OMSCS/OMSCY GEORGIA TECH

SDN Firewall with POX

Spring 2023

Project TA's for Fall 2023

Jeffrey Randow (Project Lead)
Michael Teal
Manas Chakka

SDN Firewall with POX

Table of Contents

SDN Firewall with POX Project

In this project, you will use Software Defined Networking (SDN) principles to create a configurable firewall using an OpenFlow enabled Switch. The Software Defined Networking (OpenFlow) functionality allows you to programmatically control the flow of traffic on the network.

This project has three phrases as follows:

1. Mininet Tutorial – This phase is a brief overview of Mininet. There are no deliverables for this phase and may be skipped, especially if you completed the Optional Simulating Networks project (Project 0).
2. Wireshark Tutorial – This phase is a brief introduction to packet capture using Wireshark/tshark. You will examine the packet format for various traffic to learn of the different header values used in Phase 3. There is a deliverable of a simple packet capture file.
3. SDN Firewall – This phase involves completing code to build a simple traffic blocking firewall using OpenFlow with the POX Controller based on rules passed to it from a configuration file. In addition, you will create a set of rules to test the firewall implementation.

Part 0: Project References

You will find the following resources useful in completing this project. It is recommended that you review these resources before starting the project.

- IP Header Format - https://erg.abdn.ac.uk/users/gorry/course/inet-pages/ip-packet.html
- TCP Packet Header Format - https://en.wikipedia.org/wiki/Transmission_Control_Protocol
- UDP Packet Header Format - https://en.wikipedia.org/wiki/User_Datagram_Protocol
- The ICMP Protocol - https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol
- IP Protocols - https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers
- TCP and UDP Service and Port References - https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers
- Wireshark - https://www.wireshark.org/docs/wsug_html/
- CIDR Calculator - https://account.arin.net/public/cidrCalculator
- CIDR - https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing

Part 1: Files Layout

Unzip the SDNFirewall-Spring2023.zip file into your Virtual Machine. Do this by running the following command:

```
unzip SDNFirewall-Spring2023.zip
```

This will extract the files for this project into a directory named SDNFirewall at your current path (it is recommended that your use the mininet root directory to aid in troubleshooting (`cd ~`). The following files will be extracted:

- cleanup.sh – this file called by using following command line: **./cleanup.sh**
  This file will clean up the Mininet Environment and kill all zombie Python and POX processes.
- sdn-topology.py – this file creates the Mininet topology used in this assignment. This is like what you created in the Simulating Networks project. When evaluating your code against the ruleset specified in this project, do not change it. However, you are encouraged to make your own topologies (and rules) to test the firewall. Look at the start-topology.sh file to see how to start a different topology.
- ws-topology.py – this file is substantially like sdn-topology, but it does not call the POX Controller. You will use this during the wireshark exercise.
- setup-firewall.py – this file sets up the frameworks used in this project. **DO NOT MODIFY THIS FILE.** This file will create the appropriate POX framework and then integrates the rules implemented in sdn-firewall.py into the OpenFlow engine. It will also read in the values from the configure.pol file and validate that the entries are valid. If you make changes to this file, the autograder will likely have issues with your final code as the autograder uses the unaltered distribution version of this file.
- start-firewall.sh – this is the shell script that starts the firewall. This file must be started before the topology is started. It will copy files to the appropriate directory and then start the POX OpenFlow controller. This file is called by using following command line: **./start-firewall.sh**
- start-topology.sh – this is the shell script that starts the Mininet topology used in the assignment. All it does is call the sdn-topology.py file with superuser permissions. This file is called by using following command line: **./start-topology.sh**
- test-client.py – this is a python test client program used to test your firewall. This file is called using the following command line: **python test-client.py PROTO SERVERIP PORT SOURCEPORT** where PROTO is either T for TCP, U for UDP, or G for GRE, SERVERIP is the IP address of the server (destination), PORT is the destination port, and optionally SOURCEPORT allows you to configure the source port that you are using. Example: **python test-client.py T 10.0.1.1 80**
- test-server.py – this is a python test server program used to test your firewall. This file is called using the following command line: **python test-server.py PROTO SERVERIP PORT** where PROTO is either T for TCP, U for UDP, G for GRE, SERVERIP is the IP address of the server (the server you are running this script on), and PORT is the service port.
  Example: **python test-server.py T 10.0.1.1 80**
- test-suite – This is a student developed test script that was developed in 2021 that can be used to test your implementation AFTER YOU FINISH BOTH THE IMPLEMENTATION FILES. The test cases in the main folder will be used to evaluate your implementations for the first run. An alternate configuration and topology will also be used to evaluate your implementations. This will be similar to, but not identical to what is found in the extra sub-folder. See Appendix A for information on how to use the test suite.

Project Deliverables

- configure.pol - this file is where you will supply the configuration to the firewall that specifies the traffic that should either be blocked or allowed (override blocks). The format of this file will be specified later in this document. **This file is one of the deliverables that must be included in your ZIP submission to Canvas.**
- sdn-firewall.py –This file implements the firewall using POX and OpenFlow functions. It receives a copy of the contents of the configure.pol file as a python list containing a dictionary for each rule and you will need to implement the code necessary to process these items into POX policies to create the firewall. **This file is one of the deliverables that must be included in your ZIP submission to Canvas.**

- packetcapture.pcap – This will be the packet capture completed in Part 4.

Part 2: Before You Begin

This project assumes basic knowledge about IP and TCP/UDP Protocols. It is highly encouraged that you review the following items before starting. This will help you in understanding the contents of IP packet headers and what you may need to match.

- What is the IP (Internet Protocol)? What are the different types of Network Layer protocols?
- Review TCP and UDP? How does TCP or UDP differ from IP?
- Examine the packet header for a generic IP protocol entry. Contrast that with the packet header for a TCP packet, and for a UDP packet. What are the differences? What does each field mean?
- What constitutes a TCP Connection? How does this contrast with a UDP connection.
- A special IP protocol is ICMP. Why is ICMP important? What behavior happens when you do an ICMP Ping? If you block an ICMP response, what would you expect to see?
- If you block a host from ICMP, will you be able to send TCP/UDP traffic to it?
- Can you explain what happens if you get a ICMP Destination Unreachable response?
- What is CIDR notation? How do you subnet a network?
- What IP Protocols use Source or Destination Ports?

Part 3: Review of Mininet

IF YOU HAVE FAMILIARITY WITH MININET OR IF YOU COMPLETED THE OPTIONAL PROJECT SIMULATING NETWORKS, YOU MAY SKIP THIS SECTION AND START WITH PART 4: WIRESHARK

Mininet is a network simulator that allows you to explore SDN techniques by allowing you to create a network topology including virtual switches, links, hosts/nodes, and controllers. It will also allow you to set the parameters for each of these virtual devices and will allow you to simulate real-world applications on the different hosts/nodes.

The following code sets up a basic Mininet topology similar to what is used for this project:

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net  import Mininet
from mininet.node import CPULimitedHost, RemoteController
from mininet.util import custom
from mininet.link import TCLink
from mininet.cli  import CLI

class FirewallTopo(Topo):
    def __init__(self, cpu=.1, bw=10, delay=None, **params):
        super(FirewallTopo,self).__init__()

        # Host in link configuration
        hconfig = {'cpu': cpu}
        lconfig = {'bw': bw, 'delay': delay}

        # Create the firewall switch
```

```
s1 = self.addSwitch('s1')

hq1 = self.addHost('hq1',ip='10.0.0.1',mac='00:00:00:00:00:1e', **hconfig)
self.addLink(s1,hq1)


us1 = self.addHost( 'us1', ip='10.0.1.1', mac='00:00:00:01:00:1e', **hconfig)
self.addLink(s1,us1)
```

This code defines the following virtual objects:

- Switch s1 – this is a single virtual switch with the label 's1'. In Mininet, you may have as many virtual ports as you need – for Mininet, "ports" are considered to be a virtual ethernet jack, not an application port that you would use in building your firewall.
- Hosts hq1 and us1 – these are individual virtual hosts that you can access via xterm and other means. You can define the IP Address, MAC/Hardware Addresses, and configuration parameters that can define cpu speed and other parameters using the hconfig dictionary.
- Links between s1 and hq1 and s1 and us1 – consider these like an ethernet cable that you would run between a computer and the switch port. You can define individual port numbers on each side (i.e., port on the host and port on the virtual switch), but it is advised to let Mininet automatically wire the network. Like hosts, you can define configuration parameters to set link speed, bandwidth, and latency. **REMINDER – PORTS MENTIONED IN MININET TOPOLOGIES ARE WIRING PORTS ON THE VIRTUAL SWITCH, NOT APPLICATION PORT NUMBERS.**

Useful Mininet Commands:

- For this project, you can start Mininet and load the firewall topology by running the **./start-topology.sh** from the project directory. You can quit Mininet by typing in the **exit** command.
- After you are done running Mininet, it is recommended that you cleanup Mininet. There are two ways of doing this. The first is to run the **sudo mn -c** command from the terminal and the second is to use the **./cleanup.sh** script provided in the project directory. Do this after every run to minimize any problems that might hang or crash Mininet.
- You can use the xterm command to start an xterm window for one of the virtual hosts. This command is run from the mininet> prompt. For example, you can type in **us1 xterm &** to open a xterm window for the virtual host us1. The & causes the window to open and run in the background. In this project, you will run the test-*-client.py and test-*-server.py in each host to test connectivity.
- The **pingall** command run from the mininet> prompt will cause all hosts to ping all other hosts. Note that this may take a long time. To run a ping between two hosts, you can specify **host1 ping host2** (for example, **us1 ping hq1** which will show the result of host us1 pinging hq1).
- The **help** command will show all Mininet commands and **dump** will show information about all hosts in the topology.

Part 4: Wireshark

Wireshark is a network packet capture program that will allow you to capture a stream of network packets and examine them. Wireshark is used extensively to troubleshoot computer networks and in the field of information

security. We will be using Wireshark to examine the packet headers to learn how to use this information to match traffic that will be affected by the firewall we are constructing.

tshark is a command line version of Wireshark that we will be using to capture the packets between mininet hosts and we will use Wireshark for the GUI to examine these packets. However, you will be allowed to use the Wireshark GUI if you would like in doing the packet capture.

Please watch the video referenced in Part 2 if you would like to follow along in time for a live packet capture.

- Step 1: Open up a Terminal Window and change directory to the SDNFirewall directory that was extracted in Part 1.

- Step 2: The first action is to startup the Mininet topology used for the wireshark capture exercise. This topology matches the topology that you will be using when creating and testing your firewall. To start this topology, run the following command:

  sudo python ws-topology.py

  This will startup a Mininet session with all hosts created.

- Step 3: Start up two xterm windows for hosts us1 and us2. After you start each xterm window, it is recommended that you run the following command in each xterm window as you load them to avoid confusion about which xterm belongs to which host:

  export PS1="hostname >" replacing hostname with the actual hostname.

  Type in the following commands at the Mininet prompt.

  us1 xterm & (then run **export PS1="us1 >"** in the xterm window that pops up)
  us2 xterm & (likewise, run **export PS1="us2 >"** in the second xterm window)

- Step 4: In this step, we want to start capturing all the traffic that traverses through the ethernet port on host us1. We do this by running tshark (or alternatively, wireshark) as follows from the mininet prompt:

  us1 sudo tshark -w /tmp/packetcapture.pcap

  This will start tshark and will output a pcap formatted file to /tmp/capture.pcap. Note that this file is created as root, so you will need to change ownership to mininet to use it in future steps – **chown mininet:mininet /tmp/packetcapture.pcap**

  If you wish to use the Wireshark GUI instead of tshark, you would call **us1 sudo wireshark &**. You may use this method, but the TA staff will not provide support for any issues that may occur.

- Step 5: Now we need to capture some traffic. Do the following tasks in the appropriate xterm window:

  in us1 xterm: **ping 10.0.1.2** (hit control C after a few ping requests)

In us2 xterm: **ping 10.0.1.1** (likewise hit control C after a few ping requests)
In us1 xterm: **python test-server.py T 10.0.1.1 80**
In us2 xterm: **python test-client.py T 10.0.1.1 80**
After the connection completes, in the us1 xterm, press Control-C to kill theserver.
In us1 xterm: **python test-server.py U 10.0.1.1 8000**
In us2 xterm: **python test-client.py U 10.0.1.1 8000**
In us1 xterm: press Control C to kill the server
In Mininet Terminal: press Control C to stop tshark

- Step 6: At the mininet prompt, type in exit and press enter. Next, do the chown command as described in step 4 above to your packet capture. You may also close the two xterm windows as they are finished. Copy the /tmp/packetcapture.pcap to your project directory. This file is the deliverable for this phase of the project.

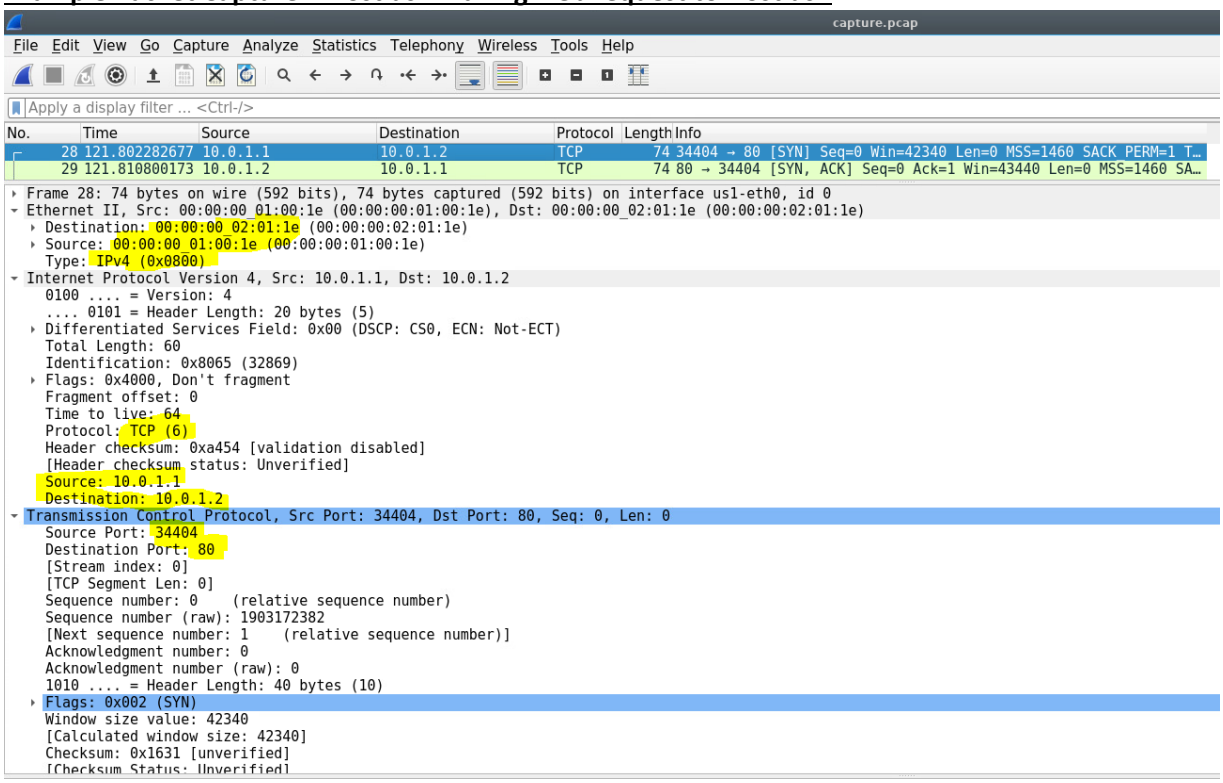
- Step 7: At the bash prompt on the main terminal, run:

  sudo wireshark

  Go to the File => Open menu item, browse to the /tmp directory and select the pcap file that you saved using tshark.

You will get a GUI that looks like the example packet capture. You will have a numbered list of all the captured packets with brief information consisting of source/destination, IP protocol, and a description of the packet. You can click on an individual packet and will get full details including the Layer 2 and Layer 3 packet headers, TCP/UDP/ICMP parameters for packets using those IP protocols, and the data contained in the packet.

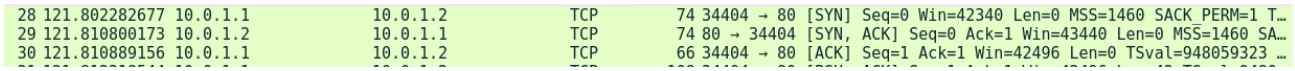
Example Packet Capture – Host us1 making web request to Host us2



Note the highlighted fields. You will be using the information from these fields to help build your firewall implementation and ruleset. Note the separate header information for TCP. This will also be the case for UDP packets.

Also, examine the three-way handshake that is used for TCP. What do you expect to find for UDP? ICMP?

Example TCP Three-Way Handshake



Please examine the other packets that were captured to help you familiarize yourself with Wireshark.

You can use the techniques shown here to help troubleshoot your firewall implementation

Part 5: SDN Firewall Implementation Details

Using the information that you learned above in running Wireshark, you will be creating two files – one is a firewall configuration file that will specify different header parameters to match in order to allow or block certain traffic and the second is the implementation code to create OpenFlow Flow Modification objects that will create the firewall using the parameters given in the firewall configuration file.

You may create temporary rulesets to help you complete Part 5b below.

Part 5a: Specifications of configure.pol

The configure.pol file is used by the firewall implementation code to specify the rules that the firewall will use to govern a connection. You do not need to code this first, but the format of the file is important as your implementation code will need to use these items. The format of the file is a collection of lines that have the proper format:

Rule Number, Action, Source MAC, Destination MAC, Source IP, Destination IP, Protocol, Source Port, Destination Port, Comment/Note

- **Rule Number** = this is a rule number to help you track a particular rule - it is not used in the firewall implementation. It can be of any value and is NOT validated in setup-firewall.py. The value need not be unique and can be numeric or text.

- **Action** = **Block** or **Allow** **Block** rules will block traffic that matches the remaining parameters of this rule. **Allow** rules will override Block rules to allow specific traffic to pass through the firewall (see below for an example). The entry is a string in (Block,Allow).

- **Source / Destination MAC** address in form of xx:xx:xx:xx:xx:xx (example: 00:a2:c4:3f:11:09) or a "-" if you are not matching this item. You may use MAC Addresses to match an individual host. In the real world, you would use it to match a particular piece of hardware. The MAC address of a particular host is defined inside the sdn-topology.py file.

- **Source / Destination IP Network** Address in form of xxx.xxx.xxx.xxx/xx in CIDR notation or a "-" if you are not matching this item.. You can use this to match either a single IP Address (using it's IP address and a subnet mask of /32, or a particular Subnet. An entry here would look like: 10.0.0.1/32. **NOTE: If you are using a CIDR mask other than /32 (individual host), make sure that the IP Address shown is the Network Address.**

  PRIMER ABOUT CIDR NOTATION:
  An IP Address consists of 32 bits which contain both the network and the host addresses. These 32 bits are divided into 4 sections consisting of 8 bits. The subnet mask /24 defines how many bits of the IP Address define the network. For a /24 network, there are 24 bits defining the network and 8 bits that define the host. Thus, if you specify 192.168.10.0/24, the first 24 bits (the 192.168.10) define the network address, and the 0 specifies the host (255 hosts). The IP specified here must be the network address (for a /24, it must represent the first 24 bits). For the /32 address, the entire 32 bits is a network address and represents a single host.

  The IP address of a particular host is defined inside the sdn-topology.py file.

- **Protocol** = integer IP protocol number per IANA (0-254) or a "-" if you are not matching this item.. An example is ICMP is IP Protocol 1, TCP is IP Protocol 6, etc. This must be an integer.

- **Source / Destination Port** = if Protocol is TCP or UDP, this is the Application Port Number per IANA. For example, web traffic is generally TCP Port 80. Do not try to use port numbers to differentiate the

different elements of the ICMP protocol. If you are not matching this item or are using an IP Protocol other than TCP or UDP, this field should be a "-".

- **Comment/Note** = this is for your use in tracking rules.

Special Notes About Firewall Configurations:

- Any field not being used for a match should have a '-' character as its entry. A '-' means that the item is not being used for matching traffic. **It is valid for any rule element except for Action to have a '-'.** (i.e., a rule like: **1,Block,-,-,-,-,-,-,-,Block the world** is valid, but not a rule that will be tested). ***HINT: Do not use any item that has a "-" in its field as an element that you will match. If you pass a "-" to a field in a match rule, you will cause POX to crash and your firewall will not work.***

  A "-" is valid for ALL FIELDS except Action DO NOT PASS A "-" INTO ONE OF THE OPENFLOW MATCH VARIABLES OR YOUR CODE WILL CRASH..

- **When a rule states to block the world from accessing a particular host, this means that you are matching against all possible hosts which may include hosts that are not in your topology.** ***HINT: Think about how you would match arbitrary traffic from anywhere on the network. Don't overthink this. Also, due to restrictions placed on the implementation by POX, please do not use 0.0.0.0/0 as an address for "world". In a real-world situation, this address would be valid as addressing any host on the internet.***

- Note that a rule does not necessarily need a MAC or IP Address. Also, it is possible to have a rule that only has network addresses and no ports/protocols. **What won't ever be tested is using a src/dst port WITHOUT an IP Protocol**.

- What is the difference between source and destination? Source makes a request of the destination. For ports, you will most often use destination ports, **but make sure that your firewall implements both source and destination ports**. For IP and MAC addresses, you will use both most of the time.

- When should I use MAC vs IP Addresses? You will want to interchange them in this file to test the robustness of your implementation. It is valid to specify a Source MAC address and a Destination IP Address.

Example Rules (included in the project files:

1,Block,-,-,10.0.0.1/32,10.0.1.0/24,6,-,80,Block 10.0.0.1 host from accessing a web server on the 10.0.1.0/24 network
2,Allow,-,-,10.0.0.1/32,10.0.1.125/32,6,-,80,Allow 10.0.0.1 host to access a web server on 10.0.1.125 overriding rule

What do these rules do?

The first rule basically blocks host hq1 (IP Address 10.0.0.1/32) from accessing a web server on any host on the us network (the subnet 10.0.1.0/24 network). The web server is running on the TCP IP Protocol (6) and uses TCP Port 80.

The second rule overrides the initial rule to allow hq1 (IP Address 10.0.0.1/32) to access a web server running on us5 (IP Address 10.0.1.125/32)

By definition – from the sdn-topology.py file:

```
This class defines the Mininet Topology for the network used in this project. This
network consists of the following hosts/networks:

    Headquarters Network (hq1-hq5). Subnet 10.0.0.0/24

    US Network (us1-us5). Subnet 10.0.1.0/24

    India Network (in1-in5). Subnet 10.0.20.0/24

    China Network (cn1-cn5). Subnet 10.0.30.0/24

    UK Network (uk1-uk5). Subnet 10.0.40.0/24
```

In Part 6, you will be given a set of firewall conditions that you will need to create the configure.pol needed for your submission.

You may create temporary rulesets to help you complete Part 5b below.

Part 5b: Implementing the Firewall in Code

After reviewing the format of the configure.pol file, you will now code a generic implementation of a firewall that will use the values provided from the configuration file (passed to you as dictionary items). As it is provided, the firewall implementation code blocks no traffic. You must implement code that does the following:

- **Create a OpenFlow Flow Modification object**
- **Create a POX Packet Matching object that will integrate the elements from a single entry in the firewall configuration rule file (which is passed in the policy dictionary) to match the different IP and TCP/UDP headers if there is anything to match (i.e., no "-" should be passed to the match object, nor should None be passed to a match object if a "-" is provided).**
- **Create a POX Output Action, if needed, to specify what to do with the traffic.**

Please reference code examples in Appendix C, or you may refer to the POX API documentation (WARNING, this is long and the API is confusing).

You will need to rewrite the rule = None to reference your Flow Modification object.

Your code will go into a section that will repeat itself for every line in the firewall configuration file that is passed to it. The "rule" item that is added to the "rules" list is an OpenFlow Modification object. The process of injecting this rule into the POX controller is handled automatically for you in the start-firewall.py file.

TIP: If your implementation code segment is more than 25-30 lines, you are making it too difficult. The POX API can provide many features that are not used in this project. The Appendix provides all of the information that you will need to code the project.

Key Information:

- policies is a python list that contains one entry for each rule line contained in your configure.pol file. Each individual line of the configure.pol file is represented as a dictionary object named policy. This dictionary has the following keys:
  - policy['mac-src'] = Source MAC Address (00:00:00:00:00:00) or "-"
  - policy['mac-dst'] = Destination MAC Address (00:00:00:00:00:00)) or "-"
  - policy['ip-src'] = Source IP Address (10.0.1.1/32) in CIDR notation) or "-"
  - policy['ip-dst'] = Destination IP Address (10.0.1.1/32)) or "-"
  - policy['ipprotocol'] = IP Protocol (6 for TCP)) or "-"
  - policy['port-src'] = Source Port for TCP/UDP (12000)) or "-"
  - policy['port-dst'] = Destination Port for TCP/UDP (80)) or "-"
  - policy['rulenum'] = Rule Number (1)
  - policy['comment'] = Comment (Example Rule)
  - policy['action'] = Allow or Block

  Use these to match traffic. Please note that all fields are strings and may contain a '-' character. You may either use policy['ip-dst'] or the split policy['ip-dst-address']/[policy['ip-dst-subnet'] in your implementation (the split was requested by prior semesters), but realize that if you use the ip-dst-address and ip-dst-subnet, you will need to carefully check your implementation to ensure that it is blocking the addresses you intend to block.

- You will need to assume that all traffic is IPV4. It is acceptable to hard-code this item.

- **Do not hardcode a solution by attempting to build code that internalizes the rules from Part 6. Your code should be generic enough to handle any possible configuration.**

- Hints:
  - The difference between an Allow or a Block is dependent on an Action and the Priority.
  - You don't necessarily need an action. See Appendix C for a discussion of what happens to a packet after it is matched.
  - There should be two priorities – one for ALLOW and one for BLOCK. Separate them sufficiently to override any exact matching behavior that the POX controller implements). It is suggested one priority be 0 or 1 and the other one above 10000. The reasoning for this is discussed in Appendix C.

- Outputting extra print debug lines will not adversely impact the autograder.

Part 6: Configuration Rules

You will need to submit a **configure.pol** file to create policies that implement the following scenarios. You may implement your rules in any manner that you want, but it is recommended using this step as an opportunity to check your firewall code implementation. The purpose of these rules is to test your firewall and to help determine how traffic flows across the network (source vs destination, protocols, etc).

DO NOT block all traffic by default and only allow traffic specified. You will lose many points because the firewall is open by default and only blocks the traffic that is specified.

You work for GT-SDN Corporation that has offices in the US, China, India, and the UK, with a US headquarters that acts as the datacenter for the company. Your task is to implement a firewall that accomplishes the following goals:

- **Task 1:** On the headquarters network, you have two active DNS servers (using the newer DNS-over-TLS standard operating on TCP and UDP Port 853). hq1 provides DNS service to the public (the world) and hq2 provides a private DNS service that should be accessible only to the 5 corporate networks (i.e., the US, China, India, UK, and Headquarters network). (DNS-over-TLS is restricted to TCP and UDP Protocol 853 for the purpose of satisfying the ruleset for this project)

  Rule Objective: A connection from any host in the word should connect to hq1 on TCP and UDP 853. However, only hosts on the US, China, India, UK, and HQ networks should be able to connect to TCP and UDP 853 on host hq2. All other hosts should NOT be able to connect to the DNS Server TCP and UDP 853 on host hq2.

- **Task 2:** On the headquarters network, the host hq3 acts as a VPN server that connects to each of the other sites (hosts us3, uk3, in3, and cn3) using the OpenVPN server (standard ports – using both TCP and UDP Ports 1194 will satisfy the requirements for this rule). Create a set of firewall rules that will only allow the 4 offsite hosts (us3, uk3, in3, and cn3) access to the hq3 OpenVPN server. No other hosts in the world should be able to access the OpenVPN server on hq3.

  Rule Objective: Only hosts us3, uk3, in3, and cn3 should connect to TCP and UDP Port 1194 on host HQ3. No other host should be able to connect to TCP and UDP 1194 on hq3. Note that no other potential VPN servers on other servers that may exist will be impacted by this rule

- **Task 3:** Allow the hosts on the Headquarters network to be reachable via an ICMP ping from the world (including from the us, uk, in, and cn networks). However, the us, uk, in, and cn networks should not be reachable from the world (due to firewall implementation limitations, the hq network would be able to ping the us, uk, in, and cn network. Why? What changes could be made to the implementation requirements to allow this?)

Rule Objective: All hosts can receive a complete ping request/response to any HQ network computers. Any hosts attempting to ping the US, UK, IN, and CN networks should NOT get a complete ping request/response from these hosts EXCEPT for the HQ network. In order to satisfy the first part of this rule, the HQ network must be able to ping the US, UK, IN, and CN network.

- **Task 4:** One of the main routes for ransomware to enter a corporate network is through a remote desktop connection with either an insecure version of a server protocol or via leaked or weak credentials (using either the Microsoft Remote Desktop protocol or the Virtual Networking Computing (VNC) protocols as the remote desktop server). For this task, write a set of rules that will block the internet from connecting to a remote desktop server on the five corporate networks. Allow the uk, us, in, and cn to connect to a remote desktop server on the headquarters network. (Use TCP Port 3389 for Remote Desktop and TCP Port 5900 for VNC)

  Rule Objective: Block any hosts outside of the corporate network (HQ, US, UK, IN, and CN) from connecting to any hosts on corporate network on TCP ports 3389 and 5900). Computers on the corporate network CAN connect to TCP ports 3389 and 5900 on the headquarters network. Connections between the other corporate networks (cn1 to us1) are note defined and can be set as desirec.

- **Task 5:** The servers located on hosts us3 and us4 run a micro webservice on TCP Port 8510 that processes financial information. Access to this service should be blocked from hosts uk2, uk3, uk4, uk5, in4, in5, us5, and hq5. (Hint: Note the IP Addresses for these hosts and you may use the smallest subnet mask that handles the listed hosts using CIDR notation).

  Rule Objective: This rule is designed to test using different CIDR notations to bracket hosts together. Otherwise, the rule is to be interpreted as written. You do not need to use CIDR notation to combine hosts, but it will result in many additional rules.

TIPS:

- Note that you only need to use an ALLOW rule to override a BLOCK rule. So if no block rule is supplied, the ALLOW is not needed. This is important for Rule #1. You will definitely need to use ALLOW rules for Rule #3.
- Note that if a rule specifies that TCP Port 8510 should be blocked, it does not mean that UDP Port 8510 should be blocked. The autograder checks to ensure that connections are not being overblocked.

What to Turn In

You need to submit your copy of packetcapture.pcap, sdn-firewall.py and configure.pol from you project directory using the instructions from the Piazza Post “How to Submit / Zip Our Projects” (#7 for Fall 2022). To recap, zip up the two files using the following command, replacing gtlogin with your GT Login that you use to log

into Canvas:

zip gtlogin_sdn.zip packetcapture.pcap configure.pol sdn-firewall.py

The key to properly zipping the project is to NOT zip up the directory. ZIP only the files you are included. If you do not ZIP this properly, you may be assessed a 5 point penalty as your submission will not be autograded.

You may also include an additional text file if you have comments, criticisms, or suggestions for improvement for this project. If you wish to provide this information, add it to your ZIP file with the name comments.txt. This is completely optional.

Please check your submission after uploading. As usual, we do not accept resubmissions past the stated deadlines.

What you can and cannot share

Do not share the content of your sdn-firewall.py, configure.pol, or packetcapture.pcap with your fellow students, on Ed Discussions, or elsewhere publicly. You may share any new topologies, testing rulesets, or testing frameworks, as well as packet captures that do not address the requirements of Part 4b.

Rubric

For the Spring 2023 Semester, this project is worth a total of 100 points which is distributed in the following fashion:

- 5 points for submitting a version of sdn-firewall.py that indicates effort was done.
- 5 points for submitting a version of configure.pol that indicates effort was done.
- 15 points for submitting a version of packetcapture.pcap that indicates effort was done.
- 35 points - Your configure.pol file will be tested by a known good firewall implementation and by your sdn-firewall.py file with a series of unit tests to make sure that the rules were implemented. The higher of the two grades will be used (thus, you will not be penalized if your sdn-firewall.py file is not complete or has issues).
- 15 points – This is just a test of your provided sdn-firewall.py and configure.pol to ensure that your code is working (i.e., it tests your implementation, and not your ruleset)
- 25 points – The final portion of the grade consists of testing your sdn-firewall.py file with a different grading ruleset and/or topology to make sure that your code is robust enough to handle any valid configuration file.

Appendix A: How to Test Host Connectivity

Part A: How to Test Manually

When you are developing your implementation or troubleshooting a firewall rule, you will want to test by hand. Unfortunately this process is a bit difficult.

1,Block,-,-,10.0.0.1/32,10.0.1.0/24,6,-,80,Block 10.0.0.1 from accessing a web server on the 10.0.1.0/32 network

Startup Procedure:

- Step 1: Open two terminal windows or tabs on the VM and change to the SDNFirewall directory.
- Step 2: In the first terminal window, type in: **./start-firewall.sh configure.pol**

  If you get the following error, run chmod +x start-firewall.sh and chmod +x start-topology.sh

```
mininet@mininet:~/SDNFirewall/student-test-suite/extra$ ./start-firewall.sh configure.pol
bash: ./start-firewall.sh: Permission denied
```

  This should start up POX, read in your rules, and start up an OpenFlow Controller. You will see something like this in your terminal window:

```
mininet@mininet:~/SDNFirewall$ ./start-firewall.sh configure.pol
~/pox ~/SDNFirewall
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
Starting POX Instance
Starting date and time : 2021-02-08 01:09:59


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 1] connected
List of Policy Objects imported from configure.pol:
--------------------------------------------------
[{'rulenum': '1', 'action': 'Block', 'mac-src': '-', 'mac-dst': '-', 'ip-src': '10.0.0.1/32', 'ip-dst': '10.0.1.0/24', 'ippro
tocol': '6', 'port-src': '-', 'port-dst': '80', 'comment': 'Block 10.0.0.1 from accessing a web server on the 10.0.1.0/32 net
work'}, {'rulenum': '2', 'action': 'Allow', 'mac-src': '-', 'mac-dst': '-', 'ip-src': '10.0.0.1/32', 'ip-dst': '10.0.1.125/32
', 'ipprotocol': '6', 'port-src': '-', 'port-dst': '80', 'comment': 'Allow 10.0.0.1 to access a web server on 10.0.1.125 over
riding previous rule'}]
Added Rule  1 :  Block 10.0.0.1 from accessing a web server on the 10.0.1.0/32 network
Added Rule  2 :  Allow 10.0.0.1 to access a web server on 10.0.1.125 overriding previous rule
```

  TA Note: Note that you will not see the "List of Policy Objects imported from configure.pol" and the "Added Rule" lines until after you complete Step 3 below.

- Step 3: In the second terminal window, type in: **./start-topology.sh**

  This should start up mininet and load the topology. You should see the following:

```
mininet@mininet:~/SDNFirewall$ ./start-topology.sh
Starting Mininet Topology...
If you see a Unable to Contact Remote Controller, you have an error in your code...
Remember that you always use the Server IP Address when calling test scripts...
mininet>
```

  This will start the firewall and set the topology. You do not need to repeat Steps 1-3 unless you are

done testing, need to restart the firewall, or need to restart mininet. When you are done with testing all of the rules you intend to use, type in "quit" in the mininet window, close all of the extraneous xterm windows generated, and run the mininet cleanup script **./cleanup.sh**

How to test connectivity between two hosts:

- Step 1: To test the rule shown above, we want to use host us1 as server/destination and host hq1 as the client. The rule we are testing involves the hq1 host attempting to connect to the web server port (TCP Port 80) on host us1. At the mininet prompt, type in the following two commands on two different lines:

  hq1 xterm &
  us1 xterm &

  Two windows should have popped up. You can always identify which xterm is which by running the command: **ip address** from the bash shell. This will give you the IP address for the xterm window, which will then let you discover which xterm window belongs to which host.

- Step 2: In the xterm window for us1 (which is the destination host of the rule – remember that the destination is always the server), type in the following command:

  python test-server.py T 10.0.1.1 80

  This sets up the test server for us1 that will be listening on TCP port 80. The IP Address specified is always the IP address of the machine you are running it on. **If you attempt to start the test-server on a machine that does not have the IP address that is specified in the command, you will get the following error: OSError: [Errno 99] Cannot assign requested address.**

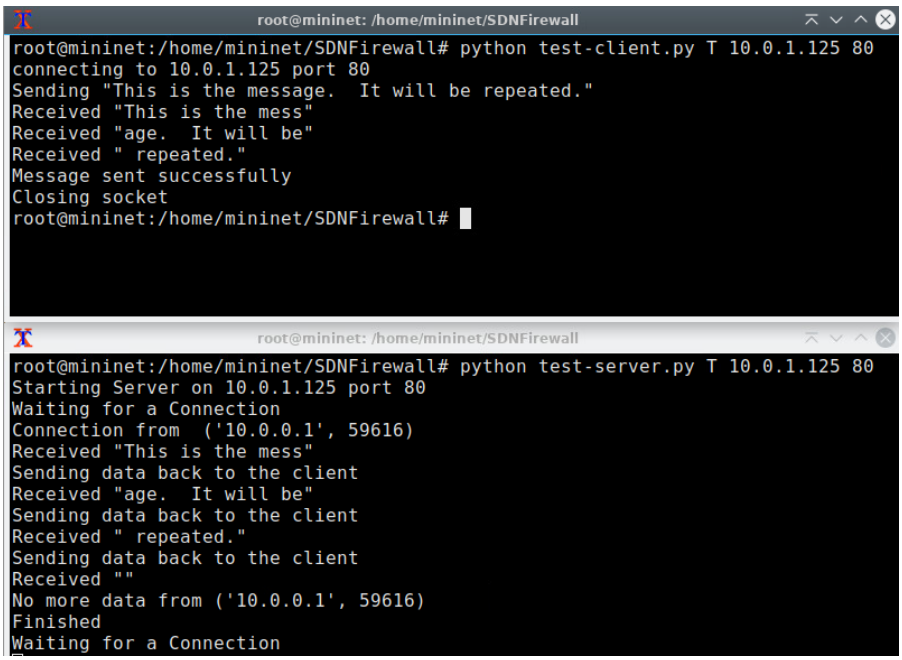
- Step 6: In the xterm window for hq1 (which is the source host of the rule – remember that the source is always the client), type in the following command:

  python test-client.py T 10.0.1.1 80

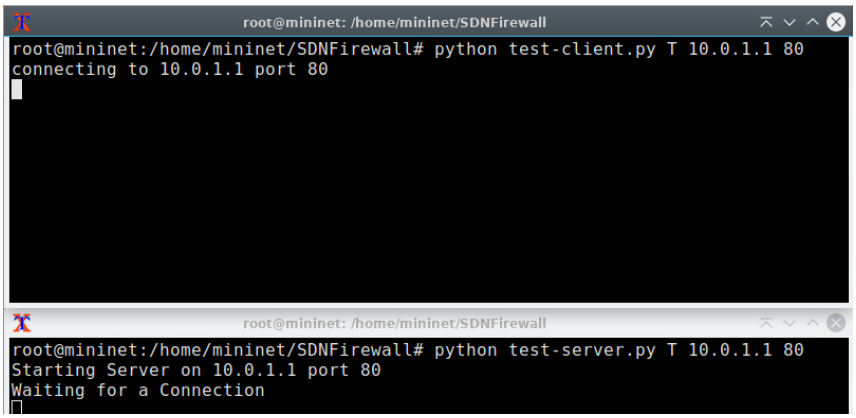
  This will start up a client that will connect to the TCP Port 80 on the server 10.0.1.1 (destination IP address) and will send a message string to the server. However, if the firewall is set to block this connection, you will never see the message pass on either of the client or the server.

Examples of Connection Status:

- The two windows below depict a successful un-blocked connection between the client and the server.

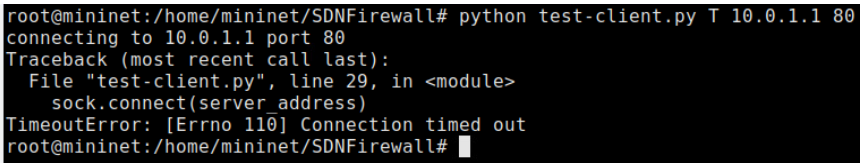


- A blocked connection will look like this (note that the client may take a while to timeout):



You may hit Control C to kill both the server and the client.

- A timed out connection is shown below. The difference between a timed-out connection on how the connection was blocked or if it was blocked on a different side of the connection.



- If you get an error that says “No route to destination”, you have blocked the routing protocol. Ensure that you do not have a Unspecified Prerequisite Error

Repeat this process for every rule you wish to test. If you feel that after some initial testing that your implementation and ruleset is good, you may then proceed to using the automated test suite.

Part B: Automated Testing Suite

The automated Test Suit was developed by a student in Spring 2021 (htian66) and has been updated to match the current version of this project. The Autograder works in a similar manner, but is instrumented differently to grade. The Autograder will use the exact same test-cases to grade your configure.pol and sdn-firewall.py files, but the alternate topology is similar, but not the same as the extra test cases.

How to test normal cases:

1. Change to the test-scripts directory
2. Copy your `sdn-firewall.py` and `configure.pol` into this directory.
3. Run `./start-firewall.sh configure.pol` as usual.
4. Open a new window, run `sudo python test_all.py`.
5. Total passed cases are calculated. Wrong cases will be displayed. For example, `2: us1 -> hq1 with U at 53, should be True, current False` means the connection from client us1 to host hq1 using UDP at hq1 53 port is failed, which should be successful. The first number is the index (0-based) of testcases.

Notes:

- One testcase in the `testcases.txt` file is given here: `us1 hq1 U 53 True` -> us1 client should be able to access hq1 host with TCP protocol at port 80. Please pull request to fill the `testcases.txt` file.
- For goal 3, `P` is used in `testcaeses.txt` to represent `ping`.
- `test-server.py` and `test-client.py` are slighted modified from the original version to support `GRE` protocol testing.

How to test extra cases:

1. Change to the test-scripts/extras directory
2. Copy your `sdn-firewall.py` into this directory (do NOT copy configure.pol as there are different rules and hosts being used).
3. Run `./start-firewall.sh configure.pol` as usual.
4. Open a new window, run `sudo python test_all.py`.
5. Total passed cases are calculated. Wrong cases will be displayed. For example, `2: us1 -> hq1 with U at 53, should be True, current False` means

Note that you can extend the extra test cases to test other scenarios like source ports and additional IP Protocols. Feel free to post the topology, configure.pol, and testcases.txt file for these additional scenarios.

Appendix B: Troubleshooting Information

General Coding Issues

- Watch for type mismatches.
- Do not run “pip3 install pox”. The pox module installed by pip is not the library used in this project.
- This project is virtually impossible to use the Debug utilities inside of VSCode or Pycharm since it requires running under the mininet framework. For debugging, it is suggested that you use print statements in your code to help determine where issues may be occurring.
- You do NOT need to reparse any of the data provided in the dictionary other than possibly changing the type from strings.
- If you use Visual Studio code, add the following to your workspace settings:
  "python.autoComplete.extraPaths": ["/home/mininet/pox/"],

  Also, with Visual Studio code, it sometimes “recommends” _dl_type and other names prepended with a _. Note that this is incorrect – the name is dl_type, not _dl_type.

Firewall Implementation (sdn-firewall.py) Errors and Issues

- Pay attention to the Fields ignored due to unspecified prerequisites warning. Do not ignore this message or your firewall will overblock (i.e., it ignores the field specified). Remember that there are items required when you specify TCP or UDP, and when you specify IPV4.
- If you get a struct.pack or struct.unpack error message, take a look at https://github.com/att/pox/blob/7f76c9e3c9bc999fcc97961d408ab0b71cbc186d/pox/OpenFlow/libOpenFlow_01.py for more information. Also, the struct.pack error might reference how to fix (i.e., not an integer, EthAddr(), etc).
- A struct.unpack error may be ignored if it occurs on your GRE rule.
- The following error means that you should check your output action: “TypeError: ord() expected string of length 1, but int found”
- If you find that all traffic is blocked, double check the unspecified prerequisite warning.
- Note that you may have issues with certain rules that don’t seem to work when you first start the firewall. This is typically caused by starting the Spanning Tree. You may run “pingall” to overcome this before testing.

Mininet/Topology Issues

- On the topology terminal window, if you get an error message that states “Unable to contact Remote Controller”, it means that the POX controller had crashed and normally shows that there is a bug in your implementation code. Look at the windows where you started the firewall.
- If you get the following error message, please run the cleanup.sh utility: "Exception: Error creating interface pair (s1-eth0,hq1-eth0): RTNETLINK answers: File exists”
- However if it is “Error:packet: (dns) parsing questions: ord() expected string of length 1, but int found”, this is a known issue with the POX implementation in Python 3 and can be ignored.

Appendix C: POX API Excerpt

This section contains a highly modified excerpt from the POX Manual (modified to remove extraneous features not used in this project and to provide clarifications). You should not need to use any other POX objects for this project. TA Comments are highlighted. Everything on these pages is important to complete the project.

Excerpted and modified from: https://noxrepo.github.io/pox-doc/html/

Object Definitions:

Flow Modification Object

The main object used for this project is a "Flow Modification" object. This adds a rule to the OpenFlow controller that will affect a modification to the traffic flow based on priority, packet characteristic matchin, and an action that will be done to the traffic that is matched. IF AN OBJECT is matched, it is pulled from the network stream and will only be forwarded, modified, or redirected if you do an action. If you do not specify an action and the packet is matched, the packet will basically be dropped.

```
class ofp_flow_mod (ofp_header):
  def __init__ (self, **kw):
    ofp_header.__init__(self)
    self.header_type = OFPT_FLOW_MOD
    self.match = ofp_match()
    self.priority = OFP_DEFAULT_PRIORITY
    self.actions = []
```

Match Structure

OpenFlow defines a match structure – ofp_match – which enables you to define a set of headers for packets to match against.

The match structure is defined in pox/OpenFlow/libOpenFlow_01.py in class ofp_match. Its attributes are derived from the members listed in the OpenFlow specification, so refer to that for more information, though they are summarized in the table below.

You should create a match object and attach it to the flow modification object.

Attribute	Meaning
in_port	The Physical or virtual port on the switch that the packet arrived from. This is not an application port – Think of this as the port in a switch where you plug in an ethernet cable. Do you know where this is in the defined network?

Attribute	Meaning
dl_src	Ethernet/MAC source address (Type of EthAddr)
dl_dst	Ethernet/MAC destination address (Type of EthAddr)
dl_type	Ethertype / length (e.g. 0x0800 = IPv4) (Type of Integer)
nw_proto	IP protocol (e.g., 6 = TCP) or lower 8 bits of ARP opcode (Type of integer)
nw_src	IP source address (Type of String or IPAddr)
nw_dst	IP destination address (Type of String or IPAddr)
tp_src	TCP/UDP source application port (Type of Integer)
tp_dst	TCP/UDP destination application port (Type of Integer)

TA Note: (IMPORTANT
If you use VSCode or Pycharm, it may make the recommendation to use _dl_src and _dl_dst. These are not valid. Please use what is specified above.

Attributes may be specified either on a match object or during its initialization. That is, the following are equivalent:

```
matchobj = of.ofp_match(tp_src = 5, dl_dst = EthAddr("01:02:03:04:05:06"))
#.. or ..
matchobj = of.ofp_match()
matchobj.tp_src = 5
matchobj.dl_dst = EthAddr("01:02:03:04:05:06")
```

IP Address Handling

IP addresses can be specified in many different ways. When crafting your rule, you can set a network address than can contain a subset of hosts on a particular network.

```
my_match.nw_src = "192.168.42.0/24"
my_match.nw_src = (IPAddr("192.168.42.0"), 24)
my_match.nw_src = "192.168.42.0/255.255.255.0"
```

IMPORTANT NOTE ABOUT IP ADDRESSES

TA Note: What isn't very clear by this documentation is that nw_* is expecting a network address. If you are calling out an IP Address like 10.0.1.1/32, it is an acceptable response to nw_*. However, if you are calling out a subnet like 10.0.1.0/24, the IP address portion of the response MUST BE the Network Address.

From Wikipedia: IP addresses are described as consisting of two groups of bits in the address: the most significant bits are the network prefix, which identifies a whole network or subnet, and the least significant set forms the host identifier, which specifies a particular interface of a host on that network.

This division is used as the basis of traffic routing between IP networks and for address allocation policies. (https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing)

Thus for a /24 network, the first 24 bits of the address comprises the network address. Thus, it would be 10.0.1.0. For a /25 network, there would be two networks in the 10.0.1.x space – a 10.0.1.0/25 and a 10.0.1.128/25.

Your implementation code does NOT need to convert the given IP Address into a network – you can assume that any given address in a possible configuration file must be valid. However, your configure.pol file MUST be using the proper form if you are using a CIDR notation other than /32. Why would you do this? To reduce the number of rules needed. You may use this for the 5th rule from Part 6.

Note that some fields have *prerequisites*. Basically this means that you can't specify higher-layer fields without specifying the corresponding lower-layer fields also. For example, you can not create a match on a TCP port without also specifying that you wish to match TCP traffic. And in order to match TCP traffic, you must specify that you wish to match IP traffic. Thus, a match with only tp_dst=80, for example, is invalid. You must also specify nw_proto=6 (TCP), and dl_type=0x800 (IPv4). If you violate this, you should get the warning message 'Fields ignored due to unspecified prerequisites'. For more information on this subject, see the FAQ entry "I tried to install a table entry but got a different one. Why?" as shown below:

FAQ Entry:

This question also presents itself as "What does the Fields ignored due to unspecified prerequisites warning mean?"

Basically this means that you specified some higher-layer field without specifying the corresponding lower-layer fields also. For example, you may have tried to create a match in which you specified only tp_dst=80, intending to capture HTTP traffic. You can't do this. To match TCP port 80, you must also specify that you intend to match TCP (nw_proto=6). And in order to match on the TCP protocol, you must also match on IPV4 type (dl_type=0x800).

OpenFlow Actions

The final aspect needed to fully implement a flow modification object is the action. With this, you specify what you want done to a port. This can include forwarding, dropping, duplicating and redirecting, or modify the header parameters. For the purposes of this project, we are only dealing with forwarding of match traffic to its destination. But please note that for a Software Defined Network system, you can do all sorts of actions including round robin server, DDOS blocking, and many other possible options.

Output

Forward packets out of a physical or virtual port. Physical ports are referenced to by their integral value, while virtual ports have symbolic names. Physical ports should have port numbers less than 0xFF00.

Structure definition:

```
class ofp_action_output (object):
  def __init__ (self, **kw):
    self.port = None # Purposely bad -- require specification
```

port (int) the output port for this packet. Value could be an actual physical switch port number or one of the following virtual ports expressed as constants:

- of.OFPP_IN_PORT - Send back out the physical switch port the packet was received on. Except possibly OFPP_NORMAL, *this is the only way to send a packet back out its incoming port. (This redirects a packet back to the sender)*
- of.OFPP_NORMAL - Process via normal L2/L3 legacy switch configuration (i.e., send traffic to its destination without modification) – See https://study-ccna.com/layer-3-switch/ for information on how normal L2/L3 legacy switches work.
- of.OFPP_FLOOD – output to all OpenFlow ports except the input port and those with flooding disabled via the OFPPC_NO_FLOOD port config bit (generally, this is done for STP) This is an option where the packet will be forwarded to all physical or virtual switch ports except the source. This is not an ideal situation as it causes excessive traffic and could be used to create a denial of service attack because of the packet amplification that it may cause. (See DNS amplification attack for an example of the ramifications of this.
- of.OFPP_ALL - output all OpenFlow ports except the in port. This is similar to OFPP_FLOOD, except it sends to ports that had flooding disable. This may cause even more excessive traffic than OFPP_FLOOD
- of.OFPP_CONTROLLER - Send to the OpenFlow controller. This would send a packet to the switch hardware itself, and not to a particular host system.

Think carefully about the definitions given above for output actions. **Remember that if you match a packet, no action (i.e., packet will be dropped) will be done unless you set an output action**.

Example: Sending a FlowMod Object

The following example describes how to create a flow modification object including matching a destination IP Address, IP Type, and Destination IP Port, and setting an action that would redirect the matching packet out to physical switch port number 4 (note that you generally DO NO KNOW what physical switch port to use.

```
rule = of.ofp_flow_mod()
rule.priority = 42
```

```
rule.match.dl_type = 0x800
rule.match.nw_dst = "192.168.101.101/32"
rule.match.tp_dst = 80
rule.actions.append(of.ofp_action_output(port = 4))
```

Flow Modification Objects work as thus:

1. Packet enters the system and is examined by the Flow Modification Objects (1 for each rule in your configuration ruleset)
2. The packet will then be examined to see if the different header items match the items specified for that particular rule.
3. If the packet matches all of the applicable items, it is pulled from the stream for you to program an action for it (forward it, readdress it, change it). If you don't do an action for it, the package is essentially dropped. If the packet does not match all of the applicable header items, it continues to the next Flow Modification rule to test it. If it isn't matched by any rules, it is passed on to the specific destination.

For this project, you are making a flow modification object and action while using a matching pattern that can match any or all of the different parameters of the header. Make your implementation generic.