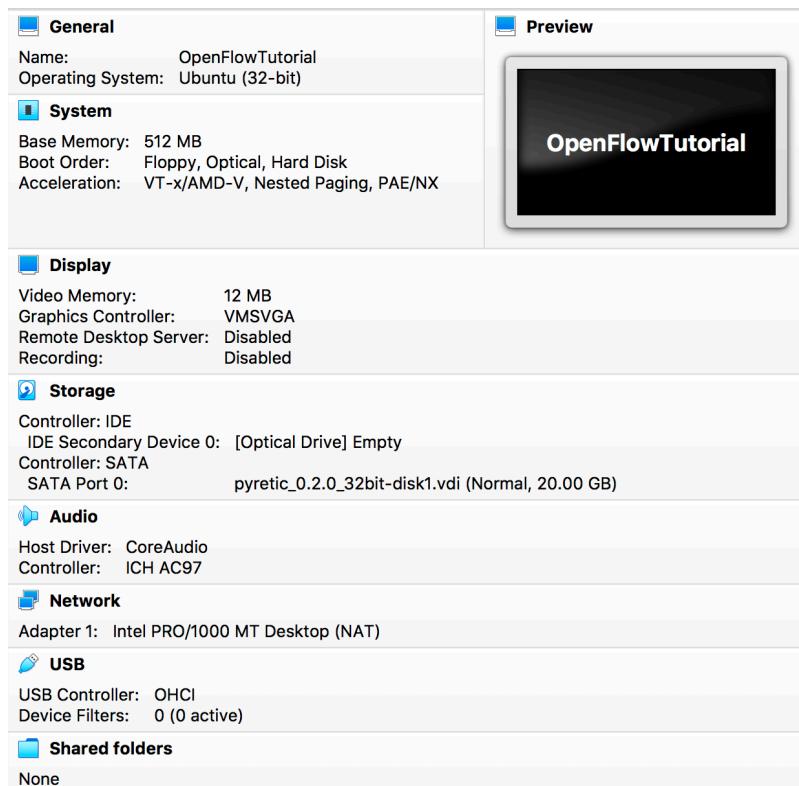


گزارش پروژه پایانی درس شبکه‌های کامپیوتری

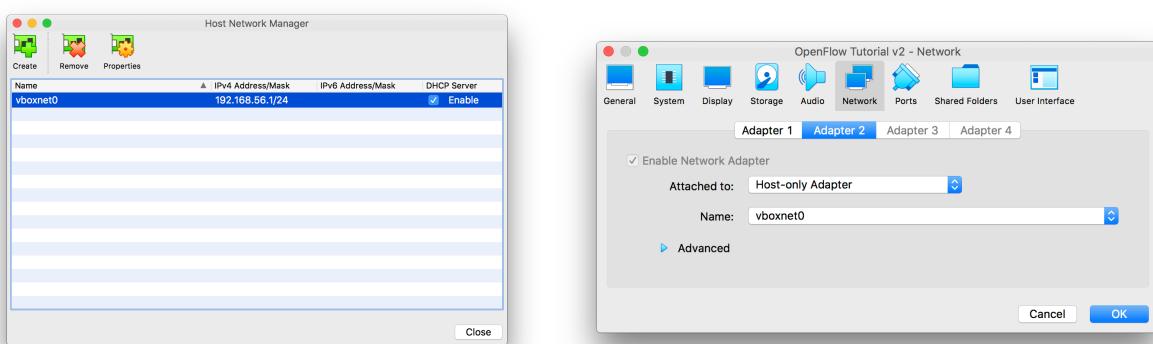
۱. بدون توضیح

۲. پروژه را در سیستم عامل مک اواس اجرا می‌کنیم. بنابراین نیاز به نرم‌افزارهای VirtualBox برای اجرای mininet و XQuartz برای راهاندازی ترمینال‌ها با رابط گرافیکی نیاز داریم. این نرم‌افزارها را دانلود و نصب می‌کنیم.

۳. پس از import کردن و راهاندازی اولیه‌ی mininet در VirtualBox، نوبت به پیکربندی واسطه‌های شبکه می‌رسد.



یک واسط Cable Connected و DHCP Enabled، با تنظیمات Host-only اضافه می‌کنیم.



این واسط برای ارتباط با دستگاه host استفاده خواهد شد. (با استفاده از SSH، قسمت بعدی). پس از این، سیستم عامل را اجرا کرده، با دستور `ifconfig -a`، تنظیمات واسطه‌ها را بررسی کرده و مطمئن می‌شویم که تمام واسطه‌ها آدرس IP دارند. (در صورت نداشتن آدرس IP واسطه ethX، با دستور `sudo dhclient ethX`، به کمک پروتکل DHCP، به آن IP اختصاص می‌دهیم).

پیش از اختصاص آدرس IP:

```
mininet@mininet:~$ ifconfig -a
eth0      Link encap:Ethernet HWaddr 00:00:27:04:78:8b
          inet addr:10.0.2.15 Bcast:10.0.2.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe04:788b/64 Scope:Link
             UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
             RX packets:21 errors:0 dropped:0 overruns:0 frame:0
             TX packets:27 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:2466 (2.4 KB) TX bytes:2534 (2.5 KB)

eth2      Link encap:Ethernet HWaddr 00:00:27:c5:0e:f3
          BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

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:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:0
             RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

mininet@mininet:~$ _
```

نمونه 1 واسط eth2 آدرس IP ندارد.

پس از اختصاص آدرس IP:

```
mininet@mininet:~$ sudo dhclient eth2
mininet@mininet:~$ ifconfig -a
eth0      Link encap:Ethernet HWaddr 00:00:27:04:78:8b
          inet addr:10.0.2.15 Bcast:10.0.2.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe04:788b/64 Scope:Link
             UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
             RX packets:26 errors:0 dropped:0 overruns:0 frame:0
             TX packets:32 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:3006 (3.0 KB) TX bytes:2914 (2.9 KB)

eth2      Link encap:Ethernet HWaddr 00:00:27:c5:0e:f3
          inet addr:192.168.56.3 Bcast:192.168.56.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fec5:ef3/64 Scope:Link
             UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
             RX packets:2 errors:0 dropped:0 overruns:0 frame:0
             TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:1180 (1.1 KB) TX bytes:922 (922.0 B)

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:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:0
             RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

mininet@mininet:~$ _
```

نمونه 2 واسط eth2 آدرس IP برابر 192.168.56.3 دارد.

از طریق پروتکل SSH به دستگاه مجازی خود متصل می‌شویم. برای این کار یوزر مورد نظر، به همراه آدرس IP را به عنوان ورودی به دستور SSH می‌دهیم. همچنین دستور X-را اضافه می‌کنیم، تا با فعال کردن XQuartx، بتوانیم از رابط گرافیکی استفاده کنیم.

```
Danial — ssh -X mininet@192.168.56.3 — Solarized Dark ansi — 80x24
danials-MacBook-Pro-2:~ Danialh$ ssh -X mininet@192.168.56.3
mininet@192.168.56.3'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 Thu Feb  4 01:03:15 PST 2021

 System load:  0.16           Processes:      71
 Usage of /:   9.4% of 18.94GB  Users logged in:    1
 Memory usage: 9%            IP address for eth0: 10.0.2.15
 Swap usage:   0%            IP address for eth2: 192.168.56.3

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

Last login: Thu Feb  4 00:53:00 2021
mininet@mininet:~$ |
```

.۴

** توضیح نرم‌افزارها **

آشنایی مختصر با mininet

برای ایجاد یک شبکه، با ۳ تا host مجازی (که آدرس MAC شان همان آدرس IP اشان باشد)، به همراه یک نرم‌افزار OpenFlow مسیریابی و جلوگیری، از دستور زیر استفاده می‌کنیم.

```
sudo mn --topo single,3 --mac
```

سپس، می‌توانیم با دستور nodes، اجزای شبکه‌ای که ساختیم را مشاهده کنیم.

```
Daniel — ssh-> mininet@192.168.56.3 — Solarized Dark ansi — 80x24
mininet@mininet:~$ sudo mn --topo single,3 --mac
*** 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> nodes
available nodes are:
h2 h3 h1 c0 s1
[mininet> |
```

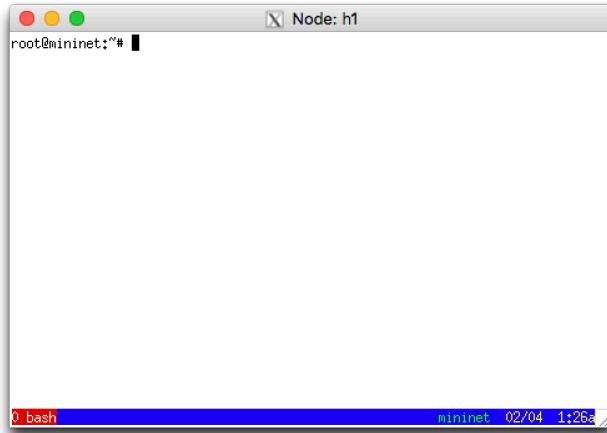
همچنین می‌توانیم با اضافه کردن پیشوند `hi`, هر دستور را برای آن `host` خاص اجرا کنیم، به عنوان مثال `hi ifconfig`، تنها پیکربندی `hi` را نشان خواهد داد.

```
mininet> h1 ifconfig
h1-eth0 Link encap:Ethernet HWaddr 00:00:00:00:00:01
      inet addr:10.0.0.1 Bcast:10.255.255.255 Mask:255.0.0.0
          inet6 addr: fe80::200:ff:fe00:1/64 Scope:Link
             UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
             RX packets:16 errors:0 dropped:0 overruns:0 frame:0
             TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:1224 (1.2 KB) TX bytes:558 (558.0 B)

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:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:0
             RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

[mininet> |
```

با دستور `hi xterm` هم می‌توانیم رابط ترمینال `hi` را در پنجره‌ای جدا باز کنیم.



نمونه استفاده از dpctl برای مشاهده، و خطایابی در Flow Table ها استفاده می شود.

```
Danial — ssh -X mininet@192.168.56.3 — Solarized Dark ansi — 81x24
mininet@mininet:~$ dpctl show tcp:127.0.0.1:6634
features_reply (xid=0x85f175e4): ver:0x1, dpid:1
n_tables:255, n_buffers:256
features: capabilities:0xc7, actions:0xffff
1(s1-eth1): addr:a6:26:1d:5d:7f:b7, config: 0, state:0
    current: 10GB-FD COPPER
2(s1-eth2): addr:5a:3f:90:80:6c:93, config: 0, state:0
    current: 10GB-FD COPPER
3(s1-eth3): addr:7a:57:5a:69:76:34, config: 0, state:0
    current: 10GB-FD COPPER
LOCAL(s1): addr:0:e:a1:d4:28:4f:41, config: 0x1, state:0x1
get_config_reply (xid=0xf82a2d43): miss_send_len=0
mininet@mininet:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x3a5314d0): flags=none type=1(flow)
mininet@mininet:~$ |
```

چون هنوز controller را شروع نکرده‌ایم، Flow Table خالی است.

تست ping

از طریق زیر به جدول Flow Table می‌توانیم Entry اضافه کنیم، تا جلوگاهی از h2 به h1، و برعکس، انجام شود.

```

Danial — ssh -X mininet@192.168.56.3 — Solarized Dark ansi — 124x21
mininet@mininet:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=1,idle_timeout=120,actions=output:2
mininet@mininet:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=2,idle_timeout=120,actions=output:1
mininet@mininet:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x5f6795c8): flags=none type=1(flow)
  cookie=0, duration_sec=7s, duration_nsec=3310000000s, 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=2s, duration_nsec=5950000000s, 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:~$ |

```

همان طور که در Flow Table بالا می‌بینید، دو Entry داریم. یکی برای مسیر h1 به h2، و دیگری بر عکس تا بین ترتیب هر بسته از h1 به h2 برسد، و پاسخش از h2 به h1 بگردد. هر Entry پارامترهای زیر را دارد.

Flow Table: مدت زمان سپری شده از اضافه شدن این Entry به

Entry: الیت این priority –

table_id: شماره‌ی شناسایی این جدول –

n_packets: تعداد بسته‌هایی که از طریق این Entry حرکت کرده‌اند. –

idle_timeout: مدت زمانی که این Entry می‌تواند بی استفاده بماند قبل از این که حذف شود. –

in_port: بسته‌های مطابق با این Entry باید از چه پورتی وارد شوند؟ –

actions: با بسته‌های مطابق این Entry باید چه کار کرد؟ برای مثال output:2 یعنی از پورت ۲ خارج شوند. از

action: دیگر می‌توان به drop کردن، یا buffer کردن بسته اشاره کرد. –

حال با ping کردن h1 از h2 خواهیم داشت:

```

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=0.245 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.028 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.028/0.106/0.245/0.098 ms
mininet> |

```

همان طور که می‌بینید، با ایجاد شدن مسیرها، ping موفقیت‌آمیز خواهد بود.

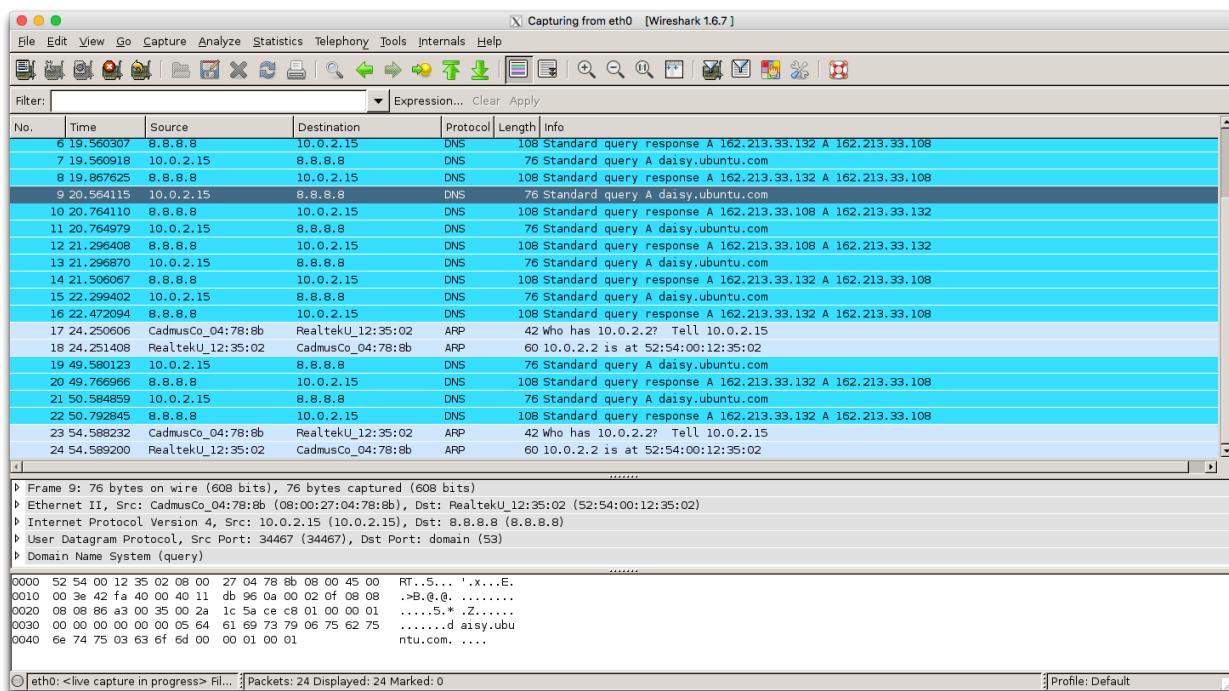
شروع Wireshark

پس از برقراری ارتباط SSH با دستگاه مجازی، با دستور `sudo wireshark &` آن را اجرا می‌کنیم.

پس از باز شدن Wireshark، می‌توانیم شروع به شنود بسته‌ها کنیم. برای این کار از طریق Capture->Interfaces، واسط هدف (مثالاً

۱۰) را انتخاب کرده، و با زدن `start`، شنود شروع می‌کنیم. همچنین می‌توانیم از طریق منوی Filter، بسته‌های مورد شنود را طبق پروتکل‌شان و ... فیلتر کنیم.

در مثال زیر، واسط eth0 (بدون فیلتر) شنود شده است.



شروع Wireshark و مشاهده‌ی Startup Message Controller های

در حین این که Wireshark در حال گوش دادن به واسط ۱۰ (با فیلتر پروتکل‌های OF و چشم‌پوشی از دیگر پروتکل‌ها) است، با دستور

controller ptcp:

یک کنترلر ساده، در نقش learning switch می‌سازیم. منظور از learning switch این است که کنترلر سوییچ با این که در ابتداء جدولش خالی است، و آدرس دستگاه‌های اطرافش را نمی‌داند، می‌تواند پس از گرفتن هر بسته، یافتن هر مسیر از طریق پروتکل ARP، این مسیر را در جدول خود ذخیره می‌کند، تا برای مسیریابی بسته‌های بعدی، به درستی مسیر را انتخاب کند.

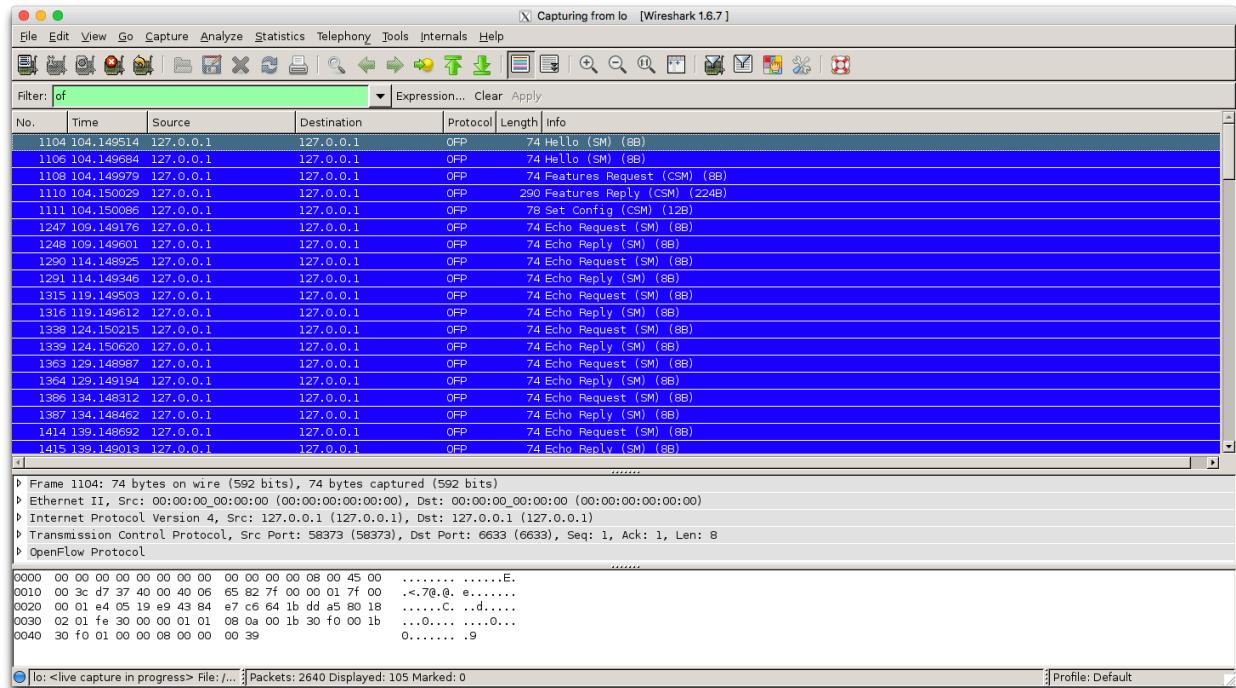
در طی ارتباط بین سوییچ و کنترلر، پیام‌های زیر رد و بدل می‌شوند.

- Hello: کنترلر به سوییچ version number لیش را اعلام می‌کند. سپس سوییچ به کنترلر version number مورد پشتیبانی اش را اطلاع می‌دهد.

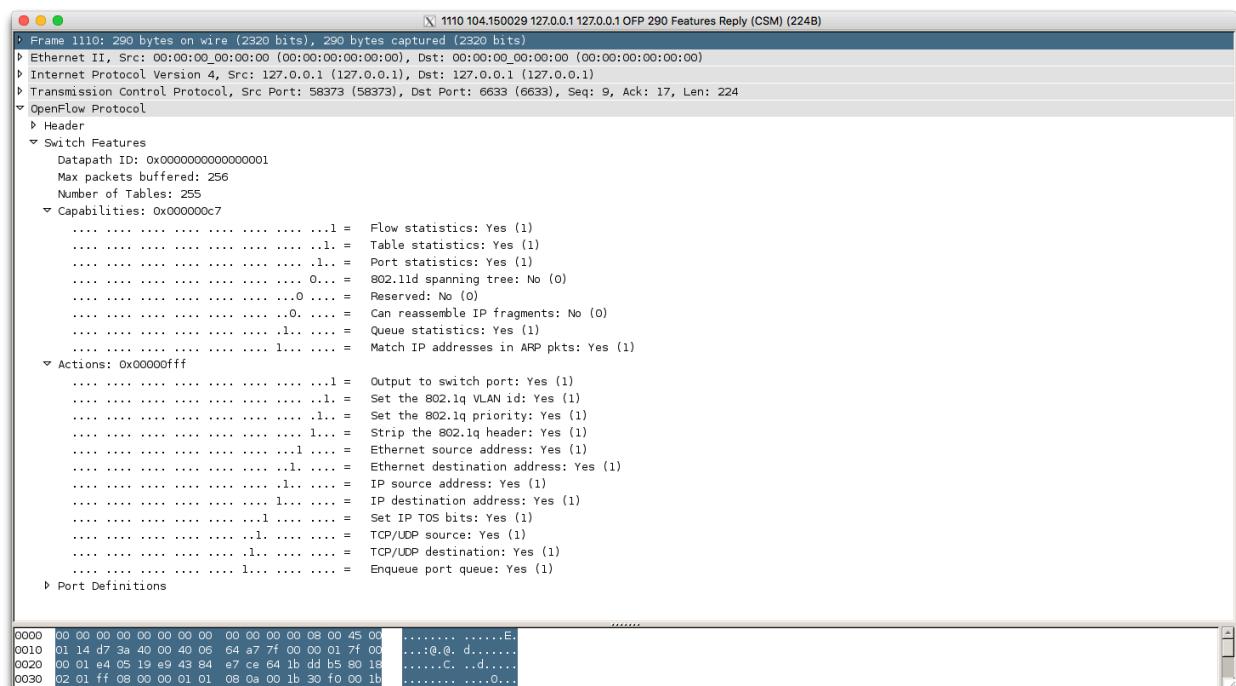
- Features Request: کنترلر از سوییچ شماره‌ی پورت‌های فعال و موارد دیگری را درخواست می‌کند.

- Set Config: کنترلر از سوییچ زمان انقضای هر Flow را درخواست می‌کند.

- Features Reply: سوییچ به کنترلر لیست هر یک از پورت‌ها و سرعتشان، و جدول‌ها و action‌های مورد پشتیبانی اش را اعلام می‌کند.



برای مثال با بررسی بیشتر یک بسته از نوع Features Reply Message خواهیم داشت:



در این تصویر، مشخصات قسمت OpenFlow بسته‌ی شنود شده را می‌توان مشاهده کرد. برای مثال .Capabilities و .Datapath ID

مشاهده‌ی Ping برای OpenFlow Messages

در قسمت قبل پیام‌های ابتدایی ایجاد ارتباط بین کنترلر و سویچ را بررسی کردیم. طی این پیام‌ها، کنترلر و سویچ تنظیمات اولیه‌ای مانند شماره‌ی **version**، زمان انقضای **Flow Entry**، تعداد پورت‌ها و سرعت هر کدام و ... رد و بدل می‌شود. به سراغ روند ایجاد یک **Flow Entry** در جدول می‌رویم. برای این کار، ابتدا با افزودن فیلتر **(of.type != 3) && (of.type != 2)**، بسته‌های **echo-request/reply** را که برای زنده نگه داشتن ارتباط بین سویچ و کنترلر استفاده می‌شوند، نادیده می‌گیریم.

سپس **h2** را با **h1** پینگ می‌کنیم. قاعده‌ای در ابتداء **Flow Table** خالی بوده، و مسیر **h2** به **h1** مشخص نیست. اما طی این ارتباط، سویچ **h1** و **h2** را شناسایی می‌کند، و در ارتباط‌های بعدی می‌توان از **Flow Entry** اضافه شده استفاده کند. طی این روند، این پیام‌ها رد و بدل می‌شوند.

- **Packet In**: سویچ بسته‌ای دریافت کرده، که با هیچ‌کدام از **Entry**‌های جدولش مطابقت ندارد، پس آن را به کنترلر ارسال می‌کند.

- **Packet Out**: کنترلر بسته‌ای را از یک یا چندین پورت سویچ ارسال می‌کند.

- **Flow Mod**: کنترلر پس از این که از طریق **Packet Out** برای بسته‌ی دریافت شده یک سیاست مناسب پیدا کرد، این سیاست را در قالب یک **Flow Entry** به سویچ می‌دهد تا به **Flow Table** اضافه کند.

- **Flow Expired**: سویچ به کنترلر اطلاع می‌دهد که یک **Flow Entry** مدت انقضائی گذشته، و از جدول پاک می‌شود.

نمونه پیام‌های بالا طی دستور :**h1 ping -c1 h2**

16333 1483.473138 00:00:00_00:00:01	Broadcast	OFP+ARP	126 Packet In (AM) (BufID=271) (60B) => who has 10.0.0.2? Tell 10.0.0.1
16334 1483.474124 127.0.0.1	127.0.0.1	OFP	90 Packet out (CSM) (BufID=271) (24B)
16336 1483.474228 00:00:00_00:00:02	00:00:00_00:00:01	OFP+ARP	126 Packet In (AM) (BufID=272) (60B) => 10.0.0.2 is at 00:00:00:00:00:02
16337 1483.474267 127.0.0.1	127.0.0.1	OFP	146 Flow Mod (CSM) (80B)
16338 1483.474379 10.0.0.1	10.0.0.2	OFP+ICM	182 Packet In (AM) (BufID=273) (116B) => Echo (ping) request id=0x08d9, seq=1/256, ttl=64
16339 1483.474436 127.0.0.1	127.0.0.1	OFP	146 Flow Mod (CSM) (80B)
16340 1483.474498 10.0.0.2	10.0.0.1	OFP+ICM	182 Packet In (AM) (BufID=274) (116B) => Echo (ping) reply id=0x08d9, seq=1/256, ttl=64
16341 1483.474552 127.0.0.1	127.0.0.1	OFP	146 Flow Mod (CSM) (80B)
16762 1488.479398 00:00:00_00:00:02	00:00:00_00:00:01	OFP+ARP	126 Packet In (AM) (BufID=275) (60B) => who has 10.0.0.1? Tell 10.0.0.2
16763 1488.479778 127.0.0.1	127.0.0.1	OFP	146 Flow Mod (CSM) (80B)
16765 1488.480173 00:00:00_00:00:01	00:00:00_00:00:02	OFP+ARP	126 Packet In (AM) (BufID=276) (60B) => 10.0.0.1 is at 00:00:00:00:00:01
16766 1488.480348 127.0.0.1	127.0.0.1	OFP	146 Flow Mod (CSM) (80B)

می‌دانیم **h1** آدرس IP 10.0.0.1 و **h2** آدرس 10.0.0.2 را دارد. **h1** قصد ارسال بسته به **h2** را دارد، اما چون **h2** هنوز توسط سویچ شناخته نشده، مکانش مشخص نیست. پس یک پیغام (نوع اول دسته‌بندی بالا، **Packet In**) ارسال می‌شود، تا آدرس فیزیکی دستگاهی که آدرس 10.0.0.2 را دارد پرسیده شود.

طی یک پیغام OFP. (نوع دوم بالا، **Packet Out**) کنترلر از طریق تمام پورت‌های سویچ، این آدرس را جست و جو می‌کند.

دستگاه **h2** که آدرس 10.0.0.2 را دارد، پیغام را پاسخ داده، و آدرس فیزیکی اش را اعلام می‌کند.

طی یک پیغام OFP. (نوع سوم بالا، **Flow Mod**) کنترلر به سویچ دستور اضافه شدن یک **Entry** می‌دهد. این **Entry** شماره‌ی پورت خروجی مختص پکت‌های به مقصد **h2** را مشخص می‌کند.

این مسیر به طور بر عکس نیز طی می‌شود. یعنی این بار، باید یک **Entry** مختص پکت‌های به مقصد **h1** (10.0.0.1) مشخص شود، که طی **Flow Mod** و **Packet Out** انجام می‌شود.

طی چنین روندی، جلوه‌نگاری بسته‌ها انجام خواهد شد، و ping انجام شده هم موفقیت‌امیز خواهد بود.

```

mininet> h1 ping -c1 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=0.938 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.938/0.938/0.938/0.000 ms
mininet>

```

:Benchmark Controller w/iperf

دستور iperf، یک ابزار خط فرمان برای اندازه‌گیری سرعت ارتباط میان دو دستگاه است. نمونه‌ی خروجی این دستور را، برای توضیح شده در تصویر، در ادامه آمده است.

```

Danial — ssh -X mininet@192.168.56.3 — Solarized Dark ansi — 91x5
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
waiting for iperf to start up...*** Results: ['5.98 Gbits/sec', '5.98 Gbits/sec']
mininet>
mininet>

```

طبق دستور بالا، سرعت ارتباط TCP بین hostهای h1 و h3، برابر 5.98 Gbits/sec می‌باشد.
رونده کار این دستور به این صورت است که یک iperf server در دستگاه اول، و یک iperf client در دستگاه دوم ایجاد کرده، و پس از برقراری ارتباط این دو، بسته‌هایی بینشان رد و بدل می‌کند، و سرعت را اندازه می‌گیرد.

سرعت را برای یک تپولوژی شبکه‌ی دیگر، اندازه می‌گیریم.
در این تپولوژی به علت این که بسته‌ها نیازی به انتقال از User Space در گره‌ها ندارند، سرعت ارتباط بالاتر می‌رود.

```

Danial — ssh -X mininet@192.168.56.3 — Solarized Dark ansi — 91x24
mininet@mininet:~$ sudo mn --topo single,3 --mac
*** 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
*** Results: ['6.07 Gbits/sec', '6.08 Gbits/sec']
mininet>

```

۵. ایجاد یک learning switch

در این قسمت می‌خواهیم یک Learning switch را برنامه‌نویسی کنیم. سویچ این گونه کار خواهد کرد که برای هر بسته، آدرس فیزیکی آن را به پورت ورودی اش نسبت می‌دهد. پس از آن برای جلوانی هر بسته‌ی جدید، به آدرس فیزیکی مقصد نگاه کرده، و طبق آن پورت خروجی را مشخص می‌کند.

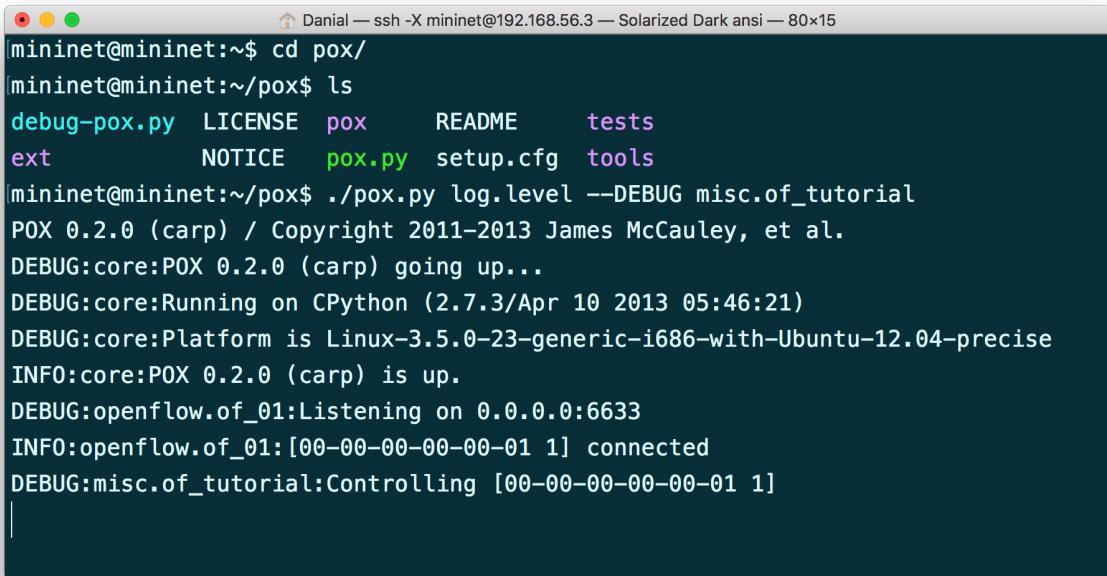
برای این کار برنامه‌ی یک Hub را گرفته، و به آن قابلیت‌های یک switch را اضافه می‌کنیم. برای نوشتن برنامه گزینه‌های زبان برنامه‌نویسی متفاوتی وجود دارد.

در اینجا ما از زبان Python و کتابخانه POX استفاده می‌کنیم.

- ابتدا source code این کتابخانه را روی سیستم عامل mininet clone می‌کنیم.
- به دایرکتوری این کتابخانه رفته، و دستور زیر را اجرا می‌کنیم.

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

- با این دستور کامپونت hub ساده است، شروع به کار کرده، و سویچ به آن متصل می‌شود.



```
Danial — ssh -X mininet@192.168.56.3 — Solarized Dark ansi — 80x15
[mininet@mininet:~$ cd pox/
[mininet@mininet:~/pox$ ls
debug-pox.py LICENSE pox README tests
ext NOTICE pox.py setup.cfg tools
[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-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]
```

اطمینان از کار کرد درست hub

می‌دانیم یک hub با انتقال هر بسته‌ی ورودی، به تمام خروجی‌ها، یک توبولوزی Bus می‌سازد. بنابراین در صورت کار کرد درست hub، تمام بسته‌ها باید برای تمام host‌های شبکه قابل دسترسی باشند. برای چک کردن این موضوع، مراحل زیر را طی می‌کنیم.

- با دستور xterm h1 h2 h3، ترمینال هر یک از host‌ها را در یک پنجره‌ی جدا باز می‌کنیم.
- از h1 ping را h2 می‌کنیم.
- بسته‌های دریافتی و ارسالی هر host را طی فرآیند ping، به کمک دستور tcpdump چاپ می‌کنیم. این دستور در هر host بسته‌های دریافت شده‌اش را به طور پیوسته چاپ می‌کند.

:h1 پینگ کردن h2 از

```

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=17.3 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 17.375/17.375/17.375/0.000 ms
root@mininet:~#

```

0 bash mininet 02/05 12:55a

بسته‌های دریافتی h2 طی ping بالا به صورت زیر است:

```

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
00:49:05.411353 ARP, Request who-has 10.0.0.2 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 0002  .....
00:49:05.411381 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
    0x0000: 0000 0000 0001 0000 0000 0002 0806 0001  .....
    0x0010: 0800 0604 0002 0000 0000 0002 0a00 0002  .....
    0x0020: 0000 0000 0001 0a00 0001  .....
00:49:05.417041 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 5614, seq 1, length 64
    0x0000: 0000 0000 0002 0000 0000 0001 0800 4500  ....E.
    0x0010: 0054 0000 4000 4001 26a7 0a00 0001 0a00 .T..@.R.&.....
    0x0020: 0002 0800 0e84 15ee 0001 8106 1d60 4623  .....
    0x0030: 0600 0809 0a0b 0c0d 0e0f 1011 1213 1415  .....
    0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ....!#$%.
    0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*,-,./012345
    0x0060: 3637 67
00:49:05.417115 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 5614, seq 1, length 64
    0x0000: 0000 0000 0001 0000 0000 0002 0800 4500  ....E.
    0x0010: 0054 fdd0 0000 4001 68d6 0a00 0002 0a00 .T....@.h....F#
    0x0020: 0001 0000 1484 15ee 0001 8106 1d60 4623  .....
    0x0030: 0600 0809 0a0b 0c0d 0e0f 1011 1213 1415  .....
    0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ....!#$%.
    0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*,-,./012345
    0x0060: 3637 67
00:49:10.417947 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  .....
00:49:10.429915 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  .....

```

همان‌طور که مشاهده می‌شود، به ترتیب پیغام‌های (h1) ARP Request – به دنبال آدرس فیزیکی h2 است. (h2) ARP Reply – آدرس فیزیکی اش را به h1 اعلام می‌کند.

(h2 ping به h1) ICMP echo request	-
میخواهد به h1 ping را پاسخ دهد.	-
(h2 ARP request (درخواست آدرس فیزیکی h1 توسط h2	-
(معرفی آدرس فیزیکی h1 توسط h1 ARP reply	-

اما چون شبکه از hub استفاده می‌کند، و توبولوژی آن bus است، تمام پیغام‌ها برای تمام host‌ها قابل مشاهده است، در نتیجه پیغام‌های دریافتی h3 هم به صورت زیر خواهد بود.

```
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
00:49:05.411352 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
    0x0000: ffff ffff ffff 0000 0001 0806 0001 .....
    0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
    0x0020: 0000 0000 0000 0a00 0002 .....
00:49:05.413058 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
    0x0000: 0000 0000 0001 0000 0000 0002 0806 0001 .....
    0x0010: 0800 0604 0002 0000 0000 0002 0a00 0002 .....
    0x0020: 0000 0000 0001 0a00 0001 .....
00:49:05.417024 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 5614, seq 1, length 64
    0x0000: 0000 0000 0002 0000 0000 0001 0800 4500 ....E.
    0x0010: 0054 0000 4000 4001 26a7 0a00 0001 0a00 .T..@.0.&.....
    0x0020: 0002 0800 0e84 15ee 0001 8106 1d60 4623 ....F#
    0x0030: 0600 0809 0a0b 0c0d 0e0f 1011 1213 1415 .....
    0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 ....!#$%
    0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &(')*+,-./012345
    0x0060: 3637 67
00:49:05.419620 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 5614, seq 1, length 64
    0x0000: 0000 0000 0001 0000 0000 0002 0800 4500 ....E.
    0x0010: 0054 fdd0 0000 4001 68d6 0a00 0002 0a00 .T....@.h.....
    0x0020: 0001 0000 1484 15ee 0001 8106 1d60 4623 ....F#
    0x0030: 0600 0809 0a0b 0c0d 0e0f 1011 1213 1415 .....
    0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 ....!#$%
    0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &(')*+,-./012345
    0x0060: 3637 67
00:49:10.428729 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 .....
00:49:10.429912 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 .....
```

و در صورتی که آدرس IP ناموجودی را ping کنیم، قاعداً پیغام ARP Request بی‌پاسخ خواهد ماند. نمونه‌ی این سناریو را در ادامه با ping کردن 10.0.0.5 می‌بینیم.

```

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:~#

```

```

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
01:09:24.328234 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
    0x0000: ffff ffff 0000 0000 0001 0806 0001 .....
    0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
    0x0020: 0000 0000 0000 0a00 0005 .....
01:09:25.317559 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
    0x0000: ffff ffff 0000 0000 0001 0806 0001 .....
    0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
    0x0020: 0000 0000 0000 0a00 0005 .....
01:09:26.312254 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
    0x0000: ffff ffff 0000 0000 0001 0806 0001 .....
    0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
    0x0020: 0000 0000 0000 0a00 0005 .....

```

بنج مارک گرفتن از Hub

ابتدا از اتصال تمام host‌ها به هم مطمئن می‌شویم. همان‌طور که مشاهده می‌شود هیچ بسته‌ای drop نشده، و ارتباطات برقرار است.

```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (0/6 lost)
mininet>

```

سپس با دستور iperf سرعت ارتباط را اندازه می‌گیریم.

```

*** Starting CLI:
[mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['9.17 Mbits/sec', '9.44 Mbits/sec']
mininet> |

```

در اینجا چون هر بسته نیاز است به کنترلر فرستاده شود، سرعت ارتباط به طرز چشم‌گیری کاهش پیدا کرده است.

شروع به تغییر کد hub برای ایجاد switch

فایل `vim /pox/pox/misc/ofTutorial` را باز کرده و شروع به ویرایش آن می‌کنیم. برای این کار از ادیتور `vim` استفاده می‌کنیم.

```
vim /pox/pox/misc/ofTutorial
```

در ابتدا تابع `act_like_hub()` صدای شده شده، که نقش یک `hub` را شبیه‌سازی می‌کند. یعنی هر پکت ورودی را به تمام پورت‌های خروجی ارسال می‌کند.

```

# def _handle_PacketIn(self, event):
    """
    Handles packet in messages from the switch.

    packet = event.parsed # This is the parsed packet data.
    if not packet.parsed:
        log.warning("Ignoring incomplete packet")
        return

    packet_in = event.ofp # The actual ofp_packet_in message.

    # Comment out the following line and uncomment the one after
    # when starting the exercise.
    self.act_like_hub(packet, packet_in)
    #self.act_like_switch(packet, packet_in)

def act_like_hub(self, packet, packet_in):
    """
    Implement hub-like behavior -- send all packets to all ports besides
    the input port.

    We want to output to all ports -- we do that using the special
    # OFPP_ALL port as the output port. (We could have also used
    # OFPP_FLOOD.)
    self.resend_packet(packet_in, of.OFPP_ALL)

    # 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).

```

به جای `act_like_hub`، تابع `act_like_switch` را صدای زنیم، و شروع به پیاده‌سازی بدنی آن می‌کنیم. به طور کلی و همان‌طور که در کامنت‌های این تابع آمده است، باید انجام سه کار را در این تابع پیاده‌سازی کنیم.
۱. آدرس فیزیکی مبدأ هر بسته دریافتی را به همراه پورت ورودی اش در `dictionary` تعریف شده `self.mac_to_port` در صورت نبود، اضافه کنیم.

```
# Learn the port for the source MAC
if packet.src not in self.mac_to_port:
    log.debug("New MAC Address {} found at port {}".format(packet.src, packet_in.in_port))
    self.mac_to_port[packet.src] = packet_in.in_port
```

۲. برای جلوگیری از پورت متناظر آدرس مقصد بسته را در `self.mac_to_port` داشتیم، بسته را از آن پورت خارج می‌کنیم؛ در غیر این صورت، آن را روی تمام پورت‌ها، به جز پورت مبدأ، ارسال می‌کنیم. (`Flood`)

```

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

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

```

3. اضافه کردن یک Flow برای بسته جدید

```

# 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...")
# Maybe the log statement should have source/destination/port?
log.debug("Source MAC: {}", source port: {}".format(packet.src, packet_in.in_port))
log.debug("Destination MAC: {}, destination port: {}".format(packet.dst, self.mac_to_port[packet.dst]))

msg = of.ofp_flow_mod()
# Set fields to match received packet
msg.match = of.ofp_match.from_packet(packet)
# < Set other fields of flow_mod (timeouts? buffer_id?) >
msg.idle_timeout = 60
msg.hard_timeout = 120
msg.buffer_id = packet_in.buffer_id
# Add an output action, and send -- similar to resend_packet() >
msg.actions.append(of.ofp_action_output(port=self.mac_to_port[packet.dst]))
self.connection.send(msg)

```

تست کنترلر

طمئن می شویم تنها بسته های Broadcast (مثل ARP) و بسته های با مقصد ناشناخته (که Flood می کنیم) از تمام پورت های غیر ورودی خارج می شوند. برای این کار روی h2 و h3 با فرمان tcpdump شروع به شنود کرده، و سپس از h1 دستگاه ping را می کنیم. در صورت کار کرد درست، h3 جز پیام همه پخشی ARP Request مشاهده است.

```

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=51.8 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 51.800/51.800/51.800/0.000 ms
root@mininet:~#

```

```

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
01:31:01.384658 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
    0x0000: ffff ffff 0000 0000 0001 0806 0001 .....
    0x0010: 0800 0604 0001 0000 0000 0a00 0001 .....
    0x0020: 0000 0000 0000 0a00 0002 .....
01:31:01.384691 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
    0x0000: 0000 0000 0001 0000 0000 0002 0806 0001 .....
    0x0010: 0800 0604 0002 0000 0000 0a00 0002 .....
    0x0020: 0000 0000 0001 0a00 0001 .....
01:31:01.389025 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 3377, seq 1, length 64
    0x0000: 0000 0000 0002 0000 0000 0001 0800 4500 .....E.
    0x0010: 0054 0000 4000 4001 26a7 0a00 0001 0a00 .T..@.8. ....;.
    0x0020: 0002 0800 f4dc 0d31 0001 d561 1e60 122c .....1...a. ..
    0x0030: 0500 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
01:31:01.389049 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 3377, seq 1, length 64
    0x0000: 0000 0000 0001 0000 0000 0002 0800 4500 .....E.
    0x0010: 0054 152a 0000 4001 517d 0a00 0002 0a00 .T.*..@.Q}....;.
    0x0020: 0001 0000 fc0c 0d31 0001 d561 1e60 122c .....1...a. ..
    0x0030: 0500 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
01:31:06.394099 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 .....
01:31:06.403109 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 .....

```

```

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
01:31:01.384653 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
    0x0000: ffff ffff 0000 0000 0001 0806 0001 .....+.
    0x0010: 0800 0604 0001 0000 0000 0a00 0001 .....+.
    0x0020: 0000 0000 0000 0a00 0002 .....+.

```

همچنین با فرمان `dpctl dump-flows` از درستی Flow ها مطمئن می شویم.

```

mininet@mininet:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x2606241e): flags=none type=1(flow)
  cookie=0, duration_sec=22s, duration_nsec=636000000s, table_id=0, priority=32768, n_packets=1, n_bytes=42, idle_timeout=60, hard_timeout=120, arp, dl_vlan=0xffff, dl_vlan_pcp=0x00, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:00:02, nw_src=10.0.0.1, nw_dst=10.0.0.2, nw_proto=2, actions=output:2
  cookie=0, duration_sec=22s, duration_nsec=640000000s, table_id=0, priority=32768, n_packets=1, n_bytes=42, idle_timeout=60, hard_timeout=120, arp, dl_vlan=0xffff, dl_vlan_pcp=0x00, dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:00:01, nw_src=10.0.0.2, nw_dst=10.0.0.1, nw_proto=1, actions=output:1
  cookie=0, duration_sec=27s, duration_nsec=664000000s, table_id=0, priority=32768, n_packets=1, n_bytes=42, idle_timeout=60, hard_timeout=120, arp, dl_vlan=0xffff, dl_vlan_pcp=0x00, dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:00:01, nw_src=10.0.0.2, nw_dst=10.0.0.1, nw_proto=2, actions=output:1
  cookie=0, duration_sec=27s, duration_nsec=662000000s, table_id=0, priority=32768, n_packets=1, n_bytes=98, idle_timeout=60, hard_timeout=120, icmp, dl_vlan=0xffff, dl_vlan_pcp=0x00, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:00:02, nw_src=10.0.0.1, nw_dst=10.0.0.2, nw_tos=0x00, icmp_type=8, icmp_code=0, actions=output:2
  cookie=0, duration_sec=27s, duration_nsec=660000000s, table_id=0, priority=32768, n_packets=1, n_bytes=98, idle_timeout=60, hard_timeout=120, icmp, dl_vlan=0xffff, dl_vlan_pcp=0x00, dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:00:01, nw_src=10.0.0.2, nw_dst=10.0.0.1, nw_tos=0x00, icmp_type=0, icmp_code=0, actions=output:1

```

یک نشانه‌ی دیگر از درستی کارکرد برنامه، کاهش چشم‌گیر pingها پس از پیام اول می‌باشد. علت این موضوع این است که پس از پیام اول، Flow در جدول ایجاد شده، و پیام‌ها دیگر نیازی به فرستاده شدن به کنترلر ندارند.

```

root@mininet:~# ping -c10 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=15.5 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.162 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.108 ms
64 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=0.120 ms
64 bytes from 10.0.0.2: icmp_req=5 ttl=64 time=0.195 ms
64 bytes from 10.0.0.2: icmp_req=6 ttl=64 time=0.169 ms
64 bytes from 10.0.0.2: icmp_req=7 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_req=8 ttl=64 time=0.217 ms
64 bytes from 10.0.0.2: icmp_req=9 ttl=64 time=0.121 ms
64 bytes from 10.0.0.2: icmp_req=10 ttl=64 time=0.040 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9000ms
rtt min/avg/max/mdev = 0.040/1.677/15.576/4.633 ms
root@mininet:~# 

```

همان طور که مشاهده می شود، مدت زمان ping از 15 ms به $\sim 0.1\text{ ms}$ می رسد.

۷. در این قسمت یک روتر ناقص می سازیم. روتر ما فیلدهای IP TTL (یا همان Checksum) و یا Time to Live را تغییر نمی دهد، اما طبق سیاست Longest prefix matching کار جلوانی و مسیریابی را طبق آدرس IP مقصد انجام خواهد داد.

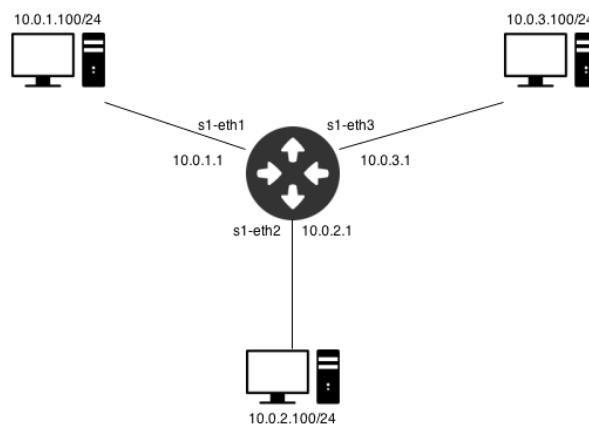
ساخت توپولوژی و پیکربندی hostها

در این قسمت از توپولوژی موجود در مسیر

`/~/mininet/custom/topo-2sw-2host.py`

استفاده می کنیم و آن را تغییر می دهیم تا توپولوژی زیر را بسازیم.

این توپولوژی از سه host و دو سوییچ تشکیل شده است، که به صورت زیر به هم متصلند.



هنگام نوشتن کد توپولوژی، به هر یک از hostها آدرس IP و Default Gateway تخصیص می دهیم.

```

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost = self.addHost('h1', ip='10.0.1.100/24', defaultRoute='10.0.1.1')
        rightHost = self.addHost('h2', ip='10.0.3.100/24', defaultRoute='10.0.3.1')
        bottomHost = self.addHost('h3', ip='10.0.2.100/24', defaultRoute='10.0.2.1')

        switch = self.addSwitch('s1', ip='', defaultRoute='')

        # Add links
        self.addLink(leftHost, switch)
        self.addLink(rightHost, switch)
        self.addLink(bottomHost, switch)

```

پیکربندی host‌ها را می‌توانستیم در رابط خط فرمان سیستم عامل هم انجام دهیم.

این تopo لوژی را اجرا، و با دستور pingall ارتباطات را بررسی می‌کنیم. چون هنوز کنترلری با روتر در ارتباط نیست، ارتباطات قطع هستند.

```

Danial — ssh -Y mininet@192.168.56.3 — Solarized Dark ansi — 80x24
mininet@mininet:~$ sudo mn --custom mytopo.py --topo mytopo --mac
*** 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> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (6/6 lost)
mininet>
mininet>
mininet>

```

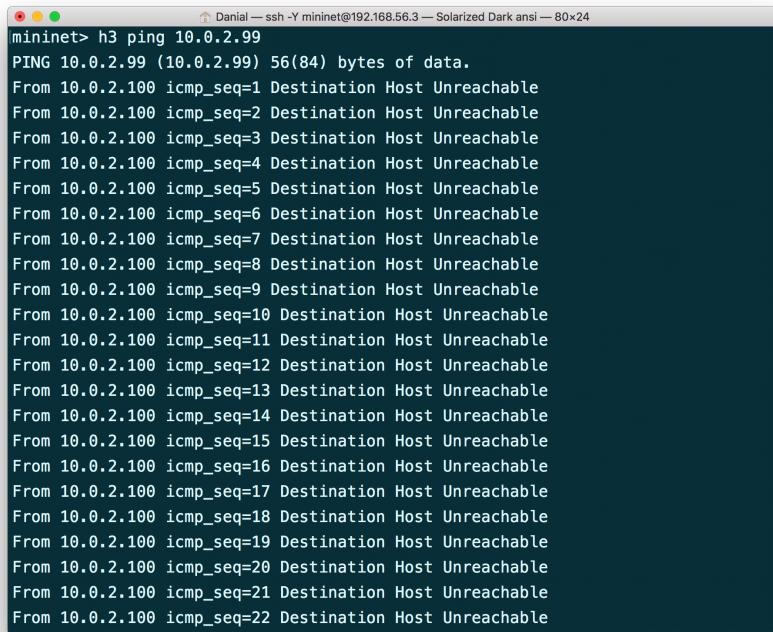
اضافه کردن پشتیبانی ARP، مسیریابی ایستا و ICMP

در این قسمت منطق اصلی روتر را پیاده‌سازی می‌کنیم. به طور کلی بسته‌های ورودی را طبق نوع و پروتکل شان فیلتر کرده، و برای هر یک تصمیم مناسب می‌گیریم.

تست مسیریاب

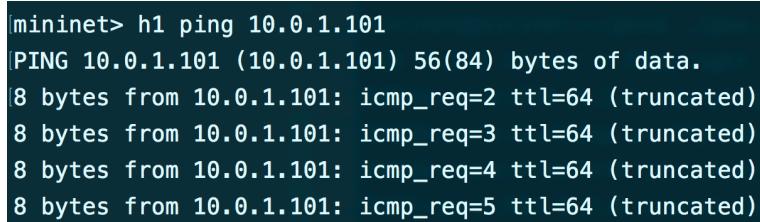
در صورت کارکرد درست، تلاش برای ارسال ICMP Request از شبکه‌ی 10.0.2.100 به 10.99.0.1 با پیام unreachable مواجه می‌شود. (آدرس‌های IP مطابق شکل داده شده در لینک زیر تغییر داده شده‌اند.)

<https://github.com/mininet/openflow-tutorial/wiki/Router-Exercise>



```
Danial — ssh -Y mininet@192.168.56.3 — Solarized Dark ansi — 80x24
mininet> h3 ping 10.0.2.99
PING 10.0.2.99 (10.0.2.99) 56(84) bytes of data.
From 10.0.2.100 icmp_seq=1 Destination Host Unreachable
From 10.0.2.100 icmp_seq=2 Destination Host Unreachable
From 10.0.2.100 icmp_seq=3 Destination Host Unreachable
From 10.0.2.100 icmp_seq=4 Destination Host Unreachable
From 10.0.2.100 icmp_seq=5 Destination Host Unreachable
From 10.0.2.100 icmp_seq=6 Destination Host Unreachable
From 10.0.2.100 icmp_seq=7 Destination Host Unreachable
From 10.0.2.100 icmp_seq=8 Destination Host Unreachable
From 10.0.2.100 icmp_seq=9 Destination Host Unreachable
From 10.0.2.100 icmp_seq=10 Destination Host Unreachable
From 10.0.2.100 icmp_seq=11 Destination Host Unreachable
From 10.0.2.100 icmp_seq=12 Destination Host Unreachable
From 10.0.2.100 icmp_seq=13 Destination Host Unreachable
From 10.0.2.100 icmp_seq=14 Destination Host Unreachable
From 10.0.2.100 icmp_seq=15 Destination Host Unreachable
From 10.0.2.100 icmp_seq=16 Destination Host Unreachable
From 10.0.2.100 icmp_seq=17 Destination Host Unreachable
From 10.0.2.100 icmp_seq=18 Destination Host Unreachable
From 10.0.2.100 icmp_seq=19 Destination Host Unreachable
From 10.0.2.100 icmp_seq=20 Destination Host Unreachable
From 10.0.2.100 icmp_seq=21 Destination Host Unreachable
From 10.0.2.100 icmp_seq=22 Destination Host Unreachable
```

بسته‌هایی که از یک host به host ناموجود فرضی در زیرشبکه‌ی یکسان ارسال می‌شوند، باید دریافت شوند.



```
mininet> h1 ping 10.0.1.101
PING 10.0.1.101 (10.0.1.101) 56(84) bytes of data.
8 bytes from 10.0.1.101: icmp_req=2 ttl=64 (truncated)
8 bytes from 10.0.1.101: icmp_req=3 ttl=64 (truncated)
8 bytes from 10.0.1.101: icmp_req=4 ttl=64 (truncated)
8 bytes from 10.0.1.101: icmp_req=5 ttl=64 (truncated)
```

بسته‌هایی که به یک شبکه‌ی شناخته شده ارسال می‌شوند، باید فیلد آدرس فیزیکی مقصدشان (MAC Destination)، به آدرس فیزیکی مسیریاب بعدی مسیرشان (Next-hop router) تغییر کند. در کد:

```

if destination_ip in self.arp_table:
    etherFrame.src = EthAddr(
        self.arp_table[self.routing_table[destination_network]['RouterInterface']])
    etherFrame.dst = EthAddr(
        self.arp_table[destination_ip])
    self.resend_packet(etherFrame, output_port)
    log.debug("Destination MAC address changed to {}".format(etherFrame.dst))

```

تمام ارتباطات باید برقرار باشند، با دستور pingall این مورد را چک می کنیم.

```

mininet@mininet:~$ sudo mn --custom mytopo.py --topo mytopo --mac
*** 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> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (0/6 lost)
[mininet>

```