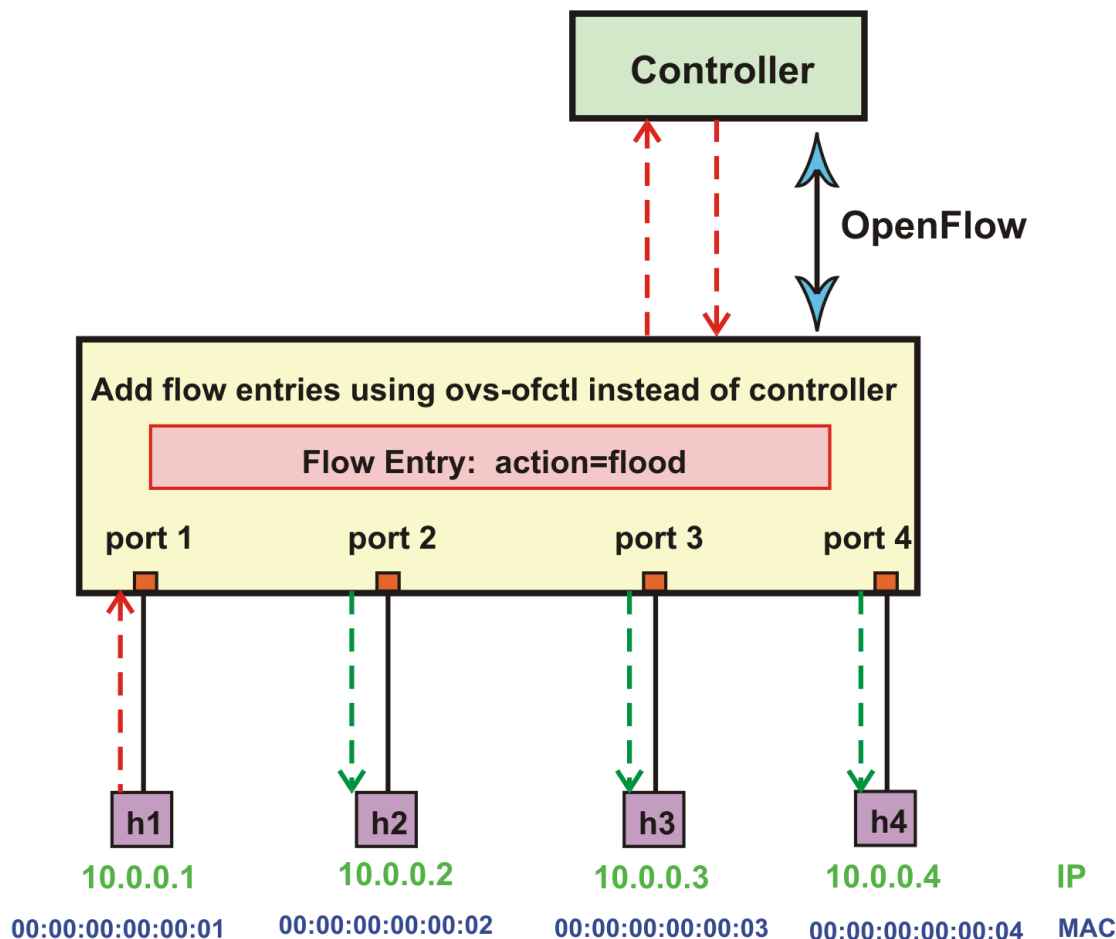
ovs-ofctl is a command-line utility that sends basic OpenFlow messages to a switch. It communicates directly with a switch and does not need a controller. It is especially useful for debugging, viewing flow state and flow counters.

To obtain this information you can query the switch on port 6654. In the beginning, the port used was 6634, but now it has been allocated 6654 port. It gives error when you try to connect on port 6634. You can also add/delete flow table entries using this utility.

```
root@mininet-vm:~# ovs-ofctl show tcp:127.0.0.1:6654
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST
        _NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
1(s1-eth1): addr:1e:96:0a:d4:fc:ad
    config:      0
    state:       0
    current:     10GB-FD COPPER
    speed: 10000 Mbps now, 0 Mbps max
2(s1-eth2): addr:ba:92:49:6e:af:b9
    config:      0
    state:       0
    current:     10GB-FD COPPER
    speed: 10000 Mbps now, 0 Mbps max
LOCAL(s1): addr:fe:12:46:2e:36:45
    config:      0
    state:       0
    speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
root@mininet-vm:~#
root@mininet-vm:~# ovs-ofctl show tcp:127.0.0.1:6634
ovs-ofctl: connecting to tcp:127.0.0.1:6634 (Connection refused)
root@mininet-vm:~#
```


Hub Application using ovs-ofctl

Adding flow entries for hub



Network applications such as hub, switch, router can be created using mininet & ovs-ofctl. The ovs-ofctl can be used to create more complex applications like firewall and load balancer. Now we are going to use “ovs-ofctl” for converting our dump simple ovs datapath into useful devices such as hub, switch and firewall.

If flow table contains a flow entry which contains the action to flood the packet that arrives at specific port of forwarding device, then that forwarding device behaves like a hub.

Figure shows all hosts belongs to the same network. when host h1 wants to send a packet to host h4, then it first sends a packet to forwarding device at port 1. When a packet arrives at port 1, then it is matched against flow entry. When match is found, then it is flooded to all ports except the incoming port according to action specified in flow entry.

Step 1: Create Topology

```
root@mininet-vm:~# mn --mac --topo single,4 --controller none
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
```

Create mininet topology consisting of 1 switch & 4 hosts. We do not want to use any controller so we have specified the option “*--controller none*”.

You can use the following command for creating required topology.

```
# mn --mac --topo single,4 --controller none
```

on mininet prompt, try to ping *h2 from h1*. As can be seen, we are not able to ping.

Step 2: Add Flow

```
root@mininet-vm:~# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
root@mininet-vm:~#
root@mininet-vm:~# ovs-ofctl add-flow s1 actions=flood
root@mininet-vm:~#
root@mininet-vm:~# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.031s, table=0, n_packets=0, n_bytes=0, idle_age=6, actions=FLOOD
root@mininet-vm:~#
```

Before adding flow, check the flow by using command

```
# ovs-ofctl dump-flows s1
```

(it is not showing any rules)

```
# ovs-ofctl add-flow s1 actions=flood
```

(Now add flow entry by using the “*ovs-ofctl*” utility. Add flow entry by specifying action as “flood”)

now rules are visible on using “*ovs-ofctl*” with option “dump-flows”.

Step 3: Checking Connectivity

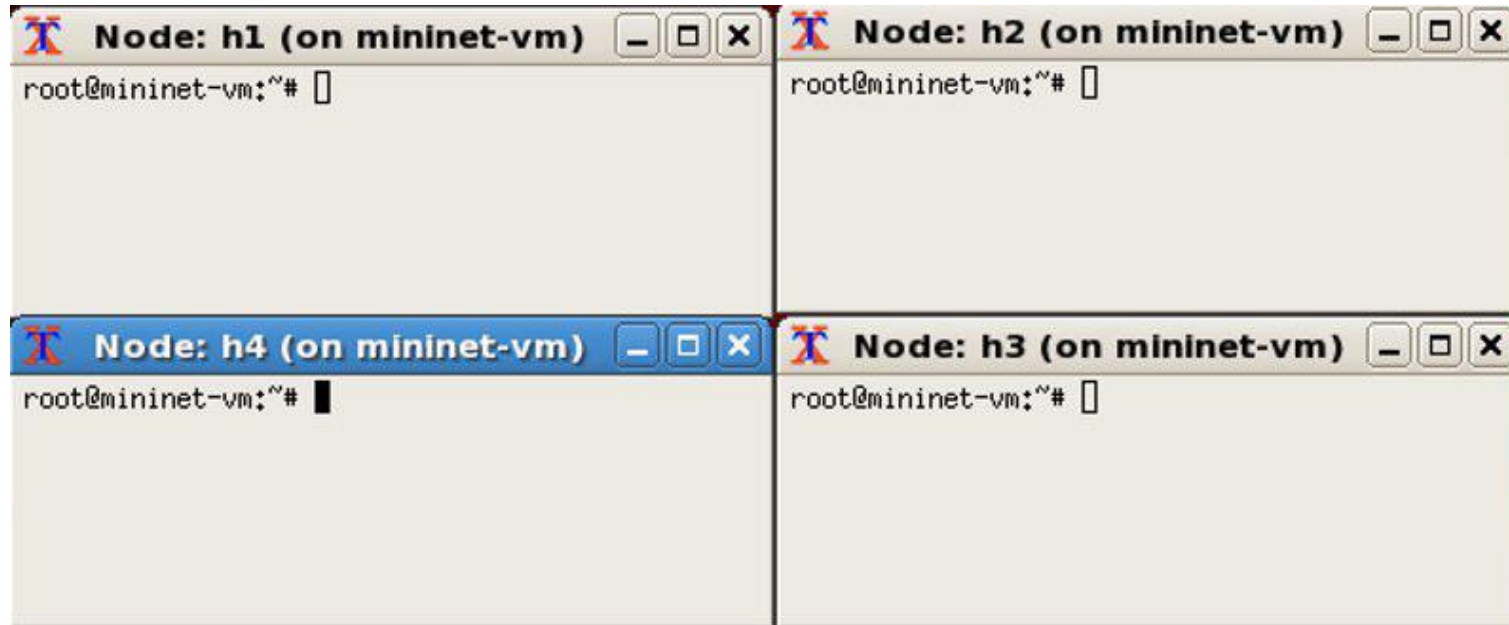
```
mininet> h1 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.851 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.071 ms

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 0.071/0.461/0.851/0.390 ms
mininet>
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
mininet>
```

As can be seen, now h1 is able to ping to h2 & h3 is able to ping h4. The option “-c2” specified with the ping commands tell it to send only 2 packets. You can also use pingall command to test the connectivity.

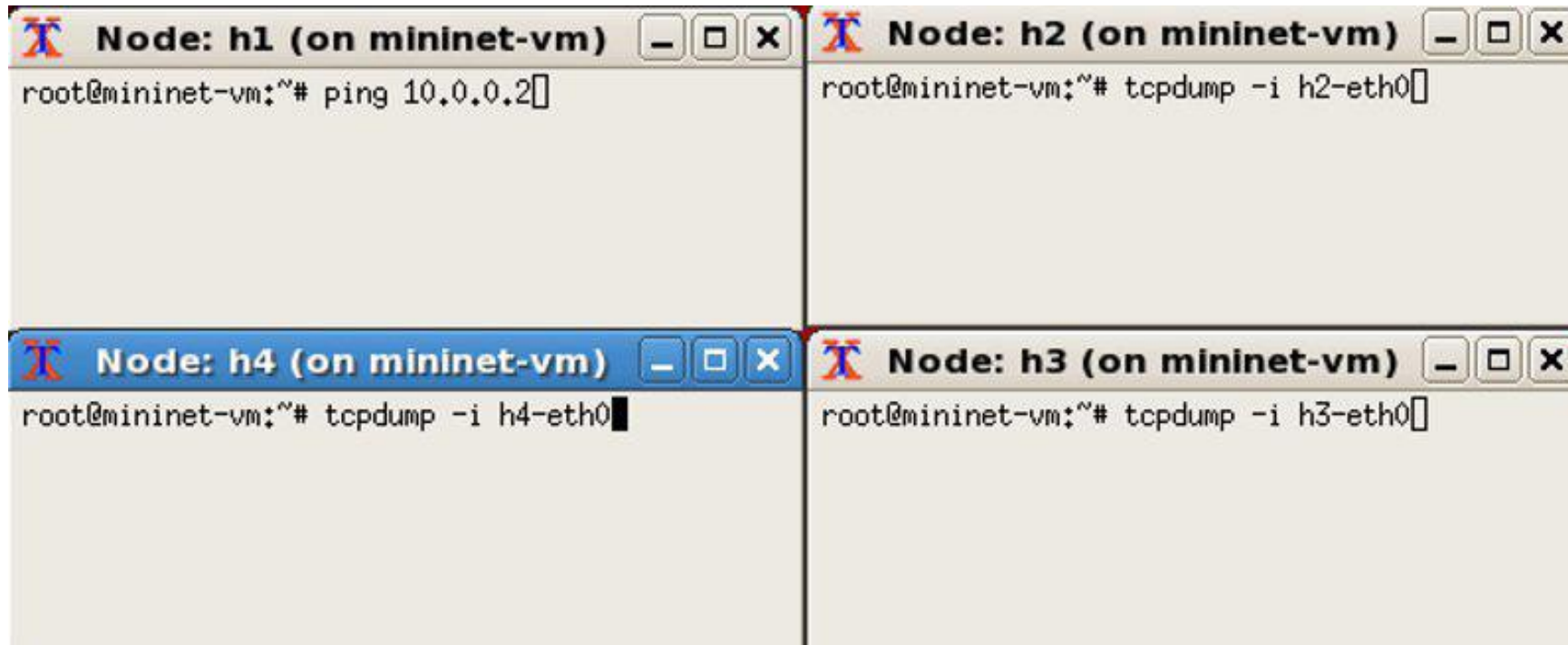
Step 4: Verify Hub Behavior

```
mininet> xterm h1 h2 h3 h4  
mininet>
```



There is another way by which you can check the functionality. Spawn x terminals by running the command “*xterm h1 h2 h3 h4*” on mininet prompt. It will open x terminals as shown in screenshot.

Step 5: Verify Hub Behavior



We will use “tcpdump” utility for capturing the traffic on hosts h2, h3 & h4. We will ping from host h1 to host h2 (10.0.0.2).

Type the following command on different hosts.

<u>host</u>	<u>command</u>
h2	tcpdump -i h2-eth0
h3	tcpdump -i h3-eth0
h4	tcpdump -i h4-eth0
h1	ping 10.0.0.2

Step 6: Verify Hub Behavior

Node: h1 (on mininet-v)	Node: h2 (on mininet-v)
<pre>root@mininet-vm:~# ping 10.0.0.2 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=2.04 ms 64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.105 ms 64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.108 ms 64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.104 ms ^C --- 10.0.0.2 ping statistics --- 4 packets transmitted, 4 received, 0% packet loss, time 3001ms rtt min/avg/max/mdev = 0.104/0.591/2.047/0.840 ms root@mininet-vm:~#</pre>	<pre>listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes 23:25:03.160776 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28 23:25:03.160838 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02 (oui Ethernet), length 28 23:25:03.161972 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1565, seq 1, length 64 23:25:03.162009 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1565, seq 1, length 64 23:25:04.162345 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1565, seq 2, length 64 23:25:04.162381 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1565, seq 2, length 64 23:25:05.161383 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1565, seq 3, length 64 23:25:05.161420 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1565, seq 3, length 64 23:25:06.161396 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1565, seq 4, length 64 23:25:06.161433 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1565, seq 4, length 64 23:25:08.173339 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28 23:25:08.173414 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01 (oui Ethernet), length 28 ^</pre>
Node: h4 (on mininet-v)	Node: h3 (on mininet-v)
<pre>listening on h4-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes 23:25:03.160774 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28 23:25:03.161533 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02 (oui Ethernet), length 28 23:25:03.161971 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1565, seq 1, length 64 23:25:03.162477 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1565, seq 1, length 64 23:25:04.162343 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1565, seq 2, length 64 23:25:04.162389 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1565, seq 2, length 64 23:25:05.161381 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1565, seq 3, length 64 23:25:05.161427 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1565, seq 3, length 64</pre>	<pre>listening on h3-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes 23:25:03.160770 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28 23:25:03.161527 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02 (oui Ethernet), length 28 23:25:03.161969 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1565, seq 1, length 64 23:25:03.162473 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1565, seq 1, length 64 23:25:04.162340 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1565, seq 2, length 64 23:25:04.162387 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1565, seq 2, length 64 23:25:05.161379 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1565, seq 3, length 64 23:25:05.161425 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1565, seq 3, length 64</pre>

ping from host h1 to host h2 (10.0.0.2). As can be seen, traffic is also received by h3 & h4 hosts although the packet was destined for h2 host.

Deleting Flow Entries

```
root@mininet-vm:~# ovs-ofctl del-flows s1
root@mininet-vm:~#
root@mininet-vm:~# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
root@mininet-vm:~#
```

```
# ovs-ofctl del-flows s1
(delete all flow entries in device s1)
```

```
# ovs-ofctl dump-flows s1
(show all flow entries)
```

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X X X
h3 -> X X X
h4 -> X X X
*** Results: 100% dropped (0/12 received)
mininet> exit
*** Stopping 0 controllers

*** Stopping 4 links
....
*** Stopping 1 switches
s1
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
completed in 7719.143 seconds
root@mininet-vm:~#
root@mininet-vm:~# mn -c
```

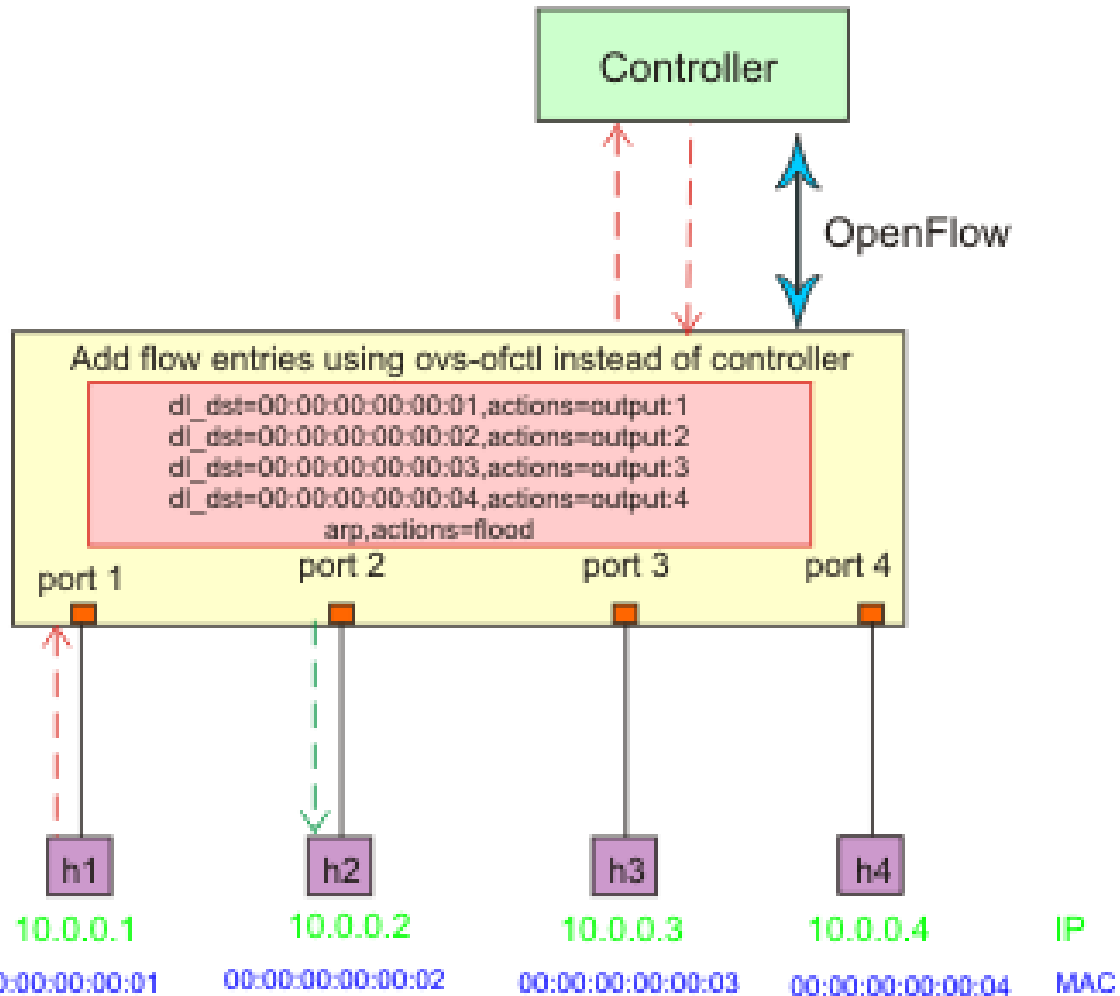
```
mininet> pingall
(testing ping reachability. Not pinging
because we deleted the rules.)
```

```
mininet> exit
(destroy topology)
```

```
# mn -c
(clean topology leftovers)
```

Switch Application using ovs-ofctl

Adding Flow Entries for Switch



If flow table contains entry with action to forward the packet to specific port in same network, then forwarding device acts as switch. Figure shows that host h1 and h2 belongs to the same network. When host h1 wants to send a packet to host h2, then it first sends the packet to forwarding device at port 1. When a packet arrives at port 1, then it is matched against flow entry. When match is found, then it is forwarded to port 2 according to action specified in flow entry

Step 1: Create Topology

```
root@mininet-vm:~# mn --mac --topo single,4 --controller none
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X X X
h3 -> X X X
h4 -> X X X
*** Results: 100% dropped (0/12 received)
mininet> _
```

Create mininet topology consisting of 1 switch & 4 hosts. We do not want to use any controller so we have specified the option “*--controller none*”.

You can use the following command for creating required topology.

```
# mn --mac --topo single,4 --controller none
```

on mininet prompt, run pingall. As can be seen, none of the hosts are able to ping each other.

Step 2: Add Flows

```
root@mininet-vm:~# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
root@mininet-vm:~#
root@mininet-vm:~# ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:01,actions=output:1
root@mininet-vm:~#
root@mininet-vm:~# ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:02,actions=output:2
root@mininet-vm:~#
root@mininet-vm:~# ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:03,actions=output:3
root@mininet-vm:~#
root@mininet-vm:~# ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:04,actions=output:4
root@mininet-vm:~#
root@mininet-vm:~# ovs-ofctl add-flow s1 arp,actions=flood
root@mininet-vm:~# _
```

Before adding flow, check the flows
ovs-ofctl dump-flows s1
(it is not showing any rules)

(The first flow table entry specifies that if destination mac address is 00:00:00:00:00:01 then send the packet out through port 1. 2nd, 3rd, 4th rules are of same variety. Fifth entry specifies that if the packet is of arp type then flood the packet. By adding the above 5 flow entries into device s1, we are simulating the behavior of the switch.)

now rules are visible on using “ovs-ofctl” with option “dump-flows”.

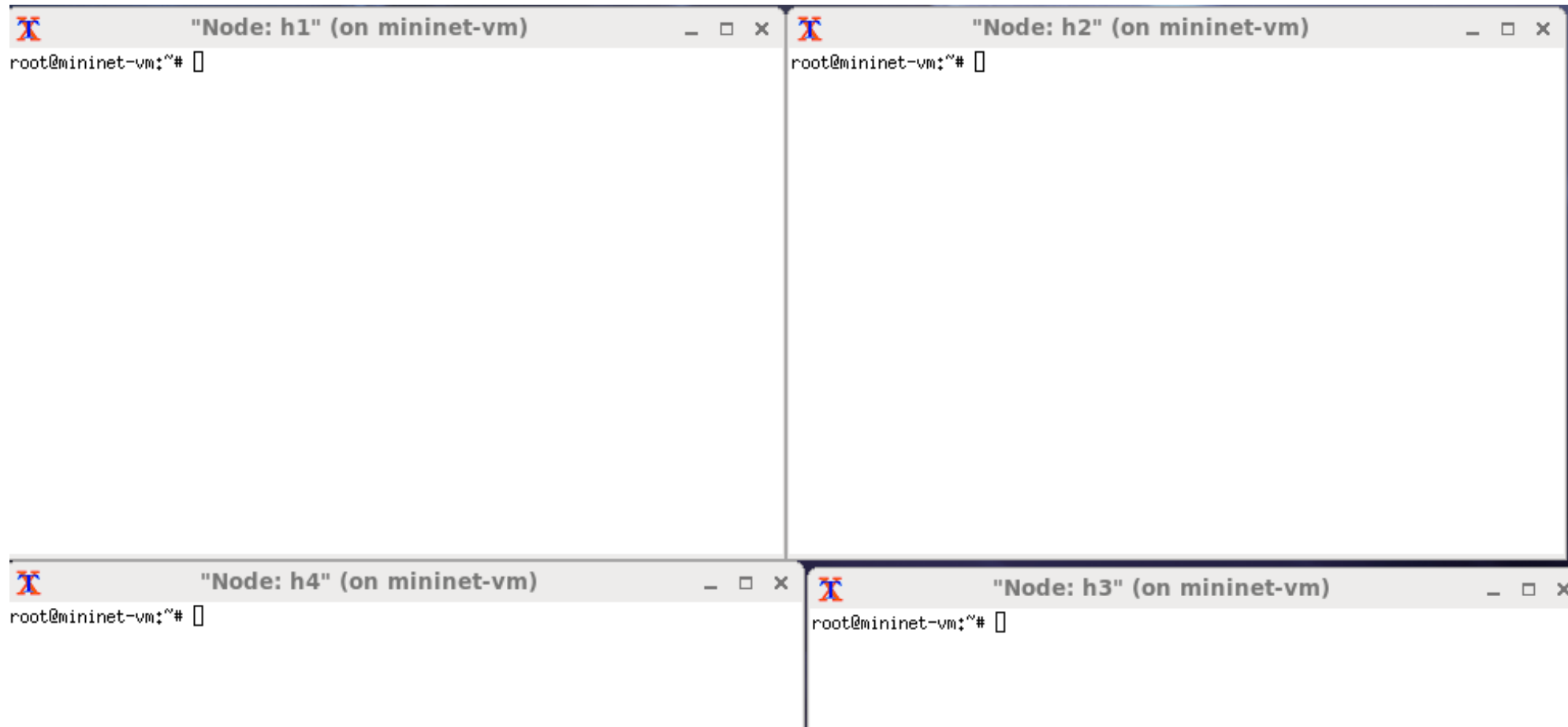
```
root@mininet-vm:~# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=199.828s, table=0, n_packets=0, n_bytes=0, idle_age=199, dl_dst=00:00:00:00:00:02 actions=output:2
cookie=0x0, duration=189.007s, table=0, n_packets=0, n_bytes=0, idle_age=189, dl_dst=00:00:00:00:00:03 actions=output:3
cookie=0x0, duration=217.49s, table=0, n_packets=0, n_bytes=0, idle_age=217, dl_dst=00:00:00:00:00:01 actions=output:1
cookie=0x0, duration=178.401s, table=0, n_packets=0, n_bytes=0, idle_age=178, dl_dst=00:00:00:00:00:04 actions=output:4
cookie=0x0, duration=133.228s, table=0, n_packets=0, n_bytes=0, idle_age=133, arp actions=FL00D
root@mininet-vm:~# _
```

Step 3: Checking Connectivity

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

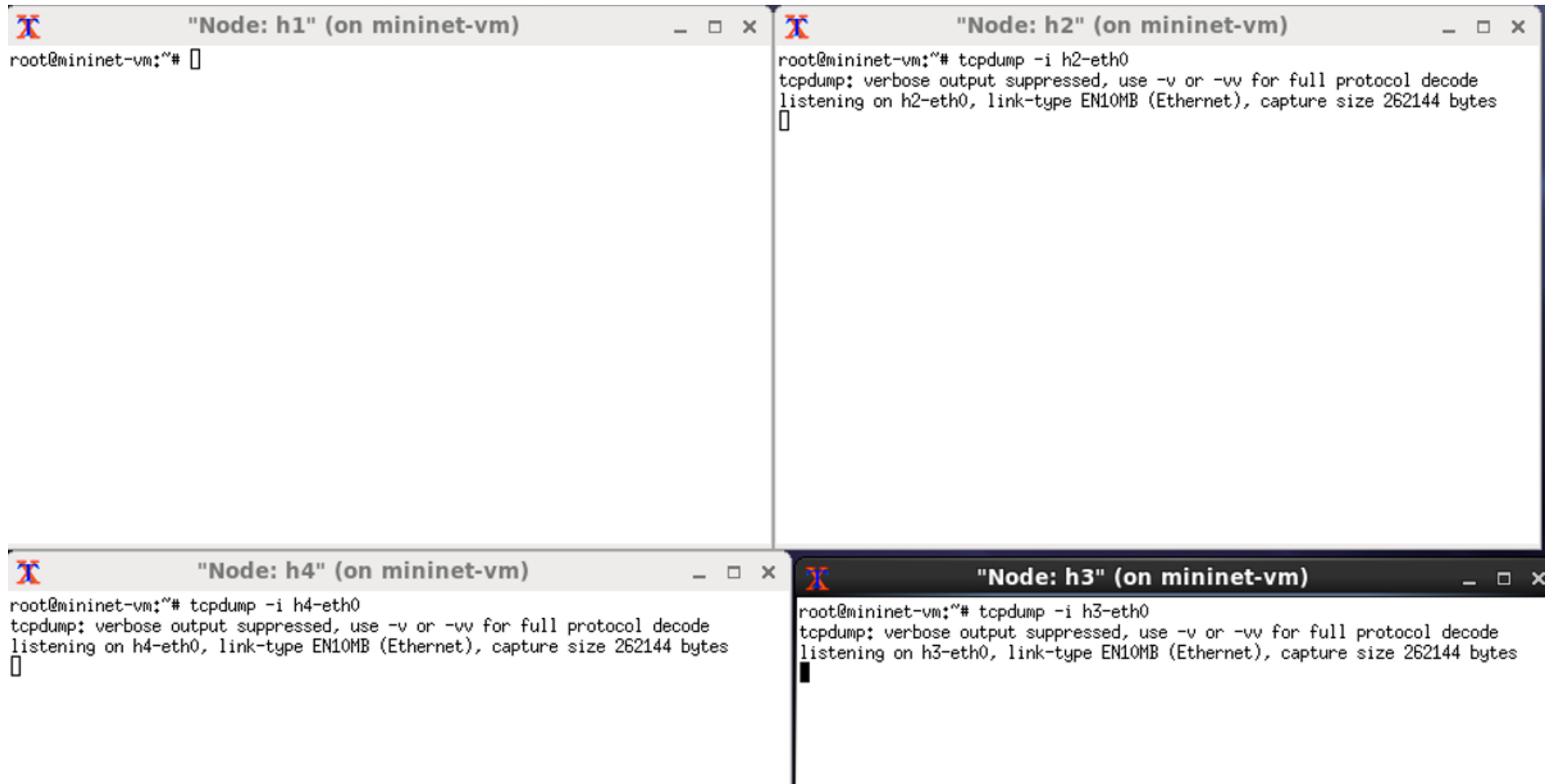
As can be seen, each host is able to ping to each other

Step 4: Verify Switch Behavior



There is another way by which you can check the functionality. Spawn x terminals by running the command `"xterm h1 h2 h3 h4"` on mininet prompt. It will open x terminals as shown in screenshot.

Step 5: Verify Switch Behavior



The image displays four terminal windows arranged in a 2x2 grid, each representing a different host in a Mininet VM environment. The windows are titled "Node: h1" (on mininet-vm), "Node: h2" (on mininet-vm), "Node: h4" (on mininet-vm), and "Node: h3" (on mininet-vm). Each window shows the root user at the mininet-vm prompt. Hosts h2, h3, and h4 have executed the command `tcpdump -i h2-eth0`, `tcpdump -i h4-eth0`, and `tcpdump -i h3-eth0` respectively, which has resulted in the message: "tcpdump: verbose output suppressed, use -v or -vv for full protocol decode listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes". Host h1 is currently at the prompt with no command entered.

```
"Node: h1" (on mininet-vm)
root@mininet-vm:~#

"Node: h2" (on mininet-vm)
root@mininet-vm:~# tcpdump -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

"Node: h4" (on mininet-vm)
root@mininet-vm:~# tcpdump -i h4-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h4-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

"Node: h3" (on mininet-vm)
root@mininet-vm:~# tcpdump -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

We will use “tcpdump” utility for capturing the traffic on hosts h2, h3 & h4. We will ping from host h1 to host h2 (10.0.0.2).

Step 6: Verify Switch Behavior

```
"Node: h1" (on mininet-virtual-machine)
root@mininet-virtual-machine:~# ping -c4 10.0.0.2
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=1.05 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.070 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 0.070/0.319/1.058/0.426 ms
root@mininet-virtual-machine:~#
root@mininet-virtual-machine:~#

"Node: h2" (on mininet-virtual-machine)
root@mininet-virtual-machine:~# tcpdump -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
08:03:18.574737 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2856, seq 1, length 64
08:03:18.574749 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 2856, seq 1, length 64
08:03:19.576158 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2856, seq 2, length 64
08:03:19.576182 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 2856, seq 2, length 64
08:03:20.577633 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2856, seq 3, length 64
08:03:20.577656 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 2856, seq 3, length 64
08:03:21.578101 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2856, seq 4, length 64
08:03:21.578123 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 2856, seq 4, length 64
08:03:23.586803 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
08:03:23.588390 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01 (oui Ethernet), length 28
root@mininet-virtual-machine:~#

"Node: h4" (on mininet-virtual-machine)
root@mininet-virtual-machine:~# tcpdump -i h4-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h4-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
root@mininet-virtual-machine:~#

"Node: h3" (on mininet-virtual-machine)
root@mininet-virtual-machine:~# tcpdump -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
root@mininet-virtual-machine:~#
```

ping from host h1 to host h2 (10.0.0.2). Also our device s1 is not flooding the traffic to hosts h3 & h4. Our device is sending the traffic to only specific ports.

Deleting Flow Entries

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X X X
h3 -> X X X
h4 -> X X X
*** Results: 100% dropped (0/12 received)
mininet> exit
*** Stopping 0 controllers

*** Stopping 8 terms
*** Stopping 4 links
....
*** Stopping 1 switches
s1
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
completed in 2668.497 seconds
root@mininet-vm:~#
root@mininet-vm:~# mn -c
```

```
root@mininet-vm:~# ovs-ofctl del-flows s1
root@mininet-vm:~#
root@mininet-vm:~# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
root@mininet-vm:~# _
```

ovs-ofctl del-flows s1
(delete all flow entries in device s1)

ovs-ofctl dump-flows s1
(show all flow entries)

mininet> pingall
(testing ping reachability. Not pinging
because we deleted the rules.)

mininet> exit
(destroy topology)

mn -c
(clean topology leftovers)