

# ۳

## کنترلرهای OpenFlow

در این فصل به بررسی نقش کنترلرهای OpenFlow، اینترفیسی که از کنترلر به سوئیچ متصل شده (اینترفیس جنوبی) و همچنین API‌هایی که در دسترس Net App‌ها قرار دارند، می‌پردازیم. علاوه بر این به موارد زیر نیز خواهیم پرداخت:

- قابلیتهای کلی کنترلرهای (SDN).OpenFlow
- پیاده‌سازیهای موجود در حوزه کنترلرهای (از جمله Floodlight، NodeFlow، NOX/POX و OpenDaylight)
- کنترلرهای ویژه یا اپلیکیشنها (یعنی که در لایه بالاتر کنترلرهای اصلی قرار می‌گیرند (مانند RouteFlow و FlowVisor)

### کنترلرهای SDN

همانطور که در تصویر بعد آمده است، معماری SDN و بویژه OpenFlow که به جدا سازی Control و Data Plane می‌پردازد را می‌توان با یک سیستم عامل و سخت افزار کامپیوتري مقایسه کرد. کنترلر OpenFlow همچون یک سیستم عامل بوده و یک اینترفیس برنامه‌پذیر (منتظر اینترفیس جنوبی است) را برای سوئیچ‌های OpenFlow (مانند سخت افزار کامپیوتري) فراهم می‌کند. با استفاده از این اینترفیس برنامه‌پذیر، می‌توان اپلیکیشنهاي شبکه را (که از آن به عنوان Net App یاد می‌شود) برای اعمال کنترل و مدیریت وظایف سوئیچ OpenFlow ایجاد کرده و توسعه داد. به یاد دارد که Net App‌ها با استفاده از یک اینترفیس شمالی به کنترلر متصل هستند و کنترلر با استفاده از اینترفیس جنوبی به سوئیچ OpenFlow متصل است. Net App‌ها از طریق کنترلر می‌توانند بر روی سوئیچ‌ها

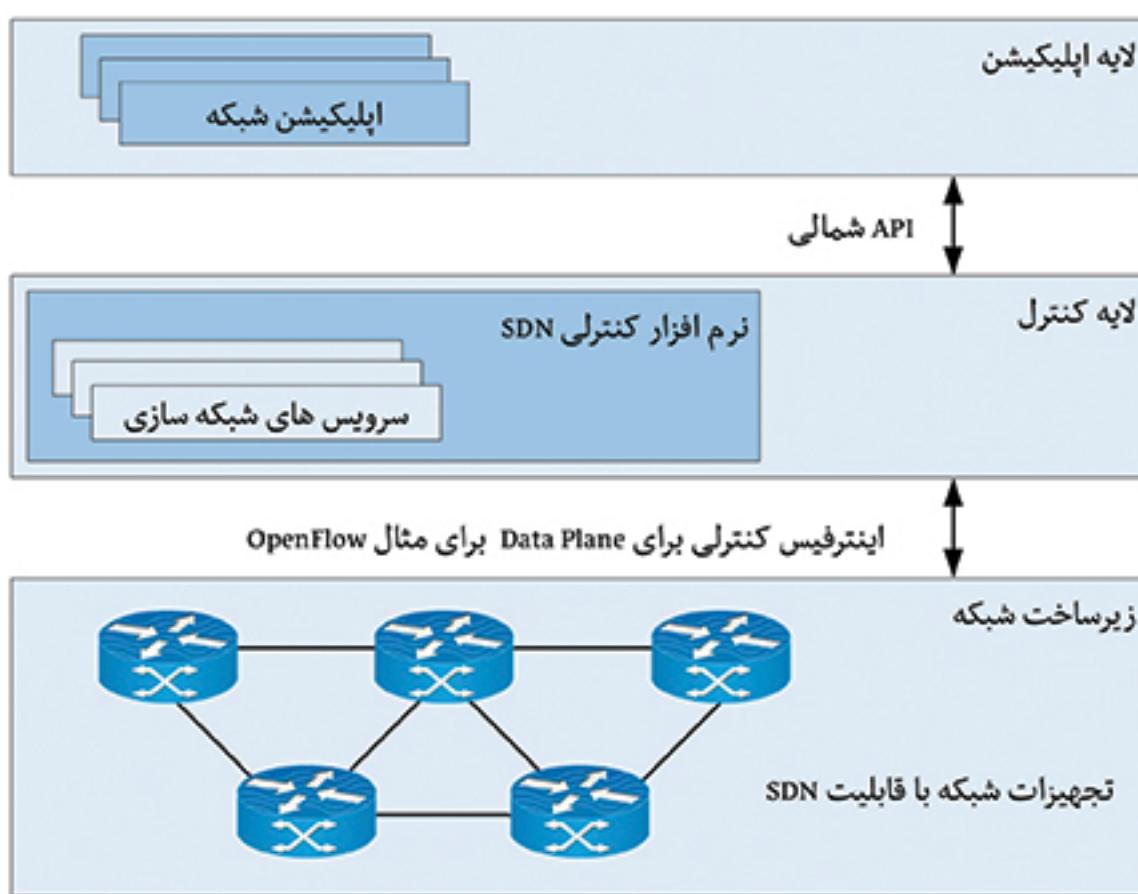
کنترل داشته باشند. Control Plane در SDN و بالاخص OpenFlow از نظر منطقی ساختاری متمرکز دارد و Net App هایی که برای این نوع شبکه ایجاد شده و توسعه داده می‌شوند، معماری‌ای را نمایش می‌دهند که گویی شبکه، یک سیستم واحد است.

حال به نحوه تعاملات بین سویچ و کنترلر می‌پردازیم. براساس یک مدل کنترل واکنشی (Reactive)، سویچ‌های OpenFlow هنگام اتخاذ هر تصمیمی باید با یک کنترلر OpenFlow همفکری کنند؛ مانند زمانی که بسته‌ای از یک جریان جدید وارد سویچ OpenFlow می‌شود (همانطور که در فصول قبل در این مورد توضیحاتی داده شد، این رویداد [Packet\\_in](#) نامیده می‌شود). در چنین شرایطی که ما آن را کنترل تک به تک جریان می‌نامیم، سویچ OpenFlow اولین بسته هر جریان جدید را برای اتخاذ تصمیم (بطور مثال ارسال به پورت خاصی از سویچ یا دور ریختن آن بسته) به کنترلر ارسال می‌کند و پس از آن، بسته‌های بعدی مربوط به همان جریان، براساس فرخ خط داخل سخت افزار سویچ، ارسال می‌شوند؛ در نتیجه تأخیر اندکی در کارائی سویچ خواهیم داشت. اگر چه تأخیری که به موجب ارسال اولین بسته برای تصمیم‌گیری به کنترلر ایجاد می‌شود، در بسیاری از موارد ناقیز است، ولی اگر کنترلر مرکزی OpenFlow از نظر جغرافیایی دور باشد یا اگر بیشتر جریان‌ها کوتاه مدت باشند (مانند جریانهای تک بسته‌ای)، ممکن است تأخیر ایجاد شده چشمگیر و قابل توجه باشد. یک رویکرد پیشگیرانه (Proactive) جایگزین در OpenFlow وجود دارد که می‌تواند باعث افزایش کارائی شبکه و سویچ شود. در این رویکرد، سیاستهایی (Policy Rules) که در کنترلر تعریف شده و وجود دارند، به سویچ‌ها منتقل می‌شوند. در این وضعیت، تصمیم‌گیرنده در رابطه با بسته‌های شبکه، سویچ OpenFlow خواهد بود. همچنین هیچگاه رویداد [Packet-in](#) رخ نداده و بسته به کنترلر ارسال نمی‌شود. اگر چه این رویکرد باعث سهل‌تر شدن کنترل، مدیریت و اعمال سیاستها بر سویچ می‌گردد، ولی از هم فاصله گرفتن این دو عنصر، تبعاتی را به همراه دارد و پیشنهاد می‌شود این دو در کنار یکدیگر قرار گیرند. از طرفی استفاده از رویکرد واکنشی نگرانی‌هایی را بدنبال دارد. اولین نگرانی مهم در خصوص کنترل متمرکز، مقیاس پذیری سیستم و نگرانی دوم تعیین مکان کنترلرهای چندین کنترلر OpenFlow (مانند کنترلرهای NOX-MT، Maestro و Beacon) که در شبکه مدل شده بزرگی با ۱۰۰،۰۰۰ میزبان و نزدیک به ۲۵۶ سویچ انجام شد، نشان داد که تمام کنترلرهای OpenFlow قادر بودند در هر کدام از سناریوهای آزمایشی، هر ثانیه دست کم به ۵۰،۰۰۰ درخواست جریان جدید که از سمت سویچ‌های OpenFlow ارسال می‌شود، رسیدگی کنند. علاوه بر این، کنترلرهای جدید و در حال توسعه OpenFlow مانند Mc-Nettle، مسرونهای قدرتمند چند هسته‌ای را هدف گذاری کرده‌اند و برای حجم‌های کاری بسیار بالای مراکز داده، در حال طراحی و توسعه هستند (به طور مثال، ۲۰ میلیون درخواست جریان در ثانیه و حدود ۵،۰۰۰ سویچ). پس در این خصوص جای نگرانی وجود ندارد و می‌توان به مدل واکنشی اعتماد داشت.

در شبکه‌های قدیمی که مبتنی بر سوئیچینگ بسته‌ها بوده‌اند، هر بسته به طور متداول حاوی اطلاعات مورد نیاز برای سویچ شبکه است، که براساس آن اطلاعات، سویچ تصمیمات مجزای مسیریابی را بر

روی آن بسته اتخاذ می‌کند. این اطلاعات شامل آدرس MAC و آدرس IP می‌باشد. با این حال بیشتر اپلیکیشنها، داده‌ها را در قالب جریانی از تعداد زیادی بسته مجزا ارسال می‌کنند. کنترل تک به تک بسته‌ها در OpenFlow، حول جریانها عمل می‌کند، یعنی کنترلر بر اساس جریانها تصمیم‌گیری می‌کند، نه بر اساس بسته‌ها. تصمیم گرفته شده برای اولین بسته جریان را می‌توان برای تمام بسته‌های بعدی آن جریان، در Data Plane لحاظ کرد (منظور از Data Plane همان سوئیچ‌های OpenFlow است). این سریار را می‌توان با دسته بندی جریان‌ها کاهش بیشتری داد. به طور مثال می‌توان کل ترافیک بین دو میزبان و تمامی جریان‌هایی که بین این دو ایجاد می‌شود را در یک دسته قرار داد و برای آن یک سیاست در نظر گرفت. در این صورت، سوئیچ، ارجاعات کمتری به کنترلر خواهد داشت.

شکل ۳-۱  
نقش کنترلر  
در SDN



چندین کنترلر می‌توانند در استقرار شبکه، مبتنی بر OpenFlow بکار روند. این نوع پیاده‌سازی مزایایی همچون کاهش زمان تأخیر در پاسخ به سوئیچ از سمت کنترلر، افزایش مقیاس‌پذیری و افزایش تحمل‌پذیری در برابر خرابی کنترلر را به همراه دارد. پروتکل OpenFlow اجازه ارتباط کنترلرهای متعدد با یک سوئیچ را می‌دهد و بدین ترتیب کنترلرهای پشتیبان مجاز می‌شوند تا هنگام بروز خطا و قطع ارتباط کنترلر اصلی، نقش تعريف شده خود را در برابر سوئیچ ایفا کنند. Onix و HyperFlow هر دو

کنترلرهای توزیع شده‌ای هستند که این ویژگی را در خود دارند؛ به گونه‌ای که از نظر منطقی، ساختاری متمرکز اما از نظر فیزیکی ساختاری توزیع شده دارند. این امر به دلیل فعالسازی ارتباط سوئیچ‌ها با کنترلرهای محلی خود، باعث کاهش سریار جستجو می‌شود و در عین حال این اجازه را می‌دهد که اپلیکیشن‌های شبکه (Net App) با رویکرد متمرکز و ساده شده شبکه نوشته شوند. وجه منفی عمدۀ ای که بصورت بالقوه در این روش وجود دارد، نگاه داشتن وضعیت پایدار در کل سیستم توزیع شده، مشکل از کنترل‌ها است. این امر ممکن است باعث شود Net App‌ها (که باور داریم نگاه دقیقی به شبکه دارند)، به دلیل عدم ثبات در وضعیت شبکه، به درستی کار نکنند.

همانند مقایسه‌ای که پیشتر بین سیستم عامل و کنترل صورت گرفت، یک کنترل OpenFlow مانند یک سیستم عامل شبکه عمل می‌کند. در این ساختار (که برای یادآوری آن را تکرار می‌کنیم)، کنترل دست کم باید دو اینترفیس را در خود داشته باشد: یک اینترفیس جنوبی که به سوئیچ‌های OpenFlow اجازه می‌دهد با کنترل ارتباط داشته باشد و یک اینترفیس شمالی که یک API برنامه پذیر را برای کنترل شبکه و اپلیکیشن‌های شبکه ارائه می‌دهد. اینترفیس جنوبی کنونی که در حال حاضر در شبکه‌های SDN موجود است، پروتکل OpenFlow را در خود دارد (در فصل ۲ بدان پرداختیم). سیستم / نرم افزارهای مدیریت و کنترل بیرونی یا سرویس‌های شبکه، ممکن است بخواهند اطلاعاتی درباره شبکه‌های زیربنایی بدست آورند، یا سیاستهایی را اعمال کنند، یا یک جنبه از رفتار شبکه را کنترل کنند. علاوه بر این، یک کنترلر اصلی OpenFlow، ممکن است نیاز به اشتراک اطلاعات مرتبط با سیاستهای امنیتی را با یک کنترلر پشتیبان داشته باشد و یا در بین چندین دامنه کنترل، با کنترلرهای آن دامنه‌ها ارتباط داشته باشد. در حالی که اینترفیس جنوبی (به عنوان مثال OpenFlow یا ForCES) به خوبی تعریف شده است و می‌تواند به عنوان یک استاندارد بالفعل (Defacto) در نظر گرفته شود، متأسفانه تا به امروز برای تعاملات شمالی هیچ استاندارد پذیرفته شده فراگیری وجود ندارد و این پیاده‌سازی در مواردی خاص برای برنامه‌های خاص صورت می‌پذیرد.

### پیاده‌سازی برخی کنترلرهای موجود

در حال حاضر پیاده‌سازی‌های متفاوتی از کنترلرهای OpenFlow (و SDN) وجود دارد که در فصل ۸، به عنوان بخشی از پروژه‌های متن باز، با جزئیات بیشتر بدانها خواهیم پرداخت. در این فصل، به منظور معرفی برخی از کنترلرهای OpenFlow و بررسی احتمالات ممکن جهت انتخاب یک زبان برنامه نویسی برای توسعه اپلیکیشن‌های شبکه، ما خود را به کنترلرهای Floodlight، NodeFlow، POX، NOX (که کنترلر Floodlight خود از Beacon منشعب شده است) و OpenDaylight محدود خواهیم کرد.

## کنترل‌های NOX و POX

کنترل NOX اولین کنترل OpenFlow بوده که با زبان C++ نوشته شده و یک API را به زبان Python فراهم می‌کند. این کنترل پیش زمینه بسیاری از پروژه‌های توسعه و تحقیق در شناخت اولیه OpenFlow و فضای SDN بوده است. NOX دو مسیر جداگانه را برای توسعه خود پیش گرفت:

- NOX کلاسیک
- NOX جدید

NOX کلاسیک سیر شناخته شده‌ای در توسعه خود دارد که عبارت است از پشتیبانی از Python و C++ همراه با مجموعه‌ای از اپلیکیشن‌های شبکه. از آنجاییکه، سیر توسعه NOX کلاسیک متوقف شده است، هیچ نقشه و برنامه‌ایی برای این کنترل وجود ندارد. اما وضعیت NOX جدید متفاوت است. این کنترل تنها از C++ پشتیبانی می‌کند و در مقایسه با NOX کلاسیک، اپلیکیشن‌های شبکه کمتری دارد، ولی کنترل مربوطی است و کد پایه (Codebase) بسیار واضحتر و شفافتری دارد. کنترل POX نسخه ای از NOX است که تنها با Python سازگار است و آن را می‌توان به عنوان یک کنترل متن باز عمومی OpenFlow و نیز پلتفرمی برای توسعه سریع و نمونه‌سازی اپلیکیشن‌های شبکه در نظر گرفت. هدف اصلی POX، فعالیت در محیط‌های آزمایشگاهی است. از آنجاییکه بسیاری از پروژه‌های تحقیقاتی به طور ذاتی عمر کوتاهی دارند، توسعه دهنده‌گان POX سعی کردند رویکرده متفاوت را در پیش بگیرند. به جای آنکه از یک API پایدار با کاربرد محدود پشتیبانی کند، تمرکز آنها روی اینترفیس‌های مناسب و پرکاربرد است. کدهای منبع NOX (و POX) در مخازن کد منبع Git در GitHub قرار داشته و مدیریت می‌شوند. همگن‌سازی (Cloning) مخزن<sup>1</sup> Git، روش توصیه شده و مناسبی برای بدست آوردن NOX و POX است. توسعه دهنده‌گان کنترل POX روش توسعه این محصول را به دو شاخه منشعب کرده‌اند: فعال (Active) و انتشار یافته (Released).

شاخه‌های فعال، شاخه‌هایی هستند که برنامه نویسان فعالانه در حال توسعه کدهای آن هستند. این شاخه‌ها ویژگیهای جدید زیادی را در خود دارند. شاخه‌های انتشار یافته، شاخه‌هایی هستند که توسعه آنها به ندرت انجام می‌پذیرد و بروزرسانی کمی را در خود دارند اما نسخ پایداری به حساب می‌آیند.

جدیدترین نسخه NOX و POX را می‌توانید با استفاده از فرمانهای زیر بدست آورید:

```
$ git clone http://www.github.com/noxrepo/nox
$ git clone http://www.github.com/noxrepo/pox
```

اگر بیاد داشته باشید، در فصل ۲ با استفاده از محیط مدل شده Mininet، یک آزمایشگاه OpenFlow برپا کردیم. در این فصل ما قصد داریم آزمایشی را با یک Net App آغاز کنیم. این Net App، همچون یک دستگاه هاب اترنت ساده رفتار می‌کند. می‌توانید به عنوان تمرین آن را به یک سوئیچ اترنت لایه ۲ پادگیرنده (Learning Switch) تبدیل کنید. در این اپلیکیشن، سوئیچ، هر بسته را برمی‌می‌کند و نگاشت پورت مبدأ را فرا می‌گیرد. این نگاشت، شامل آدرس MAC بسته به همراه پورتی است که بسته از آن وارد سوئیچ شده. اگر آدرس MAC مقصد بسته، قبل از پورت مرتبط نگاشت شده باشد، بسته به همان پورت فرستاده می‌شود؛ در غیر این صورت اگر سوئیچ آدرس MAC مقصد را نداند، بسته به مقصد تمام پورتهای سوئیچ پخش می‌شود. حال می‌خواهیم آزمایش خود را شروع کنیم. اولین مرحله، راه اندازی ماشین مجازی OpenFlow است. سپس لازم است با استفاده از فرمانهای زیر، POX را در ماشین مجازی خودتان دانلود کنید،

```
$ git clone http://www.github.com/noxrepo/pox  
$ cd pox
```

### اجرای یک اپلیکیشن POX

پس از دریافت کنترلر POX از اینترنت، می‌توانید یک نمونه هاب با عملکردی ساده را به صورت آزمایشی در POX به شرح زیر اجرا کنید:

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

این خط فرمان درخواستی را به POX ارسال می‌کند که به موجب آن، قابلیت Verbos Logging را (که باعث ایجاد Log با جزئیات بیشتر شده و اشکال زدایی را تسريع می‌بخشد) فعال کرده تا کامپوننت آغاز به کار کند. کامپوننت of\_tutorial یک هاب - اترنت عمل می‌کند. حال می‌توانید آزمایشگاه OpenFlow را در Mininet، با استفاده از فرمان زیر به کار بیندازید:

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

آماده سازی سناریوی بالا ممکن است اندکی زمانبر باشد. دلیل این تأخیر زمانی است که یک سوئیچ OpenFlow ارتباط خود را با کنترلر از دست می‌دهد. سوئیچ به طور طبیعی بازه زمانی‌ای را که سعی می‌کند به کنترلر متصل شود، حداقل تا ۱۵ ثانیه افزایش می‌دهد. به دلیل آنکه سوئیچ OpenFlow هنوز متصل نشده است، این تأخیر ممکن است بین ۰ و ۱۵ ثانیه به طول بیانجامد. تعیین این مدت زمان می‌تواند توسط کاربر انجام گیرد. اگر این زمان برای انتظار بسیار طولانی باشد، می‌توان سوئیچ را با پارامتر `--max-backoff` به گونه‌ای پیکربندی کرد که بیش از `N` ثانیه انتظار نکشد. پس از مدت

زمانی، اپلیکیشن مشخص می‌کند که ارتباط سوئیچ OpenFlow با کنترلر برقرار شده است. هنگامی که سوئیچ به کنترلر متصل شد، POX پیغام زیر را چاپ خواهد کرد:

```
INFO:openflow.of_01:[Con 1/1] Connected to 00-00-00-00-00-01
DEBUG:samples.of_tutorial:Controlling [Con 1/1]
```

خط اول، متعلق به کنترلر POX است که ارتباطات سوئیچ OpenFlow را مورد رسیدگی قرار می‌دهد.

خط دوم، متعلق به کامپوننت آموزشی مورد استفاده در این آزمایش است.

اکنون قانید می‌کنیم که میزبانها می‌توانند یکدیگر را Ping کنند و تمام میزبان‌ها دقیقاً یک نوع ترافیک را مشاهده می‌کنند. این همان رفتاری است که یک هاب - اترنت از خود نشان می‌دهد. برای انجام این کار، ما پنجره Xterm‌ها را برای هر میزبان ایجاد و ترافیک را در هر کدام مشاهده خواهیم کرد. برای این کار در کنسول Mininet، سه Xterm را راه اندازی کنید: (فراموش نکنید برای اجرایی کردن این فرمان، باید سرویس Xming را بر روی سیستم عامل خود راه اندازی کرده و در زمان اجرای این آزمایش آن را اجرا کنید)

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

هر پنجره Xterm را طوری بچینید که همه آنها در صفحه نمایش باشند. در های Xterm مربوط به میزبانهای h2 و h3، فرمان `tcpdump` را اجرا کنید. ابزاری برای چاپ بسته‌های دریافتی و دیده شده توسط یک میزبان است. فرمان زیر را در میزبان h2 وارد کنید:

```
# tcpdump -xx -n -i h2-eth0
```

و سپس خط فرمان زیر را در h3 اجرا کنید:

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

در پنجره Xterm مربوط به میزبان h1، یک فرمان Ping را صادر کنید:

```
# ping -c1 10.0.0.2
```

حال بسته‌های Ping به کنترلر ارسال می‌شوند و سپس کنترلر، بسته‌ها را بر روی همه اینترفیسها، بجز اینترفیس فرستنده، پخش می‌کند. شما باید بسته‌های ARP و ICMP مرتبط با Ping را در هر دو پنجره Xterm مربوط به میزبانهای h2 و h3 (که `tcpdump` را اجرا می‌کنند)، مشاهده کنید. یک هاب - اترنت ساده در دنیای شبکه به همین صورت عمل می‌کند که تمام بسته‌ها را به همه پورتهای سوئیچ در

شبکه می‌فرستد. حال خواهید دید زمانی که میزبانی وجود نداشته و فرمان Ping را اجرا می‌کنید، چه اتفاقی رخ می‌دهد. از پنجره Xterm مربوط به میزبان h1، فرمان زیر را اجرا کنید:

```
# ping -c1 10.0.0.5
```

شما باید سه درخواست بی‌پاسخ ARP در پنجره Xterm مربوط به میزبانهای h2 و h3 را ببینید. اگر کد شما بعد از آن قطع شود، آن سه درخواست بی‌پاسخ ARP، بیانگر این است که شما ممکن است در حال دور دیختن بسته‌ها در میزبانهای h2 و h3 باشید. حالا می‌توانید Xterm‌ها را ببینید. آزمایش اول انجام شد و مشاهده کردید که چطور می‌توان یک هاب - اترنت ساده را در محیط Mininet پیاده‌سازی کرد. در ادامه قصد داریم آزمایش دوم را آغاز کنیم. به منظور تبدیل رفتار یک هاب به یک سوئیچ یادگیرنده، باید قابلیت سوئیچ یادگیرنده را به فایل `of_tutorial.py` اضافه کنید. به ترتیب SSH رفته و کنترل‌هاب آموزشی را با فشردن دکمه‌های `Ctrl + C`، متوقف کنید. فایلی که شما تغییر خواهید داد `pox/misc/of_tutorial.py` است. در ویرایشگر مورد نظر خود فایل `pox/misc/of_tutorial.py` را باز کنید. همانطور که مشاهده خواهید کرد، کد مذکور، تابع `act_like_hub()` را از بخش Handler برای پیغامهای `packet_in` فرا می‌خواند تا رفتار هاب - اترنت را پیاده‌سازی کند. در همان آدرس یعنی `pox/misc/of_tutorial.py`، شما می‌توانید با تغییر در بخش Handler، از قابلیت `act_like_switch()` استفاده کنید که حاوی طرحی اولیه از آن چیزی است که کد سوئیچ یادگیرنده نهایی شما باید شبیه به آن باشد. هر بار که این فایل را تغییر می‌دهید و ذخیره می‌کنید، از راه اندازی مجدد POX اطمینان حاصل کنید. سپس از فرمانهای Ping استفاده کنید تا مجموع رفتار سوئیچ و کنترلر، به ترتیب زیر بررسی شود:

۱. به عنوان هاب
۲. به عنوان سوئیچ یادگیرنده اترنت مبتنی بر کنترلر
۳. به عنوان سوئیچ یادگیرنده شتاب دهنده جریان

در موارد ۲ و ۳، میزبانهایی که مقصد یک Ping نیستند، پس از مشاهده نخستین درخواست Broadcast، نباید ترافیک `tcpdump` را نمایش دهند.

Python یک زبان پویا و مفسر بوده که در آن هیچ مرحله جداگانه‌ای برای کامپایل وجود ندارد. به همین دلیل تنها نیاز است که کد خود را بروز کرده، ذخیره کنید و آنرا مجددآ اجرا کنید. Python جدول‌های Hash از پیش گنجانده شده‌ای (Built-in) تحت عنوان دیکشنری و همچنین بردارهای (Vector) تحت عنوان لیست دارد. برخی از کارهای متداولی که شما برای سوتیچ یادگیرنده بدان نیاز دارید از قرار زیر است:

برای راه اندازی اولیه یک دیکشنری:

```
mactable = {}
```

برای افزودن یک عنصر به یک دیکشنری:

```
mactable[0x123] = 2
```

برای چک کردن عضویت دیکشنری:

```
if 0x123 in mactable:
```

print 'element 2 is in mactable'

```
if 0x123 not in mactable:
```

print 'element 2 is not in mactable'

برای چاپ پیغام Debug در POX:

```
log.debug('saw new MAC'))
```

برای چاپ پیغام خطأ در POX:

```
log.error('unexpected packet causing system
meltdown!')
```

برای چاپ کلیه متغیرهای عضو و توابع یک شیء:

```
print dir(object)
```

برای کامنت کردن یک خط از کد:

```
#Prepend comments with a #: no // or/**/
```

علاوه بر توابع فوق، شما به برخی جزئیات درباره API‌های POX نیاز دارید که این جزئیات برای توسعه سوتیچ یادگیرنده بسیار مفید است. شما می‌توانید منابع بیشتری از Python را در آدرس‌های زیر یافت کنید.

فهرست عملکردهای داخلی در Python:

<http://docs.python.org/2/library/functions.html>

آموزش رسمی Phyton

<http://docs.python.org/2/tutorial/>

ارسال پیغامهای OpenFlow با POX

```
connection.send( ... ) # send an OpenFlow message to
a switch
```



زمانیکه کنترلر ارتباط خود را با یک سوئیچ آغاز می‌کند، یک رویداد [ConnectionUp](#) ارسال می‌شود. کد موجود در مثال، یک شیء جدید با نام [Tutorial](#) ایجاد می‌کند که حاوی یک ارجاع به شیء [Connection](#) مرتبط است. فرمانی که خواهید دید می‌تواند در آینده برای ارسال دستورات (پیغامهای [OpenFlow](#)) به سوئیچ مورد استفاده قرار گیرد:

```
ofp_action_output class
```

این اقدامی است که برای استفاده از پیغامهای [ofp\\_flow\\_mod](#) و [ofp\\_packet\\_out](#) بکار می‌رود و پورت سوئیچی را مشخص می‌کند که مایلید بسته از آن به بیرون فرستاده شود (پورت خروجی). این کلاس می‌تواند شماره پورتهای گوناگون خاصی را در خود داشته باشد. یک مثال از کلامی [OFPP\\_FLOOD](#) است، که بسته‌ها را بر روی تمام پورتهای، به جز پورتی که بسته قبلاً از طریق آن وارد سوئیچ شده است، ارسال می‌کند. مثال پیش رو یک Action را ایجاد می‌کند که به موجب آن، بسته‌ها را به تمامی پورتهای سوئیچ می‌فرستد:

```
out_action = of.ofp_action_output(port = of.OFPP_FLOOD)
ofp_match class
```

اشیای این کلام، فیلد های مربوط به هدر بسته و یک پورت ورودی برای تطابق را مشخص می‌کنند. تمام فیلد ها اختیاری هستند و مواردی که مشخص نشده اند [Wildcard](#) ها هستند که با هر مقداری منطبق می‌شوند. برخی از فیلد های قابل توجه اشیای [ofp\\_match](#) عبارتند از:

- آدرس MAC مبدأ: `dl_src`
- آدرس MAC مقصد: `dl_dst`
- پورت سوئیچی که بسته از آن وارد سوئیچ شده: `in_port`

مثال: یک انطباق (Match) ایجاد کنید که با بسته های رسیده به پورت ۳ سوئیچ، منطبق باشد:

```
match = of.ofp_match()
match.in_port = 3
ofp_packet_out OpenFlow message
```

پیغام [ofp\\_packet\\_out](#)، به سوئیچ دستور می‌دهد که بسته را ارسال کند. بسته ممکن است در کنترلر ساخته شود، یا بسته‌ای باشد که سوئیچ دریافت کرده، آن را بافر کرده و به کنترلر فرستاده است (واز این پس با یک [buffer\\_id](#) ارجاع می‌شود). فیلد های مهم عبارتند از:

- فیلد `buffer_id`: شناسه مربوط به یک بافر است که قصد ارسال آن را دارد.
- اگر یک بسته را به طور کامل ارسال می‌کنید، این فیلد را تنظیم نکنید.

- بایتهای خامی که قصد دارد سوئیچ آنها را ارسال کند. اگر یک بسته بافر شده را ارسال می‌کنید، این فیلد را تنظیم نکنید.
- فهرستی از Action‌هایی که باید بر روی بسته انجام شود (برای این بخش آموزشی، این فیلد فقط Action با نام `ofp_action_output` است).
- اگر بسته را با `buffer_id` به سمت کنترل ارسال می‌کنید، مقدار این فیلد مشخص کننده شماره پورتی است که بسته در ابتدا به آن رسیده بود، در غیر این صورت مقدار این فیلد `OFPP_NONE` خواهد بود.

مثال: متدهای `send_packet()` در کلاس `of_tutorial` از کلاس `send_packet` داده شده است:

```
def send_packet (self, buffer_id, raw_data, out_port, in_port):
    """
        Sends a packet out of the specified switch port.
        If buffer_id is a valid buffer on the switch, use that.
        Otherwise, send the raw data in raw_data.
        The "in_port" is the port number that packet arrived on.  Use
        OFPP_NONE if you're generating this packet.
    """

    msg = of.ofp_packet_out()
    msg.in_port = in_port
    if buffer_id != -1 and buffer_id is not None:
        # We got a buffer ID from the switch; use that
        msg.buffer_id = buffer_id    else:
        # No buffer ID from switch -- we got the raw data
        if raw_data is None:
            # No raw_data specified -- nothing to send!
            return
        msg.data = raw_data

    action = of.ofp_action_output(port = out_port)
    msg.actions.append(action)
    # Send message to switch
    self.connection.send(msg)
ofp_flow_mod OpenFlow message
```

این فرامین به سوئیچ دستور می‌دهند یک ورودی جریان در جدول جریان سوئیچ نصب کند. همانطور که در فصول قبل به آن اشاره شد، ورودی‌های جریان، سطرهایی در داخل جدول جریان سوئیچ هستند که با برخی از مشخصه‌های بسته‌های ورودی منطبق می‌شوند و فهرستی از Action‌ها را بر روی آن بسته‌ها اجرا می‌کنند. این Action‌ها مشابه Action‌هایی هستند که برای پیغام `ofp_packet_out` اجرا می‌شد (که در بخش‌های قبل به آنها اشاره شد). برای آزمایش بالا، تنها موردی که نیاز است فقط

تابع `ofp_action_output` می‌باشد. عمل انطباق با شیء `ofp_match`، توصیف می‌شود. فیلد‌های مهم عبارتند از:

- `idle_timeout`: تعداد ثانیه‌های عدم استفاده (Idle) از یک ورودی جریان، پیش از آن که ورودی جریان فوق حذف شود. بصورت پیش فرض Idle Timeout غیر فعال است.
  - `hard_timeout`: تعداد ثانیه‌ها پیش از آن که ورودی جریان حذف شود. بصورت پیش فرض Hard Timeout غیر فعال است.
  - `actions`: فهرستی از Action‌ها که باید روی بسته‌های منطبق، انجام گیرد. (به طور مثال یک Action معادل `(ofp_action_output)` است).
  - `priority`: هنگامیکه عمل انطباق به طور کامل رخ نمی‌دهد (Wildcarded)، این فیلد Priority است که انطباقهای همپوشان را مشخص می‌کند. مقادیر بالاتر، اولویتهای بالاتری دارند. این مورد برای ورودی‌های کاملاً منطبق و دقیق یا غیر همپوشان اهمیتی ندارد.
  - `buffer_id`: این فیلد مربوط به بافری است که بطور سریع Action‌ها را اعمال می‌کند. برای شرایطی که هیچ Action تعریف نشده باشد، این فیلد نامشخص باقی می‌ماند.
  - `in_port`: اگر از یک استفاده می‌کنید، محتوای این فیلد با پورت ورودی بسته‌ای که بافر شده، مرتبط می‌شود.
  - `match`: یک شیء `ofp_match` است. بصورت پیش فرض این فیلد با هر مقداری منطبق می‌شود، بنابراین احتمالاً باید برخی از فیلد‌های آن را از حالت Wildcard در آورده و با دادن یک مقدار مشخص، آن را تنظیم کنید.
- مثال: یک `flow_mod` ایجاد کرده که بسته‌هایی را که از پورت ۳ وارد سوئیچ شده‌اند، از پورت ۴ خارج کند.

```
fm = of.ofp_flow_mod()
fm.match.in_port = 3
fm.actions.append(of.ofp_action_output(port = 4))
```

برای اطلاعات بیشتر درباره ثابت‌های (Constants) مربوط به OpenFlow، فایل `openflow.h` و همینطور `types/enums/structs` مربوط به اصلی OpenFlow را می‌توانید در آدرس

`~/.openflow/include/openflow/openflow.h` مشاهده کنید. در

صورت تمایل می‌توانید در آدرس

`~/pox/pox/openflow/libopenflow_01.py` در خصوص کتابخانه

POX و البته ویژگی‌های OpenFlow نسخه 1.0 نظرات مشاوره‌ای خود را ارائه

دهید.



کتابخانه‌های POX برای تجزیه (Parse) بسته‌ها مورد استفاده قرار می‌گیرد و هر فیلد بسته را برای زبان Python قابل دسترس می‌سازد. این کتابخانه همچنین می‌تواند برای ساختن بسته‌ها جهت ارسال به کار رود. کتابخانه‌های تجزیه کننده (Parsing Libraries) در آدرس `pox/lib/packet/` موجود هستند.

هر پروتکل فایل تجزیه مرتبط با خود را دارد. برای اولین تمرین، شما تنها نیاز دارید که به فیلدهای مبدأ و مقصد اترنت (Mac-src/Mac-Dst) دسترسی داشته باشید. برای استخراج آدرس مبدأ یک بسته، از فرمان زیر استفاده کنید:

`packet.src`

فیلدهای `src` و `dst` مربوط به اترنت تحت عنوان اشیاء `pox.lib.addresses.EthAddr` ذخیره شده‌اند. این فیلدها به سادگی قابل تبدیل به نمایش رشته‌ای متداول هستند (متغیر `str(addr)`)، چیزی شبیه "01:ea:be:02:05:01" را باز می‌گردانند) یا می‌توانند به صورت نمایش معمول خطی خود، `EthAddr("01:ea:be:02:05:01")` ایجاد شوند. برای مشاهده تمام اعضای شیء یک بسته تجزیه شده، از فرمان زیر استفاده کنید:

`print dir(packet)`

این خروجی برای یک بسته ARP است:

```
['HW_TYPE_ETHERNET', 'MIN_LEN', 'PROTO_TYPE_IP', 'REPLY',
'REQUEST', 'REV_REPLY',
'REV_REQUEST', '__class__', '__delattr__', '__dict__',
['__doc__', '__format__',
 '__getattribute__', '__hash__', '__init__', '__len__',
 '__module__', '__new__']
```

```
'__nonzero__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__',
'__init__', 'err',
'find', 'hdr', 'hwdst', 'hwlen', 'hwsrct', 'hwtype', 'msg',
'next', 'opcode',
'pack', 'parse', 'parsed', 'payload', 'pre_hdr', 'prev',
'protodst', 'protolen',
'protosrc', 'prototype', 'raw', 'set_payload', 'unpack', 'warn']
```

بسیاری از فیلدها برای تمام اشیای زبان Python متداول هستند و می‌توان آن‌ها را نادیده گرفت، ولی این کار می‌تواند یک روش سریع برای اجتناب از مراجعه به مستندات توابع باشد، که ما آنرا به شما توصیه نمی‌کنیم.

## NodeFlow

کنترل NodeFlow (که توسط گری برگر (Gary Berger) رهبر فنی و مدیر ارشد فناوری شرکت Cisco توسعه داده شد) یک کنترل OpenFlow ساده و مینیمالیستی است که با زبان جاوا اسکریپت (Server-Side System) برای محیط Node.js نوشته شد. Node.js یک سیستم نرم افزاری سمت سرور (HTTP). است که جهت نوشن اپلیکیشن‌های اینترنتی مقیاس‌پذیر بکار می‌رود (به طور مثال سرورهای این سیستم را می‌توان به عنوان بسته‌ای تلفیقی از موتور جاوا اسکریپت نسخه ۸ گوگل، لایه انتزاعی پلتفرم Libuv و یک کتابخانه مرکزی محسوب کرد که در جاوا اسکریپت نوشته می‌شود. سیستم Node.js از یک مدل I/O رویداد محور<sup>۱</sup> (Event-Driven) (مسدود نشدنی<sup>۲</sup>) استفاده می‌کند که آن را سبک و کارا می‌سازد و برای اپلیکیشن‌هایی از نوع Data-Intensive و بلادرنگ، که در سرتاسر تجهیزات توزیع شده در یک شبکه اجرا می‌شوند، گزینه بسیار مناسبی است. برنامه‌ها در سمت سرور با زبان جاوا اسکریپت نوشته می‌شوند و با استفاده از I/O ناهمگام<sup>۳</sup> رویداد محور، سریار را به حداقل و مقیاس‌پذیری را به حداقل می‌رسانند. بنابراین برخلاف بسیاری از برنامه‌های جاوا اسکریپت، برنامه NodeFlow در یک مرورگر وب اجرا نمی‌شود. در عوض، همانند یک اپلیکیشن جاوا اسکریپت سمت سرور اجرا می‌شود. NodeFlow در واقع برنامه بسیار ساده‌ای است و به شدت به یک سیستم مفسر پروتکل به نام OFLIB-NODE متکی است، که توسط زولتان لاجوس کیس

<sup>۱</sup> برنامه‌نویسی رویداد محور یک الگوی برنامه‌نویسی است که در آن، روند اجرای برنامه توسط رویدادها مشخص می‌شود. (متترجم)

<sup>۲</sup> الگوریتمی است که براساس آن از مسدود شدن جریان برنامه جلوگیری می‌شود. (متترجم)

<sup>۳</sup> I/O ناهمگام (Asynchronous I/O) نوعی روش I/O داده‌ها بوده که با جلوگیری از مسدود شدن یک فرایند، به آن اجازه می‌دهد تا در حین انجام عملیات I/O، بتواند پردازش انجام دهد. (متترجم)

NodeFlow (Zoltan LaJos Kis) نوشته شده است. یک سیستم تجربی و آزمایشگاهی است که کدهای منبع در GitHub همراه با انشعابی از کتابخانه‌های OFLIB-NODE، در دسترس است. زیبایی NodeFlow، سادگی آن در اجرای یک کنترلر OpenFlow با کدی کمتر از ۵۰۰ خط است. نفوذ جاوا اسکریپت و کارائی بالای موتور جاوا اسکریپت ورژن ۸ گوگل، به معماران شبکه امکان تجربه ویژگیهای مختلف SDN را در این کنترلر می‌دهد، بدون این که مجبور به استفاده از کد تکراری<sup>۰</sup> لازم برای تنظیم اولیه برنامه نویسی رویداد محور باشند.

سرور NodeFlow (که کنترلر OpenFlow محسوب می‌شود) از طریق فراخوانی تابع `net.createServer`، نمونه‌سازی یک سرور جدید TCP را انجام می‌دهد. آدرس و پورت شنونده (Listening Port) برای اسکریپت شروع (اسکریپت ذکر شده در ادامه)، پیکربندی می‌شوند:

```
NodeFlowServer.prototype.start = function(address, port) {
  var self = this
  var socket = []
  var server = net.createServer()
  server.listen(port, address, function(err, result) {
    util.log("NodeFlow listening on:" + address + '@' + port)
    self.emit('started', { "Config": server.address() })
  })
}
```

پس از ایجاد یک سرور TCP جدید با استفاده از اسکریپت بالا، کنترلر یک Session ID ایجاد می‌کند تا بتواند مسیر هر کدام از ارتباطات سوئیچ را دنبال کند. شنونده رویداد، محتوای سوکت را نگه می‌دارد. هر زمان که داده از کانال سوکت دریافت شود، حلقه اصلی پردازندۀ رویدادها فراخوانده می‌شود. کتابخانه جریان برای بافر داده، مورد استفاده قرار می‌گیرد و پیغامهای رمز گشایی شده OpenFlow را با شیء `msgs` باز می‌گرداند. شیء `msgs` برای پردازش بیشتر، به تابع `_ProcessMessage` تحويل داده می‌شود:

```
server.on('connection',
  function(socket) {
    socket.setNoDelay(noDelay = true)
    var sessionID = socket.remoteAddress + ":" +
    socket.remotePort
    sessions[sessionID] = new sessionKeeper(socket)
    util.log("Connection from : " + sessionID)

    socket.on('data', function(data) {
      var msgs = switchStream.process(data);
```

<sup>۰</sup> کد تکراری با در اصطلاح BoilerPlate Code به ارجاعاتی به بخشهایی از کد گفته می‌شود که همواره در قسمتهای مختلف برنامه برای یک منظور خاص، تکرار می‌شوند. (متترجم)

```
        msgs.forEach(function(msg) {
            if (msg.hasOwnProperty('message')) {
                self._processMessage(msg, sessionID)
            } else {
                util.log('Error: Cannot parse the message.')
                console.dir(data)
            }
        })
    }
}
})
```

آخرین بخش، رسیدگی کننده رویداد (Event Handler) است. `EventEmitter` ها مربوط به چهارچوب Node.js بوده و برای Trigger کردن فراخوانی بازگشتی (Callback)، استفاده می‌شود. رسیدگی کننده رویدادها منتظر می‌مانند تا رویداد خاصی رخ دهد و سپس پردازش را شروع کنند. `NodeFlow` دو مدل خام (Raw) و مدل مگنت (Magnet) را ارائه می‌کند.

که رویداد اصلی برای گوش دادن به پیغامهای `PACKET_IN` مربوط به `OFPT_PACKET_IN` است. اگر خاطر قان باشد، پیغام `PACKET_IN` مربوط به زمانی است که سوئیچ یک نمونه از OpenFlow بسته دارد و این پیغام به کنترل ارسال می‌کند.

۲. همچنین رویداد **SENDPACKET** که به سادگی پیغامهای OpenFlow را رمزگشایی کرده و می‌فرستد:

```
self.on('OFPT_PACKET_IN',
  function(obj) {
    var packet = decode.decodeethernet(obj.message.body.data, 0)
    nfutils.do_12_learning(obj, packet)
    self._forward_12_packet(obj, packet)
  })
self.on('SENDPACKET',
  function(obj) {
    nfutils.sendPacket(obj.type, obj.packet.outmessage,
    obj.packet.sessionID)
  })

```

یک Net App ساده مبتنی بر NodeFlow می‌تواند یک سوئیچ یادگیرنده باشد (تابع `do_12_learning` که در ادامه ذکر شده است). سوئیچ یادگیرنده تنها آدرس MAC مبدأ را جستجو کرده و در صورتی که آدرس قبلًا در جدول یادگیری (Learning Table) موجود نباشد، این آدرس را به همراه پورت مبدأ، در جدول ارسال درج می‌کند:

```
do_12_learning: function(obj, packet) {  
    self = this
```

```

var dl_src = packet.shost
var dl_dst = packet.dhost
var in_port = obj.message.body.in_port
var dpid = obj.dpid
if (dl_src == 'ff:ff:ff:ff:ff:ff') {
    return
}
if (!l2table.hasOwnProperty(dpid)) {
    l2table[dpid] = new Object() //create object
}
if (l2table[dpid].hasOwnProperty(dl_src)) {
    var dst = l2table[dpid][dl_src]
    if (dst != in_port) {
        util.log("MAC has moved from " + dst + " to " + in_port)
    } else {
        return
    }
} else {
    util.log("learned mac " + dl_src + " port : " + in_port)
    l2table[dpid][dl_src] = in_port
}
if (debug) {
    console.dir(l2table)
}
}

```

سرور NodeFlow با کلیه امکانات، `server.js` نامیده می‌شود که از مخزن Git مربوط به قابل دانلود است. برای اجرای کنترلر NodeFlow، در ابتدا `node.js` را اجرا کنید و سپس سرور NodeFlow را (که `server.js` است) به طور مثال `node.exe server.js` در ویندوز<sup>۱</sup>:

C:\program Files\nodejs>node server.js

## Floodlight

یک کنترلر OpenFlow جاوا محور و مبتنی بر پیاده‌سازی Beacon است که از هر دو نوع سویچ فیزیکی و مجازی OpenFlow پشتیبانی می‌کند. یک کنترلر Beacon OpenFlow مازولار چند پلتفرمی<sup>۲</sup> است که در جاوا نیز پیاده‌سازی می‌شود. این کنترلر از عملیات رویداد محور و همچنین

---

<sup>۱</sup> این ویزگی اشاره به نرم افزارهایی دارد که نتوان اجرا شدن روی چندین پلتفرم مختلف را دارا هستند. بسیاری از اپلیکیشنها توانایی آن را دارند که روی ویندوز یا لینوکس اجراشوند. به این اپلیکیشنها، اپلیکیشن‌های چند پلتفرمی (Cross-Platform) گفته می‌شود. (متترجم)

ویسمانی (Threaded) پشتیبانی می‌کند. Beacon توسط دیوید اریکسون (David Ericson) در دانشگاه استنفورد، به عنوان یک کنترلر OpenFlow مبتنی بر جاوای چند پلتفرمی ساخته شد. Floodlight قبل از این که تحت لیسانس GPL نسخه ۲ قرار گیرد، از Beacon، که دارای لیسانس Apache است، منشعب شد. Floodlight بدون استفاده از چهارچوب<sup>۷</sup> OSGi، مجدد طراحی گردید. بنابراین می‌تواند بدون نیاز به تجربه‌ای با OSGi، ساخته، اجرا و اصلاح شود. علاوه بر این، جامعه Floodlight هم اکنون شامل تعدادی از توسعه دهندگان شرکت Big Switch Networks می‌شود که بطور فعال کنترل را آزمایش و باگ‌ها را رفع می‌کنند و همچنین مشغول ساختن دیگر ابزارها، پلاگینها و طراحی ویژگی‌هایی برای این محصول هستند. کنترلر Floodlight درصد از پلتفرمی برای گونه‌های متعددی از اپلیکیشن‌های شبکه باشد. Net App‌ها از اهمیت ویژه‌ای برخوردارند، زیرا راه حل‌هایی برای مشکلات شبکه در دنیای واقعی ارائه می‌کنند. برخی از Net App‌های Floodlight عبارتند از:

- Virtual Networking Filter، بسته‌هایی را که وارد شبکه می‌شوند و با جریان موجود انطباق ندارند شناسایی می‌کند. این اپلیکیشن تعیین می‌کند آیا مبدأ و مقصد در همان شبکه مجازی هستند یا خیر؛ اگر باشند، اپلیکیشن به کنترلر علامت می‌دهد که فرایند ایجاد ورودی جریان را ادامه دهد. این فیلتر در واقع یک شبکه مجازی ساده مبتنی بر لایه ۲ (مبتنی بر آدرس MAC) است که به کاربر این امکان را می‌دهد تا چندین شبکه لایه ۲ منطقی در یک دامنه شبکه لایه ۲ منفرد ایجاد کند.
- Static Flow Pusher، برای ساختن یک جریان براساس اولین بسته از هر جریان که وارد شبکه می‌شود، بکار می‌رود. این Net App در REST API متعلق به Floodlight ظاهر می‌شود و به کاربر اجازه می‌دهد جریانها را بطور دستی وارد یک شبکه OpenFlow کند.
- The Circuit Pusher، یک جریان ایجاد می‌کند و سوئیچهایی را که در طول مسیر مقصد بسته قرار دارند، تأمین می‌کند. بطور ساده می‌توان این گونه بیان کرد که کنترلر، سوئیچ‌هایی که در بین راه دو دستگاه مبدأ و مقصد قرار دارد را تشخیص داده و یک ورودی جریان دانمی و دو طرفه در تمامی آن سوئیچ‌ها نصب می‌کند.
- Firewall Modules، برای دستگاه‌های موجود در SDN، امنیتی همانند فایروالهای سنتی در شبکه فیزیکی ایجاد می‌کنند. قوانین<sup>۸</sup> ACL کنترل می‌کنند که آیا یک جریان باید برای یک Floodlight مشخص، در سوئیچ ایجاد شود یا خیر. اپلیکیشن فایروال به عنوان یک ماژول

<sup>7</sup> OSGi یک چهارچوب جاوا برای استقرار و توسعه برنامه‌های نرم افزاری ماژولار و همچنین کتابخانه‌ها است. (متترجم)

<sup>8</sup> Access Control List<sup>۹</sup>

پیاده‌سازی می‌شود که مقررات ACL را بر سوئیچهای هایبریدی اعمال می‌کند. مانیتور کردن بسته‌ها با استفاده از پیغامهای Packet-in انجام می‌شود.

- Floodlight را می‌توان با استفاده از کامپوننت<sup>۱</sup> Neutron به عنوان یک پلاگین شبکه برای سیستم عامل آبری OpenStack، بکار برد. پلاگین مربوط به کامپوننت Neutron، یک ماژول Floodlight را از طریق یک REST API (که توسط Floodlight پیاده‌سازی شده است)، نمایش می‌دهد. این راه حل دو کامپوننت دارد: یک ماژول VirtualNetworkFilter در Floodlight (که Neutron API را پیاده می‌کند)، همچنین پلاگین Neutron RestProxy که Floodlight را به Neutron وصل می‌کند. زمانی که یک کنترلر Floodlight در OpenStack ادغام شود، مهندسان شبکه می‌توانند بطور پویا منابع شبکه را همراه دیگر منابع فیزیکی و مجازی کامپیوتری، تدارک ببینند. این امر انعطاف‌پذیری و کارائی کلی سیستم را بهبود می‌بخشد.

برای جزئیات بیشتر و دسترسی به منابع آموزشی، به صفحه FloodLight در آدرس زیر مراجعه کنید:

<http://www.projectfloodlight.org/floodlight/>



### OpenDaylight

کنترلر OpenDaylight پروژه مشترک بنیاد لینوکس بوده که در آن، یک مجموعه گرد هم آمدند تا نیاز مربوط به چهارچوبی مرجع و باز برای برنامه نویسی و کنترل شبکه را، بوسیله یک راه حل SDN متن باز بروز کنند. این پروژه ترکیبی است از توسعه دهنده‌ان جامعه باز، کدهای متن باز و سیستم نظارت پروژه که فرایند تصمیم‌گیری جمعی و آزاد را در موضوعات فنی و تجاری تضمین می‌کند.

OpenDaylight می‌تواند یک کامپوننت مرکزی در انواع معماری‌های SDN محسوب شود. استفاده از یک کنترلر متن باز SDN، کاربران را قادر می‌سازد پیچیدگی عملیات را کاهش دهند، عمر زیر ساخت شبکه موجود را طولانی کنند و خدمات و قابلیتهای جدید را تنها با SDN قابل استفاده کنند. بیانیه کاری پروژه OpenDaylight به شرح زیر است: "OpenDaylight به یک چهارچوب متن باز تحت حمایت صنعت و زیر نظر جامعه مهندسان که شامل کد و معماری بوده کمک می‌کند که این امر باعث سرعت بخشیدن و پیشرفت این پلتفرم SDN قدرتمند و عمومی شده است." OpenDaylight برای

<sup>۱</sup> در فصل ۷ به OpenStack و Neutron بیشتر پرداخته خواهد شد. (متوجه Networking-as-a-Service)

همگان آزاد است. همه می‌توانند در توسعه کد نویسی آن سهیم بوده و در انتخابات کمیته هدایت فنی TSC<sup>۱۰</sup> شرکت کنند و برای هیأت مدیره انتخاب شوند، یا به روشهای گوناگون در امر نظارت بر پیشرفت پروژه کمک کنند. OpenDaylight ترکیب متنوعی از پروژه‌ها خواهد بود. هر پروژه دارای همکاران، اعضای کمیته و یک عضو منتخب کمیته خواهد بود که توسط ناظران برای رهبری پروژه انتخاب می‌شود. اولین کمیته هدایت فنی و رهبران پروژه، مشتمل از متخصصان خواهد بود که کدهای اصلی پروژه را ایجاد کرده‌اند. این امر این اطمینان را خواهد داد که مجموعه، مشتمل از متخصصانی است که بیشترین آشنایی با کدهای ایجاد شده را داشته تا نگرانی در رابطه با پیشرفت و همچنین ارائه مشاوره به اعضای جدید، وجود نداشته باشد. در میان پروژه‌های جدید با چهارچوب<sup>۱۱</sup> Bootstrap، کنترل<sup>۱۲</sup> ODL یکی از پروژه‌های قدیمی است که در فصل بعد، بیشتر در مورد آن صحبت خواهیم کرد و سپس محیط خود را برای توسعه Net App مبتنی بر ODL آماده می‌کنیم.

### کنترلرهای ویژه

علاوه بر کنترلرهای OpenFlow که در این فصل معرفی کردیم، به دو کنترل OpenFlow دیگر تحت عنوان FlowVisor و RouteFlow، با اهداف ویژه نیز می‌پردازیم. کنترل اول یعنی FlowVisor مانند یک پراکسی Transparent بین سوئیچهای OpenFlow و کنترلرهای گوناگون OpenFlow قرار می‌گیرد. این کنترل قادر است برش‌هایی (Slice) در شبکه ایجاد کرده و کنترل هر بخش را به یک کنترل OpenFlow، محل کند. بطور مفصل در فصل ۶ به مبحث برش شبکه با استفاده از FlowVisor خواهیم پرداخت. همچنین این برش‌ها را با اعمال سیاستهای مناسب، از هم مجزا می‌کند. از سوی دیگر کنترل RouteFlow مسیریابی IP مجازی را برای ساخت افزار سازگار با OpenFlow فراهم می‌کند. RouteFlow را می‌توان به عنوان یک اپلیکیشن شبکه در لایه بالاتر کنترلرهای OpenFlow در نظر گرفت. این کنترل از یک اپلیکیشن کنترل OpenFlow، یک سرور مستقل و یک محیط شبکه مجازی که ارتباط یک زیرساخت فیزیکی را باز تولید و موتورهای مسیریابی IP را راه اندازی می‌کند، تشکیل شده است. موتورهای مسیریابی، براساس پروتکلهای پیکربندی مسیریابی (مانند OSFP و BGP)، پایگاه

Technical Steering Committee<sup>۱۳</sup>

<sup>۱۰</sup> Bootstrap یک چارچوب قدرتمند برای توسعه سریعتر و آسانتر صفحات وب می‌باشد. این مجموعه ابزارها برای ایجاد صفحات وب و نرم افزارهای تحت وب شامل دستورات HTML، CSS و توابع جاوا اسکریپت جهت تولید و نمایش فرمها، دکمه‌ها، نسبت‌ها و سایر المانهای مورد نیاز طراحی وب است. از نسخه دوم Bootstrap به بعد طراحی واکنشگرا با رسپانسیو نیز در آن لحاظ شد که موجب نمایش مناسب صفحه در تلفنهای هوشمند و تبلتها می‌گردد. این چارچوب توسط تیم Twitter پیاده‌سازی شده است. (متترجم)

<sup>۱۱</sup> OpenDaylight<sup>۱۴</sup>

اطلاعاتی ارسال کننده<sup>۲۸</sup> FIB را در IP Tables مربوط به لینوکس ایجاد می‌کنند. این کنترلرهای ویژه با جزئیات بیشتر در فصل ۸ معرفی خواهند شد.

## خلاصه

کنترلر OpenFlow از یک سو (اینترفیس جنوبی) پورتهایی را برای سوئیچهای OpenFlow فراهم کرده و از سوی دیگر (اینترفیس شمالی) API مورد نیاز برای توسعه اپلیکیشن‌های شبکه را مهیا می‌سازد. در این فصل کارکرد کلی کنترلرهای OpenFlow ارائه شد و برخی از پیاده‌سازی‌های موجود (NOX / POX) این فصل کارکرد کلی کنترلرهای OpenFlow ارائه شد و برخی از پیاده‌سازی‌های موجود (NOX / POX) با جزئیات بیشتر مورد بررسی قرار گرفتند. NOX اولین کنترلر OpenFlow و NodeFlow (Floodlight) با جزئیات بیشتر مورد بررسی قرار گرفتند. Python نوشته شده است که با زبان Python نوشته شده است. یک Net App بعنوان سوئیچ اترنت یادگیرنده، معرفی شد که مبتنی بر POX API است. یک کنترلر OpenFlow است که در جاوا اسکریپت برای Node.js نوشته شده است. Floodlight یک کنترلر OpenFlow جاوا محور و مبتنی بر پیاده‌سازی Beacon محسوب می‌گردد و با سوئیچهای فیزیکی و مجازی OpenFlow کار می‌کند. RouteFlow و FlowVisor به عنوان کنترلرهای ویژه نیز در این فصل معرفی شدند. اکنون، ما تمامی مواد مورد نیاز برای ایجاد و راهاندازی محیط توسعه SDN / OpenFlow را شرح دادیم. در فصل بعد این محیط، پیاده‌سازی خواهد شد.