Διαχείριση Δικτύων Βασισμένων στο Λογισμικό

8ο εργαστήριο: "Open vSwitch"

ΟΝΟΜΑΤΕΠΩΝΥΜΟ:	Νικόλας Μαυρόπουλος
Α.Μ.:	21865

Creating applications without using a Controller

Consult the "Without-controller" file.

The ovs-ofctl program is a command line tool for monitoring and administering OpenFlow switches. It can also show the current state of an OpenFlow switch, including features, configuration, and table entries.

Conduct the following experiments using **–controller none** option when creating your Mininet topology!

Part 1: Hub

*Create the following **hub** application using **ovs-ofctl**.*

- *Show a screenshot proving the installed flow entry (flood) in the switch.*

```
mininet> sh ovs-ofctl dump-flows s1
mininet> sh ovs-ofctl add-flow s1 actions=flood
mininet> sh ovs-ofctl dump-flows s1
 cookie=0x0, duration=8.259s, table=0, n_packets=0, n_bytes=0, actions=FLOOD
mininet>
```

- *Show a screenshot that verifies the hub behaviour (showing all 4 hosts/terminals).*

"Node: h1"

```
root@nikolas:~# ping 10.0.0.2
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=0.195 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.057 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.068 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.071 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.079 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.104 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.137 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=0.119 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=0.098 ms
64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=21 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=22 ttl=64 time=0.202 ms
```

"Node: h2"

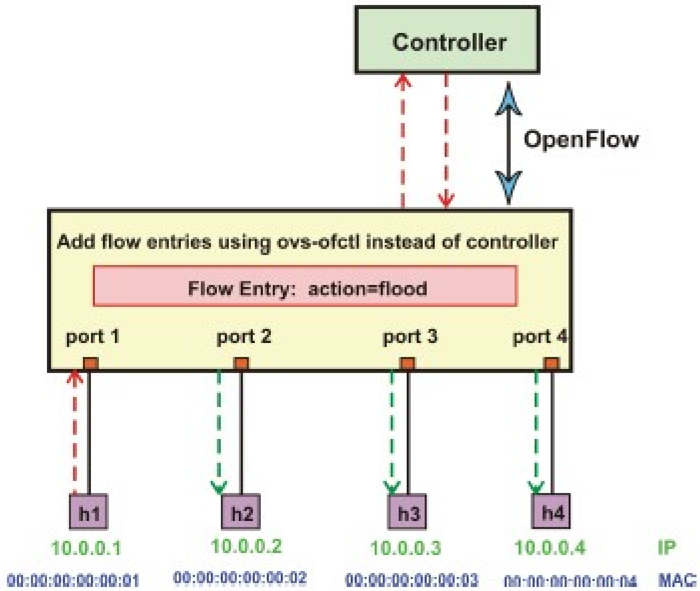
```
root@nikolas:~# tcpdump -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
17:37:19.548143 IP 10.0.0.1 > nikolas: ICMP echo request, id 2815, seq 3, length
 64
17:37:19.548165 IP nikolas > 10.0.0.1: ICMP echo reply, id 2815, seq 3, length 6
4
17:37:20.572775 IP 10.0.0.1 > nikolas: ICMP echo request, id 2815, seq 4, length
 64
17:37:20.572803 IP nikolas > 10.0.0.1: ICMP echo reply, id 2815, seq 4, length 6
4
17:37:21.596032 IP 10.0.0.1 > nikolas: ICMP echo request, id 2815, seq 5, length
 64
17:37:21.596060 IP nikolas > 10.0.0.1: ICMP echo reply, id 2815, seq 5, length 6
4
17:37:22.556549 ARP, Request who-has 10.0.0.1 tell nikolas, length 28
17:37:22.556780 ARP, Request who-has nikolas tell 10.0.0.1, length 28
17:37:22.556785 ARP, Reply nikolas is-at 00:00:00:00:00:02 (oui Ethernet), lengt
h 28
17:37:22.556834 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01 (oui Ethernet), leng
th 28
17:37:22.620323 IP 10.0.0.1 > nikolas: ICMP echo request, id 2815, seq 6, length
 64
17:37:22.620352 IP nikolas > 10.0.0.1: ICMP echo reply, id 2815, seq 6, length 6
```

"Node: h3"

```
root@nikolas:~# tcpdump -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
17:37:20.572776 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2815, seq 4, lengt
h 64
17:37:20.572810 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 2815, seq 4, length
64
17:37:21.596034 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2815, seq 5, lengt
h 64
17:37:21.596066 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 2815, seq 5, length
64
17:37:22.556749 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
17:37:22.556781 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
17:37:22.556787 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02 (oui Ethernet), leng
th 28
17:37:22.556834 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01 (oui Ethernet), leng
th 28
17:37:22.620324 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2815, seq 6, lengt
h 64
17:37:22.620360 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 2815, seq 6, length
64
17:37:23.644152 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2815, seq 7, lengt
h 64
17:37:23.644190 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 2815, seq 7, length
```

"Node: h4"

```
root@nikolas:~# tcpdump -i h4-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h4-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
17:37:20.572773 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2815, seq 4, lengt
h 64
17:37:20.572810 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 2815, seq 4, length
64
17:37:21.596031 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2815, seq 5, lengt
h 64
17:37:21.596066 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 2815, seq 5, length
64
17:37:22.556748 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
17:37:22.556780 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
17:37:22.556787 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02 (oui Ethernet), leng
th 28
17:37:22.556834 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01 (oui Ethernet), leng
th 28
17:37:22.620321 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2815, seq 6, lengt
h 64
17:37:22.620360 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 2815, seq 6, length
64
17:37:23.644149 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 2815, seq 7, lengt
h 64
17:37:23.644190 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 2815, seq 7, length
```



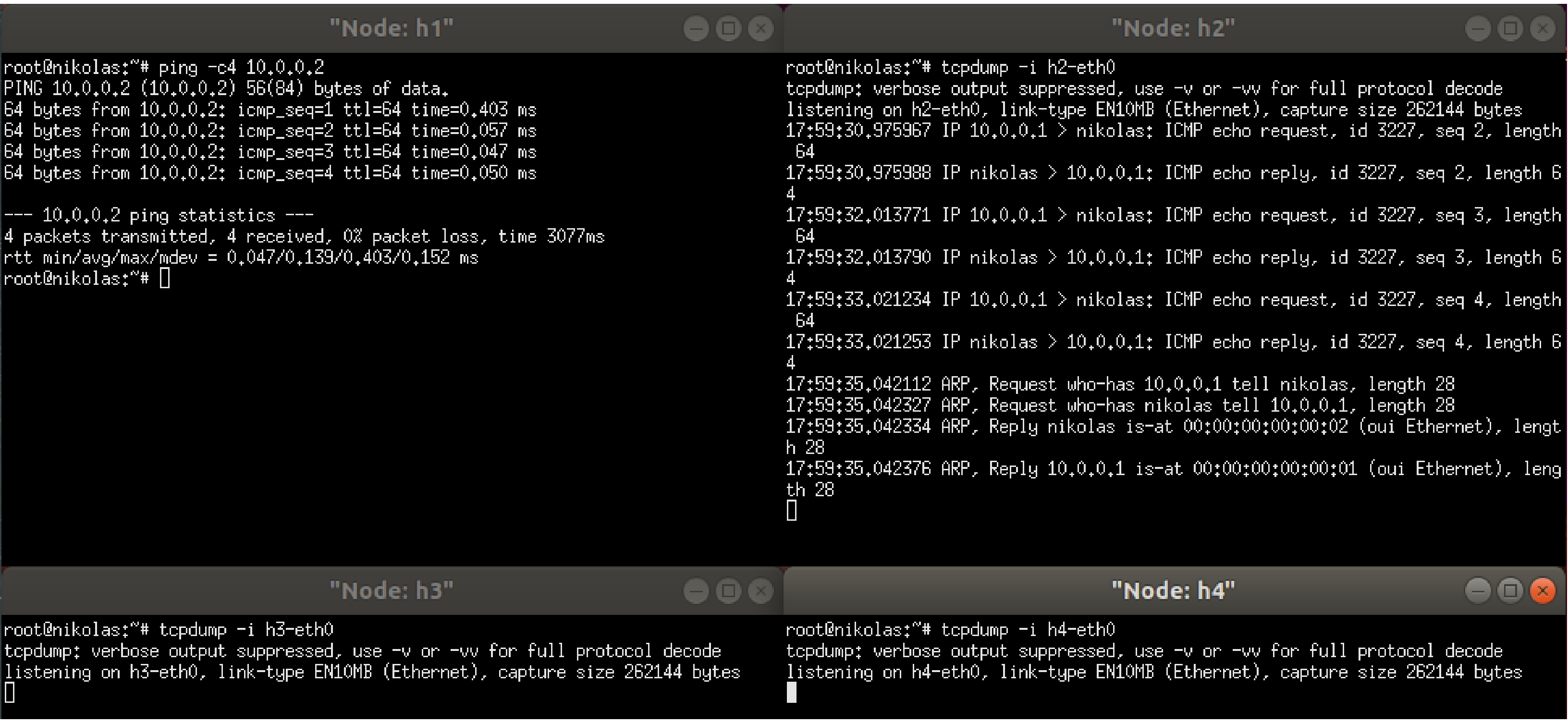
<u>Part 2: Switch</u>

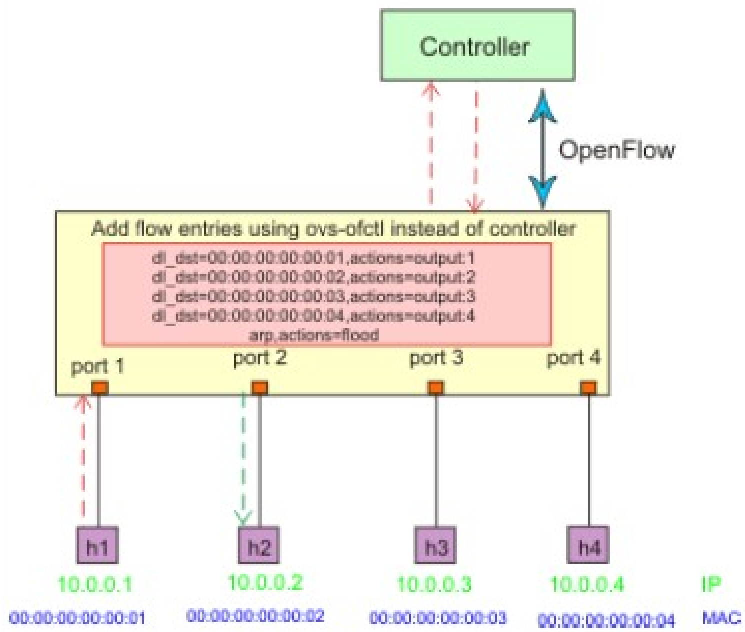
*Create the following **switch** application using **ovs-ofctl**.*

- *Show a screenshot proving the five installed flow entries in the switch.*

```
mininet> sh ovs-ofctl dump-flows s1
mininet>
mininet> sh ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:01,actions=output:1
mininet>
mininet> sh ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:02,actions=output:2
mininet>
mininet> sh ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:03,actions=output:3
mininet>
mininet> sh ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:04,actions=output:4
mininet>
mininet> sh ovs-ofctl add-flow s1 arp,actions=flood
mininet>
mininet> sh ovs-ofctl dump-flows s1
 cookie=0x0, duration=144.016s, table=0, n_packets=0, n_bytes=0, dl_dst=00:00:00:00:00:01
 actions=output:"s1-eth1"
 cookie=0x0, duration=120.972s, table=0, n_packets=0, n_bytes=0, dl_dst=00:00:00:00:00:02
 actions=output:"s1-eth2"
 cookie=0x0, duration=96.712s, table=0, n_packets=0, n_bytes=0, dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
 cookie=0x0, duration=79.176s, table=0, n_packets=0, n_bytes=0, dl_dst=00:00:00:00:00:04
actions=output:"s1-eth4"
 cookie=0x0, duration=38.299s, table=0, n_packets=0, n_bytes=0, arp actions=FLOOD
mininet>
```

- *Show a screenshot that verifies the switch behaviour when h1 pings h2 (showing all 4 hosts/terminals).*





Part 3: Firewall

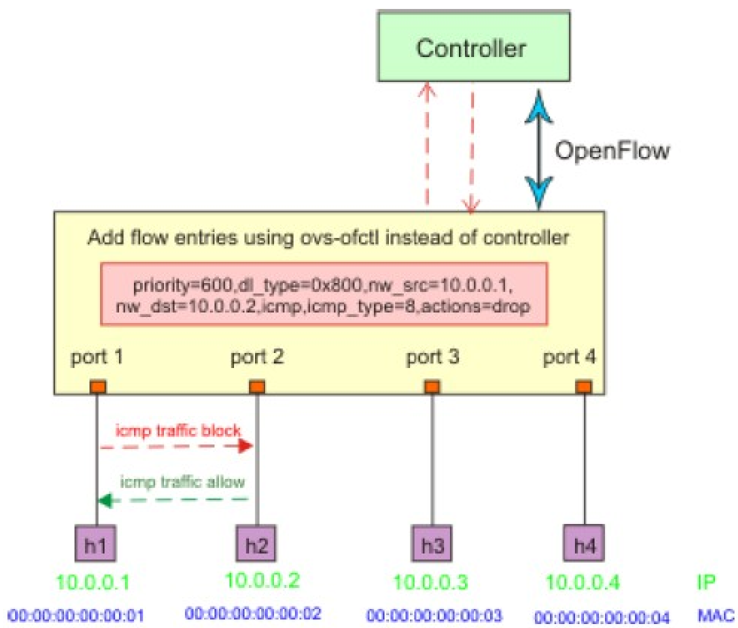
*Using the logic of the creation of the switch application using ovs-ofctl, create a **firewall** application, as shown in the next figure (i.e., h1 should not be able to ping h2 but h2 should be able to ping h1).*

- *Show a screenshot proving all installed flow entries in the switch.*

```
mininet> sh ovs-ofctl dump-flows s1
mininet>
mininet> sh ovs-ofctl add-flow s1 priority=500,dl_dst=00:00:00:00:00:01,actions=output:1
mininet> sh ovs-ofctl add-flow s1 priority=500,dl_dst=00:00:00:00:00:02,actions=output:2
mininet> sh ovs-ofctl add-flow s1 priority=500,dl_dst=00:00:00:00:00:03,actions=output:3
mininet> sh ovs-ofctl add-flow s1 priority=500,dl_dst=00:00:00:00:00:04,actions=output:4
mininet> sh ovs-ofctl add-flow s1 priority=500,arp,actions=flood
mininet>
mininet> sh ovs-ofctl add-flow s1 priority=600,dl_type=0x800,nw_src=10.0.0.1,nw_dst=10.0.0.2,icmp,icmp_ty
pe=8,actions=drop
mininet>
mininet> sh ovs-ofctl dump-flows s1
 cookie=0x0, duration=3.730s, table=0, n_packets=0, n_bytes=0, priority=600,icmp,nw_src=10.0.0.1,nw_dst=1
0.0.0.2,icmp_type=8 actions=drop
 cookie=0x0, duration=159.103s, table=0, n_packets=0, n_bytes=0, priority=500,dl_dst=00:00:00:00:00:01 ac
tions=output:"s1-eth1"
 cookie=0x0, duration=153.933s, table=0, n_packets=0, n_bytes=0, priority=500,dl_dst=00:00:00:00:00:02 ac
tions=output:"s1-eth2"
 cookie=0x0, duration=149.107s, table=0, n_packets=0, n_bytes=0, priority=500,dl_dst=00:00:00:00:00:03 ac
tions=output:"s1-eth3"
 cookie=0x0, duration=143.512s, table=0, n_packets=0, n_bytes=0, priority=500,dl_dst=00:00:00:00:00:04 ac
tions=output:"s1-eth4"
 cookie=0x0, duration=118.724s, table=0, n_packets=0, n_bytes=0, priority=500,arp actions=FLOOD
```

- *Verify the firewall behaviour using pingall and copy the screenshot here.*

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 8% dropped (11/12 received)
mininet>
```



Are we able to ping from h2 to h1 (i.e., opposite direction)? How can we control this?
Yes I am able to ping from h2 to h1. We can control this through the type

Hint: First create all 5 flow table entries, as in the switch application, with "priority=500". Then, to add the firewall rules, add a new flow table entry with higher priority (e.g. priority=600), with the corresponding nw_src and nw_dst addresses, as well as the respective action to drop packets (**consult the red box in the figure**).