

## **P4 destekli ağ anahtarlarının IP paketlerini işlerken kullanıcı tarafından belirlenecek IP adreslerini engelleyecek şekilde programlanması**

Önceki ödevde basit bir SDN anahtarı üzerinden openflow protokolü ile birlikte usom listesinin ip'lerini bir dosyaya tanımlayıp, bu dosya üzerindeki her bir adres için paketin drop edilmesi yönünde flow'lar eklemiştik. Böylelikle SDN anahtarına istediğimiz akışları ve işlemlerin beynini oluşturmuştuk.

Şimdi ise aynı durumu daha gelişmiş yapılar olan p4 destekli anahtarlar ile gerçekleştireceğiz. Bunun için p4 kodlarının olduğu repository'nin çekilmesi gerekmektedir.

```
p4@p4:~$ git clone https://github.com/p4lang/tutorials.git
```

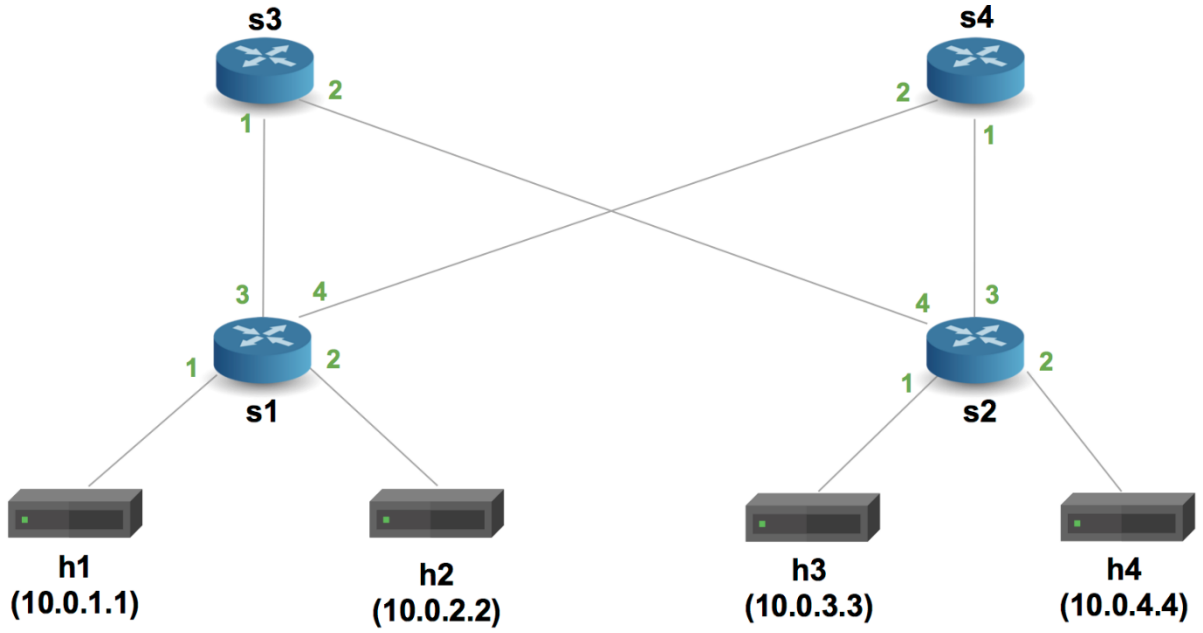
Gerekli repository çekildikten sonra kodu inceleyip yapılması gereken değişiklikler tespit edilir.

Kodu incelediğimizde basic.p4 kodunda MyIngress yapısında sdn'deki flowlara benzer olan aksiyonlar var. Kullanmamız gereken aksiyon drop olması gerekmektedir.

```
control MyIngress(inout headers hdr,  
                  inout metadata meta,  
                  inout standard_metadata_t standard_metadata) {  
    action drop() {  
        mark_to_drop(standard_metadata);  
    }  
}
```

4 anahtar ve 4 sunucu gerektiren gerekli linkleri içeren bir topolojiye ihtiyacımız vardır. Pod-topo topolojisi işimize yarıyor. Aynı zamanda Makefile'in çalıştırdığı topoloji dosyası pod-topo üzerinden çalışıyor. Bu bilgilere p4lang tutorials repository'si üzerindeki README dosyası üzerinden ulaşılabilir.

<https://github.com/p4lang/tutorials/blob/ccc56938075eefd5ea36c083cf53de9fb55c933a/exercises/basic/README.md>



```
Welcome Guide | basic.p4 | topology.json
{
  "hosts": {
    "h1": { "ip": "10.0.1.1/24", "mac": "08:00:00:00:01:11",
      "commands": ["route add default gw 10.0.1.10 dev eth0",
        "arp -i eth0 -s 10.0.1.10 08:00:00:00:01:00"]},
    "h2": { "ip": "10.0.2.2/24", "mac": "08:00:00:00:02:22",
      "commands": ["route add default gw 10.0.2.20 dev eth0",
        "arp -i eth0 -s 10.0.2.20 08:00:00:00:02:00"]},
    "h3": { "ip": "10.0.3.3/24", "mac": "08:00:00:00:03:33",
      "commands": ["route add default gw 10.0.3.30 dev eth0",
        "arp -i eth0 -s 10.0.3.30 08:00:00:00:03:00"]},
    "h4": { "ip": "10.0.4.4/24", "mac": "08:00:00:00:04:44",
      "commands": ["route add default gw 10.0.4.40 dev eth0",
        "arp -i eth0 -s 10.0.4.40 08:00:00:00:04:00"]}
  },
  "switches": {
    "s1": { "runtime_json" : "pod-topo/s1-runtime.json" },
    "s2": { "runtime_json" : "pod-topo/s2-runtime.json" },
    "s3": { "runtime_json" : "pod-topo/s3-runtime.json" },
    "s4": { "runtime_json" : "pod-topo/s4-runtime.json" }
  },
  "links": [
    ["h1", "s1-p1"], ["h2", "s1-p2"], ["s1-p3", "s3-p1"], ["s1-p4", "s4-p2"],
    ["h3", "s2-p1"], ["h4", "s2-p2"], ["s2-p3", "s4-p1"], ["s2-p4", "s3-p2"]
  ]
}
```

Yukarıdaki topolojiye göre anahtarların akışları pod-topo klasörü içerisindeki ayrı dosyalar üzerinden tanımlanıyor. S1 anahtarı üzerinde kural eklememiz gerekiyorsa “s1-runtime.json” dosyasına bu kuralları eklememiz gerekecektir.

Kuralları eklememiz gereken formatın bulunması gerekecektir.

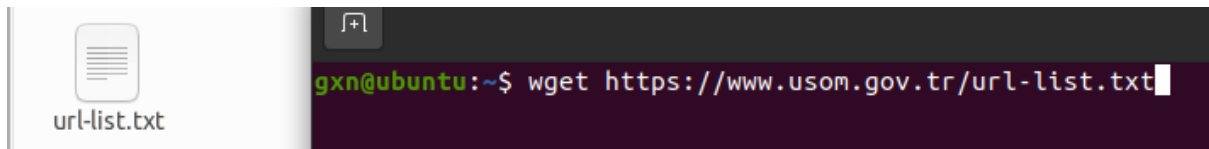
```
{
  "target": "bmv2",
  "p4info": "build/basic.p4.p4info.txt",
  "bmv2_json": "build/basic.json",
  "table_entries": [
    {
      "table": "MyIngress.ipv4_lpm",
      "default_action": true,
      "action_name": "MyIngress.drop",
      "action_params": {}
    },
  ],
}
```

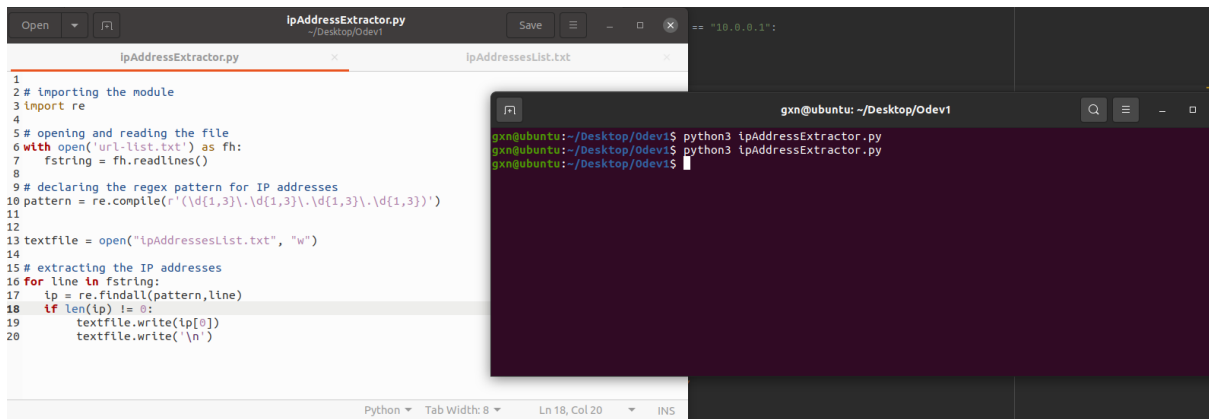
Dosya üzerinde table ipv4\_lpm üzerinden seçiliyor ve yukarıda MyIngress class'ı üzerinden drop fonksiyonu seçilebilmesi için action\_name="MyIngress.drop" şeklinde tanımlanmalıdır. Her IP için ise match kısmında ip'nin dstAddr parametresi üzerinden tanımlanması gerekmektedir.

```
{
  "table": "MyIngress.ipv4_lpm",
  "match": {
    "hdr.ipv4.dstAddr": [
      "145.249.107.244",
      32
    ]
  },
  "action_name": "MyIngress.drop",
  "action_params": {}
},
}
```

Yapılması gereken belirlenmiştir. "s1-runtime.json" dosyasında her ip için yukarıdaki formatta JSON girişi eklenmesi gerekmektedir.

Usom engelli site verilerini önceki ödevden yapıldığı sırayla dns sorgulama yapıp, bir text dosyasına yazılır. Önceki ödevden olan görüntüler aşağıda özetle paylaşılmıştır.



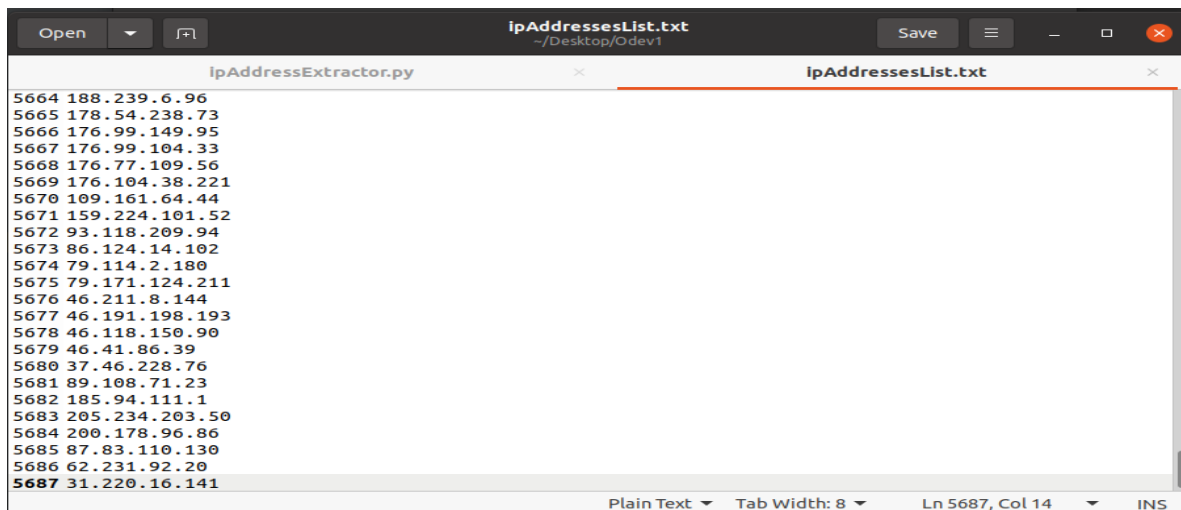


The screenshot shows a code editor with two tabs: `ipAddressExtractor.py` and `ipAddressesList.txt`. The `ipAddressExtractor.py` script is open and shows the following code:

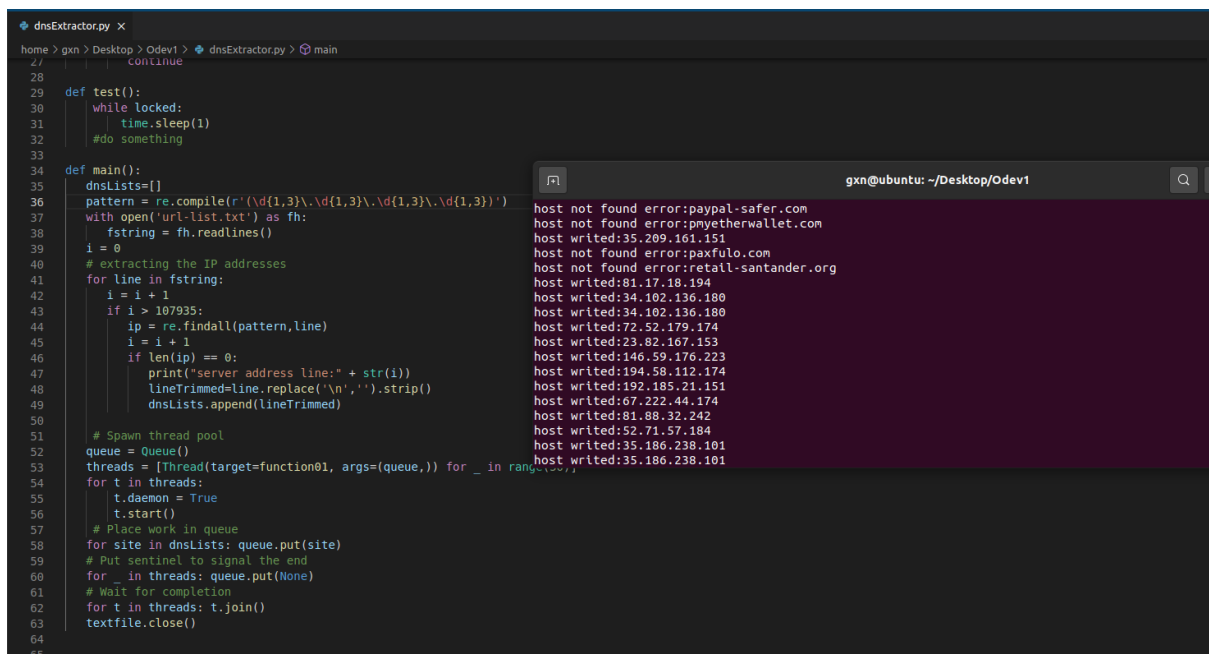
```
1
2 # importing the module
3 import re
4
5 # opening and reading the file
6 with open('url-list.txt') as fh:
7     fstring = fh.readlines()
8
9 # declaring the regex pattern for IP addresses
10 pattern = re.compile(r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}')
11
12
13 textfile = open("ipAddressesList.txt", "w")
14
15 # extracting the IP addresses
16 for line in fstring:
17     ip = re.findall(pattern, line)
18     if len(ip) != 0:
19         textfile.write(ip[0])
20         textfile.write('\n')
```

A terminal window is open in the foreground, showing the execution of the script:

```
gxn@ubuntu: ~/Desktop/Odev1
gxn@ubuntu:~/Desktop/Odev1$ python3 ipAddressExtractor.py
gxn@ubuntu:~/Desktop/Odev1$ python3 ipAddressExtractor.py
gxn@ubuntu:~/Desktop/Odev1$
```



The screenshot shows the `ipAddressesList.txt` file, which contains a list of IP addresses extracted from the `url-list.txt` file. The IP addresses are listed in a single column, separated by newlines. The status bar at the bottom indicates the file is in Plain Text mode, with a tab width of 8, and the cursor is at line 5687, column 14.

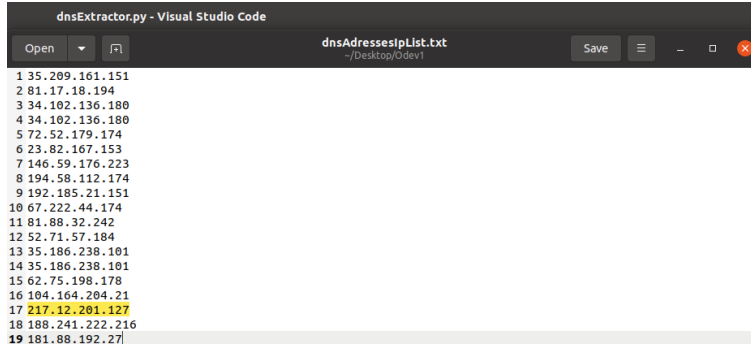


The screenshot shows a code editor with two tabs: `dnsExtractor.py` and `ipAddressesList.txt`. The `dnsExtractor.py` script is open and shows the following code:

```
28
29
30 def test():
31     while locked:
32         time.sleep(1)
33         #do something
34
35
36 def main():
37     dnsLists=[]
38     pattern = re.compile(r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}')
39     with open('url-list.txt') as fh:
40         fstring = fh.readlines()
41         i = 0
42         # extracting the IP addresses
43         for line in fstring:
44             i = i + 1
45             if i > 107935:
46                 ip = re.findall(pattern, line)
47                 i = i + 1
48                 if len(ip) == 0:
49                     print("server address line:" + str(i))
50                     lineTrimmed=line.replace('\n','').strip()
51                     dnsLists.append(lineTrimmed)
52
53     # Spawn thread pool
54     queue = Queue()
55     threads = [Thread(target=function01, args=(queue,)) for _ in range(10)]
56     for t in threads:
57         t.daemon = True
58         t.start()
59     # Place work in queue
60     for site in dnsLists: queue.put(site)
61     # Put sentinel to signal the end
62     for _ in threads: queue.put(None)
63     # Wait for completion
64     for t in threads: t.join()
65     textfile.close()
```

A terminal window is open in the foreground, showing the execution of the script:

```
gxn@ubuntu: ~/Desktop/Odev1
host not found error:paypal-safer.com
host not found error:pmetherwallet.com
host writed:35.209.161.151
host not found error:paxfulco.com
host not found error:retall-santander.org
host writed:81.17.18.194
host writed:34.102.136.180
host writed:34.102.136.180
host writed:72.52.179.174
host writed:23.82.167.153
host writed:146.59.176.223
host writed:194.58.112.174
host writed:192.185.21.151
host writed:67.222.44.174
host writed:81.88.32.242
host writed:52.71.57.184
host writed:35.186.238.101
host writed:35.186.238.101
```



```
1 35.209.161.151
2 81.17.18.194
3 34.102.136.180
4 34.102.136.180
5 72.52.179.174
6 23.82.167.153
7 146.59.176.223
8 194.58.112.174
9 192.185.21.151
10 67.222.44.174
11 81.88.32.242
12 52.71.57.184
13 35.186.238.101
14 35.186.238.101
15 62.75.198.178
16 104.164.204.21
17 217.12.201.127
18 188.241.222.216
19 181.88.192.27
```

Tüm dosyaları bir text dosyasında topladıktan sonra text dosyasındaki ip'leri her satır için json girişi yapılmalıdır. Bunun için bir python programı yazılır.

```
# importing the module
import re
import socket
import json

ipList = []

def extract_ip_address_from_txt_file():
    with open('ipAddressesList.txt') as fh:
        fstring = fh.readlines()
        for line in fstring:
            ip=line.strip()
            ipList.append(ip)
        print("Removing duplicates...")
        converted_set = list(set(ipList))
        print("Size after removed duplicates:", len(converted_set))
        fh.close()
        return converted_set
```

İlk olarak ip adreslerini listeye alıp, duplicate girişleri temizleyen bir fonksiyon yazılır. Böylelikle bu liste üzerinden geçilerek her eleman için giriş eklenebilir hale getirilir.

```
set = extract_ip_address_from_txt_file()[0:1000]
jsonReader = open('s1-runtime.json')
currentJsonData = json.load(jsonReader)
currentDict = currentJsonData['table_entries']
added_entries = []
```

“s1-runtime.json” dosyası açılır ve hazırda yüklenir olan json girişlerini alınır. Key olarak “table\_entries” seçilir çünkü ekleyeceğimiz drop modeli table entries içerisinde eklenecektir.

```

added_entries = []
try:
    print(len(set))
    for element in set:
        jsonString = {
            "table": "MyIngress.ipv4_lpm",
            "match": {
                "hdr.ipv4.dstAddr": [element, 32]
            },
            "action_name": "MyIngress.drop",
            "action_params": {}
        }
        if jsonString in added_entries:
            print("No need to add")
        else:
            added_entries.append(jsonString)
            currentDict.append(jsonString)
            print("Entry Number Added #", len(added_entries))
    currentJsonData['table_entries'] = currentDict
    jsonFileWriter = open("s1-runtime.json", "w")
    json.dump(currentJsonData, jsonFileWriter, default = vars, indent=2)
except:
    print("Error while writing")

```

Her eleman için for iteration başlatılır. IP'ye göre bir json verisi oluşturulur. Yukarıda belirtildiği gibi drop aksiyonu tanımlanır ve dstAddr filtresi olarak sıradaki ip adresi verilir. Geçici veri oluşturulduktan sonra bu verinin daha önce eklenip eklenmediği kontrol edilir, eklendiyse ona göre gerekli mesaj basılır. Eğer eklenmediyse, eklenenler listesine eklenip, json dosyasındaki sözlük değişkenindeki “table\_entries” kısmına eklenir. Tüm veriler eklendikten sonra dosyanın üzerinde güncelleme yapılır. Böylelikle her adres için bir tablo girişi yapılmış olur ve adreslerin drop kuralları da tanımlanmış olur.

Bu python programı güncel s1-runtime.json dosyası üzerinde çalıştırıldıktan sonra “pod-topo” klasörü içerisine taşınır. Artık mini-net topolojisini başlatıp kuralların çalışıp çalışmadığını görebiliriz.

cd tutorials/exercises/basic -> bu komut ile basic klasörüne geçilir.

mv basic.p4 basic.p4\_old

cp solution/basic.p4 basic.p4

make run -> bu komut ile birlikte Makefile üzerinden mini-net çalışmaya başlanır.

Başlatıldıktan sonra drop kurallarının mininet üzerinde log'larının düştüğü görülür.

```

- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['34.130.179.120', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['104.21.67.246', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['68.7.197.250', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['185.117.75.251', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['63.141.242.46', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['68.96.205.9', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['190.14.39.100', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['213.183.59.104', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['64.188.16.140', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['185.250.240.28', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['209.141.62.19', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['91.193.19.97', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['158.247.208.230', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['20.190.39.195', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['185.225.226.16', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['185.225.19.187', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['176.65.78.202', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['157.90.155.38', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['143.244.158.61', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['104.21.61.124', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['95.216.112.62', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['45.77.249.192', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['172.67.165.79', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['174.71.80.109', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['75.119.128.12', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['67.198.149.218', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['35.223.30.36', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['45.123.190.168', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['192.99.28.172', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['185.172.110.245', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['72.218.188.5', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['37.49.230.103', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['117.20.41.87', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['27.102.107.63', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['51.158.125.92', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['107.6.74.76', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['109.208.185.186', 32] => MyIngress.drop()
- MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['185.112.146.83', 32] => MyIngress.drop()

```

simple\_switch\_CLI --thrift-port 9090 komutu ile p4 shell'i açılır.

table\_dump MyIngress.ipv4\_lpm -> Bu komut ile yukarıdaki resimde görüldüğü gibi akışların tablosu görüntülenir. Action entry görüldüğü gibi MyIngress.drop gözükmemektedir.

```

Action entry: MyIngress.drop -
*****
Dumping entry 0x36f
Match key:
* ipv4.dstAddr      : LPM      d18d2856/32
Action entry: MyIngress.drop -
*****
Dumping entry 0x370
Match key:
* ipv4.dstAddr      : LPM      5bda7226/32
Action entry: MyIngress.drop -
*****
Dumping entry 0x371
Match key:
* ipv4.dstAddr      : LPM      d401d267/32
Action entry: MyIngress.drop -
*****
Dumping entry 0x372
Match key:
* ipv4.dstAddr      : LPM      73babd06/32
Action entry: MyIngress.drop -
*****
Dumping entry 0x373
Match key:
* ipv4.dstAddr      : LPM      c6d30a0a/32
Action entry: MyIngress.drop -
*****
Dumping entry 0x374
Match key:
* ipv4.dstAddr      : LPM      c0b9beb4/32
Action entry: MyIngress.drop -
*****
Dumping entry 0x375
Match key:
* ipv4.dstAddr      : LPM      ac43c534/32
Action entry: MyIngress.drop -

```

Mininet üzerinde pingAll yaparak drop kurallarından bazıları çalışıp çalışmadığını görülebilir.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 X
h2 -> X X X
h3 -> h1 X h4
h4 -> X X h3
*** Results: 66% dropped (4/12 received)
mininet> █
```

Paketlerin %66'sı düşürülmüş. Bu da drop kurallarının bazılarının çalıştığını gösterir.

Sonuç olarak, P4 yazılımlarıyla, SDN'in sağladığı güvenlik beyni inşa edilebiliyor. Daha farklı yapılarda gözüke de, komut alan anahtarları yönetmede avantaj sağlıyorlar. USOM üzerindeki tehlikeli statüdeki siteleri de böylelikle ağ üzerindeki dolaşımda kısıtlamış olduk.

Projenin github linki: <https://github.com/goksunonal/P4SmartSwitch>

Basic klasöründeki pod-topo klasörü kopyalanarak test edilebilir.

**Göksun ÖNAL**

**211111034**