CS341 Network Lab 3: Building and Controlling Your Own Network with SDN

Logistics

- The due date is 11:59pm on November 8, 2023
- Submit a single zip file composed of `task_topology.py`, and `task_controller.py` via Gradescope.
- Discussion and Q&A: Lab 3 @ Piazza

Overview

In Lab 3, you will implement a fully-functioning network of multiple routers/switches on your personal computer (or a single cloud instance). An open-source testbed tool, mininet (http://mininet.org/) will be used throughout this lab. Mininet lets network experimenters avoid many manual hassles for setting up routers and hosts at a large scale.

After implementing your own multi-router-and-host network and testing basic connectivity, you are asked to implement a couple of simple routing applications using software-defined networking (SDN) principles. The application will be implemented using the OpenFlow protocol and the POX controller. You will then show that the routing protocol you implement correctly forward packets between hosts.

Along with the basic connectivity, you will also implement a simple censorship system. You are asked to implement DNS-based censorship mechanisms. You will first implement a simple yet less efficient censorship mechanism with the SDN features. Then, you will be asked to improve your censorship system by minimizing the number of packets being sent to the SDN controller. You will examine whether your censorship system blocks only the censored domains while correctly handling other domains.

The lab will be graded based on the correctness of your implementation.

Mininet Overview

What is Mininet?

Mininet is a network emulation of hosts, links, and switches. It is useful for testing OpenFlow and SDN, providing Python APIs. In this lab, we will use Mininet for emulating custom networks. For more information, visit Mininet official document.

Mininet consists of a controller, switches, and hosts. A controller is a component that manages all switches in the network. It can insert routing rules in switches, and manually handle unhandled packets from switches. Switches are components that relay packets from and to other switches and hosts. Switches connect to the controller, then receive routing rules and send unhandled packets. Hosts are components that work as normal end-nodes. These hosts can communicate with other hosts through the network.

Project Setup

We provide two options to setup the lab environment:

1. Environment Setup via Vagrant and Virtualbox
2. Environment Setup on Ubuntu 20.04

Choose a proper setup based on your local computing environment:
If you are using **Windows with Hyper-V off**, or **Ubuntu 22.04**, follow Environment Setup via Vagrant and Virtualbox

If you are using **macOS**, **Windows with Hyper-V on**, or Other OS, prepare Ubuntu 20.04 virtual machine your own (WSL does not work), then follow Environment Setup on Ubuntu 20.04

Environment Setup via Vagrant and Virtualbox

Virtualbox and Vagrant installation

To ensure Mininet works correctly, we provide a VM with Mininet and several requirements pre-installed. We use Virtualbox for a VM system, and Vagrant as a VM management software. Follow the official document for setup. We recommend you to conduct this lab on your own machine because mininet is lightweight and designed to run on a single laptop.

- Virtualbox: Installation document, Download page
- Vagrant: Installation document, Download page

Important: Please troubleshoot installation problems with Virtualbox or Vagrant on your machine on your own. TAs are not familiar with all the configuration issues with the tools, and thus it is your responsibility to install these popular software projects. Search the web first before you ask any questions at piazza.
For your information, we have confirmed that following environments work:

- Ubuntu 22.04 with Intel CPU
- Windows 11 with Intel CPU (Hyper-V disabled)

We have also confirmed that following environments **not** work:

- macOS 13 with M1
- Windows 11 with Intel CPU (Hyper-V enabled)

If your environment do not work, please install Ubuntu 20.04 (virtual machine) by yourself, and follow Environment Setup on Ubuntu 20.04

Environment Setup

Start from cloning the github repository.

```
$ git clone https://github.com/NetSP-KAIST/cs341-23f-lab3-handout.git
```

Setup VM environment by following command:

```
$ cd cs341-23f-lab3-handout
$ vagrant up
```

This will show following messages:

```
cs341-23f-lab3-handout$ vagrant up
Bringing machine 'mnvm' up with 'virtualbox' provider...
==> mnvm: Importing base box 'ubuntu/focal64'...
==> mnvm: Matching MAC address for NAT networking...
==> mnvm: Checking if box 'ubuntu/focal64' version '20230506.0.0' is up to date...
==> mnvm: Setting the name of the VM:
cs341-23f-lab3-handout_mnvm_xxxxxxxxxxxxx_xxxxx
==> mnvm: Clearing any previously set network interfaces...
==> mnvm: Preparing network interfaces based on configuration...

...

    mnvm: The following additional packages will be installed:
    mnvm:   binutils binutils-common binutils-x86-64-linux-gnu blt cpp cpp-9
    mnvm:   fontconfig-config fonts-dejavu-core gcc-9 gcc-9-base libasan5
libatomic1

...

    mnvm: Installing POX into /home/vagrant/pox...
    mnvm: Cloning into 'pox'...
    mnvm: ++ ln -s /vagrant/controller.py /vagrant/dump.py /vagrant/graph.py
/vagrant/server.py /vagrant/task_controller.py /vagrant/task_topology.py
/vagrant/test.py /vagrant/topology.py /vagrant/dns.c /vagrant/Makefile
/vagrant/client.sh /home/vagrant/
    mnvm: ++ ln -s /vagrant/controller.py /home/vagrant/pox/pox/misc/
    mnvm: ++ cd /home/vagrant/
    mnvm: ++ make
    mnvm: gcc -Wall dns.c -o dns
```

Red text does not always mean error. Please check if the setup is correctly done, by reading the content of the text.

Now, connect to vm by following command:

```
$ vagrant ssh
```

This will show the following message, as you connect to the remote server via ssh:

```
cs341-23f-lab3-handout$ vagrant ssh
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-148-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue May  9 18:51:53 UTC 2023

  System load:  0.03              Processes:               124
  Usage of /:   5.6% of 38.70GB   Users logged in:         0
  Memory usage: 13%               IPv4 address for enp0s3: 10.0.2.15
  Swap usage:   0%


Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

4 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

New release '22.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


vagrant@mnvm:~$
```

Now, test if the VM is correctly configured by testing mininet with following command:

```
$ sudo mn --test pingall
```

This will show following messages:

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
```

```
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 6.060 seconds
```

Congratulations! You now have installed Mininet on your local environment.

If you want to make SSH connection through external software (vscode, etc), you can get SSH information via `vagrant ssh-config`

Environment Setup on Ubuntu 20.04

If your machine does not support Virtualbox, try installing Ubuntu 20.04 with other virtualization software, or natively.

- For Windows 11 with Hyper-V enabled, you can use VirtualBox
- For macOS, you can use UTM

Note that choosing an option to update while installing Ubuntu 20.04 might update Ubuntu to 22.04, which makes Mininet not work. Currently, **Mininet 2.3.0 does not support Ubuntu 22.04** and some other latest OSes.

If you have prepared Ubuntu 20.04 machine, start from cloning github repository:

```
$ git clone https://github.com/NetSP-KAIST/cs341-23f-lab3-handout.git
```

Setup VM environment by following command:

```
$ cd cs341-23f-lab3-handout
$ ./setup.sh
```

Now, test if the VM is correctly configured by testing mininet with following command:

```
$ sudo mn --test pingall
```

This will show following messages:

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
```

```
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 6.060 seconds
```

Congratulations! You now have installed Mininet on your local environment.

Task 1: Building Your Own Network

Here, you will implement a “network topology builder”, which generates a network topology from a given network graph.

open `task_topology.py`, you will see following code:

```
from typing import List, Tuple

from mininet.topo import Topo

class Topology(Topo):
    def build(self, switches: List[str], hosts: List[str], links: List[Tuple[str,
str, int]]) -> None:
        # KAIST CS341 SDN Lab Task 1: Building Your Own Network
        #
        # input:
        # - switches: List[str]
        #     -> List of switch names
        # - hosts: List[str]
        #     -> List of host names
        # - links: List[Tuple[str,str,int]]
```

```
        #     -> List of links, which is represented by a tuple
        #        The first and the second components represents name of the
components
        #        The third component represents cost of the link; not used in this
task

        ###
        # YOUR CODE HERE
        ###

        '''
        # ex) switches = ['s1'], hosts = ['h1'], links = [('s1','h1',100),]
        self.addSwitch('s1')
        self.addHost('h1')
        self.addLink('s1', 'h1')
        '''
```

You can test your code by following:

```
$ sudo ./test.py --task 1
```

This will open Mininet CLI. Type `help` for available commands and other information. You can check your link state with `net`, and check connectivity with `pingall`.
If properly configured, the result of the net should match with the given graph, and pingall should make a 0% drop (because we will use connected graphs only for testing in this lab). Result of the `net` command can be different, as the graph is created randomly at every execution.

```
mininet> net
h1 h1-eth0:s2-eth2
h2 h2-eth0:s4-eth4
h3 h3-eth0:s3-eth2
s1 lo:  s1-eth1:s4-eth2
s2 lo:  s2-eth1:s4-eth3 s2-eth2:h1-eth0
s3 lo:  s3-eth1:s5-eth2 s3-eth2:h3-eth0
s4 lo:  s4-eth1:s5-eth1 s4-eth2:s1-eth1 s4-eth3:s2-eth1 s4-eth4:h2-eth0
s5 lo:  s5-eth1:s4-eth1 s5-eth2:s3-eth1
c0
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

Task 2: Getting familiarized with POX

So far, all the networks in Mininet you tested are composed of switches and hosts. From now on, there will be one more type of node in the network, a “controller”. In our experiments, there exists one controller in a network, and it manages forwarding rules of all the switches. In a given

Mininet network, a controller connects to all the switches, and pushes forwarding rules at the network startup phase.

Here, you will learn how to build a logic for a POX controller, using OpenFlow. You are required to implement a very basic forwarding rule, by writing a couple of algorithms for the controller.

Now open `task_controller.py`, you will see following code:

```
#!/usr/bin/python3

import pox.openflow.libopenflow_01 as of

# KAIST CS341 SDN Lab Task 2, 3, 4
#
# All functions in this file runs on the controller:
#   - init(net):
#       - runs only once for network, when initialized
#       - the controller should process the given network structure for future
behavior
#   - addrule(switchname, connection):
#       - runs when a switch connects to the controller
#       - the controller should insert routing rules to the switch
#   - handlePacket(packet, connection):
#       - runs when a switch sends unhandled packet to the controller
#       - the controller should decide whether to handle the packet:
#           - let the switch route the packet
#           - drop the packet
#
# Task 2: Getting familiarized with POX
#   - Let switches "flood" packets
#   - This is not graded
#
# Task 3: Implementing a Simple Routing Protocol
#   - Let switches route via Dijkstra
#   - Match ARP and ICMP over IPv4 packets
#
# Task 4: Implementing simple DNS censorship
#   - Let switches send DNS packets to Controller
#       - By default, switches will send unhandled packets to controller
#   - Drop DNS requests for asking cs341dangerous.com, relay all other packets
correctly
#
# Task 5: Implementing simple HTTP censorship
#   - Let switches send HTTP packets to Controller
#       - By default, switches will send unhandled packets to controller
#   - Additionally, drop HTTP requests for heading cs341dangerous.com, relay all
other packets correctlys


###
# If you want, you can define global variables, import libraries, or do others
###
```

```
def init(net) -> None:
    #
    # net argument has following structure:
    #
    # net = {
    #    'hosts': {
    #         'h1': {
    #             'name': 'h1',
    #             'IP': '10.0.0.1',
    #             'links': [
    #                 # (node1, port1, node2, port2, link cost)
    #                 ('h1', 1, 's1', 2, 3)
    #             ],
    #         },
    #         ...
    #     },
    #     'switches': {
    #         's1': {
    #             'name': 's1',
    #             'links': [
    #                 # (node1, port1, node2, port2, link cost)
    #                 ('s1', 2, 'h1', 1, 3)
    #             ]
    #         },
    #         ...
    #     }
    # }
    #
    ###
    # YOUR CODE HERE
    ###
    pass

def addrule(switchname: str, connection) -> None:
    #
    # This function is invoked when a new switch is connected to controller
    # Install table entry to the switch's routing table
    #
    # For more information about POX openflow API,
    # Refer to [POX official document](https://noxrepo.github.io/pox-doc/html/),
    # Especially [ofp_flow_mod - Flow table
modification](https://noxrepo.github.io/pox-doc/html/#ofp-flow-mod-flow-table-modif
ication)
    # and [Match
Structure](https://noxrepo.github.io/pox-doc/html/#match-structure)
    #
    # your code will be look like:
    # msg = ....
    # connection.send(msg)
    ###
    # YOUR CODE HERE
    ###
    pass
```

```
from scapy.all import * # you can use scapy in this task

def handlePacket(switchname, event, connection):
    global bestport
    packet = event.parsed
    if not packet.parsed:
        print('Ignoring incomplete packet')
        return
    # Retrieve how packet is parsed
    # Packet consists of:
    #  - various protocol headers
    #  - one content
    # For example, a DNS over UDP packet consists of following:
    # [Ethernet Header][           Ethernet Body            ]
    #                  [IPv4 Header][       IPv4 Body       ]
    #                               [UDP Header][ UDP Body  ]
    #                                           [DNS Content]
    # POX will parse the packet as following:
    #   ethernet --> ipv4 --> udp --> dns
    # If POX does not know how to parse content, the content will remain as `bytes`
    #     Currently, HTTP messages are not parsed, remaining `bytes`. you should
parse it manually.
    # You can find all available packet header and content types from
pox/pox/lib/packet/
    packetfrags = {}
    p = packet
    while p is not None:
        packetfrags[p.__class__.__name__] = p
        if isinstance(p, bytes):
            break
        p = p.next
    print(packet.dump()) # print out unhandled packets
    # How to know protocol header types? see name of class

    # If you want to send packet back to switch, you can use of.ofp_packet_out()
message.
    # Refer to [ofp_packet_out - Sending packets from the
switch](https://noxrepo.github.io/pox-doc/html/#ofp-packet-out-sending-packets-from
-the-switch)
    # You may learn from [l2_learning.py](pox/pox/forwarding/l2_learning.py), which
implements learning switches
```

There are three functions: `init`, `addrule`, `handlePacket`

- `init` is called only once at network initialisation, before the `addrule` function is called. The controller should learn how the network is structured from the first argument, and get prepared for inserting rules to switches. The argument includes information about hosts, switches, and links.
- `addrule` is called when a controller connects to a switch. Because the controller connects to all the switches, the function will be called N times, where N is the number of switches in the network. The first argument is the name of the connected switch (ex: s1), and the second argument is a connection object described in the official document. You

do not need to know too many details about the object, the only thing you will need is that you can push new rules to switch by `connection.send`. You can find several examples in the official document.
- `handlePacket` is called when a controller receives unhandled packets from switches. The controller can make various actions, including followings:
    - parsing and reading content of the packet
    - sending the packet back to the switch, possibly providing which port to relay

In this task, you should make changes in `init` and `addrule`. Your controller should let switches to "flood" incoming packets, by forwarding packets to all ports except the ingress port. In this task, we will only test two types of packets, ARP and IPv4, and you can ignore other packets. You may refer to EtherType to check if packets are these types.

To test your code, you should open two sessions with two terminals. The first session will execute the POX controller, and the second session will initiate the Mininet composed of switches and hosts. Launch **POX controller first**, and **Mininet later**, so that switches in the Minnet can find the controller to connect. Execute following commands in that order, at each terminal session:

`$ sudo pox/pox.py misc.controller`	`$ sudo ./test.py --task 2`

This command will initialize the Mininet network with POX controller with your implemented logic. The startup takes some time, waiting for all switches to connect a controller and pushing rules. This command will finally open Mininet CLI as you have already used in Task 1. You can test the network with `pingall` command, which should yield 0% drop rate.

- `$ sudo ./test.py --task 2` will show `Unable to contact the remote controller at 127.0.0.1:6653`. It is totally fine; the controller is running on `127.0.0.1:6663`, and mininet tries to find the controller from `127.0.0.1:6653` and `127.0.0.1:6663` in order.
- This task will *not* be graded. However, we recommend you to do this task, as we think this helps you understand how to use OpenFlow before following tasks.
- You may not need to use the `init` function here.

Task 3: Implementing a Simple Routing Protocol

In this task, you will implement a simple, straightforward routing protocol: Dijkstra. You should let the switches route packets with minimum cost. Now, you should compute how to forward packets beforehand, make rules for all packets, and push rules to switches.
You should make following code changes:
In `init` function,

- Parse network structure
- Compute forwarding rules for all switches (you can do this in `addrule` function instead)

In `addrule` function,

- Compute forwarding rules for all switches (you can do this in `init` function instead)
- Push forwarding rules to switches

You can test your code by following commands in each terminal session. Again, launch **POX controller first**, and **Mininet later**.

`$ sudo pox/pox.py misc.controller`	`$ sudo ./test.py --task 3`

As described in Task 2, this command will initialize the Mininet network with POX controller with your implemented logic. You can test your logic with the `pingall` command, which should make 0% packet drop rate. You may use shell commands in Mininet CLI, with pre-defined variables: host and switch names. For example, `h1 ping h2` and `h1 tcpdump -w h1.pcap &`. You may get more information from the official document.

- `$ sudo ./test.py --task 3` will show `Unable to contact the remote controller at 127.0.0.1:6653`. Again, it is totally fine.
- Your switch should forward one packet to one output port.
- You may want to edit `graph.py`, managing construction of network structure
- You cannot use flood rules here. If you simply flood messages to all ports (i.e., no meaningful routing in practice) you will surely get 0 points even if `pingall` yields a 0% drop rate.
- Remember that you need to apply Dijkstra to **both** ARP and IPv4 packets. We will test both when grading.

Task 4: Implementing a Simple DNS-based Censorship

In this task, you will implement a simple DNS-based censorship. You should let the switches send DNS packets to the controller, and the controller manually checks if the DNS requests are asking for a censored domain.

For Task 4, you will use a very simple network structure:

- One controller: c1
- One switch: s1
- Several hosts:
    - two hosts with HTTP server: cs341dangerous.com and cs341safe.com
    - one host with DNS server
    - the other hosts

You should make following code changes:
In `addrule` function,

- Create proper forwarding rules, which configures switches to send all DNS queries and responses to the controller
    - Note that HTTP traffic should not be forwarded to the controller.

In `handlePacket` function,

- Check if the unhandled packet is a DNS request, containing a query of `cs341dangerous.com`
    - For such a DNS query, drop the query. Then create a DNS response packet that informs the client of non-existence of such a domain and send it back to the client.
    - All other DNS packets for non-censored domains should be served properly.

You can test your code by following commands in each terminal session.

`$ sudo pox/pox.py misc.controller`	`$ sudo ./test.py --task 4`

`$ sudo ./test.py --task 4` might take longer than previous tasks, since it waits for `h1` to launch an HTTP server, and `h2` to launch a DNS server.

If implemented correctly, you can test via following:

```
$ sudo ./test.py --task 4
[sudo] password for <username>:
Unable to contact the remote controller at 127.0.0.1:6653
running cs341dangerous.com on h5(10.0.0.5)
running cs341safe.com on h4(10.0.0.4)
running DNS on h1(10.0.0.1)
You can test via following commands:
h6 dig @10.0.0.1 cs341safe.com
h6 dig @10.0.0.1 cs341dangerous.com
h6 curl -H "Host: cs341safe.com" -m 10 http://10.0.0.4/
h6 curl -H "Host: cs341dangerous.com" -m 10 http://10.0.0.5/
mininet> h6 dig @10.0.0.1 cs341safe.com

; <<>> DiG 9.16.1-Ubuntu <<>> @10.0.0.1 cs341safe.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50051
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;cs341safe.com.                 IN      A

;; ANSWER SECTION:
cs341safe.com.          3600    IN      A       10.0.0.4

;; Query time: 3 msec
;; SERVER: 10.0.0.1#53(10.0.0.1)
;; WHEN: 토 10월 28 17:26:45 KST 2023
;; MSG SIZE  rcvd: 60

mininet> h6 dig @10.0.0.1 cs341dangerous.com
```

```
; <<>> DiG 9.16.1-Ubuntu <<>> @10.0.0.1 cs341dangerous.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24160
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;cs341dangerous.com.            IN      A

;; Query time: 47 msec
;; SERVER: 10.0.0.1#53(10.0.0.1)
;; WHEN: 토 10월 28 17:26:50 KST 2023
;; MSG SIZE  rcvd: 36
```

- You can check if services are correctly running `h1 netstat -tulpn` or `h2 netstat -tulpn`: each should show that port 80 listening, and port 53 listening
- You can assume that DNS servers and HTTP servers always use their default port
- For creating DNS response, we recommend to use `scapy` module

Task 5: Implementing a More Efficient DNS-based Censorship

In this task, you will implement a slightly improved DNS-based censorship.

You will use the simple network structure used for Task 4. That is,

- One controller: c1
- One switch: s1
- Several hosts:
  - two hosts with HTTP server: cs341dangerous.com and cs341safe.com
  - one host with DNS server
  - the other hosts

You should make following code changes:
In `addrule` function,

- Compute proper forwarding rules, which makes switches to send only DNS queries to the controller.
  - Note that DNS responses and HTTP traffic should not be forwarded to the controller.

In `handlePacket` function,

- Check if the unhandled packet is a DNS query, containing a query of `cs341dangerous.com`
  - For such a query, the controller configures the switch to forward the corresponding DNS response to the controller so that the controller can read the IP address of the queried (censored) domain.

- Then, the controller sends the DNS query (with the censored domain) to the intended DNS server.
- When the corresponding DNS response arrives at the controller (via the switch), the controller then learns the IP address of the queried domain and configures the switch to drop all packets to/from the IP address.
- The corresponding response is then forwarded to the client.

This way, the network operator can minimize the number of packets to be forwarded to the controller while still being able to censor the queried domain.

You can test your code by following commands in each terminal session.

`$ sudo pox/pox.py misc.controller`	`$ sudo ./test.py --task 5`

`$ sudo ./test.py --task 5` is actually the same with `$ sudo ./test.py --task 4`; there is no code difference.

If implemented correctly, you can test via following:

```
$ sudo ./test.py --task 5
[sudo] password for <username>:
Unable to contact the remote controller at 127.0.0.1:6653
running cs341dangerous.com on h5(10.0.0.5)
running cs341safe.com on h4(10.0.0.4)
running DNS on h1(10.0.0.1)
You can test via following commands:
h6 dig @10.0.0.1 cs341safe.com
h6 dig @10.0.0.1 cs341dangerous.com
h6 curl -H "Host: cs341safe.com" -m 10 http://10.0.0.4/
h6 curl -H "Host: cs341dangerous.com" -m 10 http://10.0.0.5/
mininet> h6 dig @10.0.0.1 cs341safe.com

; <<>> DiG 9.16.1-Ubuntu <<>> @10.0.0.1 cs341safe.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50051
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;cs341safe.com.                 IN      A

;; ANSWER SECTION:
cs341safe.com.          3600    IN      A       10.0.0.4

;; Query time: 3 msec
;; SERVER: 10.0.0.1#53(10.0.0.1)
;; WHEN: 토 10월 28 17:26:45 KST 2023
;; MSG SIZE  rcvd: 60
```

```
mininet> h6 dig @10.0.0.1 cs341dangerous.com

; <<>> DiG 9.16.1-Ubuntu <<>> @10.0.0.1 cs341dangerous.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24160
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;cs341dangerous.com.            IN      A

;; Query time: 47 msec
;; SERVER: 10.0.0.1#53(10.0.0.1)
;; WHEN: 토 10월 28 17:26:50 KST 2023
;; MSG SIZE  rcvd: 36

mininet> h6 curl -H "Host: cs341safe.com" -m 10 http://10.0.0.4/
Hello 10.0.0.6, I am cs341safe.com
mininet> h6 curl -H "Host: cs341dangerous.com" -m 10 http://10.0.0.5/
curl: (28) Connection timed out after 10171 milliseconds
```

Troubleshooting

On `vagrant up`, timed out while waiting for machine to boot error

When following messages show up:

```
cs341-23f-lab3-handout>vagrant up
Bringing machine 'mnvm' up with 'virtualbox' provider...
==> mnvm: Importing base box 'ubuntu/focal64'...
==> mnvm: Matching MAC address for NAT networking...
==> mnvm: Checking if box 'ubuntu/focal64' version '20230506.0.0' is up to date...
==> mnvm: Setting the name of the VM:
cs341-23f-lab3-handout_mnvm_1683656478422_26689
==> mnvm: Clearing any previously set network interfaces...
==> mnvm: Preparing network interfaces based on configuration...
    mnvm: Adapter 1: nat
==> mnvm: Forwarding ports...
    mnvm: 22 (guest) => 2222 (host) (adapter 1)
==> mnvm: Running 'pre-boot' VM customizations...
==> mnvm: Booting VM...
==> mnvm: Waiting for machine to boot. This may take a few minutes...
    mnvm: SSH address: 127.0.0.1:2222
    mnvm: SSH username: vagrant
    mnvm: SSH auth method: private key
Timed out while waiting for the machine to boot. This means that
Vagrant was unable to communicate with the guest machine within
the configured ("config.vm.boot_timeout" value) time period.
```

```
If you look above, you should be able to see the error(s) that
Vagrant had when attempting to connect to the machine. These errors
are usually good hints as to what may be wrong.

If you're using a custom box, make sure that networking is properly
working and you're able to connect to the machine. It is a common
problem that networking isn't setup properly in these boxes.
Verify that authentication configurations are also setup properly,
as well.

If the box appears to be booting properly, you may want to increase
the timeout ("config.vm.boot_timeout") value.
```

Open Virtualbox manually, and then open a newly created machine (named cs341-23f-lab3-handout-mnvm_xxxxxxxxxxxxx_xxxxx), and wait for its boot. If the VM screen shows the login page (showing `ubuntu-focal login:` text), try `vagrant up` again.
If the error persists, follow Environment Setup on Ubuntu 20.04 instead.

Address already in use error while executing POX controller

Error log similar to below shows up, failing to execute POX controller

```
INFO:core:POX 0.7.0 (gar) is up.
ERROR:openflow.of_01:Error 98 while binding 0.0.0.0:6633: Address already in use
ERROR:openflow.of_01: You may have another controller running.
ERROR:openflow.of_01: Use openflow.of_01 --port=<port> to run POX on another port.
```

Usually, killing already running POX controller solves problem:

```
vagrant@mnvm:~$ sudo pkill python3
```

If you want some python scripts not to be killed, you may check all listening ports, get process ID, and kill it manually.

If the problem still exists, try killing pox controller and ovs virtual switches:

```
vagrant@mnvm:~$ sudo pkill python3; sudo pkill ovs-vswit
```

and reload your machine (If you are not using vagrant, `reboot` your VM):

```
yourname@yourmachine: ~/yourpath$ vagrant reload
```

Error occurs while executing Mininet: `File exists` exception

Error log similar to below shows up, failing to execute mininet operation

```
RTNETLINK answers: File exists
Traceback (most recent call last):
```

```
  File "test_task2.py", line 28, in <module>
    net = Mininet(topo=t, controller=RemoteController)
  File "/usr/local/lib/python3.8/dist-packages/mininet/net.py", line 178, in
__init__
    self.build()
  File "/usr/local/lib/python3.8/dist-packages/mininet/net.py", line 508, in build
    self.buildFromTopo( self.topo )
  File "/usr/local/lib/python3.8/dist-packages/mininet/net.py", line 495, in
buildFromTopo
    self.addLink( **params )
  File "/usr/local/lib/python3.8/dist-packages/mininet/net.py", line 406, in
addLink
    link = cls( node1, node2, **options )
  File "/usr/local/lib/python3.8/dist-packages/mininet/link.py", line 456, in
__init__
    self.makeIntfPair( intfName1, intfName2, addr1, addr2,
  File "/usr/local/lib/python3.8/dist-packages/mininet/link.py", line 501, in
makeIntfPair
    return makeIntfPair( intfname1, intfname2, addr1, addr2, node1, node2,
  File "/usr/local/lib/python3.8/dist-packages/mininet/util.py", line 270, in
makeIntfPair
    raise Exception( "Error creating interface pair (%s,%s): %s " %
Exception: Error creating interface pair (h2-eth0,s4-eth2): RTNETLINK answers: File
exists
```

This happens because the previous mininet execution did not end correctly, leaving leftovers. Remove leftovers by following command:

```
$ sudo mn -c
```

`struct.error` while sending openflow message to switch

Error log similar to below shows up, failing to send openflow message to switches

```
ERROR:core:Exception while handling OpenFlowNexus!ConnectionUp...
Traceback (most recent call last):
  File "/home/vagrant/pox/pox/lib/revent/revent.py", line 242, in
raiseEventNoErrors
    return self.raiseEvent(event, *args, **kw)
  File "/home/vagrant/pox/pox/lib/revent/revent.py", line 295, in raiseEvent
    rv = event._invoke(handler, *args, **kw)
  File "/home/vagrant/pox/pox/lib/revent/revent.py", line 168, in _invoke
    return handler(self, *args, **kw)
  File "/home/vagrant/pox/pox/misc/controller.py", line 41, in connectionUp
    route.addrule(switchname, event.connection)
  File "/home/vagrant/route.py", line 87, in addrule
    connection.send(msg)
  File "/home/vagrant/pox/pox/openflow/of_01.py", line 875, in send
    data = data.pack()
  File "/home/vagrant/pox/pox/openflow/libopenflow_01.py", line 2349, in pack
    packed += i.pack()
  File "/home/vagrant/pox/pox/openflow/libopenflow_01.py", line 1586, in pack
```

```
    packed += struct.pack("!HHHH", self.type, len(self), self.port,
struct.error: required argument is not an integer
```

In this case, you are passing arguments with the wrong type while you are creating open flow messages. In the example above, the port number should be integer. If you pass port numbers with types other than int, the error above occurs. Refer to POX document for correct argument types.

Q&A

Can I use external libraries?

You can only use libraries that are already included in the project. You cannot install additional libraries, and I believe you do not need them.
If your code does not run due to missing libraries or other errors, you get 0 points.

Can we assume that every host is connected to only 1 switch?

Yes

Is every graph an undirected graph?

Yes

Are there any test cases?

No. You can test via changing the number of switches and hosts in `graph.py`.
For grading, we will increase:

- the number of switches and hosts for task 1,2,3
- the number of hosts for task 4,5

Does efficiency matter?

No, as long as your implementation runs in a reasonable time (exponential time not accepted) assume (# of switches) + (# of hosts) < 100

Useful Information

- If you have installed through vagrant, the files you should edit (/home/vagrant/*.py in the VM machine) are actually symbolic links of scripts in the synced folder. Modification of original file (/cs341-23f-lab3-handout/*.py in the host machine) is applied in the files in the VM, and vice versa.
- For Task 2 to Task 5, you can see forwarding of packets in the switches by tcpdump-ing network interfaces. While your Mininet is running, type ifconfig to see the interfaces of your switches; it will be displayed as `s1-eth1`, `s1-eth2`, `s2-eth1`, …

- For Lab1 to 3, random network structure is made at every test. You can find the network structure at `/tmp/net.json`

Submissions

Zip task_topology.py, task_controller.py into single zip file
Upload the zip file via gradescope.
Gradescope sometimes returns turnitin error. In this case, try submitting a few minutes later.