دانشگاه شهید بهشتی

دانشکده مهندسی و علوم کامپیوتر

گزارش پژوهشی درس شبکه پیشرفته

پیاده سازی پروتکل OpenFlow در Mininet

نگارش

سهیل ضیائی قهنویه

استاد

دکتر مقصود عباسپور

بهمن ۱۳۹۹

چکیده

پروتکل OpenFlow برای ارتباط کنترلر با سوئیچ ها در شبکه های نرم افزار محور (SDN) ایجاد شده است. در این گزارش بعد از معرفی محیط شبیه سازی Mininet و پروتکل OpenFlow، به تشریح یکی از کنترلرهای مطرح پژوهشی یعنی POX و نحوه پیاده سازی OpenFlow در این کنترلر پرداخته شده است. در پایان به کمک یک مثال گام به گام فراخوانی و اجرای این پیاده سازی ها در Mininet بررسی می شود.

کلیدواژه‌ها

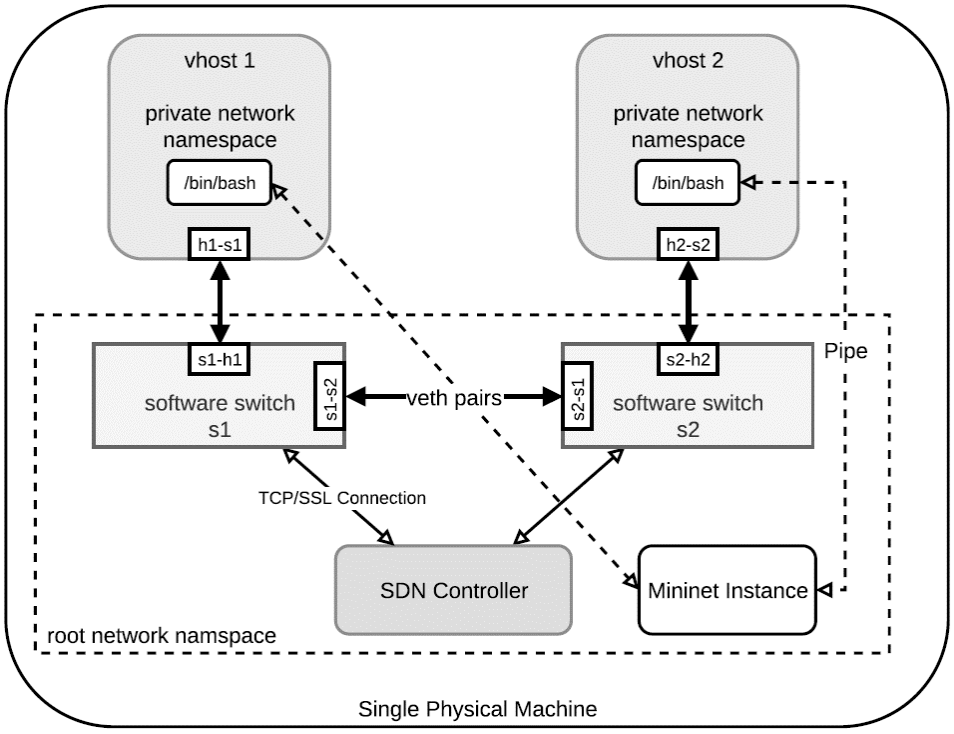
OpenFlow، SDN، Mininet، POX

فهرست نوشتار

بخش اول: مقدمه

معرفی Mininet

Mininet یک سیستم سبک شبیه سازی شبکه برای نمونه سازی سریع محیط شبکه ای کامل است. این برنامه از فناوری های مجازی سازی سطح سیستم عامل برای ایجاد یک شبکه مجازی که هاست ها، سوئیچ ها، روترها و برنامه های شبکه را روی یک ماشین فیزیکی اجرا می کند، استفاده می کند. Mininet با استفاده از پروتکل OpenFlow از شبیه سازی شبکه SDN پشتیبانی می کند. از Mininet به دلیل سادگی و قابلیت تکرار بودن، در آزمایش های آموزشی شبکه در دانشگاه استنفورد استفاده می شود. در شکل زیر یک شبکه Mininet با دو هاست که مستقیماً به یک سوئیچ متصل هستند نشان داده شده است. [1]



Mininet از مکانیزم کانتینر ارائه شده توسط هسته GNU / Linux برای شبیه سازی گره های شبکه استفاده می کند. به طور پیش فرض، همه هاست های Mininet پردازش های عادی هستند که از هسته سیستم عامل، شناسه های پردازش، نام کاربری و

سیستم فایل یکسان استفاده می کنند. هر هاست Mininet دارای پشته شبکه مستقل و منابع مختص خود، از جمله اینترفیس های شبکه، حافظه پنهان، پروتکل رزولوشن آدرس(ARP) ، و جداول مسیریابی است. علاوه بر این هر هاست دارای یک اینترفیس مجازی است که می تواند به یک سوئیچ مجازی (نرم افزاری) مثلا Open vSwitch از طریق یک لینک مجازی با پارامترهای قابل تنظیم (به عنوان مثال پهنای باند، تأخیر یا میزان از دست دادن) متصل شود. در مقایسه با بسترهای آزمایش فیزیکی و شبیه سازهای سنگین مبتنی بر ماشین مجازی، Mininet با استفاده از این فناوری های سبک می تواند مقیاس پذیری را نسبت به توپولوژی های نسبتاً بزرگ (یعنی بیش از صدها گره) فراهم کند. [1]

گردش کار Mininet

سفارشی سازی و شبیه سازی شبکه در Mininet به مراحل زیر نیاز دارد:

۱- ایجاد توپولوژی شبکه

۲- تعامل با یک شبکه

۳- برنامه نویسی کنترلر

این مراحل در ادامه تشریح خواهند شد.

ایجاد توپولوژی شبکه

درMininet ، می توان توپولوژی های پارامتری شده شبکه را با API پایتون آن ایجاد کرد. گره های شبکه و لینک ها را می توان با بازنویسی متد () build کلاس mininet.topo.Topo اضافه و پیکربندی کرد. قطعه کد زیر یک توپولوژی ساده را نشان می دهد که شامل N هاست متصل به یک سوئیچ است:

```
#!/usr/bin/python3
from mininet.topo import Topo
from mininet.node import CPULimitedHost
from mininet.net import Mininet
from mininet.link import TCLink
class SingleSwitchTopo(Topo):
    """ N hosts connected to a single switch """
    def build(self, n):
        switch = self.addSwitch("s1")
        for h in range(n):
            host = self.addHost(
                "h%s" %(h+1),
                ip="10.0.0.%s" % (h+1), cpu=0.5/n)
            self.addLink(switch, host,
```

```
                bw=10, delay="50ms", loss=3,
                max_queue_size=1000, use_htb=True)
    def perfTest():
        topo = SingleSwitchTopo(n=3)
        net = Mininet(topo=topo,
            host=CPULimitedHost, link=TCLink)
        net.start()
        net.pingAll()
        print("Test the bandwidth between h1 and h3")
        h1, h3 = net.get("h1", "h3")
        net.iperf((h1, h3))
        net.stop()
```

علاوه بر پیکربندی انواع گره ها و اتصالات آنها، می توان با بکارگیری کلاسهای ویژه گره و لینک، محدودیت های عملکرد را نیز طراحی کرد. [1]

تعامل با یک شبکه

پس از شروع موفقیت آمیز شبکه، می توان دستورات دلخواه را بر روی هر گره در توپولوژی اجرا کرد. هر هاست در Mininet اساساً یک پردازش پوسته ای است که در فضای نام شبکه خود اجرا می شود. دستورات قابل اجرا را می توان با متد ()cmd هر اینستنس گره به ورودی استاندارد پوسته ارسال کرد. این متد منتظر خروجی دستور می ماند و این خروجی را در قالب رشته برمی گرداند. متدهای اضافی برای برقراری ارتباط با گره ها در API پایتون Mininet ارائه شده است.

Mininet همچنین یک کلاس mininet.cli.CLI داخلی برای ارائه یک رابط خط فرمان (CLI) برای اجرای دستورات تعاملی در هنگام شبیه سازی دارد. می توان رابط خط فرمان را با فراخوانی متد () CLI روی یک اینستنس Mininet فراخوانی کرد CLI (net). گزینه های مفیدی در CLI گنجانده شده است، که می توان به موارد زیر اشاره کرد:

- اسکریپت های پایتون را می توان با دستور py اجرا کرد.

- وضعیت لینک و سوییچ ایجاد شده را می توان با دستورات link و switch پیکربندی کرد.

- تست های اساسی عملکرد، از جمله تست پهنای باند با استفاده از Iperf و تست تأخیر با استفاده از ping را می توان انجام داد.

- پنجره های ترمینال هر گره (به طور پیش فرضXterm) می توانند برای اجرای تعاملی دستورات ایجاد شوند. [1]

شبکه قابل برنامه نویسی با SDN

سوئیچ ها در شبکه های Mininet با استفاده از پروتکل OpenFlow قابل برنامه نویسی هستند. برای استفاده ازOpenFlow ، یک کنترلر SDN باید در شیء Mininet پیکربندی شود. به طور پیش فرض، هنگام نصبMininet ، کنترلر داخلی مرجع استنفورد

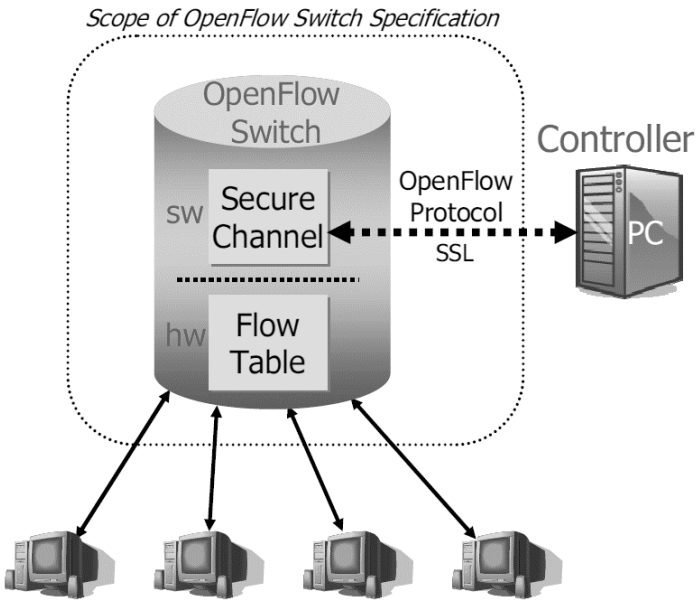
انتخاب می شود. همانطور که در مثال زیر نشان داده شده است، یک کنترلر SDN موجود می تواند با کمک کلاس RemoteController به شبکه اضافه شود. در اینجا باید از RemoteController به عنوان سازنده کلاس استفاده شود.

```
from mininet.net import Mininet
from mininet.topo import SingleSwitchTopo
from mininet.node import RemoteController
from functools import partial
net = Mininet(topo=SingleSwitchTopo,
    controller=partial(RemoteController,ip='127.0.0.1', port=6633)
)
```

برای شروع و متوقف کردن خودکار برنامه کنترل در یک شبیه سازی، باید یک زیر کلاس از mininet.node.Controller با متدهای بازنویسی شده () start و () stop ایجاد شود. [1]

معرفی OpenFlow

OpenFlow یک پروتکل برای ارتباط south bound (بین کنترلر و سوییچ ها) در شبکه های نرم افزار محور SDN می باشد که به میزان وسیعی مورد استفاده و پشتیبانی قرار گرفته است. [2]



جدول جریان (Flow table)

هر سوییچ دارای یک جدول جریان است که در آن تمام قوانین مسیریابی ذخیره شده است. اجزای هر سطر این جدول به شرح زیر است:

۱- فیلدهای تطبیق (Match Field): فیلدهایی از هدر بسته ها که تشخیص جریان از روی آن انجام می شود.

۲- اقدامات (Actions): دستور العمل پردازش بسته ها

۳- اطلاعات آماری، برای کمک به حذف جریانهای غیرفعال

فیلدهای تطبیق (Match Fields) جهت تشخیص جریان ها در سوئیچ ها به شرح جدول زیر می باشد:

In Port	VLAN Id	Ethernet			IP			TCP	
		SA	DA	Type	SA	DA	Proto	Src	Dst

اگر یک بسته با ورودی جدول جریان خاص مطابقت داشته باشد اقدام (Action) خاصی روی بسته انجام می شود. مهمترین اقدامات در جدول جریان به شرح زیر می باشد:

۱- بسته های این جریان به یک پورت مشخص فرستاده شود (Forward).

۲- بسته های این جریان کپسوله شده و به کنترل کننده فرستاده شود تا بتواند تصمیم بگیرد که آیا جریان باید به جدول جریان اضافه شود یا خیر.

۳- بسته های این جریان دور ریخته شود (Drop).

۴- بسته های این جریان به خط پردازش عادی سوئیچ فرستاده شود. [2]

کاربردهای OpenFlow

مثال هایی از استفاده از شبکه های دارای OpenFlow برای آزمایش برنامه های جدید و معماری های شبکه به شرح زیر است:

مثال ۱: مدیریت شبکه و کنترل دسترسی - ایده اصلی Ethane این است که به مدیران شبکه اجازه دهد یک سیاست کلی شبکه در کنترل کننده مرکزی تعریف کنند، که مستقیماً با تصمیم گیری کنترل پذیرش هر جریان جدید اجرا می شود.

مثال ۲: VLAN ها - OpenFlow به راحتی می تواند شبکه مجزای اختصاصی شبیه VLAN در اختیار کاربران قرار دهد. این شرایط با مشخص کردن پورتهای قابل دسترسی توسط ترافیک در یک شناسه VLAN انجام می شود.

مثال ۳: سرویس گیرندگان متحرک VOIP بی سیم - یک کنترل کننده برای ردیابی مکان مشتری پیاده سازی می شود که با برنامه نویسی مجدد جداول جریان، مسیریابی ارتباط را دوباره انجام دهد.

مثال ۴: شبکه غیر IP - به عنوان مثال، جریان ها را می توان با استفاده از هدر اترنت (آدرس های مبدا و مقصد MAC) آنها شناسایی کرد. این امید وجود دارد که یک مسک عمومی (آفست + مقدار + مسک) برای کنترل کننده ها ایجاد شود، که به بسته ها اجازه می دهد تا به روش مشخص شده توسط پژوهشگر پردازش شوند.

مثال ۵: پردازش بسته ها به جای جریان ها - به عنوان مثال، یک سیستم تشخیص نفوذ که همه بسته ها را بازرسی می کند، یا یک

مکانیسم کنترل ازدحام صریح. [2]

معرفی POX

کنترلرهای متن باز متعددی وجود دارد به عنوان مثال، Floodlight، Opendaylight، Faucet، Ryu، Beacon، Pox و Nox. کنترلر مرجع برای پیاده سازی پروتکل OpenFlow در واقع Nox که مبتنی بر C++ است و برادر آن Pox که مبتنی بر Python است بودند. هر دو پیاده سازی پژوهش محور بودند و در حال حاضر استفاده نمی شوند. با این حال، مثال های بسیاری را ارائه می دهند، که می تواند اصلاح شود و متناسب با نیازهای مختلف مورد استفاده قرار گیرد. در مقابل، پیاده سازی های صنعتی، مانند Floodlight، Opendaylight و Faucet، اغلب REST API ارائه می دهند و از طریق یک فایل پیکربندی به فرمت های YANG یا YAML پیکربندی می شوند. [1]

در ادامه گزارش، پیاده سازی پروتکل OpenFlow در POX و فراخوانی این کنترلر در Mininet مورد بررسی قرار خواهد گرفت.

بخش دوم: OpenFlow در POX

همانگونه که اشاره شد مرجع اصلی این بخش مستندات POX Wiki در وب سایت دانشگاه استنفورد به آدرس https://openflow.stanford.edu/display/ONL/POX+Wiki.html بوده است. همچنین یادآوری می شود در این وب سایت اشاره شده که آخرین نسخه مستندات در github به آدرس [3] موجود می باشد.

شناسه مسیر داده (DPID)

مستندات OpenFlow تصریح می کند که مسیرهای داده (سوئیچ ها) هر کدام دارای یک شناسه مسیر داده یا DPID منحصر به فرد هستند که یک مقدار ۶۴ بیتی است و از طریق سوئیچ به کنترلر در هنگام handshaking از طریق پیام ofp_switch_features منتقل می شود. به طور پیش فرض، Mininet شناسه ها (DPIDs) را به صورت ساده ای به سوئیچ ها اختصاص می دهد: اگر نام یک سوئیچ s3 باشد، DPID آن ۳ خواهد بود.

ارتباط با مسیرهای داده (سوئیچ ها)

سوئیچ ها به POX متصل می شوند و پس از آن مشخصاً می توان از طریق POX با آن سوئیچ ها ارتباط برقرار کرد. این ارتباط ممکن است یا از کنترلر به یک سوئیچ یا از یک سوئیچ به کنترلر برسد. هنگامی که ارتباط از طریق کنترلر به سوئیچ است، این کار توسط کد کنترلر انجام می شود که پیام OpenFlow را به یک سوئیچ خاص می فرستد. هنگامی که پیام ها از سوئیچ می آیند، در POX به عنوان رویدادهایی نشان داده می شوند که می توان رسیدگی کننده رویداد (Event handler) برای آنها نوشت. به طور کلی یک نوع رویداد مختص هر نوع پیام که ممکن است سوئیچ ارسال کند وجود دارد. در حالی که پیام ها خود در مستندات OpenFlow توصیف شده اند و رویدادها در زیر بخش های زیر شرح داده شده اند، این زیر بخش به طور دقیق بر نحوه دقیق ارسال این پیام ها و نحوه تنظیم آن دسته از رویدادها متمرکز است.

اساساً دو روش برای برقراری ارتباط با یک مسیر داده در POX وجود دارد: از طریق یک شیء Connection برای آن مسیر داده خاص یا از طریق OpenFlow Nexus که آن مسیر داده را مدیریت می کند. برای هر مسیر داده متصل به POX یک شیء Connection وجود دارد و معمولا یک OpenFlow Nexus وجود دارد که همه اتصالات را مدیریت می کند. در پیکربندی عادی، یک رابطه OpenFlow منفرد وجود دارد که به عنوان core.openflow در دسترس است. همپوشانی زیادی بین اتصالات و Nexus وجود دارد. از هر یک می توان برای ارسال پیام به یک سوئیچ استفاده کرد و اکثر رویدادها در هر دو مورد فعال می شوند.

بعضی اوقات استفاده از یکی یا دیگری راحت تر است. اگر برنامه تمایل به دریافت رویدادها از همه سوئیچ ها دارد، ممکن است منطقی باشد که به Nexus گوش دهد، که رویدادها را برای همه سوئیچ ها فعال می کند. اگر فقط دریافت رویدادها از یک سوئیچ تمایل دارد، گوش دادن به اتصال خاص ممکن است منطقی باشد.

شیء Connection

هر بار که سوئیچ به POX متصل می شود، یک شیء Connection نیز وجود دارد. اگر در کد به آن شیء Connection ارجاع وجود دارد، می توان از متد ()send آن برای ارسال پیام به مسیر داده استفاده کرد. اشیای Connection، علاوه بر امکان ارسال دستورات به سوئیچ ها و ایجاد رویداد از سمت سوئیچ ها، دارای یکسری ویژگی های مفید دیگر هستند که به بعضی از آن ها اشاره می شود:

نام ویژگی	شرح
ofnexus	ارجاعی به شیء nexus مربوط به این connection (معمولا همان core.openflow است).
dpid	شناسه مسیر داده ی سوئیچ
features	پاسخ ofp_switch_features ارسال شده از سوئیچ را در حین handshaking می دهد.
ports	پورت های روی سوئیچ. از آنجا که این موارد ممکن است در طول عمر connection تغییر کنند، POX سعی می کند چنین تغییراتی را ردیابی کند. با این حال، همیشه این احتمال وجود دارد که تعدادی از آن ها از رده خارج شده باشند. این ویژگی ارجاع به یک شیء PortCollection خاص است. این شیء به نوعی مانند دیکشنری است که در آن مقادیر اشیای ofp_phy_port هستند و کلیدها انعطاف پذیر هستند – می توان با توجه به شماره پورت، آدرس اترنت آنها، و نام پورت آن ها جستجو کرد.
sock	شیء سوکت که می توان برای مثال آدرس سمت سوئیچ connection را با connection.sock.getpeername() به دست آورد.
send(msg)	متدی برای ارسال پیام OpenFlow به سوئیچ

اشیای Connection علاوه بر ویژگی های خود و متد ()send، رویدادهایی را متناسب با مسیرهای داده خاص ایجاد می کنند، به عنوان مثال هنگامی که یک مسیر داده یک اعلان را قطع یا ارسال می کند. با ثبت شنونده (listener) رویداد در Connection مربوطه، می توان برای رویدادها در یک مسیر داده خاص رسیدگی کننده (handler) ایجاد کرد.

گرفتن ارجاع به شیء Connection

سه راه برای گرفتن ارجاع به شیء Connection به منظور استفاده از ویژگی های بالا وجود دارد:

۱- گوش کردن به رویدادهای ConnectionUp روی nexus، این رویدادها شیء جدید Connection را پاس می کنند.

۲- می توان با متد getConnection(<DPID>) از nexus یک connection را با DPID مسیرداده پیدا کرد.

۳- از طریق پیمایش ویژگی connections از nexus به همه اتصالات موجود دسترسی داشت.

به عنوان اولین مثال، ممکن است در کلاس کامپوننت کدی وجود داشته باشد که اتصالات را ردیابی کرده و منابع مربوط به آنها را ذخیره کند. این کار را با گوش دادن به رویداد ConnectionUp در OpenFlow nexus انجام می شود. این رویداد شامل ارجاع به connection جدید است که به مجموعه connections خود اضافه می کند. کد زیر این موضوع را نشان می دهد (توجه داشته باشید که یک پیاده سازی کامل تر نیز می خواهد از رویداد ConnectionDown برای حذف connectionها از مجموعه استفاده کند):

```
class MyComponent (object):
    def __init__ (self):
        self.connections = set()
        core.openflow.addListeners(self)

    def _handle_ConnectionUp (self, event):
        self.connections.add(event.connection) # See ConnectionUp event documentation
```

The OpenFlow Nexus

OpenFlow Nexus اساساً مدیر مجموعه ای از اتصالات OpenFlow است. معمولا، یک nexus واحد وجود دارد که اتصالات را به همه سوئیچ ها مدیریت می کند، و از طریق core.openflow در دسترس است.

تعدادی از ویژگی های nexus به شرح زیر است:

نام ویژگی	شرح
miss_send_len	هنگامی که یک بسته با هیچ ورودی جدول در یک مسیر داده مطابقت ندارد، مسیر داده بسته را به کنترلر درون یک پیام OpenFlow هدایت می کند. برای حفظ پهنای باند، مسیر داده در واقع کل بسته را ارسال نمی کند، بلکه فقط به تعداد miss_send_len از اولین بایت های بسته را ارسال می کند. با تنظیم این مقدار در اینجا، هر مسیر داده ای که متعاقباً متصل می شود، پیکربندی می شود تا فقط این تعداد بایت را ارسال کند.
clear_flows_of_connect	وقتیTrue (پیش فرض) باشد، POX هنگام اتصال، تمام جریانهای جدول اول یک سوئیچ را حذف می کند.
connections	مجموعه خاصی که ارجاع به تمام اتصال های این nexus را ذخیره کرده است.

getConnection(<dpid>)	یک connection object خاص یک مسیر داده را با DPID آن به دست می آورد و اگر در دسترس نباشد None بر می گرداند.
sendToDPID(<dpid>,<msg>)	یک پیام OpenFlow به یک مسیر داده خاص ارسال می کند، و اگر مسیر داده متصل نباشد بسته را دور می اندازد و لاگ هشدار ثبت می کند.

مجموعه connections در اصل یک دیکشنری است که کلیدها DPID و مقادیر آن اشیای Connection است. با این حال، اگر آن را پیمایش کنید، برخلاف فرهنگ لغت معمولی، Connection را پیمایش می کند نه DPID ها. برای پیمایش DPID ها می توان از متد()iter_dpids استفاده کرد. علاوه بر این، می توان از اپراتور "in" برای بررسی اینکه آیا یک Connection خاص در این مجموعه وجود دارد و همچنین اینکه آیا یک DPID خاص در مجموعه وجود دارد استفاده کرد. و یک ویژگی()dpids نیز وجود دارد که در واقع همان ()keys است.

همانند اشیای Connection، می توانید شنوندگان رویداد را بر روی خود شیء nexus تنظیم کرد. در حالی که یک شیء Connection فقط رویدادهای مربوط به مسیر داده مرتبط با آن را فعال می کند، شیء nexus رویدادهای مربوط به هر یک از Connection هایی را که مدیریت می کند، فعال می نماید.

رویدادهای OpenFlow: پاسخ به سوئیچ ها

بیشتر رویدادهای مرتبط با OpenFlow در پاسخ مستقیم به پیامی که از سوئیچ دریافت می شود، فعال می شوند. به عنوان یک راهنمای کلی، رویدادهای مرتبط با OpenFlow دارای سه ویژگی زیر هستند:

نام ویژگی	نوع داده	شرح
connection	Connection	کانکشن با سوئیچ مربوطه ای که باعث این رویداد شده است.
dpid	long	شناسه سوئیچ مربوطه که باعث این رویداد شده است.
ofp	ofp_header subclass	پیام OpenFlow که باعث این رویداد شده است.

در ادامه این بخش، برخی از رویدادهای ارائه شده توسط ماژول OpenFlow و ماژول topology را شرح می دهیم. برای شروع، در اینجا یک کامپوننت POX بسیار ساده وجود دارد که از همه سوئیچ ها به رویدادهای ConnectionUp گوش می دهد و در صورت بروز، یک پیام را لاگ می کند. می توانید این قطعه کد را در یک فایل قرار دهید (مثلا ext/connection_watcher.py) و سپس آن را اجرا کنید (با pox.py connection_watcher./) و ببینید سوئیچ ها متصل شوند.

```
from pox.core import core
from pox.lib.util import dpid_to_str

log = core.getLogger()
```

```
class MyComponent (object):
  def __init__ (self):
    core.openflow.addListeners(self)

  def _handle_ConnectionUp (self, event):
    log.debug("Switch %s has come up.", dpid_to_str(event.dpid))

def launch ():
  core.registerNew(MyComponent)
```

رویداد ConnectionUp

برخلاف اکثر رویدادهای OpenFlow دیگر، این رویداد در پاسخ به دریافت یک پیام خاص OpenFlow از یک سوئیچ فعال نمی شود – بلکه به سادگی در پاسخ به ایجاد یک کانال کنترل جدید با سوئیچ اجرا می شود.

همچنین توجه داشته باشید که در حالی که اکثر رویدادهای OpenFlow هم در Connection و هم در OpenFlow Nexus فعال می شوند، رویداد ConnectionUp فقط در nexus ایجاد می شود. منطقی است زیرا رویداد ConnectionUp اولین علامت وجود یک Connection است و شنونده ای بر روی آن تنظیم نشده است.

ویژگی اضافی این رویداد (علاوه بر ویژگی های استاندارد رویداد در OpenFlow) عبارت است از:

نام ویژگی	نوع داده	شرح
ofp	ofp_switch_features	شامل اطلاعات مربوط به سوئیچ، به عنوان مثال انواع اقدامات پشتیبانی شده (مثلاً آیا بازنویسی فیلدها در دسترس است)، و اطلاعات پورت (مثلاً آدرس ها و نام های MAC). این ویژگی در ویژگی features در Connection نیز موجود است.)

این رویداد را می توان به صورت زیر نشان داد:

```
def _handle_ConnectionUp (self, event):
  print "Switch %s has come up." % event.dpid
```

رویداد ConnectionDown

مشابه ConnectionUp اما برخلاف اکثر رویدادهای مرتبط با OpenFlow، این رویداد در پاسخ به یک پیام واقعی OpenFlow فعال نمی شود. بلکه به سادگی هنگامی که اتصال به یک سوئیچ خاتمه یافته است (خواه صراحتاً بسته شده است، یا سوئیچ ریست شده یا غیره) اجرا می شود.

توجه شود که برخلاف ConnectionUp، این رویداد هم در nexus و هم در خود Connection ایجاد می شود. یادآور می

شود که این رویداد هیچ ویژگی ofp ندارد.

رویداد PortStatus

رویدادهای PortStatus هنگامی فعال می شوندکه کنترلر یک پیام وضعیت پورت OpenFlow (ofp_port_status) را از یک سوئیچ دریافت می کند، که نشان می دهد پورت ها تغییر کرده اند. بنابراین، ویژگی ofp آن یک وضعیت ofp_port_status است

```
class PortStatus (Event):
  def __init__ (self, connection, ofp):
    Event.__init__(self)
    self.connection = connection
    self.dpid = connection.dpid
    self.ofp = ofp
    self.modified = ofp.reason == of.OFPPR_MODIFY
    self.added = ofp.reason == of.OFPPR_ADD
    self.deleted = ofp.reason == of.OFPPR_DELETE
    self.port = ofp.desc.port_no
```

یک مثال سریع:

```
def _handle_PortStatus (self, event):
  if event.added:
    action = "added"
  elif event.deleted:
    action = "removed"
  else:
    action = "modified"
  print "Port %s on Switch %s has been %s." % (event.port, event.dpid, action)
```

رویداد FlowRemoved

رویدادهای FlowRemoved هنگامی فعال می شوند که کنترلر پیام حذف جریان OpenFlow (ofp_flow_removed) را از یک سوئیچ دریافت می کند، این پیام ها هنگام حذف یک سطر جدول بر روی سوئیچ یا به دلیل پایان مدت یا حذف صریح ارسال می شوند. این اعلان ها فقط هنگامی ارسال می شوند که جریان با فلگ OFPFF_SEND_FLOW_REM نصب شده باشد.

در حالی که می توانید طبق معمول، مستقیماً از طریق ویژگی ofp رویداد به ofp_flow_removed دسترسی پیدا کنید، این رویداد چندین ویژگی برای راحتی کار دارد:

نام ویژگی	نوع داده	شرح
idleTimeout	bool	True است اگر سطر جدول به خاطر بی استفاده ماندن حذف شده باشد.

hardTimeout	bool	True است اگر سطر جدول به خاطر پایان مدت صریح حذف شده باشد.
timeout	bool	True است اگر سطر جدول به خاطر هر نوع پایان مدتی حذف شده باشد.
deleted	bool	True است اگر سطر جدول صراحتاً حذف شده باشد.

رویدادهای آماری

رویدادهای آماری (Statistics) هنگامی فعال می شوند که کنترل کننده پیام پاسخ آماری OpenFlow (ofp_stats_reply یا OFPT_STATS_REPLY) را از سوئیچ دریافت می کند، که در پاسخ به درخواست آماری ارسال شده توسط کنترلر ارسال می شود.

تعدادی رویداد آماری وجود دارد. اصلی ترین آن ها RawStatsReply است که به سادگی در پاسخ به یک پیام ofp_stats_reply از سوئیچ اجرا می شود. با این حال، این پیام (و بنابراین رویداد مرتبط) مناسب نیست، زیرا کاربر تعیین می کند که نوع رویداد آماری چیست و احتمالاً پاسخ های آماری چند بخشی را "دوباره بهم بچسباند".

برای رفع این مشکل، POX رویدادهای جداگانه ای برای هر نوع پاسخ آماری دارد و این رویدادها با دریافت کل پاسخ (از جمله چندین قسمت احتمالی) فعال می شوند. این رویدادها عبارتند از:

رویداد	نوع آمار OpenFlow
SwitchDescReceived	ofp_desc_stats
FlowStatsReceived	ofp_flow_stats
AggregateFlowStatsReceived	ofp_aggregate_stats_reply
TableStatsReceived	ofp_table_stats
PortStatsReceived	ofp_port_stats
QueueStatsReceived	ofp_queue_stats

هر یک از این رویدادها یک زیر کلاس از کلاس StatsReply است. هنگام مدیریت این رویدادهای مبتنی بر StatsReply، ویژگی stats شامل یک مجموعه کامل از آمار (مثلاً یک آرایه ofp_flow_stats) است. به طور خاص، اطلاعات زیر برای همه زیر کلاس های StatsReply در نظر گرفته می شود:

نام ویژگی	شرح
ofp	از آنجا که یک StatsReply ممکن است چندین پیام OpenFlow منفرد را بهم چسبانده باشد، ویژگی ofp لیستی از پیامهای ofp_stats_reply است. (با این حال، در حالت معمول، این لیست یک رکورد دارد.)
stats	همه آمار منفرد در یک لیست واحد.

رویداد PacketIn

هنگامی که کنترلر پیام ورود بسته OpenFlow (inp_packet_in / OFPT_PACKET_IN) را از سوئیچ دریافت می کند فعال می شود، که نشان می دهد بسته ای که به یک پورت سوئیچ می رسد یا با تمام ورودی های جدول تطبیق نمی شود یا سطر تطبیق شده اقدامی دارد مبنی بر اینکه بسته باید به کنترلر ارسال شود.

علاه بر ویژگی های عادی رویدادها، این رویداد ویژگی های زیر را دارد:

نام ویژگی	نوع داده	شرح
port	int	شماره پورتی که بسته به آن وارد شده است.
data	bytes	دیتای خام بسته
parsed	packet subclasses	ورژن پارس شده pox.lib.packet
ofp	ofp_packet_in	پیام OpenFlow که منجر به این رویداد شده است.

رویداد ErrorIn

وقتی کنترلر خطای OpenFlow (ofp_error_msg / OFPT_ERROR_MSG) را از سوئیچ دریافت می کند، فعال می شود.

علاه بر ویژگی های عادی رویدادها، این رویداد ویژگی های زیر را دارد:

نام ویژگی	شرح
should_log	معمولاً، یک خطای OpenFlow منجر به یک پیام لاگ می شود. با مقداردهی این ویژگی به False پیام لاگ پیش فرض غیرفعال می شود.
asString()	خطا را به فرمت رشته شکل می دهد.

رویداد BarrierIn

هنگامی که کنترلر پاسخ OpenFlow (OFPT_BARRIER_REPLY) را از سوئیچ دریافت می کند، فعال می شود، که نشان می دهد سوییچ پردازش دستورات ارسال شده توسط کنترلر را قبل از درخواست barrier مربوطه به پایان رسانده است.

علاه بر ویژگی های عادی رویدادها، این رویداد ویژگی زیر را دارد:

نام ویژگی	نوع داده	شرح
xid	integer	شناسه تراکنش. برای رویدادهایی که به دستورات ارسال شده توسط کنترلر پاسخ می دهند، این مقدار همان مقدار xid دستور را خواهد داشت. به عنوان مثال، xid مربوط به یک BarrierIn همان مقداری خواهد بود که در پیام ofp_barrier_quest استفاده شده است.

پیام های OpenFlow

پیام های OpenFlow نحوه ارتباط سوئیچ های OpenFlow با کنترلرها است. POX شامل کلاسها و ثابتهای مربوط به عناصر پروتکل OpenFlow است که در فایل pox/openflow/libopenflow_01.py تعریف شده است (01 اشاره به ورژن OpenFlow که در POX پشتیبانی می شود دارد). در بیشتر قسمتها، نامها همان موارد مندرج در مستندات هستند.

تعدادی مهمترین پیام های OpenFlow عبارتند از:

پیام ارسال بسته ها از سوئیچ ofp_packet_out

هدف اصلی این پیام، آموزش سوئیچ برای ارسال یک بسته(یا در صف قرار دادن آن) است. با این حال می تواند به عنوان روشی برای آموزش سوئیچ برای دور انداختن یک بسته بافر شده نیز مفید باشد (با تعیین نکردن هیچ اقدامی برای آن).

نام ویژگی	نوع داده	پیش فرض	شرح
buffer_id	int/None	None	شناسه بافری که بسته در آن در مسیر داده ذخیره شده است. اگر بافر با شناسه ارسال مجدد نمی شود، از None استفاده گردد.
in_port	int	OFPP_NONE	در ارسال مجدد بسته، پورت ورودی که بسته به آن وارد شده است.
actions	list of ofp_action_XXXX	[ ]	لیست اقدامات
data	bytes / ethernet / ofp_packet_in	''	داده ای که باید ارسال شود (یا None اگر یک بافر موجود از طریق buffer_id ارسال می شود).

پیام اصلاح جدول جریان سوئیچ ofp_flow_mod

```
class ofp_flow_mod (ofp_header):
  def __init__ (self, **kw):
    ofp_header.__init__(self)
    self.header_type = OFPT_FLOW_MOD
    if 'match' in kw:
      self.match = None
    else:
```

```
      self.match = ofp_match()
    self.cookie = 0
    self.command = OFPFC_ADD
    self.idle_timeout = OFP_FLOW_PERMANENT
    self.hard_timeout = OFP_FLOW_PERMANENT
    self.priority = OFP_DEFAULT_PRIORITY
    self.buffer_id = None
    self.out_port = OFPP_NONE
    self.flags = 0
    self.actions = []
```

ویژگی های این پیام به شرح زیر است:

نام ویژگی	نوع داده	شرح
cookie	int	شناسه برای این قانون جریان. (اختیاری)
command	int	یکی از مقادیر زیر: OFPFC_ADD – یک قانون به مسیر داده اضافه کند (پیش فرض) OFPFC_MODIFY – قوانین تطبیق را اصلاح کند OFPFC_MODIFY_STRICT – قوانینی را اصلاح کند که کاملاً با مقادیر wildcard مطابقت داشته باشند. OFPFC_DELETE – همه قانون های تطبیق را حذف کند. OFPFC_DELETE_STRICT – قوانینی را که کاملاً با مقادیر wildcard مطابقت دارند حذف کند.
idle_timeout	int	اگر قانون در ظرف مدت این زمان بر حسب ثانیه تطبیق نداشته باشد، منقضی می شود. مقدار OFP_FLOW_PERMANENT به این معنی است که این مدت زمان وجود ندارد (پیش فرض).
hard_timeout	int	قانون پس از این زمان بر حسب ثانیه منقضی می شود. مقدار OFP_FLOW_PERMANENT به این معنی است که هرگز منقضی نمی شود (پیش فرض)
priority	int	اولویت تطبیق یک قانون، اعداد بالاتر دارای اولویت بالاتر. توجه: تطبیق های دقیق بیشترین اولویت را دارند.

buffer_id	int	یک بافر در مسیر داده که جریان جدید به آن اعمال می شود. از None برای هیچ استفاده شود. این مورد برای حذف جریان معنی ندارد.
out_port	int	این فیلد برای مطابقت با دستورات DELETE استفاده می شود. ممکن است از OFPP_NONE برای نشان دادن عدم وجود محدودیت استفاده شود.
flags	int	بیت فیلد عدد صحیح که می توان فلگ های زیر را با آن مقدار دهی کرد: OFPFF_SEND_FLOW_REM – با انقضای قانون، پیام حذف شدن جریان را به کنترلر ارسال کند OFPFF_CHECK_OVERLAP – هنگام نصب هم پوشانی سطر ها بررسی شود. در صورت وجود هم پوشانی، یک خطا به کنترلر ارسال گردد. OFPFF_EMERG – این جریان را به عنوان یک جریان اضطراری در نظر بگیرد و فقط در صورت قطع اتصال کنترلر به سوئیچ از آن استفاده کند.
actions	list	شیء اقدام مورد نظر به این لیست اضافه می شود و به ترتیب اجرا می شوند.
match	ofp_match	ساختار تطبیق (در ادامه مورد بحث قرار خواهد گرفت).

مثال: نصب یک سطر در جدول جریان

```
# Traffic to 192.168.101.101:80 should be sent out switch port 4

# One thing at a time...
msg = of.ofp_flow_mod()
msg.priority = 42
msg.match.dl_type = 0x800
msg.match.nw_dst = IPAddr("192.168.101.101")
msg.match.tp_dst = 80
msg.actions.append(of.ofp_action_output(port = 4))
self.connection.send(msg)

# Same exact thing, but in a single line...
self.connection.send( of.ofp_flow_mod( action=of.ofp_action_output( port=4 ),
                                       priority=42,
                                       match=of.ofp_match( dl_type=0x800,
                                                 nw_dst="192.168.101.101",
                                                 tp_dst=80 )))
```

مثال: پاک کردن جدول جریان همه سوئیچ ها

```
# create ofp_flow_mod message to delete all flows
# (note that flow_mods match all flows by default)
msg = of.ofp_flow_mod(command=of.OFPFC_DELETE)

# iterate over all connected switches and delete all their flows
for connection in core.openflow.connections: # _connections.values() before betta
  connection.send(msg)
  log.debug("Clearing all flows from %s." % (dpidToStr(connection.dpid),))
```

پیام درخواست اطلاعات آماری از سوئیچ ofp_stats_request

```
class ofp_stats_request (ofp_header):
  def __init__ (self, **kw):
    ofp_header.__init__(self)
    self.header_type = OFPT_STATS_REQUEST
    self.type = None # Try to guess
    self.flags = 0
    self.body = b''
```

نام ویژگی	نوع داده	شرح
type	int	نوع درخواست آمار (مثلاً OFPST_PORT). پیش فرض این است که بر اساس body حدس زده شود.
flags	int	هیچ فلگی در OpenFlow 1.0 تعریف نشده است.
body	flexible	متن اصلی درخواست آمار. این می تواند یک شیء بایتی خام یا یک کلاس قابل بسته بندی باشد (مثلاً ofp_port_stats_request)

مثال: اطلاعات آماری جریان وب

جدول جریان و اطلاعات dump مربوط به ترافیک وب را از یک سوییچ درخواست کند. این مثال قرار است همراه با مولفه forwarding.l2_learning اجرا شود. می تواند در مفسر تعاملی POX کپی شود (اگر POX ی که شامل مولفه py اجرا شود).

```
import pox.openflow.libopenflow_01 as of
log = core.getLogger("WebStats")

# When we get flow stats, print stuff out
def handle_flow_stats (event):
  web_bytes = 0
```

```
  web_flows = 0
  for f in event.stats:
    if f.match.tp_dst == 80 or f.match.tp_src == 80:
      web_bytes += f.byte_count
      web_flows += 1
  log.info("Web traffic: %s bytes over %s flows", web_bytes, web_flows)

# Listen for flow stats
core.openflow.addListenerByName("FlowStatsReceived", handle_flow_stats)

# Now actually request flow stats from all switches
for con in core.openflow.connections: # make this _connections.keys() for pre-betta
  con.send(of.ofp_stats_request(body=of.ofp_flow_stats_request()))
```

ساختار تطبیق (Match Structure)

OpenFlow یک ساختار تطبیق ofp_match تعریف می کند که این امکان را می دهد که مجموعه ای از سرآیند ها برای بسته ها مطابقت تعریف شود. ساختار مطابقت در pox/openflow/libopenflow_01.py در کلاس ofp_match تعریف شده است. ویژگی های مهم این کلاس در جدول زیر خلاصه شده اند:

نام ویژگی	شرح
in_port	شماره پورتی از سوئیچ که بسته به آن رسیده است.
dl_src	آدرس اترنت مبدأ
dl_dst	آدرس اترنت مقصد
dl_vlan	شناسه VLAN
dl_vlan_pcp	اولویت VLAN
dl_type	Ethertype / length (e.g. 0x0800 = IPv4)
nw_tos	نوع سرویس IP
nw_proto	پروتکل IP، برای مثال ۶ به معنای TCP است.
nw_src	آدرس IP مبدأ
nw_dst	آدرس IP مقصد
tp_src	پورت TCP/UDP مبدأ

tp_dst	پورت TCP/UDP مقصد

ویژگی ها را می توان بر روی یک شیء تطبیق یا در هنگام مقداردهی اولیه آن مشخص کرد. یعنی موارد زیر معادل هستند:

```
my_match = of.ofp_match(in_port = 5, dl_dst = EthAddr("01:02:03:04:05:06"))
#.. or ..
my_match = of.ofp_match()
my_match.in_port = 5
my_match.dl_dst = EthAddr("01:02:03:04:05:06")
```

تطبیق جزئی و Wildcardها

فیلدهای مشخص نشده به صورت wildcard هستند و با هر بسته ای مطابقت دارند. با تنظیم آن روی None، می توان یک فیلد را به صورت wildcard تنظیم کرد.

فیلدهای آدرس IP کمی پیچیده تر هستند، زیرا می توانند مانند سایر فیلدها کاملاً wildcard شوند، یا بخشی از آن ها نیز wildcard باشند. با این کار می توان کل زیرشبکه ها را مطابقت داد. برای انجام این کار چندین راه وجود دارد که برخی از راه های معادل آن به شرح زیر است:

```
my_match.nw_src = "192.168.42.0/24"
my_match.nw_src = (IPAddr("192.168.42.0"), 24)
my_match.nw_src = "192.168.42.0/255.255.255.0"
my_match.set_nw_src(IPAddr("192.168.42.0"), 24)
```

به طور خاص، توجه داشته باشید که ویژگی های nw_src و nw_dst هنگام کار با تطابق های جزئی می توانند مبهم باشند – به ویژه هنگام خواندن یک ساختار تطبیق (به عنوان مثال، همانطور که در یک پیام flow_removed یا پاسخ flow_stats برگردانده شده است). برای رفع این مورد، می توان از ()get_nw_src، ()set_nw_src و معادل های مقصد آن ها استفاده کرد. این توابع یک تاپل (IPAddr("192.168.42.0"), 24) را برمی گردانند که شامل تعداد بیت های تطبیق است – عددی که در نمایش CIDR بعد از ممیز نشان داده می شود (192.168.42.0/24).

توجه داشته باشید که برخی از فیلدها پیش نیاز دارند. اساساً این بدان معناست که نمی توان فیلدهای لایه بالاتر را بدون تعیین فیلدهای مربوط به لایه پایین نیز تعیین کرد. به عنوان مثال، شما نمی توانید در پورت TCP تطبیق ایجاد کنید بدون اینکه مشخص کنید که می خواهید تطبیق ترافیک TCP داشته باشید. و برای تطبیق ترافیک TCP، باید مشخص کنید که می خواهید ترافیک IP را تطبیق دهید. بنابراین، به عنوان مثال تطبیق فقط با tp_dst = 80، نامعتبر است. همچنین باید nw_proto = 6 (TCP) و dl_type = 0x800 (IPv4) را مشخص کنید. اگر این مورد نقض شوند، پیام هشدار دریافت خواهد شد.

متدهای ofp_match

نام متد	شرح

from_packet(packet, in_port=None, spec_frags=False)	تعریف تطبیق از روی بسته موجود (بخش بعدی)
clone()	یک کپی از ofp_match بر می گرداند.
flip()	یک کپی از ofp_match بر می گرداند که مبدء و مقصد آن برعکس شده اند.
show()	بصورت رشته بر می گرداند.
get_nw_src()	ادرس IP مبدأ را بر می گرداند.
set_nw_src(IP and bits)	آدرس IP مبدأ را به همراه تعداد بیتی که باید تطبیق شود مقداردهی می کند.
get_nw_dst()	ادرس IP مقصد را بر می گرداند.
set_nw_dst(IP and bits)	آدرس IP مقصد را به همراه تعداد بیتی که باید تطبیق شود مقداردهی می کند.

تعریف تطبیق از روی بسته موجود

یک روش ساده برای ایجاد تطبیق دقیق بر اساس یک شیء بسته موجود (یعنی یک شیء اترنت از pox.lib.packet) یا از یک ofp_packet_in موجود وجود دارد. این کار با استفاده از متد ofp_match.from_packet() انجام می شود:

```
my_match = ofp_match.from_packet(packet, in_port)
```

پارامتر packet یک بسته parse شده یا ofp_packet_in است که در آن تطبیق ایجاد می شود. از آنجا که پورت ورودی در واقع در یک سربرگ بسته نیست، هنگام فراخوانی این روش با یک بسته، تطبیق حاصل به صورت پیش فرض پورت ورودی را wildcard می کند. البته می توان بعداً فیلد in_port را تنظیم کرد، اما به عنوان یک میانبر می توان آن را به سادگی به from_packet() پاس کرد، که در این صورت in_port به صورت پیش فرض گرفته می شود.

توجه داشته باشید که اگر می خواهید تطبیق جزئی داشته باشد، می توانید فیلدهای مربوط به نتیجه تطبیق را روی None تنظیم کنید (wildcard).

مثال: تطبیق ترافیک وب

```
import pox.openflow.libopenflow_01 as of # POX convention
import pox.lib.packet as pkt # POX convention
my_match = of.ofp_match(dl_type = pkt.ethernet.IP_TYPE, nw_proto = pkt.ipv4.TCP_PROTOCOL,
tp_dst = 80)
```

اقدامات (OpenFlow Actions)

اقدامات OpenFlow برای بسته هایی اعمال می شوند که با یک قانون نصب شده در مسیر داده تطبیق دارند. قطعه کدهای

موجود در اینجا را می توان در libopenflow_01.py در pox/openflow یافت.

ارسال به پورت – Output

بسته ها را از یک درگاه فیزیکی یا مجازی به جلو هدایت می کند. به درگاه های فیزیکی با مقدار Integral داده می شود، در حالی که درگاه های مجازی دارای اسامی نمادین هستند. درگاههای فیزیکی باید شماره پورتهای آنها کمتر از 0xFF00 باشد. تعریف ساختار:

```
class ofp_action_output (object):
  def __init__ (self, **kw):
    self.port = None # Purposely bad -- require specification
```

port (int) پورت خروجی این بسته است. مقدار آن می تواند یک شماره پورت واقعی یا یکی از پورت های مجازی زیر باشد:

OFPP_IN_PORT – به همان پورتی که بسته وارد شده ارسال شود.

OFPP_TABLE – اقدامی که در جدول جریان مشخص شده را اجرا کند.

OFPP_NORMAL – همانند سوئیچ عادی لایه ۲ یا ۳ عمل کند.

OFPP_FLOOD – به همه پورت های OpenFlow به جز پورت ورودی بسته و پورت هایی که به صورت OFPPC_NO_FLOOD تنظیم شده است، flood کند.

OFPP_ALL – به همه پورت های OpenFlow flood کند.

OFPP_CONTROLLER – به کنترلر بفرستد.

OFPP_LOCAL – به پورت OpenFlow محلی بفرستد.

OFPP_NONE – به هیچ جا نفرستد.

ارسال به صف – Enqueue

یک بسته را از طریق صف تعیین شده برای پیاده سازی رفتار مقدماتی QoS به جلو هدایت می کند:

```
class ofp_action_enqueue (object):
  def __init__ (self, **kw):
    self.port = 0
    self.queue_id = 0
```

port (int) باید یک پورت فیزیکی باشد.

queue_id (int) شناسه صف است.

توجه شود که تعریف صف ها بخشی از OpenFlow نیست و مختص سوئیچ است.

مقداردهی شناسه VLAN

اگر بسته دارای هدر VLAN نباشد، به آن اضافه می کند و شناسه آن را به مقدار مشخص شده و اولویت آن را به ۰ مقداردهی می کند. اگر بسته از قبل دارای هدر VLAN باشد، فقط شناسه آن را تغییر می دهد:

```
class ofp_action_vlan_vid (object):
  def __init__ (self, **kw):
    self.vlan_vid = 0
```

vlan_vid (int) شناسه VLAN است که باید کمتر از ۴۰۹۴ باشد.

مقداردهی اولویت VLAN

اگر بسته دارای هدر VLAN نباشد، به آن اضافه می کند و اولویت آن را به مقدار مشخص شده و شناسه آن را به ۰ مقداردهی می کند. اگر بسته از قبل دارای هدر VLAN باشد، فقط اولویت آن را تغییر می دهد:

```
class ofp_action_vlan_pcp (object):
  def __init__ (self, **kw):
    self.vlan_pcp = 0
```

vlan_pcp (short) اولویت VLAN است که باید کمتر از ۸ باشد.

مقداردهی آدرس مبدأ و مقصد اترنت

برای مقداردهی آدرس مبدأ یا مقصد (MAC اترنت) استفاده می شود:

```
class ofp_action_dl_addr (object):
  @classmethod
  def set_dst (cls, dl_addr = None):
    return cls(OFPAT_SET_DL_DST, dl_addr)
  @classmethod
  def set_src (cls, dl_addr = None):
    return cls(OFPAT_SET_DL_SRC, dl_addr)

  def __init__ (self, type = None, dl_addr = None):
    self.type = type
    self.dl_addr = EMPTY_ETH
```

type (int) یا باید OFPAT_SET_DL_SRC باشد یا OFPAT_SET_DL_DST.

dl_addr (EthAddr) آدرس MAC است که باید مقداردهی شود.

ممکن است استفاده از دو متد کلاس به جای ایجاد مستقیم اینستنسی از این کلاس مناسب باشد. به عنوان مثال، برای ایجاد Action برای بازنویسی آدرس MAC مقصد، می توان از روش زیر استفاده کرد:

```
action = ofp_action_dl_addr.set_dst(EthAddr("01:02:03:04:05:06"))
```

مقداردهی آدرس مبدأ و مقصد IP

برای مقداردهی آدرس IP مبدأ یا مقصد استفاده می شود:

```
class ofp_action_nw_addr (object):
  @classmethod
  def set_dst (cls, nw_addr = None):
    return cls(OFPAT_SET_NW_DST, nw_addr)
  @classmethod
  def set_src (cls, nw_addr = None):
    return cls(OFPAT_SET_NW_SRC, nw_addr)

  def __init__ (self, type = None, nw_addr = None):
    self.type = type
    if nw_addr is not None:
      self.nw_addr = IPAddr(nw_addr)
    else:
      self.nw_addr = IPAddr(0)
```

type (int) یا باید OFPAT_SET_NW_SRC باشد یا OFPAT_SET_NW_DST.

nw_addr (IPAddr) آدرس IP است که باید مقداردهی شود.

همانند آدرسهای MAC، به جای ساخت مستقیم اینستنسی از این کلاس، استفاده از متدهای set_src() و set_dst() راحت تر است:

```
action = ofp_action_nw_addr.set_dst(IPAddr("192.168.1.14"))
```

مقداردهی نوع سرویس IP

فیلد TOS بسته IP را مقداردهی می کند:

```
class ofp_action_nw_tos (object):
  def __init__ (self, nw_tos = 0):
    self.nw_tos = nw_tos
```

nw_tos (short) نوع سرویسی است که باید مقداردهی شود.

مقداردهی پورت مبدأ و مقصد TCP/UDP

پورت مبدأ و مقصد TCP/UDP را مقداردهی می کند:

```
class ofp_action_tp_port (object):
  @classmethod
```

```
  def set_dst (cls, tp_port = None):
    return cls(OFPAT_SET_TP_DST, tp_port)
  @classmethod
  def set_src (cls, tp_port = None):
    return cls(OFPAT_SET_TP_SRC, tp_port)

  def __init__ (self, type=None, tp_port = 0):
    self.type = type
    self.tp_port = tp_port
```

type (int) باید یا OFPAT_SET_TP_SRC باشد یا OFPAT_SET_TP_DST.

tp_port (short) مقدار پورت که باید کمتر از ۶۵۵۳۴ باشد.

همانند آدرسهای MAC و IP، به جای ساخت مستقیم اینستنسی از این کلاس، استفاده از متدهای set_src() و set_dst() راحت تر است.

مثال: ارسال تغییر مسیر جریان (FlowMod)

برای ارسال تغییر مسیر یک جریان باید یک ساختار تطبیق (قبلا اشاره شد) ایجاد شده و پارامترهای خاصی به شکل زیر مقداردهی شوند:

```
msg = ofp_flow_mod()
msg.match = match
msg.idle_timeout = idle_timeout
msg.hard_timeout = hard_timeout
msg.actions.append(of.ofp_action_output(port = port))
msg.buffer_id = <some buffer id, if any>
connection.send(msg)
```

با استفاده از متغیر connection که هنگام اتصال مسیر داده بدست می آید، می توان تغییر جریان را به سوئیچ ارسال کرد.

مثال: ارسال جریان به یک پورت

مشابه مثال قبل باید پارامترهای خاصی به شکل زیر مقداردهی شوند:

```
msg = of.ofp_packet_out(in_port=of.OFPP_NONE)
msg.actions.append(of.ofp_action_output(port = outport))
msg.buffer_id = <some buffer id, if any>
connection.send(msg)
```

ورودی in_port روی OFPP_NONE تنظیم شده است زیرا این بسته در کنترلر تولید شده است و از مسیر داده منشا نمی گیرد.

بخش سوم: مثال عملی ایجاد یک سوئیچ یادگیرنده

در این بخش به عنوان یک مثال عملی مراحل گام به گام ایجاد یک سوئیچ یادگیرنده در کنترلر POX و در محیط Mininet تشریح می شود. مرجع اصلی این بخش OpenFlow Tutorial در github به آدرس [4] می باشد.

برای شروع کد پایه برای کنترلر ارائه شده است. پس از آشنایی با آن، هاب (Hub) ارائه شده اصلاح می شود تا به عنوان یک سوئیچ یادگیرنده لایه ۲ عمل کند. در این برنامه، سوییچ هر بسته را بررسی می کند و نگاشت از پورت مبدأ را یاد می گیرد. پس از آن، آدرس MAC مبدأ با پورت مرتبط خواهد شد. اگر مقصد بسته از قبل به پورتی مرتبط شده باشد، بسته به آن پورت ارسال می شود، در غیر این صورت به همه پورت های سوئیچ flood می شود.

بعد از آن، این سوئیچ به یک سوئیچ مبتنی بر جریان تبدیل خواهد شد، به طوری که دیدن بسته ای با مبدأ و مقصد شناخته شده باعث می شود که ورودی جریان به پورت مناسب هدایت شود.

کنترلر پایه آموزش را اجرا کنید

در این مثال گام به گام از کنترل کننده مرجع استفاده نمی شود، و لازم است اگر هنوز در حال اجرا باشد، در پنجره اجرای برنامه کنترلر Ctrl-C را فشار دهید، یا از پنجره SSH دیگر آن را از بین ببرید:

```
$ sudo killall controller
```

همچنین باید sudo mn -c را اجرا کرده و Mininet را مجدداً راه اندازی کنید تا اطمینان حاصل شود همه چیز "تمیز" است. از کنسول Mininet:

```
mininet> exit
$ sudo mn -c
```

در نسخه های جدید Mininet از قبل POX نصب شده است. اگر در VM شما وجود ندارد، یا اگر نسخه بعدی POX می خواهید، آن را از مخزن POX موجود در github در VM خود دانلود کنید:

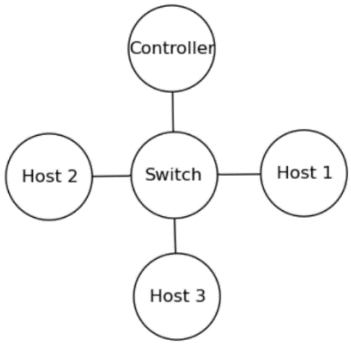
```
$ git clone http://github.com/noxrepo/pox
$ cd pox
```

اکنون می توانید مثال کد پایه آموزش را امتحان کنید:

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

این کد به POX می گوید که لاگ verbose را فعال کرده و مولفه of_tutorial را که از آن استفاده خواهید کرد (که در حال حاضر مانند یک هاب عمل می کند) شروع کند.

در کنسول دیگری توپولوژی به شکل زیر ایجاد می کنیم:



دستور زیر توپولوژی به شکل فوق ایجاد می کند:

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

ممکن است کمی زمان ببرد تا سوئیچ ها متصل شوند. هنگامی که یک سوئیچ OpenFlow اتصال خود را به کنترلر از دست بدهد، معمولا بازه زمانی تماس مجدد را تا سقف ۱۵ ثانیه افزایش می دهد. از آنجا که سوئیچ OpenFlow هنوز متصل نشده است، ممکن است این تاخیر بین ۰ تا ۱۵ ثانیه باشد. اگر مدت زمان انتظار طولانی است، می توان سوئیچ را طوری تنظیم کرد که بیش از N ثانیه با استفاده از پارامتر max-backoff-- صبر نکند. همچنین می توانید از Mininet خارج می شوید تا سوئیچ (ها) را حذف کرده، کنترلر را روشن کرده و سپس Mininet را شروع کنید تا بلافاصله متصل شود.

صبر کنید تا برنامه نشان دهد که سوئیچ OpenFlow متصل شده است. هنگامی که سوئیچ متصل می شود، POX چیزی مانند این را چاپ می کند:

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

خط اول از بخشی از POX است که اتصالات OpenFlow را مدیریت می کند. خط دوم از خود مولفه این آموزش است.

ارزیابی رفتار هاب با tcpdump

اکنون بررسی می کنیم که میزبان ها می توانند یکدیگر را پینگ کنند، و اینکه همه هاست ها دقیقاً ترافیک مشابهی را مشاهده می کنند – رفتاری که از یک سوئیچ انتظار می رود. برای انجام این کار، برای هر هاست xterm ایجاد کرده و ترافیک هر یک را مشاهده خواهیم کرد. در کنسول Mininet، سه xterm راه اندازی کنید:

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

توجه: دستور xterm اگر بخواهید آنرا مستقیماً از virtual box فراخوانی کنید کار نمی کند و خطا می دهد. در عوض از پنجره

ترمینال دیگری برای فراخوانی xterm استفاده کنید.

xterm ها را مرتب کنید تا همه آنها در کنار هم روی صفحه باشند. ممکن است نیاز به کاهش ابعاد پنجره ها برای صفحه نمایش کوچک لپ تاپ باشد.

در xterm برای h2 و h3، tcpdump را که ابزاری برای پرینت بسته های دیده شده توسط یک هاست است اجرا کنید:

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

و به همین ترتیب:

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

در xterm برای h1، یک پینگ ارسال کنید:

```
# ping -c1 10.0.0.2
```

بسته های پینگ اکنون به کنترلر می روند، که آن ها را به همه اینترفیس ها به غیر از فرستنده flood می کند. شما باید بسته های ARP و ICMP یکسان مربوط به پینگ را در هر دو xterms در حال اجرا tcpdump مشاهده کنید. هاب اینگونه کار می کند. همه بسته ها را به همه پورت های شبکه می فرستد.

حال، ببینید چه اتفاقی می افتد وقتی یک میزبان غیر موجود پاسخ ندهد. از h1 xterm:

```
# ping -c1 10.0.0.5
```

باید سه درخواست ARP بدون پاسخ را در tcpdump xterms مشاهده کنید. بعد از این که کد تمام شد، سه درخواست پاسخ داده نشده ARP نشانه این است که شما ممکن است به طور تصادفی بسته ها را دور می اندازید.

اکنون می توانید xterms را ببندید.

ارزیابی کنترلر با iperf

در اینجا، هاب of_tutorial را محک خواهید زد.

ابتدا قابلیت دستیرسی را بررسی کنید. Mininet باید همراه با هاب POX در پنجره دیگری در حال اجرا باشد. در کنسول Mininet، اجرا کنید:

```
mininet> pingall
```

این فقط یک بررسی عقلانی برای اتصال است. اکنون، در کنسول Mininet، اجرا کنید:

```
mininet> iperf
```

اکنون، شماره خود را با کنترلر مرجعی که قبلاً مشاهده کرده اید مقایسه کنید. چگونه مقایسه می شود؟

راهنمایی: هر بسته اکنون به کنترلر می رود.

کنترلر پایه آموزش را تغییر دهید

به ترمینال SSH خود بروید و با استفاده از Ctrl-C کنترلر آموزشی را متوقف کنید. فایلی که تغییر می دهید pox/misc/of_tutorial.py است. این فایل را در ویرایشگر مورد علاقه خود باز کنید. vim گزینه خوبی است، از قبل با ترمینال دانلود شده است. برای استفاده از vim، مطمئن شوید که در دایرکتوری صحیح قرار دارید (pox/pox/misc) و وارد کنید:

```
$ vi of_tutorial.py
```

کد فعلی act_like_hub() را از handler فراخوانی می کند تا برای پیام های packet_in رفتار هاب را پیاده سازی کند. می خواهیم به جای آن از تابع act_like_switch() استفاده کنیم که طراحی اولیه از شکل کد نهایی سوئیچ یادگیرنده شما است.

هر بار که این فایل را تغییر داده و ذخیره می کنید، اطمینان حاصل کنید که POX مجدداً راه اندازی شود، سپس از پینگ ها برای بررسی رفتار ترکیبی سوئیچ و کنترل کننده به عنوان (۱) هاب، (۲) سوئیچ اترنت یادگیرنده مبتنی بر کنترلر و (۳) سوئیچ یادگیرنده شتاب یافته با جریان استفاده کنید. برای (۲) و (۳)، هاست هایی که مقصد پینگ نیستند، پس از پخش درخواست اولیه ARP، هیچ ترافیکی از tcpdump نباید نمایش دهند.

برای تست کد: اطمینان حاصل کنید که mininet در حال اجرا است و سپس در پنجره ترمینال دیگری دستور زیر را اجرا کنید:

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

پس از اتصال، چند پینگ را امتحان کنید تا ببینید سوئیچ کار می کند یا نه. پهنای باند برگشتی توسط iperf از یک سوئیچ (Gbits) باید بسیار سریعتر از یک هاب (Mbits) باشد.

تجزیه (parse) بسته ها با کتابخانه های بسته POX

از کتابخانه بسته POX برای تجزیه بسته ها و در دسترس قرار دادن هر قسمت پروتکل در اختیار پایتون استفاده می شود. از این کتابخانه می توان برای ساخت بسته هایی برای ارسال نیز استفاده کرد.

کتابخانه های تجزیه در مسیر زیر قرار دارد:

```
pox/lib/packet/
```

هر پروتکل دارای یک فایل تجزیه مربوطه است.

برای اولین تمرین، فقط باید به فیلدهای مبدأ و مقصد اترنت دسترسی داشته باشید. برای استخراج مبدأ یک بسته، از علامت نقطه استفاده کنید:

```
packet.src
```

فیلدهای مبدأ و مقصد اترنت به عنوان اشیا pox.lib.address.EthAddr ذخیره می شوند. این فیلدها به سادگی رشته تبدیل می شوند (str (addr) چیزی مانند "01:ea:be:02:05:01" را برمی گرداند) و به راحتی از روی رشته ایجاد می شود

.(EthAddr("01:ea:be:02:05:01"))

برای دیدن همه اعضای یک بسته تجزیه شده:

```
print dir(packet)
```

آنچه برای بسته ARP مشاهده خواهید کرد:

```
['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', 'hwsrc', 'hwtype', 'msg', 'next', 'opcode',
  'pack', 'parse', 'parsed', 'payload', 'pre_hdr', 'prev', 'protodst', 'protolen',
  'protosrc', 'prototype', 'raw', 'set_payload', 'unpack', 'warn']
```

بسیاری از فیلدهای بالا در همه اشیای پایتون مشترک است و می توان آنها را نادیده گرفت.

of_tutorial.py

داخل فایل کنترلر پایه آموزش، کلاس اصلی Tutorial وجود دارد. به ازای هر سوئیچ که به این کنترلر وصل شود یک شیء از کلاس مذکور ساخته می شود:

```
class Tutorial (object):
  def __init__ (self, connection):
  ...
  def resend_packet (self, packet_in, out_port):
  ...
  def act_like_hub (self, packet, packet_in):
  ...
  def act_like_switch (self, packet, packet_in):
  ...
  def _handle_PacketIn (self, event):
  ...
```

در مقداردهی اولیه (متد _init_)، متغیرهای اینتنس کلاس مقداردهی می شوند و یک دیکشنری خالی (که بعداً از کلیدهای آدرس MAC برای مقادیر تعداد پورت تشکیل می شود) ایجاد می شود:

```
  def __init__ (self, connection):
    self.connection = connection
    connection.addListeners(self)
    self.mac_to_port = {}
```

اولین متدی که می بینید، resend_packet است که به سوئیچ آموزش می دهد که بسته ای که قبلاً به کنترلر ارسال کرده است را به چه پورتی بفرستد:

```
  def resend_packet (self, packet_in, out_port):
    msg = of.ofp_packet_out()
    msg.data = packet_in
    action = of.ofp_action_output(port = out_port)
    msg.actions.append(action)
    self.connection.send(msg)
```

متد بعدی act_like_hub، رفتار هاب یا ارسال بسته های دریافتی به همه پورت ها را با استفاده از resend_packet توصیف می کند:

```
  def act_like_hub (self, packet, packet_in):
    self.resend_packet(packet_in, of.OFPP_ALL)
```

باید بدنه متد act_like_switch را بنویسید تا رفتار سوئیچ یادگیرنده را پیاده سازی کنید. اولین کاری که باید انجام دهید این است که پورت مبدأ MAC را یاد بگیرید، که در اصل به معنای پر کردن دیکشنری با آدرس مبدأ MAC بسته ورودی و پورت ورودی آن است. (اگر آدرس مبدأ قبلا در دیکشنری وجود داشته باشد به روز می شود و اگر وجود نداشته باشد اضافه می گردد.)

سپس باید یک شیء ofp_match ایجاد کنید و اگر پورت مقصد بسته (که اخیراً پیدا کردید) خالی نیست، بسته را با استفاده از resend_packet به آن پورت ارسال کنید و یک تغییر مسیر جریان ایجاد کنید. در غیر این صورت، بسته را به همه پورت ها (رفتار "Flood") با استفاده از پورت "of.OFPP_ALL" ارسال کنید.

ایجاد یک ofp_flow_mod ساده است. از روشهایی که در بالا توضیح داده شده استفاده کنید و قسمتهای مناسب را مقداردهی کنید. فیلدهایی که باید مقداردهی کنید عبارتند از match (با استفاده از شیء ایجاد شده ofp_match)، idle-timeout و hard-timeout (اختیاری است)، و ofp_action_output جدیداً ایجاد شده را به فیلد actions اضافه کنید. در آخر، پیام تغییر مسیر جریان را به سوئیچ ارسال کنید.

بدنه متد مذکور به شکل زیر خواهد بود:

```
  def act_like_switch (self, packet, packet_in):
    self.mac_to_port[packet.src] = packet_in.in_port
    if packet.dst in self.mac_to_port:
      out_port = self.mac_to_port[packet.dst]
      msg = of.ofp_flow_mod()
      msg.match = of.ofp_match.from_packet(packet)
      msg.idle_timeout = 30
      msg.buffer_id = packet_in.buffer_id
      action = of.ofp_action_output(port = out_port)
      msg.actions.append(action)
```

```
      self.connection.send(msg)
    else:
      self.resend_packet(packet_in, of.OFPP_ALL)
```

تست کنترلر

برای آزمایش سوئیچ اترنت مبتنی بر کنترلر، ابتدا بررسی کنید که وقتی همه بسته ها به کنترل کننده می رسند، فقط بسته های برودکست (مانند ARP) و بسته هایی با مکان نامعلوم مقصد (مانند اولین بسته ارسال شده برای جریان)، به همه پورت های غیر از پورت ورودی ارسال می شوند. می توانید این کار را با اجرای tcpdump روی xterm برای هر هاست انجام دهید.

هنگامی که سوئیچ دیگر رفتار هاب را نداشت، تلاش کنید تا هنگامی که پورت مبدا و مقصد مشخص هستند، جریان بدون نیاز به ارسال به کنترلر هدایت شود. برای تأیید شمارنده های جریان می توانید از ovs-ofctl استفاده کنید و اگر پینگ های بعدی خیلی سریعتر کامل می شوند، می دانید که بسته ها از طریق کنترلر عبور نمی کنند. همچنین می توانید با اجرای iperf در Mininet و بررسی عدم ارسال پیام بسته OpenFlow، این رفتار را تأیید کنید. پهنای باند iperf گزارش شده نیز باید بسیار بیشتر باشد و باید با عددی که قبلا هنگام استفاده از کنترلر مرجع دریافت کرده اید مطابقت داشته باشد.

در اینجا تست به پایان رسیده و کنترلر کار می کند.

مراجع

[1] F. H. Fitzek, F. Granelli and P. Seeling, Computing in Communication Networks, From Theory to Practice, Elsevier Inc., 2021.

[2] N. McKeown and et.al., "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review,* vol. 38, no. 2, 2008.

[3] [Online]. Available: https://noxrepo.github.io/pox-doc/html/#openflow-in-pox.

[4] [Online]. Available: https://github.com/mininet/openflow-tutorial/wiki/Create-a-Learning-Switch.