Q1

A Software Defined Network, SDN, is a network whose Control Plane is separated from it’s Data Plane through software making a network configurable through Application Programmable Interfaces, APIs rather than having to physically connect to possibly hundreds of switches and routers to configure them.

Q2

Continuing this configuration is done by an SDN controller which manages the entire networks forwarding decisions/schemes. The SDN controller is the software that the network administrator< uses.

Q3

An SDN switch is a a switch that runs a SDN protocol like OpenFlow and is configurable by an SDN controller. These can be pure SDN switches that is fully reliant on SDN, Hybrid that runs both SDN and traditional swicthing configuration and Virtual Switches (vSwitch) that are fully software-based switches that resides strictly in cloud and virtualized environments.

Q4

Match-plus-action is an SDN principle whilst desitination-based routing is a traditional principle. Destination-based routing means that packets that are forwarded in a router or switch are based only on it’s destination IP adress that is matched with the switch MAC table or routers routing table. Match-plus-action examines packets incoming and can match them against rules that now can have multiple attributes and after rules are applied performs an action like forward, drop or modify. It can apply priority, ports firewall rules filtering etc all since the Data Plane, control plane and management plane now is abstracted.

Q5

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=16425>
<Host h2: h2-eth0:10.0.0.2 pid=16427>
<OVSSwitch{'port': 6634} s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=16432>
<RemoteController{'port': 6653} c0: 127.0.0.1:6653 pid=16419>
mininet>
```

Q6

```
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
mininet>
```

Q7

```
mininet> h1 ifconfig -a
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        ether 00:00:00:00:00:01  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet>
```

Q8

```
mininet> h2 ifconfig -a
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.2  netmask 255.0.0.0  broadcast 10.255.255.255
        ether 00:00:00:00:00:02  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet>
```

Q9

```
mininet> s1 ifconfig -a
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        ether 08:00:27:15:3c:55  txqueuelen 1000  (Ethernet)
        RX packets 691043  bytes 943456769 (943.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 50462  bytes 3175877 (3.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 885  bytes 61812 (61.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 885  bytes 61812 (61.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ovs-system: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 96:2f:59:d4:89:ea  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether ea:b5:26:21:ed:4f  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether e6:b8:c7:2f:69:86  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 32:4c:29:a7:f4:05  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet>
```

Q10

```
mininet> h1 ping -c4 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
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3064ms
pipe 4
mininet>
```

Q11

```
mininet> h2 ping -c4 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable
From 10.0.0.2 icmp_seq=3 Destination Host Unreachable
From 10.0.0.2 icmp_seq=4 Destination Host Unreachable

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3065ms
pipe 4
mininet>
```

Q12

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

The reason is that as of yet we have no flow rules to determine routes inside the network so packets are by default dropped due to the lack of rules.

Q13

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
mininet>
```

Q14

```
mininet@mininet-VirtualBox:~$ sudo ovs-ofctl dump-flows s1
 cookie=0x0, duration=68.313s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s1-eth1" actions=output:"s1-eth2"
 cookie=0x0, duration=33.503s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s1-eth2" actions=output:"s1-eth1"
mininet@mininet-VirtualBox:~$
```

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=160.534s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s1-eth1" actions=output:"s1-eth2"
 cookie=0x0, duration=125.724s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s1-eth2" actions=output:"s1-eth1"
mininet>
```

Q15

Cookie =0x0 means no cookie was attached meaning the flow rule it's not set up to be modified elsewhere.

Duration is how long the flow rule has been active

Table=0 means these flow rules are the first entry in the table of flow rules.

n_packets is a value of how many packets have been matched with the specific rule

n_bytes is how many bytes have been matched with the specific rule

priority, which we set to 1 in the config line, is applied when there are multiple rules that apply to a packet. Priority 1 is low the rule with the highest priority is the one that is going to apply.

in_port is the port on the switch where the rule applies for in coming packets. We specified this with alias names 1 and 2 in the command line and here their full name is viewed.

Actions is where the rule applies it's action(drop,forward etc.) on. In this case it's to forward(output) to the other rules in-port.

Q16

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

It works now since the flow rules have port forwarded outputs to corresponding inputs of the hosts.

Q17

```
mininet> dpctl del-flows
*** s1 ------------------------------------------------------------------------
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)
mininet>
```

No, ping packets dropped and as you can see the output of the dpctl del-flows is a dump of the flows which shows there are none. Del-flows also suspiciosly looks like an abbreviation of delete flows.

Q18

Destination	Protocol	Length	Info
127.0.0.1	OpenFl…	74	Type: OFPT_HELLO
127.0.0.1	OpenFl…	74	Type: OFPT_FEATURES_REQUEST
127.0.0.1	OpenFl…	98	Type: OFPT_FEATURES_REPLY
127.0.0.1	OpenFl…	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
127.0.0.1	OpenFl…	274	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
127.0.0.1	OpenFl…	146	Type: OFPT_FLOW_MOD

OpenFlow Protocol Type, OFPT: Hello, Features Request and Reply, Multipart Message Port Descriptions Request and Reply as well as a Flow Modifyer (add, modify or delete flow entries).

Q19

```
127.0.0.1          OpenFl…    150 Type: OFPT_PACKET_IN
127.0.0.1          OpenFl…    148 Type: OFPT_PACKET_OUT
127.0.0.1          OpenFl…    150 Type: OFPT_PACKET_IN
127.0.0.1          OpenFl…    170 Type: OFPT_FLOW_MOD
127.0.0.1          OpenFl…    148 Type: OFPT_PACKET_OUT
127.0.0.1          OpenFl…    206 Type: OFPT_PACKET_IN
127.0.0.1          OpenFl…    170 Type: OFPT_FLOW_MOD
127.0.0.1          OpenFl…    204 Type: OFPT_PACKET_OUT
```

The FLOW_MOD message adds a flow entry.

```
mininet> h1 ping -c4 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=5.20 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.099 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.068 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.032 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3043ms
rtt min/avg/max/mdev = 0.032/1.351/5.205/2.225 ms
mininet>
```

The first ping was slower due to the time needed for configuration of flow.

```
▾ Match
    Type: OFPMT_OXM (1)
    Length: 32
  ▾ OXM field
      Class: OFPXMC_OPENFLOW_BASIC (0x8000)
      0000 000. = Field: OFPXMT_OFB_IN_PORT (0)
      .... ...0 = Has mask: False
      Length: 4
      Value: 2
  ▾ OXM field
      Class: OFPXMC_OPENFLOW_BASIC (0x8000)
      0000 011. = Field: OFPXMT_OFB_ETH_DST (3)
      .... ...0 = Has mask: False
      Length: 6
      Value: 00:00:00_00:00:01 (00:00:00:00:00:01)
  ▾ OXM field
      Class: OFPXMC_OPENFLOW_BASIC (0x8000)
      0000 100. = Field: OFPXMT_OFB_ETH_SRC (4)
      .... ...0 = Has mask: False
      Length: 6
      Value: 00:00:00_00:00:02 (00:00:00:00:00:02)
▾ Instruction
    Type: OFPIT_APPLY_ACTIONS (4)
    Length: 24
    Pad: 00000000
  ▾ Action
      Type: OFPAT_OUTPUT (0)
      Length: 16
      Port: 1
      Max length: 65509
      Pad: 000000000000
```

```
Cookie: 0x0000000000000000
Cookie mask: 0x0000000000000000
Table ID: 0
Command: OFPFC_ADD (0)
Idle timeout: 0
Hard timeout: 0
Priority: 1
Buffer ID: OFP_NO_BUFFER (4294967295)
Out port: 0
Out group: 0
▸ Flags: 0x0000
Pad: 0000
```

OpenFlow Extensible Match, OXM, fields specify input port of the switch, source MAC and destination MAC. The instruction field explains the action the flow rule is supposed to apply OpenFlow Protocol Action Type, OFPAT, which is output(forward). To the right is the cookie info, Table ID and Priority(1). This is the info we set manually before and got as the output of the flow dump.

Q20

```
mininet> h1 ping -c4 h4
PING 10.0.0.4 (10.0.0.4) 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
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable

--- 10.0.0.4 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3055ms
pipe 4
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
*** s2 ------------------------------------------------------------------------
*** s3 ------------------------------------------------------------------------
mininet>
```

Destination	Protocol	Length	Info
Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.1
Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.1
Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.1
Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.1
Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.1
Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.1

ARP frames.

Q21

ARP aims to find the Media Access Protocol, MAC, address of the device that has a specific IP address and then return that MAC address to the IP address of who asked for it. The ARPs corresponds to the packets since it never gets to know the MAC address due to no flow rules and therefore keeps asking.

Q22

The problem is when a switch recieves a broadcast from a switch and then broadcast the same request if the first switch were to broadcast again we would have an infinite loop and therefore if a switch has broadcast a request it should have only handle the request and then drop each echoed request.

Q23

```
#arp packets are small and could therefore benefit low latency(link 3)
sudo ovs-ofctl add-flow s1 table=0,priority=1,action=drop
sudo ovs-ofctl add-flow s1 table=0,priority=10,arp,action=output:"1 2 3"
#if it's coming from link 3 don't bounce the request back to avoid crowding the links
sudo ovs-ofctl add-flow s1 table=0,priority=100,in_port=3,arp,action=output:"1 2"


sudo ovs-ofctl add-flow s2 table=0,priority=1,action=drop
sudo ovs-ofctl add-flow s2 table=0,priority=10,arp,action=output:"1 2 3"
#if it's coming from link 3 don't bounce the request back to avoid crowding the links
sudo ovs-ofctl add-flow s2 table=0,priority=100,in_port=3,arp,action=output:"1 2"
```

Now ARP traffic going from switch to switch dies out when it reaches the host connected switches.

Q24

```
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>
```

1	0.000000000	00:00:00_00:00:01	Broadcast	ARP	42 Who has 10.0.0.2? Tell 10.0.0.1
2	0.003206699	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42 10.0.0.2 is at 00:00:00:00:00:02
3	0.004224482	10.0.0.1	10.0.0.2	ICMP	98 Echo (ping) request id=0x0cc3, se
4	10.002827970	00:00:00_00:00:01	Broadcast	ARP	42 Who has 10.0.0.3? Tell 10.0.0.1
5	10.012328065	00:00:00_00:00:03	00:00:00_00:00:01	ARP	42 10.0.0.3 is at 00:00:00:00:00:03
6	10.013395389	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request id=0x0cc5, se
7	20.031676943	10.0.0.1	10.0.0.4	ICMP	98 Echo (ping) request id=0x0cc7, se
8	25.048823995	00:00:00_00:00:01	00:00:00_00:00:04	ARP	42 Who has 10.0.0.4? Tell 10.0.0.1
9	25.063701846	00:00:00_00:00:04	00:00:00_00:00:01	ARP	42 10.0.0.4 is at 00:00:00:00:00:04
10	35.275200426	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42 Who has 10.0.0.1? Tell 10.0.0.2
11	35.276682558	00:00:00_00:00:01	00:00:00_00:00:02	ARP	42 10.0.0.1 is at 00:00:00:00:00:01
12	40.047559656	00:00:00_00:00:02	Broadcast	ARP	42 Who has 10.0.0.3? Tell 10.0.0.2
13	40.060215486	00:00:00_00:00:03	00:00:00_00:00:02	ARP	42 10.0.0.3 is at 00:00:00:00:00:03
14	50.056300803	00:00:00_00:00:02	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.2
15	50.065170535	00:00:00_00:00:04	00:00:00_00:00:02	ARP	42 10.0.0.4 is at 00:00:00:00:00:04
16	65.230200701	00:00:00_00:00:03	00:00:00_00:00:01	ARP	42 Who has 10.0.0.1? Tell 10.0.0.3
17	65.231368982	00:00:00_00:00:01	00:00:00_00:00:03	ARP	42 10.0.0.1 is at 00:00:00:00:00:01
18	75.217173789	00:00:00_00:00:03	00:00:00_00:00:02	ARP	42 Who has 10.0.0.2? Tell 10.0.0.3
19	75.219562245	00:00:00_00:00:02	00:00:00_00:00:03	ARP	42 10.0.0.2 is at 00:00:00:00:00:02
20	80.092048278	00:00:00_00:00:03	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
21	80.094677449	00:00:00_00:00:04	00:00:00_00:00:03	ARP	42 10.0.0.4 is at 00:00:00:00:00:04
22	95.211140156	00:00:00_00:00:04	00:00:00_00:00:01	ARP	42 Who has 10.0.0.1? Tell 10.0.0.4
23	95.228189355	00:00:00_00:00:01	00:00:00_00:00:04	ARP	42 10.0.0.1 is at 00:00:00:00:00:01
24	105.187452210	00:00:00_00:00:04	00:00:00_00:00:02	ARP	42 Who has 10.0.0.2? Tell 10.0.0.4
25	105.205285379	00:00:00_00:00:02	00:00:00_00:00:04	ARP	42 10.0.0.2 is at 00:00:00:00:00:02
26	115.177274600	00:00:00_00:00:04	00:00:00_00:00:03	ARP	42 Who has 10.0.0.3? Tell 10.0.0.4
27	115.180255434	00:00:00_00:00:03	00:00:00_00:00:04	ARP	42 10.0.0.3 is at 00:00:00:00:00:03

It behaves as expected. It handles the Address Resolution but since ping uses ICMP which isn't ARP frame but an Internet Protocol datagram it gets automatically dropped.

Q25

```
mininet> h1 arp -a
? (10.0.0.4) at 00:00:00:00:00:04 [ether] on h1-eth0
? (10.0.0.3) at 00:00:00:00:00:03 [ether] on h1-eth0
? (10.0.0.2) at 00:00:00:00:00:02 [ether] on h1-eth0
mininet> h2 arp -a
? (10.0.0.1) at 00:00:00:00:00:01 [ether] on h2-eth0
? (10.0.0.4) at 00:00:00:00:00:04 [ether] on h2-eth0
? (10.0.0.3) at 00:00:00:00:00:03 [ether] on h2-eth0
mininet> h3 arp -a
? (10.0.0.4) at 00:00:00:00:00:04 [ether] on h3-eth0
? (10.0.0.1) at 00:00:00:00:00:01 [ether] on h3-eth0
? (10.0.0.2) at 00:00:00:00:00:02 [ether] on h3-eth0
mininet> h4 arp -a
? (10.0.0.1) at 00:00:00:00:00:01 [ether] on h4-eth0
? (10.0.0.2) at 00:00:00:00:00:02 [ether] on h4-eth0
? (10.0.0.3) at 00:00:00:00:00:03 [ether] on h4-eth0
mininet>
```

It maps IP addresses with their respective MAC addresses on a specific link in this case the hosts ethernet-links.

Q26

```
#This is just an exchangepoint everything is passed
sudo ovs-ofctl add-flow s3 table=0,priority=1,action=all

#High Bandwidth Connection for everything between h1 <-> h4
#output is the thing that should be restrictive
sudo ovs-ofctl add-flow s1 table=0,priority=5,ip,nw_src=10.0.0.1,action=output:5
sudo ovs-ofctl add-flow s2 table=0,priority=5,ip,nw_src=10.0.0.4,action=output:4

#input non arp traffic should have addressing sorted out
sudo ovs-ofctl add-flow s1 table=0,priority=5,ip,nw_dst=10.0.0.1,action=output:1
sudo ovs-ofctl add-flow s2 table=0,priority=5,ip,nw_dst=10.0.0.4,action=output:2
```

Relevant commands are: Priority=5 which is lower than ARP priority to not disturb ARP communications functionality. Ip to target Internet Protocol traffic (non Data Link only traffic like ARP). nw_src to set which link a host of that source IP address should use and nw_dst to match packets with a certain destination ip address to it’s respective port. I only specify which link a certain host uses and does not restrict which destination it uses since the specifications do not require a restrictive links I choose to keep possibilities open and perform restrictions later. Scalability.

Q27

```
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['2.75 Mbits/sec', '3.35 Mbits/sec']
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['1.76 Mbits/sec', '2.17 Mbits/sec']
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['2.92 Mbits/sec', '3.48 Mbits/sec']
mininet>
```

Between 1.76 and 3.48 Mbits/sec

Q28

```
mininet> h1 ping -c4 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=326 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=344 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=378 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=340 ms

--- 10.0.0.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 326.032/347.285/378.323/19.169 ms
mininet>
```

Average latency is 347ms since packet has to travel over the high latency link 2*150ms + 2*10ms = 320ms which also is almost the reported minimum latency (326ms).

Q29

```
#Low latency link between h2 <-> h3
#output is the thing that should be restrictive
sudo ovs-ofctl add-flow s1 table=0,priority=5,ip,nw_src=10.0.0.2,action=output:3
sudo ovs-ofctl add-flow s2 table=0,priority=5,ip,nw_src=10.0.0.3,action=output:3

#input non arp traffic should have addressing sorted out
sudo ovs-ofctl add-flow s1 table=0,priority=5,ip,nw_dst=10.0.0.2,action=output:2
sudo ovs-ofctl add-flow s2 table=0,priority=5,ip,nw_dst=10.0.0.3,action=output:1
```

Relevant commands are the same as Q26.

Q30

```
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['958 Kbits/sec', '1.56 Mbits/sec']
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['958 Kbits/sec', '1.56 Mbits/sec']
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['682 Kbits/sec', '1.26 Mbits/sec']
mininet>
```

Between 0.682 and 1.56 Mbit/sec.

Q31

```
mininet> h2 ping -c4 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=10.3 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=10.4 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=10.7 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=10.7 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3014ms
rtt min/avg/max/mdev = 10.309/10.570/10.796/0.252 ms
mininet>
```

Latency is about 10ms which is about right. 2*3ms = 6ms. The rest of the 4ms is the switching delay which can be seen in the discrepancy of Q28 where propagation-delay = 320ms and the real delay=326ms: 320-326=6ms switching delay. 6ms/3switches = 2ms/switch. 2ms * 2 switches = 4ms which confirmes the observation.

Q32

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 h4
h2 -> X h3 h4
h3 -> h1 h2 X
h4 -> h1 h2 X
*** Results: 33% dropped (8/12 received)
mininet>
```

No host is connected to it’s “network neighbor” only to the other network.

Q33

```
mininet> h3 ping -c4 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=168 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=168 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=168 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=178 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 168.084/170.818/178.048/4.199 ms
mininet>
```

Yes I can ping h1 from h3. Lowest latency is 168ms. h3 sends ICMP request over link 3; 3ms. h1 sends ICMP response over link 5/4; 150+10=160ms. 160+3+switch-delay≈168ms. Throughput = 64*8/168=3047.6bps

Q34

```
mininet> h1 ping -c4 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=169 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=168 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=180 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=186 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3009ms
rtt min/avg/max/mdev = 168.200/176.191/186.332/7.604 ms
mininet>
```

No it does not change anything. As it shouldn't. H1 is configured to communicate via ip on link 5/4 no matter what and h3 on link 3.

Q35

```
#Block all ip communication between h1 and h4 for 300 seconds
sudo ovs-ofctl add-flow s1 table=0,priority=65535,hard_timeout=300,ip,nw_src=10.0.0.1,nw_dst=10.0.0.4,action=drop
sudo ovs-ofctl add-flow s2 table=0,priority=65535,hard_timeout=300,ip,nw_src=10.0.0.4,nw_dst=10.0.0.1,action=drop
```

Relevant commands are: hard_timeout which specifies after how much time a flow rule will be active. Priority 65535 is the highest priority according to the man-page which makes the firewall definitive.

Q36

```
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
^C
--- 10.0.0.4 ping statistics ---
15 packets transmitted, 0 received, 100% packet loss, time 14439ms
```

h1 → h4 is blocked for 300 seconds as expected and all packets are dropped at s1.

Q37

```
mininet> h4 ping h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
^C
--- 10.0.0.1 ping statistics ---
13 packets transmitted, 0 received, 100% packet loss, time 12297ms
```

h4 → h1 is blocked for 300 seconds as expected and all packets are dropped at s2.

Q38

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 X
h2 -> X h3 h4
h3 -> h1 h2 X
h4 -> X h2 X
*** Results: 50% dropped (6/12 received)
mininet>
```

Only h1 → h4 and h4 → h1 changed as expected. Since only ip packets with {src:10.0.0.1 AND dst:10.0.0.4} OR {src:10.0.0.4 AND dst:10.0.0.1} are dropped. Switch is treating the matches with

logical AND. If packets from src:10.0.0.1 OR src:10.0.0.4 have any other destination the firewall does not apply. This also has highest priority and thats why it’s chosen over previous flow rules.

Whole flow configuration

```
#arp packets are small and could therefore benefit low latency(link 3)
sudo ovs-ofctl add-flow s1 table=0,priority=1,action=drop
sudo ovs-ofctl add-flow s1 table=0,priority=10,arp,action=output:"1 2 3"
#if it's coming from link 3 don't bounce the request back to avoid crowding the links
sudo ovs-ofctl add-flow s1 table=0,priority=100,in_port=3,arp,action=output:"1 2"


sudo ovs-ofctl add-flow s2 table=0,priority=1,action=drop
sudo ovs-ofctl add-flow s2 table=0,priority=10,arp,action=output:"1 2 3"
#if it's coming from link 3 don't bounce the request back to avoid crowding the links
sudo ovs-ofctl add-flow s2 table=0,priority=100,in_port=3,arp,action=output:"1 2"

#This is just an exchangepoint everything is passed
sudo ovs-ofctl add-flow s3 table=0,priority=1,action=all

#High Bandwidth Connection for everything between h1 <-> h4
#output is the thing that should be restrictive
sudo ovs-ofctl add-flow s1 table=0,priority=5,ip,nw_src=10.0.0.1,action=output:5
sudo ovs-ofctl add-flow s2 table=0,priority=5,ip,nw_src=10.0.0.4,action=output:4

#input non arp traffic should have addressing sorted out
sudo ovs-ofctl add-flow s1 table=0,priority=5,ip,nw_dst=10.0.0.1,action=output:1
sudo ovs-ofctl add-flow s2 table=0,priority=5,ip,nw_dst=10.0.0.4,action=output:2

#Low latency link between h2 <-> h3
#output is the thing that should be restrictive
sudo ovs-ofctl add-flow s1 table=0,priority=5,ip,nw_src=10.0.0.2,action=output:3
sudo ovs-ofctl add-flow s2 table=0,priority=5,ip,nw_src=10.0.0.3,action=output:3

#input non arp traffic should have addressing sorted out
sudo ovs-ofctl add-flow s1 table=0,priority=5,ip,nw_dst=10.0.0.2,action=output:2
sudo ovs-ofctl add-flow s2 table=0,priority=5,ip,nw_dst=10.0.0.3,action=output:1

#Block all ip communication between h1 and h4 for 300 seconds
sudo ovs-ofctl add-flow s1 table=0,priority=65535,hard_timeout=300,ip,nw_src=10.0.0.1,nw_dst=10.0.0.4,action=drop
sudo ovs-ofctl add-flow s2 table=0,priority=65535,hard_timeout=300,ip,nw_src=10.0.0.4,nw_dst=10.0.0.1,action=drop
```