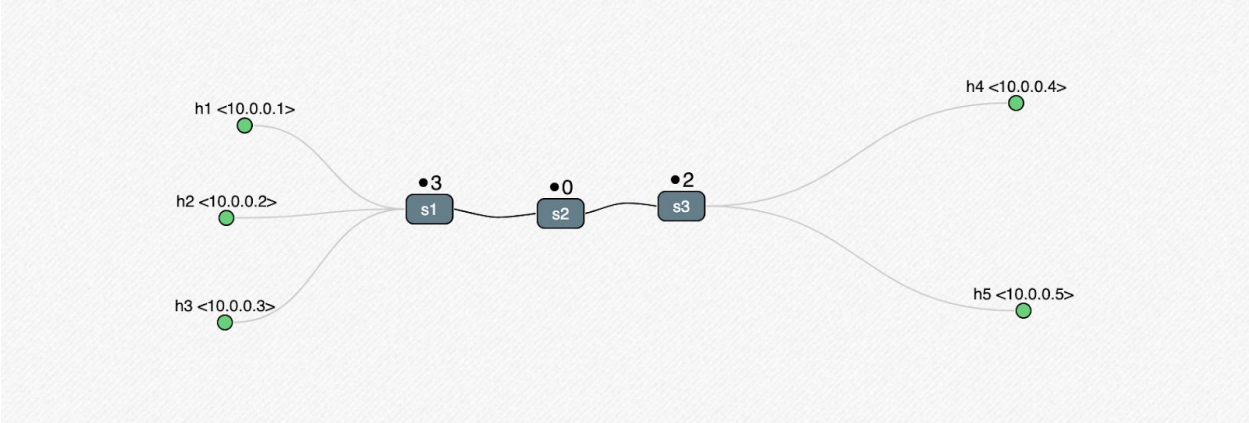
Advanced Computer Networks Assignment - 2: SDN

Submitted By: Pranshu Shrivastava

Topology Set-up:



1. Question 1:

    a. h1 ping h2: Average RTT = 28.790ms
    Traffic was observed for h2, h3, h4, h5, s1, s2, s3
    h1 ping h5: Average RTT = 29.3601ms
    Traffic was observed for h2, h3, h4, h5, s1, s2, s3
    No difference was observed for pings.

    b. iperf h1 h2 result:
    mininet> iperf h1 h2
    *** Iperf: testing TCP bandwidth between h1 and h2
    *** Results: ['16.1 Mbits/sec', '18.4 Mbits/sec']
    iperf h1 h5 result:
    mininet> iperf h1 h5
    *** Iperf: testing TCP bandwidth between h1 and h5
    *** Results: ['9.46 Mbits/sec', '10.7 Mbits/sec']
    The TCP bandwidth is more for h1 and h2 as it has fewer hops than h1 h5.

    c. Pingall results:
    *** Ping: testing ping reachability
    h1 -> h2 h3 h4 h5
    h2 -> h1 h3 h4 h5
    h3 -> h1 h2 h4 h5
    h4 -> h1 h2 h3 h5
    h5 -> h1 h2 h3 h4

*** Results: 0% dropped (20/20 received)

2. Question 2:

    a. h1 ping h2: Average RTT = 27.883ms
        Traffic was observed for h2, s1
        A single packet on h3, h4, h5, s2, s3 when ran the first time. No traffic observed from second run onwards on these.
    h1 ping h5: Average RTT = 33.836ms
        Traffic was observed for h5, s1, s2, s3
        A single packet on h2, h3, h4 when ran the first time. No traffic observed from second run onwards.
    No difference was observed for pings.
    Differences from HUB controller:
        - Traffic is not directed to all the routes as the flow rules are installed.
        - The controller makes the choice as to where the packets will be directed.
        - A single packet was observed on all nodes which was needed to install the initial rules on the controller.

    b. iperf h1 h2 result:
    mininet> iperf h1 h2
    *** Iperf: testing TCP bandwidth between h1 and h2
    *** Results: ['17.2 Mbits/sec', '19.6 Mbits/sec']

    Iperf h1 h5 result:
    mininet> iperf h1 h5
    *** Iperf: testing TCP bandwidth between h1 and h5
    *** Results: ['4.96 Mbits/sec', '5.84 Mbits/sec']

    A difference can be noted in the throughput which got significantly lower for h1 h5 after installing rules in the controller. This is because only s1 is needed to reach h2 while h1 to h5 will have a lot of hops.

    Difference from HUB controller:
    There isn't much difference in terms of bandwidth.

    c. Pingall results:
    mininet> pingall
    *** Ping: testing ping reachability
    h1 -> h2 h3 h4 h5
    h2 -> h1 h3 h4 h5
    h3 -> h1 h2 h4 h5
    h4 -> h1 h2 h3 h5

h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)

3. Question 3:
    a. h1 ping h2: Average RTT = 0.076ms
        Traffic was observed for h2, s1
        A single packet on h3, h4, h5, s2, s3 when ran the first time. No traffic observed from second run onwards.
    h1 ping h5: Average RTT = 0.097ms
        Traffic was observed for h5, s1, s2, s3
        A single packet on h2, h3, h4 when ran the first time. No traffic observed from second run onwards.
    There isn't any significant difference in RTT. Although, RTT for h1 ping h5 is higher as they are farther away.
    Difference from part1:
    - The switches have been made smart and learn the flow rules. All requests are not sent to the controller. Switches can decide where to send the traffic.
    - A single packet is observed on all nodes for installing the rules initially.

    b. iperf h1 h2 result:
    mininet> iperf h1 h2
    *** Iperf: testing TCP bandwidth between h1 and h2
    *** Results: ['34.6 Gbits/sec', '34.7 Gbits/sec']

    Iperf h1 h5 result:
    mininet> iperf h1 h5
    *** Iperf: testing TCP bandwidth between h1 and h5
    *** Results: ['36.5 Gbits/sec', '36.6 Gbits/sec']

    c. Pingall results:
    mininet> pingall
    *** Ping: testing ping reachability
    h1 -> h2 h3 h4 h5
    h2 -> h1 h3 h4 h5
    h3 -> h1 h2 h4 h5
    h4 -> h1 h2 h3 h5
    h5 -> h1 h2 h3 h4
    *** Results: 0% dropped (20/20 received)

    d. ovs-ofctl dump-flows s2 results after pingall:

Optimization done to improve the number of flows which can be found after the results.

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.349s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=6,
icmp,vlan_tci=0x0000,dl_src=c6:bf:27:b7:93:95,dl_dst=ba:af:66:3e:f3:4b,nw_src=
10.0.0.5,nw_dst=10.0.0.3,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:1
 cookie=0x0, duration=7.864s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=7,
icmp,vlan_tci=0x0000,dl_src=c6:bf:27:b7:93:95,dl_dst=42:d7:56:fa:cd:bd,nw_src
=10.0.0.5,nw_dst=10.0.0.2,nw_tos=0,icmp_type=0,icmp_code=0
actions=output:1
 cookie=0x0, duration=6.647s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=6,
icmp,vlan_tci=0x0000,dl_src=c6:bf:27:b7:93:95,dl_dst=c6:b6:ea:83:6c:0c,nw_src
=10.0.0.5,nw_dst=10.0.0.1,nw_tos=0,icmp_type=8,icmp_code=0
actions=output:1
 cookie=0x0, duration=8.492s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=8,
icmp,vlan_tci=0x0000,dl_src=c6:bf:27:b7:93:95,dl_dst=c6:b6:ea:83:6c:0c,nw_src
=10.0.0.5,nw_dst=10.0.0.1,nw_tos=0,icmp_type=0,icmp_code=0
actions=output:1
 cookie=0x0, duration=8.784s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=8,
icmp,vlan_tci=0x0000,dl_src=c6:b6:ea:83:6c:0c,dl_dst=82:f5:de:a2:35:6a,nw_src
=10.0.0.1,nw_dst=10.0.0.4,nw_tos=0,icmp_type=8,icmp_code=0
actions=output:2
 cookie=0x0, duration=8.712s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=8,
icmp,vlan_tci=0x0000,dl_src=82:f5:de:a2:35:6a,dl_dst=c6:b6:ea:83:6c:0c,nw_src
=10.0.0.4,nw_dst=10.0.0.1,nw_tos=0,icmp_type=0,icmp_code=0
actions=output:1
 cookie=0x0, duration=6.573s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=6,
icmp,vlan_tci=0x0000,dl_src=c6:b6:ea:83:6c:0c,dl_dst=c6:bf:27:b7:93:95,nw_src
=10.0.0.1,nw_dst=10.0.0.5,nw_tos=0,icmp_type=0,icmp_code=0
actions=output:2
 cookie=0x0, duration=6.94s, table=0, n_packets=0, n_bytes=0, idle_timeout=60,
hard_timeout=600, idle_age=6,
icmp,vlan_tci=0x0000,dl_src=82:f5:de:a2:35:6a,dl_dst=ba:af:66:3e:f3:4b,nw_src=
10.0.0.4,nw_dst=10.0.0.3,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:1
```

cookie=0x0, duration=8.16s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=8, icmp,vlan_tci=0x0000,dl_src=42:d7:56:fa:cd:bd,dl_dst=82:f5:de:a2:35:6a,nw_src=10.0.0.2,nw_dst=10.0.0.4,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:2
cookie=0x0, duration=7.307s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=7, icmp,vlan_tci=0x0000,dl_src=c6:bf:27:b7:93:95,dl_dst=ba:af:66:3e:f3:4b,nw_src=10.0.0.5,nw_dst=10.0.0.3,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:1
cookie=0x0, duration=7.166s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=7, icmp,vlan_tci=0x0000,dl_src=c6:b6:ea:83:6c:0c,dl_dst=82:f5:de:a2:35:6a,nw_src=10.0.0.1,nw_dst=10.0.0.4,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:2
cookie=0x0, duration=6.275s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=6, icmp,vlan_tci=0x0000,dl_src=ba:af:66:3e:f3:4b,dl_dst=c6:bf:27:b7:93:95,nw_src=10.0.0.3,nw_dst=10.0.0.5,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:2
cookie=0x0, duration=7.268s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=7, icmp,vlan_tci=0x0000,dl_src=82:f5:de:a2:35:6a,dl_dst=c6:b6:ea:83:6c:0c,nw_src=10.0.0.4,nw_dst=10.0.0.1,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:1
cookie=0x0, duration=7.348s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=7, icmp,vlan_tci=0x0000,dl_src=ba:af:66:3e:f3:4b,dl_dst=c6:bf:27:b7:93:95,nw_src=10.0.0.3,nw_dst=10.0.0.5,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:2
cookie=0x0, duration=7.568s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=7, icmp,vlan_tci=0x0000,dl_src=ba:af:66:3e:f3:4b,dl_dst=82:f5:de:a2:35:6a,nw_src=10.0.0.3,nw_dst=10.0.0.4,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:2
cookie=0x0, duration=7.017s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=7, icmp,vlan_tci=0x0000,dl_src=42:d7:56:fa:cd:bd,dl_dst=82:f5:de:a2:35:6a,nw_src=10.0.0.2,nw_dst=10.0.0.4,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:2
cookie=0x0, duration=6.867s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=6, icmp,vlan_tci=0x0000,dl_src=ba:af:66:3e:f3:4b,dl_dst=82:f5:de:a2:35:6a,nw_src=10.0.0.3,nw_dst=10.0.0.4,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:2
cookie=0x0, duration=8.084s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=8, icmp,vlan_tci=0x0000,dl_src=82:f5:de:a2:35:6a,dl_dst=42:d7:56:fa:cd:bd,nw_src

=10.0.0.4,nw_dst=10.0.0.2,nw_tos=0,icmp_type=0,icmp_code=0
actions=output:1
 cookie=0x0, duration=6.422s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=6,
icmp,vlan_tci=0x0000,dl_src=42:d7:56:fa:cd:bd,dl_dst=c6:bf:27:b7:93:95,nw_src
=10.0.0.2,nw_dst=10.0.0.5,nw_tos=0,icmp_type=0,icmp_code=0
actions=output:2
 cookie=0x0, duration=6.498s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=6,
icmp,vlan_tci=0x0000,dl_src=c6:bf:27:b7:93:95,dl_dst=42:d7:56:fa:cd:bd,nw_src
=10.0.0.5,nw_dst=10.0.0.2,nw_tos=0,icmp_type=8,icmp_code=0
actions=output:1
 cookie=0x0, duration=7.936s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=7,
icmp,vlan_tci=0x0000,dl_src=42:d7:56:fa:cd:bd,dl_dst=c6:bf:27:b7:93:95,nw_src
=10.0.0.2,nw_dst=10.0.0.5,nw_tos=0,icmp_type=8,icmp_code=0
actions=output:2
 cookie=0x0, duration=8.564s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=8,
icmp,vlan_tci=0x0000,dl_src=c6:b6:ea:83:6c:0c,dl_dst=c6:bf:27:b7:93:95,nw_src
=10.0.0.1,nw_dst=10.0.0.5,nw_tos=0,icmp_type=8,icmp_code=0
actions=output:2
 cookie=0x0, duration=7.496s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=7,
icmp,vlan_tci=0x0000,dl_src=82:f5:de:a2:35:6a,dl_dst=ba:af:66:3e:f3:4b,nw_src=
10.0.0.4,nw_dst=10.0.0.3,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:1
 cookie=0x0, duration=7.09s, table=0, n_packets=0, n_bytes=0, idle_timeout=60,
hard_timeout=600, idle_age=7,
icmp,vlan_tci=0x0000,dl_src=82:f5:de:a2:35:6a,dl_dst=42:d7:56:fa:cd:bd,nw_src
=10.0.0.4,nw_dst=10.0.0.2,nw_tos=0,icmp_type=8,icmp_code=0
actions=output:1
 cookie=0x0, duration=7.42s, table=0, n_packets=0, n_bytes=0, idle_timeout=60,
hard_timeout=600, idle_age=7,
arp,vlan_tci=0x0000,dl_src=c6:bf:27:b7:93:95,dl_dst=ba:af:66:3e:f3:4b,arp_spa=
10.0.0.5,arp_tpa=10.0.0.3,arp_op=2 actions=output:1
 cookie=0x0, duration=1.888s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=1,
arp,vlan_tci=0x0000,dl_src=ba:af:66:3e:f3:4b,dl_dst=82:f5:de:a2:35:6a,arp_spa=
10.0.0.3,arp_tpa=10.0.0.4,arp_op=2 actions=output:2
 cookie=0x0, duration=2.037s, table=0, n_packets=0, n_bytes=0,
idle_timeout=60, hard_timeout=600, idle_age=2,
arp,vlan_tci=0x0000,dl_src=42:d7:56:fa:cd:bd,dl_dst=82:f5:de:a2:35:6a,arp_spa
=10.0.0.2,arp_tpa=10.0.0.4,arp_op=2 actions=output:2

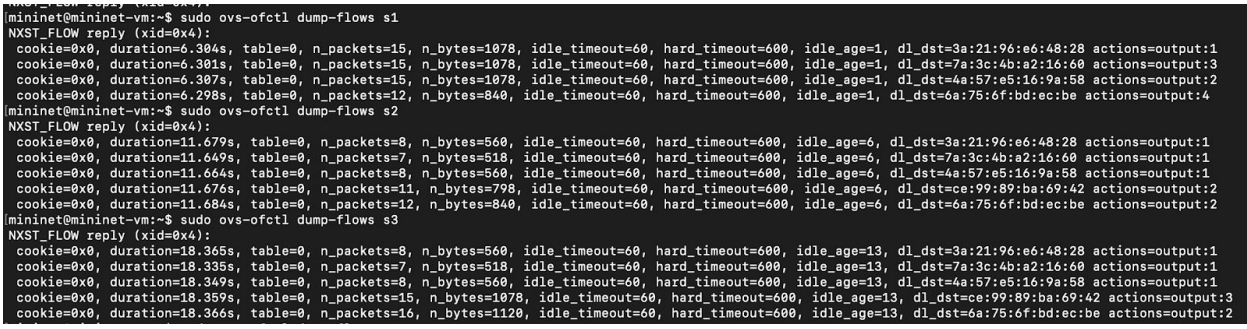
cookie=0x0, duration=2.244s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=2, arp,vlan_tci=0x0000,dl_src=ba:af:66:3e:f3:4b,dl_dst=c6:bf:27:b7:93:95,arp_spa=10.0.0.3,arp_tpa=10.0.0.5,arp_op=2 actions=output:2
cookie=0x0, duration=1.96s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=1, arp,vlan_tci=0x0000,dl_src=82:f5:de:a2:35:6a,dl_dst=ba:af:66:3e:f3:4b,arp_spa=10.0.0.4,arp_tpa=10.0.0.3,arp_op=1 actions=output:1
cookie=0x0, duration=2.244s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=2, arp,vlan_tci=0x0000,dl_src=82:f5:de:a2:35:6a,dl_dst=c6:b6:ea:83:6c:0c,arp_spa=10.0.0.4,arp_tpa=10.0.0.1,arp_op=1 actions=output:1
cookie=0x0, duration=8.636s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=8, arp,vlan_tci=0x0000,dl_src=c6:bf:27:b7:93:95,dl_dst=c6:b6:ea:83:6c:0c,arp_spa=10.0.0.5,arp_tpa=10.0.0.1,arp_op=2 actions=output:1
cookie=0x0, duration=3.5s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=3, arp,vlan_tci=0x0000,dl_src=c6:b6:ea:83:6c:0c,dl_dst=c6:bf:27:b7:93:95,arp_spa=10.0.0.1,arp_tpa=10.0.0.5,arp_op=2 actions=output:2
cookie=0x0, duration=7.64s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=7, arp,vlan_tci=0x0000,dl_src=82:f5:de:a2:35:6a,dl_dst=ba:af:66:3e:f3:4b,arp_spa=10.0.0.4,arp_tpa=10.0.0.3,arp_op=2 actions=output:1
cookie=0x0, duration=8.856s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=8, arp,vlan_tci=0x0000,dl_src=82:f5:de:a2:35:6a,dl_dst=c6:b6:ea:83:6c:0c,arp_spa=10.0.0.4,arp_tpa=10.0.0.1,arp_op=2 actions=output:1
cookie=0x0, duration=2.918s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=2, arp,vlan_tci=0x0000,dl_src=c6:bf:27:b7:93:95,dl_dst=42:d7:56:fa:cd:bd,arp_spa=10.0.0.5,arp_tpa=10.0.0.2,arp_op=1 actions=output:1
cookie=0x0, duration=8.232s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=8, arp,vlan_tci=0x0000,dl_src=82:f5:de:a2:35:6a,dl_dst=42:d7:56:fa:cd:bd,arp_spa=10.0.0.4,arp_tpa=10.0.0.2,arp_op=2 actions=output:1
cookie=0x0, duration=2.323s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=2, arp,vlan_tci=0x0000,dl_src=c6:bf:27:b7:93:95,dl_dst=ba:af:66:3e:f3:4b,arp_spa=10.0.0.5,arp_tpa=10.0.0.3,arp_op=1 actions=output:1
cookie=0x0, duration=3.573s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=3,

arp,vlan_tci=0x0000,dl_src=c6:bf:27:b7:93:95,dl_dst=c6:b6:ea:83:6c:0c,arp_spa=10.0.0.5,arp_tpa=10.0.0.1,arp_op=1 actions=output:1
 cookie=0x0, duration=8.008s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=8, arp,vlan_tci=0x0000,dl_src=c6:bf:27:b7:93:95,dl_dst=42:d7:56:fa:cd:bd,arp_spa=10.0.0.5,arp_tpa=10.0.0.2,arp_op=2 actions=output:1
 cookie=0x0, duration=2.84s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=2, arp,vlan_tci=0x0000,dl_src=42:d7:56:fa:cd:bd,dl_dst=c6:bf:27:b7:93:95,arp_spa=10.0.0.2,arp_tpa=10.0.0.5,arp_op=2 actions=output:2
 cookie=0x0, duration=2.196s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=2, arp,vlan_tci=0x0000,dl_src=c6:b6:ea:83:6c:0c,dl_dst=82:f5:de:a2:35:6a,arp_spa=10.0.0.1,arp_tpa=10.0.0.4,arp_op=2 actions=output:2
 cookie=0x0, duration=2.108s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=600, idle_age=2, arp,vlan_tci=0x0000,dl_src=82:f5:de:a2:35:6a,dl_dst=42:d7:56:fa:cd:bd,arp_spa=10.0.0.4,arp_tpa=10.0.0.2,arp_op=1 actions=output:1

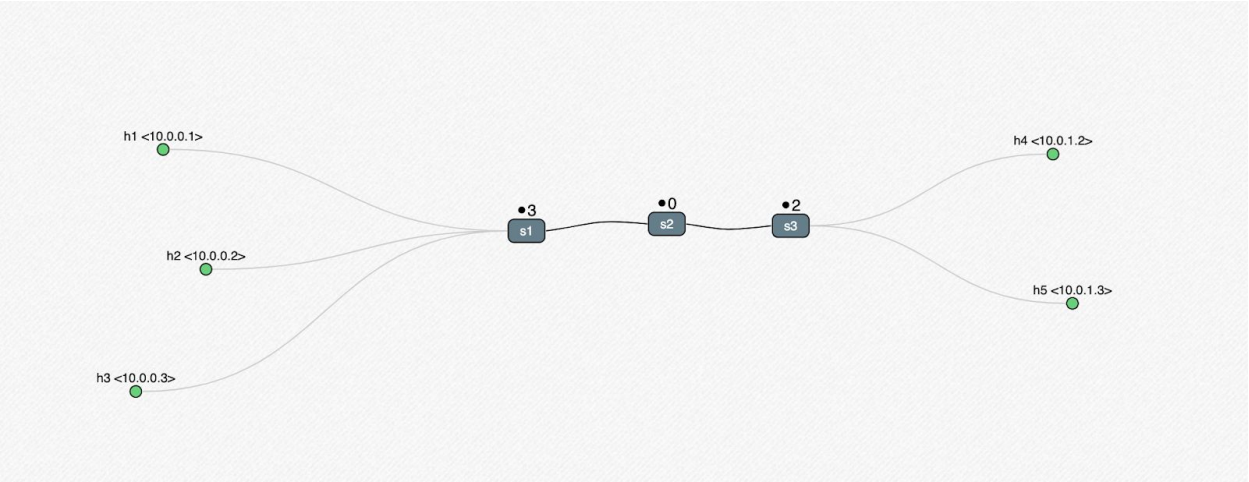
Above is the output of the flow rules which were installed, the rules are getting appended with each ping. The distinct rules are much less than the total number of rules. They are getting duplicated.

BONUS:

Used msg.match.dl_dst = packet.dst instead of of.ofp_match.from_packet to match the packets and reduce the number of rules. Now, s1 has 4 rules, s2 and s3 have 5 rules.



4. Change in topology for the last part:



Installed flows:

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=56.16s, table=0, n_packets=0, n_bytes=0, idle_age=56, icmp,nw_dst=10.0.1.0/24 actions=output:2
 cookie=0x0, duration=56.154s, table=0, n_packets=0, n_bytes=0, idle_age=56, icmp,nw_dst=10.0.0.0/24 actions=output:1
 cookie=0x0, duration=56.143s, table=0, n_packets=0, n_bytes=0, idle_age=56, arp,arp_tpa=10.0.0.0/24 actions=output:1
 cookie=0x0, duration=56.148s, table=0, n_packets=0, n_bytes=0, idle_age=56, arp,arp_tpa=10.0.1.0/24 actions=output:2
mininet@mininet-vm:~$
```

a. h1 ping h2 results:
   Average RTT: 0.079ms
   Traffic was observed for s1 and h2
   A single packet on s2 and h3 when ran the first time. No traffic observed from second run onwards.

   h1 ping h5 results:
   Average RTT: 0.077ms
   Traffic was observed for s1,s2,s3,h5
   A single packet on h2,h3 when ran the first time. No traffic observed from second run onwards.

   There is not much significant difference between the ping results for h1 h2 and h1 h5.
   Difference from the earlier controllers:
   - Average RTT is much less compared to part-A while it is almost the same for rest.

b. iperf h1 h2
   mininet> iperf h1 h2
   *** Iperf: testing TCP bandwidth between h1 and h2
   *** Results: ['34.4 Gbits/sec', '34.5 Gbits/sec']

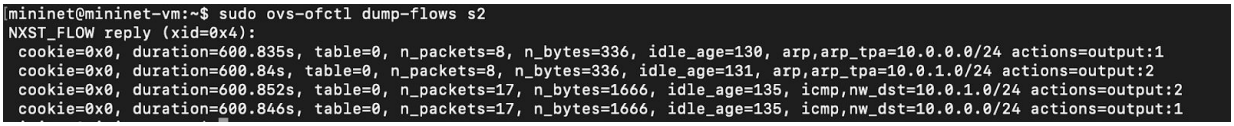
iperf h1 h5
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['34.0 Gbits/sec', '34.0 Gbits/sec']

There is not much significant difference between the iperf results for h1 h2 and h1 h5.
Difference from the earlier controllers:

- Bandwidth is significantly more when compared to part-A and part-B.

c. Pingall results
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)

d. ovs-ofctl dump-flows s2 results:



There are 4 rules which are the same as the ones we have installed initially. For every action, these are updated. Two are ICMP while the other two are ARP.

e. Comparison with previous controller:

- s2 is made smart to do routing and no more flows are installed as seen in the image in d.
- IP-matching rules improves the performance.
- The same rules are updated each time instead of appending duplicates.

This is better as compared to the previous controllers as the switch is configured for routing the necessary predetermined traffic. The same set of rules are updated, controller is less burdened and layer-3 routing improves the performance.

Appendix:

1. **Code for setting up the topology:**

```
from mininet.topo import Topo
```

```
class MyTopo( Topo ):
   "Simple topology example."

   def __init__( self ):
      "Create custom topo."

      # Initialize topology
      Topo.__init__( self )

      # Add hosts and switches
      leftHost1 = self.addHost( 'h1' )
            leftHost2 = self.addHost( 'h2' )
            leftHost3 = self.addHost( 'h3' )
      rightHost1 = self.addHost( 'h4' )
            rightHost2 = self.addHost( 'h5' )
      leftSwitch = self.addSwitch( 's1' )
            centerSwitch = self.addSwitch('s2')
      rightSwitch = self.addSwitch( 's3' )

      # Add links
      self.addLink( leftHost1, leftSwitch )
            self.addLink( leftHost2, leftSwitch )
            self.addLink( leftHost3, leftSwitch )
      self.addLink( leftSwitch, centerSwitch )
            self.addLink( centerSwitch, rightSwitch )
      self.addLink( rightSwitch, rightHost1 )
            self.addLink( rightSwitch, rightHost2 )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

2. Code for of_tutorial.py(changes in yellow)

```
from pox.core import core
import pox.openflow.libopenflow_01 as of

log = core.getLogger()

class Tutorial (object):
  """
  A Tutorial object is created for each switch that connects.
  A Connection object for that switch is passed to the __init__ function.
  """
  def __init__ (self, connection):
    # Keep track of the connection to the switch so that we can
    # send it messages!
    self.connection = connection

    # This binds our PacketIn event listener
    connection.addListeners(self)

    # Use this table to keep track of which ethernet address is on
```

```
  # which switch port (keys are MACs, values are ports).
  self.mac_to_port = {}


def resend_packet (self, packet_in, out_port):
  """
  Instructs the switch to resend a packet that it had sent to us.
  "packet_in" is the ofp_packet_in object the switch had sent to the
  controller due to a table-miss.
  """
  msg = of.ofp_packet_out()
  msg.data = packet_in

  # Add an action to send to the specified port
  action = of.ofp_action_output(port = out_port)
  msg.actions.append(action)

  # Send message to switch
  self.connection.send(msg)


def act_like_hub (self, packet, packet_in):
  """
  Implement hub-like behavior -- send all packets to all ports besides
  the input port.
  """

  # We want to output to all ports -- we do that using the special
  # OFPP_ALL port as the output port.  (We could have also used
  # OFPP_FLOOD.)
  self.resend_packet(packet_in, of.OFPP_ALL)

  # Note that if we didn't get a valid buffer_id, a slightly better
  # implementation would check that we got the full data before
  # sending it (len(packet_in.data) should be == packet_in.total_len)).


def act_like_switch (self, packet, packet_in):

  # Learn the port for the source MAC
  if packet.src not in self.mac_to_port:
    log.debug("Learned %s from Port %d!" % (packet.src, packet_in.in_port))
    self.mac_to_port[packet.src] = packet_in.in_port

  if packet.dst in self.mac_to_port:
    # Send packet out the associated port
    log.debug("CAM table hit, sending out packet to Port %d" % self.mac_to_port[packet.dst])
    self.resend_packet(packet_in, self.mac_to_port[packet.dst])

  else:
    # Flood the packet out everything but the input port
    self.resend_packet(packet_in, of.OFPP_ALL)
```

```
  def _handle_PacketIn (self, event):
    """
    Handles packet in messages from the switch.
    """

    packet = event.parsed # This is the parsed packet data.
    if not packet.parsed:
      log.warning("Ignoring incomplete packet")
      return

    packet_in = event.ofp # The actual ofp_packet_in message.

    # Comment out the following line and uncomment the one after
    # when starting the exercise.
   # self.act_like_hub(packet, packet_in)
    self.act_like_switch(packet, packet_in)



def launch ():
  """
  Starts the component
  """
  def start_switch (event):
    log.debug("Controlling %s" % (event.connection,))
    Tutorial(event.connection)
  core.openflow.addListenerByName("ConnectionUp", start_switch)
```

3. Code for of_tutorial.py(changes in yellow)

```
from pox.core import core
import pox.openflow.libopenflow_01 as of

log = core.getLogger()

class Tutorial (object):
  """
  A Tutorial object is created for each switch that connects.
  A Connection object for that switch is passed to the __init__ function.
  """
  def __init__ (self, connection):
    # Keep track of the connection to the switch so that we can
    # send it messages!
    self.connection = connection

    # This binds our PacketIn event listener
    connection.addListeners(self)

    # Use this table to keep track of which ethernet address is on
    # which switch port (keys are MACs, values are ports).
    self.mac_to_port = {}
```

```
def resend_packet (self, packet_in, out_port):
  """
  Instructs the switch to resend a packet that it had sent to us.
  "packet_in" is the ofp_packet_in object the switch had sent to the
  controller due to a table-miss.
  """
  msg = of.ofp_packet_out()
  msg.data = packet_in

  # Add an action to send to the specified port
  action = of.ofp_action_output(port = out_port)
  msg.actions.append(action)

  # Send message to switch
  self.connection.send(msg)


def act_like_hub (self, packet, packet_in):
  """
  Implement hub-like behavior -- send all packets to all ports besides
  the input port.
  """

  # We want to output to all ports -- we do that using the special
  # OFPP_ALL port as the output port.  (We could have also used
  # OFPP_FLOOD.)
  self.resend_packet(packet_in, of.OFPP_ALL)

  # Note that if we didn't get a valid buffer_id, a slightly better
  # implementation would check that we got the full data before
  # sending it (len(packet_in.data) should be == packet_in.total_len)).


 def act_like_switch (self, packet, packet_in):

   self.mac_to_port[packet.src] = packet_in.in_port

   if packet.dst in self.mac_to_port:
     # Send packet out the associated port
     log.debug("CAM table hit, sending out packet to Port %d" % self.mac_to_port[packet.dst])

     log.debug("Installing flow ...")
     #log.debug("MATCH: In Port =  %s" % packet_in.in_port)
     # log.debug("MATCH: Source MAC =  %s" % packet.src)
     log.debug("MATCH: Destination MAC =  %s" % packet.dst)
     log.debug("ACTION: Out Port =  %s" % self.mac_to_port[packet.dst])

     msg = of.ofp_flow_mod()
     msg.match.dl_dst = packet.dst
     msg.data = packet_in
     msg.actions.append(of.ofp_action_output(port=self.mac_to_port[packet.dst]))
     msg.idle_timeout = 60
     msg.hard_timeout = 600
     self.connection.send(msg)
```

```
    else:
      # Flood the packet out everything but the input port
      self.resend_packet(packet_in, of.OFPP_ALL)


  def _handle_PacketIn (self, event):
    """
    Handles packet in messages from the switch.
    """

    packet = event.parsed # This is the parsed packet data.
    if not packet.parsed:
      log.warning("Ignoring incomplete packet")
      return

    packet_in = event.ofp # The actual ofp_packet_in message.

    # Comment out the following line and uncomment the one after
    # when starting the exercise.
   # self.act_like_hub(packet, packet_in)
    self.act_like_switch(packet, packet_in)



def launch ():
  """
  Starts the component
  """
  def start_switch (event):
    log.debug("Controlling %s" % (event.connection,))
    Tutorial(event.connection)
  core.openflow.addListenerByName("ConnectionUp", start_switch)
```

4. Command to install flows on s2 while s1 and s3 are MAC learning switches:

```
sudo ovs-ofctl add-flow s2 icmp,nw_dst=10.0.1.0/24,actions=output:2
sudo ovs-ofctl add-flow s2 icmp,nw_dst=10.0.0.0/24,actions=output:1
sudo ovs-ofctl add-flow s2 arp,arp_tpa=10.0.1.0/24,actions=output:2
sudo ovs-ofctl add-flow s2 arp,arp_tpa=10.0.0.0/24,actions=output:1
```

Assigned IP:

```
from mininet.topo import Topo

class MyTopo( Topo ):

   def __init__( self ):

      # Initialize topology
      Topo.__init__( self )

      # Add hosts and switches
```

```
        leftHost1 = self.addHost( 'h1', ip = '10.0.0.1' )
        leftHost2 = self.addHost( 'h2', ip = '10.0.0.2' )
        leftHost3 = self.addHost( 'h3', ip = '10.0.0.3' )
        rightHost1 = self.addHost( 'h4', ip = '10.0.1.2' )
        rightHost2 = self.addHost( 'h5', ip = '10.0.1.3' )
        leftSwitch = self.addSwitch( 's1' )
        centerSwitch = self.addSwitch('s2')
        rightSwitch = self.addSwitch( 's3' )

        # Add links
        self.addLink( leftHost1, leftSwitch )
        self.addLink( leftHost2, leftSwitch )
        self.addLink( leftHost3, leftSwitch )
        self.addLink( leftSwitch, centerSwitch )
        self.addLink( centerSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost1 )
        self.addLink( rightSwitch, rightHost2 )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```