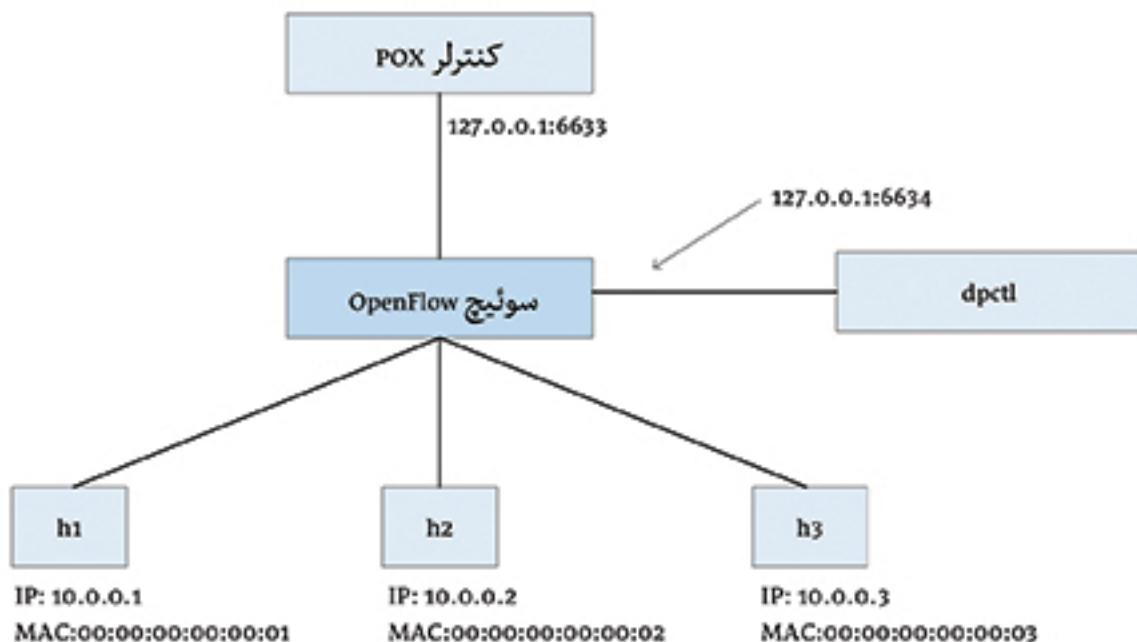


توسعه “Net App”

تا حال، جزئیاتی مربوط به عملکردهای پروتکل OpenFlow، نقش سوئیچ OpenFlow و کنترلر آنرا در اکوسیستم SDN بررسی نمودیم. در فصل ۴ توانستیم محیط توسعه خود را راهاندازی کنیم. در این فصل با توسط کنترلر POX و همچنین کنترلر OpenDayLight (که در فصل پیش آنرا معرفی و راهاندازی کردیم)، به سراغ برخی اپلیکیشن‌های شبکه (Net App) می‌رویم. بخاطر داشته باشید قابلیتها و تواناییهای کنترلرهای Net App، بیشتر از Net App‌هایی است که در این فصل معرفی خواهیم کرد. با این حال هدف ما در این درس، شروعی مقدماتی و رو به جلو به سمت استقرار و توسعه Net App‌هایی است که از چهارچوب OpenFlow استفاده کرده و سرویس‌های خود را ارائه می‌دهند. در اولین بخش از این فصل با آزمایشگاه OpenFlow شروع خواهیم کرد (با استفاده از Mininet) و به سراغ ایجاد سناریوهایی خواهیم رفت که منجر به ایجاد یک هاب - اترنت، سوئیچ یادگیرنده اترنت و یک فایروال ساده شود. سپس جزئیات یک سوئیچ یادگیرنده در کنترلر OpenDaylight را مورد بررسی قرار خواهیم داد.

شماره ۱- سوئیچ یادگیرنده اترنت Net App

ما با استفاده از آزمایشگاه OpenFlow مبتنی بر Mininet، یک شبکه ساده متشکل از یک سوئیچ OpenFlow، سه میزبان و یک کنترلر POX را تشکیل خواهیم داد. توپولوژی شبکه در شکل زیر نشان داده شده است:



ما علاوه بر کنترلر POX، از برنامه dpctl برای آزمایش جدول جریان سوئیچ خود، نیز بهره خواهیم برد. همانطور که پیش از این ذکر شد، سوئیچهای OpenFlow معمولاً روی پورت ۶۶۴۴ گوش (Listen) می‌دهند. dpctl از این کانال برای ارتباط با سوئیچ استفاده می‌کند. حتی بدون یک کنترلر OpenFlow، می‌توانیم از برنامه dpctl برای ارتباط با سوئیچ OpenFlow، در آزمایشگاه Mininet متعلق به OpenFlow خود استفاده کنیم و به ورودی‌های جدول جریان رسیدگی کنیم یا جریانها را اصلاح کنیم. به منظور راهاندازی توبولوژی شبکه (که در شکل پیشین تحت عنوان آزمایشگاه Mininet معرفی شد)، راهاندازی توبولوژی شبکه با پارامترهای خط فرمان زیر (یعنی یک شبکه به همراه ۳ میزبان مجازی)، راه اندازی می‌کنیم:

```
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovsk
--controller remote
```

توجه داشته باشید که Mininet پیغامی نمایش می‌دهد، که قادر نیست با کنترلر راه دور از طریق آدرس 127.0.0.1:6633 ارتباط برقرار کند.

```
*** Adding controller
Unable to connect the remote controller at 127.0.0.1:6633
```

در حقیقت، بدلیل آنکه ما هیچ کنترلر POX را تا کنون راهاندازی نکرده‌ایم، سوئیچ OpenFlow قادر نیست با کنترلر راه دور (که توسط --controller remote در پارامترهای خط فرمان مربوط به

`--controller remote` بصورت `mn1` مشخص می‌شود)، ارتباط برقرار کند. توجه داشته باشید پیش فرض به یک کنترلر OpenFlow با آدرس Localhost (که با آدرس 127.0.0.1 مشخص می‌شود)، اشاره می‌کند. حال به سراغ میزبانهای درون Mininet می‌رویم. شما می‌توانید آدرس IP یا آدرس MAC مربوط به میزبان h1 (و دیگر میزبانها) را با فرمان زیر بررسی کنید:

```
mininet> h1 ifconfig
```

حال می‌توانید وضعیت ارتباط بین میزبانهای h1، h2 و h3 را با استفاده از فرمان `pingall` در Mininet بررسی کنید:

```
mininet> pingall
```

پیغامهای زیر بعنوان خروجی، نمایش داده خواهد شد:

```
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (6/6 lost)
```

این نتایج نشان می‌دهد که میزبانها (علیرغم ارتباط فیزیکی با یکدیگر) در تopoلوجی کنونی به دلیل نبود هیچ ورودی جریانی در جدول جریان سوئیچ، نمی‌توانند به یکدیگر متصل شوند. به عبارت دیگر هیچ یک از میزبانهای h1، h2 و h3 در دسترس نیستند. با استفاده از فرمان زیر می‌توانید یک نمونه برداری از محتویات جدول جریان سوئیچ OpenFlow، داشته باشید. (شما باید یک ارتباط ترمینال SSH با ماشین مجازی Mininet داشته تا بتوانید فرمان زیر را صادر کنید):

```
mininet@mininet-vm:~$ dptcl dump-flows tcp:127.0.0.1:6634
```

نتیجه چنین خواهد بود:

```
status reply (xid=0xf36abb08): flags=none type=l (flow)
```

قبل از اتصال کنترل POX به تopoلوجی شبکه خود (که در آن کنترلر نقش یک هاب - اترنت را دارد)، نگاهی اجمالی و سریع به عملکرد هاب - اترنت خواهیم داشت. یک هاب - اترنت وسیله‌ای برای اتصال چندین دستگاه اترنت به یکدیگر بوده تا همچون یک بخش واحد از شبکه عمل کنند. این نوع تجهیزات چندین پورت ورودی / خروجی داشته که در آن ترافیک داده‌ای که از یک پورت وارد هاب می‌شود را در خروجی همه پورتها، به جز پورت دریافت کننده اصلی نشان می‌دهد. هیچ اطلاعاتی برای ارسال کردن

بسته، در سوئیچ ذخیره نمی‌شود. حال به آزمایشگاه خود باز می‌گردیم. برای آنکه بتوانیم یک هاب را در راه اندازی کنیم، می‌توانیم به سراغ کد `hub.py` از نسخه منتشر شده POX (توسعه یافته Mininet) توسط جیمز مک‌کلی (James McCauley) رفته و از آن استفاده کنیم. این برنامه (همراه با کد مربوط به عملکرد سوئیچ یادگیرنده لایه ۲) در پوشه `~/pox/pox/forwarding` قرار داده شده است. با نگاهی به `hub.py` می‌توانید روش آغاز به کار را بیابید. در کد فوق برای سوئیچهای OpenFlow، شنوندهای (Listener) اضافه می‌شود تا از طریق آن، اتصال صورت گیرد.

```
def launch():
    core.openflow.addlisternetByName("connectionUp",
    _handle_ConnectionUp)
    log.info("Hub running.")
```

متدهای `_handle_ConnectionUp` که متدهای در کد `hub.py` است، در واقع یک پیغام OpenFlow برای سوئیچ OpenFlow ایجاد می‌کند. Action که به پیغام، پیوست شده است در واقع بسته را در تمام پورتهای سوئیچ OpenFlow پخش می‌کند (به جز پورت دریافتی). سپس پیغام ایجاد شده به سوئیچ OpenFlow (در توپولوژی شبکه آزمایشی خود) ارسال می‌شود:

```
def _handle_ConnectionUp (event):
    msg= of.ofp_flow_mod()
    msg.actions.append(of.ofp_action_output(port= of.OFPP_FLOOD))
    event.connection.send(msg)
    log.info("Hubifying %s", dpidToStr(event.dpid))
```

به این ترتیب بخش رسیدگی کننده رویدادها که در کد، `_handle_ConnectionUp` نامیده شده است، یک رویداد را از سوئیچ OpenFlow دریافت کرده و سپس قانون پخش کردن را در درون جدول جریان سوئیچ ذخیره (Cache) می‌کند. اجازه بدھید کنترلر POX را با قابلیت هاب - اترنت آغاز کنیم:

```
mininet@mininet-vm:~/pox$ ./pox.py
forwarding.hub
```

بنابراین نتیجه زیر را خواهیم داشت:

```
POX POX 0.0.0 / Copyright 2011 James McCauley
INFO:forwarding.hub:Hub running.
DEBUG:core:POX 0.0.0 going up...
DEBUG:core:Running on CPython (2.7.3/Sep 26 2012 21:51:14)
INFO:core:POX 0.0.0 is up.
This program comes with ABSOLUTELY NO WARRANTY. This program is
free software, and you are welcome to redistribute it under
certain conditions.
```

```
Type 'help(pox.license)' for details.
DEBUG:openflow.of_01:Listening for connections on 0.0.0.0:6633
Ready.
POX> INFO:openflow.of_01:[Con 1/1] Connected to 00-00-00-00-00-01
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
```

به پاد داشته باشید به محض آنکه کنترلر POX (که به عنوان یک هاب - اترنت عمل می‌کند) شروع به کار کند، یک پیغام از نوع Information ایجاد می‌شود و تأیید می‌کند که سوئیچ OpenFlow به کنترلر POX متصل شده است. سوئیچ دارای یک شناسه DataPath ID با همان ID است که در اینجا با مقدار ۰۰-۰۰-۰۰-۰۰-۰۰-۰۱ نمایش داده شده است. شما می‌توانید به قسمت Command مربوط به Mininet بازگردید و فرمان `net` را برای دیدن عناصر شبکه صادر کنید. خروجی این فرمان، عناصر مختلف را نمایش می‌دهد که از آن جمله می‌توان به کنترلر `co` اشاره کرد. اکنون با استفاده از فرمان `pingall`، ارتباط بین کلیه میزبانها با یکدیگر را مورد بررسی قرار می‌دهیم:

```
mininet> pingall
```

نتیجه چنین خواهد بود:

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

همچنین می‌توان با استفاده از برنامه `dpctl` محتويات جدول جریان سوئیچ OpenFlow خود را مشاهده کرد:

```
mininet@mininet-vm:~$ dpctl dump-flows tcp:127.0.0.1:6634
```

نتیجه چنین خواهد بود:

```
stats_reply (xid=0x2f0cd1c7):
flags=none type=1(flow)
  cookie=0, duration_sec=800s, duration_nsec=467000000s,
  table_id=0,
  priority=32768, n_packets=24,
  n_bytes=1680,
  idle_timeout=0, hard_timeout=0, actions=FLOOD
```

به این ترتیب، توپولوژی شبکه آزمایشی در Mininet آغاز به کار کرده و باعث اتصال سوئیچ OpenFlow به کنترلر POX می‌گردد. در این سناریو، کارکرد این سوئیچ همچون کارکرد یک هاب - اترنت است. نکته

جالب توجه در خصوص اولین Net App ما، این است که تنها با ۱۲ خط کد در Python (که در فایل `hub.py` قرار دارد)، موفق به اجرای قابلیتهای هاب - ارنت شدیم.

ساخت یک سوئیچ یادگیرنده

در بخش قبل مشاهده کردیم که چگونه می‌توانیم با استفاده از سوئیچ OpenFlow در Mininet، یک هاب - ارنت داشته باشیم. اکنون، ما رفتار سوئیچ OpenFlow را تغییر داده و آن را به یک سوئیچ ارنت هوشمند (یادگیرنده) تبدیل می‌کنیم و سپس عملکرد آن را بهبود می‌بخشیم. قبل از آنکه آزمایشگاه خود را راهاندازی کنیم، اجازه دهید عملیات یک سوئیچ یادگیرنده را مرور کنیم. زمانی که یک بسته به هر کدام از پورتهای سوئیچ یادگیرنده می‌رسد، سوئیچ می‌تواند این را فرا گیرد که فرستنده بسته فوق، در آن پورتی قرار گرفته که بسته از آن وارد شده است. لذا سوئیچ از یک جدول Lookup برای نگهداری این اطلاعات استفاده می‌کند. محتوای جدول فوق، آدرس MAC بسته‌ها به همراه پورت مرتبط است که میزبان مورد نظر از طریق آن پورت به سوئیچ متصل شده است. بنابراین سوئیچ، نگاشتی بین آدرس MAC مبدأ بسته و پورتی که بسته از طریق آن وارد سوئیچ شده، ایجاد کرده و این نگاشت را در جدول Lookup خود ذخیره می‌کند. به محض رسیدن یک بسته، سوئیچ به آدرس MAC مقصد بسته نگاه می‌کند و در صورت مشاهده یک انطباق بین آدرس MAC فوق و یکی از سطوح جدول Lookup، سوئیچ، پورت خروجی را شناسایی کرده و بجای پخش کردن بسته بر روی کلیه پورتهای خود، آن را تنها به میزبان مقصد مورد نظر ارسال می‌کند. حال به شبکه مبتنی بر OpenFlow خود باز گردیم. در الگوی OpenFlow، هر بسته ورودی اساساً یک ورودی جریان جدید در جدول جریان سوئیچ OpenFlow، ایجاد می‌کند. به منظور مشاهده این رفتار، ما شبکه تجربی خود را این بار با قابلیت سوئیچ یادگیرنده لایه ۲، دوباره راهاندازی می‌کنیم (کد سوئیچ یادگیرنده در فایل `12_learning.py` قرار دارد).

الگوریتم سوئیچ یادگیرنده از مراحل زیر تشکیل شده است:

- اولین مرحله، استفاده از آدرس MAC مبدأ بسته و پورت سوئیچ، برای بروز کردن جدول Lookup است. این جدول در درون کنترلر در قالب یک جدول Hash نگهداری می‌شود.
- مرحله دوم دور دیختن نوع معینی از بسته‌ها است. بسته‌هایی که EtherType آنها LLDP است یا بسته‌هایی با یک آدرس مقصد فیلتر شده.
- در مرحله سوم، کنترلر برسی می‌کند که آیا آدرس مقصد، یک آدرس Multicast است یا خیر. اگر Multicast بود، بسته پخش می‌شود.
- اگر آدرس MAC مقصد بسته، قبل از درون جدول Lookup نباشد (منظور جدول Hash است که درون کنترلر نگهداری می‌شود)، کنترلر به سوئیچ OpenFlow دستور می‌دهد بسته را روی تمامی پورتها به جز پورت ورودی، پخش کند.

- اگر پورت خروجی همان پورت ورودی باشد، کنترلر به سوئیچ دستور می‌دهد که برای اجتناب از ایجاد حلقه، بسته را دور ببریزد.
 - اگر موارد بالا در مورد وضعیت بسته صدق نکند، کنترلر با استفاده از آدرس MAC مبدأ و پورت فیزیکی مرتبط، فرمان اصلاح ورودی جریان (Flow Mod) در جدول جریان را به سوئیچ ارسال می‌کند. این فرمان به سوئیچ دستور می‌دهد که بسته‌های آتی، که به همان آدرس MAC اشاره دارند، به پورت خروجی مربوط به آن آدرس MAC ارسال شوند؛ به جای آنکه بسته را در کل پورتهای سوئیچ پخش کند.
- برای مشاهده رفتار سوئیچ یادگیرنده در آزمایشگاه خود، باید در ابتدا تنظیمات موجود را پاک کرده و دوباره شبکه خود را راهاندازی کنید:

```
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
... (screen messages are removed)
mininet@mininet-vm:~$ ./pox.py forwarding.12_learning
```

اکنون، با استفاده از یک ترمینال SSH دیگر، به ماشین مجازی Mininet خود متصل شده و کنترلر POX را راهاندازی کنید؛ فرمان زیر الگوریتم سوئیچ یادگیرنده اترنت لایه ۲ را اجرا می‌کند:

```
mininet@mininet-vm:~/pox$ ./pox.py forwarding.12_learning
```

به محض شروع بکار کنترلر POX (مانند آنچه که در سناریوی قبلی، در رابطه با هاب - اترنت دیدیم) می‌توانید مشاهده کنید که سوئیچ OpenFlow به کنترلر متصل می‌شود. حال اگر به کنسول Mininet مراجعه کرده و فرمان `pingall` را صادر کنید، خواهید دید همه میزبانهای h1، h2 و h3 در دسترس هستند:

```
mininet> pingall
```

خواهیم دید که همه میزبانها می‌توانند یکدیگر را Ping کنند:

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

تاکنون رفتار مورد نظر، شبیه هاب - اترنت بوده است. با این حال، اگر ما از جدول جریان سوئیچ (با استفاده از برنامه `dptctl` نمونه برداری کنیم، می‌توانیم مجموعه‌ای از ورودی‌های مختلف جدول جریان را مشاهده کنیم. در حقیقت، ورودی‌های جدول جریان، آدرس‌های MAC مقصد گوناگونی را به همراه پورتهای خروجی (بر روی سوئیچ) مرتبط با آن آدرسها، نشان می‌دهند. در نظر داشته باشد که این

پورتهای خروجی، پورتهای هستند که بسته‌ها از طریق آنها به مقصد دسترسی خواهند داشت. برای مثال بسته‌هایی که آدرس MAC مقصد آنها باشند، به پورت خروجی شماره ۲ ارسال می‌شوند.

```
mininet@mininet-vm:~$ dpctl dump-flows tcp:127.0.0.1:6634
```

نتیجه چنین خواهد بود:

```
stats_reply (xid=0xabab6ce): flags=none type=1(flow)
  cookie=0, duration_sec=7s, duration_nsec=912000000s,
  table_id=0,
  priority=32768, n_packets=1, n_bytes=98,
  idle_timeout=10, hard_timeout=30, icmp, dl_vlan=0xffff, dl_vlan_pcp=0
  x00,
  dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:00:03, nw_src=10.0.0.2
  , nw_dst=10.0.0.3, nw_tos=0x00, icmp_type=0, icmp_code=0, actions=output:3
  ...
  ...
  (more entries are not shown)
```

بیایید نگاهی به کد Python (که در فایل `12_learning.py` قرار دارد) بیندازیم. این کد، عملکرد سوئیچ یادگیرنده اترنت را پیاده‌سازی کرده است. متند اجرایی، طبق معمول شیء `12_learning` را با کنترلر POX مرکزی، ثبت می‌کند. به محض انجام این عمل، شیء `12_learning`، شنونده‌ای اضافه کرده تا مطمئن شود آن شنونده می‌تواند رویدادهای اتصالهای (Connection Up) مربوط به سوئیچ‌های OpenFlow متصل به کنترلر POX را برسی کند. این شیء سپس، شیء سوئیچ یادگیرنده را معرفی کرده و رویداد اتصال با آن شیء را تایید می‌کند. (به کد مشخص شده در بخش زیر توجه کنید):

```
...
...
class l2_learning(EventMixin):
    """
    Waits for OpenFlow switches to connect and makes them learning
    switches.
    """
    def __init__(self, transparent):
        self.listenTo(core.openflow)
        self.transparent = transparent

    def _handle_ConnectionUp(self, event):
        log.debug("Connection %s" % (event.connection,))
        LearningSwitch(event.connection, self.transparent)

    def launch(transparent=False):
        """
        Starts an L2 learning switch.
        """
```

```
core.registerNew(l2_learning, str_to_bool(transparent))
```

توسط شیء سوئیچ یادگیرنده (Learning Switch)، مشاهده می شود که بمحض معرفی آدرس / پورت، جدول Hash ایجاد می گردد (که `self.macToPort = {}` است) و یک شنونده برای پیغامهای Packet-in ثبت می شود (که `connection.addListeners(self)` است). سپس می توانیم متده `_handle_PacketIn (self, event)` که Packet-in Handler قسمت الگوریتم سوئیچ یادگیرنده مربوط به کد، به شرح زیر است:

```
self.macToPort[packet.src] = event.port
    if not self.transparent:
        if packet.type == packet.LLDP_TYPE or
        packet.dst.isBridgeFiltered():
            drop()
            return
        if packet.dst.isMulticast():
            flood()
        else:
            if packet.dst not in self.macToPort:
                log.debug("Port for %s unknown -- flooding" %
                (packet.dst,))
                flood()
            else:
                port = self.macToPort[packet.dst]
            if port == event.port:
                log.warning("Same port for packet from %s -> %s on %s.
                Drop." %
                (packet.src, packet.dst, port), dpidToStr(event.dpid))
                drop(10)
            return
            log.debug("installing flow for %s.%i -> %s.%i" %
            (packet.src, event.port, packet.dst, port))
            msg = of.ofp_flow_mod()
            msg.match = of.ofp_match.from_packet(packet)
            msg.idle_timeout = 10
```

در ادامه قصد داریم شرحی مختصر از الگوریتم بالا را بیان نماییم. اولین مرحله، بروز کردن جدول Hash (جدولی که در آن نگاشت بین آدرس و پورت قرار دارد) است (که

فرستنده را `self.macToPort[packet.src] = event.port` این امر آدرس MAC به پورت سوئیچی مرتبط می کند که بسته از آن پورت وارد سوئیچ شده است. در مراحل بعد، انواع

معینی از بسته ها دور ریخته می شوند. سپس ترافیک Multicast به درستی پخش می شود. حال اگر مقصد بسته در جدول Hash (همان جدول Lookup) در دسترس نباشد، بسته نیز بر روی کلیه پورتها پخش می شود. اگر پورتهای ورودی و خروجی یکی باشند، بسته برای اجتناب از ایجاد حلقه، دور ریخته می شود (`if port == event.port:`). سرانجام یک ورودی جریان مناسب، داخل جدول جریان

سوئیچ OpenFlow نصب می‌شود. بطور خلاصه باید بگوییم برنامه `12_learning.py`، منطق و الگوریتم مورد نیاز برای تغییر رفتار سوئیچ OpenFlow و تبدیل آن به یک سوئیچ یادگیرنده اترنت را پیاده‌سازی می‌کند. در بخش بعد یک قدم فراتر رفته و سوئیچ یادگیرنده خود را به یک فایروال ساده تبدیل می‌کنیم.

شماره ۲ - یک فایروال ساده Net App

در این بخش، قصد داریم Net App مربوط به سوئیچ یادگیرنده را انتخاب کرده و آن را توسعه دهیم تا تصمیمات مربوط به ارسال بسته را براساس قوانین ساده فایروال (که در کنترلر POX نصب می‌کنیم)، اتخاذ کنیم. ما از استقرار این Net App به دنبال دو هدف مهم هستیم. اولین هدف این است که نشان دهیم تغییر رفتار تجهیزات شبکه، تنها از طریق تغییر Net App در کنترلر OpenFlow، تا چه اندازه آسان است. هدف دوم، ارائه اطلاعات بیشتر درباره کنترلر POX است. در یک Net App که نقش یک فایروال ساده را ایفا کرده و در این بخش می‌خواهیم از آن استفاده کنیم، مایلیم براساس ارزش آدرس MAC مبدأ بسته‌ها، سوئیچ، تصمیم به ارسال یا دور ریختن آن بسته بگیرد. شبکه آزمایشی ما در این سناریو، همان شبکه‌ای خواهد بود که در شکل قبل نشان داده شد. با این حال، ما Net App مربوطه یعنی `12_learning.py` را (که سوئیچ یادگیرنده لایه ۲ است) تکمیل می‌کنیم تا عملکرد یک فایروال ساده را اجرا کند. بنابراین برنامه `12_learning.py` را با نام جدیدی کپی می‌کنیم (بطور مثال `simple_firewall.py`) و قوانین و منطق فایروال را بر روی سوئیچ یادگیرنده لایه ۲ اضافه می‌کنیم. در این Net App که نقش فایروال را ایفا خواهد کرد، سوئیچ، آدرس MAC مبدأ بسته‌های دریافتی را بررسی می‌کند و براساس نتیجه مقایسه با قوانین فایروال، بسته را دور ریخته یا ارسال می‌کند. اگر کنترلر تصمیم بگیرد که بسته باید ارسال شود، مانند سناریوی قبل، اقدام به اجرای عملکردهای سوئیچینگ لایه ۲ می‌کند. بنابراین، پس از بروز کردن جدول Lookup سوئیچ یادگیرنده لایه ۲، قدم بعدی چنین خواهد بود که آدرس MAC مبدأ بسته دریافتی را با قوانین فایروال چک کند. این امر تنها نیازمند اضافه کردن تعدادی افزونه ساده به کد سوئیچ یادگیرنده است. ابتدا، ما به یک جدول Hash (جدول Lookup) نیاز داریم تا جفتهای پورت سوئیچ و MAC مبدأ را ذخیره کنیم. این امر با مشخص کردن این که آیا بسته باید ارسال شده یا با دور ریخته شود، برای سوئیچ و MAC مبدأ یک ارزش منطقی `True` یا `False` تعریف می‌کند. کنترلر، در دو وضعیت بسته را دور می‌ریزد:

- در صورتی که ورودی (Entry) فایروال در حالت `False` قرار داده شده باشد (که `FirewallTable(switch, Source MAC) == False` است).

۲. در صورتیکه برای آدرس MAC مبدأ در جدول Hash فایروال، ورودی فایروال وجود نداشته باشد. کنترلر در صورتی تصمیم به ارسال ترافیک می‌گیرد که یک ورودی برای تابع `FirewallTable` با ارزش True وجود داشته باشد. این شرایط را می‌توان به روش زیر، به کد سوئیچ یادگیرنده اضافه نمود:

```
...
# Initializing our FirewallTable
self.firewallTable = {}
# Adding some sample firewall rules
self.AddRule('00-00-00-00-00-01', EthAddr('00:00:00:00:00:01'))
self.AddRule('00-00-00-00-00-01', EthAddr('00:00:00:00:00:03'))

...
...
# Check the Firewall Rules
if self.CheckFirewallRule(dpidstr, packet.src) == False:
    drop()
    return

...
```

متدهای مورد نیاز برای فرایند فایروال کردن سوئیچ را اجرا می‌کند. این متدها اگر جدول فایروال، قانونی برای آدرس MAC مبدأ داشته باشد، ارزش True را باز می‌گرداند.

```
# check if the incoming packet is compliant to the firewall rules
before normal proceeding
def CheckFirewallRule (self, dpidstr, src=0):
    try:
        entry = self.firewallTable[(dpidstr, src)]
        if (entry == True):
            log.debug("Rule (%s) found in %s: FORWARD",
                      src, dpidstr)
        else:
            log.debug("Rule (%s) found in %s: DROP",
                      src, dpidstr)
        return entry
    except KeyError:
        log.debug("Rule (%s) NOT found in %s: DROP",
                  src, dpidstr)
    return False
```

در این مثال، قوانین فایروال به گونه‌ای تنظیم شده‌اند که تنها بسته‌هایی توسط سوئیچ پردازش و ارسال می‌شوند که آدرس MAC آنها ۰۰:۰۰:۰۰:۰۰:۰۰:۰۱ و ۰۰:۰۰:۰۰:۰۰:۰۰:۰۳ باشد و مابقی ترافیک‌های ورودی به سوئیچ، اجازه عبور نداشته و دور ریخته می‌شوند. حال نرم افزار Mininet و کنترلر POX را به همراه Net App فایروال، با استفاده از فرمانهای زیر آغاز کنیم:

```
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovsk
...
controller remote
...
```

یک ترمینال SSH دیگر ایجاد کنید و فرمان زیر را در آن اجرا کنید:

```
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG
forwarding.simple_firewall.py
```

به ياد داشته باشيد ما پارامترهای جدیدی را در خط فرمان، به سمت کنترلر POX ارسال کرده‌ایم تا بتوانیم پیغامهای اشکال زدایی را مشاهده کنیم. خروجی این پیغامها، وضعیت کنترلر POX را در هنگام اجرای Net App فایروال نمایش می‌دهد. بدلیل آنکه در جدول فایروال قانونی وجود ندارد که به میزبان اجازه ارسال ترافیک به دیگر میزبانها را بدهد، ما انتظار داریم که این وضعیت، در خروجی فرمان h2 pingall مشهود باشد:

```
mininet> pingall
```

خروجی ما چنین خواهد بود:

```
*** Ping: testing ping reachability
h1 -> X h3
h2 -> X X
h3 -> h1 X
*** Results: 66% dropped (4/6 lost)
```

همچنین می‌توانیم از پیغامهای اشکال زدایی POX دریابیم که کنترلر براساس ارزش آدرس MAC مبدأ بسته‌های دریافتی روی سوئیچ، اقدام به ارسال یا دور ریختن آن بسته می‌کند. در ضمن این نکته نیز قابل توجه است که وقتی کنترلر تصمیم می‌گیرد بسته‌ای را ارسال کند، قانونی را در جدول جریان OpenFlow ذخیره می‌کند که به موجب آن قانون، به سوئیچ اجازه داده می‌شود آن بسته را به سمت مقصد ارسال کند. تا زمانی که آن ورودی در جدول جریان سوئیچ باقی می‌ماند، تمام بسته‌هایی که با ورودی جریان فوق تعابق دارند، می‌توانند کماکان در سوئیچ ارسال شوند. این ذخیره کردن (که مدت زمان محدودی از حضور ورودی جریان در جدول جریان سوئیچ را شامل می‌شود) باعث بهبود کارائی سوئیچ در ارسال بسته می‌شود. منظور ما از ذخیره کردن، دسترس پذیری یک ورودی جریان در جدول جریان سوئیچ است که اجازه می‌دهد ارسال سریع بسته‌ها بدون دخالت کنترلر، صورت پذیرد. کارائی در ارسال کردن بسته‌ها تنها زمانی دچار آفت می‌شود که اولین بسته از یک جریان^{*} باید منتظر

^{*} مجموعه‌ای از بسته‌ها، با مبدأ و مقصد یکسان، یک جریان را تشکیل می‌دهند. (ترجم)

تصمیم کنترلر در خصوص وضعیت ارسالش به سمت مقصد بماند. این فرایند به عنوان اولین تأخیر یک جریان شناخته می‌شود. اجازه دهد آن را به این روش بازگو کنیم: میزبان ۱ میزبان ۲ را Ping می‌کند. از خروجی Ping می‌توانیم دریابیم که اولین بسته به دلیل آنکه جدول جریان سوئیچ خالی است و سوئیچ OpenFlow باید با کنترلر ارتباط برقرار کند، با تأخیر بالایی مواجه می‌شود. بعد از ذخیره کردن دستور العمل فوق در جدول جریان سوئیچ، بسته‌های بعدی از همان جریان، توسط سوئیچ (و بدون دخالت کنترلر) ارسال می‌شوند. پس از ۲۰ ثانیه ورودی جدول جریان از حافظه Cache سوئیچ پاک شده و دوباره شاهد تأخیر نسبتاً زیادی بین دو میزبان نهایی (که در اینجا h1 و h3 هستند) خواهیم بود، زیرا یک بار دیگر ترافیک به کنترلر تغییر مسیر می‌دهد تا کنترلر دستور ارسال بسته، را به سوئیچ اعلام کند.

فرمان مربوطه چنین است:

```
mininet> h1 ping h3
```

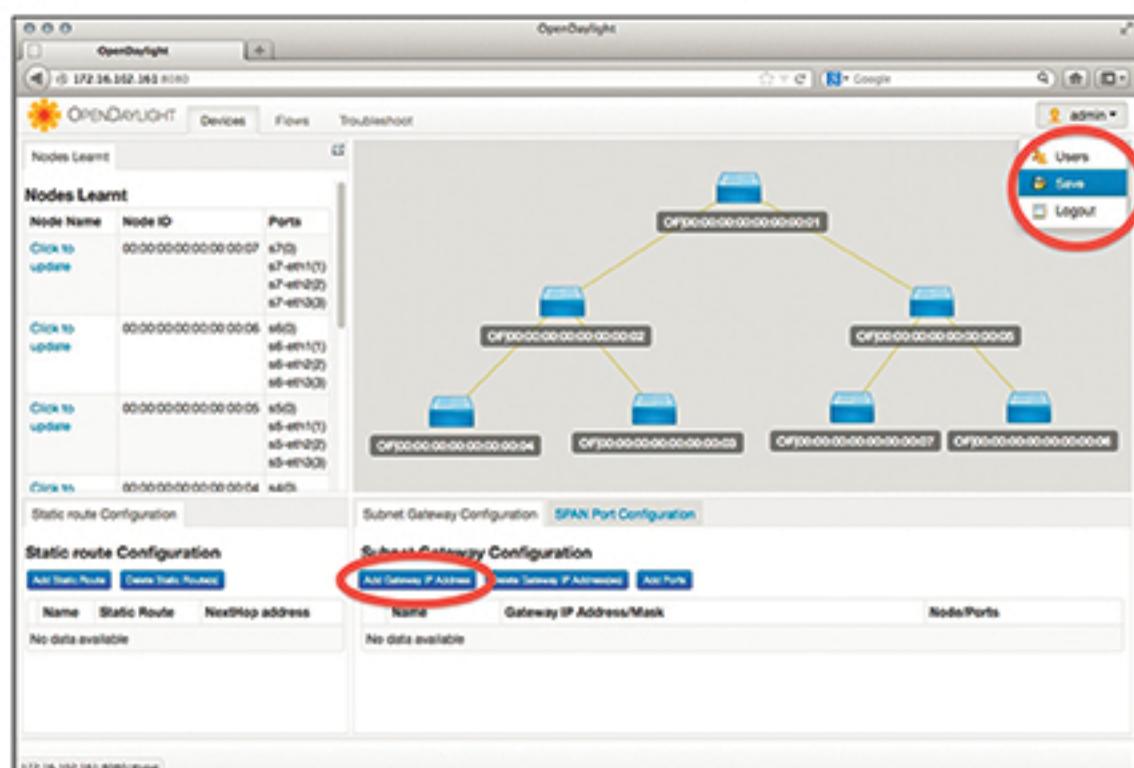
نتیجه چنین خواهد بود:

```
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=38.6 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.264 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=0.056 ms
...
64 bytes from 10.0.0.3: icmp_req=32 ttl=64 time=26.8 ms
64 bytes from 10.0.0.3: icmp_req=33 ttl=64 time=0.263 ms
64 bytes from 10.0.0.3: icmp_req=34 ttl=64 time=0.053 ms
```

OpenDaylight شماره ۲- اپلیکیشن شبکه Net App در فصل ۴، یک آزمایشگاه SDN مبتنی بر کنترلر OpenDaylight را راهاندازی نمودیم. در این بخش به سراغ یک نمونه اپلیکیشن ارسال کننده (Forwarding App) می‌رویم که در نسخه منتشر شده Simple Net App موجود است. کنترلر OpenDaylight دارای یک App به نام Forwarding موجود است که به شما اجازه می‌دهد از خدمات ابتدایی برای اتخاذ تصمیمات مربوط به ارسال کردن بسته و همچنین نصب جریانها در سرتاسر تجهیزات شبکه OpenFlow استفاده کنید. این اپلیکیشن از طریق پیغام ARP به حضور یک میزبان پی برده و ورودی‌هایی با Subnet Mask معادل ۳۲/۳۲ را به همراه پورتهای خروجی مرتبطی که به سمت آن میزبان هدایت می‌شوند، در سرتاسر سوئیچهای شبکه نصب می‌کند. برای دستورالعمل مربوط به راهاندازی آزمایشگاه SDN به فصل ۴ مراجعه کنید. جهت یادآوری دوباره ذکر می‌کنیم که شبکه Mininet باید با فرمان زیر راهاندازی شود:

```
sudo mn --controller=remote,ip=<OpenDaylight IP> --topo tree,3
```

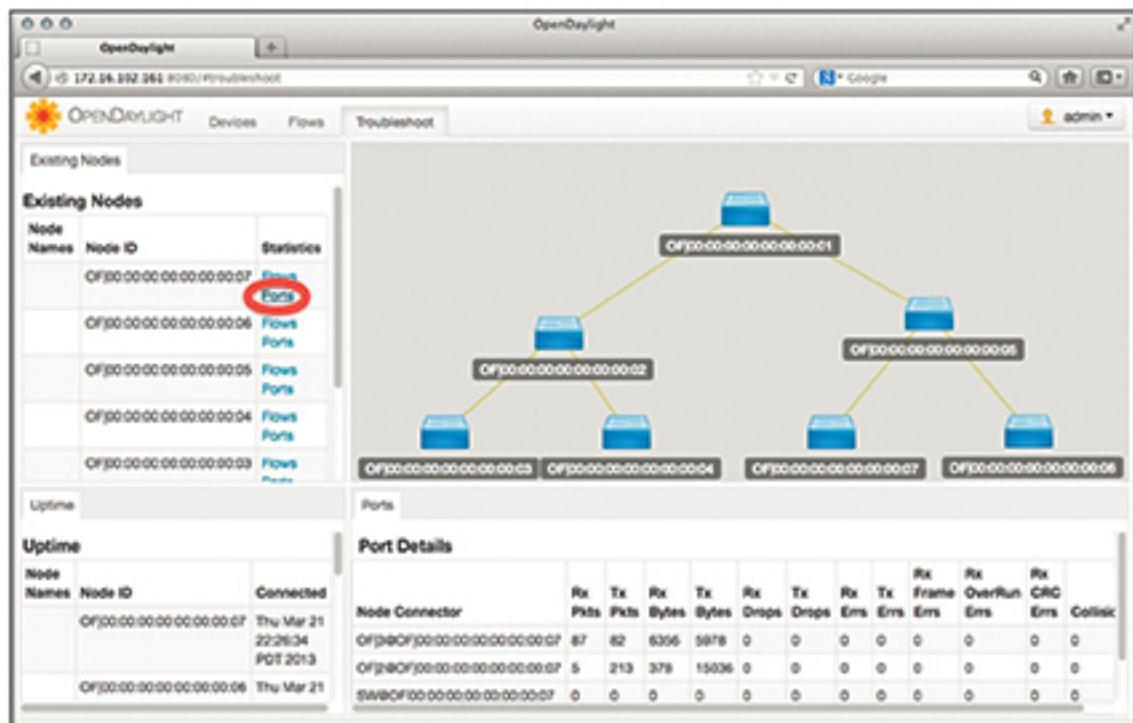
با اجرای کنترلر Mininet و OpenDaylight (به گونه‌ای که در فصلهای پیشین شرح داده شد)، وارد اینترفیس وب OpenDaylight شوید. برای رسیدن به چیدمان منطقی توپولوژی یعنی توپولوژی درختی، تجهیزات را Drag & Drop کرده و سپس این پیکربندی را ذخیره کنید. در نهایت بر روی دکمه Add Gateway IP Address کلیک کرده و آدرس IP و Subnet مربوط به Gateway IP Address، یعنی آدرس 10.0.0.254/8 را اضافه کنید (تصویر بعد را مشاهده کنید). با این اقدام، درخواستها به کنترلر ارسال شده و در نتیجه جدول جریان سوئیچها بروز می‌شوند.



۵-۲
توپولوژی درختی
شبکه Mininet
در محیط ODL

در کنسولی که Mininet در آن در حال اجراست، فرمان `pingall` را صادر کنید تا از دسترس پذیری کلیه میزبانها به یکدیگر اطمینان حاصل کنید. روی منوی Troubleshoot کلیک کرده و جزئیات جریان را برای یکی از سوئیچها بازگزاری کنید. در تصویر بعد جزئیات مربوط به یک پورت را مشاهده می‌کنید.

۵-۳ شکل ۵-۳
جزئیات پورت در
محیط ODL



در کنسول OSGi (اینترفیس خط فرمان کنسول، جایی که کنترلر ODL شروع شده است)، عبارت `ss simple` را تایپ کنید. خواهید دید که اپلیکیشن Simple Forwarding (همانطور که در شکل زیر مشاهده می‌کنید) فعال شده است (خط انتهایی شکل بعد، این وضعیت را نمایش می‌دهد).

۵-۴ شکل ۵-۴
وضعیت اپلیکیشن
Simple Forwarding
در کنسول OSGI

```
osgi> ss simple
"Framework is launched."
id  State      Bundle
45  ACTIVE    org.opendaylight.controller.samples.simpleforwarding_0.4.0.SNAPSHOT
```

خلاصه

در این فصل، چند نمونه از اپلیکیشن‌های شبکه را معرفی کردیم. این اپلیکیشن‌ها سوئیچ OpenFlow و کنترلرهای SDN را به عنوان یک پلتفرم، برای اجرای قابلیت‌های خود مورد استفاده قرار دادند. به طور کلی، ما یک هاب - ارزن - ساده را با استفاده از کنترلر POX راهاندازی کردیم و سپس به عملکرد سوئیچ یادگیرنده لایه ۲ پرداختیم. با افزودن ویژگی‌های منطقی بیشتر به این سوئیچ یادگیرنده، نشان دادیم که به سادگی می‌توانیم با توسعه قابلیت سوئیچ یادگیرنده، عمل تشخیص بسته را به گونه‌ای انجام دهیم که گویی یک فایروال ساده در شبکه داریم. سرانجام Net App ساده‌ای برای ارسال بسته معرفی کردیم که از کنترلر OpenDaylight استفاده می‌کرد. در فصل بعد به مفهوم مجازی‌سازی شبکه و چگونگی داشتن یک برش از شبکه خواهیم پرداخت.