

«به نام پروردگار»

# گزارش پروژه دوم شبکه های کامپیوتری

«OpenFlow-tutorial»

استاد: دکتر پویا حجازی

دانشجو: محسن محمدیان

شماره دانشجویی: 9831502

## بررسی اجمالی:

این پروژه به مفاهیمی چون SDN controller و مدیریت شبکه به صورت متمرکز یا centralized می پردازد و برای این کار از رابط open flow (interface) استفاده می کند که به کمک آن ما می توانیم روترها و سویچ ها را program کنیم و به کمک آن جدول های مسیریابی یا flow table های این تجهیزات را مدیریت کنیم.

همانطور که می دانیم open flow از 4 درایه یا ویژگی تشکیل شده است:

1. Pattern: ساختار فیلدهای مختلف در header های لایه های مختلف که هنگام رسیدن یک datagram به روتر، هدرهای آن با این ساختارها یا Match می شود یا خیر.
  2. Actions: drop, modify, forward, ...
  3. Priority: هنگامی که یک datagram با چندین هدر Match می شود، این فیلد اولویت pattern ها را نسبت به یکدیگر مشخص می کند. (ممکن هست مثلاً با دو pattern تطبیق یابد و در این حالت عمل نسبت به آن pattern ای که اولویت بالاتری دارد باید انجام شود)
  4. برای log گیری از بسته ها استفاده می شود.
- همچنین open flow امکاناتی نظیر امنیت بیشتر شبکه های default-off، شبکه های بی سیم بدون انتقال دستی، شبکه های مرکز داده مقیاس پذیر (scalable data center networks) و host mobility و... را فراهم می آورد.

## بخش اول:

### نیازمندی های نرم افزاری و سخت افزاری:

این بخش به ما اطلاعاتی نظیر حداقل RAM و حافظه ی مورد نیاز و سیستم عامل هایی که این پروژه روی آنها قابل پیاده سازی است را می دهد.

## بخش دوم:

### نصب virtual box و سایر برنامه های مورد نیاز:

چون از قبل این برنامه را در سیستم من نصب شده بود، تنها نیاز به دانلود و نصب سرور Xming و Putty terminal پیدا شد؛ که هر دو را نصب و راه اندازی کردم.

## بخش سوم:

### راه اندازی ماشین مجازی:

این بخش درباره کار با vBox و راه اندازی یک ماشین مجازی با import کردن فایل image موردنظر می باشد؛ که همانطور که در دستور کار ذکر شده، فایل tester.ova را import می کنیم و ماشین مجازی خود را بالا می آوریم.

یک کار دیگر نیز که در این مرحله باید انجام دهیم، باید در قسمت network، یک Ethernet interface دیگر نیز برای ماشین خود ایجاد کنیم و "Host-only adapter" را به آن attach کنیم.

با دستور `ifconfig -a` می توانیم interface های ماشین خود را مشاهده کنیم و اگر به آن ها آدرس ip اختصاص نیافته بود، با دستور `sudo dhclient ethX` که X شماره interface می باشد، می توانیم به آن آدرس ip اختصاص دهیم.

```
mininet@mininet-vm:~$ sudo dhclient eth1
mininet@mininet-vm:~$ ifconfig -a
eth0      Link encap:Ethernet  HWaddr 08:00:27:6e:79:9d
          inet addr:192.168.56.101 Bcast:192.168.56.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe6e:799d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2540 (2.5 KB)  TX bytes:1956 (1.9 KB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:b2:01:1c
          inet addr:10.0.2.15 Bcast:10.0.2.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:feb2:11c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:19 errors:0 dropped:0 overruns:0 frame:0
          TX packets:25 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3386 (3.3 KB)  TX bytes:2536 (2.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:698 errors:0 dropped:0 overruns:0 frame:0
          TX packets:698 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:56212 (56.2 KB)  TX bytes:56212 (56.2 KB)

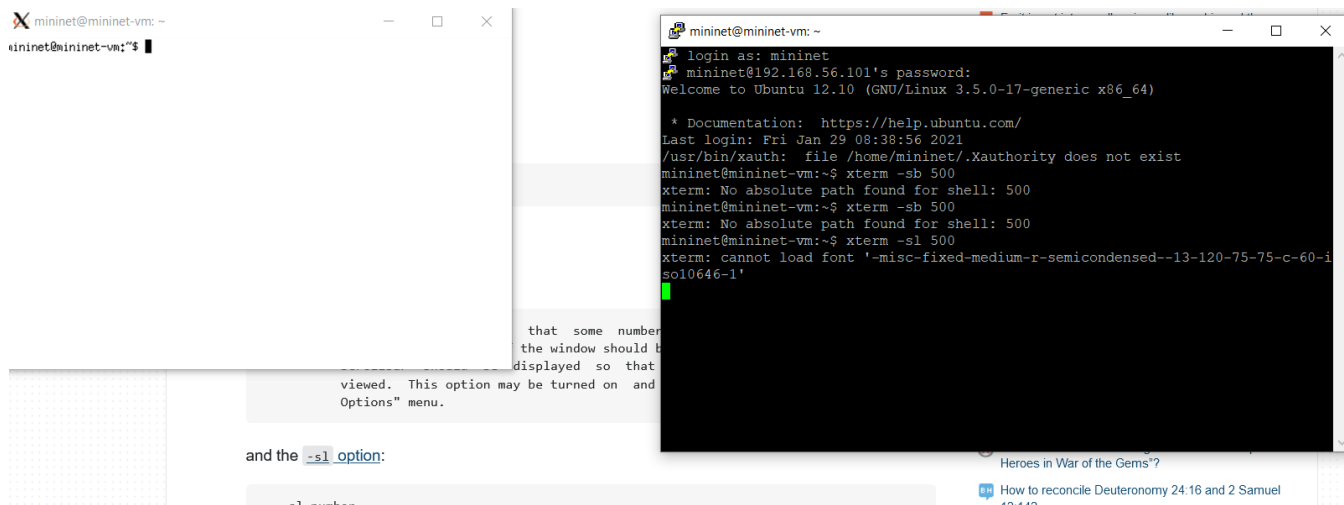
mininet@mininet-vm:~$
```

همانطور که مشاهده می شود، سه interface برای ما نمایش داده می شود که `eth0`، رابط NAT ما می باشد و آدرس 192.168.56.101 را دارد و `eth1`، رابط host-only ما می باشد و آدرس 10.0.2.15 را دارد.

### دستیابی به VM از طریق SSH:

برای استفاده از برنامه های X11، نظیر `wireshark` و `xterm`، باید ابتدا سرور `Xming` اجرا شود و یک ارتباط `ssh` همراه با `X11 forwarding`، برقرار شود.

بنابراین ابتدا سرور `Xming` را اجرا می کنیم و سپس فایل `Putty.exe` را به صورت ادمین اجرا می کنیم و آدرس رابط NAT خود که در بالا ذکر شده را به آن داده و `X11 forwarding` را `enable` می کنیم و `enter` را می زنیم تا ترمینال `mininet` باز شود. پس از برقراری `SSH connection`، در ترمینال مربوطه دستور `xterm -sl 500` را وارد می کنیم تا یک ترمینال X باز شود (به رنگ سفید)



آشنایی با ابزارهای توسعه دهنده:

توضیح مختصر درباره Mininet:

ابتدا با دستور

```
sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

یک توپولوژی با سه host و یک switch و یک controller را می‌سازیم. کاری که دستور فوق انجام می‌دهد، شامل پنج بخش می‌شود:

1. سه تا host می‌سازد که هرکدام آدرس IP جداگانه‌ای خود را دارند.
2. یک سویچ نرم افزاری OpenFlow با سه پورت را در کرنل ایجاد می‌کند.
3. هر host مجازی را با یک کابل Ethernet مجازی به سویچ متصل می‌کند.
4. آدرس MAC هر host را برابر با آدرس IP آن قرار می‌دهد.
5. سویچ OpenFlow را پیکربندی می‌کند تا به remote controller متصل شود.

دستورات گوناگونی در Mininet وجود دارد. مانند:

```
mininet> nodes
mininet> help
mininet> h1 ifconfig
mininet> xterm h1 h2
```

```

mininet@mininet-vm: ~
login as: mininet
mininet@192.168.56.101's password:
Welcome to Ubuntu 12.10 (GNU/Linux 3.5.0-17-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Fri Jan 29 05:51:58 2021 from 192.168.56.1
mininet@mininet-vm:~$ xterm -sl 500
xterm: cannot load font '-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-i
so10646-1'
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovsk --controller r
emote
*** 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
*** Starting 1 switches
s1
*** Starting CLI:
mininet>

```

### :dcctl

یک ابزار می باشد که همراه OpenFlow می آید و مشاهده پذیری و کنترل روی یک flow table یک سویچ فراهم می کند و بسیار کارا برای debugging به کمک مشاهده وضعیت flow ها و counter ها می باشد. حال یک پنجره ی SSH دیگر را می سازیم و در آن دستورات زیر را وارد می کنیم.

```

dpctl show tcp:127.0.0.1:6634
dpctl dump-flows tcp:127.0.0.1:6634

```

دستور show در بالا به یک سویچ متصل می شود و پورت و گنجایش های آن سویچ را نمایش می دهد.

همانطور که در تصویر زیر مشاهده می شود، چون هنوز ما controller ای را راه اندازی نکرده ایم، flow table خالی می باشد.

```

mininet@mininet: ~
login as: mininet
mininet@192.168.56.102's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-23-generic i686)

 * Documentation:  https://help.ubuntu.com/

System information as of Fri Jan 29 08:30:50 PST 2021

System load:  0.0               Processes:            83
Usage of /:   9.4% of 18.94GB   Users logged in:     1
Memory usage: 13%              IP address for eth0: 10.0.2.15
Swap usage:   0%               IP address for eth2: 192.168.56.102

Graph this data and manage this system at https://landscape.canonical.com/

New release '14.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Fri Jan 29 08:29:09 2021 from 192.168.56.1
mininet@mininet:~$ dpctl show tcp:127.0.0.1:6634
features_reply (xid=0xa2805db8): ver:0x1, dpid:1
n_tables:255, n_buffers:256
features: capabilities:0xc7, actions:0xffff
1(s1-eth1): addr:3e:1b:c7:5f:3c:f8, config: 0, state:0
current:    10GB-FD COPPER
2(s1-eth2): addr:da:cd:df:b2:17:ba, config: 0, state:0
current:    10GB-FD COPPER
3(s1-eth3): addr:62:dd:00:2a:6b:7f, config: 0, state:0
current:    10GB-FD COPPER
LOCAL(s1):  addr:d6:cd:7c:47:3f:44, config: 0x1, state:0x1
get_config_reply (xid=0x3cc16d5b): miss send len=0
mininet@mininet:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x3e176f1b): flags:none type=1(flow)
mininet@mininet:~$

```

### تست Ping:

اگر همین حالا از h1، h3 را ping کنیم، همانطور که در عکس زیر مشاهده می شود، همه ی packet ها Lost خواهند شد چراکه هیچ flow ای میان این دو host تعریف نشده است.

```

mininet@mininet: ~
*** 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
*** Starting 1 switches
s1
*** Starting CLI:
mininet> h1 ping -c3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 1999ms
pipe 3
mininet>

```

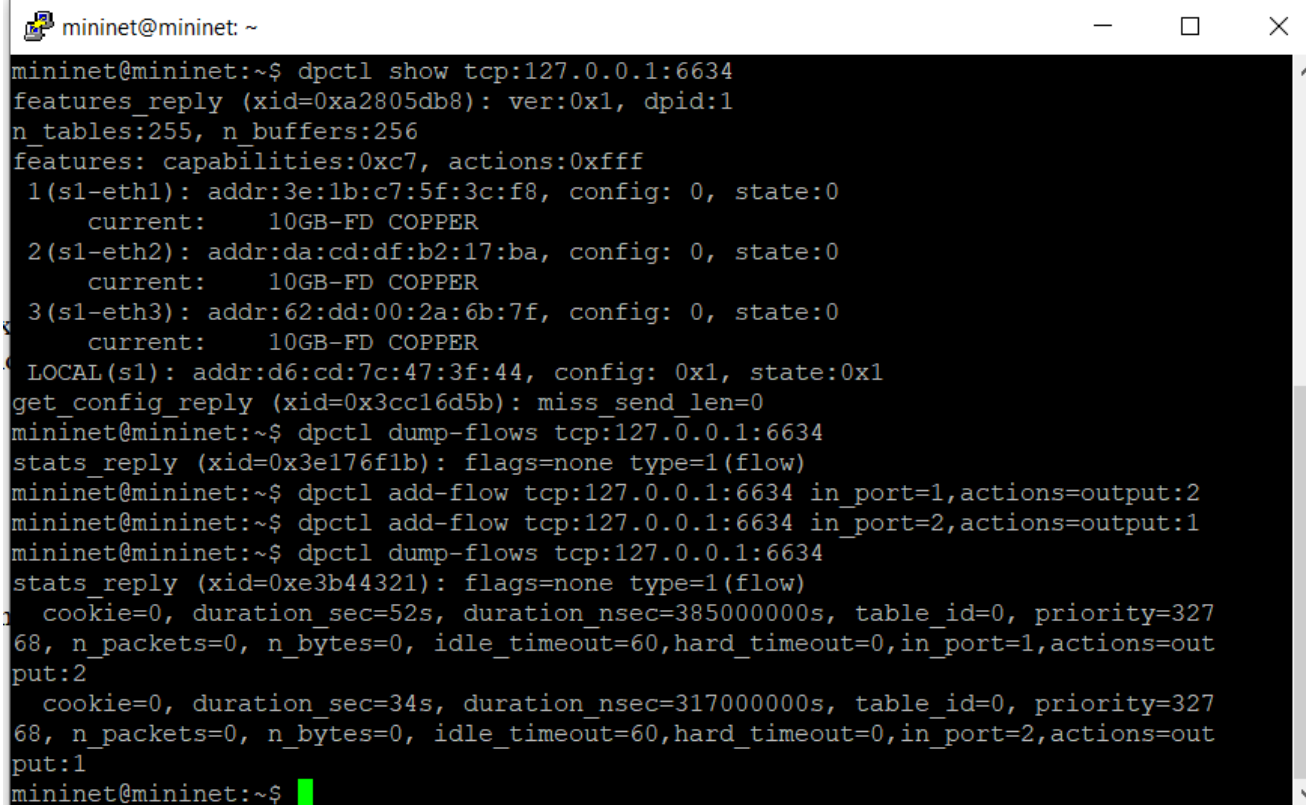
حال در ترمینال دو دستور زیر را وارد می کنیم.

```
$ dpctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=output:2
$ dpctl add-flow tcp:127.0.0.1:6634 in_port=2,actions=output:1
```

حال این دو دستور باعث می شود که بسته های ورودی به پورت شماره 2 سوییچ روی پورت 1 ارسال شوند و بالعکس.

دوباره جدول flow خود را مشاهده می کنیم:

```
$ dpctl dump-flows tcp:127.0.0.1:6634
```



```
mininet@mininet:~$ dpctl show tcp:127.0.0.1:6634
features_reply (xid=0xa2805db8): ver:0x1, dpid:1
n_tables:255, n_buffers:256
features: capabilities:0xc7, actions:0xffff
  1(s1-eth1): addr:3e:1b:c7:5f:3c:f8, config: 0, state:0
    current: 10GB-FD COPPER
  2(s1-eth2): addr:da:cd:df:b2:17:ba, config: 0, state:0
    current: 10GB-FD COPPER
  3(s1-eth3): addr:62:dd:00:2a:6b:7f, config: 0, state:0
    current: 10GB-FD COPPER
LOCAL(s1): addr:d6:cd:7c:47:3f:44, config: 0x1, state:0x1
get_config_reply (xid=0x3cc16d5b): miss_send_len=0
mininet@mininet:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x3e176f1b): flags=none type=1(flow)
mininet@mininet:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=output:2
mininet@mininet:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=2,actions=output:1
mininet@mininet:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0xe3b44321): flags=none type=1(flow)
  cookie=0, duration_sec=52s, duration_nsec=385000000s, table_id=0, priority=327
68, n_packets=0, n_bytes=0, idle_timeout=60,hard_timeout=0,in_port=1,actions=out
put:2
  cookie=0, duration_sec=34s, duration_nsec=317000000s, table_id=0, priority=327
68, n_packets=0, n_bytes=0, idle_timeout=60,hard_timeout=0,in_port=2,actions=out
put:1
mininet@mininet:~$
```

حال اگر دوباره Ping کنیم، داریم:

```
mininet> h1 ping -c3 h2
```

```

mininet@mininet: ~
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2016ms
pipe 3
mininet> h1 ping -c3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2014ms
pipe 3
mininet> h1 ping -c3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=0.375 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.057 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.053 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.053/0.161/0.375/0.151 ms
mininet>

```

همانطور که در تصویر بالا مشاهده می شود به دلیل آنکه به صورت دستی روی سویچ دو flow را تعریف کردیم، بسته ها به درستی به مقصد می رسند و reply آن ها به میزبان 1 می رسد.

در تصویر زیر نیز اگر مشاهده کنید ابتدا flow entry ها در جدول وجود دارند، اما پس از expire شدن زمان آنها از جدول حذف می شوند.

```

mininet@mininet: ~
mininet@mininet:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x9acf9216): flags=none type=1(flow)
  cookie=0, duration_sec=38s, duration_nsec=163000000s, table_id=0, priority=327
  cookie=0, duration_sec=14s, duration_nsec=800000000s, table_id=0, priority=327
mininet@mininet:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x5b7f7f8a): flags=none type=1(flow)
mininet@mininet:~$

```

### :Start Wireshark

ابتدا یک ترمینال دیگر را باز کرده و با دستور `sudo wireshark &` وایرشارک را اجرا می کنیم. سپس ترافیک های کنترلی OpenFlow را با وارد کردن of در قسمت فیلتر، فیلتر می کنیم. حال درحالی که وایرشارک درحال شنود می باشد، با دستور `controller ptcp` در `ssh terminal`، کنترلر را اجرا می کنیم.



پیغام های متفاوتی را نظیر Hello message (همان tcp handshaking)، 'set config'، Features request و... را دریافت و مشاهده می کنیم.

The image displays two screenshots. The left screenshot shows a Wireshark packet capture from interface 'lo' on host '127.0.0.1'. The packet list shows several messages: Hello, Features Request, Set Config, Features Reply, and Port Status. The packet details pane for the 'Set Config' packet (No. 5947) shows an OpenFlow Protocol packet with a duration of 127.0.0.1. The right screenshot shows a Mininet terminal window with the following commands and output:

```

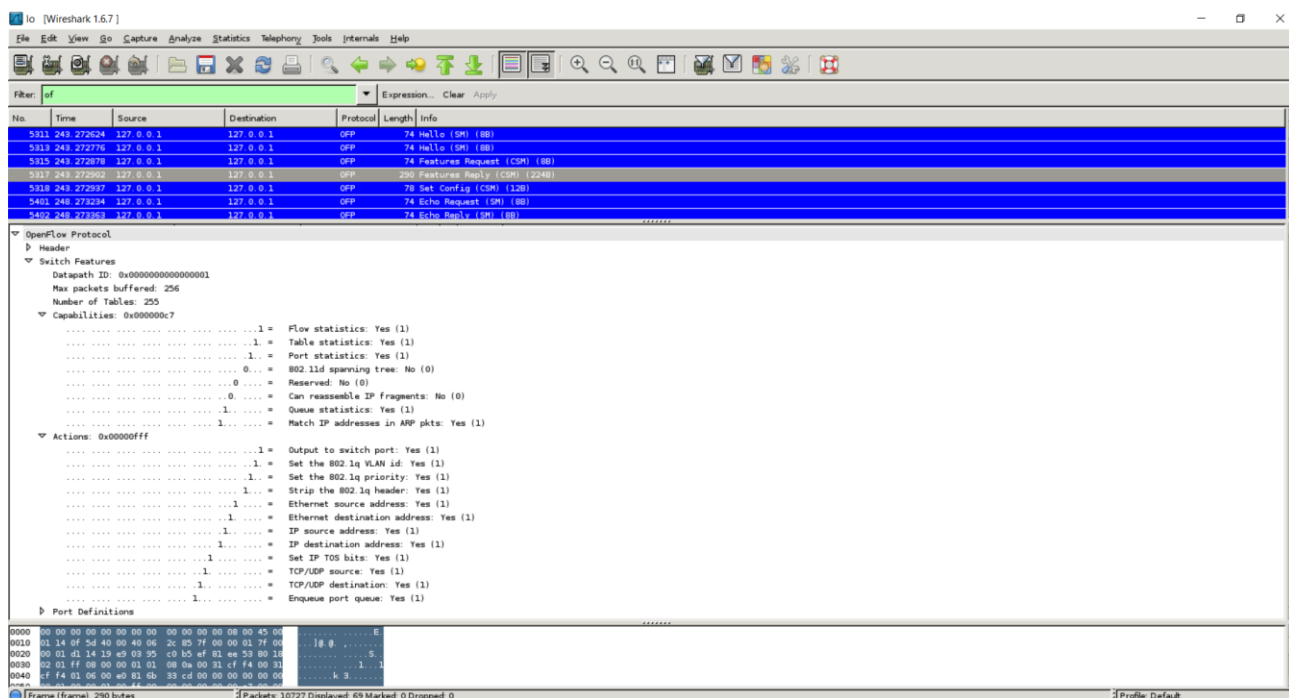
mininet@mininet:~$ stats reply (xid=0x5f629ca): flags=none type=1(flow)
  cookie=0, duration sec=55s, duration_nsec=950000000, table_id=0, priority=32768, n
  packets=0, n_bytes=0, idle_timeout=120,hard_timeout=0,in_port=2,actions=output:1
mininet@mininet:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=1,idle_timeout=120,acti
ns=output:2
mininet@mininet:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats reply (xid=0x38d3be87): flags=none type=1(flow)
  cookie=0, duration sec=8s, duration_nsec=883000000, table_id=0, priority=32768, n
  packets=0, n_bytes=0, idle_timeout=120,hard_timeout=0,in_port=1,actions=output:2
  cookie=0, duration sec=102s, duration_nsec=910000000, table_id=0, priority=32768, n
  packets=0, n_bytes=0, idle_timeout=120,hard_timeout=0,in_port=2,actions=output:1
mininet@mininet:~$ h1 ping -c3 h2
h1: command not found
mininet@mininet:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=2,idle_timeout=240,acti
ns=output:1
mininet@mininet:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=1,idle_timeout=2400,acti
ons=output:2
mininet@mininet:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats reply (xid=0x966e38f): flags=none type=1(flow)
  cookie=0, duration sec=11s, duration_nsec=326000000, table_id=0, priority=32768, n
  packets=0, n_bytes=0, idle_timeout=2400,hard_timeout=0,in_port=1,actions=output:2
  cookie=0, duration sec=26s, duration_nsec=445000000, table_id=0, priority=32768, n
  packets=0, n_bytes=0, idle_timeout=240,hard_timeout=0,in_port=2,actions=output:1
mininet@mininet:~$ controller ptcp:

```

خلاصه ی پیام ها و توضیحات آنها را می توان در جدول زیر مشاهده کرد:

Message	Type	Description
<b>Hello</b>	Controller->Switch	following the TCP handshake, the controller sends its version number to the switch.
<b>Hello</b>	Switch->Controller	the switch replies with its supported version number.
<b>Features Request</b>	Controller->Switch	the controller asks to see which ports are available.
<b>Set Config</b>	Controller->Switch	in this case, the controller asks the switch to send flow expirations.
<b>Features Reply</b>	Switch->Controller	the switch replies with a list of ports, port speeds, and supported tables and actions.
<b>Port Status</b>	Switch->Controller	enables the switch to inform that controller of changes to port speeds or connectivity. Ignore this one, it appears to be a bug.

همانطور که در تصویر زیر نیز مشاهده می شود، با استفاده از این برنامه می توان اطلاعات بسیاری را درباره بسته ها بدست آورد. برای مثال در تصویر زیر در پیغام Features reply می توان ظرفیت های datapath را مشاهده کرد. اینکه flow تعریف شده ایستا است یا پویا، جدول سوئیچ ایستا است یا خیر و پورت ایستا است یا خیر و ... .



### مشاهده پیام های OpenFlow حاصل از Ping:

ابتدا برای چشم پوشی از echo-request/reply message، در قسمت فیلتر عبارت زیر را تایپ کرده و enter را می زنیم.

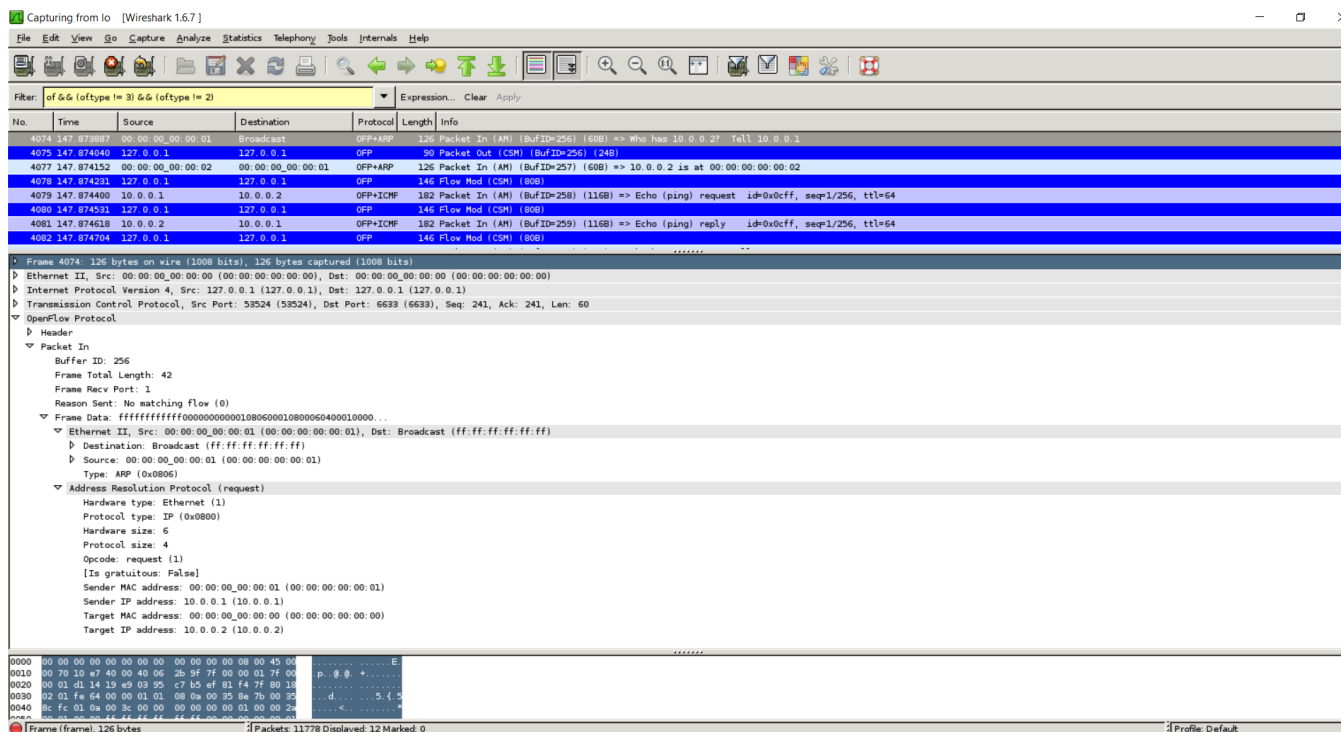
```
of && (of.type != 3) && (of.type != 2)
```

حال از h1، h2 را یکبار Ping می کنیم، پیام هایی مطابق جدول زیر را مشاهده خواهیم کرد.

Message	Type	Description
<b>Packet-In</b>	Switch->Controller	a packet was received and it didn't match any entry in the switch's flow table, causing the packet to be sent to the controller.
<b>Packet-Out</b>	Controller->Switch	controller send a packet out one or more switch ports.
<b>Flow-Mod</b>	Controller->Switch	instructs a switch to add a particular flow to its flow table.
<b>Flow-Expired</b>	Switch->Controller	a flow timed out after a period of inactivity.

در ابتدا یک پیغام ARP را مشاهده می کنیم، که در miss flow table شده و به همین دلیل broadcast شده. در گام بعدی یک پاسخ پیام ARP دریافت می شود که حالا هم آدرس MAC مبدا و هم مقصد برای کنترلر شناخته شده است.

سوییچ می تواند flow هایی را برای پیام های ICMP ایجاد کند و درخواست های ping در این حالت مستقیماً از طریق dataPath منتقل می شوند و هیچ پیام اضافه را شامل نمی شوند.



## Benchmark Controller w/iperf:

iperf، یک ابزار cmd برای بررسی سرعت میان دو کامپیوتر می باشد.

```
mininet> iperf
```

اگر دستور بالا را برای توپولوژی حاضر و همچنین یک توپولوژی شبیه توپولوژی حاضر، منتهی با یک سویچ فضای کاربر وارد کنیم، مشاهده خواهیم کرد که سرعت تبادل اطلاعات در سویچ فضای کاربر بسیار پایین تر خواهد بود؛ چراکه در این سویچ، بسته ها مدام باید میان فضای کاربر و فضای کرنل در هر گام (hop) رد و بدل شوند بجای آنکه هنگامی که از سویچ مربوطه گذر می کنند، در همان کرنل بمانند.

سویچ فضای کاربر برای دست کاری و تغییرات آسان تر است، اما شبیه سازی آن کند تر از سویچ کرنل می باشد.

تصویر اول مربوط به سویچ کرنل و تصویر دوم مربوط به سویچ فضای کاربر می باشد.

```

mininet@mininet: ~
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
waiting for iperf to start up...*** Results: ['1.88 Gbits/sec', '1.88 Gbits/sec']
mininet>

mininet@mininet:~$ sudo mn --topo single,3 --controller remote --switch user
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
waiting for iperf to start up...*** Results: ['478 Mbits/sec', '478 Mbits/sec']

```

## بخش پنجم:

### ساخت سویچ مبتنی بر یادگیری:

در این بخش هدف برنامه کردن یک سویچ هست که این سویچ بررسی می کند اگر آدرس MAC مقصد بسته با پورتهی تناظر داشت، از آن پورت بسته را خارج می کند ولی اگر تناظر نداشت، آن بسته را broadcast می کند.

در ادامه نیز این سویچ را بیشتر توسعه داده و یک سویچ بر مبنای flow در واقع خواهیم داشت.

### انتخاب کنترلر: POX

در ابتدا با وارد کردن ctrl+D در برنامه کنترلر یا زدن دستور sudo killall controller در ترمینال ssh، کنترلر ارجاعی را پایان می بخشیم.

سپس با دستور exit از mininet خارج شده و با دستور sudo mn -c مطمئن می شویم همه چیز پاک شده و mininet را در واقع restart می کنیم.

سپس با دستور

```
sudo mn --topo single,3 --mac --controller remote --switch ovsk
```

توپولوژی جدید را می سازیم.

چون mininet ما POX را به طور پیش فرض نصب شده دارد، تنها با دستور

```
./pox.py log.level --DEBUG misc.of_tutorial
```

یک hub ساده و مبنا را اجرا می کنیم.

این دستور به POX می گوید تا verbose logging را enable کند و فایل of\_tutorial را که ما قرار هست آنرا توسعه دهیم را، اجرا کند. (که در حال حاضر مانند یک hub عمل می کند)

```

mininet@mininet: ~/pox
login as: mininet
mininet@192.168.56.103's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-23-generic i686)

 * Documentation:  https://help.ubuntu.com/

System information as of Sat Jan 30 06:56:16 PST 2021

System load:  0.03          Processes:      74
Usage of /:   8.8% of 18.94GB    Users logged in:  1
Memory usage: 36%          IP address for eth0: 10.0.2.15
Swap usage:  0%             IP address for eth2: 192.168.56.103

Graph this data and manage this system at https://landscape.canonical.com/

276 packages can be updated.
196 updates are security updates.

New release '14.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sat Jan 30 06:54:28 2021 from 192.168.56.1
mininet@mininet:~$ sudo killall controller
controller: no process found
mininet@mininet:~$ git clone http://github.com/noxrepo/pox
fatal: destination path 'pox' already exists and is not an empty directory.
mininet@mininet:~$ cd pox
mininet@mininet:~/pox$ ./pox.py log.level --DEBUG misc.of_tutorial
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.3/Apr 10 2013 05:46:21)
DEBUG:core:Platform is Linux-3.5.0-23-generic-i686-athlon-with-Ubuntu-12.04-precise
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 1]
INFO:openflow.of_01:[00-00-00-00-00-01 1] closed
INFO:openflow.of_01:[None 2] closed
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 3]

```

### تایید رفتار hub با tcpdump:

حال برای آنکه تایید کنیم همه ی host می توانند یکدیگر را Ping کنند و پیام های یکدیگر را دریافت کنند، برای هر host ما یک xterm می سازیم و ترافیک را روی هر کدام مشاهده می کنیم.

ابتدا دستور

```
mininet> xterm h1 h2 h3
```

را وارد کرده و سپس در xterm مربوط به h2 و h3، دستورات زیر را به ترتیب وارد می کنیم.

```
# tcpdump -XX -n -i h2-eth0
# tcpdump -XX -n -i h3-eth0
```

و در نهایت در xterm مربوط به h1، میزبان یا h2 را ping می کنیم با دستور:

```
# ping -c1 10.0.0.2
```

همانطور که در تصویر زیر مشاهده می شود، ping req برای هر دو host دو و سه ارسال می شود. و ترافیک های دریافتی آنها دقیقا مشابه یکدیگر می باشد.

```

Node: h2
0x0020: 0002 0800 b24b 1a04 0001 1a5f 1560 08ed .....K.....
0x0030: 0800 0809 0a0b 0c0d 0e0f 1011 1213 1415 .....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!###%
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 67
05:28:58.629586 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 6660, seq 1, length 64
0x0000: 0000 0000 0001 0000 0000 0002 0800 4500 .....E.
0x0010: 0054 897f 0000 4001 dd27 0a00 0002 0a00 .T....@.....
0x0020: 0001 0000 ba4b 1a04 0001 1a5f 1560 08ed .....K.....
0x0030: 0800 0809 0a0b 0c0d 0e0f 1011 1213 1415 .....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!###%
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 67
05:29:03.641077 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
0x0000: 0000 0000 0001 0000 0000 0002 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0002 0a00 0002 .....
0x0020: 0000 0000 0000 0a00 0001 .....
05:29:03.681388 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
0x0000: 0000 0000 0002 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0002 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0002 0a00 0002 .....

Node: h3
0x0020: 0002 0800 b24b 1a04 0001 1a5f 1560 08ed .....K.....
0x0030: 0800 0809 0a0b 0c0d 0e0f 1011 1213 1415 .....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!###%
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 67
05:28:58.632160 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 6660, seq 1, length 64
0x0000: 0000 0000 0001 0000 0000 0002 0800 4500 .....E.
0x0010: 0054 897f 0000 4001 dd27 0a00 0002 0a00 .T....@.....
0x0020: 0001 0000 ba4b 1a04 0001 1a5f 1560 08ed .....K.....
0x0030: 0800 0809 0a0b 0c0d 0e0f 1011 1213 1415 .....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!###%
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 67
05:29:03.679904 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
0x0000: 0000 0000 0001 0000 0000 0002 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0002 0a00 0002 .....
0x0020: 0000 0000 0000 0a00 0001 .....
05:29:03.681382 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
0x0000: 0000 0000 0002 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0002 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0002 0a00 0002 .....

```

them out all interfaces except the sending one

1 xterm:

. If your code is off later, three unanswered AR

IX hub in a second window. In the Mininet con

```

Node: h1
root@mininet:~# ping -c 1 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_req=1 ttl=64 time=47.1 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 47.195/47.195/47.195/0.000 ms
root@mininet:~#

```

g in both xterms running tcpdump. Thi

ولی اگر یک آدرس ناشناخته مانند 10.0.0.5 را ping کنیم، مشاهده خواهیم کرد که بسته ارسال lost می شود.

```

Node: h2
root@mininet:~# tcpdump -XX -n -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 65535 bytes
^

Node: h3
root@mininet:~# tcpdump -XX -n -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 65535 bytes
^

Node: h1
root@mininet:~# ping -c1 10.0.0.5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data:
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
--- 10.0.0.5 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
root@mininet:~#

```

## :Benchmark Hub Controller w/iperf

حال باید دسترسی ها را بررسی کنیم :

```

mininet@mininet: ~
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (0/6 lost)
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['7.27 Mbits/sec', '7.58 Mbits/sec']
mininet>

```

همانطور که مشاهده می شود، به دلیل باز تبادل پیام ها میان سوییچ و کنترلر سرعت ما کاهش یافته نسبت به حالت قبل.

## بازکردن کد Hub:

ابتدا در مسیر `pox/pox/misc`، فایل `of_tutorial.py` را با دستور `sudo vi of_tutorial.py` باز کرده و شروع به ادیت تابع `act_like_switch()` می کنیم.

```
mininet@mininet: ~/pox/pox/misc

# Note that if we didn't get a valid buffer_id, a slightly better
# implementation would check that we got the full data before
# sending it (len(packet_in.data) should be == packet_in.total_len)).

def act_like_switch(self, packet, packet_in):
    """
    Implement switch-like behavior.
    """

    # Learn the port for the source MAC
    in_port = packet_in.in_port
    dl_src = str(packet.src)
    dl_dst = str(packet.dst)
    self.mac_to_port[dl_src] = in_port

    if dl_dst in self.mac_to_port:
        # Send packet out the associated port
        dst_in_port = self.mac_to_port[dl_dst]
        self.resend_packet(packet_in, dst_in_port)

        # Once you have the above working, try pushing a flow entry
        # instead of resending the packet (comment out the above and
        # uncomment and complete the below.)

        log.debug(
            "Installing flow for mac address {} on port {}. Destination mac is {}".format(dl_src, in_port, dl_dst))
        # Maybe the log statement should have source/destination/port?

        msg = of.ofp_flow_mod()

        # Set fields to match received packet
        # msg.match = of.ofp_match.from_packet(packet)
        msg.match = of.ofp_match.from_packet(packet)

        # < Set other fields of flow_mod (timeouts? buffer_id?) >

        # < Add an output action, and send -- similar to resend_packet() >
        action = of.ofp_action_output(port=dst_in_port)
        msg.actions.append(action)

        # log.debug(msg)
        self.connection.send(msg)

    else:
        # Flood the packet out everything but the input port
        # This part looks familiar, right?
        self.resend_packet(packet_in, of.OFPP_ALL)
```

در این قسمت همانطور که مشاهده می شود، ابتدا ما پورت ورودی یک بسته را استخراج می کنیم و سپس آدرس Mac مبدا بسته را در `dl_src` و آدرس مقصد آن را در `dl_dst` می ریزیم و در نهایت این جفت را به عنوان یک درایه ی جدید به دیکشنری `mac_to_port` اضافه می کنیم.

اگر آدرس مقصد بسته در دیکشنری ما موجود بود، پورت مقصد به راحتی همان پورت متناظر با آدرس مقصد خواهد شد و بدست می آید و در نهایت با فراخوانی تابع `resend` و دادن مقادیر پورت خروجی و بسته ی ورودی، بسته از پورت مناسب به سمت مقصد ارسال خواهد شد. در غیر اینصورت broadcast خواهد شد.

حال می توانیم به ازای هر بسته با مبدا و مقصد مشخص، یک `flow entry` به جدول `flow` خود اضافه کنیم تا دیگر مجبور نباشیم تمامی پیام ها را ابتدا به کنترلر بفرستیم و این کار پهنای باند ما را بسیار افزایش خواهد داد.



## تست کنترلر:

اکنون می‌توانیم با بهره‌گیری از POX و ادیتور Vim، کد of\_tutorial را تغییر داده و یک learning switch بسازیم؛ و در تصاویر زیر تست‌های انجام شده روی این learning switch را مشاهده می‌کنید.

The image displays three terminal windows from a Mininet environment, illustrating network testing procedures.

**Node: h1** terminal shows a successful ping command:

```
root@mininet:~# ping -c1 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_req=1 ttl=64 time=100 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 100.099/100.099/100.099/0.000 ms
root@mininet:~#
```

**Node: h2** terminal shows a series of network events, including an ICMP echo reply and ARP requests/replies:

```
09:44:53.429371 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 31546, seq 1, length 64
09:44:58.441084 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
09:44:58.498658 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
```

**Node: h3** terminal shows a tcpdump capture of the ARP request and reply:

```
root@mininet:~# tcpdump -XX -n -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 65535 bytes
09:44:53.383504 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
0x0000: 0000 0000 0001 0000 0000 0002 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0002 .....
09:44:58.498658 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
0x0000: 0000 0000 0002 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0002 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0002 0a00 0002 .....

```

The background shows a Mininet prompt with the command `mininet> pingall` and a status message: "his is just a sanity check for connecti". Below this, the command `mininet> iperf` is entered.

## Open Hub Code and Begin

io to your SSH terminal and stop the /of\_tutorial.py. Open  
his file in your favorite editor. Vim is a funky commands  
edit text on <http://vim.sourceforge.net> can be of use. To use vim, make sure you are in the correct directory (say /usr/bin/vim) and

Arrange screen. In the x # tc and res # tc In the x

```

Node: h3
root@mininet:~# tcpdump -n -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 65535 bytes
09:53:53.945632 ARP, Request who-has 10.0.0.3 tell 10.0.0.1, length 28
09:53:53.945654 ARP, Reply 10.0.0.3 is-at 00:00:00:00:00:03, length 28
09:53:53.989270 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 32340, seq 1, length 64
09:53:53.989303 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 32340, seq 1, length 64
09:53:59.001023 ARP, Request who-has 10.0.0.1 tell 10.0.0.3, length 28
09:53:59.093235 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
09:54:28.768922 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
09:54:29.744961 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
09:54:30.769419 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28

```

# ping -c1 10.0.0.2

The ping packets are now going up should see identical ARP and ICMP works; it sends all packets to every

Now, see what happens when a no

```

Node: h2
root@mininet:~# tcpdump -XX -n -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 65535 bytes
09:53:53.945633 ARP, Request who-has 10.0.0.3 tell 10.0.0.1, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0003 .....
09:54:28.768924 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0005 .....
09:54:29.744963 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0005 .....
09:54:30.769420 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0005 .....

```

```

Node: h1
root@mininet:~# ping -c1 10.0.0.3
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=83.8 ms
--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 83.802/83.802/83.802/0.000 ms
root@mininet:~# ping -c1 10.0.0.5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
--- 10.0.0.5 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
root@mininet:~#

```

# ping -c1 10.0.0.5

You should see three unanswered A is a signal that you might be accide

You can close the xterms now.

همانطور که در تصاویر بالا مشاهده می شود، دیگر پیام های مربوط به یک host خاص از تمامی پورت ها broadcast نمی شود و همینطور سه تا ARP Reply مربوط به ping آدرس ناشناخته ی 10.0.0.5 نیز کاملاً قابل مشاهده است.

برای pingall و iperf نیز داریم:

```
mininet@mininet:~/pox$ cd pox/misc
mininet@mininet:~/pox/pox/misc$ sudo vim of_tutorial.py
mininet@mininet:~/pox/pox/misc$ cd ~/pox
mininet@mininet:~/pox$ cd pox/misc
mininet@mininet:~/pox/pox/misc$ sudo vim of_tutorial.py
mininet@mininet:~/pox/pox/misc$ cd ~/pox
mininet@mininet:~/pox$ ./pox.py log.level --DEBUG misc.of tutorial
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.3/Apr 10 2013 05:46:21)
DEBUG:core:Platform is Linux-3.5.0-23-generic-i686-athlon-with-Ubuntu-12.04-precise
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01]

mininet@mininet:~$ sudo mn --topo single,3 --mac --controller remote --switch ovsk
*** 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
*** Starting 1 switches
s1
*** Starting CLI:
mininet> ping all
*** Unknown command: ping
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (0/6 lost)
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
waiting for iperf to start up...*** Results: ['9.13 Mbits/sec', '9.21 Mbits/sec']
mininet>
```

حال یک مرحله دیگر نیز پیش رفته و flow entry ها را ایجاد می کنیم و یک سوییچ بر مبنای flow را می سازیم. همانطور که مشاهده می شود نرخ انتقال و پهنای باند بسیار افزایش می یابد و شبیه حالتی می شود که ما از یک کرنل سوییچ ارجاعی در توپولوژی های قدیمی تر استفاده می کردیم. مطالب ذکر شده ی فوق در دو تصویر زیر قابل تصدیق است.

```
mininet@mininet:~$ ip link del s1-eth1
mininet@mininet:~$ ip link del s1-eth2
mininet@mininet:~$ ip link del s1-eth3
*** Cleanup complete.
mininet@mininet:~$ sudo mn --topo single,3 --mac --controller remote --switch ovsk
*** 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
*** Starting 1 switches
s1
*** Starting CLI:
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['1.96 Gbits/sec', '1.96 Gbits/sec']
mininet>
```



## بخش هشتم:

## ساخت firewall:

در این قسمت مانند قسمت پنجم ابتدا یک فایل مانند firewall.py را در مسیر ./pox/pox/misc ایجاد کرده و کد مورد نظر خود را در آن پیاده می کنیم.

قسمتی از کد در تصویر زیر آمده است:

```
mininet@mininet: ~/pox/pox/misc
MAX_BUFFERED_PER_IP = 5
# Maximum time to hang on to a buffer for an unknown IP in seconds
MAX_BUFFER_TIME = 1

class Entry(object):
    def __init__(self, port, mac):
        self.timeout = time.time() + ARP_TIMEOUT
        self.port = port
        self.mac = mac

    def __eq__(self, other):
        if type(other) == tuple:
            return (self.port, self.mac) == other
        else:
            return (self.port, self.mac) == (other.port, other.mac)
    def __ne__(self, other):
        return not self.__eq__(other)

    def isExpired(self):
        if self.port == of.OFPP_NONE: return False
        return time.time() > self.timeout

def dpid_to_mac(dpid):
    return EthAddr("%012x" % ((dpid*16*16 + dpid) & 0xffffffff))

class L3Switch(EventMixin):
    def __init__(self, fakeways = [], arp_for_unknowns = False, firewall = {}):
        # These are "fake gateways" -- we'll answer ARPs for them with MAC
        # of the switch they're connected to.
        self.fakeways = fakeways

        # If this is true and we see a packet for an unknown
        # host, we'll ARP for it.
        self.arp_for_unknowns = arp_for_unknowns

        # (dpid, IP) -> expire_time
        # We use this to keep from spamming ARPs
        self.outstanding_arps = {}

        # (dpid, IP) -> [(expire_time, buffer_id, in_port), ...]
        # These are buffers we've gotten at this datapath for this IP which
        # we can't deliver because we don't know where they go.
        self.lost_buffers = {}
```

حال مانند دو بخش قبل در دایرکتوری ./pox دستور

```
./pox.py log.level --DEBUG misc.firewall
```

را وارد می کنیم و در ترمینال mininet، دستور

```
$ sudo mn --topo single,3 --mac --controller remote --switch ovsk
```

را وارد می کنیم.

سپس با دستور xterm h1 h2، xterm مربوط به این دو host را بالا می آوریم.

و دستور

```
$ iperf -s
```

را در h2 وارد می کنیم تا به عنوان یک سرور عمل کند و دستور

```
$ iperf -c 10.0.0.2
```

در xterm h3 وارد می کنیم و همانطور که در تصویر زیر مشاهده می کنید، firewall پیام های این host یعنی h3 را فیلتر می کند.

```

Node: h3
root@mininet:~# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
^Croot@mininet:~# iperf -c 10.0.0.2
connect failed: Network is unreachable
root@mininet:~#

Node: h2
root@mininet:~# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)

```

باسپاس از توجه شما