Software-Defined Networking
Assignment 05

Peipei He, st169741@stud.uni-stuttgart.de, 3442500
Huicheng Qian, st169665@stud.uni-stuttgart.de, 3443114
Kuang-Yu Li, st169971@stud.uni-stuttgart.de, 3440829

Task1

simple_firewall

start our pyretic firewall module:

```
ex5$ pyretic.py -v low -m r0 simple_firewall
```

start mininet and ssh & web services on srv & inet, and a xterm for all nodes:

```
ex5$ sudo ./mininet5.py mininet
> startservers mininet
> xterm h1 h2 srv inet mon
```

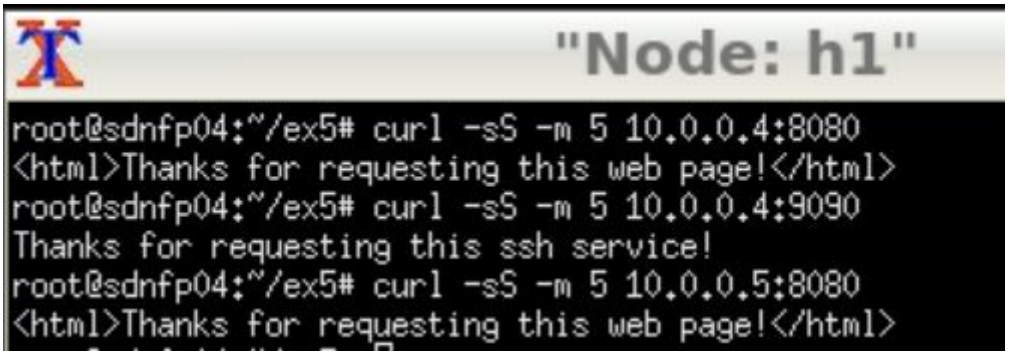
On h1, h2, and inet access web & ssh services through:

```
curl -sS -m 5 10.0.0.X:Y0Y0
```

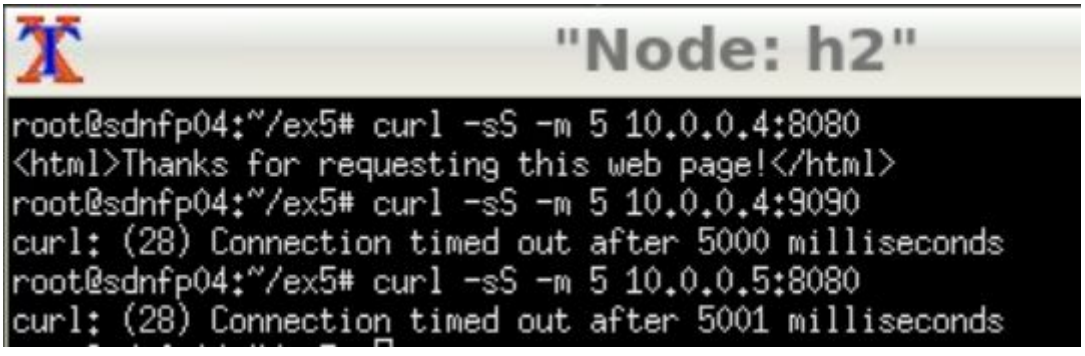
On mon start monitoring with the packet capture tool `tcpdump` to print information about received packets

Output:

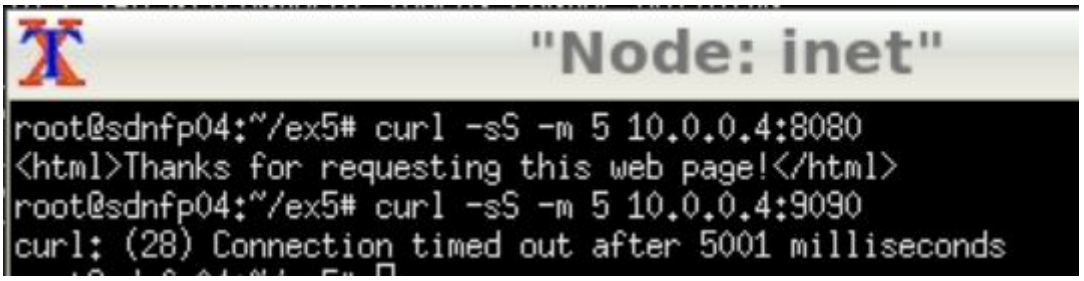
h1 can access all services running on srv and on inet



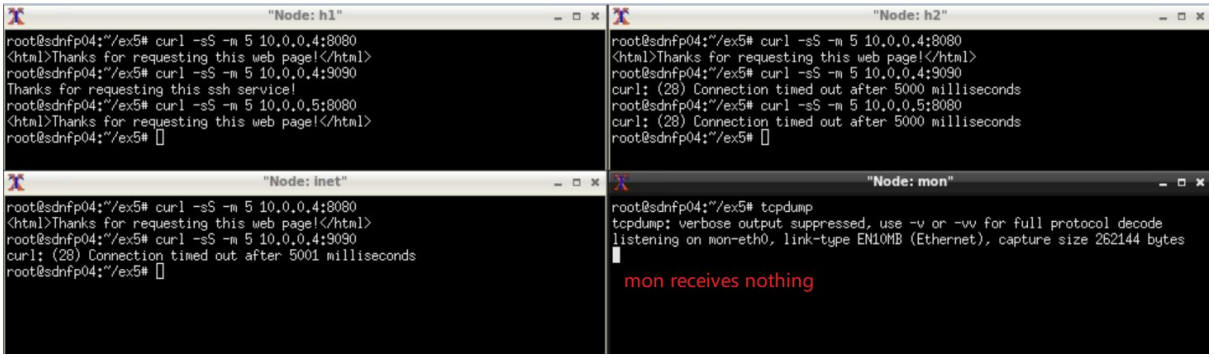
h2 can only access the web service on srv



inet can only access the web service on srv



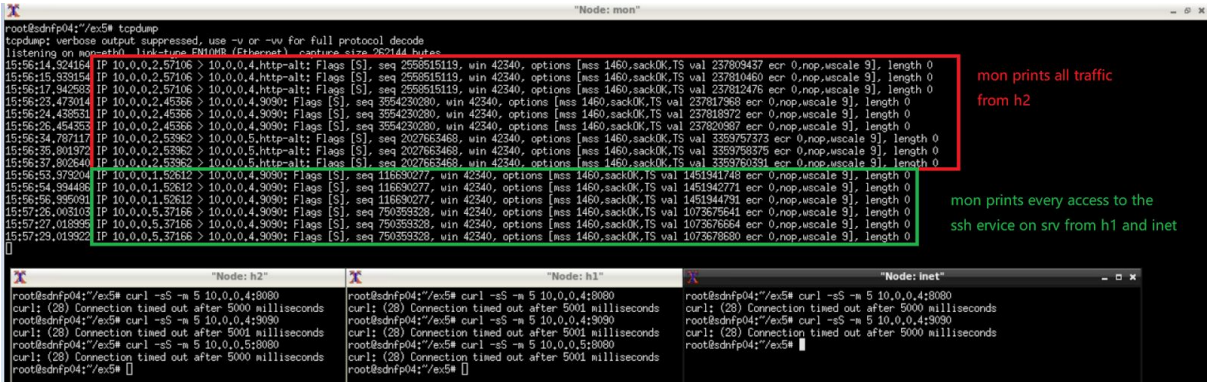
monitor does not receive any packet



simple_monitor

The monitor policy is: if there is a packet sent from h2 or a packet that wants to access the ssh service on srv, we forward it to mon.

```
monitorPolicy = (match(srcip=ip_h2) | match(dstip=ip_srv, dstport=9090, ethtype=packet.IPV4, protocol=packet.TCP_PROTO)) >> (fwd(3))
```



Task2

Start the pyretic module:

```
ex5$ pyretic.py -v low -m r0 qos
```

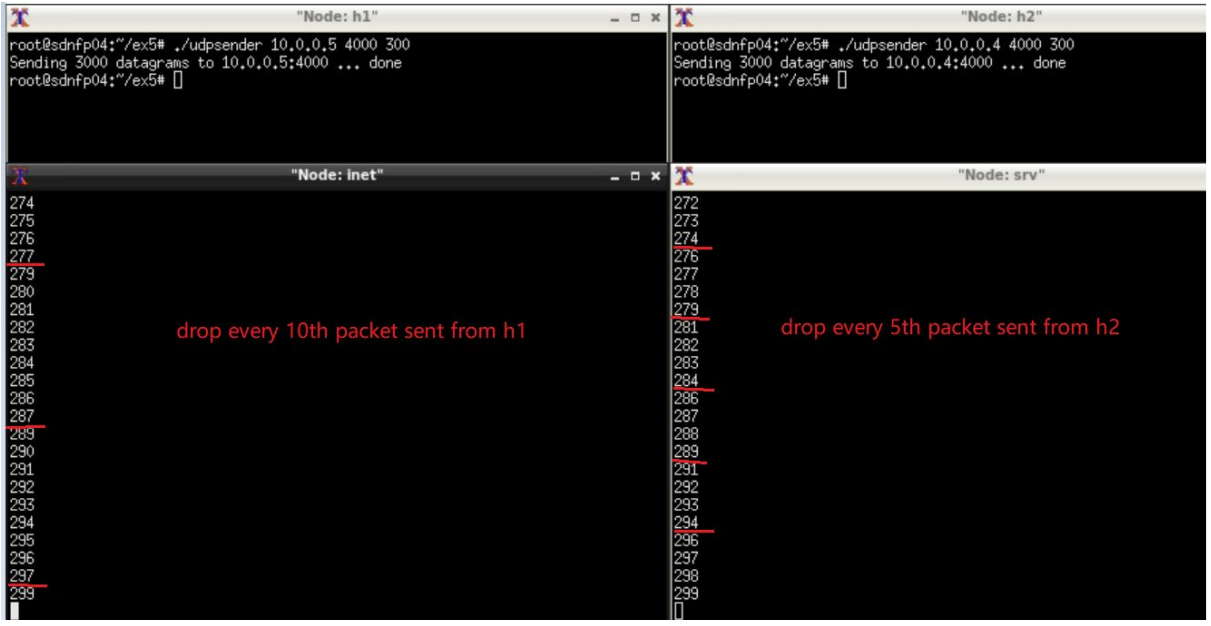
On inet and srv, run `./udpreceiver 4000`, receiving datagrams on port 4000

On h1 and h2, run `./udpsender 10.0.0.X 4000 300`, sending 300 datagrams to node inet:4000 or node srv:4000

The udpsender sends 10 datagrams per second and our policy `count_packets(0.05, ['srcip'])` calls its listeners every 50 milliseconds and provides each listener with a dictionary mapping source IP addresses to the cumulative number of packets containing that source IP address received by the bucket. The dictionary is shown as below:

```
{match: ('srcip', IPv4Network('10.0.0.1/32')): 256}
```

In the callback function, we perform the remainder calculation. If the remainder of the count divided by 10 is 0 for the sender h1, we change the policy to drop the packet sending from h1, otherwise we do not change the policy. Similarly, if the remainder of the count divided by 5 is 0 for the sender h2, the policy will be changed.



Task3

monitor_firewall

start the pyretic module:

```
ex5$ pyretic.py -v low -m r0 monitor_firewall
```

start mininet and ssh & web services on srv & inet, and a xterm for all nodes:

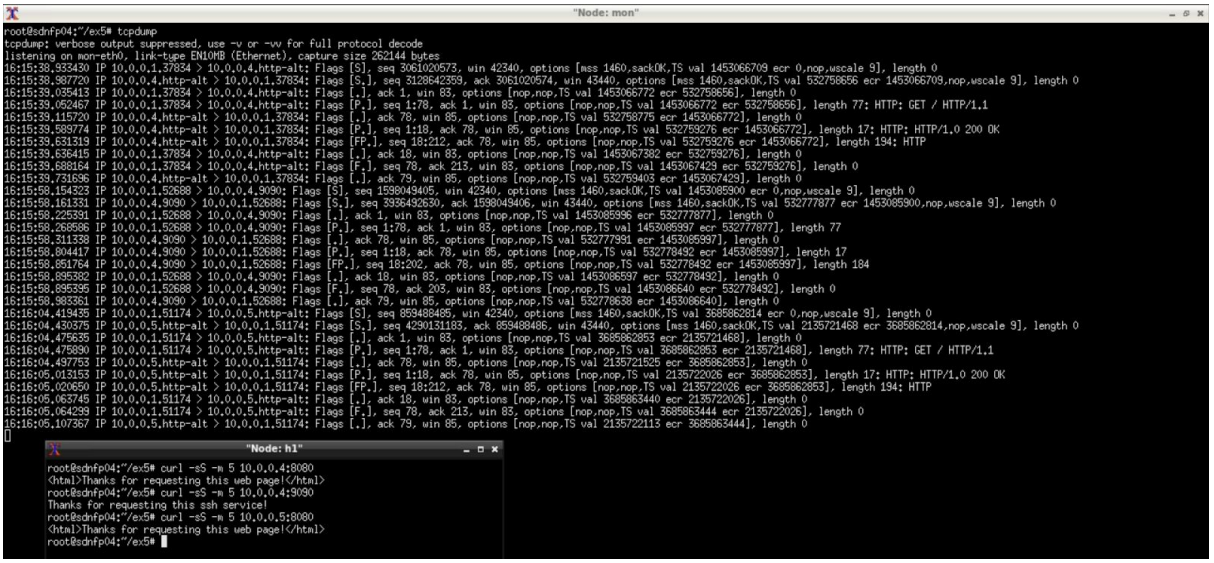
```
ex5$ sudo ./mininet5.py mininet
> startservers mininet
> xterm h1 h2 srv inet mon
```

On h1, h2, and inet access web & ssh services through:

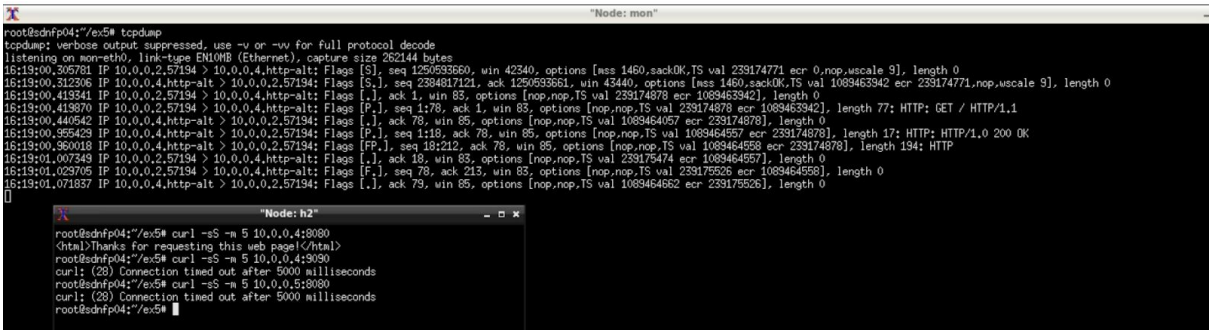
```
curl -sS -m 5 10.0.0.X:Y0Y0
```

On mon start monitoring with the packet capture tool `tcpdump` to print information about received packets

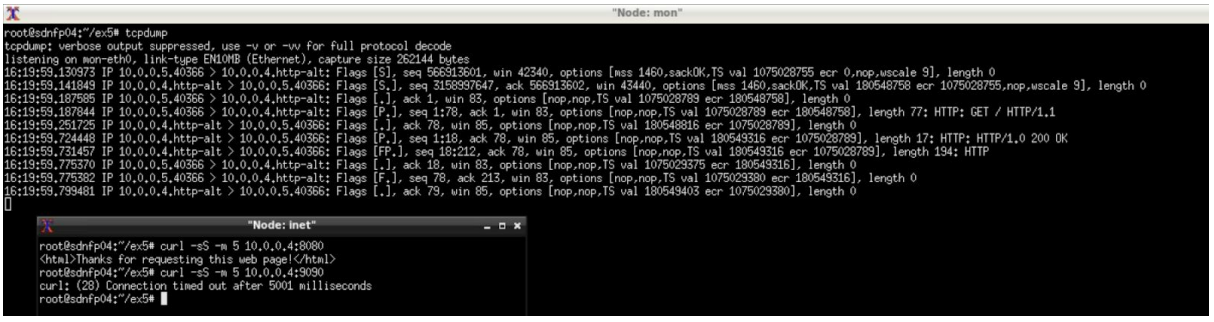
Output:
mon prints the traffic between h1 and web & ssh services on srv & inet



mon only prints the traffic between h2 and the web service on srv



mon only prints the traffic between inet and web service on srv



monitor_firewall_qos

The basic logic is: if there is an UDP datagram, then we perform the QoS policy, otherwise we perform the firewall and monitoring policy:

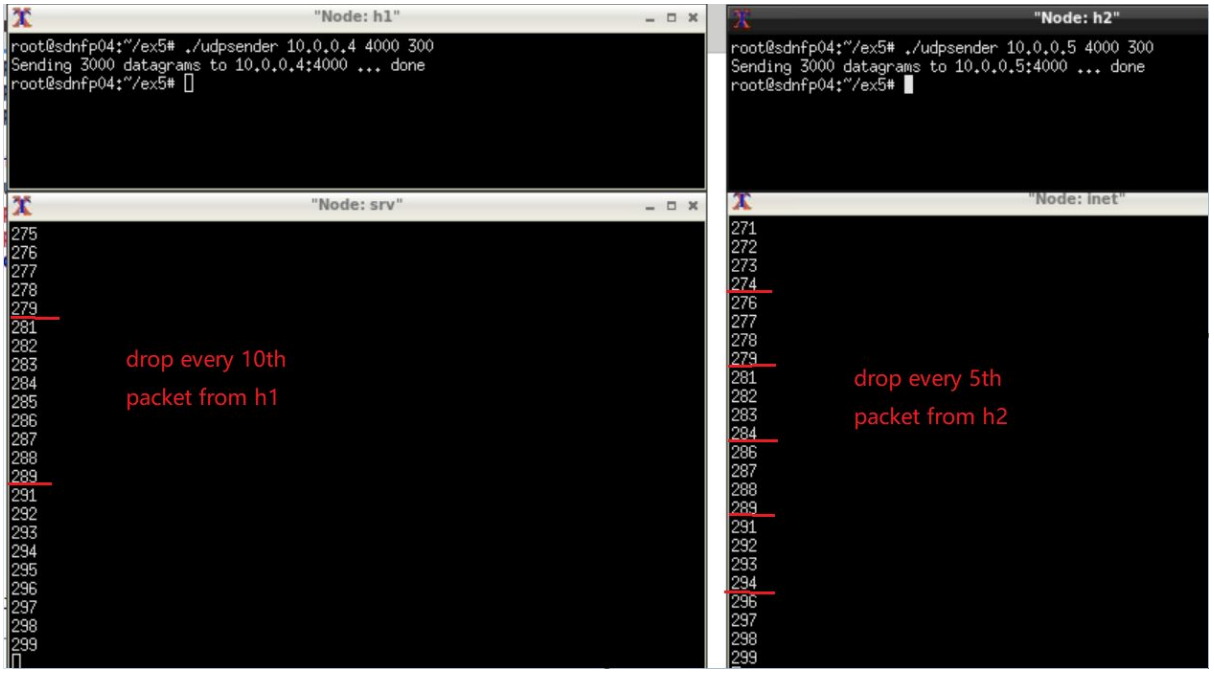
```
if_(match(protocol=packet.UDP_PROTO), qos(), monitor_firewall)
```

Start the pyretic module:

```
ex5$ pyretic.py -v low -m r0 monitor_firewall_qos
```

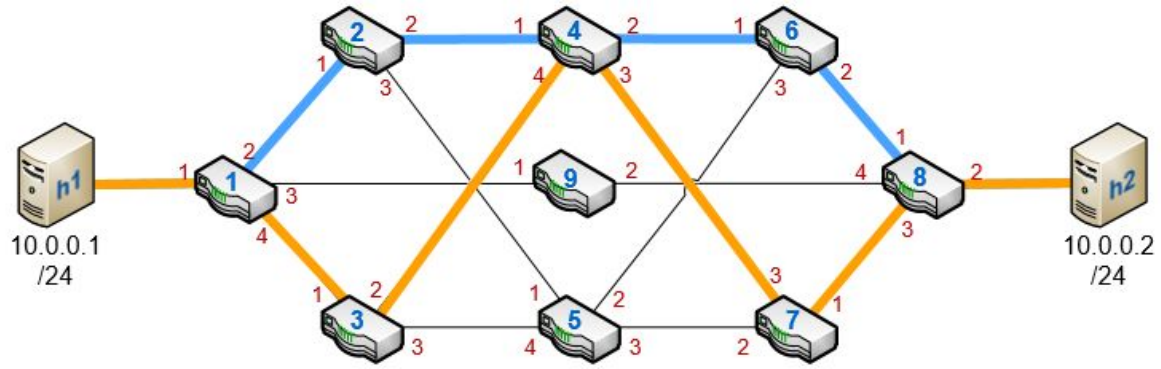
```
student@sdnfp04:~/ex5$ pyretic.py -v low -m r0 monitor_firewall_qos
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
Connected to pyretic frontend.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
```

On inet and srv, run `./udpreceiver 4000`, receiving datagrams on port 4000
On h1 and h2, run `./udpsender 10.0.0.X 4000 300`, sending 300 datagrams to node inet:4000 or node srv:4000



The monitor and firewall part is the same as above.

Task4



blue.py
`(match(switch=1) | match(switch=2) | match(switch=4) | match(switch=6) | match(switch=8)) & match(inport=1) >> fwd(2)`

orange.py

```
(match(switch=1,inport=1) >> fwd(4)) +
((match(switch=3,inport=1)| match(switch=8,inport=3)) >> fwd(2))
+
(match(switch=4,inport=4) >> fwd(3)) +
(match(switch=7,inport=3) >> fwd(1))
```

Forwarding abstraction of Pyretic is not considered purly behavioral. In this task, we have to specify the port of a switch in each forwarding path. It is desired to specify only the switch of the designated route without specifying the exact port or lower layer connectivity.

It would be desired to only specify switch on routing path and leave compiler to match the topology with ports. For example, we, the user, only want to declare two flows, with special function in pyretic `CREATE_FLOW(switch=1, 2, 4, 6, 8)` and `CREATE_FLOW(switch=1, 3, 4, 7, 8)` and don't want to take care of the detail connection.

For pyretic programming, the space complexity is O(N*M), where N is the number of total switches and M is the number of the port for each switch, because at worst case, we need to specifty every in-out-port for every switches.

Space complexity of `blue.py` is lower than that of `orange.py` because there is only one action (`fwd(2)`) in the former case. However, since the inport and outport for each switch is different in the orange route, we have to define different match conditions and actions for different switches, so the policy is more complex and requires more memory space to store.