Specialization Software-defined Networking (Winter 2014/2015))

Institute of Computer Science
Georg-August-Universität Göttingen

Lecturer: Dr.Mayutan Arumaithurai
Author: Hari Raghavendar Rao Bandari
11334055
h.bandari@stud.uni-goettingen.de

Date: April 30, 2015

Exercise II : Fattree and Generic

2. Expand it to make it generic

```
Expand it to make it generic with different bandwidth requirements

Comment out simpleTest() and uncomment simpleTest_perf() in datacenter_run.py.

ouputput : simpleTest() function

mininet@mininet-vm:~/1.Data_centers
$ sudo ./datacenter_run.py
 ['s00', 's10', 's11', 's20', 's21', 's22', 's23', 's30', 's31', 's32', 's33', 's3
*** Creating network
*** Adding controller
*** Adding hosts:
h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
*** Adding switches:
core s00 s10 s11 s20 s21 s22 s23 s30 s31 s32 s33 s34 s35 s36 s37
*** Adding links:
(s00, s10) (s00, s11) (s10, s20) (s10, s21) (s11, s22) (s11, s23) (s20, s30) (s20,
*** Configuring hosts
h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
*** Starting controller
 c0
*** Starting 16 switches
core s00 s10 s11 s20 s21 s22 s23 s30 s31 s32 s33 s34 s35 s36 s37
Dumping host connections
h0 h0-eth0:s30-eth2 h1 h1-eth0:s30-eth3 h2 h2-eth0:s31-eth2 h3 h3-eth0:s31-eth3 h4
Testing network connectivity
*** Ping: testing ping reachability
h0 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h1 -> h0 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h2 -> h0 h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
```

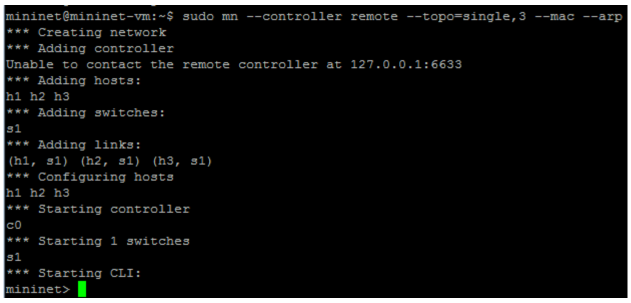
```
h3 -> h0 h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h4 -> h0 h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h5 -> h0 h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h6 -> h0 h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12 h13 h14 h15
h7 -> h0 h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12 h13 h14 h15
h8 -> h0 h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12 h13 h14 h15
h9 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12 h13 h14 h15
h10 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12 h13 h14 h15
h11 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12 h13 h14 h15
h12 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13 h14 h15
h13 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h14 h15
h14 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h15
h15 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14
*** Results: 0% dropped (240/240 received)
*** Stopping 1 controllers
c0
*** Stopping 16 switches
core s00 ..s10 ...s11 ...s20 ...s21 ...s22 ...s23 ...s30 ...s31 ...s32 ...s33 ...s
*** Stopping 30 links
*** Stopping 16 hosts
h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
*** Done

ouputput: simpleTest_perf() function



mininet@mininet-vm:~/1.Data_centers
$ sudo ./datacenter_run.py
['s00', 's10', 's11', 's20', 's21', 's22', 's23', 's30', 's31', 's32', 's33', 's34
s00 s00 s10 s10 s11 s11 s20 s20 s21 s21 s22 s22 s23 s23 8
*** Creating network
*** Adding controller
*** Adding hosts:
h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
```

```
*** Adding switches:
core s00 s10 s11 s20 s21 s22 s23 s30 s31 s32 s33 s34 s35 s36 s37
*** Adding links:
(s00, s10) (s00, s11) (s10, s20) (s10, s21) (s11, s22) (s11, s23) (s20, s30) (s20,
*** Configuring hosts h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
*** Starting controller
c0
*** Starting 16 switches
core s00 s10 s11 s20 s21 s22 s23 s30 s31 s32 s33 s34 s35 s36 s37
Dumping host connections
h0 h0-eth0:s30-eth2 h1 h1-eth0:s30-eth3 h2 h2-eth0:s31-eth2 h3 h3-eth0:s31-eth3 h4
Testing network connectivity
*** Ping: testing ping reachability
h0 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h1 -> h0 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h2 -> h0 h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h3 -> h0 h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h4 -> h0 h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h5 -> h0 h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h6 -> h0 h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12 h13 h14 h15
h7 -> h0 h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12 h13 h14 h15
h8 -> h0 h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12 h13 h14 h15
h9 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12 h13 h14 h15
h10 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12 h13 h14 h15
h11 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12 h13 h14 h15
h12 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13 h14 h15
h13 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h14 h15
h14 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h15
h15 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14
*** Results: 0% dropped (240/240 received)
Testing bandwidth between h1 and h7
*** Iperf: testing TCP bandwidth between h1 and h7
*** Results: ['5.65 Gbits/sec', '5.66 Gbits/sec']
*** Stopping 1 controllers
```

```
c0
*** Stopping 16 switches
core s00 ..s10 ...s11 ...s20 ...s21 ...s22 ...s23 ...s30 ...s31 ...s32 ...s33 ...s3
*** Stopping 30 links
 *** Stopping 16 hosts
h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
*** Done
```

EXERCISE VI Pyretic Firewall

```
\$ sudo mn controller remote topo=single,3 mac arp
```



```
mininet@mininet-vm:~$ sudo mn --controller remote --topo=single,3 --mac --arp
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Figure 1: Screenshot of topology generated[1]

```
$ pyretic.py v high pyretic.examples.pyretic_firewall
```

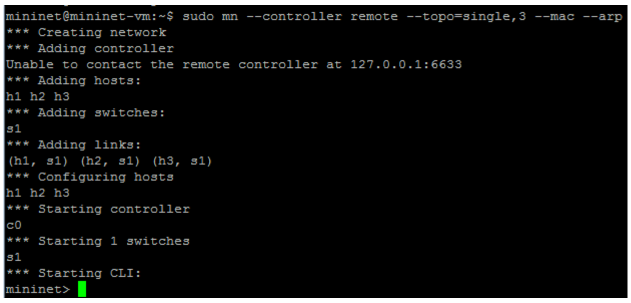
Output:

```
On executing the above command, firewall rules are installed to block packet moveme
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5014ms

mininet> h2 ping h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3010ms

mininet> h2 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=396 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=116 ms
```

```
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=70.8 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 70.899/194.855/396.686/143.951 ms

mininet> h3 ping h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=277 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=111 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=85.1 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2007ms
rtt min/avg/max/mdev = 85.157/158.181/277.563/85.115 ms
mininet>
```

EXERCISE VII Kinetic Firewall

```
$ sudo mn controller remote topo=single,3 mac arp
```



```
mininet@mininet-vm:~$ sudo mn --controller remote --topo=single,3 --mac --arp
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Figure 2: Screenshot of topology generated[2]

```
$ pyretic.py v high pyretic.kinetic.examples.kinetic_gardenwall
```

Outputs:

When h1 is not infected, it is able to reach h2

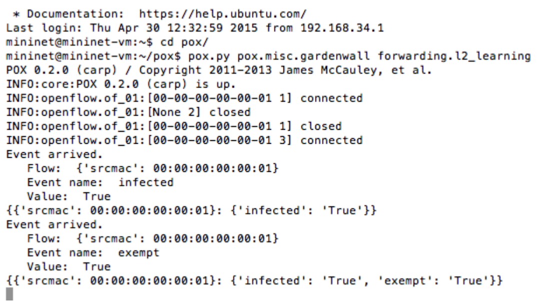
```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=112 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=85.6 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=72.6 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=107 ms

When h1 is infected, it is not able to reach h2
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
14 packets transmitted, 0 received, 100% packet loss, time 13045ms
```

Exercise VIII: Kinetic like firewall using pox

```
$ mv pox_gardenwall.py ~/pox/pox/misc/gardenwall.py
```

1. Run your Pox controller application (Part 2):

```
$ cd ~/pox
$ pox.py pox.misc.gardenwall forwarding.l2_learning
```



```
 * Documentation:  https://help.ubuntu.com/
Last login: Thu Apr 30 12:32:59 2015 from 192.168.34.1
mininet@mininet-vm:~$ cd pox/
mininet@mininet-vm:~/pox$ pox.py pox.misc.gardenwall forwarding.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:openflow.of_01:[None 2] closed
INFO:openflow.of_01:[00-00-00-00-00-01 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
Event arrived.
   Flow:  {'srcmac': 00:00:00:00:00:01}
   Event name:  infected
   Value:  True
{{'srcmac': 00:00:00:00:00:01}: {'infected': 'True'}}
Event arrived.
   Flow:  {'srcmac': 00:00:00:00:00:01}
   Event name:  exempt
   Value:  True
{{'srcmac': 00:00:00:00:00:01}: {'infected': 'True', 'exempt': 'True'}}
```

Figure 3: Screenshot of gardenwall forwarding with POX[3]

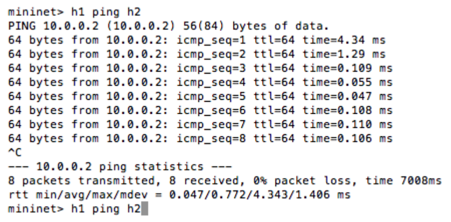
2. Start the mininet setup in a new console.

```
$ sudo mn --controller=remote --topo=single,3 --mac --arp
```

For either of these controllers, start a ping from h1 to h2

```
mininet> h1 ping h2
```

Send Event to block traffic ”h1 ping h2” (in ” /pyretic/pyretic/kinetic” directory)



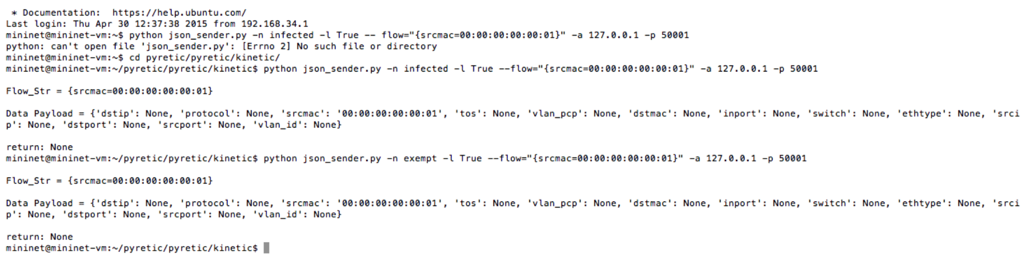
```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4.34 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.29 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.109 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.108 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.110 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.106 ms
^C
--- 10.0.0.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7008ms
rtt min/avg/max/mdev = 0.047/0.772/4.343/1.406 ms
mininet> h1 ping h2
```

Figure 4: Screenshot of h1 ping h2[4]

```
$ python json_sender.py -n infected -l True --
flow="{srcmac=00:00:00:00:00:01}" -a 127.0.0.1 -p 50001
```

Make h1's flow not be affected by IDS infection event, h1's traffic should be forwarded to 10.0.0.3 after issuing this command:

```
$ python json_sender.py -n exempt -l True --flow="{srcmac=00:00:00:00:00:01}"
-a 127.0.0.1 -p 50001
```



```
 * Documentation:  https://help.ubuntu.com/
Last login: Thu Apr 30 12:37:38 2015 from 192.168.34.1
mininet@mininet-vm:~$ python json_sender.py -n infected -l True -- flow="{srcmac=00:00:00:00:00:01}" -a 127.0.0.1 -p 50001
python: can't open file 'json_sender.py': [Errno 2] No such file or directory
mininet@mininet-vm:~$ cd pyretic/pyretic/kinetic/
mininet@mininet-vm:~/pyretic/pyretic/kinetic$ python json_sender.py -n infected -l True --flow="{srcmac=00:00:00:00:00:01}" -a 127.0.0.1 -p 50001

Flow_Str = {srcmac=00:00:00:00:00:01}

Data Payload = {'dstip': None, 'protocol': None, 'srcmac': '00:00:00:00:00:01', 'tos': None, 'vlan_pcp': None, 'dstmac': None, 'inport': None, 'switch': None, 'ethtype': None, 'srci
p': None, 'dstport': None, 'srcport': None, 'vlan_id': None}

return: None
mininet@mininet-vm:~/pyretic/pyretic/kinetic$ python json_sender.py -n exempt -l True --flow="{srcmac=00:00:00:00:00:01}" -a 127.0.0.1 -p 50001

Flow_Str = {srcmac=00:00:00:00:00:01}

Data Payload = {'dstip': None, 'protocol': None, 'srcmac': '00:00:00:00:00:01', 'tos': None, 'vlan_pcp': None, 'dstmac': None, 'inport': None, 'switch': None, 'ethtype': None, 'srci
p': None, 'dstport': None, 'srcport': None, 'vlan_id': None}

return: None
mininet@mininet-vm:~/pyretic/pyretic/kinetic$ 
```

Figure 5: Screenshot of topology generated[5]

Events to now allow traffic again:

```
$ python json_sender.py -n infected -l False --
flow="{srcmac=00:00:00:00:00:01}" -a 127.0.0.1 -p 50001
```

Exercise IX: Pyretic Debugging

```
HINT: You might have to use the
"$ dpctl dump-flows tcp:127.0.0.1:6634" or
 "mininet> dpctl dump-flows" command frequently.
1.In this debugging exercise, we take solutions available
in the Internet for the gardenwall problem and try to fix bugs in it.
2. We have done kinetic firewall in exercise VII
and imitated the same firewall
using pox in exercise VIII.
3.Now, we will imitate the same firewall using pyretic.
The basic solution is taken from the Internet [9],
test if it is able to block h1 when "infected".
Note that we will only use the "infected == True" for this exercise.
3.1 Copy the above code into
/home/mininet/pyretic/pyretic/examples as gardenwall_internetsolution.py
3.2 start controller (in /home/mininet/pyretic folder): pyretic.py pyretic
.examples.
gardenwall_internetsolution
start mininet: sudo mn --controller=remote --topo=single,3 --mac --arp
3.3 check h1 ping h2
3.4 Now infect h1 (in /home/mininet/pyretic/pyretic/kinetic folder):
python json_sender.py -n infected -l True --flow="{srcmac=00:00:00:00:00:01}"
-a 127.0.0.1 -p 50001
check h1 ping h2. We should be able to observe
 that this traffic is blocked.
3.5 Now, we move on to the debugging part
     check h2 ping h3, what happens?
     Now, modify the given code to allow h2 traffic
     to pass through to h3, when h1 is "infected".
3.6 Now, check if the "exempt" case is
working fine too
if time permits, check and improve code to
allow h1 to ping h2
```

```
If time permits, try fixing this code for the "infected" case.
```

output:

```
After improving the gardenwall from internet we get
the following ping results  h1 to h2 remains blocked
but h2 to h3 works

mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3021ms

mininet> h2 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=293 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=94.4 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=98.4 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=65.5 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=69.1 ms
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4013ms
rtt min/avg/max/mdev = 65.529/124.273/293.773/85.761 ms
mininet>
```