**Mininet示例工作流程**

Mininet使您能够快速[创建](http://mininet.org/sample-workflow/" \l "create)，[交互](http://mininet.org/sample-workflow/#interact)，[自定义](http://mininet.org/sample-workflow/#customize)和[共享](http://mininet.org/sample-workflow/#share)软件定义的网络原型，并提供[在硬件](http://mininet.org/sample-workflow/#run)上[运行](http://mininet.org/sample-workflow/#run)的平滑路径。本页面说明Mininet的基本工作流程，Mininet [演练](http://mininet.org/walkthrough)，OpenFlow [教程](https://github.com/mininet/openflow-tutorial/wiki)和Mininet [文档](https://github.com/mininet/mininet/wiki/Documentation)中提供了许多其他详细信息。

**创建网络**

您可以使用单个命令创建网络。例如，

**sudo mn --switch ovs --controller ref --topo tree,depth=2,fanout=8 --test pingall**

使用OpenFlow / Stanford参考控制器控制下的Open vSwitch交换机启动具有深度2和扇出8（即，连接到9个交换机的64个主机）的树形拓扑的网络，并运行**pingall**测试以检查每对节点之间的连通性。（我的笔记本电脑需要大约45秒。）

**与网络互动**

Mininet的CLI允许您从单个控制台控制和管理整个虚拟网络。例如，CLI命令

**mininet> h2 ping h3**

告诉主机**h2**ping主机**h2**的IP地址。*任何可用的Linux命令或程序都可以在任何虚拟主机上运行*。您可以轻松地在一台主机上启动Web服务器，并从另一台主机发出HTTP请求：

**mininet> h2 python -m SimpleHTTPServer 80 >& /tmp/http.log &**

**mininet> h3 wget -O - h2**

**自定义网络**

Mininet的API允许您使用几行Python创建自定义网络。例如，以下脚本

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | **from mininet.net import Mininet**  **from mininet.topolib import TreeTopo**  **tree4 = TreeTopo(depth=2,fanout=2)**  **net = Mininet(topo=tree4)**  **net.start()**  **h1, h4 = net.hosts[0], net.hosts[3]**  **print h1.cmd('ping -c1 %s' % h4.IP())**  **net.stop()** |

创建一个小型网络（4台主机，3台交换机），并从另一台主机上ping一台主机（在当前版本的约4秒内）。

Mininet发行版包括几个基于文本和图形的应用程序（请参阅上文），我们希望这些应用程序具有启发性，并鼓励您为自己的网络设计创建酷炫而实用的应用程序。

**共享网络**

Mininet作为虚拟机（VM）映像进行分发，预先安装了所有依赖项，可在VMware，Xen和VirtualBox等通用虚拟机监视器上运行。这为分发提供了一个方便的容器; 一旦原型开发完成，虚拟机映像可能会分发给其他人运行，检查和修改。一个完整的，压缩的Mininet虚拟机大约1GB。（Mininet也可以在本地安装 - **apt-get install mininet**在Ubuntu上）。如果您正在阅读关于软件定义网络的一篇很好的SIGCOMM（或其他）论文，您不希望能够点击，下载并运行一个生动的呼吸示例系统的？如果是这样，考虑开发一个Mininet版本的自己的系统，你可以与他人分享！

**在硬件上运行**

一旦Mininet设计出来，它就可以部署在硬件上用于实际使用，测试和测量。

要在第一次尝试中成功连接到硬件，每个Mininet仿真组件必须以与其相应物理组件相同的方式运行。虚拟拓扑应与物理拓扑匹配; 虚拟以太网对必须由链路级以太网连接替代。主机仿真为进程应由具有自己的操作系统映像的主机替换。另外，每个模拟的OpenFlow交换机应该被配置为指向控制器的物理交换机替代。但是，控制器不需要更改。当Mininet运行时，控制器“看到”交换机的物理网络，通过具有良好定义的状态语义的接口变得可能。

**Mininet演练**

本演练演示了大多数Mininet命令，以及与Wireshark解剖器一起使用的典型用法。

*演练假定您的基本系统是Mininet VM，或者安装了所有OpenFlow工具和Mininet的本机Ubuntu安装（通常使用Mininet完成****install.sh****）。*

整个演练应该在一个小时内完成。

* [第1部分：每日Mininet使用](http://mininet.org/walkthrough/#part-1-everyday-mininet-usage)
  + [显示启动选项](http://mininet.org/walkthrough/#display-startup-options)
  + [启动Wireshark](http://mininet.org/walkthrough/#start-wireshark)
  + [与主机和交换机进行交互](http://mininet.org/walkthrough/#interact-with-hosts-and-switches)
  + [测试主机之间的连接](http://mininet.org/walkthrough/#test-connectivity-between-hosts)
  + [运行一个简单的Web服务器和客户端](http://mininet.org/walkthrough/#run-a-simple-web-server-and-client)
  + [清理](http://mininet.org/walkthrough/#cleanup)
* [第2部分：高级启动选项](http://mininet.org/walkthrough/#part-2-advanced-startup-options)
  + [运行回归测试](http://mininet.org/walkthrough/#run-a-regression-test)
  + [更改拓扑大小和类型](http://mininet.org/walkthrough/#changing-topology-size-and-type)
  + [链接变化](http://mininet.org/walkthrough/#link-variations)
  + [可调整的详细程度](http://mininet.org/walkthrough/#adjustable-verbosity)
  + [自定义拓扑](http://mininet.org/walkthrough/#custom-topologies)
  + [ID = MAC](http://mininet.org/walkthrough/#id--mac)
  + [XTerm显示](http://mininet.org/walkthrough/#xterm-display)
  + [其他交换机类型](http://mininet.org/walkthrough/#other-switch-types)
  + [Mininet基准](http://mininet.org/walkthrough/#mininet-benchmark)
  + [它自己的命名空间中的所有内容（仅限用户切换）](http://mininet.org/walkthrough/#everything-in-its-own-namespace-user-switch-only)
* [第3部分：Mininet命令行界面（CLI）命令](http://mininet.org/walkthrough/#part-3-mininet-command-line-interface-cli-commands)
  + [显示选项](http://mininet.org/walkthrough/#display-options)
  + [Python解释器](http://mininet.org/walkthrough/#python-interpreter)
  + [链接向上/向下](http://mininet.org/walkthrough/#link-updown)
  + [XTerm显示](http://mininet.org/walkthrough/#xterm-display-1)
* [第4部分：Python API示例](http://mininet.org/walkthrough/#part-4-python-api-examples)
  + [每台主机的SSH守护进程](http://mininet.org/walkthrough/#ssh-daemon-per-host)
* [第5部分：演练完成！](http://mininet.org/walkthrough/#part-5-walkthrough-complete)
  + [下一步掌握Mininet](http://mininet.org/walkthrough/#next-steps-to-mastering-mininet)
* [附录：补充信息](http://mininet.org/walkthrough/#appendix-supplementary-information)
  + [使用遥控器](http://mininet.org/walkthrough/#using-a-remote-controller)
  + [NOX经典](http://mininet.org/walkthrough/#nox-classic)

*注意：如果您使用的是Ubuntu Mininet 2.0.0d4软件包，它使用的语法略有不同，****Topo()****例如****add\_switch****vs****addSwitch****等。如果您从源代码签出Mininet，则可能希望查看****2.0.0d4****标签以查看代码（包括代码****examples****）与2.0.04包一致。*

**第1部分：每日Mininet使用**

首先，本演练的命令语法（可能很明显）：

* **$** 在应该在shell提示符下输入的Linux命令之前
* **mininet>** 在Mininet命令行输入Mininet命令之前，
* **#** 在根shell提示符下输入的Linux命令之前

*在每种情况下，只应在提示的右侧键入命令（然后按****return****！）！*

**显示启动选项**

让我们开始使用Mininet的启动选项。

键入以下命令以显示描述Mininet启动选项的帮助消息：

**$ sudo mn -h**

本演练将涵盖列出的大多数选项的典型用法。

**启动Wireshark**

要使用OpenFlow Wireshark解剖器查看控制流量，请先在后台打开wireshark：

**$ sudo wireshark &**

在Wireshark过滤器框中，输入此过滤器，然后单击**Apply**：

**of**

在Wireshark中，单击捕获，然后单击接口，然后在回送接口（**lo**）上选择启动。

目前，主窗口中不应该显示OpenFlow数据包。

*注意：Wireshark默认安装在Mininet VM映像中。如果您使用的系统没有安装Wireshark和OpenFlow插件，则可以使用Mininet的****install.sh****脚本安装它们，如下所示：*

**$ cd ~**

**$ git clone https://github.com/mininet/mininet # if it's not already there**

**$ mininet/util/install.sh -w**

如果Wireshark已安装但无法运行（例如，您遇到类似的错误 **$DISPLAY not set**，请参阅FAQ：[https](https://github.com/mininet/mininet/wiki/FAQ" \l "wiki-x11-forwarding)：[//github.com/mininet/mininet/wiki/FAQ#wiki-x11-forwarding](https://github.com/mininet/mininet/wiki/FAQ" \l "wiki-x11-forwarding)。）

正确设置X11将使您能够运行**xterm**本演练稍后使用的其他GUI程序和终端仿真程序。

**与主机和交换机进行交互**

启动最小化拓扑并输入CLI：

**$ sudo mn**

默认拓扑结构是**minimal**拓扑结构，其中包括连接到两台主机的一个OpenFlow内核交换机以及OpenFlow参考控制器。这个拓扑结构也可以在命令行中指定**--topo=minimal**。其他拓扑结构也可以直接使用; 请参阅**--topo**输出中的部分**mn -h**。

所有四个实体（2个主机进程，1个切换进程，1个基本控制器）现在都在VM中运行。控制器可以位于虚拟机外部，并且指令位于虚拟机的底部。

如果没有特定的测试作为参数传递，Mininet CLI出现。

在Wireshark窗口中，您应该看到内核开关连接到参考控制器。

显示Mininet CLI命令：

**mininet> help**

显示节点：

**mininet> nodes**

显示链接：

**mininet> net**

转储有关所有节点的信息：

**mininet> dump**

您应该看到交换机和两台主机列出。

如果键入Mininet CLI的第一个字符串是主机，交换机或控制器名称，则该命令将在该节点上执行。在主机进程上运行命令：

**mininet> h1 ifconfig -a**

您应该看到主机**h1-eth0**和loopback（**lo**）接口。请注意，此接口（**h1-eth0**）在**ifconfig**运行时不会被主Linux系统看到，因为它特定于主机进程的网络名称空间。

相反，默认情况下，交换机在根网络名称空间中运行，因此在“switch”上运行命令与从常规终端运行命令相同：

**mininet> s1 ifconfig -a**

这将显示交换机接口，以及虚拟机的连接out（**eth0**）。

对于突出显示的主机已经分离的网络状态，运行其它实例**arp**和**route**在两个**s1**和**h1**。

将每个主机，交换机和控制器放置在自己的独立网络命名空间中是可能的，但除非您想复制复杂的多控制器网络，否则这样做并没有真正的优势。Mininet支持这一点; 看到**--innamespace**选项。

请注意，*只有*网络是虚拟化的; 每个主机进程都会看到相同的一组进程和目录。例如，从主机进程打印进程列表：

**mininet> h1 ps -a**

这应该与根网络名称空间所看到的完全相同：

**mininet> s1 ps -a**

有可能在Linux容器中使用单独的进程空间，但目前Mininet不这样做。在“根”的过程命名空间冒一切方便调试，因为它允许您使用的是看所有从控制台的过程中**ps**，**kill**等

**测试主机之间的连接**

现在，验证您可以从主机0 ping主机1：

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

如果一个字符串稍后出现在具有节点名称的命令中，则该节点名称会被其IP地址替换; 这发生在h2。

您应该看到OpenFlow控制流量。第一个主机ARP用于第二个主机的MAC地址，这会导致**packet\_in**消息进入控制器。然后控制器发送**packet\_out**消息将广播数据包泛洪到交换机上的其他端口（在本例中是唯一的其他数据端口）。第二个主机看到ARP请求并发送回复。这个回复发送给控制器，控制器将其发送给第一个主机并下推一个流表项。

现在第一台主机知道第二台主机的MAC地址，并且可以通过ICMP回显请求发送ping。这个请求以及来自第二个主机的相应答复都会进入控制器，并导致流量条目下降（以及实际发送的数据包）。

重复最后一项**ping**：

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

你应该看到**ping**第二次尝试（<100us）的时间要短得多。**ping**之前在交换机中安装了覆盖ICMP 流量的流表项，因此没有生成控制流量，并且数据包立即通过交换机。

运行此测试的一种更简单的方法是使用Mininet CLI内置**pingall**命令，该命令完成以下操作**ping**：

**mininet> pingall**

**运行一个简单的Web服务器和客户端**

请记住，这**ping**不是您可以在主机上运行的唯一命令！Mininet主机可以运行底层Linux系统（或VM）及其文件系统可用的任何命令或应用程序。您也可以输入任何**bash**命令，包括作业控制（**&**，**jobs**，**kill**，等。）

接下来，尝试启动一个简单的HTTP服务器**h1**，从中发出请求**h2**，然后关闭Web服务器：

**mininet> h1 python -m SimpleHTTPServer 80 &**

**mininet> h2 wget -O - h1**

**...**

**mininet> h1 kill %python**

退出CLI：

**mininet> exit**

**清理**

如果Mininet因某种原因崩溃，请清理它：

**$ sudo mn -c**

**第2部分：高级启动选项**

**运行回归测试**

您不需要放入CLI; Mininet也可以用来运行自包含的回归测试。

运行回归测试：

**$ sudo mn --test pingpair**

该命令创建了一个最小的拓扑结构，启动了OpenFlow参考控制器，运行了全配对**ping**测试，并拆除了拓扑和控制器。

另一个有用的测试是**iperf**（完成约10秒）：

**$ sudo mn --test iperf**

该命令创建相同的Mininet，在一台主机上运行iperf服务器，在第二台主机上运行iperf客户端，并分析实现的带宽。

**更改拓扑大小和类型**

默认拓扑是连接到两台主机的单个交换机。您可以将其更改为不同的拓扑**--topo**，并传递该拓扑创建的参数。例如，要验证使用一台交换机和三台主机进行全配对ping连接：

运行回归测试：

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

另一个例子是线性拓扑结构（每个交换机有一个主机，并且所有交换机连接成一条线路）：

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

参数化拓扑是Mininet最有用和最强大的功能之一。

**链接变化**

Mininet 2.0允许您设置链接参数，甚至可以从命令行自动设置它们：

**$ sudo mn --link tc,bw=10,delay=10ms**

**mininet> iperf**

**...**

**mininet> h1 ping -c10 h2**

如果每条链路的延迟为10毫秒，则往返时间（RTT）应该约为40毫秒，因为ICMP请求经过两条链路（一条到交换机，一条到目的地），并且ICMP应答遍历返回的两条链路。

*您可以使用*[*Mininet的Python API*](https://github.com/mininet/mininet/wiki/Introduction-to-Mininet)*自定义每个链接 ，但现在您可能会想要继续演练。*

**可调整的详细程度**

默认的详细级别是**info**，它显示Mininet在启动和拆卸过程中所做的操作。**debug**用**-v**param 比较这个完整的输出：

**$ sudo mn -v debug**

**...**

**mininet> exit**

许多额外的细节将打印出来。现在尝试一下**output**，打印CLI输出和其他一些设置：

**$ sudo mn -v output**

**mininet> exit**

在CLI之外，可以使用其他详细级别，例如**warning**，用于回归测试以隐藏不需要的函数输出。

**自定义拓扑**

自定义拓扑结构也可以使用简单的Python API轻松定义，并提供了一个示例**custom/topo-2sw-2host.py**。此示例直接连接两台交换机，每台交换机关闭一台主机：

简单的拓扑示例（topo-2sw-2host.py）[下载](http://mininet.org/mininet/custom/topo-2sw-2host.py)

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32**  **33**  **34** | **"""Custom topology example**  **Two directly connected switches plus a host for each switch:**  **host --- switch --- switch --- host**  **Adding the 'topos' dict with a key/value pair to generate our newly defined**  **topology enables one to pass in '--topo=mytopo' from the command line.**  **"""**  **from mininet.topo import Topo**  **class MyTopo( Topo ):**  **"Simple topology example."**  **def \_\_init\_\_( self ):**  **"Create custom topo."**  **# Initialize topology**  **Topo.\_\_init\_\_( self )**  **# Add hosts and switches**  **leftHost = self.addHost( 'h1' )**  **rightHost = self.addHost( 'h2' )**  **leftSwitch = self.addSwitch( 's3' )**  **rightSwitch = self.addSwitch( 's4' )**  **# Add links**  **self.addLink( leftHost, leftSwitch )**  **self.addLink( leftSwitch, rightSwitch )**  **self.addLink( rightSwitch, rightHost )**  **topos = { 'mytopo': ( lambda: MyTopo() ) }** |

当提供自定义mininet文件时，它可以将新拓扑，开关类型和测试添加到命令行。例如：

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

**ID = MAC**

默认情况下，主机以随机分配的MAC地址开始。这可能会使调试变得困难，因为每次创建Mininet时，MAC都会改变，因此将控制流量与特定主机关联起来非常困难。

该**--mac**选项非常有用，并将主机MAC和IP地址设置为小而唯一且易于阅读的ID。

之前：

**$ 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**

后：

**$ 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**

*相比之下，Linux报告的交换机数据端口MAC将保持随机。这是因为如FAQ中所述，您可以使用OpenFlow将MAC分配给数据端口。这是一个有点微妙的地方，你现在可以忽略。*

**XTerm显示**

对于更复杂的调试，您可以启动Mininet，以便生成一个或多个xterm。

要启动**xterm**每个主机和交换机，请传递**-x**选项：

**$ sudo mn -x**

一秒钟后，xterm将弹出，并自动设置窗口名称。

或者，您可以调出其他xterm，如下所示。

默认情况下，只有主机放在单独的名称空间中; 每个交换机的窗口是不必要的（即相当于一个普通的终端），但是可以方便地运行并保留交换机调试命令，例如流量计数器转储。

Xterms对于运行可能需要取消的交互式命令也很有用，您希望看到输出。

例如：

在标有“switch：s1（root）”的xterm中，运行：

**# dpctl dump-flows tcp:127.0.0.1:6634**

什么都不会打印出来; 交换机没有增加流量。要**dpctl**与其他交换机一起使用，请在详细模式下启动mininet，并在交换机创建时查看交换机的被动侦听端口。

现在，在标有“host：h1”的xterm中，运行：

**# ping 10.0.0.2**

返回**s1**并转储流程：＃dpctl dump-flows tcp：127.0.0.1：6634

你现在应该看到多个流条目。或者（通常更方便），您可以使用**dpctl**内置在Mininet CLI中的命令，而不需要任何xterms或手动指定交换机的IP和端口。

您可以通过检查ifconfig来判断xterm是否位于根名称空间中; 如果显示所有接口（包括**eth0**），则它位于根名称空间中。另外，它的标题应该包含“（根）”。

从Mininet CLI关闭安装程序：

**mininet> exit**

xterm应该自动关闭。

**其他交换机类型**

其他开关类型可以使用。例如，要运行用户空间开关：

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

请注意，与先前使用内核交换机所看到的相比，TCP Iperf报告的带宽要低得多。

如果你执行前面所示的ping测试，你应该注意到更高的延迟，因为现在数据包必须忍受额外的内核到用户空间的转换。由于表示主机的用户空间进程可能由OS调度和调出，因此ping时间将更加可变。

另一方面，用户空间交换机可能是实现新功能的一个很好的起点，尤其是在软件性能不重要的情况下。

另一种开关类型的示例是预装在Mininet VM上的Open vSwitch（OVS）。iperf报告的TCP带宽应该与OpenFlow内核模块类似，并且可能更快：

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

**Mininet基准**

要记录设置和拆除拓扑的时间，请使用test'none'：

**$ sudo mn --test none**

**它自己的命名空间中的所有内容（仅限用户切换）**

默认情况下，主机放在它们自己的名称空间中，而交换机和控制器放在根名称空间中。要将开关置于其自己的名称空间中，请传递**--innamespace**选项：

**$ sudo mn --innamespace --switch user**

交换机不会使用环回，而是通过单独的桥接控制连接与控制器通信。就其本身而言，这个选项并不是非常有用，但它确实提供了一个如何隔离不同交换机的例子。

请注意，此选项不适用于（2012年11月11日）Open vSwitch。

**mininet> exit**

**第3部分：Mininet命令行界面（CLI）命令**

**显示选项**

要查看命令行界面（CLI）选项的列表，请启动最小拓扑并保持运行状态。建立Mininet：

**$ sudo mn**

显示选项：

**mininet> help**

**Python解释器**

如果Mininiet命令行中的第一个短语是**py**，那么该命令是用Python执行的。这对于扩展Mininet可能很有用，并且可以探究其内部工作方式。每个主机，交换机和控制器都有一个关联的节点对象。

在Mininet CLI中，运行：

**mininet> py 'hello ' + 'world'**

打印可访问的局部变量：

**mininet> py locals()**

接下来，使用dir（）函数查看可用于节点的方法和属性：

**mininet> py dir(s1)**

您可以使用help（）函数阅读节点上可用方法的在线文档：

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

您也可以评估变量的方法：

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

**链接向上/向下**

对于容错测试，将链接上下链接可能会有所帮助。

要禁用虚拟以太网对的两半：

**mininet> link s1 h1 down**

您应该看到生成了OpenFlow端口状态更改通知。要恢复链接：

**mininet> link s1 h1 up**

**XTerm显示**

为h1和h2显示xterm：

**mininet> xterm h1 h2**

**第4部分：Python API示例**

Mininet源代码树中的[示例目录](https://github.com/mininet/mininet/tree/master/examples)包括如何使用Mininet的Python API以及尚未集成到主代码库中的潜在有用代码的示例。

*注意：如前所述，本演练假定您使用的是Mininet虚拟机，其中包括您需要的所有内容，或者本机安装包含所有相关工具，包括参考控制器****controller****，它是OpenFlow参考实现的一部分****install.sh -f****如果尚未安装，可以使用它进行安装。*

**每台主机的SSH守护进程**

一个可能特别有用的例子是在每台主机上运行一个SSH守护进程：

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

从另一个终端，您可以SSH入任何主机并运行交互式命令：

**$ ssh 10.0.0.1**

**$ ping 10.0.0.2**

**...**

**$ exit**

退出SSH示例mininet：

**$ exit**

阅读[介绍Mininet](https://github.com/mininet/mininet/wiki/Introduction-to-Mininet)介绍Python API 后，您将希望重新访问这些示例 。

**第5部分：演练完成！**

恭喜！你已经完成了Mininet演练。随意尝试新的拓扑和控制器或查看源代码。

**下一步掌握Mininet**

如果你还没有这样做，你应该通过 [OpenFlow教程](https://github.com/mininet/openflow-tutorial/wiki)。

尽管使用Mininet的CLI可以得到相当的效果，但是在掌握Python API时，Mininet变得更加有用和强大。在 [介绍Mininet](https://github.com/mininet/mininet/wiki/Introduction-to-Mininet) 将介绍Mininet和Python的API。

如果您想知道如何使用*遥控器*（例如在Mininet *控制器*外部运行的*遥控器*），下面将对此进行说明。

**附录：补充信息**

这些都不是必需的，但你可能会发现它们对脱脂有用。

**使用远程控制器**

*注意：这一步不是默认漫游的一部分; 如果您有一个在虚拟机之外运行的控制器，例如在VM主机或另一台物理PC上运行，那么它将非常有用。OpenFlow教程****controller --remote****用于启动您使用POX，NOX，Beacon或Floodlight等控制器框架创建的简单学习开关。*

启动Mininet网络时，每台交换机都可以连接到一个远程控制器 - 可以位于虚拟机中，虚拟机外部，本地机器或全球任何地方。

如果您已经在本地机器上安装了控制器框架和开发工具（如Eclipse）的自定义版本，或者您想测试在不同物理机器上运行的控制器（甚至可能在云中），则此设置可能会很方便。 。

如果您想尝试此操作，请填写主机IP和/或侦听端口：

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

例如，要运行POX的示例学习开关，您可以执行类似操作

**$ cd ~/pox**

**$ ./pox.py forwarding.l2\_learning**

在另一个窗口中，在另一个窗口中，启动Mininet以连接到“远程”控制器（实际上它在本地运行，但不受Mininet控制）：

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

请注意，这些实际上是默认的IP地址和端口值。

如果您生成一些流量（例如**h1 ping h2**），您应该能够在POX窗口中观察一些输出，显示交换机已连接并且已安装了一些流表条目。

许多OpenFlow控制器框架随时可用，只要您启动它们并且指定**remote**控制器选项以及控制器正在运行的计算机的正确IP地址以及正在监听的正确端口，就应该可以与Mininet 一起使用。

**NOX经典**

Mininet默认安装（使用**util/install.sh -a**）不会安装NOX Classic。如果您想安装它，请运行**sudo ~/mininet/util/install.sh -x**。

请注意，NOX Classic已弃用，未来可能不支持。

要运行与NOX运行NOX的应用程序回归测试**pyswitch**中，**NOX\_CORE\_DIR**的环境变量必须设置为包含NOX可执行文件的目录。

首先验证NOX是否运行：

**$ cd $NOX\_CORE\_DIR**

**$ ./nox\_core -v -i ptcp:**

Ctrl-C杀死NOX，然后用NOX pyswitch运行一个测试：

**$ cd**

**$ sudo -E mn --controller=nox,pyswitch --test pingpair**

请注意，该**--controller**选项有一个方便的语法来指定控制器类型的选项（在这种情况下，**nox**正在运行）**pyswitch**。

NOX加载和交换机连接时有几秒钟的犹豫，但是它应该完成**ping**。

请注意，这次**mn**是通过调用**sudo -E**来保持**NOX\_CORE\_DIR**环境变量。如果你远程运行nox，使用**--controller remote**，那么这**-E**是没有必要的。或者，您可以更改第一行**/etc/sudoers**，通过**sudo visudo**将**env\_reset**行更改为：

**Defaults !env\_reset**

...所以当运行**sudo**env var setup时不会改变。