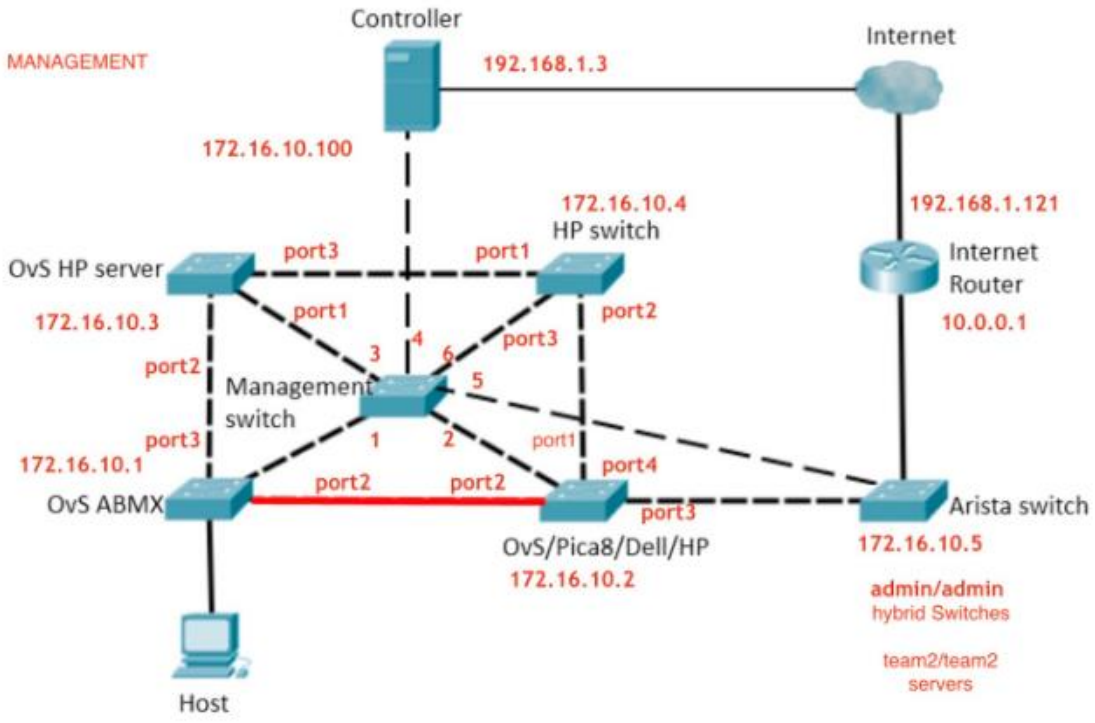
LAB 10 – SDN HARDWARE AND APPs

Team2 – Kiran, Willy, Srivaishnavi, Prathamesh

Objective – The lab involves working with hybrid switches and vendor-neutral servers in setting up the Software-Defined Network that controls and automates the network configuration, monitoring and the management. The objective involves setting up and installing an SDN controller on an x86 platform, configuring Open vSwitch on bare metal boxes, understanding the CLI of prominent vendors that support OpenFlow, basic routing, switching, and network engineering design and troubleshooting OpenFlow and network design and programming applications via API and OpenFlow

Network Topology/Design



Physical Hardware Spec:

- OvS Dell AMBX Server, OvS HP server, and OvS Dell Server
- RYU Controller
- Hybrid Aritsa, HP switches
- Cisco Router working as a Internet Router
- Cisco L2 switch working as a Management Switch

Execution :

All the physical hardware is connected and configured with the 172.16.0.0/24 as the management ip address space. And management VLAN 100 is used to redirect the relevant traffic.

1.Configure Internet Router

1. Connect 1 port of your Cisco router to the switch labelled 'NGN – Lab 10 – Internet Switch' and configure it as a DHCP client. It should receive an IP in the 192.168.1.0/24.

```
Team2Router#sh run | sec dhcp
ip dhcp pool SDNPOOL
 network 10.0.0.0 255.255.255.0
 domain-name cisco.com
 dns-server 8.8.8.8
 default-router 10.0.0.1
 ip address dhcp
```

2. Configure the other port on the router to act as a DHCP server, it will be used for one of the applications, and for inter-VLAN routing.

```
!
interface GigabitEthernet0/0
 ip address dhcp
 ip nat outside
 ip virtual-reassembly in
 duplex auto
 speed auto
 media-type rj45
!
interface GigabitEthernet0/1
 ip address 10.0.0.1 255.255.255.0
 ip nat inside
 ip virtual-reassembly in
 duplex auto
 speed auto
 media-type rj45
!
```

```
!
ip nat inside source list 1 interface GigabitEthernet0/0 overload
!
access-list 1 permit 10.0.0.0 0.0.0.255
!
!
!
!
```

2.Configure OpenFlow Switches (ABMX/HP/Pica8/Dell servers)

1.Load Ubuntu of your choice and Open vSwitch on the hardware (if not already installed).

2.Configure interfaces in proper IP subnet / VLAN for your design:

a.Apply ports in OpenFlow network to use OpenFlow.

b.Configure static route for non-OpenFlow / out-of-band network.

Ovs AMBX Server

```
team2@team2:~$ sudo ovs-vsctl show
cfb9658e-ae4b-490e-9a2c-3b953bc46bd3
    Bridge mybridge
        Controller "tcp:172.16.10.100:6633"
        Port eno2
            Interface eno2
        Port eno3
            Interface eno3
        Port mybridge
            Interface mybridge
                type: internal
    ovs_version: "2.13.8"
team2@team2:~$
team2@team2:~$ ip a show eno1
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 14:18:77:66:bd:0a brd ff:ff:ff:ff:ff:ff
    inet 22.22.22.6/24 brd 22.22.22.255 scope global eno1
       valid_lft forever preferred_lft forever
    inet 172.16.10.1/24 scope global eno1
       valid_lft forever preferred_lft forever
    inet6 fe80::1618:77ff:fe66:bd0a/64 scope link
       valid_lft forever preferred_lft forever
team2@team2:~$
```

OvS Dell

```
team2@team2:~$
team2@team2:~$ sudo   ovs-vsctl show
650e8e93-f5a6-4be5-a3d4-32624bdd6132
    Manager "ptcp:6640"
    Bridge mybridge
        Controller "tcp:172.16.10.100:6633"
        Port eno3
            Interface eno3
        Port mybridge
            Interface mybridge
                type: internal
        Port eno2
            Interface eno2
        Port eno4
            Interface eno4
    ovs_version: "2.13.5"
team2@team2:~$
team2@team2:~$ ip a show eno1
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 18:66:da:8a:50:08 brd ff:ff:ff:ff:ff:ff
    inet 172.16.10.2/24 scope global eno1
       valid_lft forever preferred_lft forever
    inet6 fe80::1a66:daff:fe8a:5008/64 scope link
       valid_lft forever preferred_lft forever
team2@team2:~$
```

OvS HP

```
team2@team2-PowerEdge-R430:~$ sudo ovs-vsctl show
[sudo] password for team2:
8571a19c-1d91-4781-9829-576f3230e9ff
    Bridge mybridge
        Controller "tcp:172.16.10.100:6633"
        Port eno3
            Interface eno3
        Port mybridge
            Interface mybridge
                type: internal
        Port eno2
            Interface eno2
    ovs_version: "2.17.7"
team2@team2-PowerEdge-R430:~$
team2@team2-PowerEdge-R430:~$
team2@team2-PowerEdge-R430:~$ ip a show eno1
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 14:18:77:3b:9b:80 brd ff:ff:ff:ff:ff:ff
    altname enp2s0f0
    inet 172.16.10.3/24 scope global eno1
       valid_lft forever preferred_lft forever
team2@team2-PowerEdge-R430:~$
```

3.Configure Vendor Hybrid Switches (Arista/Dell/HP/Juniper)

Follow Arista/HP/Dell/Juniper documentation instructions to activate OpenFlow ports on the hardware.

Configure Management network for non-OpenFlow network.

Configure the OpenFlow ports and connect the switch to the controller

Configure OpenFlow flows on Arista/HP/Dell/Juniper switches using a Controller.

HP Switch openflow-activation

```
team2@team2:~$
team2@team2:~$ ssh admin@172.16.10.4
Password:
Last login: Mon Nov 27 00:49:13 2023 from 172.16.10.100
SDN2-hpswitch>
SDN2-hpswitch>show openflow
OpenFlow configuration: Enabled
DPID: 0x0000001c737a1251
Description: SDN2-hpswitch
Controllers:
  configured: 172.16.10.100:6633
  connected: <none>
  connection count: 1
  keepalive period: 10 sec
Flow table state: Enabled
Flow table profile: full-match
Forwarding pipeline: flow
Bind mode: bindModeInterface
  interfaces:
    Et17-18
IP routing state: Disabled
Shell command execution: Disabled
Total matched: 76446 packets
SDN2-hpswitch>
```

Arista switch openflow-activation

```
team2@team2:~$
team2@team2:~$ ssh admin@172.16.10.5
Password:
Last login: Tue Nov 28 09:18:30 2023 from 172.16.10.100
SDN2-Aristaswitch>show openglow
% Invalid input
SDN2-Aristaswitch>show openflow
OpenFlow configuration: Enabled
DPID: 0x0000001c737a32fb
Description: SDN2-Aristaswitch
Controllers:
  configured: 172.16.10.100:6633
  connected: <none>
  connection count: 1
  keepalive period: 10 sec
Flow table state: Enabled
Flow table profile: full-match
Forwarding pipeline: flow
Bind mode: bindModeInterface
  interfaces:
    Et18
IP routing state: Disabled
Shell command execution: Disabled
Total matched: 89429 packets
SDN2-Aristaswitch>
```

```
SDN2-Aristaswitch#sh openflow
OpenFlow configuration: Enabled
DPID: 0x0000001c737a32fb
Description: SDN2-Aristaswitch
Controllers:
  configured: 172.16.10.100:6633
  connected: <none>
  connection count: 36
  keepalive period: 10 sec
Flow table state: Enabled
Flow table profile: full-match
Forwarding pipeline: flow
Bind mode: bindModeInterface
  interfaces:
    Et17-19
IP routing state: Disabled
Shell command execution: Disabled
Total matched: 3469805 packets
```

```
SDN2-Aristaswitch#sh ip int br
Interface              IP Address         Status     Protocol         MTU
Management1            172.16.10.5/24     up         up              1500
SDN2-Aristaswitch#
```

```
SDN2-Aristaswitch#sh openflow flows
Flow flow0000000000000000053:
  priority: 6500
  cookie: 0 (0x0)
  match:
    source Ethernet address: 00:16:9d:e5:c6:21/ff:ff:ff:ff:ff:ff
    destination Ethernet address: 5c:26:0a:24:8f:a7/ff:ff:ff:ff:ff:ff
    Ethernet type: IPv4
    IPv4 protocol: ICMP
    source TCP/UDP port or ICMP type: 0
  actions:
    output interfaces:
        Et18
  matched: 1 packets, 78 bytes
Flow flow0000000000000000003:
  priority: 5000
  cookie: 0 (0x0)
  match:
    source Ethernet address: 00:16:9d:e5:c6:21/ff:ff:ff:ff:ff:ff
    destination Ethernet address: 5c:26:0a:24:8f:a7/ff:ff:ff:ff:ff:ff
    Ethernet type: IPv4
    IPv4 protocol: TCP
  actions:
    output interfaces:
        Et18
  matched: 5455 packets, 970854 bytes
Flow flow0000000000000000002:
  priority: 5000
  cookie: 0 (0x0)
  match:
    destination Ethernet address: 00:16:9d:e5:c6:21/ff:ff:ff:ff:ff:ff
    Ethernet type: IPv4
    IPv4 protocol: TCP
  actions:
    output interfaces:
        Et17
  matched: 5977 packets, 535560 bytes
Flow flow0000000000000000183:
  priority: 3000
  cookie: 0 (0x0)
  match:
    destination Ethernet address: 5c:26:0a:24:8f:a7/ff:ff:ff:ff:ff:ff
    Ethernet type: ARP
  actions:
    output interfaces:
        Et18
  matched: 0 packets, 0 bytes
Flow flow0000000000000000173:
  priority: 2000
  cookie: 0 (0x0)
  match:
```

Objective 5 – Install and Configure the SDN Controller

Load the controller of your choice.

```
-rw-r--r-- 1 root root  3877 Nov 25 15:00 simple_switch_lacp_13.py
-rw-r--r-- 1 root root  4328 Nov 25 15:00 simple_switch_lacp.py
-rw-r--r-- 1 root root  3803 Nov 25 15:00 simple_switch.py
-rw-r--r-- 1 root root  4221 Nov 25 15:00 simple_switch_rest_13.py
-rw-r--r-- 1 root root  4954 Nov 25 15:00 simple_switch_snort.py
-rw-r--r-- 1 root root  4591 Nov 25 15:00 simple_switch_stp_13.py
-rw-r--r-- 1 root root  4979 Nov 25 15:00 simple_switch_stp.py
-rw-r--r-- 1 root root  3793 Nov 25 15:00 simple_switch_websocket_13.py
-rw-r--r-- 1 root root  1163 Dec  4 13:30 siteChecker.py
drwxr-xr-x 2 root root  4096 Dec  3 16:50 templates
-rw-r--r-- 1 root root 10436 Nov 25 15:00 wsgi.py
-rw-r--r-- 1 root root  4384 Nov 25 15:00 ws_topology.py
team2@team2:/home/willy/ryu/ryu/app$ sudo ryu-manager
[sudo] password for team2:
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
```

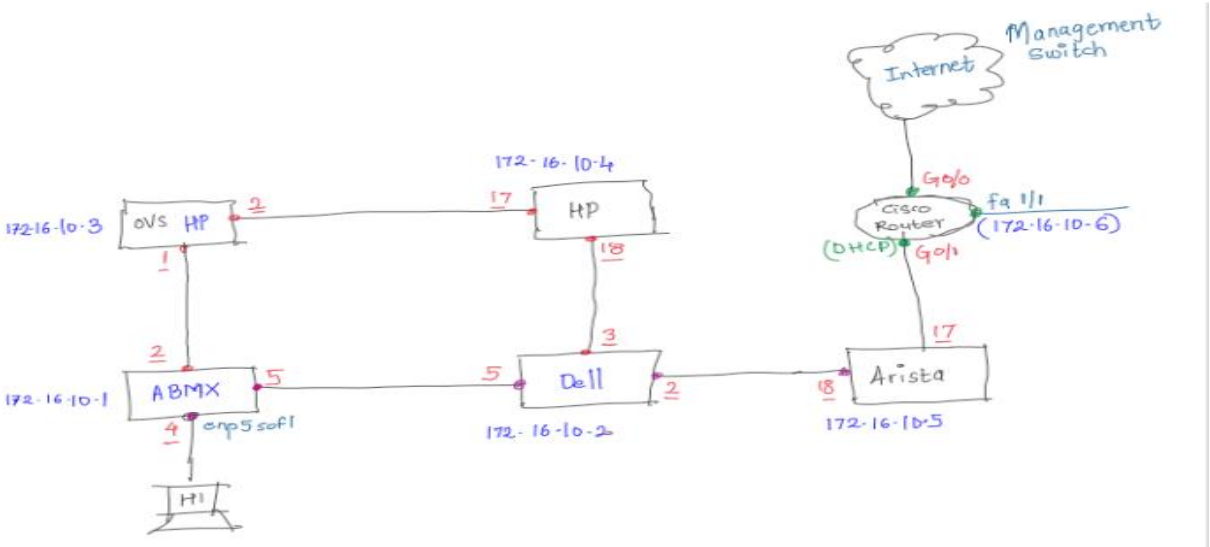
Configure the controller's IP address to be in the non-OpenFlow network upon connecting it to the Management network.

```
eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.5  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::9656:b9f8:fcd5:ebb7  prefixlen 64  scopeid 0x20<link>
        ether 14:18:77:3c:06:b6  txqueuelen 1000  (Ethernet)
        RX packets 13379868  bytes 1015888024 (1.0 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2913599  bytes 552801398 (552.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 18

eno2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.10.100  netmask 255.255.255.0  broadcast 172.16.10.255
        inet6 fe80::cc8b:267f:6b41:91cf  prefixlen 64  scopeid 0x20<link>
        ether 14:18:77:3c:06:b7  txqueuelen 1000  (Ethernet)
        RX packets 10314194  bytes 2184475251 (2.1 GB)
        RX errors 0  dropped 472  overruns 0  frame 0
        TX packets 8830249  bytes 2058893977 (2.0 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 19
```

Objective 6 – Application Programming

I. Automated OpenFlow configuration

1. Create an application that automates the OpenFlow configuration to the controller for all the hardware switches.
2. The application should fetch all the required connection information (switch IP, controller IP, OpenFlow port, OpenFlow version, OpenFlow transport protocol, etc.) from a NSOT file and then configure the switches accordingly.
3. The application should verify and display that each switch has established successful OpenFlow connectivity with the controller.



```
instantiating app /home/willy/ryu/ryu/app/myApp.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler

OvS in ABMX server is configured!
44cde7aa-fe80-4390-a31c-6446b0d20e98
    Bridge mybridge
        Controller "tcp:172.16.10.100:6633"
            is_connected: true
        Port eno4
            Interface eno4
        Port enp5s0f1
            Interface enp5s0f1
        Port eno3
            Interface eno3
        Port mybridge
            Interface mybridge
                type: internal
    ovs_version: "2.17.7"
```

```
f719b3c8-2bf0-471c-ad98-78a585686e2f
    Bridge mybridge
        Controller "tcp:172.16.10.100:6633"
            is_connected: true
        Port mybridge
            Interface mybridge
                type: internal
        Port eno4
            Interface eno4
        Port eno3
            Interface eno3
        Port enp6s0f1
            Interface enp6s0f1
    ovs_version: "2.17.7"
OvS in HP server is configured!
```

```
OvS in Dell server is configured!
f719b3c8-2bf0-471c-ad98-78a585686e2f
    Bridge mybridge
        Controller "tcp:172.16.10.100:6633"
            is_connected: true
        Port mybridge
            Interface mybridge
                type: internal
        Port eno4
            Interface eno4
        Port eno3
            Interface eno3
        Port enp6s0f1
            Interface enp6s0f1
    ovs_version: "2.17.7"
```

```
SSH successfull
OpenFlow configuration: Enabled
DPID: 0x0000001c737a1251
Description: SDN2-hpswitch
Controllers:
  configured: 172.16.10.100:6633
  connected:
    172.16.10.100:6633
      role: equal
      negotiated version: Version 1.3
  connection count: 72
  keepalive period: 10 sec
Flow table state: Enabled
Flow table profile: full-match
Forwarding pipeline: flow
Bind mode: bindModeInterface
  interfaces:
    Et17-19
IP routing state: Disabled
Shell command execution: Disabled
Total matched: 5411 packets
172.16.10.4 is configured
SSH successfull
OpenFlow configuration: Enabled
DPID: 0x0000001c737a32fb
Description: SDN2-Aristaswitch
Controllers:
  configured: 172.16.10.100:6633
  connected:
    172.16.10.100:6633
      role: equal
      negotiated version: Version 1.3
  connection count: 24
  keepalive period: 10 sec
Flow table state: Enabled
Flow table profile: full-match
Forwarding pipeline: flow
Bind mode: bindModeInterface
  interfaces:
    Et17-19
IP routing state: Disabled
Shell command execution: Disabled
Total matched: 16387 packets
172.16.10.5 is configured
```

II. DNS Blacklist

1. Create a program that will force DNS traffic, from any network device, to the controller, which will route the traffic to the “checker” website’s API to determine if the website is “good / bad.” Depending on the results returned from the website, the switches should “block/deny” traffic to the “bad” website and “allow/forward” traffic to the “good” website. OpenFlow table flow entries can be used, but a simple drop/forward is sufficient.
2. Three websites will be provided to accompany this lab:

   - **Site-Checker**: http://checksite.herokuapp.com
   - **Good-Site**: http://goodweb.herokuapp.com
   - **Bad-Site**: http://badweb.herokuapp.com

   Bonus implementation: If the requested URL is bad, instead of doing nothing, the PC will receive a modified DNS reply that will let the browser display contents of “warning" or “blocked’ website.

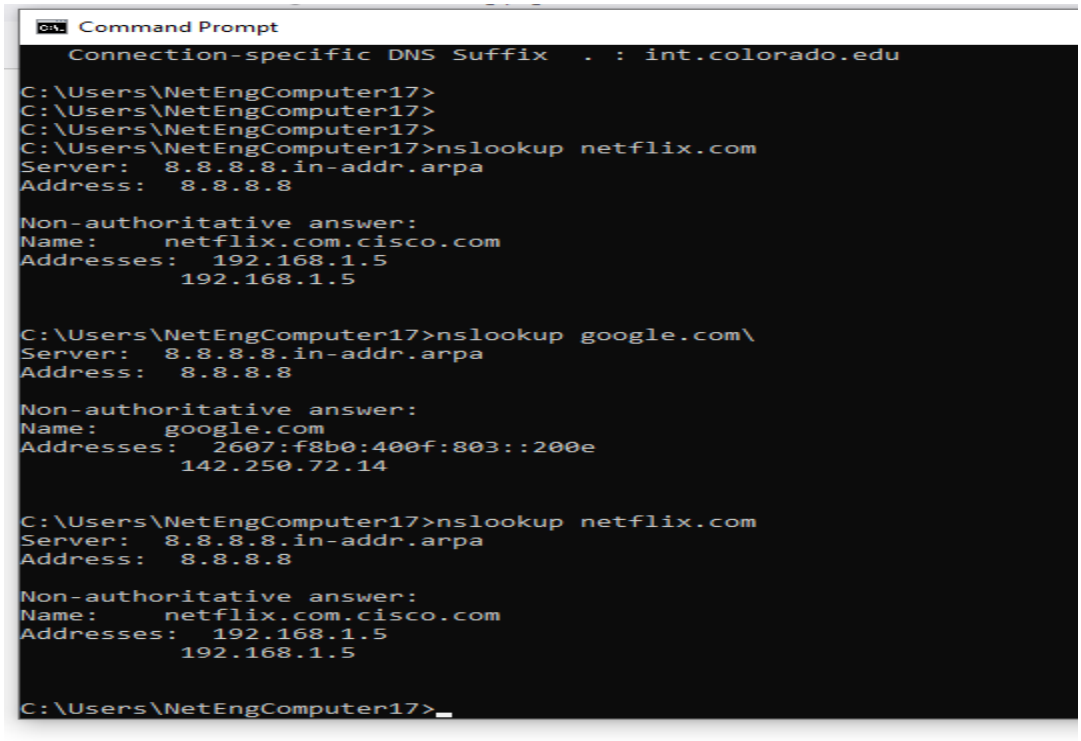
```python
from flask import Flask,render_template, request
import datetime,json
from collections import defaultdict
import requests
import json,logging

allowedSites = {
    "google": True,
    "youtube": True,
    "pict": True,
    "colorado.edu": True
}



app = Flask(__name__)
# log = logging.getLogger('werkzeug')
# log.disabled = True
icmpPath = "Path not detected yet"
@app.route("/", methods=["GET"])
def forbiddenSite_():
    return render_template("forbidden.html")

@app.route("/updatetopopath", methods=["POST"])
def icmpPathFunc_():
    global icmpPath
    icmpPath = request.json['icmpPath']
    print(icmpPath)
    return {"status": True }

@app.route("/topopath", methods=["GET"])
def showTopoPath_():
    global icmpPath
    print(icmpPath)
    return render_template("topopath.html",icmpPath=icmpPath)
```

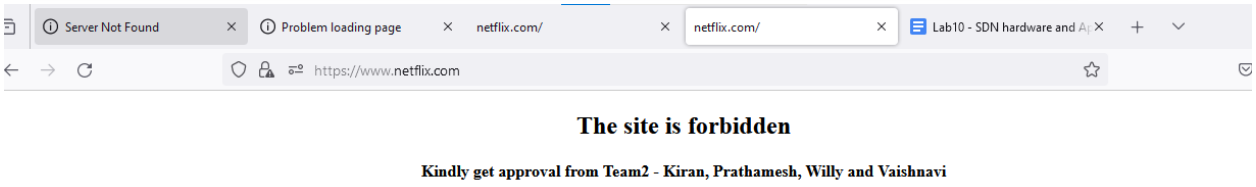
```python
@app.route("/checksite",methods=["POST"])
def checkSite_():
    url =  request.json['url']
    print("Checking for url"+url)
    for sites in allowedSites.keys():
        if sites in url:
            return {"status": True }, 200
    return {"status": False}, 200

if __name__ == "__main__":
    app.run(host='0.0.0.0',port=443,ssl_context='adhoc')
```

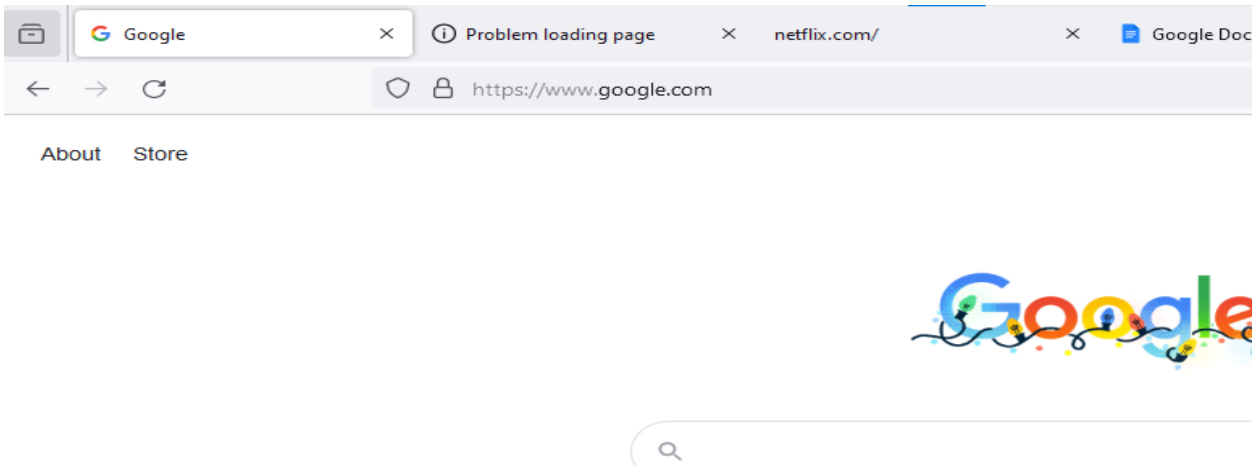
```
* DNS Request received for - google.com
Allowed checking from remote site checker
Checking for urlgoogle.com
192.168.1.5 - - [04/Dec/2023 18:30:45] "POST /checksite HTTP/1.1" 200 -
Flask Application allows domain
* Site Checker response - True
* DNS Query received on - 172.16.10.5
* Received on Port 18
* Allowing DNS query - google.com
* DNS request sent to outPort 17
* DNS Request received for - wpad.cisco.com
* Site Checker response - False
* URL denied access wpad.cisco.com
* DNS Request received for - wpad.cisco.com
* Site Checker response - False
* URL denied access wpad.cisco.com
* DNS Request received for - www.google.com
Allowed checking from remote site checker
Checking for urlwww.google.com
192.168.1.5 - - [04/Dec/2023 18:30:45] "POST /checksite HTTP/1.1" 200 -
```

```
Allowed checking from remote site checker
Checking for urlgoogle.com
192.168.1.5 - - [04/Dec/2023 18:39:53] "POST /checksite HTTP/1.1" 200 -
Flask Application allows domain
* Site Checker response - True
* DNS Query received on - 172.16.10.5
* Received on Port 18
* Allowing DNS query - google.com
* DNS request sent to outPort 17
ICMP ECHO detected on 172.16.10.1
 In Port : 4
 Out Port : 5
---------------172.16.10.1
ICMP ECHO detected on 172.16.10.2
 In Port : 5
 Out Port : 2
---------------172.16.10.2
ICMP ECHO detected on 172.16.10.5
 In Port : 18
 Out Port : 17
HOST < -- > (4) OVS - ABMX (5) < -- > (5) OVS - Dell (2) < -- > (18) OVS - Arista (Towards Gateway) (17) < -- > Gateway(Internet)
192.168.1.5 - - [04/Dec/2023 18:39:53] "POST /updatetopopath HTTP/1.1" 200 -
---------------172.16.10.5
ICMP ECHO detected on 172.16.10.1
 In Port : 4
 Out Port : 5
---------------172.16.10.1
ICMP ECHO detected on 172.16.10.2
 In Port : 5
 Out Port : 2
---------------172.16.10.2
ICMP ECHO detected on 172.16.10.5
 In Port : 18
 Out Port : 17
```

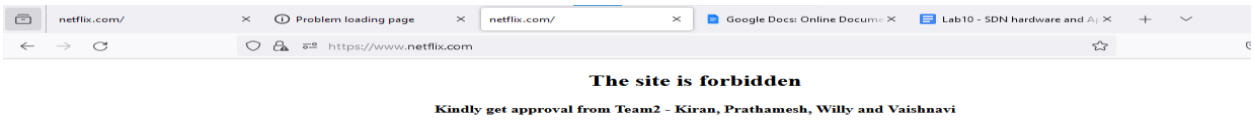
Result :


Command Prompt
Connection-specific DNS Suffix . : int.colorado.edu
C:\Users\NetEngComputer17>
C:\Users\NetEngComputer17>
C:\Users\NetEngComputer17>
C:\Users\NetEngComputer17>nslookup netflix.com
Server: 8.8.8.8.in-addr.arpa
Address: 8.8.8.8
Non-authoritative answer:
Name: netflix.com.cisco.com
Addresses: 192.168.1.5
192.168.1.5
C:\Users\NetEngComputer17>nslookup google.com\
Server: 8.8.8.8.in-addr.arpa
Address: 8.8.8.8
Non-authoritative answer:
Name: google.com
Addresses: 2607:f8b0:400f:803::200e
142.250.72.14
C:\Users\NetEngComputer17>nslookup netflix.com
Server: 8.8.8.8.in-addr.arpa
Address: 8.8.8.8
Non-authoritative answer:
Name: netflix.com.cisco.com
Addresses: 192.168.1.5
192.168.1.5
C:\Users\NetEngComputer17>


Server Not Found
Problem loading page
netflix.com/
netflix.com/
Lab10 - SDN hardware and
https://www.netflix.com
The site is forbidden
Kindly get approval from Team2 - Kiran, Prathamesh, Willy and Vaishnavi


Google
Problem loading page
netflix.com/
Google Doc
https://www.google.com
About
Store
Google

Bonus implementation: If the requested URL is bad, instead of doing nothing, the PC will receive a modified DNS reply that will let the browser display contents of “warning" or “blocked’ website.

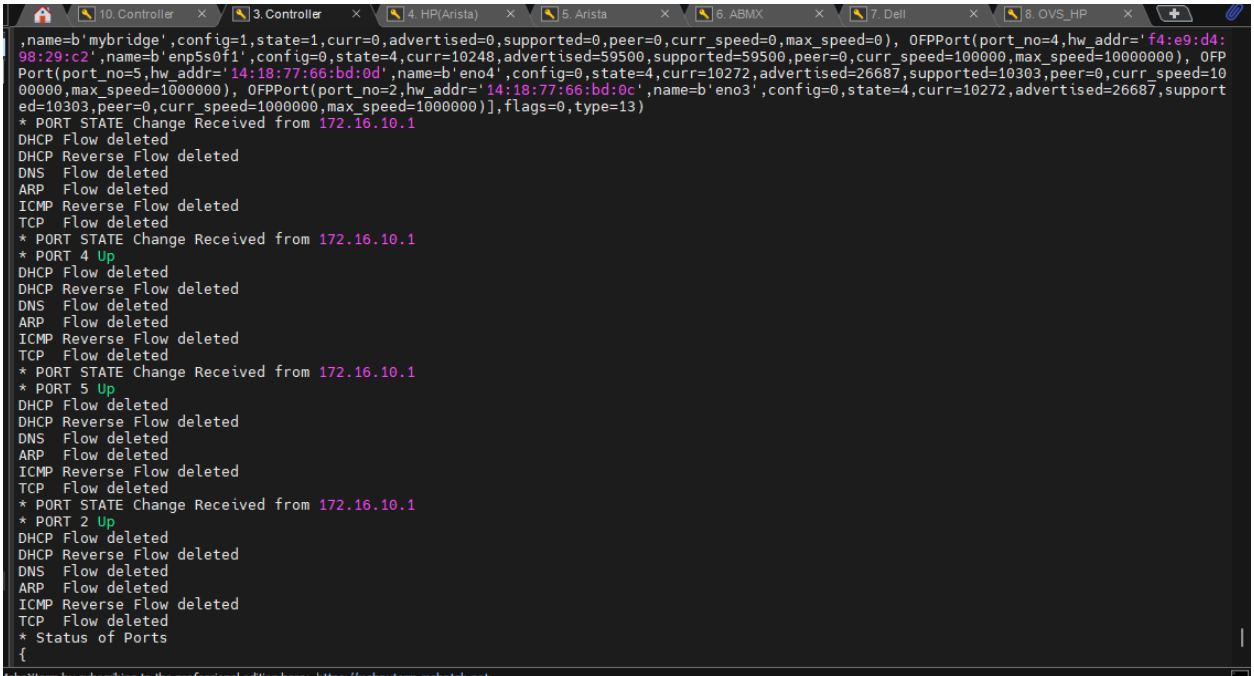


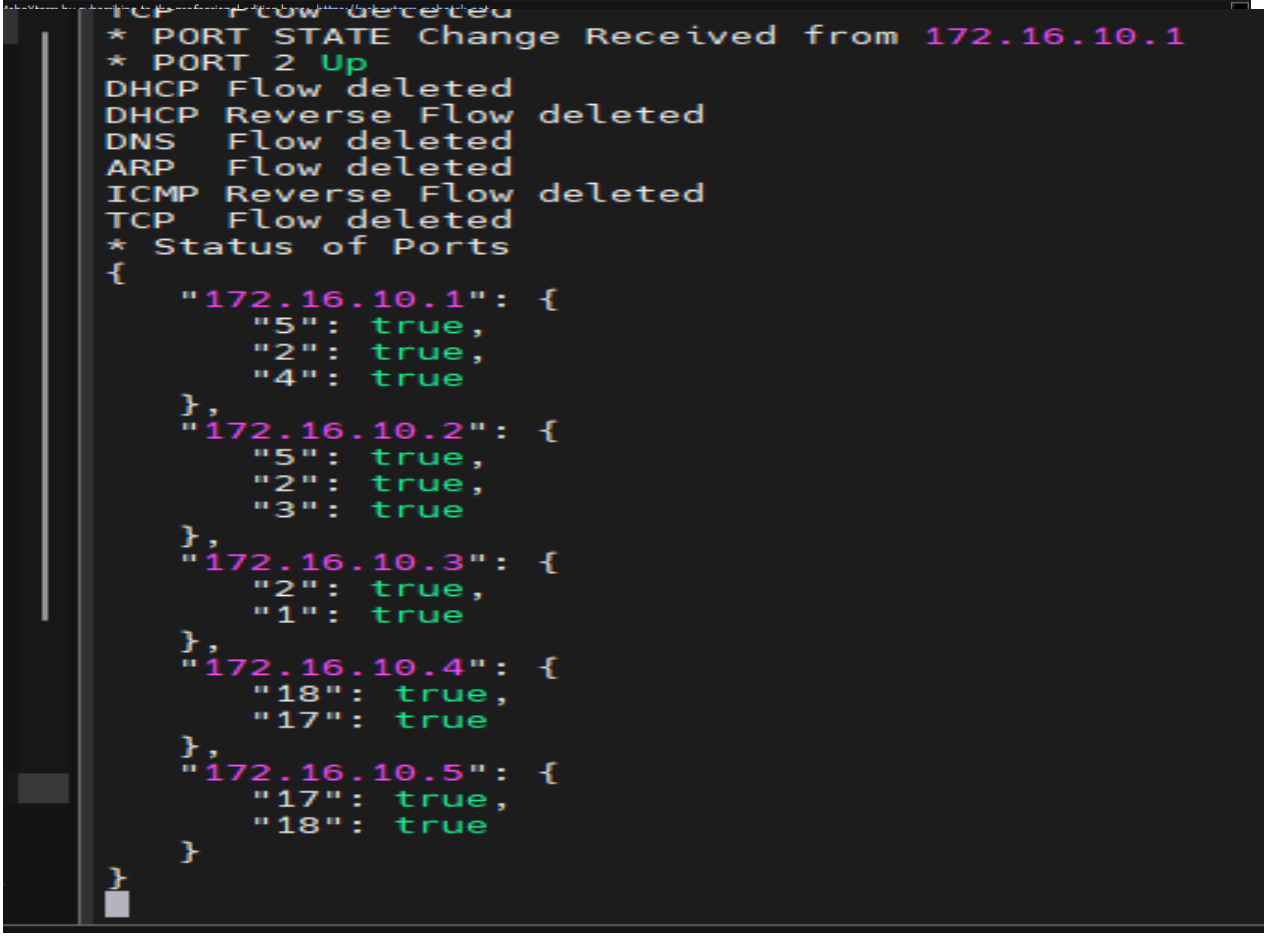
III. DHCP application

1. Create a program that will force DHCP requests originated from hosts to the Internet router (DHCP server).

```
C:\Users\NetEngComputer17>ipconfig /release

Windows IP Configuration

An error occurred while releasing interface Ethernet : An address has not yet been associated with the network endpoint.

No operation can be performed on Local Area Connection* 1 while it has its media disconnected.
No operation can be performed on Local Area Connection* 2 while it has its media disconnected.
No operation can be performed on Wi-Fi while it has its media disconnected.

C:\Users\NetEngComputer17>
C:\Users\NetEngComputer17>
C:\Users\NetEngComputer17>ipconfig /renew

Windows IP Configuration

No operation can be performed on Local Area Connection* 1 while it has its media disconnected.
No operation can be performed on Local Area Connection* 2 while it has its media disconnected.
No operation can be performed on Wi-Fi while it has its media disconnected.

Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : cisco.com
   Link-local IPv6 Address . . . . . : fe80::c21e:941c:7b94:ea47%6
   IPv4 Address. . . . . . . . . . . : 10.0.0.3
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.0.0.1

Ethernet adapter VirtualBox Host-Only Network:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::f5a:fea7:e358:2936%7
   IPv4 Address. . . . . . . . . . . : 192.168.56.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . : int.colorado.edu

C:\Users\NetEngComputer17>
C:\Users\NetEngComputer17>
```

Ensure that the DHCP requests forwarded from the switches to the server is unicast and not broadcast.





```
  import cryptography
loading app ryu.controller.ofp_handler
instantiating app myApp.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
* DHCP DISCOVER/REQUEST Packet In Received from 172.16.10.1
 destination.MAC = ff:ff:ff:ff:ff:ff     source.MAC = 5c:26:0a:24:8f:a7
 destination.IP  = 255.255.255.255   source IP = 0.0.0.0
 dataPathId 0022095314992396
 InPort 4
 * Adding flow for reverse traffic from DHCP Server directly to client:
   Switch: [ 172.16.10.1 ] OVS - ABMX
   DPID: 0022095314992396
   Match Criteria:
        MAC Destination: 5c:26:0a:24:8f:a7
        Protocol: UDP
        UDP Source Port: 67
   Actions:
        Out Port: 4
 * Packet sent out on Port 5
* DHCP DISCOVER/REQUEST Packet In Received from 172.16.10.2
 destination.MAC = ff:ff:ff:ff:ff:ff     source.MAC = 5c:26:0a:24:8f:a7
 destination.IP  = 255.255.255.255   source IP = 0.0.0.0
 dataPathId 0026830032228362
 InPort 5
 * Adding flow for reverse traffic from DHCP Server directly to client:
   Switch: [ 172.16.10.2 ] OVS - Dell
   DPID: 0026830032228362
   Match Criteria:
        MAC Destination: 5c:26:0a:24:8f:a7
        Protocol: UDP
        UDP Source Port: 67
   Actions:
        Out Port: 5
 * Packet sent out on Port 2
* DNS Request received for - dns.google
Allowed checking from remote site checker
```

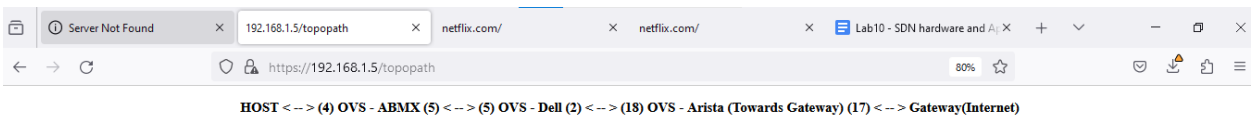
unicast

unicast

IV. SDN trace application

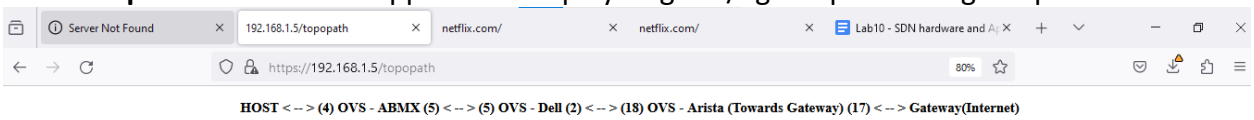
1. The objective of this application is to trace the path in an SDN OpenFlow network.
2. The application should take source and destination inputs from the user via either the CLI or GUI (Flask).
   CLI (Every time when page refresh)

```
* URL denied access detectportal.firefox.com
HOST < -- > (4) OVS - ABMX (5) < -- > (5) OVS - Dell (2) < -- > (18) OVS - Arista (Towards Gateway) (17) < -- > Gateway(Internet)
192.168.1.68 - - [04/Dec/2023 19:01:02] "GET /topopath HTTP/1.1" 200 -
HOST < -- > (4) OVS - ABMX (5) < -- > (5) OVS - Dell (2) < -- > (18) OVS - Arista (Towards Gateway) (17) < -- > Gateway(Internet)
192.168.1.68 - - [04/Dec/2023 19:01:13] "GET /topopath HTTP/1.1" 200 -
HOST < -- > (4) OVS - ABMX (5) < -- > (5) OVS - Dell (2) < -- > (18) OVS - Arista (Towards Gateway) (17) < -- > Gateway(Internet)
192.168.1.68 - - [04/Dec/2023 19:01:23] "GET /topopath HTTP/1.1" 200 -
```

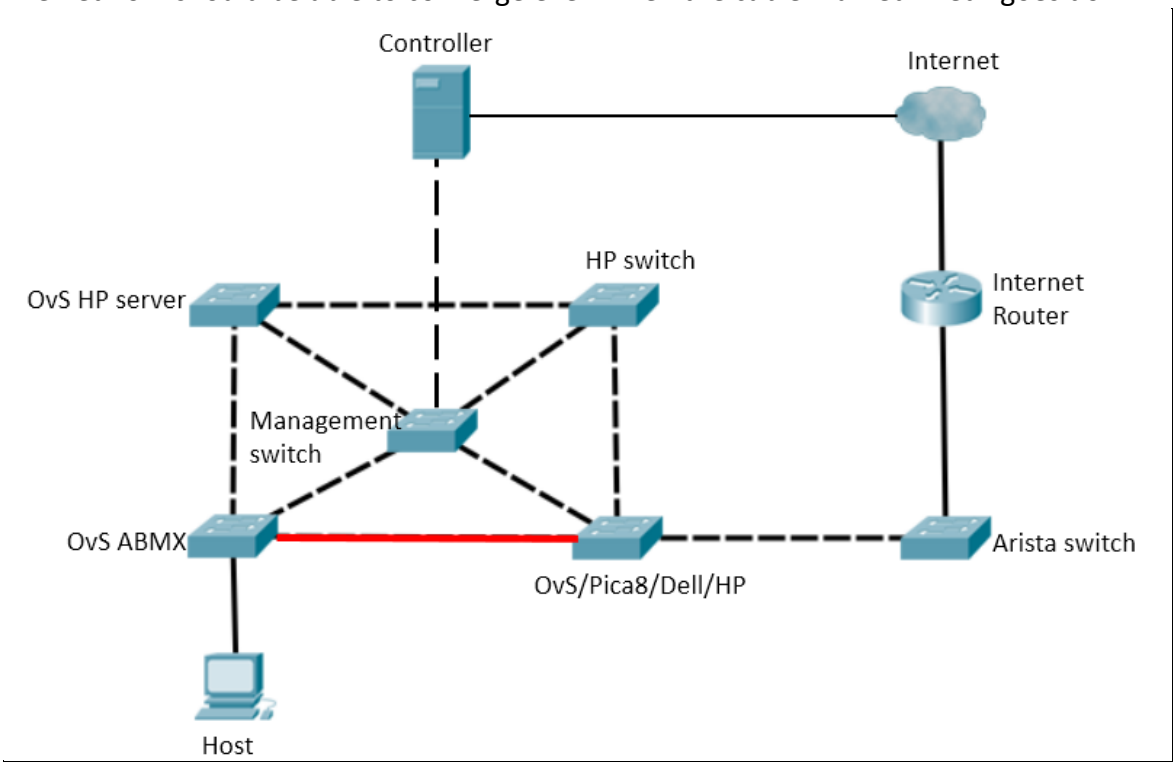


HOST < -- > (4) OVS - ABMX (5) < -- > (5) OVS - Dell (2) < -- > (18) OVS - Arista (Towards Gateway) (17) < -- > Gateway(Internet)

Using OpenFlow messages, the application traces the path (i.e. prints out the DPID's of the OpenFlow switches in the path).
Bonus implementation: the application displays ingress/egress ports along the path.



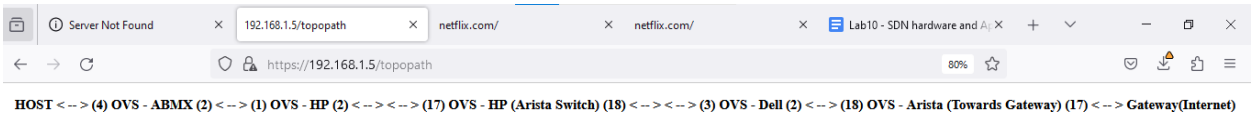
The network should be able to converge even when the cable marked "Red" goes down:



Now we disconnect the link and check the down ports

```
* PORT 2 Up
DHCP Flow deleted
DHCP Reverse Flow deleted
DNS  Flow deleted
ARP  Flow deleted
ICMP Reverse Flow deleted
TCP  Flow deleted
* Status of Ports
{
   "172.16.10.1": {
      "5": false,
      "2": true,
      "4": true
   },
   "172.16.10.2": {
      "5": false,
      "2": true,
      "3": true
   },
   "172.16.10.3": {
      "2": true,
      "1": true
   },
   "172.16.10.4": {
      "18": true,
      "17": true
   },
   "172.16.10.5": {
      "17": true,
      "18": true
   }
}
* DNS Request received for - detectportal.firefox.com
* Site Checker response - False
* URL denied access detectportal.firefox.com
* DNS Request received for - detectportal.firefox.com
```

On gui converged Path can be seen



Applications functionality explained :

DNS Blacklist - The application is used to prevent certain IP addresses or domains from communicating with a network or accessing services based on their inclusion in a blacklist. These lists contain the IP addresses or domain names of servers or sources that have been identified as sending spam, hosting malware, or engaging in other malicious activities. In our case, we are allowing the google, youtube and Colorado as allowed sites.

```
1   from flask import Flask,render_template, re
2   import datetime,json
3   from collections import defaultdict
4   import requests
5   import json,logging
6
7   allowedSites = {
8       "google": True,
9       "youtube": True,
10      "pict": True,
11      "colorado.edu": True
12  }
13
```

And the algorithm works as :

If packet_in (dstn_port = 53 && ip.protocol = udp):

1. Parse the DNS packet and fetch the **URL**:
    a. Check if the **URL** is locally **whitelisted**:
    [need to do this to reduce the load on flask application serving as site checker]
        i. If allowed locally:
            1. Generate a **POST** request to check the site on remote flask application.
            2. If allowed, then mark the site as **ALLOWED**
        ii. Else:
            1. Mark the site as **NOT ALLOWED**
    b. If **ALLOWED**:
        i. Install a reverse flow for the DNS REPLY.
        ii. Send the packet to the next hop towards the gateway.
    c. Else:
        i. Craft a DNS REPLY using scapy.(**DNS Resolution = controllerIP**)
        [since the browser will generate a TCP request with http://controllerIP:443 it will redirect it to a page showing the site is blocked]
        ii. Send the packet crafted on to the **in_port.**

DHCP Application: It's a network protocol that's responsible for automatically assigning IP addresses and other network configuration information to devices within a network. DHCP assigns IP addresses dynamically to devices on a network, eliminating the need for manual configuration. When a device connects to a network, the DHCP server provides it with an IP address from a pool of available addresses. Besides IP addresses, DHCP can also provide additional network configuration details, such as subnet mask, default gateway, DNS server information, and more. This ensures that devices connecting to the network have all the necessary settings to communicate properly. In our case, we are configuring DHCP using app. And the algorithm works as:

If packet_in (dst_port = 67 && ip.protocol =udp && dst_ip = 255.255.255.255):

1. Extract all the ports on the switch towards DHCP server.
2. Check if these extracted ports are **UP.**
3. Install flows for DHCP DISCOVER towards the DHCP server, with all the available egress ports, **with decremental priorities**.
   The **priority** with **shortest path is greater**, then the next greater and so on.
4. Install flow for DHCP reply from the DHCP server:
   [*We install flows with egress ports only if the port is UP]*
   a. Check all the ports on the switch towards the host.
   b. Extract the ports **and install flows with decremental priority**.
5. Instruct the switch to send the **port out on the port with highest priority**.

Path Tracer:

Path tracer application work with ICMP. When the edge device generates a PING request the controller received the ECHO message and starts tracking the ECHO request's **in_port** and **out_port**. Once, the Echo Packet is thrown towards the Gateway, the controller sends a POST request of the final ICMP path formed, to the flask Application on endpoint **/updatetopoPath**.

The flask application continuously refreshes the page and prints the ICMP path.
Alogrithm is as follows:

global ICMP_PATH
if packet_in(ip.protocol = ICMP && destination_MAC = GatewayMAC):

1. Extract all the ports on the switch towards the DHCP server.
2. Check if the extracted ports are UP.
3. If the ECHO request received from the **edge Port** (port towards the host):
   a. Record the **(inPort, outport, switch-Openflow-Channel IP) in ICMP_PATH**
4. If the ECHO request received from a **non-edge** port AND
   Outport is not the **Gateway Port**:
   a. Record the **(inPort, outport, switch-Openflow-Channel IP) in ICMP_PATH**
5. If the outport of ECHO Request is a Gateway Port:
   a. Record the **(inPort, outport, switch-Openflow-Channel IP) in ICMP_PATH**
   b. Send the POST request to flask application to update **ICMP_PATH**

Failover Tracking:

While installing flows we check if the port is UP before installing the flow, by doing a local lookup into the local state table of all ports on all switches.

Anytime a wire is plugged out, a **PortStats** event is sent asynchronously from the switch to the controller, we detect that and update the local state table of ports. Additionally, we remove flows for that port the switch, with that egress port. This way, the controller can install new flows since the switch won't have flows. Thus, new flows are added with new ports that are **UP.**

Contributions –

1. Kiran – Software (DNS, DHCP, Tracer application) – 100/100
2. Prathamesh – Software (DNS, DHCP, Tracer application) – 100/100
3. Srivaishnavi – Hardware Setup and Software (OpenFlow AutoConfiguration) – 100/100
4. Willy – Hardware Setup and Software (OpenConfiguration) – 100/100