Computer Networks Lab
Group id - 2
Mukul Jain - 200001053
Nilay Ganvit - 200001053
11th April 2023

Course Project

Create a custom topology of 10 nodes (decide your own topology (except linear or bus)). Randomly select the link delay (1ms – 5ms) for all links in your topology between the switches. Set link bandwidth to 50Mb.

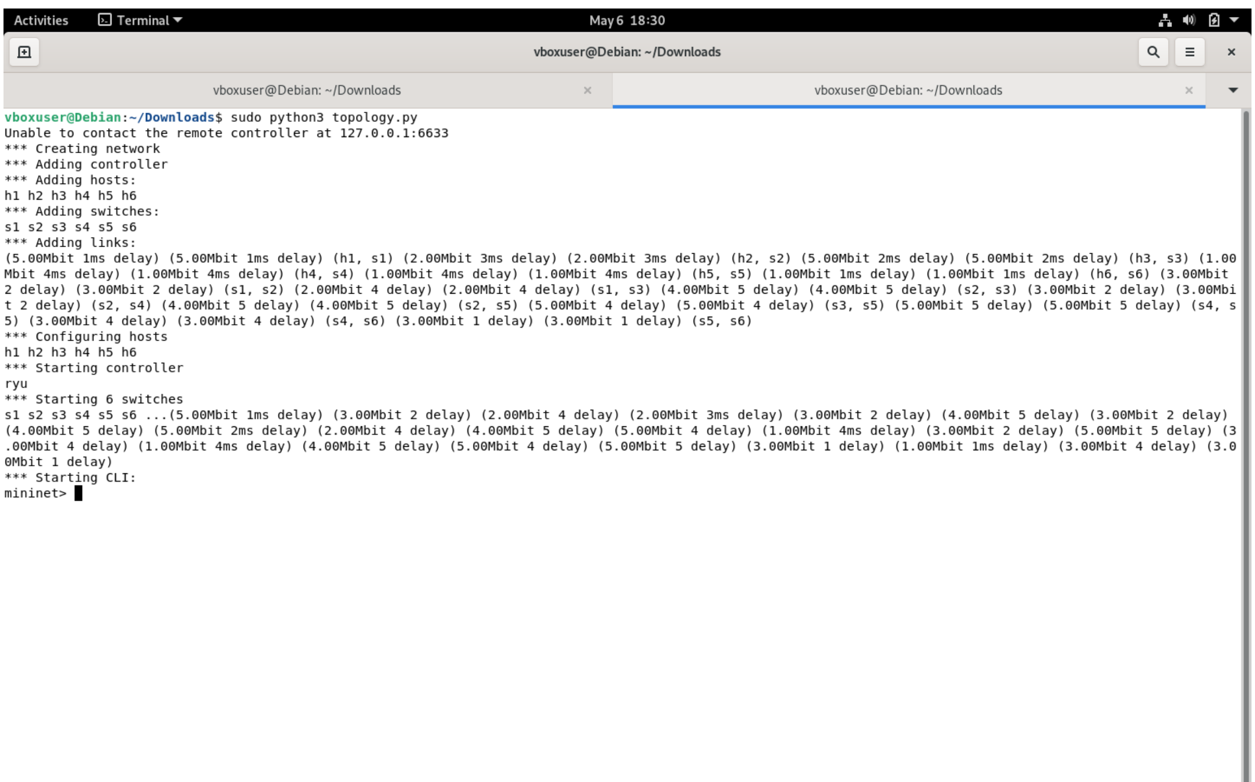
```
from random import randint

from mininet.cli import CLI
from mininet.link import TCLink
from mininet.log import setLogLevel
from mininet.net import Mininet
from mininet.node import OVSSwitch, RemoteController
from mininet.topo import Topo

file = open("values.txt", "w")


def gen_value():
    val = randint(1, 5)
    file.write(f"{val} ")
    return val


class Topology(Topo):
    def build(self):
        switches = []
        hosts = []

        for i in range(1, 7):
            switch = self.addSwitch(f"s{i}")
            switches.append(switch)
            host = self.addHost(f"h{i}")
            hosts.append(host)
```

```
            self.addLink(host, switch, bw=gen_value(),
delay=f"{gen_value()}ms")
            file.write("\n")

        file.write("0 1 ")
        self.addLink(switches[0], switches[1], bw=gen_value(),
delay=gen_value())
        file.write("\n0 2 ")
        self.addLink(switches[0], switches[2], bw=gen_value(),
delay=gen_value())
        file.write("\n1 2 ")
        self.addLink(switches[1], switches[2], bw=gen_value(),
delay=gen_value())
        file.write("\n1 3 ")
        self.addLink(switches[1], switches[3], bw=gen_value(),
delay=gen_value())
        file.write("\n1 4 ")
        self.addLink(switches[1], switches[4], bw=gen_value(),
delay=gen_value())
        file.write("\n2 4 ")
        self.addLink(switches[2], switches[4], bw=gen_value(),
delay=gen_value())
        file.write("\n3 4 ")
        self.addLink(switches[3], switches[4], bw=gen_value(),
delay=gen_value())
        file.write("\n3 5 ")
        self.addLink(switches[3], switches[5], bw=gen_value(),
delay=gen_value())
        file.write("\n4 5 ")
        self.addLink(switches[4], switches[5], bw=gen_value(),
delay=gen_value())


def createNetwork():
    "Create and test a simple network"
    topo = Topology()
    file.close()
    net = Mininet(
        topo,
        controller=RemoteController(name="ryu", port=6633),
```

```
        autoSetMacs=True,
        switch=OVSSwitch,
        link=TCLink,
    )
    net.start()
    CLI(net)
    net.stop()

if __name__ == "__main__":
    setLogLevel("info")
    createNetwork()
```

Start your topology with Mininet and connect to the RYU controller.

Sudo python3 topology.py



ryu-manager --ofp-tcp-listen-port 6633 --observe-links ./flowmanager/flowmanager.py node_discovery.py

Following tasks need to be performed:

- Write a program to discover the topology, including the switches and hosts in the network. (sample RYU controller file for node discovery is node_discovery.py)

```
import json
import time
from random import randint

from graph import Graph
from ryu.app.wsgi import ControllerBase, Response, WSGIApplication, route
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import (
    CONFIG_DISPATCHER,
    DEAD_DISPATCHER,
    MAIN_DISPATCHER,
    set_ev_cls,
)
from ryu.lib import dpid as dpid_lib
```

```python
from ryu.lib.packet import ethernet, packet
from ryu.ofproto import ofproto_v1_3
from ryu.topology import event
from ryu.topology.api import get_host, get_link, get_switch


class TopologyController(ControllerBase):
    def __init__(self, req, link, data, **config):
        super(TopologyController, self).__init__(req, link, data,
**config)
        self.topology_api_app = data["topology_api_app"]

    @route("topology", "/topology/switches", methods=["GET"])
    def list_switches(self, req, **kwargs):
        return self._switches(req, **kwargs)

    @route(
        "topology",
        "/topology/switches/{dpid}",
        methods=["GET"],
        requirements={"dpid": dpid_lib.DPID_PATTERN},
    )
    def get_switch(self, req, **kwargs):
        return self._switches(req, **kwargs)

    @route("topology", "/topology/links", methods=["GET"])
    def list_links(self, req, **kwargs):
        return self._links(req, **kwargs)

    @route(
        "topology",
        "/topology/links/{dpid}",
        methods=["GET"],
        requirements={"dpid": dpid_lib.DPID_PATTERN},
    )
    def get_links(self, req, **kwargs):
        return self._links(req, **kwargs)

    @route("topology", "/topology/hosts", methods=["GET"])
    def list_hosts(self, req, **kwargs):
```

```python
        return self._hosts(req, **kwargs)

    @route(
        "topology",
        "/topology/hosts/{dpid}",
        methods=["GET"],
        requirements={"dpid": dpid_lib.DPID_PATTERN},
    )
    def get_hosts(self, req, **kwargs):
        return self._hosts(req, **kwargs)

    def _switches(self, req, **kwargs):
        dpid = None
        if "dpid" in kwargs:
            dpid = dpid_lib.str_to_dpid(kwargs["dpid"])
        switches = get_switch(self.topology_api_app, dpid)
        body = json.dumps([switch.to_dict() for switch in switches])
        return Response(content_type="application/json", body=body)

    def _links(self, req, **kwargs):
        dpid = None
        if "dpid" in kwargs:
            dpid = dpid_lib.str_to_dpid(kwargs["dpid"])
        links = get_link(self.topology_api_app, dpid)
        body = json.dumps([link.to_dict() for link in links])
        return Response(content_type="application/json", body=body)

    def _hosts(self, req, **kwargs):
        dpid = None
        if "dpid" in kwargs:
            dpid = dpid_lib.str_to_dpid(kwargs["dpid"])
        hosts = get_host(self.topology_api_app, dpid)
        body = json.dumps([host.to_dict() for host in hosts])
        return Response(content_type="application/json", body=body)


class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {"wsgi": WSGIApplication}
```

```
    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        wsgi = kwargs["wsgi"]
        wsgi.register(TopologyController, {"topology_api_app": self})

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        match = parser.OFPMatch()
        actions = [
            parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
ofproto.OFPCML_NO_BUFFER)
        ]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions,
buffer_id=None):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(
                datapath=datapath,
                buffer_id=buffer_id,
                priority=priority,
                match=match,
                instructions=inst,
            )
        else:
            mod = parser.OFPFlowMod(
                datapath=datapath,
                priority=priority,
                match=match,
                instructions=inst,
                cookie=randint(0, 255),
```

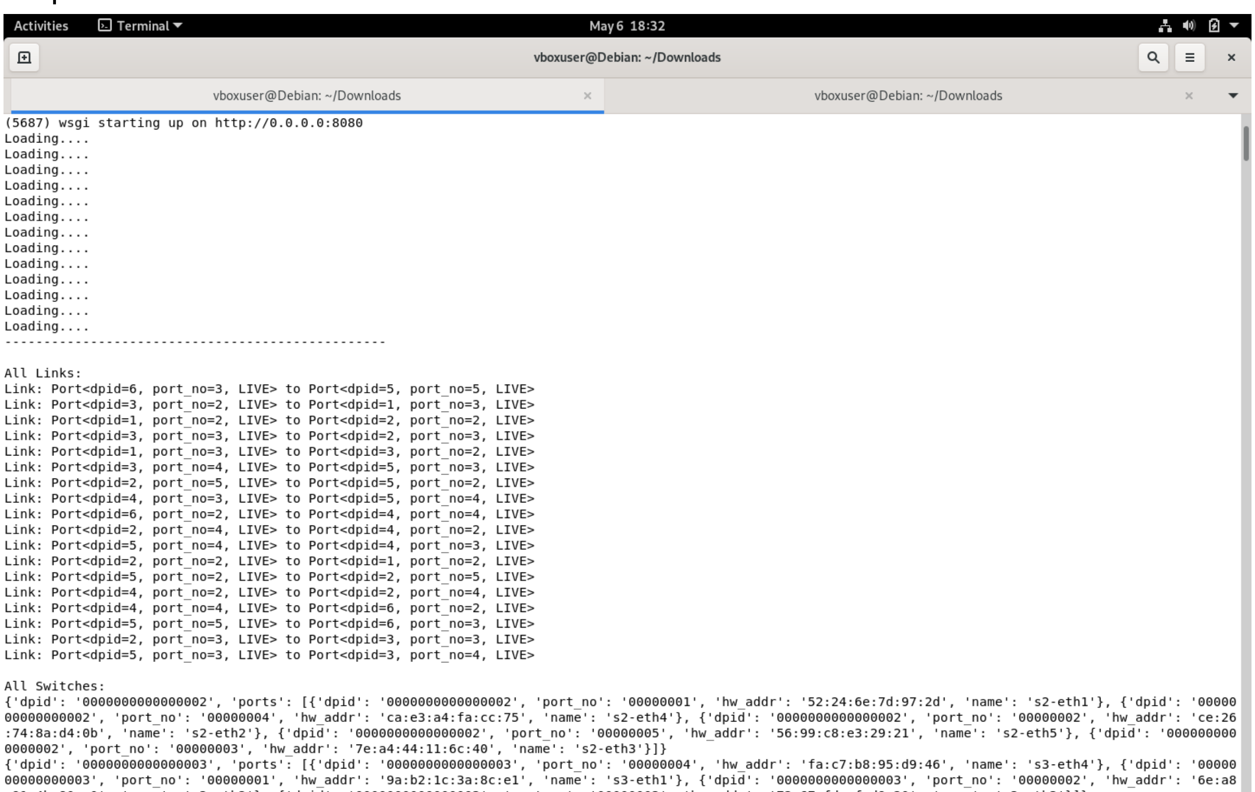
```
            )
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        if ev.msg.msg_len < ev.msg.total_len:
            self.logger.debug(
                "packet truncated: only %s of %s bytes",
                ev.msg.msg_len,
                ev.msg.total_len,
            )
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match["in_port"]

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]

        dst = eth.dst
        src = eth.src

        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})
        # self.logger.info("\tpacket in %s %s %s %s", dpid, src, dst,
in_port)
        # learn a mac address to avoid FLOOD next time.
        self.mac_to_port[dpid][src] = in_port

        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

        actions = [parser.OFPActionOutput(out_port)]

        # install a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
```

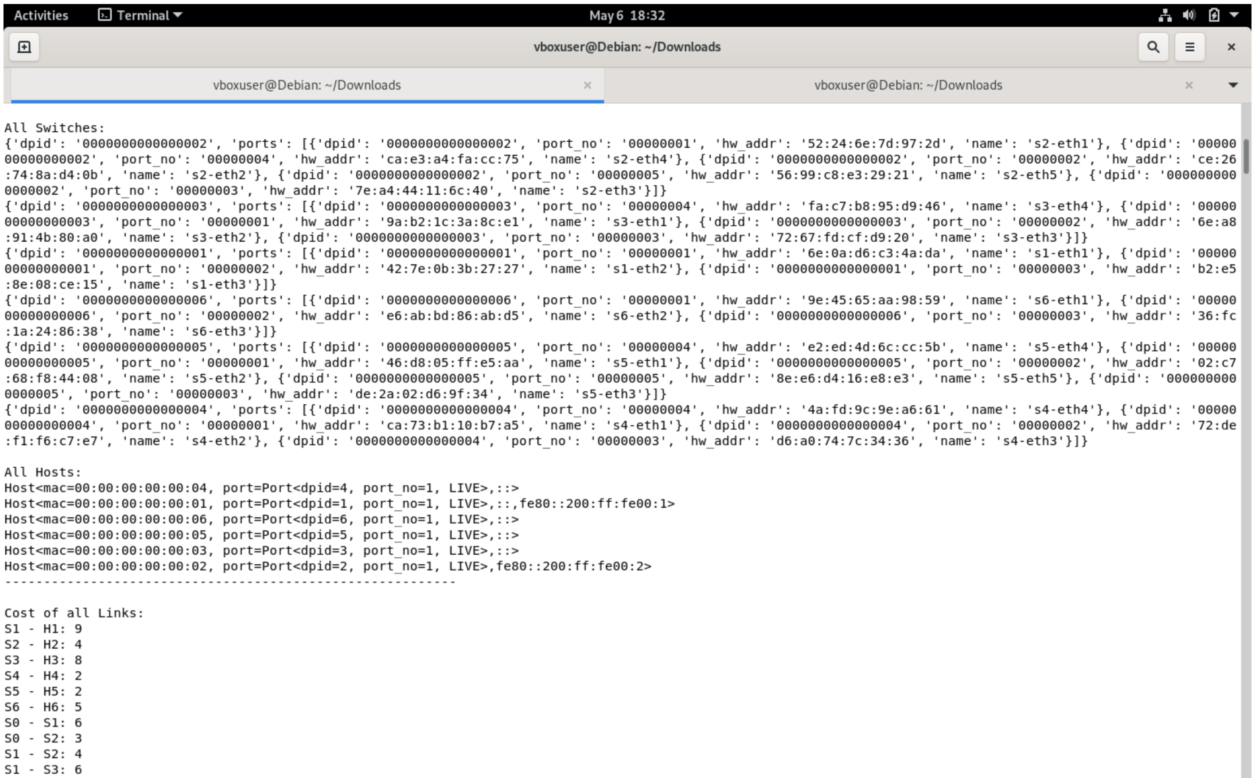
```
            # verify if we have a valid buffer_id, if yes avoid to send
both
            # flow_mod & packet_out
            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
                self.add_flow(datapath, 1, match, actions, msg.buffer_id)
                return
            else:
                self.add_flow(datapath, 1, match, actions)
        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data

        out = parser.OFPPacketOut(
            datapath=datapath,
            buffer_id=msg.buffer_id,
            in_port=in_port,
            actions=actions,
            data=data,
        )
        datapath.send_msg(out)

    def get_topology(self):
        self.switches = get_switch(self)
        self.links = get_link(self)
        self.hosts = get_host(self)

        while len(self.switches) != len(self.hosts):
            time.sleep(0.05)
            print("\rLoading....")
            self.switches = get_switch(self)
            self.links = get_link(self)
            self.hosts = get_host(self)

    """
    The event EventSwitchEnter will trigger the activation of
get_topology_data().
    """

    @set_ev_cls(event.EventSwitchEnter)
    def handler_switch_event(self, ev):
```

```
        print("New Switch", end="")
        self.get_topology()

        print("--------------------------------------------------")
        print("\nAll Links:")
        for l in self.links:
            print(l)

        print("\nAll Switches:")
        for s in self.switches:
            print(s.to_dict())

        print("\nAll Hosts:")
        for h in self.hosts:
            print(h)

        self.graph = Graph(self.switches, self.hosts, self.links)

        print("\nFlows:")
        for switch in self.switches:
            datapath = switch.dp
            ofp = datapath.ofproto
            parser = datapath.ofproto_parser

            data = switch.to_dict()
            src_id = int(data["dpid"]) - 1
            print(f"Installing flows in switch with dpid {src_id+1}")

            for host in self.hosts:
                match = parser.OFPMatch(eth_dst=host.mac,
eth_type=0x086DD)
                index = self.graph.vertices[host.mac]
                dest_id = self.graph.parents[index][src_id]
                print(src_id, dest_id)
                if dest_id < len(self.switches):
                    for link in self.links:
                        link = link.to_dict()
                        if (
                            int(link["src"]["dpid"]) == src_id + 1
                            and int(link["dst"]["dpid"]) == dest_id + 1
```

```python
                            ):
                                out_port = int(link["src"]["port_no"])
                    else:
                        out_port = int(host.port.port_no)
                    actions = [parser.OFPActionOutput(out_port)]
                    self.add_flow(datapath, 200, match, actions)

    """
    This event is fired when a switch leaves the topo. i.e. fails.
    """

    @set_ev_cls(
        event.EventSwitchLeave, [MAIN_DISPATCHER, CONFIG_DISPATCHER,
DEAD_DISPATCHER]
    )
    def handler_switch_leave(self, ev):
        self.logger.info("Not tracking Switches, switch leaved.")


app_manager.require_app("ryu.topology.switches", api_style=True)
```

Output:





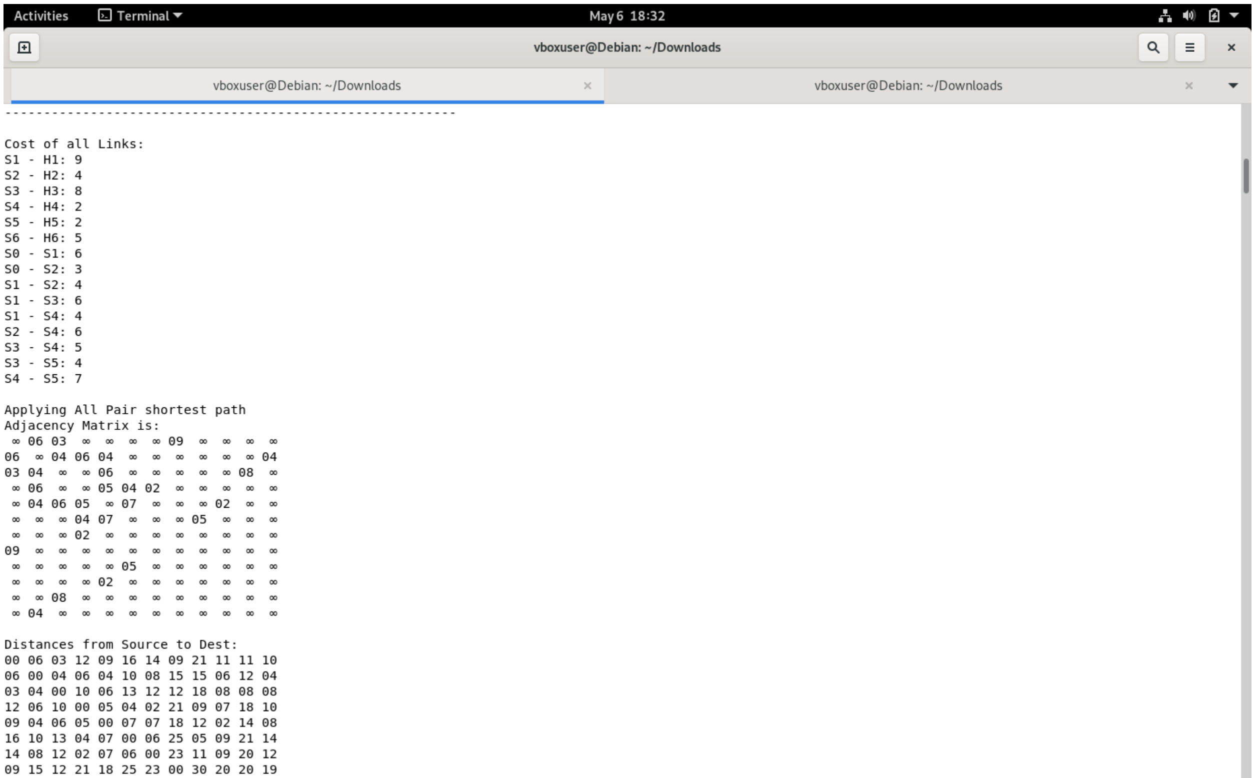

```
All Switches:
{'dpid': '0000000000000002', 'ports': [{'dpid': '0000000000000002', 'port_no': '00000001', 'hw_addr': '52:24:6e:7d:97:2d', 'name': 's2-eth1'}, {'dpid': '00000
00000000002', 'port_no': '00000004', 'hw_addr': 'ca:e3:a4:fa:cc:75', 'name': 's2-eth4'}, {'dpid': '0000000000000002', 'port_no': '00000002', 'hw_addr': 'ce:26
:74:8a:d4:0b', 'name': 's2-eth2'}, {'dpid': '0000000000000002', 'port_no': '00000005', 'hw_addr': '56:99:c8:e3:29:21', 'name': 's2-eth5'}, {'dpid': '000000000
0000002', 'port_no': '00000003', 'hw_addr': '7e:a4:44:11:6c:40', 'name': 's2-eth3'}]}
{'dpid': '0000000000000003', 'ports': [{'dpid': '0000000000000003', 'port_no': '00000004', 'hw_addr': 'fa:c7:b8:95:d9:46', 'name': 's3-eth4'}, {'dpid': '00000
00000000003', 'port_no': '00000001', 'hw_addr': '9a:b2:1c:3a:8c:e1', 'name': 's3-eth1'}, {'dpid': '0000000000000003', 'port_no': '00000002', 'hw_addr': '6e:a8
:91:4b:80:a0', 'name': 's3-eth2'}, {'dpid': '0000000000000003', 'port_no': '00000003', 'hw_addr': '72:67:fd:cf:d9:20', 'name': 's3-eth3'}]}
{'dpid': '0000000000000001', 'ports': [{'dpid': '0000000000000001', 'port_no': '00000001', 'hw_addr': '6e:0a:d6:c3:4a:da', 'name': 's1-eth1'}, {'dpid': '00000
00000000001', 'port_no': '00000002', 'hw_addr': '42:7e:0b:3b:27:27', 'name': 's1-eth2'}, {'dpid': '0000000000000001', 'port_no': '00000003', 'hw_addr': 'b2:e5
:8e:08:ce:15', 'name': 's1-eth3'}]}
{'dpid': '0000000000000006', 'ports': [{'dpid': '0000000000000006', 'port_no': '00000001', 'hw_addr': '9e:45:65:aa:98:59', 'name': 's6-eth1'}, {'dpid': '00000
00000000006', 'port_no': '00000002', 'hw_addr': 'e6:ab:bd:86:ab:d5', 'name': 's6-eth2'}, {'dpid': '0000000000000006', 'port_no': '00000003', 'hw_addr': '36:fc
:1a:24:86:38', 'name': 's6-eth3'}]}
{'dpid': '0000000000000005', 'ports': [{'dpid': '0000000000000005', 'port_no': '00000004', 'hw_addr': 'e2:ed:4d:6c:cc:5b', 'name': 's5-eth4'}, {'dpid': '00000
00000000005', 'port_no': '00000001', 'hw_addr': '46:d8:05:ff:e5:aa', 'name': 's5-eth1'}, {'dpid': '0000000000000005', 'port_no': '00000002', 'hw_addr': '02:c7
:68:f8:44:08', 'name': 's5-eth2'}, {'dpid': '0000000000000005', 'port_no': '00000005', 'hw_addr': '8e:e6:d4:16:e8:e3', 'name': 's5-eth5'}, {'dpid': '000000000
0000005', 'port_no': '00000003', 'hw_addr': 'de:2a:02:d6:9f:34', 'name': 's5-eth3'}]}
{'dpid': '0000000000000004', 'ports': [{'dpid': '0000000000000004', 'port_no': '00000004', 'hw_addr': '4a:fd:9c:9e:a6:61', 'name': 's4-eth4'}, {'dpid': '00000
00000000004', 'port_no': '00000001', 'hw_addr': 'ca:73:b1:10:b7:a5', 'name': 's4-eth1'}, {'dpid': '0000000000000004', 'port_no': '00000002', 'hw_addr': '72:de
:f1:f6:c7:e7', 'name': 's4-eth2'}, {'dpid': '0000000000000004', 'port_no': '00000003', 'hw_addr': 'd6:a0:74:7c:34:36', 'name': 's4-eth3'}]}

All Hosts:
Host<mac=00:00:00:00:00:04, port=Port<dpid=4, port_no=1, LIVE>,::>
Host<mac=00:00:00:00:00:01, port=Port<dpid=1, port_no=1, LIVE>,::,fe80::200:ff:fe00:1>
Host<mac=00:00:00:00:00:06, port=Port<dpid=6, port_no=1, LIVE>,::>
Host<mac=00:00:00:00:00:05, port=Port<dpid=5, port_no=1, LIVE>,::>
Host<mac=00:00:00:00:00:03, port=Port<dpid=3, port_no=1, LIVE>,::>
Host<mac=00:00:00:00:00:02, port=Port<dpid=2, port_no=1, LIVE>,fe80::200:ff:fe00:2>
----------------------------------------------------------

Cost of all Links:
S1 - H1: 9
S2 - H2: 4
S3 - H3: 8
S4 - H4: 2
S5 - H5: 2
S6 - H6: 5
S0 - S1: 6
S0 - S2: 3
S1 - S2: 4
S1 - S3: 6
```

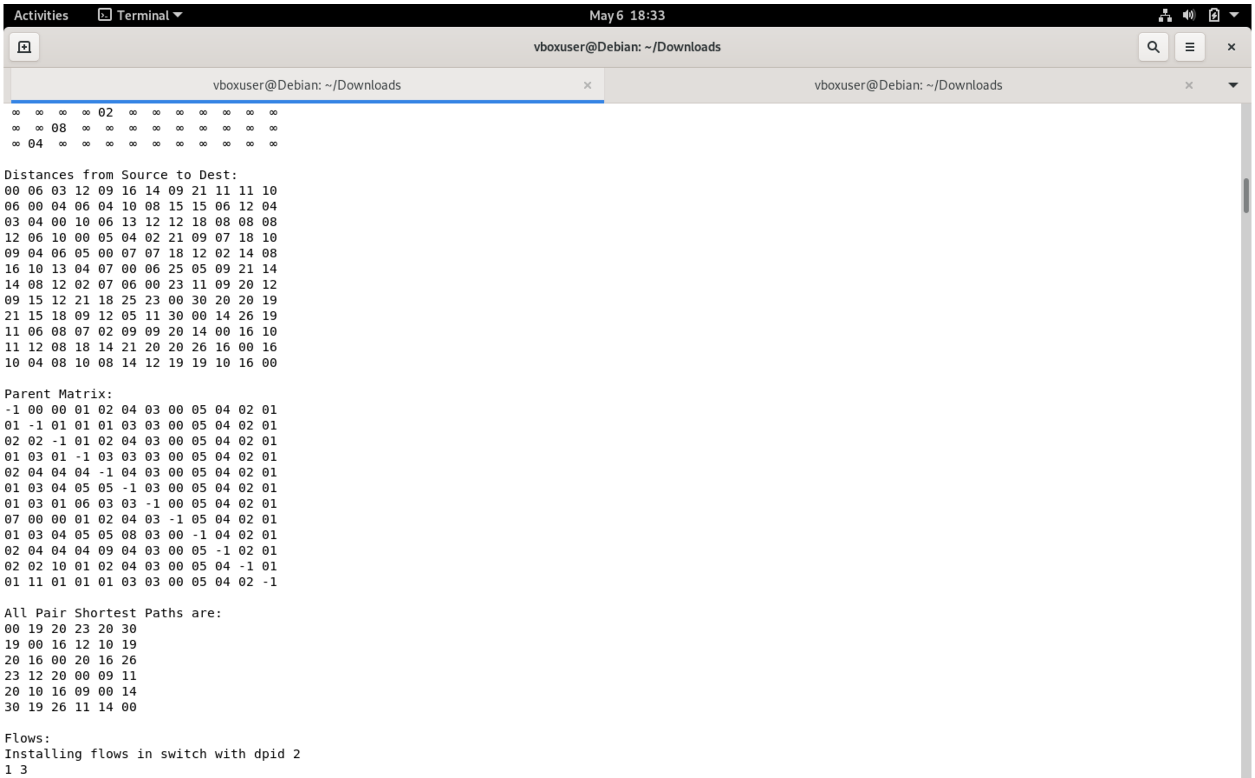
- Use the above information for computing the paths in the network for all pairs of hosts in the network.

```
with open("values.txt", "r") as file:
    values = [[int(num) for num in line.split()] for line in file]


class Graph:
    def __init__(self, switches, hosts, links):
        shift = len(switches)
        self.vertices = {}
        self.hosts = hosts
        self.switches = switches
        self.links = links

        self.nodes = len(switches) + len(hosts)
        self.edges = [
            [1e7 for column in range(self.nodes)] for row in
range(self.nodes)
        ]
        self.distances = [
```

```
            [1e7 for column in range(self.nodes)] for row in
range(self.nodes)
        ]
        self.parents = [
            [-1 for column in range(self.nodes)] for row in
range(self.nodes)
        ]
```

Output:





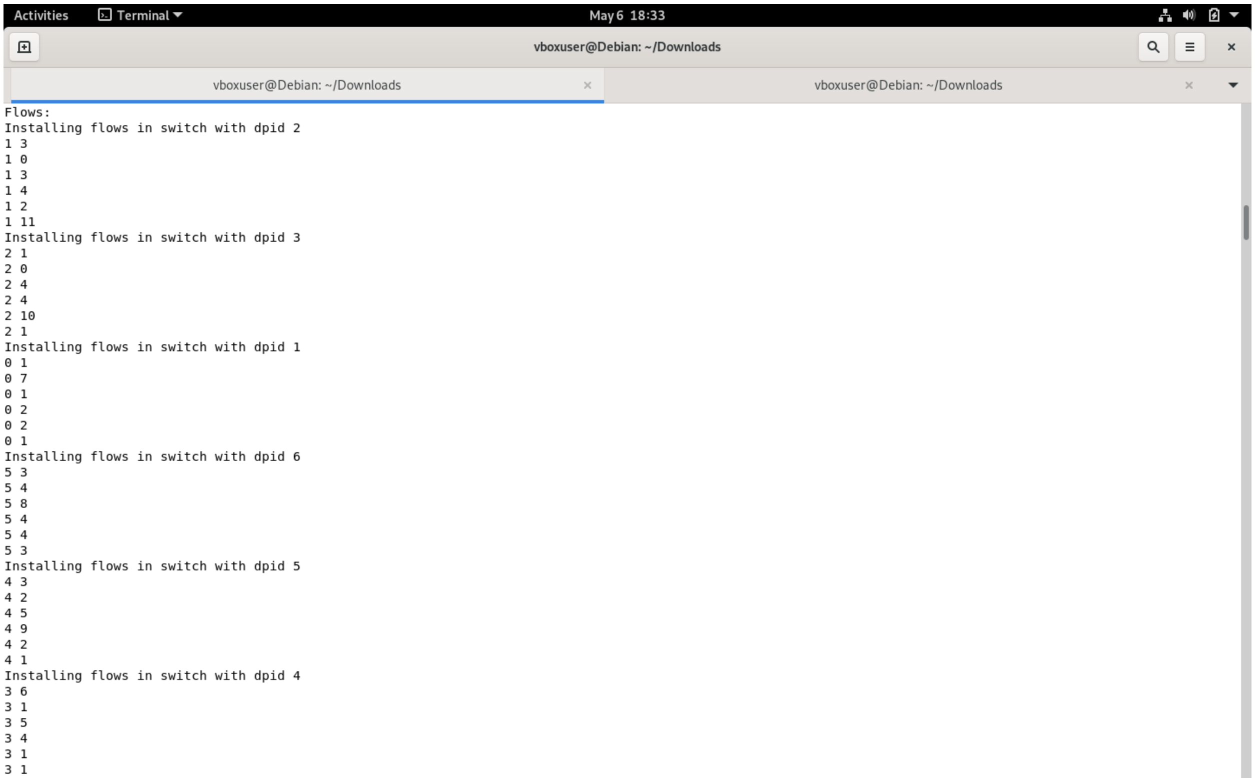

```
∞  ∞  ∞  ∞ 02  ∞  ∞  ∞  ∞  ∞  ∞  ∞
∞  ∞ 08  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞
∞ 04  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞

Distances from Source to Dest:
00 06 03 12 09 16 14 09 21 11 11 10
06 00 04 06 04 10 08 15 15 06 12 04
03 04 00 10 06 13 12 12 18 08 08 08
12 06 10 00 05 04 02 21 09 07 18 10
09 04 06 05 00 07 07 18 12 02 14 08
16 10 13 04 07 00 06 25 05 09 21 14
14 08 12 02 07 06 00 23 11 09 20 12
09 15 12 21 18 25 23 00 30 20 20 19
21 15 18 09 12 05 11 30 00 14 26 19
11 06 08 07 02 09 09 20 14 00 16 10
11 12 08 18 14 21 20 20 26 16 00 16
10 04 08 10 08 14 12 19 19 10 16 00

Parent Matrix:
-1 00 00 01 02 04 03 00 05 04 02 01
01 -1 01 01 01 03 03 00 05 04 02 01
02 02 -1 01 02 04 03 00 05 04 02 01
01 03 01 -1 03 03 03 00 05 04 02 01
02 04 04 04 -1 04 03 00 05 04 02 01
01 03 04 05 05 -1 03 00 05 04 02 01
01 03 01 06 03 03 -1 00 05 04 02 01
07 00 00 01 02 04 03 -1 05 04 02 01
01 03 04 05 05 08 03 00 -1 04 02 01
02 04 04 04 09 04 03 00 05 -1 02 01
02 02 10 01 02 04 03 00 05 04 -1 01
01 11 01 01 01 03 03 00 05 04 02 -1

All Pair Shortest Paths are:
00 19 20 23 20 30
19 00 16 12 10 19
20 16 00 20 16 26
23 12 20 00 09 11
20 10 16 09 00 14
30 19 26 11 14 00

Flows:
Installing flows in switch with dpid 2
1 3
```

Identify the switches where configuration need to be updated. Provide details of the configuration to be written over each intermediate switch on the path

```python
        for i in switches:
            switch = i.to_dict()
            value = int(switch["dpid"]) - 1
            self.vertices[i] = value
        # Switches are indexed by DPID

        for i in hosts:
            self.vertices[i.mac] = shift
            shift += 1
        # Hosts are added at the end in random order

        self.host_map = {}
        for i in hosts:
            self.host_map["H" + i.mac[-1]] = i.mac

        self.main()

    def print_matrix(self, matrix, size):
        for i in range(size):
```

```python
            for j in range(size):
                if matrix[i][j] == 1e7:
                    print(" ∞", end=" ")
                else:
                    print(f"{int(matrix[i][j]):02}", end=" ")
            print()

    def cost(self, delay, bandwidth):
        return delay - bandwidth + 5

    def create_graph(self):
        for i in self.hosts:
            dpid = i.port.dpid - 1
            delay, bandwidth = values[dpid]
            self.edges[self.vertices[i.mac]][dpid] = self.cost(delay, bandwidth)
            self.edges[dpid][self.vertices[i.mac]] = self.cost(delay, bandwidth)

        shift = len(self.switches)
        for i in range(shift, len(values)):
            delay, bandwidth = values[i][2:]
            self.edges[values[i][0]][values[i][1]] = self.cost(delay, bandwidth)
            self.edges[values[i][1]][values[i][0]] = self.cost(delay, bandwidth)

    def min_distance(self, distances, shortest_path):
        min = 1e7
        for v in range(self.nodes):
            if distances[v] < min and shortest_path[v] == False:
                min = distances[v]
                min_index = v

        return min_index

    def dijkstra(self, src):
        self.distances[src][src] = 0
        shortest_path = [False] * self.nodes
```

```
        for _ in range(self.nodes):
            u = self.min_distance(self.distances[src], shortest_path)
            shortest_path[u] = True
            for v in range(self.nodes):
                if (
                    self.edges[u][v] > 0
                    and shortest_path[v] == False
                    and self.distances[src][v]
                    > self.distances[src][u] + self.edges[u][v]
                ):
                    self.parents[src][v] = u
                    self.distances[src][v] = self.distances[src][u] +
self.edges[u][v]

    def all_pair_shortest_paths(self):
        min_cost = [
            [0 for column in range(len(self.hosts))] for row in
range(len(self.hosts))
        ]
        for i in self.hosts:
            for j in self.hosts:
                src = self.vertices[i.mac]
                dst = self.vertices[j.mac]
                min_cost[int(i.mac[-1]) - 1][int(j.mac[-1]) - 1] =
self.distances[src][
                    dst
                ]
        self.print_matrix(min_cost, len(self.hosts))

    def cost_all_links(self):
        for i in range(len(self.switches)):
            print(f"S{i+1} - H{i+1}: {self.cost(values[i][0],
values[i][1])}")
        for j in range(len(self.switches), len(values)):
            print(
                f"S{values[j][0]} - S{values[j][1]}:
{self.cost(values[j][2], values[j][3])}"
            )
```
Output:



```
Flows:
Installing flows in switch with dpid 2
1 3
1 0
1 3
1 4
1 2
1 11
Installing flows in switch with dpid 3
2 1
2 0
2 4
2 4
2 10
2 1
Installing flows in switch with dpid 1
0 1
0 7
0 1
0 2
0 2
0 1
Installing flows in switch with dpid 6
5 3
5 4
5 8
5 4
5 4
5 3
Installing flows in switch with dpid 5
4 3
4 2
4 5
4 9
4 2
4 1
Installing flows in switch with dpid 4
3 6
3 1
3 5
3 4
3 1
3 1
```

Take user input to request the connection by asking for following:

1. Source and destination host
2. Service requests are either IPv4 or MAC based
3. Bandwidth of the service (1-5Mb)

```
    def compute_path(self):
        print("Enter Source Host:", end=" ")
        src_host = input()
        print("Enter Destination Host:", end=" ")
        dest_host = input()

        print(f"{src_host} - {dest_host}:", end=" ")
        src_index = self.vertices[self.host_map[src_host]]
        dest_index = self.vertices[self.host_map[dest_host]]
        self.path(src_index, dest_index, src_host, dest_host)
        print()

    def compute_all_paths(self):
        print("Enter Source Host:", end=" ")
        src_host = input()
        for i in self.hosts:
```

```
            dest_host = "H" + i.mac[-1]
            print(f"{src_host} - {dest_host}:", end=" ")
            src_index = self.vertices[self.host_map[src_host]]
            dest_index = self.vertices[i.mac]
            self.path(src_index, dest_index, src_host, dest_host)
            print()

    def path(self, src_index, dest_index, src_host, dest_host):
        if self.parents[src_index][dest_index] == -1:
            print(src_host, end=" ")
            return
        self.path(src_index, self.parents[src_index][dest_index],
src_host, dest_host)
        if dest_index >= len(self.hosts):
            print(f"-> {dest_host}", end=" ")
        else:
            print(f"-> S{dest_index+1}", end=" ")

    def main(self):

print("-----------------------------------------------------------")
```

Include the already configured services in path computation.

1. You need to keep track of the available bandwidth of the links (how much utilized, how much unutilized)
2. Based on the delay and available bandwidth information compute the new cost for the link. Cost will be updated with changes in the available bandwidth.
3. Run step 4.

```
        print("\nCost of all Links:")
        self.create_graph()
        self.cost_all_links()

        print("\nApplying All Pair shortest path")
        for i in range(self.nodes):
            self.dijkstra(i)

        print("Adjacency Matrix is:")
        self.print_matrix(self.edges, self.nodes)
        print("\nDistances from Source to Dest:")
```

```
        self.print_matrix(self.distances, self.nodes)
        print("\nParent Matrix:")
        self.print_matrix(self.parents, self.nodes)

        print("\nAll Pair Shortest Paths are:")
        self.all_pair_shortest_paths()

        # print("\nCompute Path between Source and Destination Hosts:")
        self.compute_path()

        # print("\nCompute Path from Source to All Hosts:")
        self.compute_all_paths()


print("\n-----------------------------------------------------------")
```
Output:

Activities   Terminal   May 6 18:33

vboxuser@Debian: ~/Downloads

vboxuser@Debian: ~/Downloads | vboxuser@Debian: ~/Downloads

```
3 4
3 1
3 1
New Switch-----------------------------------------------

All Links:
Link: Port<dpid=6, port_no=3, LIVE> to Port<dpid=5, port_no=5, LIVE>
Link: Port<dpid=3, port_no=2, LIVE> to Port<dpid=1, port_no=3, LIVE>
Link: Port<dpid=1, port_no=2, LIVE> to Port<dpid=2, port_no=2, LIVE>
Link: Port<dpid=3, port_no=3, LIVE> to Port<dpid=2, port_no=3, LIVE>
Link: Port<dpid=1, port_no=3, LIVE> to Port<dpid=3, port_no=2, LIVE>
Link: Port<dpid=3, port_no=4, LIVE> to Port<dpid=5, port_no=3, LIVE>
Link: Port<dpid=2, port_no=5, LIVE> to Port<dpid=5, port_no=2, LIVE>
Link: Port<dpid=4, port_no=3, LIVE> to Port<dpid=5, port_no=4, LIVE>
Link: Port<dpid=6, port_no=2, LIVE> to Port<dpid=4, port_no=4, LIVE>
Link: Port<dpid=2, port_no=4, LIVE> to Port<dpid=4, port_no=2, LIVE>
Link: Port<dpid=5, port_no=4, LIVE> to Port<dpid=4, port_no=3, LIVE>
Link: Port<dpid=2, port_no=2, LIVE> to Port<dpid=1, port_no=2, LIVE>
Link: Port<dpid=5, port_no=2, LIVE> to Port<dpid=2, port_no=5, LIVE>
Link: Port<dpid=4, port_no=2, LIVE> to Port<dpid=2, port_no=4, LIVE>
Link: Port<dpid=4, port_no=4, LIVE> to Port<dpid=6, port_no=2, LIVE>
Link: Port<dpid=5, port_no=5, LIVE> to Port<dpid=6, port_no=3, LIVE>
Link: Port<dpid=2, port_no=3, LIVE> to Port<dpid=3, port_no=3, LIVE>
Link: Port<dpid=5, port_no=3, LIVE> to Port<dpid=3, port_no=4, LIVE>

All Switches:
```


{'dpid': '0000000000000002', 'ports': [{'dpid': '0000000000000002', 'port_no': '00000001', 'hw_addr': '52:24:6e:7d:97:2d', 'name': 's2-eth1'}, {'dpid': '0000000000000002', 'port_no': '00000004', 'hw_addr': 'ca:e3:a4:fa:cc:75', 'name': 's2-eth4'}, {'dpid': '0000000000000002', 'port_no': '00000002', 'hw_addr': 'ce:26:74:8a:d4:0b', 'name': 's2-eth2'}, {'dpid': '0000000000000002', 'port_no': '00000005', 'hw_addr': '56:99:c8:e3:29:21', 'name': 's2-eth5'}, {'dpid': '0000000000000002', 'port_no': '00000003', 'hw_addr': '7e:a4:44:11:6c:40', 'name': 's2-eth3'}]}
{'dpid': '0000000000000003', 'ports': [{'dpid': '0000000000000003', 'port_no': '00000004', 'hw_addr': 'fa:c7:b8:95:d9:46', 'name': 's3-eth4'}, {'dpid': '0000000000000003', 'port_no': '00000001', 'hw_addr': '9a:b2:1c:3a:8c:e1', 'name': 's3-eth1'}, {'dpid': '0000000000000003', 'port_no': '00000002', 'hw_addr': '6e:a8:91:4b:80:a0', 'name': 's3-eth2'}, {'dpid': '0000000000000003', 'port_no': '00000003', 'hw_addr': '72:67:fd:cf:d9:20', 'name': 's3-eth3'}]}
{'dpid': '0000000000000001', 'ports': [{'dpid': '0000000000000001', 'port_no': '00000001', 'hw_addr': '6e:0a:d6:c3:4a:da', 'name': 's1-eth1'}, {'dpid': '0000000000000001', 'port_no': '00000002', 'hw_addr': '42:7e:0b:3b:27:27', 'name': 's1-eth2'}, {'dpid': '0000000000000001', 'port_no': '00000003', 'hw_addr': 'b2:e5:8e:08:ce:15', 'name': 's1-eth3'}]}
{'dpid': '0000000000000006', 'ports': [{'dpid': '0000000000000006', 'port_no': '00000001', 'hw_addr': '9e:45:65:aa:98:59', 'name': 's6-eth1'}, {'dpid': '0000000000000006', 'port_no': '00000002', 'hw_addr': 'e6:ab:bd:86:ab:d5', 'name': 's6-eth2'}, {'dpid': '0000000000000006', 'port_no': '00000003', 'hw_addr': '36:fc:1a:24:86:38', 'name': 's6-eth3'}]}
{'dpid': '0000000000000005', 'ports': [{'dpid': '0000000000000005', 'port_no': '00000004', 'hw_addr': 'e2:ed:4d:6c:cc:5b', 'name': 's5-eth4'}, {'dpid': '0000000000000005', 'port_no': '00000001', 'hw_addr': '46:d8:05:ff:e5:aa', 'name': 's5-eth1'}, {'dpid': '0000000000000005', 'port_no': '00000002', 'hw_addr': '02:c7:68:f8:44:08', 'name': 's5-eth2'}, {'dpid': '0000000000000005', 'port_no': '00000005', 'hw_addr': '8e:e6:d4:16:e8:e3', 'name': 's5-eth5'}, {'dpid': '0000000000000005', 'port_no': '00000003', 'hw_addr': 'de:2a:02:d6:9f:34', 'name': 's5-eth3'}]}

Activities   Terminal   May 6 18:33

vboxuser@Debian: ~/Downloads

vboxuser@Debian: ~/Downloads | vboxuser@Debian: ~/Downloads

```
Cost of all Links:
S1 - H1: 9
S2 - H2: 4
S3 - H3: 8
S4 - H4: 2
S5 - H5: 2
S6 - H6: 5
S0 - S1: 6
S0 - S2: 3
S1 - S2: 4
S1 - S3: 6
S1 - S4: 4
S2 - S4: 6
S3 - S4: 5
S3 - S5: 4
S4 - S5: 7

Applying All Pair shortest path
Adjacency Matrix is:
 ∞ 06 03  ∞  ∞  ∞  ∞ 09  ∞  ∞  ∞  ∞
06  ∞ 04 06 04  ∞  ∞  ∞  ∞  ∞  ∞ 04
03 04  ∞  ∞ 06  ∞  ∞  ∞  ∞  ∞ 08  ∞
 ∞ 06  ∞  ∞ 05 04 02  ∞  ∞  ∞  ∞  ∞
 ∞ 04 06 05  ∞ 07  ∞  ∞  ∞ 02  ∞  ∞
 ∞  ∞  ∞ 04 07  ∞  ∞  ∞ 05  ∞  ∞  ∞
 ∞  ∞  ∞ 02  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞
09  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞
 ∞  ∞  ∞  ∞  ∞ 05  ∞  ∞  ∞  ∞  ∞  ∞
 ∞  ∞  ∞  ∞ 02  ∞  ∞  ∞  ∞  ∞  ∞  ∞
 ∞  ∞ 08  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞
 ∞ 04  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞

Distances from Source to Dest:
00 06 03 12 09 16 14 09 21 11 11 10
06 00 04 06 04 10 08 15 15 06 12 04
03 04 00 10 06 13 12 12 18 08 08 08
12 06 10 00 05 04 02 21 09 07 18 10
09 04 06 05 00 07 07 18 12 02 14 08
16 10 13 04 07 00 06 25 05 09 21 14
14 08 12 02 07 06 00 23 11 09 20 12
09 15 12 21 18 25 23 00 30 20 20 19
21 15 18 09 12 05 11 30 00 14 26 19
```

Activities Terminal May 6 18:34

vboxuser@Debian: ~/Downloads

vboxuser@Debian: ~/Downloads vboxuser@Debian: ~/Downloads

```
 ∞  ∞ 08  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞
 ∞ 04  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞  ∞

Distances from Source to Dest:
00 06 03 12 09 16 14 09 21 11 11 10
06 00 04 06 04 10 08 15 15 06 12 04
03 04 00 10 06 13 12 12 18 08 08 08
12 06 10 00 05 04 02 21 09 07 18 10
09 04 06 05 00 07 07 18 12 02 14 08
16 10 13 04 07 00 06 25 05 09 21 14
14 08 12 02 07 06 00 23 11 09 20 12
09 15 12 21 18 25 23 00 30 20 20 19
21 15 18 09 12 05 11 30 00 14 26 19
11 06 08 07 02 09 09 20 14 00 16 10
11 12 08 18 14 21 20 20 26 16 00 16
10 04 08 10 08 14 12 19 19 10 16 00

Parent Matrix:
-1 00 00 01 02 04 03 00 05 04 02 01
01 -1 01 01 01 03 03 00 05 04 02 01
02 02 -1 01 02 04 03 00 05 04 02 01
01 03 01 -1 03 03 03 00 05 04 02 01
02 04 04 04 -1 04 03 00 05 04 02 01
01 03 04 05 05 -1 03 00 05 04 02 01
01 03 01 06 03 03 -1 00 05 04 02 01
07 00 00 01 02 04 03 -1 05 04 02 01
01 03 04 05 05 08 03 00 -1 04 02 01
02 04 04 04 09 04 03 00 05 -1 02 01
02 02 10 01 02 04 03 00 05 04 -1 01
01 11 01 01 01 03 03 00 05 04 02 -1

All Pair Shortest Paths are:
00 19 20 23 20 30
19 00 16 12 10 19
20 16 00 20 16 26
23 12 20 00 09 11
20 10 16 09 00 14
30 19 26 11 14 00

Flows:
Installing flows in switch with dpid 2
1 3
1 0
```

Activities Terminal May 6 18:34

vboxuser@Debian: ~/Downloads

vboxuser@Debian: ~/Downloads | vboxuser@Debian: ~/Downloads

```
23 12 20 00 09 11
20 10 16 09 00 14
30 19 26 11 14 00

Flows:
Installing flows in switch with dpid 2
1 3
1 0
1 3
1 4
1 2
1 11
Installing flows in switch with dpid 3
2 1
2 0
2 4
2 4
2 10
2 1
Installing flows in switch with dpid 1
0 1
0 7
0 1
0 2
0 2
0 1
Installing flows in switch with dpid 6
5 3
5 4
5 8
5 4
5 4
5 3
Installing flows in switch with dpid 5
4 3
4 2
4 5
4 9
4 2
4 1
Installing flows in switch with dpid 4
3 6
3 1
```

Activities Terminal May 6 18:34

vboxuser@Debian: ~/Downloads

vboxuser@Debian: ~/Downloads | vboxuser@Debian: ~/Downloads

```
Installing flows in switch with dpid 2
1 3
1 0
1 3
1 4
1 2
1 11
Installing flows in switch with dpid 3
2 1
2 0
2 4
2 4
2 10
2 1
Installing flows in switch with dpid 1
0 1
0 7
0 1
0 2
0 2
0 1
Installing flows in switch with dpid 6
5 3
5 4
5 8
5 4
5 4
5 3
Installing flows in switch with dpid 5
4 3
4 2
4 5
4 9
4 2
4 1
Installing flows in switch with dpid 4
3 6
3 1
3 5
3 4
3 1
3 1
```

```
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(50.00Mbit 5ms delay) (50.00Mbit 5ms delay) (h1, s1) (50.00Mbit 4ms delay) (50.00Mbit 4ms delay) (h2, s2) (50.00Mbit 3ms delay) (50.00Mbit 3ms delay) (h3, s3) (50.00Mbit 1ms delay)
(50.00Mbit 1ms delay) (h4, s4) (50.00Mbit 4ms delay) (50.00Mbit 4ms delay) (h5, s5) (50.00Mbit 5ms delay) (50.00Mbit 5ms delay) (h6, s6) (50.00Mbit 5ms delay) (50.00Mbit 5ms delay)
(h7, s7) (50.00Mbit 1ms delay) (50.00Mbit 1ms delay) (h8, s8) (50.00Mbit 1ms delay) (50.00Mbit 1ms delay) (h9, s9) (50.00Mbit 4ms delay) (50.00Mbit 4ms delay) (h10, s10) (50.00Mbit
2ms delay) (50.00Mbit 2ms delay) (s1, s2) (50.00Mbit 3ms delay) (50.00Mbit 3ms delay) (s1, s3) (50.00Mbit 5ms delay) (50.00Mbit 5ms delay) (s1, s4) (50.00Mbit 1ms delay) (50.00Mbit
1ms delay) (s2, s5) (50.00Mbit 4ms delay) (50.00Mbit 4ms delay) (s2, s6) (50.00Mbit 2ms delay) (50.00Mbit 2ms delay) (s3, s5) (50.00Mbit 3ms delay) (50.00Mbit 3ms delay) (s3, s6) (5
0.00Mbit 5ms delay) (50.00Mbit 5ms delay) (s4, s6) (50.00Mbit 3ms delay) (50.00Mbit 3ms delay) (s4, s7) (50.00Mbit 1ms delay) (50.00Mbit 1ms delay) (s5, s8) (50.00Mbit 2ms delay) (5
0.00Mbit 2ms delay) (s5, s9) (50.00Mbit 5ms delay) (50.00Mbit 5ms delay) (s6, s9) (50.00Mbit 5ms delay) (50.00Mbit 5ms delay) (s6, s10) (50.00Mbit 3ms delay) (50.00Mbit 3ms delay) (
s7, s10) (50.00Mbit 2ms delay) (50.00Mbit 2ms delay) (s8, s10) (50.00Mbit 3ms delay) (50.00Mbit 3ms delay) (s9, s10)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10
*** Starting controller
c0
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...(50.00Mbit 5ms delay) (50.00Mbit 2ms delay) (50.00Mbit 3ms delay) (50.00Mbit 5ms delay) (50.00Mbit 4ms delay) (50.00Mbit 2ms delay) (50.00Mbit 1ms
delay) (50.00Mbit 4ms delay) (50.00Mbit 3ms delay) (50.00Mbit 3ms delay) (50.00Mbit 2ms delay) (50.00Mbit 3ms delay) (50.00Mbit 1ms delay) (50.00Mbit 5ms delay) (50.00Mbit 5ms delay
) (50.00Mbit 3ms delay) (50.00Mbit 4ms delay) (50.00Mbit 1ms delay) (50.00Mbit 2ms delay) (50.00Mbit 1ms delay) (50.00Mbit 2ms delay) (50.00Mbit 5ms delay) (50.00Mbit 4ms delay) (50
.00Mbit 3ms delay) (50.00Mbit 5ms delay) (50.00Mbit 5ms delay) (50.00Mbit 5ms delay) (50.00Mbit 5ms delay) (50.00Mbit 3ms delay) (50.00Mbit 3ms delay) (50.00Mbit 1ms delay) (50.00Mb
it 1ms delay) (50.00Mbit 2ms delay) (50.00Mbit 1ms delay) (50.00Mbit 2ms delay) (50.00Mbit 5ms delay) (50.00Mbit 3ms delay) (50.00Mbit 4ms delay) (50.00Mbit 5ms delay) (50.00Mbit 3m
s delay) (50.00Mbit 2ms delay) (50.00Mbit 3ms delay)
Waiting for switches to connect to controller...
*** Waiting for switches to connect
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
Enter source host: h1
Enter destination host: h4
Enter service request (IPv4 or MAC): ipv4
Enter bandwidth (1-5Mb): 3
Press Enter to establish connection...
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['2.92 Mbits/sec', '2.91 Mbits/sec']
*** Stopping 1 controllers
c0
*** Stopping 26 links
..........................
*** Stopping 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Stopping 10 hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10
*** Done
nil@y ~/mininet/custom
```

Input: Values.txt:

Activities | Text Editor | May 6 18:42

values.txt [Read-Only]
~/Downloads

```
4 4
2 5
1 4
5 3
1 1
5 1
0 1 4 4
0 2 4 5
1 2 3 2
1 3 3 3
1 4 3 3
2 4 1 1
3 4 1 4
3 5 5 1
4 5 4 1
```

Plain Text | Tab Width: 8 | Ln 1, Col 1 | INS