

به نام خدا



دانشگاه صنعتی امیرکبیر

**Amirkabir University
of Technology**

پروژه شبکه‌های کامپیوتری پیشرفته

عنوان پروژه: برنامه‌نویسی شبکه‌های نرم‌افزار محور در مینی‌نت (SDN)

استاد درس: دکتر خرسندی

دانشجو: امیررضا زارع

شماره دانشجویی: ۴۰۱۱۳۱۰۰۸

اسفند ۱۴۰۱

فهرست مطالب

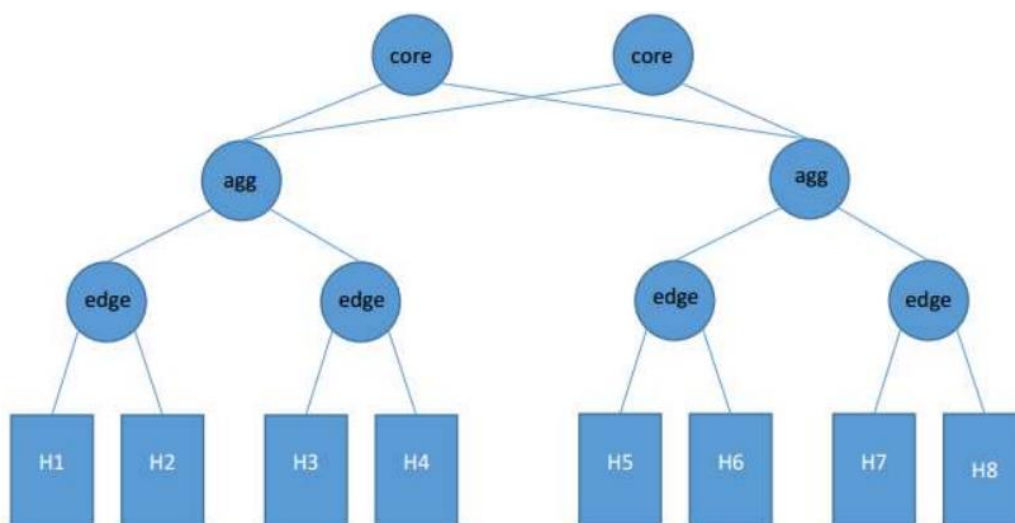
۳	طرح مسئله
۳	بخش اول:
۶	بخش دوم (اختیاری):
۷	مقدمه
۹	نصب mininet
۱۸	Run a Regression Test
۱۸	Changing Topology Size and Type
۱۹	Link variations
۱۹	Adjustable Verbosity
۲۰	Custom Topologies
۲۱	ID = MAC
۲۲	XTerm Display
۲۴	سایر انواع سوئیچ
۲۵	Mininet معیار
۲۵	Everything in its own Namespace (user switch only)
۲۵	Mininet (CLI) دستورالعمل
۲۵	Display Options
۲۶	Python Interpreter
۲۶	Link Up/Down
۲۷	XTerm Display
۲۷	پایتون API
۲۷	SSH daemon per host
۲۸	Mininet برای تسلط بیشتر بر
۲۸	Using a Remote Controller
۲۹	Ryu

طرح مسئله

در بخش اول این تمرین از شما خواسته شده ابتدا توپولوژی FatTree که در مراکز داده مورد استفاده قرار می‌گیرد را با مینیت پیاده‌سازی و اجرا کنید. در ادامه شاهد خواهید بود که به دلیل وجود حلقه در این شبکه، تست pingall با شکست مواجه خواهد شد. برای رفع این مشکل باید از ماژول‌های آماده در کنترلر POX با نام learning_l2 و tree_spanning استفاده کنید که در ادامه نتیجه‌ی تست pingall موفقیت‌آمیز خواهد بود. پس از رفع این مشکل از شما خواسته شده تا خودتان ماژولی برای کنترل دسترسی در این کنترلر بنویسید که توضیحات آن در ادامه ارائه شده است. در بخش دوم تمرین که اختیاری نیز هست، از شما خواسته شده تا این بار از زبان برنامه‌نویسی سطح بالای Pyretic استفاده کرده و یک دیواره‌ی آتش برای شبکه بنویسید. لیست ماشین‌هایی که نمی‌توانند با یکدیگر در شبکه ارتباط داشته باشند به شما داده شده است و باید ارتباطات آن‌ها را با یکدیگر مسدود کنید.

بخش اول:

هدف از این تمرین آشنایی با ماژول‌های کنترلر پاکس و شبیه‌سازی یک محیط تصدیق است. یکی از مزیت‌های محیط نرم افزار محور اجرای برنامه‌های کاربری مختلف بر روی کنترلر شبکه است. به این صورت این برنامه‌های کاربردی به صورت مستقل از هم می‌توانند در کنترلر شبکه همکاری کنند. در این تمرین شما با ماژول‌های ساخت درخت پوشا، مسیریابی الیه ۲ آشنا خواهید شد و از آن‌ها در کنار یک ماژول کنترل دسترسی که خودتان باید ایجاد کنید استفاده می‌کنید. در اولین گام این تمرین شما باید توپولوژی دارای حلقه به شکل زیر را با مینیت ایجاد کنید.



در این توپولوژی، امکان استفاده از کنترلر پیش فرض وجود ندارد چرا که در آن برای برخورد با حلقه راهکاری اندیشیده نشده است. سعی کنید در این توپولوژی عمل ping را انجام دهید و مشاهدات خود را ثبت کنید. در مرحله ی بعد شما باید کنترلر شبکه را به کنترلر remote تغییر دهید. سپس در هنگام راه اندازی کنترلر POX ماژول های درخت پوشا و مسیریابی الیه ۲ را آغاز کنید. در این مرحله شبکه شما باید کارکرد معمولی داشته باشد، یعنی بتوانید عمل ping را انجام دهید. لازم به ذکر است که ماژول tree_spanning برای اینکه عملکرد درستی داشته باشد به ماژول دیگری در کنترلر پاکس با نام discovery نیاز دارد. کد زیر نحوه انتخاب کنترلر remote برای شبکه را نشان میدهد:

```
topo = DCTreeTopo(k)

net = Mininet(topo, link=TCLink, build=False, autoSetMacs = True)

net.addController(name='c0', controller=RemoteController, ip='127.0.0.1', port=6633)

net.start()
```

کد زیر نیز نحوه راه اندازی پاکس به همراه ماژول های مذکور را نشان میدهد:

```
~/pox/pox.py openflow.spanning_tree openflow.discovery forwarding.12_learning
```

پس از راه اندازی شبکه ی دارای حلقه، حال باید ماژول کنترلر دسترسی خود را ایجاد کنید. به این منظور فرض بر این است که ماشین شماره یک وظیفه ی تصدیق و شناسایی ماشین های داخل شبکه را برعهده دارد، به این ترتیب که هر ماشین باید پیش از اینکه پیامی در شبکه ارسال نماید، یک پیام به ماشین یک ارسال و جوابی دریافت کند و فقط پس از دریافت پیام از ماشین یک می تواند با سایر ماشین های داخل شبکه ارتباط برقرار کند. به این منظور شما باید لیستی از ماشین های تصدیق شده را در ماژول خود نگهداری کنید و هر زمان که ماشینی پیغامی از ماشین یک دریافت کرد، به لیست مورد نظر اضافه شود. آنگاه دو ماشین به شرطی می توانند مکاتبه داشته باشند که هر دو تصدیق شده باشند. در طول پیاده سازی کنترلر دسترسی، آدرس MAC ماشین ها را برابر آدرس IP آن ها قرار دهید. به این صورت ماشین با آدرس MAC زیر همان تصدیق کننده است:

```
00:00:00:00:00:01
```

شما باید در ماژول خود رخداد PacketIn را با پیاده سازی تابع __PacketIn_handle__ دریافت کنید و بر اساس منطق زیر عمل کنید:

۱- اگر آدرس MAC فرستنده همان تصدیق کننده است، آدرس MAC گیرنده را به لیست مورد نظر اضافه کنید و هیچکار دیگری انجام ندهید.

۲- اگر آدرس MAC فرستنده و گیرنده در لیست موجود است و یا گیرنده تصدیق کننده است، هیچ قانونی را ثبت نمی کنید، چراکه مسیر یاب الیه ۲، بسته را به مقصد خواهد رساند.

۳- اگر آدرس فرستنده و یا گیرنده در لیست وجود ندارد، قانونی برای حذف کردن بسته ی مورد نظر وضع کنید. مقادیر timeout_idle و timeout_hard را به ترتیب برابر ۱۴ و ۹۴ ثانیه قرار دهید. دقت داشته باشید که اولویت قانونی که وضع می کنید باید از اولویت قانون های مسیر یاب الیه ۲ بیشتر باشد که اجرای همزمان این دو ماژول نتیجه ی مورد نظر، یعنی رعایت کنترل دسترسی را به دنبال داشته باشد.

برای پیاده سازی این ماژول، کار خود را با آشنا شدن با فایل learning_l2 در مسیر forwarding/pox/pox~/ شروع کنید. می توانید منطق مورد نظر را در تابع __ PacketIn_handle آن پیاده سازی کنید. اگر نام ماژول کنترل دسترسی را py.AccessCtrl و آن را در پوشه forwarding/pox/pox~/ قرار دهید، حال می توانید پاکس را به همراه ماژول خود به شکل زیر راه اندازی کنید..

```
~/pox/pox.py openflow.spanning_tree openflow.discovery forwarding.l2_learning forwarding.AccessCtrl
```

گزارش باید حاوی مطالب زیر باشد:

۱- مشاهدات خود در بخش انجام عمل ping در شبکه دارای حلقه و کنترلر پیش فرض را بنویسید

۲- ابتدا شبکه را راه اندازی کنید و فقط دو ماژول درخت پوشا و مسیر یاب الیه ۲ را راه اندازی کنید. با اجرای عمل pingall اتصال تمام ماشین ها به یکدیگر را نشان دهید. از این آزمایش اسکرین شات تهیه کنید و در گزارش خود قرار دهید.

۳- توضیح دهید که حفظ اولویت قانون های کنترل دسترسی نسبت به مسیر یاب چگونه پیاده سازی شده است.

۴- کنترلر را متوقف کنید و این بار آن را با ماژول های discovery، درخت پوشا، مسیر یاب و کنترل دسترسی خودتان اجرا کنید. اکنون عمل ping را برای ماشین های ۲ و ۳ انجام دهید. این دو ماشین در این مرحله نباید بتوانند یکدیگر را ببینند. سپس این عمل را برای ماشین های ۲ و ۱ انجام دهید که این عمل باید اجرا شود. اکنون دوباره ping بین ۲ و ۳ را تکرار کنید که در این مرحله باز هم نباید بتوانند یکدیگر را ببینند. سپس ping بین ۳ و ۱ و در نهایت بین ۲ و ۳ را اجرا کنید. از مراحل این آزمایش اسکرین شات تهیه کنید. شما همچنین باید کد

بخش توپولوژی شبکه را در قالب فایلی به نام `py.LoopTopo` و کنترل دسترسی را در فایلی به نام `py.AccessCtrl` به همراه گزارش ارسال کنید. تمام خروجی‌ها را در فایل فشرده‌ای قرار دهید و اسم آن را همان شماره دانشجویی خود قرار دهید. تمپلیت برای ایجاد برنامه‌ی توپولوژی شبکه و ماژول کنترل دسترسی نیز به همراه گزارش تمرین در اختیار شما قرار گرفته است.

بخش دوم (اختیاری):

هدف از این تمرین آشنایی با زبان `Pyretic` است. در تمرین قبلی با `API` هایی که کنترلر `POX` برای کنترل شبکه در اختیار شما قرار می‌دهد آشنا شدید. از اشکالات روش پیشین ورود به جزئیات بیش از اندازه برای پیاده‌سازی ماژول‌های مورد نظر و همچنین کم بودن قابلیت حمل ماژول‌های ایجاد شده است. در زبان `Pyretic` به راحتی میتوان دستورات سطح بالا و بدون نیاز به اطلاع از جزئیات پیاده‌سازی را استفاده کرد. در زبان `Pyretic` برای کنترل شبکه هر ماژول تابعی به نام `main` دارد که باید یک `Policy` را بازگرداند. `Policy` تکه کد یا ماژول یا ترکیبی از چند ماژول دیگر است که مشخص می‌کند شبکه در مقابل دریافت یک بسته چگونه باید رفتار کند. یک `Policy` یک بسته را می‌گیرد و میتواند صفر یا بیشتر بسته به عنوان خروجی ایجاد کند. مثلاً قطعه کد زیر یک `hub` را پیاده‌سازی میکند:

```
from pyretic.lib.corelib import*
```

```
def main():
```

```
    return flood()
```

در این قسمت شما ماژولی ایجاد خواهید کرد که باید بر روی هر توپولوژی به درستی کار کند. ماژول شما در ابتدای شروع به کار خود باید فایلی به نام `csv.hosts_blocked` را بخواند. فرمت این فایل به شکل زیر است:

```
<line num>, <host 1 mac>, <host 2 mac>
```

جفت آدرس‌های `MAC` که در هر خط این فایل قرار دارند اجازه ارتباط با همدیگر را ندارند. شما باید در تابع `main` یک `policy` ایجاد کنید که در آن مشخص شود چه هاست‌هایی نمی‌توانند با همدیگر ارتباط داشته باشند (از `match` و عملگر `|` استفاده کنید) سپس با استفاده از امکانات زبان `Pyretic` این `policy` را بر عکس کنید و به همراه ماژول `learner_mac` بازگردانید. به طور مثال اگر `Policy` نهایی شما `hosts_allowed` نام داشته‌باشد، در انتهای تابع `main` عبارت زیر باید برگردانده شود:

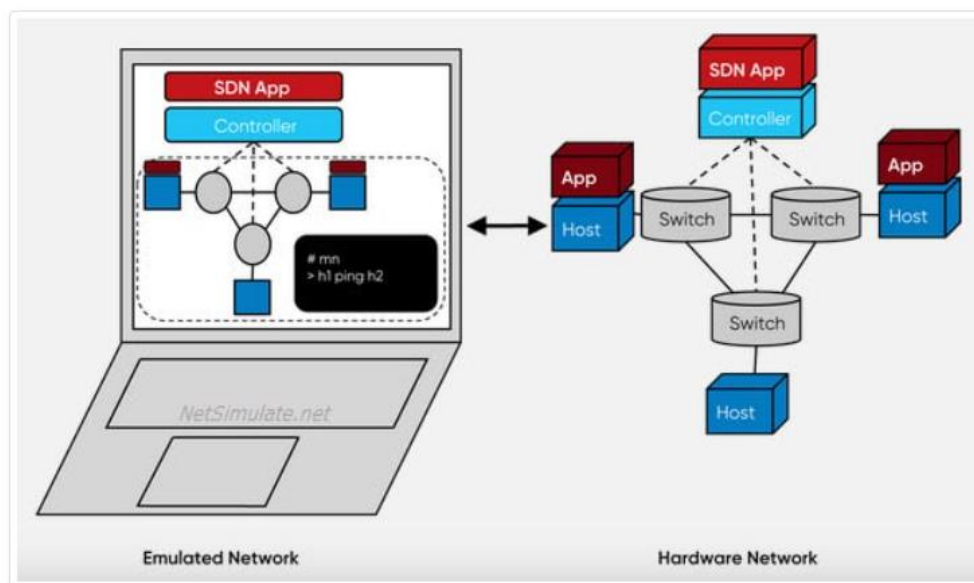
```
allowed_host >> mac_learner
```

این ماژول را پس از نوشتن، در پوشه‌ی پایرتیک و در پوشه‌ی ماژول‌ها ذخیره کنید. در ادامه باید یک توپولوژی ساده و دلخواه ایجاد کنید. پیشنهاد ما ایجاد یک توپولوژی تک سوئیچی با دستور زیر است:

```
sudo mn -topo single,5 -controller remote -mac
```

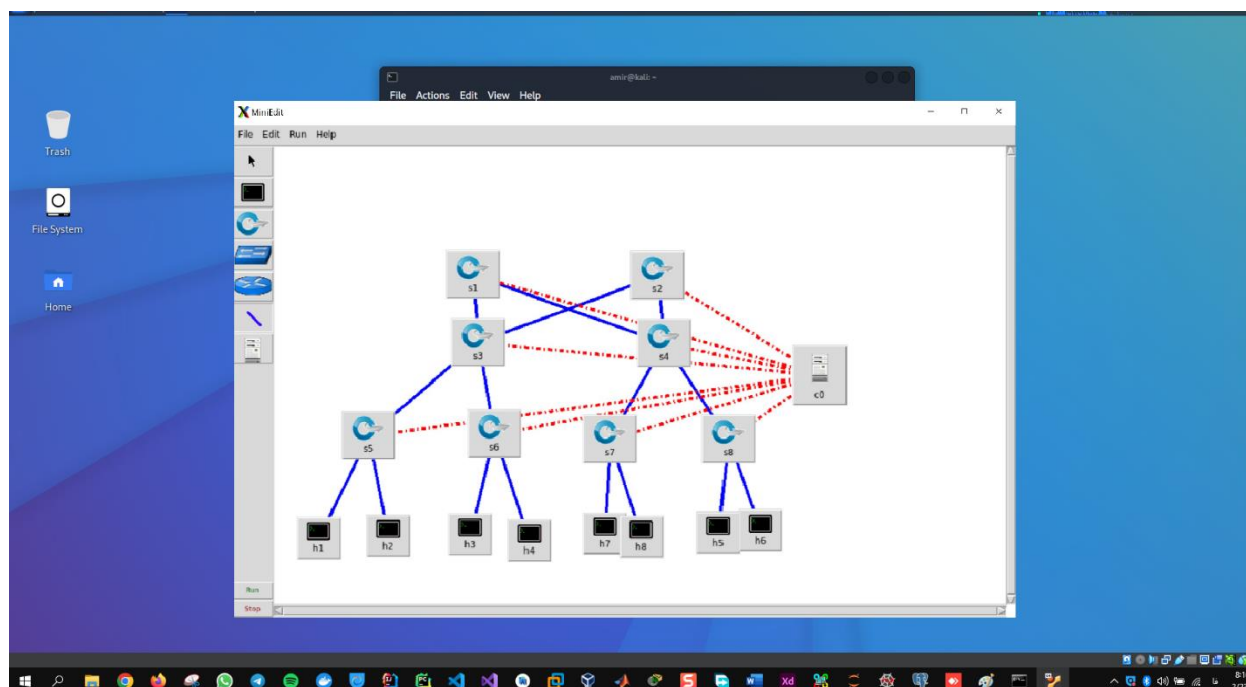
به عنوان مثال در فایل بالکشده‌ها در نظر بگیرید که هاست‌های ۲ و ۴ و همچنین ۳ و ۵ مجوز ارتباط ندارند. عمل pingall را انجام داده و در یک اسکرین‌شات گزارش کنید که این دو جفت هاست نتوانسته‌اند یکدیگر را ping کنند. تمپلیت برای این ماژول نیز به همراه گزارش تمرین، در اختیار شما قرار گرفته است.

مقدمه



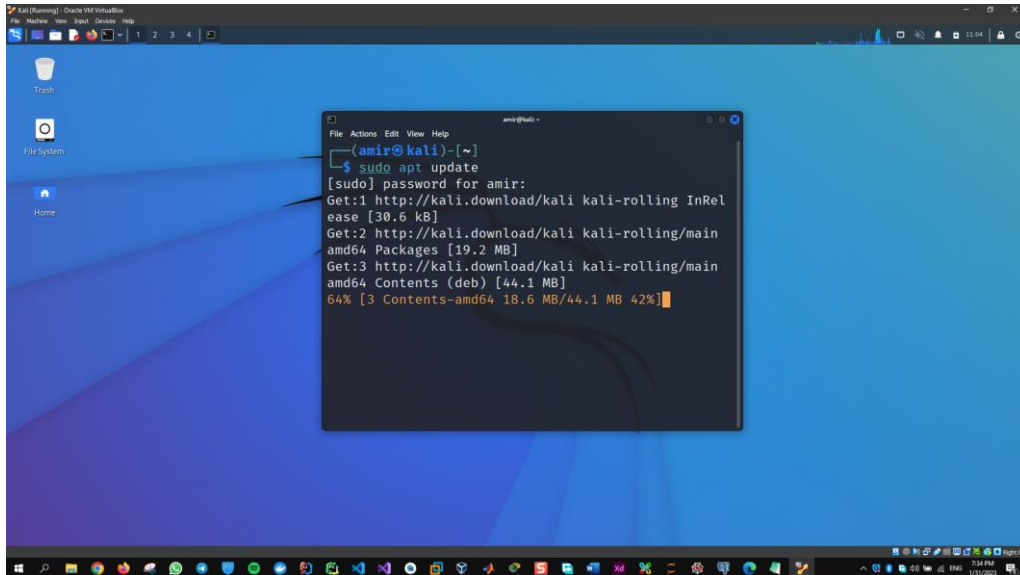
شبکه‌های نرم افزار محور (Software Defined Networking – SDN) یک معماری جدید در شبکه‌های کامپیوتری است که امکان مدیریت شبکه را در سطح بالاتری فراهم می‌کند. این امر از طریق جداسازی لایه تصمیم‌گیرنده در مورد نحوه هدایت ترافیک (لایه کنترل) از لایه زیرین که وظیفه هدایت بسته‌ها به مقصد انتخابی را دارد (لایه داده) انجام می‌شود. شبکه‌های نرم افزار محور یا شبکه‌های مبتنی بر نرم افزار (SDN)، شامل تجهیزات و زیرساختی است که اجرای آن در مقیاس تجاری، معمولاً با هزینه‌های زمانی و مالی چشمگیری همراه

است. بنابراین لازم است قبل از اجرای واقعی شبکه ، یک مدل سازی و تحلیل قبلی در مورد شبکه مورد نظر صورت گیرد و مشکلات احتمالی شبکه، شناسایی و برطرف گردد. برای این کار نیاز به ابزاری داریم که تجهیزات و ارتباطات شبکه را برای ما مدل سازی و شبیه سازی کند.

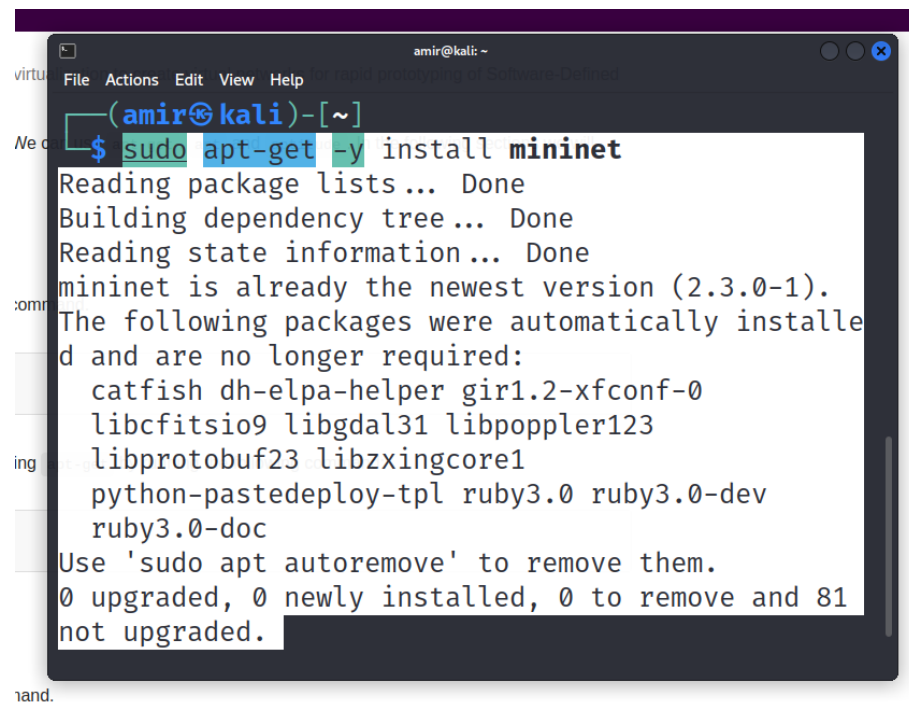


نرم افزار مینی نت (Mininet) یک شبیه ساز شبکه به صورت اپن سورس است که برای پشتیبانی از پروژه های تحقیقاتی و آموزشی در زمینه شبکه های مبتنی بر نرم افزار - شبکه های نرم افزار محور (SDN) ، توسط گروهی از اساتید دانشگاه استنفورد طراحی شده است. نرم افزار Mininet یک شبیه ساز شبکه متن باز و مبتنی بر لینوکس است که به خاطر پشتیبانی توکار از سوئیچ های OpenFlow به طور گسترده مورد استفاده قرار می گیرد. نرم افزار شبیه ساز Mininet ، شبکه ای با میزبان ها، سوئیچ ها، کنترلرها و لینک های مجازی را فراهم می کند. میزبان های مینی نت، نرم افزارهای شبکه ای استاندارد لینوکس را اجرا کرده و سوئیچ های آن از پروتکل OpenFlow برای انعطاف پذیری بالا در مسیریابی های سنتی و شبکه SDN پشتیبانی می کند. توابع موجود در شبیه ساز مینی نت ، انواع مختلف کنترل کننده ها و سوئیچ ها را ساپورت می کند. همینطور در Mininet می توان سناریوهای پیچیده سفارشی را به کمک API Mininet Python ایجاد نمود.


```
sudo apt-get update
```

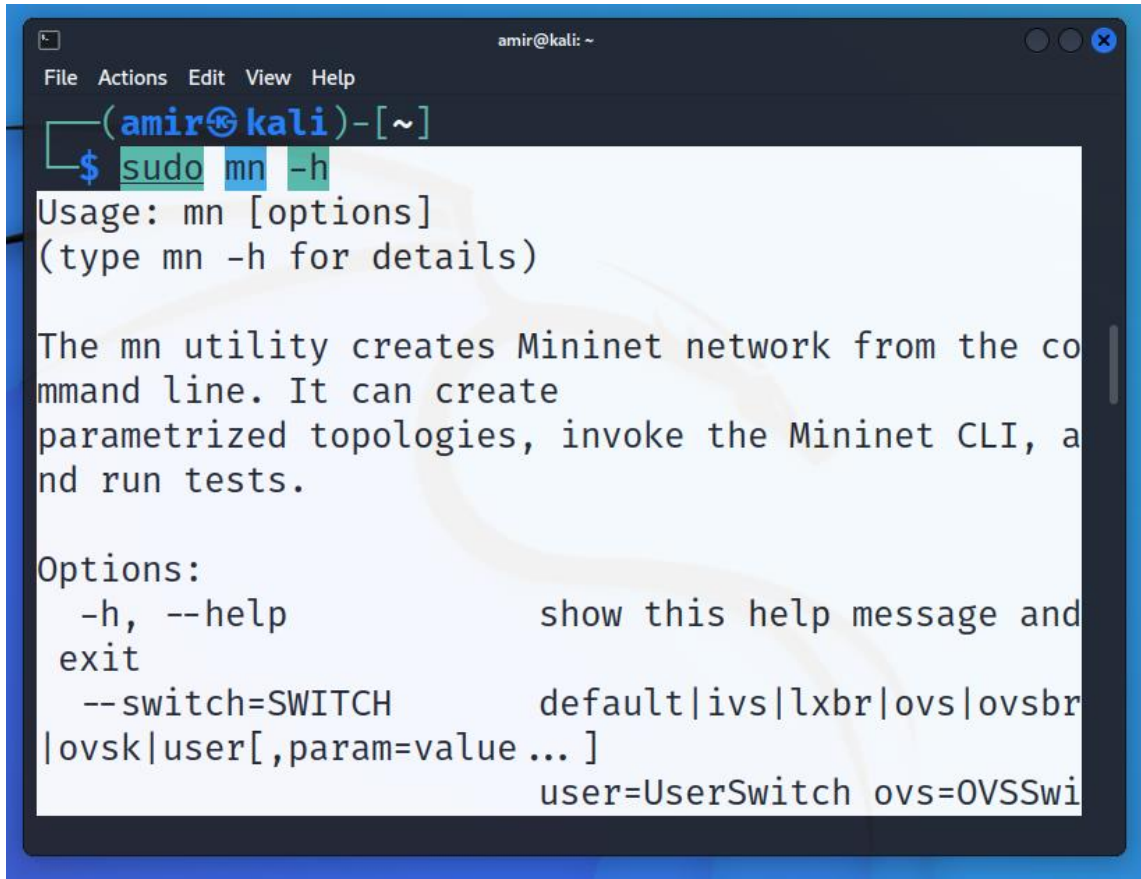


```
sudo apt-get -y install mininet
```



برای نمایش یک پیام راهنما که گزینه‌های راه‌اندازی Mininet را توصیف می‌کند، دستور زیر را تایپ کنید:

```
$ sudo mn -h
```

A screenshot of a terminal window titled 'amir@kali: ~'. The window shows the command '\$ sudo mn -h' being executed. The output displays the usage of the 'mn' utility, which creates a Mininet network from the command line. It lists options: '-h, --help' for showing the help message, and '--switch=SWITCH' for specifying the switch type (default, ivs, lxbr, ovs, ovsbr, ovsk, or user-defined). The user is currently in the 'amir@kali' environment.

```
File Actions Edit View Help
(amir@kali)-[~]
$ sudo mn -h
Usage: mn [options]
(type mn -h for details)

The mn utility creates Mininet network from the command line. It can create
parametrized topologies, invoke the Mininet CLI, and run tests.

Options:
  -h, --help                show this help message and exit
  --switch=SWITCH            default|ivs|lxbr|ovs|ovsbr|ovsk|user[,param=value ... ]
                             user=UserSwitch ovs=OVSSwitch
```

برای مشاهده ترافیک کنترل با استفاده از تفکیک کننده OpenFlow Wireshark، ابتدا wireshark را در پس زمینه باز کنید:

```
$ sudo wireshark &
```

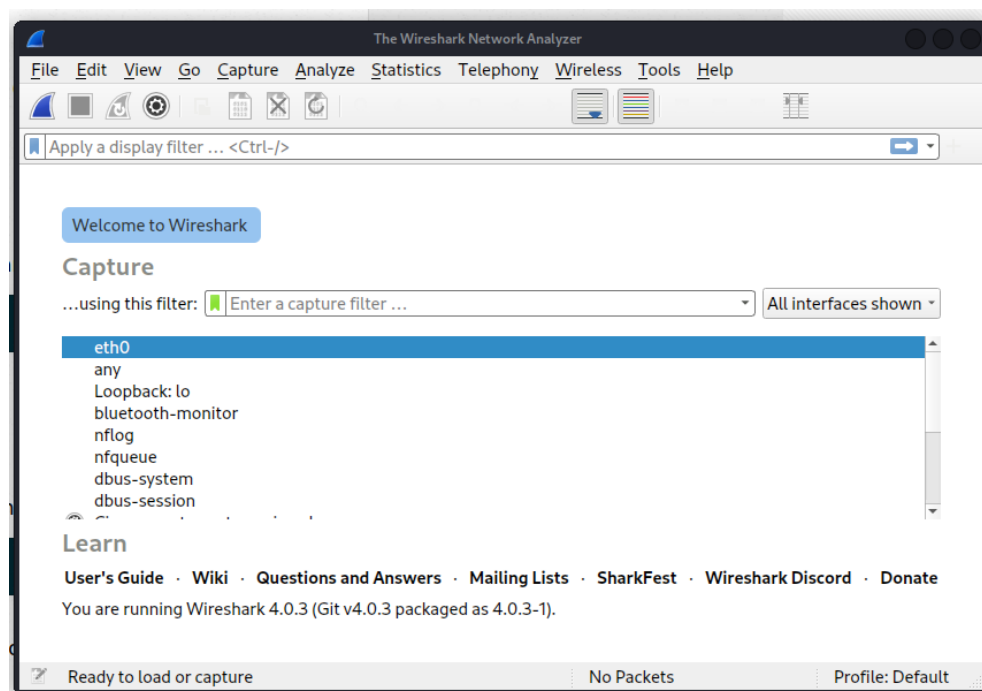
```
• mininet> precedes Mininet commands that should be typed at Mininet's CLI,
• a precedes Linux commands that are typed at a root shell prompt

File Actions Edit View Help
--cluster=server1,server2 ...
run on multiple servers (experimental!)
--placement=block|random
node placement for --cluster (experimental!)

(amir@kali)-[~]
$ sudo wireshark &
[1] 11542

(amir@kali)-[~]
$ ** (wireshark:11545) 02:02:37.319010 [GUI WARNING] -- QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
```

محیط وایرشارک:

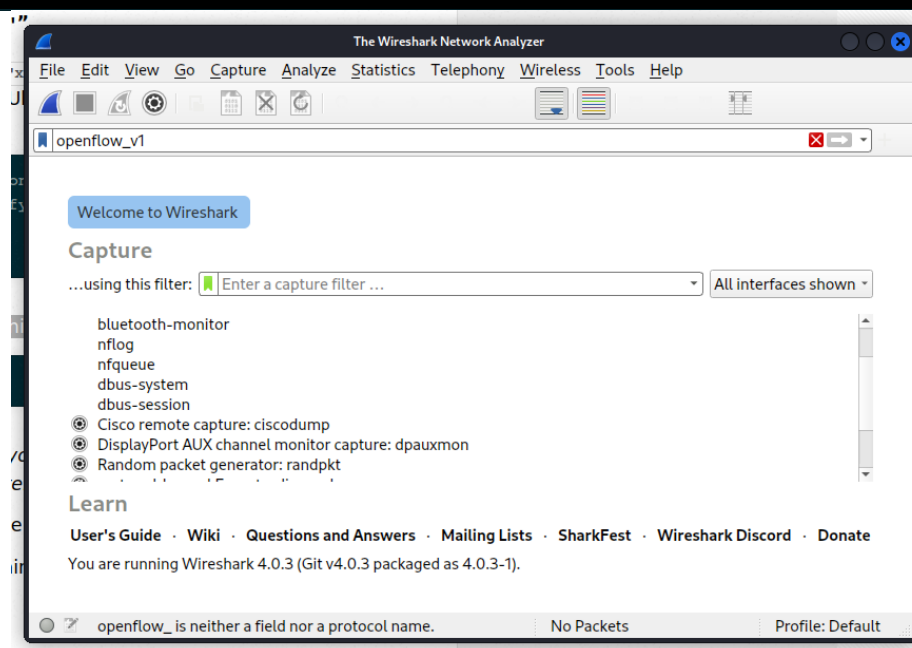


نصب POX:

```
amir@kali: ~  
File Actions Edit View Help  
ovs-vsctl exited with code 1  
*** Error connecting to ovs-db with ovs-vsctl  
Make sure that Open vSwitch is installed, that ovsdb-server is  
running, and that  
"ovs-vsctl show" works correctly.  
You may wish to try "service openvswitch-switch start".  
  
(amir@kali)-[~]  
$ service openvswitch-switch start  
Failed to start openvswitch-switch.service: Unit openvswitch-swit  
ch.service not found.  
  
(amir@kali)-[~]  
$ git clone http://github.com/noxrepo/pox  
Cloning into 'pox' ...  
warning: redirecting to https://github.com/noxrepo/pox/  
remote: Enumerating objects: 13058, done.  
remote: Counting objects: 100% (283/283), done.  
remote: Compressing objects: 100% (125/125), done.  
Receiving objects: 51% (6660/13058), 3.38 MiB | 230.00 KiB/s
```

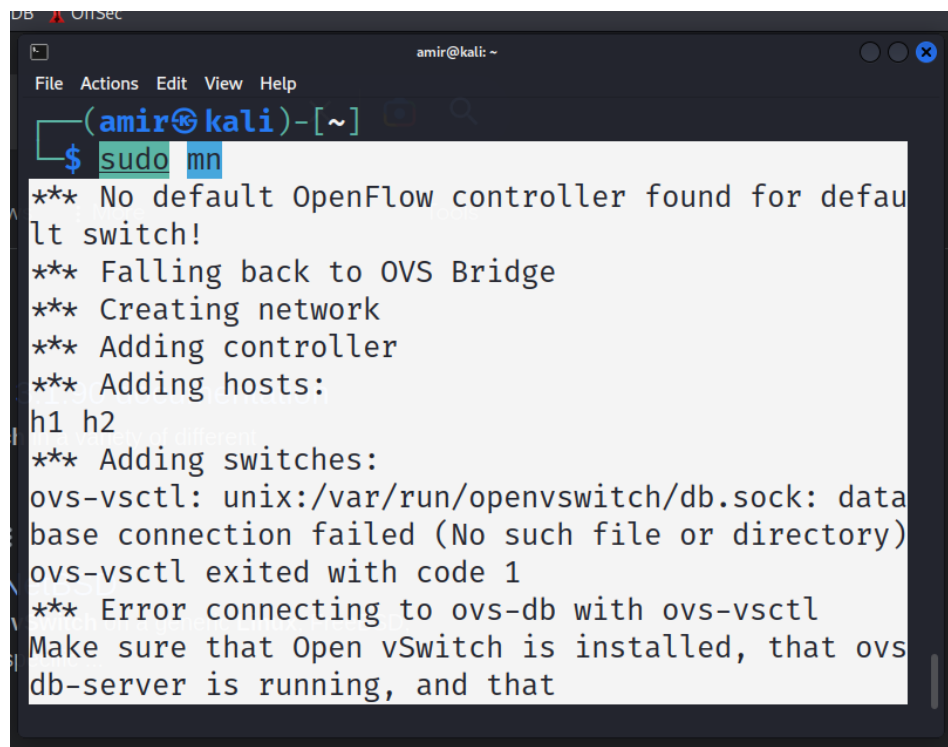
بعد، در کادر فیلتر Wireshark نزدیک بالای پنجره آن، این فیلتر را وارد کنید، سپس روی Apply کلیک کنید:

openflow_1



یک توپولوژی مینیمال را شروع می کنیم:

```
$ sudo mn
```



```
DB OnSec
amir@kali: ~
File Actions Edit View Help
(amir@kali)-[~]
$ sudo mn
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
ovs-vsctl: unix:/var/run/openvswitch/db.sock: database connection failed (No such file or directory)
ovs-vsctl exited with code 1
*** Error connecting to ovs-db with ovs-vsctl
Make sure that Open vSwitch is installed, that ovsdb-server is running, and that
```

توپولوژی پیش فرض توپولوژی حداقلی است که شامل یک سوئیچ هسته OpenFlow متصل به دو میزبان، به علاوه کنترل کننده مرجع OpenFlow است. این توپولوژی همچنین می تواند در خط فرمان با -- topo=minimal مشخص شود. توپولوژی های دیگر نیز خارج از جعبه موجود هستند. بخش -- topo را در خروجی mn -h ببینید.

هر چهار موجودیت (۲ فرآیند میزبان، ۱ فرآیند سوئیچ، ۱ کنترل کننده اصلی) اکنون در VM در حال اجرا هستند. کنترلر می تواند خارج از VM باشد و دستورالعمل های مربوط به آن در پایین است. اگر هیچ تست خاصی به عنوان پارامتر تصویب نشود، Mininet CLI ظاهر می شود. در پنجره Wireshark، باید ببینید که سوئیچ هسته به کنترل کننده مرجع متصل است.

نمایش دستورات Mininet CLI:

```
mininet> help
```

نمایش نودها:

```
mininet> nodes
```

نمایش لینک‌ها:

```
mininet> net
```

دور ریختن اطلاعات مربوط به همه گره‌ها:

```
mininet> dump
```

باید سوئیچ و دو میزبان را مشاهده کنید.

اگر اولین رشته‌ای که در Mininet CLI تایپ می‌شود یک نام میزبان، سوئیچ یا کنترلر باشد، دستور بر روی آن گره اجرا می‌شود. یک دستور را روی یک فرآیند میزبان اجرا می‌کنیم:

```
mininet> h1 ifconfig -a
```

شما باید رابط‌های h1-eth0 و loopback (lo) میزبان را ببینید. توجه داشته باشید که این رابط (h1-eth0) توسط سیستم اصلی لینوکس در هنگام اجرای ifconfig دیده نمی‌شود، زیرا مختص فضای نام شبکه فرآیند میزبان است.

در مقابل، سوئیچ به طور پیش‌فرض در فضای نام شبکه ریشه اجرا می‌شود، بنابراین اجرای یک فرمان روی «سوئیچ» مانند اجرای آن از یک ترمینال معمولی است:

```
mininet> s1 ifconfig -a
```

این رابط‌های سوئیچ، به علاوه اتصال ماشین مجازی خارج شده (eth0) را نشان می‌دهد.

برای مثال‌های دیگری که نشان می‌دهد هاست‌ها دارای حالت شبکه ایزوله هستند، arp و route را در هر دو s1 و h1 اجرا کنید.

امکان قراردادن هر میزبان، سوئیچ و کنترل‌کننده در فضای نام شبکه ایزوله خود وجود دارد، اما هیچ مزیتی برای انجام این کار وجود ندارد، مگر اینکه بخواهید یک شبکه چند کنترل‌کننده پیچیده را تکرار کنید. Mininet این را پشتیبانی می‌کند. گزینه --innamespace را ببینید.

توجه داشته باشید که فقط شبکه مجازی شده است. هر فرآیند میزبان مجموعه‌ای از فرآیندها و دایرکتوری‌ها را می‌بیند. به عنوان مثال، لیست فرآیند را از یک فرآیند میزبان چاپ کنید:

```
mininet> h1 ps -a
```

این باید دقیقاً همان چیزی باشد که توسط فضای نام شبکه ریشه مشاهده می‌شود:

```
mininet> s1 ps -a
```

استفاده از فضاهای پردازش جداگانه با کانتینرهای لینوکس ممکن است، اما در حال حاضر Mininet این کار را انجام نمی‌دهد. اجرای همه چیز در فضای نام فرآیند "ریشه" برای اشکال زدایی راحت است، زیرا به شما امکان می‌دهد تمام فرآیندهای کنسول را با استفاده از ps، kill و غیره مشاهده کنید.

تست اتصال بین هاست‌ها:

می‌توانید از میزبان ۰ به هاست ۱ پینگ کنید:

```
mininet> h1 ping -c 1 h2
```

اگر رشته‌ای بعداً در دستور با نام گره ظاهر شود، آن نام گره با آدرس IP آن جایگزین می‌شود. این اتفاق برای h2 افتاد.

باید ترافیک کنترل OpenFlow را ببینید. اولین میزبان ARP برای آدرس MAC دومی است که باعث می‌شود یک پیام packet_in به کنترلر برود. سپس کنترلر یک پیام packet_out می‌فرستد تا بسته پخش شده را به پورت‌های دیگر روی سوئیچ (در این مثال، تنها پورت داده دیگر) منتقل کند. میزبان دوم درخواست ARP را می‌بیند و یک پاسخ ارسال می‌کند. این پاسخ به کنترلر می‌رود، که آن را به میزبان اول می‌فرستد و ورودی جریان را پایین می‌آورد.

اکنون میزبان اول آدرس MAC دومی را می‌داند و می‌تواند پینگ خود را از طریق درخواست ICMP Echo ارسال کند. این درخواست، همراه با پاسخ مربوط به آن از میزبان دوم، هم به کنترلر می‌رود و هم منجر به پایین آمدن یک جریان ورودی می‌شود (همراه با ارسال بسته‌های واقعی).

آخرین پینگ را تکرار می‌کنیم:

```
mininet> h1 ping -c 1 h2
```

شما باید زمان پینگ بسیار کمتری را برای دومین تلاش مشاهده کنید (کمتر از ۱۰۰ μ s). یک ورودی جریانی که ترافیک پینگ ICMP را پوشش می‌دهد قبلاً در سوئیچ نصب شده بود، بنابراین ترافیک کنترلی ایجاد نمی‌شد و بسته‌ها بلافاصله از طریق سوئیچ عبور می‌کنند.

یک راه ساده‌تر برای اجرای این تست استفاده از دستور pingall داخلی Mininet CLI است که یک پینگ تمام جفت را انجام می‌دهد:

```
mininet> pingall
```

اجرای یک وب سرور و کلاینت ساده :

به یاد داشته باشید که پینگ تنها دستوری نیست که می‌توانید روی هاست اجرا کنید! میزبان‌های Mininet می‌توانند هر دستور یا برنامه‌ای را که برای سیستم لینوکس (یا VM) و سیستم فایل آن در دسترس است اجرا کنند. همچنین می‌توانید هر دستور bash را وارد کنید، از جمله کنترل کار (&، jobs، kill، و غیره)

سپس، سعی کنید یک سرور HTTP ساده را در h1 راهاندازی کنید، از h2 درخواست کنید، سپس وب سرور را خاموش کنید:

```
mininet> h1 python -m http.server 80 &
mininet> h2 wget -O - h1
...
mininet> h1 kill %python
```

برای پایتون ۳، سرور http.server HTTP نامیده می‌شود. برای پایتون ۲، SimpleHTTPServer نامیده می‌شود. مطمئن شوید که از نسخه مناسب برای نسخه Mininet که در حال اجرا هستید استفاده می‌کنید. برای اینکه بدانید Mininet از کدام نسخه پایتون استفاده می‌کند، می‌توانید تایپ کنید:

```
mininet> py sys.version
3.8.5 (default, Jan 27 2021, 15:41:15)
```

با دستور زیر از CLI خارج می‌شوید:

```
mininet> exit
```

پاک کردن:

اگر Mininet به دلایلی خراب شد، آن را پاک کنید:

```
$ sudo mn -c
```

Run a Regression Test

نیازی نیست وارد CLI شوید. Mininet همچنین می‌تواند برای اجرای تست‌های رگرسیون مستقل استفاده شود.

تست رگرسیون را اجرا می‌کنیم:

```
$ sudo mn --test pingpair
```

این دستور یک توپولوژی حداقل ایجاد کرد، کنترل‌کننده مرجع OpenFlow را راه‌اندازی کرد، یک تست پینگ تمام جفت را اجرا می‌کند و توپولوژی و کنترلر را مشخص می‌کند.

تست مفید دیگر iperf است:

```
$ sudo mn --test iperf
```

این دستور همان Mininet را ایجاد کرد، یک سرور iperf را روی یک هاست اجرا کرد، یک مشتری iperf را روی میزبان دوم اجرا کرد و پهنای باند به دست آمده را تجزیه کرد.

Changing Topology Size and Type

توپولوژی پیش فرض یک سوئیچ است که به دو میزبان متصل است. می‌توانید با `topo--` این را به یک `topo` دیگر تغییر دهید و پارامترهایی را برای ایجاد آن توپولوژی ارسال کنید. به عنوان مثال، برای تأیید اتصال همه جفت با یک سوئیچ و سه میزبان پینگ کنید:

تست رگرسیون را اجرا کنید:

```
$ sudo mn --test pingall --topo single,3
```

مثال دیگر، با توپولوژی خطی (که در آن هر سوئیچ یک میزبان دارد و همه سوئیچ‌ها در یک خط به هم متصل می‌شوند):

```
$ sudo mn --test pingall --topo linear,4
```

توپولوژی‌های پارامتری شده یکی از کاربردی‌ترین و قدرتمندترین ویژگی‌های Mininet هستند.

Link variations

Mininet 2.0 به شما امکان می‌دهد پارامترهای پیوند را تنظیم کنید، و حتی می‌توان آن‌ها را به طور خودکار از خط فرمان تنظیم کرد:

```
$ sudo mn --link tc,bw=10,delay=10ms
mininet> iperf
...
mininet> h1 ping -c10 h2
```

اگر تأخیر برای هر پیوند ۱۰ میلی‌ثانیه باشد، زمان رفت و برگشت (RTT) باید حدود ۴۰ میلی‌ثانیه باشد، زیرا درخواست ICMP از دو پیوند (یکی به سوئیچ، یکی به مقصد) عبور می‌کند و پاسخ ICMP دو پیوند را که برمی‌گردد، طی می‌کند.

Adjustable Verbosity

سطح پرحرفی پیش‌فرض اطلاعات است که آنچه را که Mininet در حین راه‌اندازی و خراب کردن انجام می‌دهد چاپ می‌کند. این را با خروجی کامل دیباگ با پارامتر `-v` مقایسه می‌کنیم:

```
$ sudo mn -v debug
...
mininet> exit
```

بسیاری از جزئیات اضافی چاپ خواهد شد. اکنون خروجی را امتحان می‌کنیم، تنظیمی که خروجی CLI را چاپ می‌کند و چیزهای دیگر:

```
$ sudo mn -v output
mininet> exit
```

خارج از CLI، سطوح دیگر پرحرفی را می‌توان استفاده کرد، مانند هشدار، که با تست‌های رگرسیون برای پنهان کردن خروجی عملکرد غیر ضروری استفاده می‌شود.

Custom Topologies

توپولوژی‌های سفارشی را می‌توان به راحتی با استفاده از یک API ساده پایتون تعریف کرد و یک مثال در `custom/topo-2sw-2host.py` ارائه شده است. این مثال دو سوئیچ را مستقیماً وصل می‌کند و هر سوئیچ یک هاست خاموش است:

```
1 """Custom topology example
2
3 Two directly connected switches plus a host for each switch:
4
5  host --- switch --- switch --- host
6
7 Adding the 'topos' dict with a key/value pair to generate our newly defined
8 topology enables one to pass in '--topo=mytopo' from the command line.
9 """
10
11 from mininet.topo import Topo
12
13 class MyTopo( Topo ):
14     "Simple topology example."
15
16     def build( self ):
17         "Create custom topo."
18
```

```

19 # Add hosts and switches
20 leftHost = self.addHost( 'h1' )
21 rightHost = self.addHost( 'h2' )
22 leftSwitch = self.addSwitch( 's3' )
23 rightSwitch = self.addSwitch( 's4' )
24
25 # Add links
26 self.addLink( leftHost, leftSwitch )
27 self.addLink( leftSwitch, rightSwitch )
28 self.addLink( rightSwitch, rightHost )
29
30
31 topos = { 'mytopo': ( lambda: MyTopo() ) }

```

هنگامی که یک فایل mininet سفارشی ارائه می‌شود، می‌تواند توپولوژی‌های جدید، انواع سوئیچ‌ها و تست‌ها را به خط فرمان اضافه کند. مثلاً:

```
$ sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --topo mytopo --test pingall
```

ID = MAC

به طور پیش فرض، هاست‌ها با آدرس‌های MAC اختصاص داده شده به طور تصادفی شروع می‌شوند. این می‌تواند اشکال‌زدایی را سخت کند، زیرا هر بار که Mininet ایجاد می‌شود، MAC‌ها تغییر می‌کنند، بنابراین ارتباط ترافیک کنترل با میزبان‌های خاص دشوار است.

گزینه `mac--` بسیار مفید است و افزودنی‌های MAC و IP میزبان را روی شناسه‌های کوچک، منحصر به فرد و آسان برای خواندن تنظیم می‌کند.

قبل:

```

$ sudo mn
...
mininet> h1 ifconfig
h1-eth0  Link encap:Ethernet  HWaddr f6:9d:5a:7f:41:42
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0

```

```

UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:6 errors:0 dropped:0 overruns:0 frame:0
TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:392 (392.0 B)  TX bytes:392 (392.0 B)

mininet> exit

```

بعد:

```

$ sudo mn --mac
...
mininet> h1 ifconfig
h1-eth0  Link encap:Ethernet  HWaddr 00:00:00:00:00:01
         inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet> exit

```

در مقابل، MACها برای پورت‌های داده سوئیچ گزارش شده توسط لینوکس به صورت تصادفی باقی خواهند ماند. این به این دلیل است که همانطور که در پرسش‌های متداول ذکر شده است، می‌توانید با استفاده از OpenFlow یک MAC را به درگاه داده اختصاص دهید. این یک نکته تا حدودی ظریف است که احتمالاً در حال حاضر می‌توانید آن را نادیده بگیرید.

XTerm Display

برای اشکال‌زدایی پیچیده‌تر، می‌توانید Mininet را راه‌اندازی کنید تا یک یا چند xterm ایجاد کند.

برای شروع xterm برای هر میزبان و سوئیچ، گزینه -x را استفاده کنید:

```
$ sudo mn -x
```

پس از یک ثانیه، xterms با نام پنجره به طور خودکار تنظیم می‌شود.

به طور پیش فرض، فقط هاست‌ها در یک فضای نام جداگانه قرار می‌گیرند. پنجره هر سوئیچ غیرضروری است (یعنی معادل یک ترمینال معمولی)، اما می‌تواند مکانی مناسب برای اجرای و خروج دستورات رفع اشکال سوئیچ، مانند تخلیه‌های شمارنده جریان باشد.

Xterm‌ها همچنین برای اجرای دستورات تعاملی مفید هستند که ممکن است لازم باشد آنها را لغو کنید و می‌خواهید خروجی آن را ببینید.

مثلاً:

در xterm با برچسب "switch: s1 (root)" اجرا کنید:

```
# ovs-ofctl dump-flows tcp:127.0.0.1:6654
```

هیچ چیز چاپ نمی‌شود. سوئیچ هیچ جریانی اضافه نشده است. برای استفاده از ovs-ofctl با سوئیچ‌های دیگر، mininet را در حالت کلامی راه‌اندازی کنید و هنگام ایجاد سوئیچ‌ها، به درگاه‌های گوش دادن غیرفعال برای سوئیچ‌ها نگاه کنید.

اکنون در xterm با عنوان "host: h1" اجرا کنید:

```
# ping 10.0.0.2
```

به s1 برگردید و جریان‌ها را تخلیه کنید:

```
# ovs-ofctl dump-flows tcp:127.0.0.1:6654
```

اکنون باید چندین ورودی جریان را مشاهده کنید. به طور متناوب (و به طور کلی راحت تر)، می‌توانید از دستور dpctl تعبیه شده در Mininet CLI بدون نیاز به xterms یا تعیین دستی IP و پورت سوئیچ استفاده کنید.

با بررسی ifconfig می‌توانید بفهمید که xterm در فضای نام ریشه قرار دارد یا خیر. اگر همه اینترفیس‌ها نشان داده شوند (از جمله eth0)، در فضای نام ریشه قرار دارد. علاوه بر این، عنوان آن باید حاوی "(ریشه)" باشد.

تنظیمات را از Mininet CLI ببندید:

```
mininet> exit
```

xterms باید به طور خودکار بسته شود.

سایر انواع سوئیچ

می توان از انواع دیگر سوئیچ ها استفاده کرد. به عنوان مثال، برای اجرای سوئیچ فضای کاربر:

```
$ sudo mn --switch user --test iperf
```

به پهنای باند بسیار کمتر گزارش شده توسط TCP iperf در مقایسه با آنچه قبلاً در سوئیچ هسته مشاهده شده بود توجه کنید.

اگر تست پینگ نشان داده شده در قبل را انجام دهید، باید تاخیر بسیار بیشتری را متوجه شوید، زیرا اکنون بسته ها باید انتقال اضافی هسته به فضای کاربر را تحمل کنند. زمان پینگ متغیرتر خواهد بود، زیرا فرآیند فضای کاربر که میزبان میزبان است ممکن است توسط سیستم عامل برنامه ریزی شود.

از سوی دیگر، سوئیچ فضای کاربر می تواند نقطه شروعی عالی برای اجرای عملکردهای جدید باشد، به خصوص در مواردی که عملکرد نرم افزار حیاتی نیست.

نمونه دیگر نوع سوئیچ Open vSwitch (OVS) است که روی Mininet VM از قبل نصب شده است. پهنای باند TCP گزارش شده توسط iperf باید مشابه ماژول هسته OpenFlow و احتمالاً سریع تر باشد:

```
$ sudo mn --switch ovsk --test iperf
```


معیار Mininet

برای ثبت زمان تنظیم و از بین بردن توپولوژی، از تست "none" استفاده کنید:

```
$ sudo mn --test none
```

Everything in its own Namespace (user switch only)

به طور پیش فرض، هاست‌ها در فضای نام خود قرار می‌گیرند، در حالی که سوئیچ‌ها و کنترل‌کننده‌ها در فضای نام ریشه هستند. برای قرار دادن سوئیچ‌ها در فضای نام خود، گزینه `--innamespace` را پاس کنید:

```
$ sudo mn --innamespace --switch user
```

به جای استفاده از حلقه بک، سوئیچ‌ها از طریق یک اتصال کنترلی پل شده جداگانه با کنترلر صحبت می‌کنند. این گزینه به خودی خود چندان مفید نیست، اما نمونه‌ای از نحوه جداسازی سوئیچ‌های مختلف را ارائه می‌دهد.

```
mininet> exit
```

دستورات رابط خط فرمان (CLI) Mininet

Display Options

برای مشاهده لیست گزینه‌های Command-Line Interface (CLI)، یک توپولوژی مینیمال راه‌اندازی کنید و آن را در حال اجرا بگذارید:

```
$ sudo mn
```

نمایش گزینه‌ها:

```
mininet> help
```

Python Interpreter

اگر اولین عبارت در خط فرمان Mininet py باشد، آن دستور با پایتون اجرا می‌شود. این ممکن است برای گسترش Mininet و همچنین بررسی عملکرد درونی آن مفید باشد. هر میزبان، سوئیچ و کنترل کننده یک شی Node مرتبط دارد.

در Mininet CLI، اجرا کنید:

```
mininet> py 'hello ' + 'world'
```

متغیرهای محلی قابل دسترسی را چاپ کنید:

```
mininet> py locals()
```

سپس، با استفاده از تابع dir، متدها و خواص موجود برای یک گره را ببینید:

```
mininet> py dir(s1)
```

با استفاده از تابع help() می‌توانید مستندات آنلاین روش‌های موجود در یک گره را بخوانید:

```
mininet> py help(h1) (Press "q" to quit reading the documentation.)
```

شما همچنین می‌توانید روش‌های متغیرها را ارزیابی کنید:

```
mininet> py h1.IP()
```

Link Up/Down

برای تست تحمل خطا، up و down کردن لینک‌ها می‌تواند مفید باشد.

برای غیرفعال کردن هر دو نیمه یک جفت اترنت مجازی:

```
mininet> link s1 h1 down
```

باید ببینید که یک اعلان تغییر وضعیت پورت OpenFlow ایجاد می‌شود. برای بازگردانی لینک:

```
mininet> link s1 h1 up
```

XTerm Display

برای نمایش xterm برای h1 و h2:

```
mininet> xterm h1 h2
```

پایتون API

دایرکتوری مثال‌ها در درخت منبع Mininet شامل نمونه‌هایی از نحوه استفاده از برنامه کاربردی Python Mininet و همچنین کدهای بالقوه مفیدی است که در پایه کد اصلی ادغام نشده‌اند.

همانطور که در ابتدا ذکر شد، این Walkthrough فرض می‌کند که شما یا از یک VM Mininet استفاده می‌کنید، که شامل همه چیزهایی است که نیاز دارید، یا یک نصب بومی با تمام ابزارهای مرتبط، از جمله کنترل‌کننده کنترل‌کننده مرجع، که بخشی از مرجع OpenFlow است. پیاده‌سازی و ممکن است با استفاده از `install.sh -f` نصب شود اگر نصب نشده باشد.

SSH daemon per host

یک مثال که ممکن است بسیار مفید باشد، یک شبح SSH را در هر میزبان اجرا می‌کند:

```
$ sudo ~/mininet/examples/sshd.py
```

از یک ترمینال دیگر، می‌توانید به هر میزبانی ssh کنید و دستورات تعاملی را اجرا کنید:

```
$ ssh 10.0.0.1
$ ping 10.0.0.2
...
```

```
$ exit
```

خروج از SSH نمونه Mininet:

```
$ exit
```

برای تسلط بیشتر بر Mininet

اگر نمی‌دانید چگونه از یک کنترل‌کننده از راه دور استفاده کنید (مثلاً کنترل‌کننده‌ای که خارج از کنترل Mininet اجرا می‌شود)، در زیر توضیح داده شده است.

Using a Remote Controller

این مرحله بخشی از پیش فرض پیش فرض نیست. در درجه اول مفید است اگر یک کنترلر در خارج از VM اجرا می‌شود، مانند میزبان VM، یا یک کامپیوتر فیزیکی دیگر. آموزش OpenFlow از کنترل از راه دور برای راه اندازی یک سوئیچ یادگیری ساده استفاده می‌کند که با استفاده از یک چارچوب کنترل کننده مانند NOX، POX، Beacon یا Floodlight ایجاد می‌کنید.

هنگامی که یک شبکه Mininet را راه اندازی می‌کنید، هر سوئیچ می‌تواند به یک کنترل از راه دور متصل شود که می‌تواند در ماشین مجازی، خارج از ماشین مجازی و ماشین محلی شما یا هر جای دنیا باشد.

اگر از قبل یک نسخه سفارشی از چارچوب کنترلر و ابزارهای توسعه (مانند Eclipse) را روی دستگاه محلی نصب کرده‌اید، یا می‌خواهید کنترل‌کننده‌ای را که روی یک ماشین فیزیکی متفاوت اجرا می‌شود (شاید حتی در فضای ابری) آزمایش کنید، ممکن است راحت باشد.

اگر می‌خواهید این را امتحان کنید، IP میزبان و/یا پورت گوش دادن را پر کنید:

```
$ sudo mn --controller=remote,ip=[controller IP],port=[controller listening port]
```

به عنوان مثال، برای اجرای سوئیچ یادگیری نمونه POX، می‌توانید کاری شبیه به آن انجام دهید

```
$ cd ~/pox
$ ./pox.py forwarding.12_learning
```

در یک پنجره، و در پنجره دیگر، Mininet را راه اندازی کنید تا به کنترل کننده "ریموت" متصل شود (که در واقع به صورت محلی اجرا می شود، اما خارج از کنترل Mininet):

```
$ sudo mn --controller=remote,ip=127.0.0.1,port=6633
```

توجه داشته باشید که POX از پورت قدیمی OpenFlow 6633 استفاده می کند که ثبت نشده بود و بعداً توسط سیسکو گرفته شد. پورت فعلی، ثبت شده/متعارف برای OpenFlow پورت ۶۶۵۳ است. لطفاً از شماره پورت مناسب برای کنترلر خود استفاده کنید.

به طور پیش فرض، --controller=remote از ۱۲۷.۰.۰.۱ استفاده می کند و پورت های ۶۶۵۳ و ۶۶۳۳ را امتحان می کند.

اگر مقداری ترافیک ایجاد می کنید (به عنوان مثال h1 ping h2) باید بتوانید برخی از خروجی ها را در پنجره POX مشاهده کنید که نشان می دهد سوئیچ متصل شده است و برخی از ورودی های جدول جریان نصب شده اند.

تعدادی از چارچوب های کنترل کننده OpenFlow به راحتی در دسترس هستند و باید به راحتی با Mininet کار کنند تا زمانی که آن ها را راه اندازی کنید و گزینه کنترل از راه دور را با آدرس IP صحیح دستگاهی که کنترلر شما در آن کار می کند و پورت صحیحی که در حال گوش دادن به آن است را مشخص کنید.

کنترلرهای OpenFlow زیادی در دسترس هستند و می توانید تعداد بیشتری از آن ها را با استفاده از گوگل یا موتور جستجوی مورد علاقه خود به راحتی پیدا کنید.

Ryu

Ryu، یک چارچوب کنترل کننده OpenFlow اولیه (و تا حدودی POX مانند) که در پایتون نوشته شده است FAUCET، یک کنترلر (همچنین در پایتون نوشته شده و بر اساس چارچوب Ryu) که از سوئیچینگ اترنت و مسیریابی IP و همچنین سایر ویژگی ها از طریق یک فایل پیکربندی ساده YML پشتیبانی می کند.

ONOS، یک سیستم عامل شبکه با امکانات کامل، که به زبان جاوا نوشته شده است

OpenDaylight، "بزرگترین کنترل کننده SDN منبع باز"

همه این کنترلرها را می توان با Mininet یا یک شبکه سخت افزاری استفاده کرد.

Ryu یک چارچوب کنترل کننده OpenFlow است که در پایتون نوشته شده است. در Mininet خارج از جعبه پشتیبانی می شود:

```
$ sudo pip3 install ryu # install ryu if it's not already installed
$ sudo mn --controller ryu
```

این ryu.app.simple_switch را اجرا می کند.

همچنین می توانید برنامه Ryu را در خط فرمان mn مشخص کنید:

```
$ sudo mn --controller,ryu.app.simple_switch_13
```

شما همچنین می توانید Ryu را به عنوان یک کنترل از راه دور اجرا کنید.

در یک پنجره:

```
$ ryu run ryu.app.simple_switch
```

سپس در پنجره دیگری:

```
$ sudo mn --controller remote
```

فایل های کد مربوط به پروژه و اسکرین شات های مربوطه در یک فایل زیپ شده در کنار گزارش است.

پایان گزارش