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

Logistics

- The due date is 11:59pm on November 18, 2024
- Submit a single zip file composed of `task_topology.py`, and `task_controller.py` via Gradescope.
- Discussion and Q&A: Lab 3 @ Classum

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 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 22.04/24.04 (Native, not WSL)**, 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 (note: **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 Classum.
For your information, we have confirmed that following environments work:

- Ubuntu 22.04/24.04 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/cs341-24f-lab3-handout.git
```

Setup VM environment by the following command:

```
$ cd cs341-24f-lab3-handout
$ vagrant up
```

This will show following messages:

```
cs341-24f-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:
cs341-24f-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
```

By default, `vagrant up` prints shell codes in red color.These **red code does not 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:

```
cs341-24f-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/cs341-24f-lab3-handout.git
```

Setup VM environment by following command:

```
$ cd cs341-24f-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
#

...

def init(self, net) -> None:
    ...
    ###
    # YOUR CODE HERE
    ###

def addrule(self, switchname: str, connection) -> None:
    ...
    ###
    # YOUR CODE HERE
    ###

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

def handlePacket(self, switchname, event, connection):
    ...
    ###
    # YOUR CODE HERE
    ###
```

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
  - For task 7, you might send rules to other switches as well

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 Mininet 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 message is printed in all tasks, but it is always fine.
- 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: Redirecting all DNS request packets to controller

From this task, you will implement DNS-based censorship on your network. For such behavior, you should let the switches send DNS packets to the controller, and the controller should check the content of the packets, and decide whether to return them to switches.
Since this task until task 6, you will use a very simple network structure (so-called a star network):

- One controller: c1
- One switch: s1
- Several hosts, where all of them is connected to the single switch:
    - one or more hosts with HTTP server:
        - some hosts may serve multiple domain name at once
    - one host with DNS server
    - the other hosts

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

- Compute forwarding rules for all switches (you can do this in `init` function instead)
    - For DNS request packets, send them to the controller
    - For other packets, route normally as Task 3.
- Push forwarding rules to switches

In `handlePacket` function,

- If you want, check if received packet is DNS packet

- Send received back to the switch, routing normally

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

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

This generates a star-shaped network, then opens mininet CLI. By default, the controller will print all received packets from switches.

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

If implemented correctly, you can proceed testing via following:

```
vagrant@mnvm:~$ sudo python3 test.py --task 4
Unable to contact the remote controller at 127.0.0.1:6653
running task4.com on h7(10.0.0.7)
running DNS on h6(10.0.0.6)
You can test via following commands:
h3 dig @10.0.0.6 task4.com
h3 curl --resolve task4.com:80:10.0.0.7 -m 10 http://task4.com/
mininet> h3 dig @10.0.0.6 task4.com

; <<>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <<>> @10.0.0.6 task4.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27028
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;task4.com.                     IN      A

;; ANSWER SECTION:
task4.com.              3600    IN      A       10.0.0.7

;; Query time: 0 msec
;; SERVER: 10.0.0.6#53(10.0.0.6) (UDP)
;; WHEN: Wed Nov 06 15:06:46 UTC 2024
;; MSG SIZE  rcvd: 52

mininet>
```

On the other session, you should see the DNS packet, where the default controller prints all packets received from switches.

```
vagrant@mnvm:~$ sudo pox/pox.py misc.controller
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
```

```
WARNING:libopenflow_01:Fields ignored due to unspecified prerequisites: tp_dst
[3e:54:12:f8:f5:5c>33:33:ff:f8:f5:5c IPV6][IPv6 ::>ff02::1:fff8:f55c
ICMP6][ICMP+TYPE_NEIGHBOR_SOLICITATION/0][ICMPv6/NDNeighborSolicitation num_opts:1
opts:[<NDP Option Type 14>] target:fe80::3c54:12ff:fef8:f55c]
[5a:c8:75:a2:7e:bb>33:33:00:00:00:16 IPV6][IPv6 ::>ff02::16
HOP_OPTS+ICMP6][ICMP+TYPE_MC_LISTENER_REPORT_V2/0][24 bytes: 00 00 00 01 04 ...]

... and many other unhandled IPv6 ICMP packets ...

[6a:62:e0:f9:c8:51>02:fc:e1:51:22:80 IP][IP+UDP 10.0.0.3>10.0.0.6 (cs:d400 v:4 hl:5
l:78 t:64)][UDP 53491>53 l:58 c:2e01](id:6994 fl:|RD | op:0 nq:1 na:0 nath:0
nadd:1)(q? task4.com A IN)(add:  #41 #1232 ttl:0 rdlen:12 datalen:12)

... and many other unhandled IPv6 ICMP packets ...
```

- You can see many IPv6 (ICMP6) packets from the POX controller. This is fine; we will only use IPv4, so we can ignore IPv6 packets.
- This task is not graded, but necessary for implementing the following tasks
- You can assume that all DNS only uses UDP with the known port for the service, and all packets using UDP and the port are DNS packets.

Task 5: 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.

We will use a simple network structure:

- One controller: c1
- One switch: s1
- Several hosts, where all of them is connected to the single switch:
    - one or more hosts with HTTP server:
        - some hosts may serve multiple domain name at once
    - 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 request, containing a query of `task5-block.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 5`

If implemented correctly, you can proceed via following:

```
vagrant@mnvm:~$ sudo ./test.py --task 5
Unable to contact the remote controller at 127.0.0.1:6653
running task5-open.com on h4(10.0.0.4)
running task5-block.com on h1(10.0.0.1)
running DNS on h5(10.0.0.5)
You can test via following commands:
h9 dig @10.0.0.5 task5-open.com
h9 curl --resolve task5-open.com:80:10.0.0.4 http://task5-open.com/
h9 curl --resolve task5-open.com:80:$(dig +short @10.0.0.5 task5-open.com | tail
-1) http://task5-open.com/
h9 dig @10.0.0.5 task5-block.com
h9 curl --resolve task5-block.com:80:10.0.0.1 http://task5-block.com/
h9 curl --resolve task5-block.com:80:$(dig +short @10.0.0.5 task5-block.com | tail
-1) http://task5-block.com/
You can also try running new server or dns by following commands:
<hostname> ./server.py <domain> &
<hostname> ./dns -a <domain1>=<IP1> -a <domain2>=<IP2> ... &
You can check background processes by following command:
<hostname> jobs
You can kill background process by following command:
<hostname> kill %<background job number>
mininet> h9 dig @10.0.0.5 task5-open.com

; <<>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <<>> @10.0.0.5 task5-open.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61177
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;task5-open.com.                        IN      A

;; ANSWER SECTION:
task5-open.com.         3600    IN      A       10.0.0.4

;; Query time: 3 msec
;; SERVER: 10.0.0.5#53(10.0.0.5) (UDP)
;; WHEN: Thu Nov 07 04:28:26 UTC 2024
;; MSG SIZE  rcvd: 62

mininet> h9 curl --resolve task5-open.com:80:10.0.0.4 http://task5-open.com/
```

```
Hello 10.0.0.9, I am task5-open.com
mininet> h9 curl --resolve task5-open.com:80:$(dig +short @10.0.0.5 task5-open.com
| tail -1) http://task5-open.com/
Hello 10.0.0.9, I am task5-open.com
mininet> h9 dig @10.0.0.5 task5-block.com

; <<>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <<>> @10.0.0.5 task5-block.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2638
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;task5-block.com.               IN      A

;; Query time: 7 msec
;; SERVER: 10.0.0.5#53(10.0.0.5) (UDP)
;; WHEN: Thu Nov 07 04:28:59 UTC 2024
;; MSG SIZE  rcvd: 33

mininet> h9 curl --resolve task5-block.com:80:10.0.0.1 http://task5-block.com/
Hello 10.0.0.9, I am task5-block.com
mininet> h9 curl --resolve task5-block.com:80:$(dig +short @10.0.0.5
task5-block.com | tail -1) http://task5-block.com/
curl: (6) Could not resolve host: task5-block.com
mininet>
```

As example above, your network should meet followings:

	task5-open.com	task5-block.com
DNS request	success, return correct IP	success, return empty response
HTTP request	success	success
HTTP request with DNS request	success	fail, cannot resolve domain

Here are some useful information:

- You can check if services are correctly running via `<hostname> netstat -tulpn`: HTTP server and DNS server should listening corresponding port
- You can assume that DNS servers and HTTP servers always use their default port
- For creating DNS response, we recommend to use `scapy` module
- For sending packet from controller, refer to [ofp_packet_out - Sending packets from the switch]
- You can assume that all DNS request packets has only one question query
- dns module in POX is buggy; use scapy for modifying/creating a packet
    - You can modify only dns part of POX using scapy, or create whole packet with scapy
    - We will not answer to questions about scapy how-to

Task 6: 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 5. That is,

- One controller: c1
- One switch: s1
- Several hosts, where all of them is connected to the single switch:
    - one or more hosts with HTTP server:
        - some hosts may serve multiple domain name at once
    - 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 `task6-block.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 HTTP packets to 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 6`

`$ sudo ./test.py --task 6` is almost the same with `$ sudo ./test.py --task 5`; there is no code difference in test.py.

If implemented correctly, you can test via following:

```
vagrant@mnvm:~$ sudo ./test.py --task 6
Unable to contact the remote controller at 127.0.0.1:6653
running task6-open.com on h3(10.0.0.3)
running task6-block.com on h8(10.0.0.8)
```

```
running DNS on h1(10.0.0.1)
You can test via following commands:
h4 dig @10.0.0.1 task6-open.com
h4 curl --resolve task6-open.com:80:10.0.0.3 http://task6-open.com/
h4 curl --resolve task6-open.com:80:$(dig +short @10.0.0.1 task6-open.com | tail
-1) http://task6-open.com/
h4 dig @10.0.0.1 task6-block.com
h4 curl --resolve task6-block.com:80:10.0.0.8 http://task6-block.com/
h4 curl --resolve task6-block.com:80:$(dig +short @10.0.0.1 task6-block.com | tail
-1) http://task6-block.com/
You can also try running new server or dns by following commands:
<hostname> ./server.py <domain> &
<hostname> ./dns -a <domain1>=<IP1> -a <domain2>=<IP2> ... &
You can check background processes by following command:
<hostname> jobs
You can kill background process by following command:
<hostname> kill %<background job number>
mininet> h4 curl --resolve task6-block.com:80:10.0.0.8 http://task6-block.com/
Hello 10.0.0.4, I am task6-block.com
mininet> h4 curl --resolve task6-block.com:80:$(dig +short @10.0.0.1
task6-block.com | tail -1) http://task6-block.com/
curl: (6) Could not resolve host: task6-block.com
mininet> h4 curl --resolve task6-block.com:80:10.0.0.8 http://task6-block.com/
^C
mininet>
```

From example above,

1. HTTP request to IP of task6-block.com should success on first
2. DNS request for task6-block.com should success, but result is empty
3. HTTP request with DNS resolve should fail, failing to resolve domain
4. HTTP request to IP of task6-block.com should now fail, HTTP request is also blocked

Task 7: Extending Censorship to a General Network Topology

In this task, you will extend Task 6 to work with a general network topology, which has multiple switches, and might contain internal loops.
Also, DNS records may change over time thus, you might need to unblock based on IP addresses for censorship.

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

`$ sudo pox/pox.py misc.controller`	`$ sudo ./test.py --task 7`

This constructs a random network topology, without any initial process.
You can manually turn on/off DNS and HTTP servers; for changing DNS records, restart DNS server with different arguments.

Followings will be tested while grading:

- At any time, HTTP and DNS servers can be changed by following:

        - Create new server, hosting either task7-block-<one or more digits>.com or task7-open-<one or more digits>.com
            - DNS server adds new record, HTTP server adds new domain
        - For certain domain, hosting server changes
            - DNS server changes record, HTTP server is replaced to another one
        - For certain domain, hosting stops
            - DNS server removes record, HTTP server removes the domain
        - For 3 changes above, HTTP servers and DNS servers are changed instantly
    - Assume that
        - single IP might host multiple domains
            - the IP should be blocked if it hosts at least one task7-block-<one or more digits>.com
        - Only one IP is assigned to one domain
            - If you detect different DNS response for same DNS request, assume that previous IP does not host the domain anymore

Your controller should do followings:

- Inspect all DNS request and DNS response
    - If DNS request is for task7-block-<one or more digits>.com, inspect DNS response as well, and maintain domain-IP matching list of task7-block-<one or more digits>.com
- On change of blockdomain-IP list, update blocking rules to all switches
    - if there is new IP to be blocked, push new rule
    - If there is an IP to be unblocked, modify or remove existing rule

You can test by following:

```
vagrant@mnvm:~$ sudo ./test.py --task 7
Unable to contact the remote controller at 127.0.0.1:6653
You can also try running new server or dns by following commands:
<hostname> ./server.py <domain> &
<hostname> ./dns -a <domain1>=<IP1> -a <domain2>=<IP2> ... &
You can check background processes by following command:
<hostname> jobs
You can kill background process by following command:
<hostname> kill %<background job number>
mininet> h1 ./dns -a task7-block-0.com=10.0.0.2 &
Listening on port 53.
mininet> h2 ./server.py task7-block-0.com &
mininet> h3 curl --resolve task7-block-0.com:80:10.0.0.2 http://task7-block-0.com/
Hello 10.0.0.3, I am task7-block-0.com
mininet> h3 dig @10.0.0.1 task7-block-0.com

; <<>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <<>> @10.0.0.1 task7-block-0.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56278
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
```

```
;; QUESTION SECTION:
;task7-block-0.com.             IN      A

;; Query time: 112 msec
;; SERVER: 10.0.0.1#53(10.0.0.1) (UDP)
;; WHEN: Thu Nov 07 07:52:43 UTC 2024
;; MSG SIZE  rcvd: 35

mininet> h3 curl --resolve task7-block-0.com:80:10.0.0.2 http://task7-block-0.com/
^C
mininet> h5 curl --resolve task7-block-0.com:80:10.0.0.2 http://task7-block-0.com/
^C
mininet> h1 jobs
QUERY { ID: dbd6. FIELDS: [ QR: 0, OpCode: 0 ], QDcount: 1, ANcount: 0, NScount: 0,
ARcount: 1,
  Question { qName 'task7-block-0.com', qType 1, qClass 1 }
}
Query for 'task7-block-0.com'
QUERY { ID: dbd6. FIELDS: [ QR: 1, OpCode: 0 ], QDcount: 1, ANcount: 1, NScount: 0,
ARcount: 0,
  Question { qName 'task7-block-0.com', qType 1, qClass 1 }
  ResourceRecord { name 'task7-block-0.com', type 1, class 1, ttl 3600, rd_length
4, Address Resource Record { address 10.0.0.2 }}
}
[1]+  Running                 ./dns -a task7-block-0.com=10.0.0.2 &
mininet> h1 kill %1
mininet> h2 jobs
Usage: python3 server.py <hostname1> <hostname2> ...
10.0.0.3 - - [07/Nov/2024 07:52:40] "GET / HTTP/1.1" 200 -
b'Hello 10.0.0.3, I am not task7-block-0.com\n'
[1]+  Running                 ./server.py task7-block-0.com &
mininet> h2 kill %1
mininet>
```

The example above is similar to Task 6 with two exceptions:

- You should manually startup DNS and HTTP servers; and
- You can kill servers for changing IPs for same domain

The example does not test changing domains, assigning multiple domains per single IP, removing domain, etc. Students should be responsible for testing these cases with their own test scripts.

Troubleshooting on development

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

When following messages show up:

```
cs341-24f-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:
cs341-24f-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 cs341-24f-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.
```

Check if which process is occupying the port:

```
vagrant@mnvm:~$ sudo lsof -i -P -n | grep LISTEN
sshd      14616            root    3u  IPv4  57199      0t0  TCP *:22 (LISTEN)
sshd      14616            root    4u  IPv6  57210      0t0  TCP *:22 (LISTEN)
systemd-r 14808 systemd-resolve   13u  IPv4  58650      0t0  TCP 127.0.0.53:53
(LISTEN)
sshd      42402         vagrant   10u  IPv6  95055      0t0  TCP [::1]:6010
(LISTEN)
sshd      42402         vagrant   11u  IPv4  95056      0t0  TCP 127.0.0.1:6010
(LISTEN)
sshd      42489         vagrant   10u  IPv6  94039      0t0  TCP [::1]:6011
(LISTEN)
sshd      42489         vagrant   11u  IPv4  94040      0t0  TCP 127.0.0.1:6011
(LISTEN)
python3   42568            root    7u  IPv4  95214      0t0  TCP *:6633 (LISTEN)
```

If `python3` process is occupying the port, try kill python3 process with SIGINT

```
vagrant@mnvm:~$ sudo kill -SIGINT 42568 # the process id
```

If `ovs-vswit` process is occupying the port, try followings in order

1. Manually stop openvswitch service, then run POX controller
2. Manually start openvswitch service, then start minintet testing script

```
vagrant@mnvm:~$ sudo /etc/init.d/openvswitch-switch force-stop
vagrant@mnvm:~$ sudo pox/pox.py misc.controller
```

Now, from different shell, type followings:

```
vagrant@mnvm:~$ sudo /etc/init.d/openvswitch-switch start
vagrant@mnvm:~$ sudo python3 test.py --task <task number>
```

If the problem persists, 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.

Exception reading connection while using `OFPP_NONE` as an action

It seems our switches do not support `OFPP_NONE`.

https://github.com/noxrepo/pox/issues/171#issuecomment-800229215
You may want to tell switch that there is no action to do, instead of "there is one action, and the action is doing nothing"

Frequently Asked Questions for Lab

Can you provide cloud instances?

No. If you have trouble in setup, please contact TAs.

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. In other words, only scapy is allowed.
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 1,2,3,7
- the number of hosts for task 5,6

Does efficiency matter?

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

Q&A

First, try finding a solution from this document, official POX document, and from the Internet.
If you still cannot find a solution, do followings in order:

1. Write entire question
    a. Attach all information, so that other people can *reproduce* your situation
        i. This means literally all tiny details, including OS, version, and entire command you have used

   b. Do not make images of text/code. **Always copy-paste text/code**
   c. If you have GUI problem, attach screenshot of the erroneous GUI
2. Write one-line summary for title
   a. First, ask chatgpt to do this. It is really good at it.
   b. Avoid using “regarding to…”, “problem on …”. Usually, titles with such text lack enough information. Your entire question content should be able to inferred from your question title
3. Create a post at Classum
   a. set post type to “Question”
   b. set tag as “Lab 3”

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 (/cs341-24f-lab3-handout/*.py in the host machine) is applied in the files in the VM, and vice versa.
- For Task 2 to Task 7, 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 Lab2, 3 and 7, 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. If you open the archive file with a common archive manager, you should see two files in the root of the archive.
Upload the zip file via Gradescope. Gradescope should display `task_topology.py` and `task_controller.py` without preceding directory name. (For example, `<random directory>/task_topology.py` is not allowed)
Gradescope sometimes returns turnitin error. In this case, try submitting a few minutes later.