Link to Mini Project: https://github.com/LordJatonyas/CWM-ProgNets/tree/main/firewall

Mini Project: Simple Firewall

The goal of this project firewall is to prove the capability of blocking websites. This is done by dropping packets at the reflector when the destination address matches a blacklisted website IP ADDRESS. As such, the mini project heavily relies on what's been covered in Assignment 3 (Reflector).

First, instead of relying on the MAC ADDRESS for decisions, we will look at the destination IP ADDRESS.

```
/*************************************************************************
*************  I N G R E S S   P R O C E S S I N G   *******************
*************************************************************************/

control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {

    action swap_mac_addresses() {
        macAddr_t tmp_mac;
        tmp_mac = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
        hdr.ethernet.srcAddr = tmp_mac;

        //send it back to the same port
        standard_metadata.egress_spec = standard_metadata.ingress_port;
    }

    action drop() {
        mark_to_drop(standard_metadata);
    }

    table dst_ip_drop {
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        actions = {
            swap_mac_addresses;
            drop;
            NoAction;
        }
        size = 65000;
        default_action = swap_mac_addresses();
    }

    apply {
        if (hdr.ethernet.isValid()) {
            if (hdr.ipv4.isValid()) {
                dst_ip_drop.apply();
            }
        }
    }
}
```

The P4 code section above forces packet drops when the destination IP ADDRESS matches any table entry. Another change is the size of the table. Here, I have changed it from 1024 to 65,000. This change was made to accommodate for the large number of IP ADDRESSes owned by YouTube, the target website to block.

YouTube IP Address Ranges

To support a large and growing network of web servers, YouTube owns a number of IP addresses in ranges called blocks.

These IP address blocks belong to YouTube:

- 199.223.232.0 - 199.223.239.255
- 207.223.160.0 - 207.223.175.255
- 208.65.152.0 - 208.65.155.255
- 208.117.224.0 - 208.117.255.255
- 209.85.128.0 - 209.85.255.255
- 216.58.192.0 - 216.58.223.255
- 216.239.32.0 - 216.239.63.255

Administrators who want to block access to YouTube from their network should block these IP address ranges if their router allows.

These IP ADDRESSes are compiled into a commands.txt via a Python script:

```
class YouTube_list():
    def __init__(self):
        self.youtube_ip = []

    def list_youtube_ip(self):
        self.youtube_ip = []
        for i in range(232, 240):
            for j in range(0, 256):
                self.youtube_ip.append(f'199.223.{i}.{j}')
        for i in range(160, 176):
            for j in range(0, 256):
                self.youtube_ip.append(f'207.223.{i}.{j}')
        for i in range(152, 156):
            for j in range(0, 256):
                self.youtube_ip.append(f'208.65.{i}.{j}')
        for i in range(224, 256):
            for j in range(0, 256):
                self.youtube_ip.append(f'208.117.{i}.{j}')
        for i in range(128, 256):
            for j in range(0, 256):
                self.youtube_ip.append(f'209.85.{i}.{j}')
        for i in range(192, 224):
            for j in range(0, 256):
                self.youtube_ip.append(f'216.58.{i}.{j}')
        for i in range(32,64):
            for j in range(0, 256):
                self.youtube_ip.append(f'216.239.{i}.{j}')
        return self.youtube_ip

if __name__ == "__main__":
    youtube_list = YouTube_list()
    youtube_ip = youtube_list.list_youtube_ip()
    with open('commands.txt', 'w') as f:
        for ip in youtube_ip:
            f.write(f'table_add MyIngress.dst_ip_drop MyIngress.drop {ip}/32 => ')
            f.write('\n')
    f.close()
```

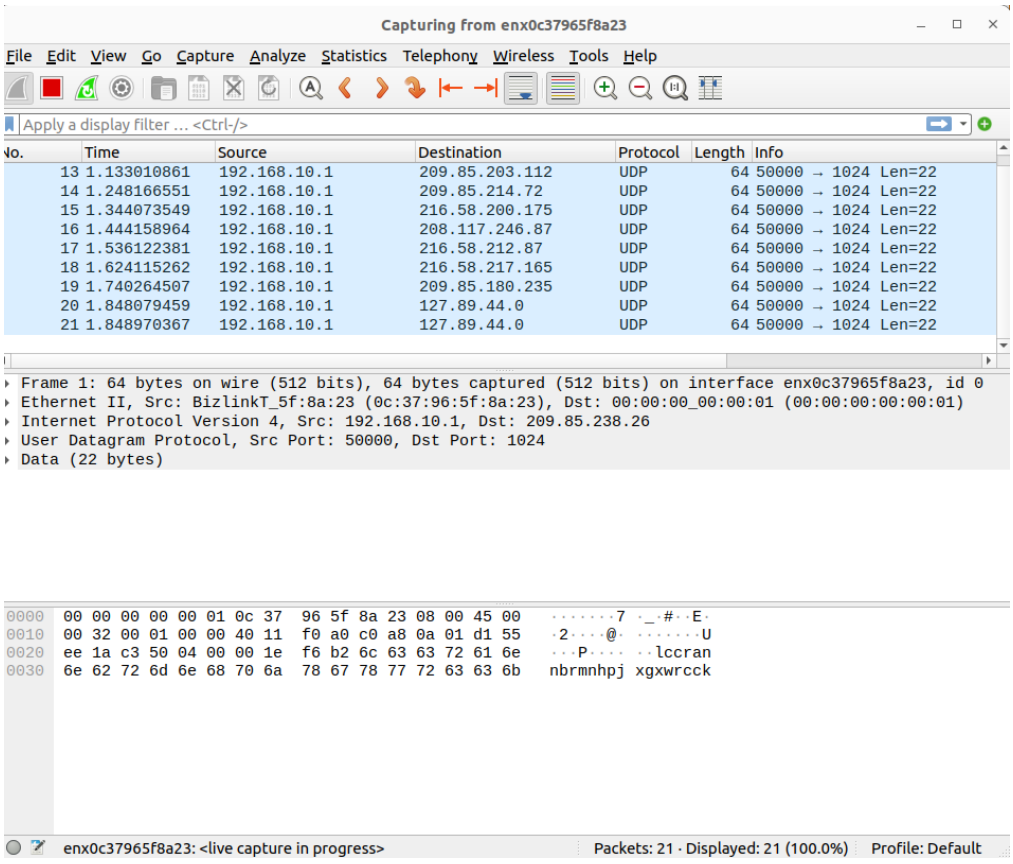
Looking back, this could have been improved by just matching the first 24 bits (since the last 8 bits are 0 to 255 anyway).

The compiled P4 program is then run as a switch on P4Pi-14, and commands.txt used to add tables to the simple switch. In total, there are 64511 entries.

The task is to test if certain destination IP ADDRESSes are affected. This is conducted via randomising destination IP ADDRESSes using the function below.

```
def random_ip():
    youtube_list = YouTube_list()
    ip_list = random.sample(youtube_list.list_youtube_ip(),50)
    ip_list.append("192.168.10.2")
    ip_list.append("8.8.8.8")
    ip_list.append("0.0.0.0")
    ip_list.append("127.89.44.0")
    return random.choice(ip_list)
```

It randomly picks 50 blacklisted destination IP ADDRESSes and puts them into an array. This array also contains 4 perfectly legal destination IP ADDRESSes, and the final line just randomly picks 1 destination IP ADDRESS from this array. The result of running this (using send.py) for 20 packets is shown below, with just 1 reflection and 19 dropped packets (1 x 2 + 19 = 21).



This matches our expectation since it is much more likely for the randomiser to select a blacklisted destination IP ADDRESS (50 vs 4), and when there was a legal destination IP ADDRESS, a reflection occurred.