# 实验 4、Mininet 的安装与运行

电子科技大学信通学院　　杨宁



参考资料：锐捷网络&广东交通职业技术学院《SDN 培训课程——Mininet 介绍 v3.1》

# 实验原理

## 1、Mininet 简介

Mininet 是由斯坦福大学研究开发的开源软件，是一个基于 Linux Container（LXC）虚拟化技术的轻量级网络模拟器，或者更准确地说是一个网络仿真编排系统。它可以在个人电脑上模拟出包括终端主机、交换机、路由器和链接，如图 1 所示。它使用轻量级虚拟化使单个系统看起来像一个完整的网络，运行相同的内核、系统和用户代码。

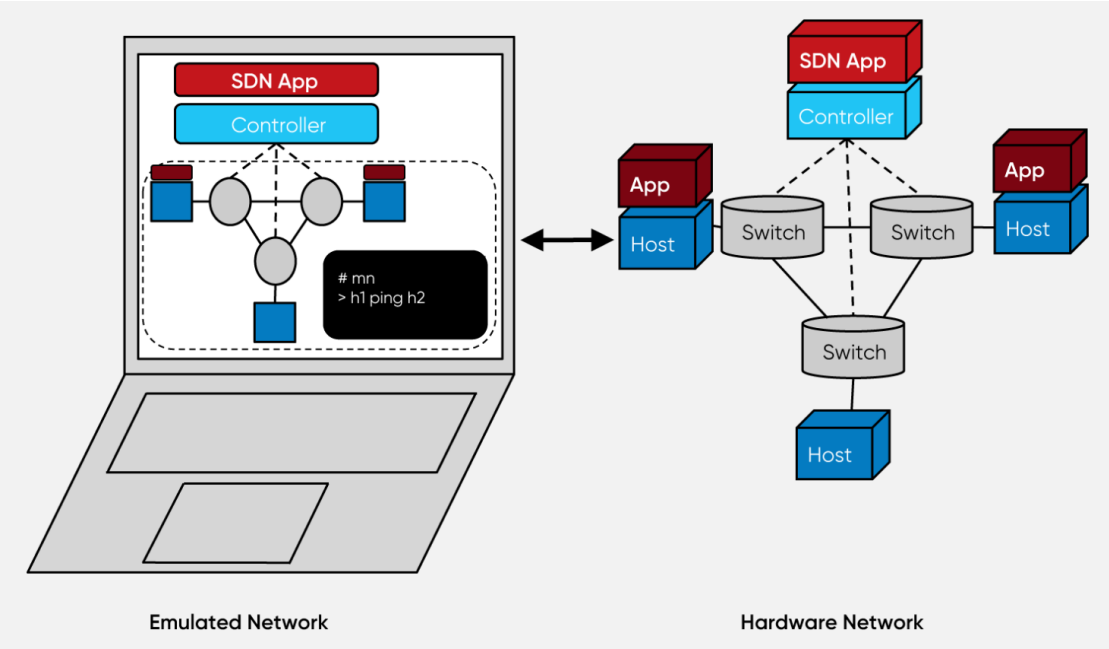



图 1 Mininet 的模拟网络

Mininet 是基于 LXC 这一内核虚拟化技术开发的，使用 Linux 内核中的 Network Namespace 资源隔离机制，让每个 Namespace 具有独立的网络设备、网络协议栈和端口等，因此 Mininet 中的节点行为就像真机一样，如果开启了 sshd 服务并桥接到物理主机网络，就可以通过 SSH 进行连接并运行任意程序。

Mininet 中的每个虚拟节点都有自己的虚拟以太网接口，每个接口对于所有系统和应用程序软件来说都是功能齐全的以太网接口。这些虚拟的以太网接口通过 Linux 的 Virtual Ethernet（veth） pair 相互连接，veth pair 就像连接两个虚拟接口或虚拟交换机端口的网线一样。每个模拟链接的数据速率由 Linux Traffic Control（TC）控制实现。

Mininet 提供三种内置控制器：OpenFlow 参考控制器、OVS 控制器和 NOX Classic，还支持外接远程 SDN 控制器，并提供 python API。

简而言之，Mininet 的虚拟主机、交换机、链接和控制器是真实的——它们只是使用软件而不是硬件创建的，并且大多数情况下它们的行为类似于分立的硬件

元素。这是 Mininet 的一个重要特点，也是其相比于 NS、OPNET 等传统网络仿真平台的一大优点，Mininet 的所有代码可以无缝迁移到真实的硬件环境，方便为网络添加新的功能并进行相关测试。

按照数据通路（datapath：在数字系统中，各个子系统通过数据总线连接形成的数据传送路径）运行权限不同，Mininet 有两种架构：kernel datapath 架构和 userspace datapath 架构。此外，Mininet 还支持以内核模式运行的 OVS，其性能和 kernel dtatpath 差不多。

（1）kernel datapath

图 2 所示的 kernel datapath 架构将分组转发的逻辑编译进 Linux 内核，效率非常高。

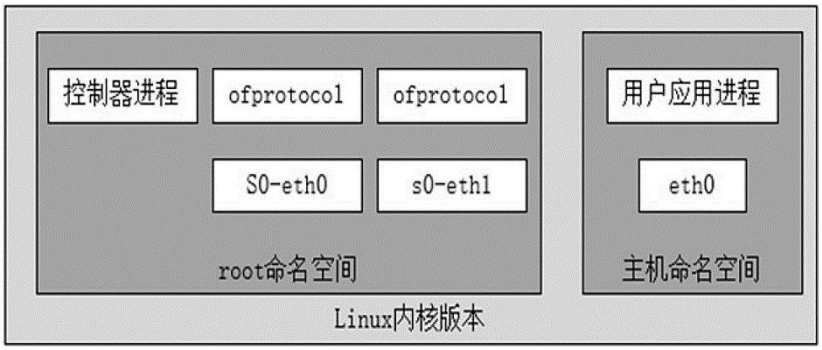



图 2 Mininet 的 kernel datapath 架构

控制器和交换机的网络接口都在 root 命名空间。控制器是一个用户进程，在 loopback 上预留的 6633 端口监听来自交换机安全信道的连接。Mininet 的每个交换机都对应几个网络接口，如 s0-eth0、s0-eth1 以及一个 ofprotocol 进程，它负责管理和维护同一控制器之间安全信道的连接。

每个主机都在自己独立命名空间中，有自己独立的虚拟网卡 eth0。

（2）userspace datapath

图 3 所示的 userspace datapath 架构将分组转发的逻辑实现为一个运行在用户空间的 ofdatpath 应用程序，效率不及 kernel datapath，但更灵活，容易重新编译。

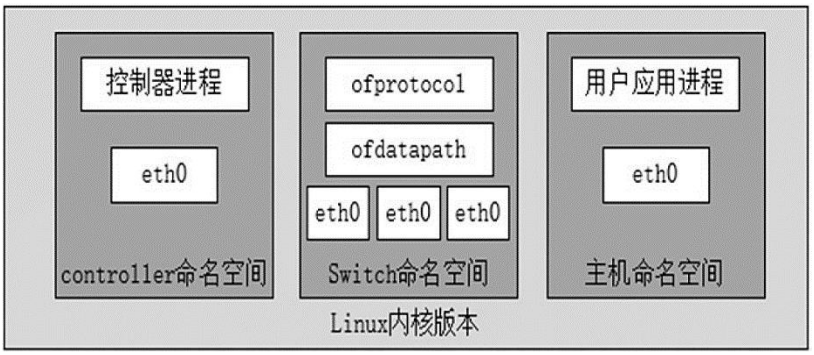



图 3 Mininet 的 userspace datapath 架构

userspace datapath 架构中的每个节点（包括控制器、交换机和主机）都有自己独立的 namespace。

Mininet 具有如下特性：

（1）灵活性：通过软件的方式（python 脚本）简单、快速的创建一个用户自定义的网络拓扑，可以自定义转发分组。

（2）可移植性：在 Mininet 上运行的代码可以轻松移植到支持 OpenFLow 的硬件设备上。

（3）可拓展性：模拟的网络规模可以拓展到成千上百节点。

（4）真实性：模拟真实网络环境，运行实际的网络协议栈，实时管理配置网络，可以运行真实的程序。

Mininet 是一款开源软件，其官网为 http://mininet.org。本实验安装和使用的版本是 Mininet 2.2.2。

## 2、Mininet 安装方式

Mininet 有以下四种安装方式：

- **Mininet 虚拟机安装**：这是最简单、最安全的安装方式。直接通过官网将 Mininet VM 镜像下载到本地电脑上，即可在本地电脑的虚拟化软件（如 VirtualBox、VMware Workstation、Linux KVM 等）打开并启动虚拟机。
- **源码安装**：这种安装方式适合在本地电脑、本地电脑中的 VM 系统或远程云服务器上安装，建议在 Ubuntu 中进行安装。可以通过 git 或其他途径下载源代码包，然后在操作系统中执行 install.sh 脚本进行编译安装。Mininet 提供了表 1 所示的几种源码安装命令组合。
- **软件包安装**：如果正在运行最新的 Ubuntu 系统，则可以安装 Mininet 软件包。这是一种便捷的安装方式，但可能安装的是旧版本的 Mininet。
- **升级安装**：这种安装方式只会升级 Mininet，其他组件如 OVS 等都需要单独升级。不推荐使用这种安装方式。

表 1 Mininet 源码安装的命令组合

| 命令 | 含义 |
|---|---|
| install.sh -a | 在用户的 home 目录中安装 Mininet 的所有内容，包括 OVS 等依赖项。 |
| install.sh -s /mydir -a | 在/mydir 目录下安装 Mininet 的所有内容。 |
| install.sh -nfv | 仅在用户的 home 目录中安装 Mininet、OpenFlow 协议支持、OVS。 |
| install.sh -s /mydir -nfv | 在/mydir 目录下安装 Mininet、OpenFlow 协议支持、OVS。 |
| install.sh -h | 查看其他安装选项。 |

**本实验采用源码安装方式。**

### 3、Mininet 可视化应用：MiniEdit

从 2.2.0 版本开始，Mininet 内置了一个可视化工具 MiniEdit，可以方便用户创建自定义拓扑，为不熟悉 python 脚本的使用者创造了更简单的操作环境。在源码安装的 Mininet 中，可视化应用 MiniEdit 在~/mininet/mininet/examples 目录下（其中“~”表示 Mininet 源码安装的目录），其启动脚本为 miniedit.py。

MiniEdit 图形化界面如图 4 所示：

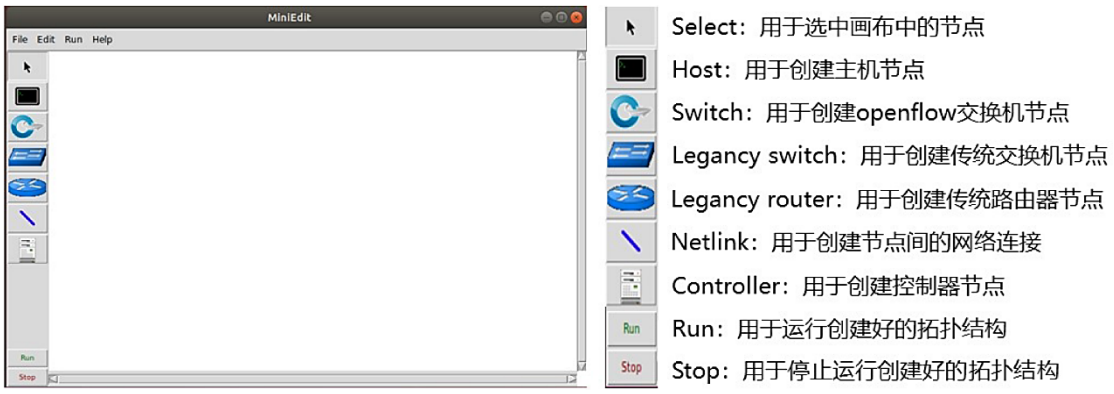



图 4 MiniEdit 界面

选择“Edit”菜单中的“Preferences”（如图 5 所示）进行 Mininet 全局配置：

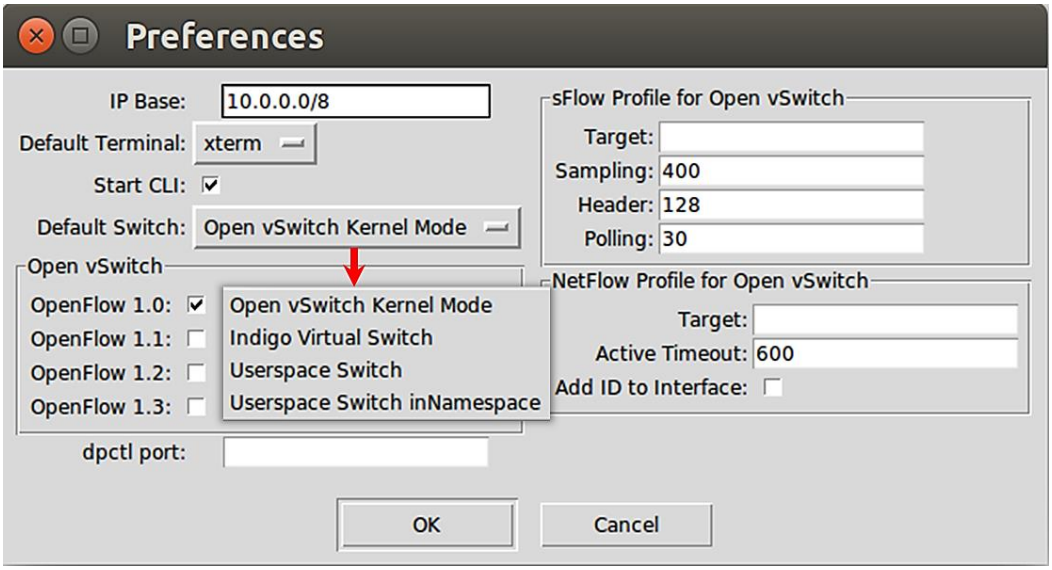



图 5 Mininet 全局设置

- 可以设置构建的 SDN 网络 IP 网段；
- 可以勾选“Start CLI”，使得点击左下角“Run”按钮运行拓扑后，能在如图 6 所示的 Mininet 命令行界面中直接执行 Mininet 命令操作；

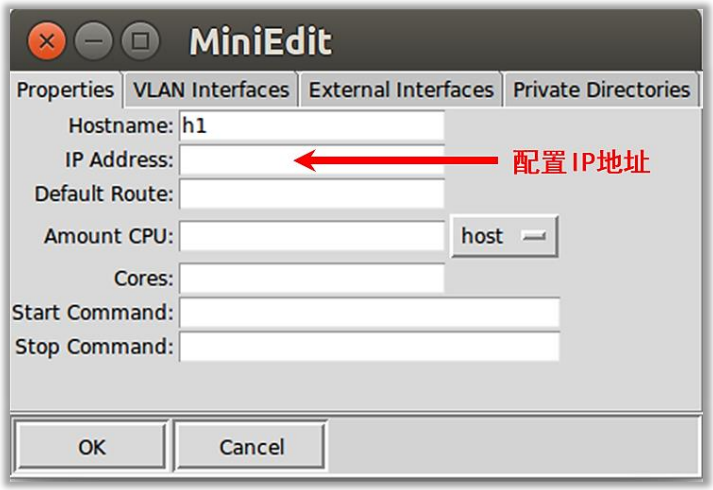
```
 NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTTON. Not exi
ting will prevent MiniEdit from quitting and will prevent you from starting the
network again during this sessoin.

*** Starting CLI:
mininet>
```

图 6 在 MiniEdit 中运行拓扑后的 Mininet 命令行界面

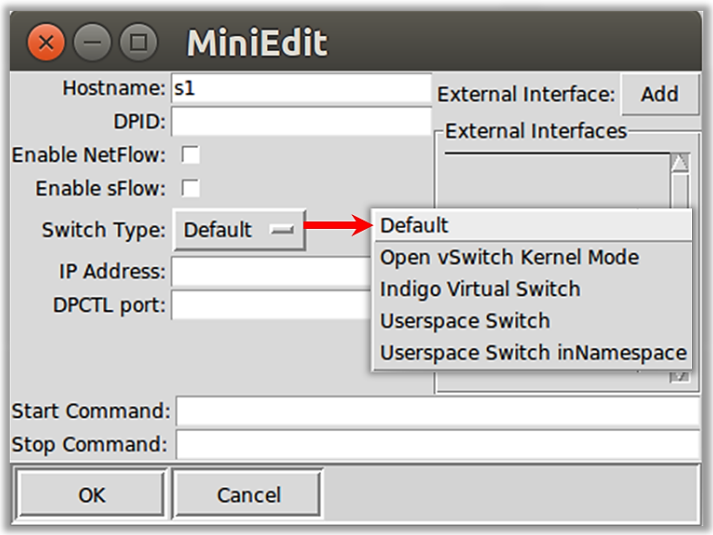
- 可以选择交换机的类型，以及支持的 OpenFlow 协议版本（可多选）。

用鼠标选择 MiniEdit 界面左侧的网络组件，然后在空白区域单击鼠标左键即可添加网络组件。在创建节点间的网络连接时，需要注意节点的端口号取决于连接顺序。例如交换机 s1 连接两个主机 h1 和 h2，如果先连接 h1 和 s1，再连接 h2 和 s1，那么 s1 的 eth1 连接 h1，s1 的 eth2 则连接 h2。

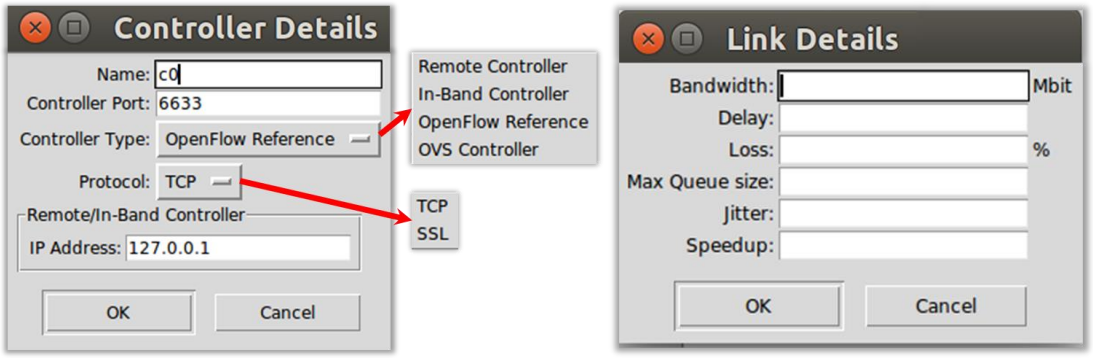
在添加的主机、交换机、控制器或链路元件上长按鼠标右键选择“Properties”，即可设置其属性。主机配置主要是 IP 地址，如图 7（a）所示。交换机配置包括 DPID 和交换机类型，如图 7（b）所示。如果不设置 DPID 值，Mininet 会自动按照交换机元件的添加顺序从 1 开始递增。控制器配置可以选择使用 Mininet 自带控制器或远程控制器，如图 7（c）所示。链路配置包括带宽、延迟、丢包率、最大队列数等，如图 7（d）所示。




（a）主机属性




（b）交换机属性




（c）控制器属性　　（d）链路属性

图 7 MiniEdit 中链路和各种节点的“Properties”配置

点击 MiniEdit 界面左下角“Run”按钮运行拓扑后，可以查看以下信息：

- 通过选择“Run”菜单中的“Show OVS Summary”，可以查看拓扑中的所有交换机概要信息，等效于 ovs-vsctl show 命令，如图 8（a）所示；
- 鼠标右键点击长按界面中的交换机元件，可以查看详细的配置信息，等效于 ovs-vsctl list br [BRIDGE]命令，如图 8（b）所示；
- 鼠标右键点击长按界面中的主机元件，可以直接打开主机的命令行终端，如图 8（c）所示；
- 鼠标右键点击长按界面中的链路元件，可以选择开启（Link Up）或关闭（Link Down）链路。


（a）所有交换机信息　　（b）交换机选项　　（c）主机选项

图 8 运行拓扑后可以在 MiniEdit 中查看的信息

使用 MiniEdit 图形界面构建的拓扑，可以通过选择“File”菜单中的“Save”选项将其保存为文件名后缀为“.mn”的 Mininet 拓扑文件，该文件可以在 MiniEdit 中打开并运行。还可以选择“File”菜单中的“Export Level 2 Script”选项，将构建的拓扑保存为 python 脚本文件。新建的 python 脚本文件没有执行权限，需要使用 chmod 命令为 python 脚本文件添加执行权限，例如：

```
# chmod +x test.py
```

此后，直接运行 python 脚本即可重现拓扑，并可在 Mininet 命令行界面中进

行操作。

一般情况下，只需关闭 MiniEdit 界面即可清除所有实验环境，包括自建的控制器、交换机、主机等，不会影响下一次实验。但是在极端情况下，例如模拟大型网络环境致使 MiniEdit 卡死，MiniEdit 可能没有办法正常自动清理所创建的实验环境，此时则需要运行以下命令进行清除：

```
# mn -c
```

**4、mn 命令**

Mininet 安装完成之后，即会在操作系统中添加一个 mn 命令。使用 mn 命令可以创建并运行网络拓扑结构。相比于 MiniEdit 图形化界面，使用 mn 命令可以创建较为复杂的网络拓扑，可以设置自定义参数。

mn 命令常用参数选项如表 2 所示：

表 2 mn 命令的常用参数选项

| 选项 | 选项值 | 含义 |
|---|---|---|
| -h 或 --help | | 查看 mn 命令的参数选项。 |
| -c 或 --clear | | 清除 Mininet 环境。 |
| --switch= | default | 无--switch 选项或使用--switch default 选项都是 OVSSwitch。 |
| | ivs | IVSSwitch，即 Indigo Virtual Switch。 |
| | lxbr,stp=1 | LinuxBridge，其中 stp=1 表示开启生成树协议（STP）。 |
| | ovs | OVSSwitch |
| | ovsbr | OVSBridge，处于桥接模式的 OVS，即如果连不上控制器则转换为进行 MAC 自学习的传统二层交换机，支持 STP。 |
| | ovsk | OVSSwitch，为了兼容 Mininet 2.0 而保留的参数。 |
| | user | UserSwitch，运行在用户空间的 OpenFlow 交换机，带宽较小，ping 延迟较大。 |
| --host= | proc | Host，普通主机；无--host 选项的默认项。 |
| | cfs | CPU 有限的主机，根据周期时间内消耗的 CPU 时间进行任务调度。 |
| | rt | CPU 有限的主机，实时任务调度。 |
| --controller= | default | Mininet 自带的 OpenFlow 参考控制器，无--controller 选项的默认项。 |
| | none | NullController，即不使用控制器。 |
| | nox | Mininet 自带的 NOX 控制器。 |
| | ovsc | Mininet 自带的 OVS 控制器。 |
| | ref | Mininet 自带的 OpenFlow 参考控制器。 |
| | remote,ip=x.x.x.x,port=y | 远程控制器，需要有 ip 和 port 参数。 |
| | ryu | Ryu 控制器。 |

| 选项 | 选项值 | 含义 |
|---|---|---|
| --link= | default | Link，由 veth pair 构成的普通链路；无--link 选项的默认项。 |
| | ovs | OVSLink，OVSSwitch 之间的 patch 链路。 |
| | tc,bw=10,delay='5ms',loss=0,max_queue_size=1000 | TCLink，使用 TC 控制链路的带宽（bw，单位 Mbit）、延迟（delay）、丢包率（loss）和最大队列长度（max_queue_size）。 |
| | tcu | TCULink，为优化 UserSwitch 而设置的 TCLink。 |
| --topo= | minimal | 最小拓扑，1 个交换机连接 2 个主机；无--topo 选项的默认项。 |
| | single,n | 星型拓扑，1 个交换机连接 n 个主机。 |
| | reversed,n | 星型拓扑，但是主机编号大小与交换机端口号大小正好相反，即最小编号的主机连接交换机最大端口号。 |
| | linear,x,y | 线性拓扑，x 个交换机线性相连，每个交换机连有 y 个主机。 |
| | tree,depth=x,fanout=y | 树型拓扑，树的深度为 x，广度（树枝节点的分支数）为 y，做为树根和树枝节点的交换机个数为$(y^x-1)/(y-1)$，做为树叶节点的主机个数为$y^x$。 |
| | torus,x,y | 双向环型拓扑，x 个交换机环形相连，每个交换机连有 y 个主机。 |
| --custom= | xxx.py | 加载 xxx.py 自定义拓扑。 |
| --mac | | 自动将主机的 MAC 地址设置为主机 ID，易于辨识，且每次启动时不会随机改变。 |
| -x 或 --xterms | | 为拓扑中的每个设备节点开启一个命令行窗口。 |

下面是一些 Mininet 基本命令示例：

- 创建并运行一个最小拓扑，交换机类型为 ovsbr

```
# mn --switch=ovsbr
```

- 执行 file.py 脚本文件，创建并运行一个自定义拓扑

```
# mn --custom=file.py --topo mytopo
```

- 创建并运行一个包含 3 个主机的星型拓扑，使用远程控制器

```
# mn --topo=single,3 --controller=remote,ip=10.1.1.10,port=6633
```

运行不带任何参数的 mn 命令即表示使用 Mininet 生成了一个最小拓扑（minimal），拓扑中只有 1 个交换机连接有 2 个主机，控制器使用 Mininet 自带的 OpenFlow 参考控制器，如图 9 所示。

```
root@SDN-PC1:~# mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

图 9 运行不带任何参数的 mn 命令

### 5、Mininet 交互式命令行

使用 MiniEdit 图形界面或 mn 命令创建并运行一个网络拓扑后，即会进入到 Mininet 命令行界面，使用 help 命令可以查看交互式命令行中可用的命令，如图 10 所示。

```
mininet> help

Documented commands (type help <topic>):
========================================
EOF    gterm  iperfudp  nodes        pingpair      py      switch
dpctl  help   link      noecho       pingpairfull  quit    time
dump   intfs  links     pingall      ports         sh      x
exit   iperf  net       pingallfull  px            source  xterm

You may also send a command to a node using:
  <node> command {args}
For example:
  mininet> h1 ifconfig

The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
  mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
  mininet> noecho h2 vi foo.py
However, starting up an xterm/gterm is generally better:
  mininet> xterm h2

mininet>
```

图 10 Mininet 交互式命令行中可用的命令

这些命令可用于对运行的拓扑进行信息查询和测试。常用命令如表 3 所示。

表 3 常用的 Mininet 交互式命令

| 命令 | | 含义 |
|---|---|---|
| 查询 | help | 查看可用的 Mininet 交互式命令。 |
| | nodes | 列出当前拓扑中的节点（包括所有控制器、主机和交换机）。 |
| | dump | 显示当前拓扑中所有节点的详细信息。 |
| | net | 显示当前拓扑中的节点及其端口连接情况 |
| | links | 显示当前拓扑中的所有链路状态 |
| 测试 | pingall | 当前拓扑中的每个主机 ping 拓扑中的其他主机 |
| | pingpair | 当前拓扑中的某个主机 ping 另一个主机，例如：pingpair h1 h2 |
| | iperf | 在两个主机之间进行 iperf TCP 带宽测试，例如：iperf h1 h2 |
| | iperfudp | 在两个主机之间进行 iperf UDP 带宽测试，例如：iperfudp bw h1 h2 |
| 操作 | link | 启用或禁用节点间的链路，例如：link s1 s2 up |
| | dpctl | 增、删、改、查当前拓扑中所有交换机的流表，等效于 OVS 中的于 ovs-ofctl 命令，例如：dpctl dump-flows |
| | xterm | 为当前拓扑中的某个节点开启一个命令行窗口，例如：xterm h1 |
| | sh | 执行一个外部 shell 命令（如 ovs-vsctl、ovs-ofctl 等），例如：<br>• 显示所有 OVS 概要信息：sh ovs-vsctl show<br>• 显示交换机 s1 的所有流表：sh ovs-ofctl dump-flows s1 |
| | py | 执行 python 表达式，例如：<br>• 为当前拓扑增加 1 个交换机：py net.addSwitch(“s3”)<br>• 为当前拓扑增加 1 个主机：py net.addHost(“h6”)<br>• 添加一条从交换机 s3 到主机 h6 的连接：<br>py net.addLink(s3,net.get(“h6”))<br>• 为 s3 添加一个用于连接主机的端口 eth3：py s3.attach(“s3-eth3”)<br>• 将主机 h6 的 IP 地址设置为 10.0.0.6：<br>py net.get(“h6”).cmd(“ifconfig h6-eth0 10.0.0.6”)<br>• 查看交换机 s3 的可用方法或命令：py help(s1) |

## 6、通过 Python 脚本调用 Mininet

Mininet 的大部分模块都是用 python 语言编写的，因此还可以通过调用 python 脚本来构建网络拓扑。

调用 python 脚本来构建网络拓扑，需要在 python 脚本中导入 Mininet 相关模块，然后调用 Mininet 函数来构建网络拓扑。在 mn 命令中使用--custom=file.py 参数项加载自定义拓扑时的 python 脚本只需要导入 topo 模块即可。但如果需要在 python 脚本中使用 Mininet 的所有命令，就必须导入所有模块，导入语句如表 4 所示。

表 4 在 python 脚本中导入 Mininet 模块

| 导入语句 | 含义 |
|---|---|
| from mininet.clean import * | 导入 clean 清理命令模块。 |
| from mininet.cli import * | 导入 cli 命令行模块。 |

| 导入语句 | 含义 |
| --- | --- |
| from mininet.log import * | 导入 log 日志模块。 |
| from mininet.net import * | 导入 net 网络命令模块。 |
| from mininet.node import * | 导入 node 节点命令模块。 |
| from mininet.nodelib import * | 导入 nodelib 节点命令模块，用于桥接或 NAT 网络。 |
| from mininet.topo import * | 导入 topo 拓扑结构模块（常用、关键模块）。 |
| from mininet.topolib import * | 导入 topolib 拓扑结构模块，用于使用树型结构拓扑。 |
| from mininet.link import * | 导入 link 模块，用于设置带宽等参数，必须先导入 node 模块才能导入。 |

一般情况下使用 python 脚本调用 Mininet 创建拓扑需要导入 net 和 topo 模块。除了导入模块、自定义拓扑参数之外，生成 Mininet 拓扑结构的 python 脚本还应包括 Mininet 的启动和关闭参数。下面是一个用 python 生成 Mininet 线性拓扑（k=4 表示 4 个交换机）的脚本文件 py-linear.py 示例：

```
#!/usr/bin/python

########导入模块########
from mininet.net import *
from mininet.topo import *

########自定义拓扑参数########
Linear4=LinearTopo(k=4)
net=Mininet(topo=Linear4)

######## Mininet 启动、测试和关闭########
net.start()
net.pingAll()
net.stop()
```

**说明 1**：文件中的第一行“**#!/usr/bin/python**”是指定用 python 解释器运行该脚本文件、以及解释器所在位置。有这一行语句，即可直接运行该脚本文件：

```
# ./py-linear.py
```

如果没有这一行语句，那么就必须使用如下命令执行该脚本文件：

```
# python py-linear.py
```

**说明 2**：如果 python 脚本文件是在 windows 系统中编辑，然后传到 Ubuntu 系统中，那么可能因为 windows 的行结尾和 Linux 行结尾字符不同，使得在 Linux 系统中执行时会报错。解决方案如下：

在 Ubuntu 系统中使用 vim 命令打开 python 脚本文件，然后在 vim 编辑器中使用如下命令查看文件的格式（结果为 fileformat=dos 或 fileformat=unix）：

```
: set ff
```

使用如下命令将文件格式修改为 unix，并保存退出：

```
: set ff=unix
: wq
```

## 实验拓扑

考虑物理电脑的性能问题，本实验仅使用 SDN-Controller 和 PC1 这两台虚拟机，在 PC1 中安装运行 Mininet，实验拓扑如图 11 所示。

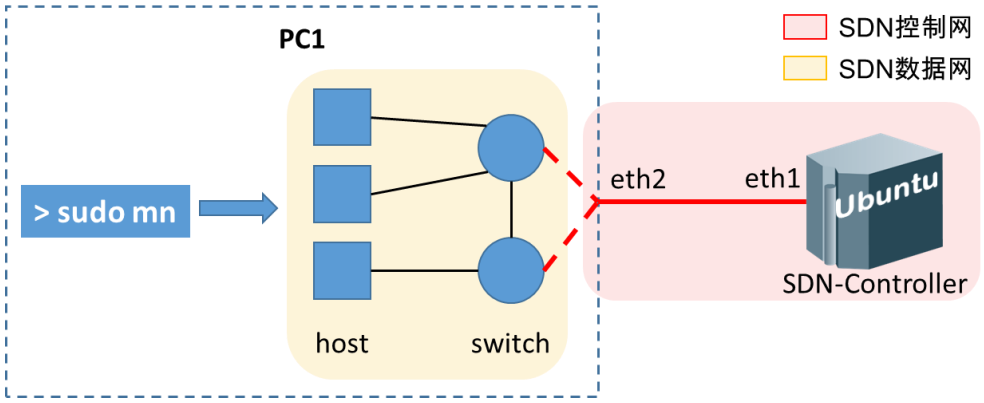



图 11 实验 4 拓扑

为了能让 PC1 中的 Mininet 外接 SDN-Controller 虚拟机上的 ODL 控制器，需要为 PC1 虚拟机添加一个虚拟网卡 3（图中的 eth2）。实验 4 拓扑中各个虚拟机的主机名、网络连接模式和 IP 地址如表 5 所示。

表 5 实验 4 的虚拟机设置

| 虚拟机主机名 | 网卡 | 网络连接模式 | | | 网络接口配置（IP 地址/掩码/GW） |
|---|---|---|---|---|---|
| | | 模式 | VirtualBox | VMware | |
| SDN-Controller | 网卡 1 | NAT | NAT 网络 | VMnet8 | eth0：DHCP（172.16.1.0/24） |
| | 网卡 2 | 内部网络 | intnet1 | LAN1 | eth1：10.1.1.10/24，无 GW |
| PC1 | 网卡 1 | NAT | NAT 网络 | VMnet8 | eth0：DHCP（172.16.1.0/24） |
| | 网卡 3 | 内部网络 | intnet1 | LAN1 | eth1：10.1.1.21/24，无 GW |

## 实验 4.1 Mininet 的安装

本实验将在 PC1 虚拟机上使用源码方式安装 Mininet。

### 步骤 1、获取源代码包

建议切换至 root 用户登录，切换后的目录为/root：

```
$ sudo - root
```

Mininet 的源代码共享在 github 上，安装 Mininet 的操作也与 git 工具密切相关。因此，安装 Mininet 之前需要安装 git 工具（此步骤需要 PC1 虚拟机能够访问因特网）：

```
# apt-get install git
```

安装好 git 工具后，执行如下命令将 Mininet 源代码包从 github 拉取到本地：

```
# git clone git://github.com/mininet/mininet
```

因为执行以上命令的当前目录是/root，所以获取的 Mininet 源代码包存放在/root 目录下。git 拉取的是 Mininet 所有版本代码，本实验安装的 Mininet 版本是 2.2.2，因此在安装前需要使用如下命令检出 Mininet 2.2.2 版本代码：

```
# cd /root/mininet
# git tag && git checkout -b 2.2.2 2.2.2
```

命令中-b 选项后的两个参数分别是版本号和版本起始点。

记录：git tag && git checkout 命令及其结果截图。

**步骤 2、安装 Mininet**

检出安装所需的源代码后，切换到代码所在目录：

```
# cd /root/mininet/util
```

在该目录下执行全部安装的命令：

```
# ./install.sh -a
```

全部安装的时间较长，因为安装过程中还需要从因特网下载一些模块。安装完毕后，/root 目录下除了 mininet 目录外，还会新增 oflops、oftest、openflow 和 pox 这四个目录。

## 实验 4.2 使用 MiniEdit 可视化工具构建 SDN 网络

**实验要求：**

- Mininet 全局设置：主机默认网段为 10.0.0.0/8，勾选“Start CLI”，使用 OpenFlow 1.0 协议；
- 1 个 Mininet 自带控制器，控制器类型为“OpenFlow Reference”；
- 2 个 openflow 交换机，交换机类型为“Open vSwitch Kernel Mode”；
- 4 个主机，IP 地址分别为 10.0.0.1、10.0.0.2、10.0.0.3 和 10.0.0.4；
- 2 个交换机（s1 和 s2）之间有一条链路，每个交换机连接 2 个主机；
- 拓扑运行后，各主机能相互 ping 通。

**步骤 1、启动 MiniEdit**

执行以下命令启动 MiniEdit：

```
# cd /root/mininet/mininet/examples
# ./miniedit.py
```

**步骤 2、在 MiniEdit 界面中构建并运行实验要求的拓扑**

记录：含有所构建拓扑的 MiniEdit 界面截图。

**步骤 3、在 Mininet 命令行界面中查看运行的拓扑信息**

在 Mininet 命令行界面中依次运行以下命令：

```
mininet> nodes
mininet> dump
mininet> net
mininet> links
```

记录：执行以上 4 条命令的结果截图。

**步骤 4、在 Mininet 命令行界面中检验拓扑中主机间的通信**

在 Mininet 命令行界面中依次运行以下命令，查看主机间 ping 通信前的 2 个交换机概要信息和流表信息：

```
mininet> sh ovs-vsctl show
mininet> dpctl dump-flows
```

记录：执行以上 2 条命令的结果截图。

接着在 Mininet 命令行界面中依次运行以下命令，检验主机间的 ping 通信，并查看 2 个交换机的流表信息：

```
mininet> pingall
mininet> dpctl dump-flows
```

记录：执行以上 2 条命令的结果截图。

**说明**：因为本实验中使用的控制器会为这次通信下发较多的流表项，因此 ping 通信结束后，立刻执行 dpctl dump-flows 命令时看到的流表项可能会不完整，建议在 ping 结束后等 20 秒左右再执行 dpctl dump-flows 命令。

**步骤 5、将构建的 SDN 拓扑保存为 python 脚本文件**

选择 MiniEdit 界面“File”菜单中的“Export Level 2 Script”选项，将构建的拓扑保存为 python 脚本文件，文件名建议为 lab4-2.py。

**步骤 6、停止运行拓扑，退出 Mininet**

Mininet 命令行界面中输入“quit”命令关闭 MiniEdit 可视化应用界面且退出 Mininet。

建议使用“mn -c“命令清除实验环境。

## 实验 4.3 使用 mn 命令构建 SDN 网络

实验要求：

- 使用 mn 命令构建实验 4.2 的拓扑；
- 要求外接 SDN-Controller 虚拟机上的 ODL 控制器；
- 交换机类型为 OVSSwitch；
- 每个主机的 ID=MAC 地址；
- 拓扑运行后，各主机能相互 ping 通。

**步骤 1、启动 SDN-Controller 虚拟机上的 ODL 控制器**

使用如下命令启动 SDN-Controller 上的 ODL 控制器：

```
# cd /mnt/distribution-karaf-0.3.0-Lithium
# ./bin/karaf
```

使用如下命令确认 ODL 已监听在默认的 TCP 端口 6633 上：

```
# netstat –anp|grep 6633
```

在 SDN-Controller 虚拟机中打开浏览器访问 ODL 的 Web-UI，网址如下：http://127.0.0.1:8181/index.html。

ODL 的 Web-UI 默认用户名和密码均为 admin。

**说明：如果物理主机的性能允许的话，可以在实验 4.3 和实验 4.4 中的 SDN-Controller 虚拟机上对 eth1 口（SDN 控制网络）进行 tcpdump 抓包并保存为后缀为.pcap 的文件，通过 Wireshark 查看.pcap 文件来帮助理解 ODL 控制器和 OVS 交换机之间 OpenFlow 协议交互。**

**步骤 2、使用 mn 命令创建并运行实验要求的拓扑**

先确保 PC1 虚拟机能 ping 通 SDN-Controller 虚拟机的 eth1 接口 IP 地址（10.1.1.10），即本实验的 SDN 控制网是连通的。

实验 4.2 的拓扑是一个 x=2（2 个交换机）、y=2（每个交换机连接 2 个主机）的线性拓扑，因此可以使用 mn 命令创建并运行。

然后在 SDN-Controller 虚拟机的 ODL Web-UI 界面中刷新查看 Topology 页面。

记录：使用的 mn 命令及其运行截图，以及 SDN-Controller 虚拟机上 ODL 的 Topology 页面截图。

**步骤 3、在 Mininet 命令行界面中查看运行的拓扑信息**

在 Mininet 命令行界面中依次运行以下命令：

```
mininet> nodes
mininet> dump
mininet> net
mininet> links
```

记录：执行以上 4 条命令的结果截图。

**步骤 4、在 Mininet 命令行界面中检验拓扑中主机间的通信**

在 Mininet 命令行界面中依次运行以下命令，查看主机间 ping 通信前的 2 个交换机概要信息和流表信息：

```
mininet> sh ovs-vsctl show
mininet> dpctl dump-flows
```

记录：执行以上 2 条命令的结果截图。

接着在 Mininet 命令行界面中依次运行以下命令，检验主机间的 ping 通信，并查看 2 个交换机的流表信息：

```
mininet> pingall
mininet> dpctl dump-flows
```

记录：执行以上 2 条命令的结果截图。

再次刷新查看 SDN-Controller 虚拟机的 ODL Web-UI 界面中 Topology 页面。

记录：SDN-Controller 虚拟机上 ODL 的 Topology 页面截图。

**步骤 5、停止运行拓扑，退出 Mininet**

Mininet 命令行界面中输入“quit”命令关闭 MiniEdit 可视化应用界面且退出 Mininet。

建议使用“mn -c“命令清除实验环境。

**步骤 6、根据实验记录回答以下问题（Q1）**

**Q1**、概要总结实验 4.2 中使用的 Mininet 自带控制器（OpenFlow Reference）和实验 4.3 中使用的 ODL 控制器在 pingall 通信中发给交换机的流表项有哪些差异。（提示：表项数量和表项的具体内容）

## 实验 4.4 使用 Python 脚本调用 Mininet 构建 SDN 网络

**实验要求：**

解读实验 4.2 中导出保存的 python 脚本文件，修改该文件构建如下拓扑；

- 外接 SDN-Controller 虚拟机上的 ODL 控制器；
- 增加 1 个与 s1、s2 相同类型的交换机 s3；
- 增加 1 个与 h1~h4 相同类型的主机 h5，其 IP 地址为 10.0.0.5；
- 设置主机 h1 的 MAC 地址=h1 的 ID，即 00:00:00:00:00:01；主机 h2 的 MAC 地址=h2 的 ID，即 00:00:00:00:00:02；主机 h3、h4、h5 的 MAC 地址也均等于各自的 ID。
- 增加 3 条链路，分别连接 s1 和 s3、s2 和 s3 以及 h5 和 s3；
- 拓扑运行后，各主机能相互 ping 通。

**步骤 1、修改实验 4.2 中保存的 python 脚本文件**

将实验 4.2 保存的 python 脚本文件另存为一份新文件（建议为 lab4-4.py），然后按照实验要求编辑修改和保存这份新文件。

**步骤 2、运行修改后的 python 脚本**

如果这份 python 脚本文件没有执行权限，则需使用如下命令为其添加执行权限：

```
# chmod +x lab4-4.py
```

确保 PC1 虚拟机能 ping 通 SDN-Controller 虚拟机的 eth1 接口 IP 地址（10.1.1.10），即本实验的 SDN 控制网是连通的。

然后，直接运行修改后的 python 脚本，并在 SDN-Controller 虚拟机的 ODL Web-UI 界面中刷新查看 Topology 页面。

记录：运行 python 脚本的命令及其运行截图，以及 SDN-Controller 虚拟机上 ODL 的 Topology 页面截图。

**步骤 3、在 Mininet 命令行界面中查看运行的拓扑信息**

在 Mininet 命令行界面中依次运行以下命令：

```
mininet> nodes
mininet> dump
mininet> net
mininet> links
```

记录：执行以上 4 条命令的结果截图。

**步骤 4、在 Mininet 命令行界面中检验拓扑中主机间的通信**

在 Mininet 命令行界面中依次运行以下命令，查看主机间 ping 通信前的 2 个交换机概要信息和流表信息：

```
mininet> sh ovs-vsctl show
mininet> dpctl dump-flows
```

记录：执行以上 2 条命令的结果截图。

接着在 Mininet 命令行界面中依次运行以下命令，检验主机间的 ping 通信，并查看 2 个交换机的流表信息：

```
mininet> pingall
mininet> dpctl dump-flows
```

记录：执行以上 2 条命令的结果截图。

再次刷新查看 SDN-Controller 虚拟机的 ODL Web-UI 界面中 Topology 页面。

记录：SDN-Controller 虚拟机上 ODL 的 Topology 页面截图。

**步骤 5、停止运行拓扑，退出 Mininet**

Mininet 命令行界面中输入“quit”命令关闭 MiniEdit 可视化应用界面且退出 Mininet。

建议使用“mn -c“命令清除实验环境。

## 实验报告任务

实验报告使用“课程实验报告模板.doc”，模板中的“二、实验步骤、数据及分析结果”请填写前述实验 4.1、实验 4.2、实验 4.3 和实验 4.4 中要求的“记录”和“问题”。