# SDN学习笔记

说明：

笔记来源主要是《软件定义网络核心原理与应用实践》、SDNLAB、《floodlight源码解读》等。（书籍在参考书籍中）

相关资料地址:

* 参考书籍

链接：http://pan.baidu.com/s/1jItCS30 密码：9a00

* 其他论文资料

链接：http://pan.baidu.com/s/1gfekbOv 密码：cqwn

* Openflow资料

链接：http://pan.baidu.com/s/1mhHFmhM 密码：fqc4

# 实践部分

## Mininet安装

安装环境 ubnutu 16.04

1. 更新软件

|  |
| --- |
| # apt-get update  # apt-get upgrade |

在 home目录下

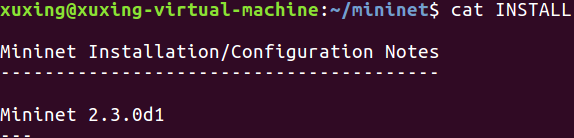
1. 从 github上获取 mininet源码

|  |
| --- |
| #git clone git://github.com/mininet/mininet |

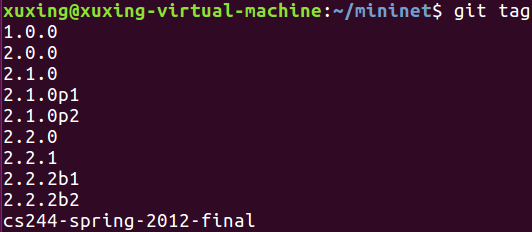
1. 获取完成之后，查看mininet的版本

#cd mininet

#cat INSTALL



说明：在~/mininet目录下，我们可以通过git tag命令列出所有可用的Mininet版本（如下截图），Mininet2.1.0p1及以后的版本可以原生支持OpenFlow1.3，所以我们这次安装的Mininet2.2.1版本支持OpenFlow1.3协议。



1. 源码树获取成功之后，安装mininet

|  |  |
| --- | --- |
| 1 | #mininet/util/install.sh[options] |

这里典型的[options]主要有下面几种：  
“-a”：完整安装包括Mininet VM，还包括如Open vSwitch的依赖关系，以及像的OpenFlow Wireshark分离器和POX。默认情况下，这些工具将被安装在你的home目录中。  
“-nfv”：安装Mininet、基于OpenFlow的交换机和Open vSwitch。  
“-s mydir” ：在其他选项使用前使用此选项可将源代码建立在一个指定的目录中，而不是在你的home目录。

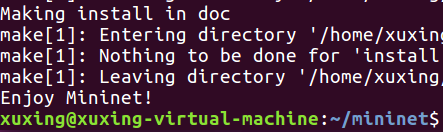
所以，你可能会用到下面的某一个命令来安装Mininet：

|  |
| --- |
| # install.sh –a ##完整安装（默认安装在home目录下）  # install.sh -s mydir –a ##完整安装（安装在其他目录）  # install.sh –nfv ##安装Mininet+用户交换机+OVS（安装在home目录下）  # install.sh -s mydir –nfv ##安装Mininet+用户交换机+OVS（安装在其他目录下） |

这里选择的安装命令是：

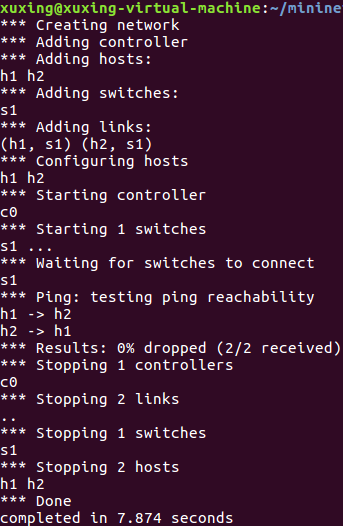
|  |  |
| --- | --- |
| 1 | # mininet/util/install.sh –a |

安装完成



5. 安装完成以后，通过简单的命令测试Mininet的基本功能。

|  |  |
| --- | --- |
| 1 | #sudo mn --test pingall |



1. 查看安装好的mininet 版本

|  |  |
| --- | --- |
| 1 | #mn --version |

参考资料：<http://www.sdnlab.com/15138.html>

## Mininet可视化应用

命令执行启动Mininet可视化界面



参考资料：<http://www.sdnlab.com/15096.html>

## Mininet命令延伸实验扩展

1 实验目的

熟悉Mininet自定义拓扑三种实现方式：命令行创建、Python脚本编写、交互式界面创建。

2 实验原理

Mininet 是一个轻量级软件定义网络和测试平台；它采用轻量级的虚拟化技术使一个单一的系统看起来像一个完整的网络运行想过的内核系统和用户代码，也可简单理解为 SDN 网络系统中的一种基于进程虚拟化平台，它支持 OpenFlow、OpenvSwith 等各种协议，Mininet 也可以模拟一个完整的网络主机、链接和交换机在同一台计算机上且有助于互动开发、测试和演示，尤其是那些使用 OpenFlow 和 SDN 技术；同时也可将此进程虚拟化的平台下代码迁移到真实的环境中。

3 实验任务

本实验分别通过命令行创建、Python脚本编写以及交互式界面创建来熟悉Mininet的基本功能。

4 实验步骤

4.1 搭建环境

本实验需要安装支持OpenFlow1.3协议的Mininet镜像。

4.2 实验操作

步骤1：命令行创建拓扑

1 最小的网络拓扑，一个交换机下挂两个主机。

Shell

|  |  |
| --- | --- |
| 1 | #sudo mn --topo minimal |

2 每个交换机连接一个主机，交换机间相连接。本例：4个主机，4个交换机。

Shell

|  |  |
| --- | --- |
| 1 | # sudo mn --topo linear,4 |

3 每个主机都连接到同一个交换机上。本例：3个主机，一个交换机。

Shell

|  |  |
| --- | --- |
| 1 | # sudo mn --topo single,3 |

4 定义深度和扇出形成基于树的拓扑。本例：深度2，扇出2。

Shell

|  |  |
| --- | --- |
| 1 | # sudo mn --topo tree, fanout=2,depth=2 |

步骤2：Python脚本定义拓扑  
1 --topo linear,4。  
（1）在装有Mininet镜像的虚拟机中新建文件linear.py，添加以下内容：

Shell

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | from mininet.net import Mininet  from mininet.topo import LinearTopo  Linear4 = LinearTopo(k=4)    #四个交换机，分别下挂一个主机  net = Mininet(topo=Linear4)  net.start()  net.pingAll()  net.stop() |

（2）修改文件linear.py为可执行文件。

Shell

|  |  |
| --- | --- |
| 1 | # chmod +x linear.py |

（3）运行脚本。

Shell

|  |  |
| --- | --- |
| 1 | #sudo python linear.py |

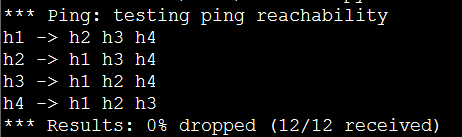
2 --topo single,3。  
这里只给出脚本内容，具体的操作参考步骤2的1。

Shell

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | from mininet.net import Mininet  from mininet.topo import SingleSwitchTopo  Single3 = SingleSwitchTopo(k=3)   #一个交换机下挂3个主机  net = Mininet(topo=Single3)  net.start()  net.pingAll()  net.stop() |

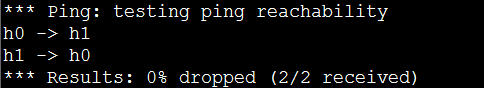
3 --topo tree,depth=2,fanout=2。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | from mininet.net import Mininet  from mininet.topolib import TreeTopo  Tree22 = TreeTopo(depth=2,fanout=2)  net = Mininet(topo=Tree22)  net.start()  net.pingAll()  net.stop() |



4 如果是非上述三种类型的拓扑，那么下面介绍一种适合各种拓扑形式的脚本创建模式。本例：1个交换机，2个主机，并且赋予主机IP地址。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | from mininet.net import Mininet  net = Mininet()  # Creating nodes in the network.  c0 = net.addController()  h0 = net.addHost('h0')  s0 = net.addSwitch('s0')  h1 = net.addHost('h1')  # Creating links between nodes in network  net.addLink(h0, s0)  net.addLink(h1, s0)  # Configuration of IP addresses in interfaces  h0.setIP('192.168.1.1', 24)  h1.setIP('192.168.1.2', 24)  net.start()  net.pingAll()  net.stop() |



5 除了可以通过Python脚本创建基本的拓扑以外，还能在此基础上对性能进行限制。观察下面给出的脚本文件，addHost()语法可以对主机cpu进行设置，以百分数的形式；addLink()语法可以设置带宽bw、延迟delay、最大队列的大小max\_queue\_size、损耗率loss。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | from mininet.net import Mininet  from mininet.node import CPULimitedHost  from mininet.link import TCLink  net = Mininet(host=CPULimitedHost, link=TCLink)  c0 = net.addController()  s0 = net.addSwitch('s0')  h0 = net.addHost('h0')  h1 = net.addHost('h1', cpu=0.5)  h2 = net.addHost('h1', cpu=0.5)  net.addLink(s0, h0, bw=10, delay='5ms',  max\_queue\_size=1000, loss=10, use\_htb=True)  net.addLink(s0, h1)  net.addLink(s0, h2)  net.start()  net.pingAll()  net.stop() |

步骤3：交互式界面创建主机、交换机等

Shell

|  |  |
| --- | --- |
| 1 | #sudo mn |

1 添加主机h3：

2 添加link：

3 给交换机s1添加端口eth3用于连接h3：

4 给h3赋予IP（10.0.0.3）：

5 h1 ping h3

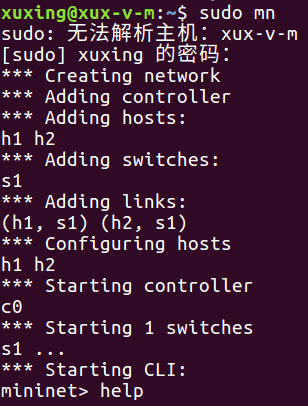
步骤4：测试网络  
1 展示所有的网络信息。

2 所有节点的ping测试。

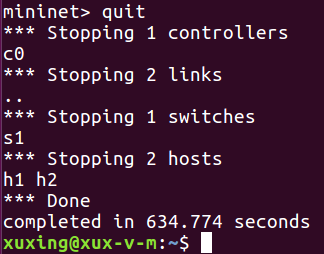
5 实验结论

本次实验对Mininet的命令进行了延伸，通过命令行创建、Python脚本编写以及交互式界面创建进一步熟悉Mininet的基本功能。

## 启动mininet CLI



## 关闭



## Mininet帮助命令

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> xterm h2

## Mininet+POX 与Python API

### SDN 与 Mininet 概述

SDN 全名为（Software Defined Network）即软件定义网络，是现互联网中一种新型的网络创新架构,其核心技术 OpenFlow 通过网络设备控制面与数据面分离开来,从而实现网络流量的灵活控制,为网络及应用提供了良好的平台。而 Mininet 是一个轻量级软件定义网络和测试平台；它采用轻量级的虚拟化技术使一个单一的系统看起来像一个完整的网络运行想过的内核系统和用户代码，也可简单理解为 SDN 网络系统中的一种基于进程虚拟化平台，它支持 OpenFlow、OpenvSwith 等各种协议，Mininet 也可以模拟一个完整的网络主机、链接和交换机在同一台计算机上且有助于互动开发、测试和演示，尤其是那些使用 OpenFlow 和 SDN 技术；同时也可将此进程虚拟化的平台下代码迁移到真实的环境中。

### Mininet 实现的特性

* 支持 OpenFlow、OpenvSwitch 等软定义网路部件
* 支持系统级的还原测试，支持复杂拓扑，自定义拓扑等
* 提供 Python API, 方便多人协作开发
* 很好的硬件移植性与高扩展性
* 支持数千台主机的网络结构

### 基础操作

完全安装之后会在home目录下产生以下几个目录

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

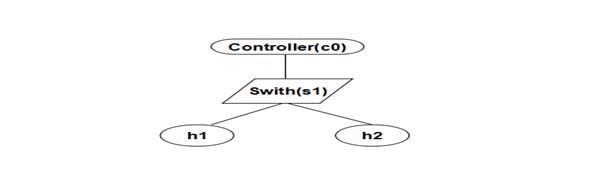
# cd mininet/util/

# ./install.sh -a

# ls

mininet of-dissector oflops oftest openflow pox

#### 默认拓扑



在 Mininet 网络系统中直接输入 mn 命令，可以在此系统中创建单层的拓扑网络，从中默认创建了两台 host 和一个交换机，并且激活了控制器和交换机。

xuxing@xux-v-m:~$ sudo mn 启动默认拓扑

[sudo] xuxing 的密码：

\*\*\* Creating network

\*\*\* Adding controller

mininet> h1 python -m SimpleHTTPServer 80 & 在主机h1开启web服务

mininet> h2 wget -O - h1 在主机h2上下载h1web站点内容

--2017-04-11 19:54:37-- http://10.0.0.1/

正在连接 10.0.0.1:80... 已连接。

已发出 HTTP 请求，正在等待回应... 200 OK

mininet> h1 kill %python #杀掉web进程

Serving HTTP on 0.0.0.0 port 80 ...

10.0.0.2 - - [11/Apr/2017 19:54:37] "GET / HTTP/1.1" 200 -

mininet> h1 ping -c4 h2

PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

64 bytes from 10.0.0.2: icmp\_seq=1 ttl=64 time=8.86 ms

64 bytes from 10.0.0.2: icmp\_seq=2 ttl=64 time=4.57 ms

64 bytes from 10.0.0.2: icmp\_seq=3 ttl=64 time=2.63 ms

64 bytes from 10.0.0.2: icmp\_seq=4 ttl=64 time=0.857 ms

--- 10.0.0.2 ping statistics ---

4 packets transmitted, 4 received, 0% packet loss, time 3015ms

rtt min/avg/max/mdev = 0.857/4.233/8.867/2.981 ms

mininet> nodes #查看节点与链接

available nodes are:

c0 h1 h2 s1

mininet> net

h1 h1-eth0:s1-eth1

h2 h2-eth0:s1-eth2

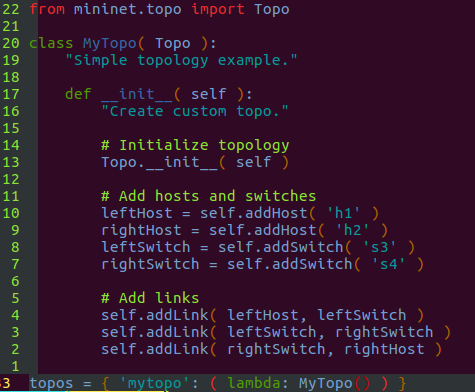
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0

c0

#### 自定义拓扑

Mininet 支持自定义拓扑结构，在 mininet/custom 目录下给出了一个实例，如在 topo-2sw-2host.py 文件中定义了一个 mytopo，则可以通过--topo 选项来指定使用这一拓扑:

代码如下：



##### 导入方式（启动mininet）

当提供自定义mininet文件时，它可以向命令行添加新的拓扑，开关类型和测试。例如：

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

sudo mn --custom~/mininet/custom/test-3sw-4host.py --topo mytopo --mac --switch ovsk--controller=remote --ip=127.0.0.1

#### Mininet 常用操作

**表 1.Mininet 常用指令**

| **名称** | **作用** |
| --- | --- |
| nodes | 查看全部节点 |
| net | 查看链路信息 |
| dump | 输出各节点的信息 |
| h1 ping -c 4 h2 | 测试主机之间的连通性 |
| iperf | 两个节点之间用指定简单的 TCP 测试 |
| iperfudp | 两个节点之间用指定款单 udp 进行测试 |
| noecho | 运行交互窗口，关闭回应 |
| pingpair | 两个主机将互 ping |
| help | 列出命令文档，查看命令帮助： help command |
| dpctl | 在所有叫交换机 |
| exit/quit | 退出 mininet 命令行 |
| hX ifconfig | 查看当前那主机的 ip 地址，如： h1 ifconfig |
| py/sh | 执行 python 表达式或运行外部 shell 程序 |

参考文献：

<https://www.ibm.com/developerworks/cn/cloud/library/1404_luojun_sdnmininet/index.html>

#### 进行回归测试

您不需要掉入CLI; Mininet也可以用于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，启动了OpenFlow参考控制器，运行了全对**ping**测试，并且同时降低了拓扑和控制器。

另一个有用的测试是**iperf**（约10秒钟完成）：

**$ sudo mn --test iperf**

参考文献：

<http://mininet.org/walkthrough/>

### 从本地查询mininet Python API 文档的方法

Mininet网站http://api.mininet.org上也提供了相同的 [文档](http://api.mininet.org/)。

您可能希望自己生成HTML（和PDF）文档doxypy：

sudo apt-get install doxypy

cd ~/mininet

make doc

cd doc

python -m SimpleHTTPServer

此时，您可以将Web浏览器指向运行Mininet主机的端口8000，并浏览每

个Mininet类的文档。

服务器地址就是我的虚拟机IP

在浏览器中输入192.168.1.44:8000 就会读取服务器中存在的文档。

pox.py是程序的入口，需要openflow.of\_01库解释后面的参数。Pox默认开启6633端口监听，也可以在address后面指定port。

### [POX维基](https://openflow.stanford.edu/display/ONL/POX+Wiki)

<https://openflow.stanford.edu/display/ONL/POX+Wiki>

### Neteork automation nerds

<http://blog.pythonicneteng.com/2013/02/openflow-tutorial-with-pox.html?showComment=1474324403969>

### 安装/启动POX、poxdesk

Pox在之前安装mininet的时候 使用 -a选项已经安装了，这里只需要安装poxdesk

1）获取POXDesk

cd pox/ext

git clone https://github.com/MurphyMc/poxdesk

2）获取qooxdoo

下载qooxdoo代码压缩包，然后把解压后文件夹名字改成qx

cd poxdesk

wget http://downloads.sourceforge.net/qooxdoo/qooxdoo-2.0.2-sdk.zip

unzip qooxdoo-2.0.2-sdk.zip

mv qooxdoo-2.0.2-sdk qx

3）初始化poxdesk   
进入poxdesk目录，执行命令./generate.py

cd poxdesk

./generate.py

4）启动POX

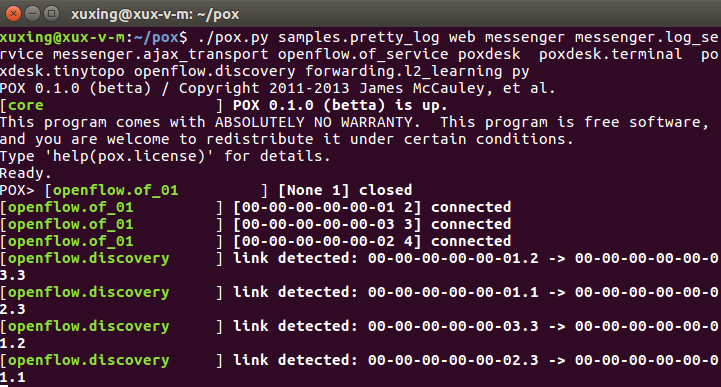
cd ../../..

./pox.py samples.pretty\_log web messenger messenger.log\_service messenger.ajax\_transport openflow.of\_service poxdesk

Samples.pretty.log是一个组件，可以让pox开启的时候有字体有颜色，不添加也可以，但是界面比较难看。

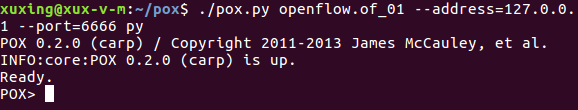
这种启动pox-ui的方式只能启动监听并且开启ui视图，但是无法给switch下发消息（poxdesk的web中的terminal是打算用其他虚拟of-switch的工具来控制下发规则，但是我们是真实的of-switch）   
完整的使用方式如下：

./pox.py samples.pretty\_log web messenger messenger.log\_service messenger.ajax\_transport openflow.of\_service poxdesk openflow.discovery poxdesk.tinytopo py



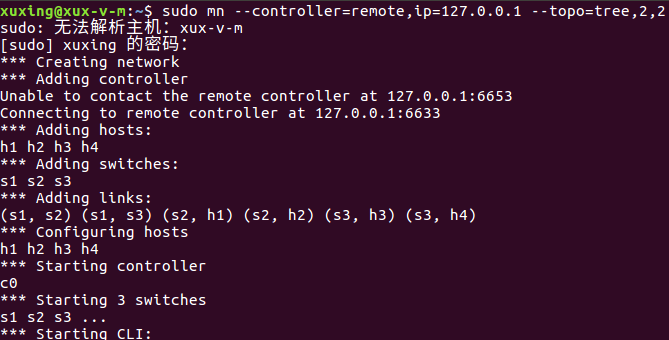
poxdesk.tinytopo可以自动识别topo。上面命令的结尾添加py就可以出现熟悉的pox>命令模式，这时可以在web上看到下联的of-switch.

5）访问Web   
用浏览器访问localhost:8000/poxdesk，默认端口8000   
点击网页左下角的图标pox,可以打开许多小框。



会有POX>输入提示，在这里可以编辑MSG或者调用组件的接口 接下来将Mininet，Pox联合使用，打开三个terminal，一个运行Pox，一个Mininet。Mininet和Pox的运行先后无关，因为Mininet始终在连接指定的controller ip。

#### 启动mininet



安装poxdesk

<http://blog.csdn.net/jk19920523/article/details/40347497>

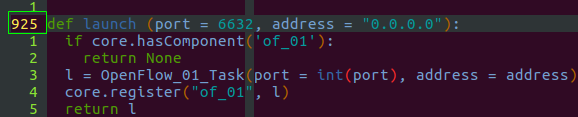
pox操作指南

<http://blog.csdn.net/qq_15437629/article/details/45919417>

<http://blog.csdn.net/jk19920523/article/details/40347497>

#### POX指定port的脚本

位置/home/xuxing/pox/pox/openflow/of\_01.py



#### 如何选择控制器

<https://www.zhihu.com/question/22599089/answer/21923981>

一句话入门选择POX，更推荐Ryu,两个都是python, 用java的话使用floodlight

#### SDN控制器选择须知 10大标准功能清单

<http://net.it168.com/a2013/0717/1508/000001508911.shtml>

#### 快捷测试

（sudo mn --test pingpair sudo mn --test iperf）

xuxing@xux-v-m:~$ sudo mn --test pingpair

\*\*\* 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 ...

\*\*\* Waiting for switches to connect

s1

h1 -> h2

h2 -> h1

\*\*\* Results: 0% dropped (2/2 received)

\*\*\* Stopping 1 controllers

c0

\*\*\* Stopping 2 links

..

\*\*\* Stopping 1 switches

s1

\*\*\* Stopping 2 hosts

h1 h2

\*\*\* Done

completed in 7.988 seconds

xuxing@xux-v-m:~$ sudo mn --test iperf

\*\*\* 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 ...

\*\*\* Waiting for switches to connect

s1

\*\*\* Iperf: testing TCP bandwidth between h1 and h2

\*\*\* Results: ['4.18 Gbits/sec', '4.18 Gbits/sec']

\*\*\* Stopping 1 controllers

c0

\*\*\* Stopping 2 links

..

\*\*\* Stopping 1 switches

s1

\*\*\* Stopping 2 hosts

h1 h2

\*\*\* Done

completed in 12.818 seconds

#### 禁用或启用某条链路

在 Mininet cli 中，使用 link 命令，禁用或启用某条链路，格式为

link node1 node2 up/down

例如临时禁用 s1 跟 h2 之间的链路，可以用

link s1 h2 down

#### 启动参数总结

-h, --help  打印帮助信息

--switch=SWITCH  交换机类型，包括 [kernel user ovsk]

--host=HOST  模拟主机类型，包括 [process]

--controller=CONTROLLER  控制器类型，包括 [nox\_dump none ref remote nox\_pysw]

--topo=TOPO,arg1,arg2,...argN 指定自带拓扑，包括 [tree reversed single linear minimal]

-c, --clean 清理环境

--custom=CUSTOM  使用自定义拓扑和节点参数

--test=TEST  测试命令，包括 [cli build pingall pingpair iperf all iperfudp none]

-x, --xterms  在每个节点上打开 xterm

--mac  让MAC 地址跟 DP ID 相同

--arp  配置所有 ARP 项

-v VERBOSITY, --verbosity=VERBOSITY [info warning critical error debug output] 输出日志级别

--ip=IP  远端控制器的IP地址

--port=PORT  远端控制器监听端口

--innamespace  在独立的名字空间内

--listenport=LISTENPORT  被动监听的起始端口

--nolistenport  不使用被动监听端口

--pre=PRE  测试前运行的 CLI 脚本

--post=POST  测试后运行的 CLI 脚本

#### 常用命令总结

help 默认列出所有命令文档，后面加命令名将介绍该命令用法

dump 打印节点信息

gterm 给定节点上开启 gnome-terminal。注：可能导致 Mininet 崩溃

xterm 给定节点上开启 xterm

intfs 列出所有的网络接口

iperf 两个节点之间进行简单的 iperf TCP测试

iperfudp 两个节点之间用指定带宽 udp 进行测试

net 显示网络链接情况

noecho 运行交互式窗口，关闭回应（echoing）

pingpair 在前两个主机之间互 ping 测试

source 从外部文件中读入命令

dpctl 在所有交换机上用 dptcl 执行相关命令，本地为 tcp 127.0.0.1:6634

link 禁用或启用两个节点之间的链路

nodes 列出所有的节点信息

pingall 所有 host 节点之间互 ping

py 执行 Python 表达式

sh 运行外部 shell 命令

quit/exit 退出

#### 使用 dpctl（NOX）

执行

dpctl show tcp:127.0.0.1:6634

可以查看到交换机的端口等基本情况，其中 tcp 端口 6634 是默认的交换机监听端口。

执行

dpctl dump-flows tcp:127.0.0.1:6634

可以看到更详细的流表信息。 此时，流表为空，执行 h2 ping h3 无法得到响应。因此我们需要通过 dpctl 手动添加流表项，实现转发。 命令为

dpctl add-flow tcp:127.0.0.1:6634 in\_port=1,actions=output:2

dpctl add-flow tcp:127.0.0.1:6634 in\_port=2,actions=output:1

此时查看流表可以看到新的转发信息，同时可以在 h2 和 h3 之间 ping 通。

#### 关闭控制器

先确定没有其他控制器在运行（占据 6633 端口）

sudo killall controller

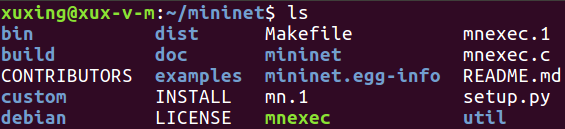
#### 连接控制器

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

Pox修改流表

<http://www.360doc.com/content/13/1220/15/10339652_338691748.shtml>

### Mininet 代码结构



#### 运行相关

bin/mn 主运行文件，安装后执行 mn 即调用的本程序，是 Python 程序。

mnexec.c 执行一些快速命令，比如关闭文件描述符等，是 C 程序，编译后生成二进制文件 mnexec 被 Python 库调用。

#### 安装相关

INSTALL：安装说明

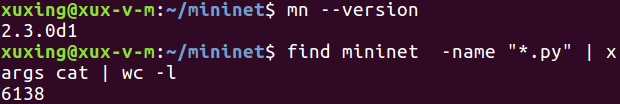
setup.py：安装 Python 包时候的配置文件，被 Makefile 中调用。

debian/：生成 deb 安装包时的配置文件。

#### 核心代码

核心代码基本都在 mininet/ 子目录下。 注：最新的 2.1.0 版本，核心 Python 代码仅为 4675 行。

$ find mininet -name "\*.py" | xargs cat | wc -l



#### 说明文件

CONTRIBUTORS：作者信息

README.md：主说明文件

doc/doxygen.cfg：执行doxygen生成文档时的配置文件。

#### 其它

LICENSE

custom/ 目录下可以放一些用户自定义的 Python 文件，比如自定义的拓扑类等。

test/ 目录下是一些测试的例子。

util/

目录下是一些辅助文件，包括安装脚本、文档辅助生成等，重要的文件包括：

m bash 脚本提供用户直接在 host 执行命令的接口。例如

m host cmd args…

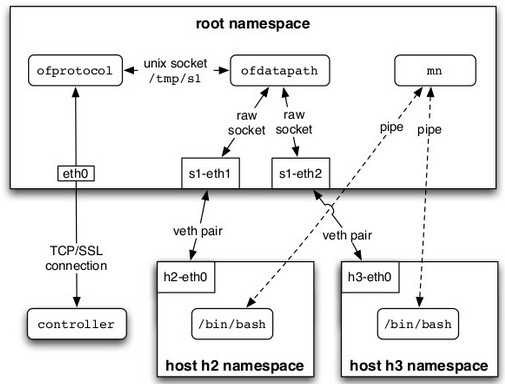
m 通过调用 mnexec 来实现对 Mininet 中的元素执行相应的命令。

mnexec C 程序，通过参数绑定到某个名字空间，并执行给定的命令。

#### 整体逻辑功能

整体上来看，Mininet 作为一个基于 Python 的网络模拟工具，可以分为两大部分：Python库和运行文件。

前者提供对网络中元素进行抽象和实现，例如定义主机类来表示网络中的一台主机。后者则基于这些库来完成各种自定义的模拟过程。 一个典型的场景如下图所示。



#### 示例程序

Mininet 代码中带有了大量的示例程序，供大家参考和理解代码。所有的示例程序都在 example 目录下，包括

baresshd.py:

使用 Mininet 的中层 API 来在一个 namespace 中创建主机、链路，并在主机上启动 sshd 进程，让用户可以登录。并未使用 OpenFlow。

consoles.py:

为每一个节点都创建一些 console 窗口，并允许用户对这些节点进行操作和观测，支持图形界面。

controllers.py:

使用一个自定义的Switch() 子类，创建一个带有多个控制器的网络。

controllers2.py:

创建一个拥有多个控制器的网络，通过创建空的网络，添加节点和手动启动交换机实现。

controlnet.py:

通过创建两个mininet对象来建模一个控制网络和数据网络。

　cpu.py:

在不同的 CPU 限制下测试 iperf 的带宽性能。

emptynet.py:

演示创建一个空的网络，之后添加节点进去。

hwintf.py:

添加一个接口 （例如一个物理接口）到一个网络中。

limit.py:

演示如何使用 link 和 CPU 限制。

linearbandwidth.py:

基于 Topo 创建一个拓扑子类，并进行简单的测试。

miniedit.py:

通过一个图形界面的编辑器来创建网络。

multiping.py:

使用 node.monitor() 来检测多个主机的输出。

multipoll.py:

检测多个主机的输出文件。

multitest.py:

创建一个网络，并在其上进行多个测试。

nat.py:

将 Mininet 的网络通过 nat 连接到外部网络中。

popen.py:

使用 host.popen() 和 pmonitor() 来检测多个主机。

popenpoll.py:

使用 node.popen() 和 pmonitor() 检测多个主机的输出。

scratchnet.py, scratchnetuser.py:

使用底层的 Mininet 函数来创建网络。

simpleperf.py:

配置网络和 CPU、带宽限制等。

sshd.py:

在每个主机里面运行一个 sshd 进程，使得用户可以通过 ssh 来访问主机。这需要将 Mininet 的数据网络连接到 root 名字空间的一个接口上。一般的，控制网络已经在 root 名字空间了，所以默认已经被连接。

tree1024.py:

创建一个 1024 主机的网络，然后运行 CLI。根据系统资源情况，可能需要利用 sysctl 进行相关修改。

treeping64.py:

创建一个 64 主机的树状网络，利用 ping 来检查连通性。

### [Pox组件](http://www.cnblogs.com/yjunzhang/p/pox_component.html)

最近在学习Pox，为了加深印象，对Pox wiki中的Pox组件写了些笔记。

<http://www.cnblogs.com/yjunzhang/p/pox_component.html>

按照组件的功能进行分类：

L2层地址学习、洪泛

forwarding.hub

forwarding.l2\_learning

forwarding.l2\_pairs

forwarding.l2\_multi

forwarding.l2\_nx

L3层地址学习

forwarding.l3\_learning

构建拓扑

openflow.discovery

openflow.spanning\_tree

forwarding.topo\_proactive

openflow 连接相关

openflow.of\_01

misc.full\_payload

openflow.keepalive

pox内部服务

py

web.webcore

messenger

openflow.debug

pox网络服务应用

proto.arp\_responder

proto.pong

proto.dns\_spy

proto.dhcp\_client

proto.dhcpd

misc.nat

misc.ip\_loadbalancer

pox功能扩展

info.packet\_dump

misc.of\_tutorial

misc.mac\_blocker

misc.gephi\_topo

openflow.webservice

原文档 Stanford Pox wiki [https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-ComponentsinPOX](https://openflow.stanford.edu/display/ONL/POX+Wiki" \l "POXWiki-ComponentsinPOX)

组件说明

py

POX的交互式Python解释执行组件，用于DEBUG和交互式实验。默认执行，除非添加命令 --no-cli。其他组件可以向该解释器添加函数和值。

forwarding.hub

该组件每个交换机添加洪泛通配符规则，将所有交换机等效于ethernet集线器

forwarding.l2\_learning

该组件使opennflow交换机实现 L2链路层上的地址学习（类似网桥？）。但当该组件学习地址学习时，向流表下发的规则会尽可能的准确，而不仅仅是L2层的地址。例如不同的TCP连接将产生不同的表项。

forwarding.l2\_pairs

类似于forwarding.l2\_learning，l2\_pairs让交换机进行地址学习，但该组件是尽可能的简化规则学习，所有安装的表项时只使用L2层信息（如Mac地址）。

forwarding.l3\_learning

该组件并不是一个完整的Router，该组件是可POX的packet library（代码）的一个实现样例，可以构造ARP请求和回复。l3\_learning关心IP从哪来，但并不关心IP的填充域，如子网等。

forwarding.l2\_multi

L2层地址学习，但该层的学习不是单个交换机的独立学习，而是通过openflow.discovery交换机之间交换拓扑信息，学习整个网络的拓扑结构。只要网络中有一个交换机学习到一个新的Mac地址及其位置，所有的交换机就都能学会。

forwarding.l2\_nx

Open vSwitch的quick-and-dirty组件，需要使用Openvswitch的Nicira扩展安装。

forwarding.topo\_proactive

基于重要拓扑的IP地址安装规则。通过DHCP进行地址分配。所有的主机都必须用指定的IP地址，绝大部分规则都是主动安装（？）。该组件被添加至聚合规则复用分支中，路由编码基于l2\_multi组件。该组件依赖openflow.discovery以及openflow.spanning\_tree组件（有待确认）。

openflow.spanning\_tree

该组件使用discovery组件来创建网络拓扑的视图，构造一棵生成树，然后使不在生成树中的交换机端口的洪泛功能失效，使得网络中不存在洪泛回路。需要注意的是该组件同生成树协议没有很大关系，只是有相似的目的。两个选项：

--no-flood，只要交换机连接上了就使该交换机的所有端口洪泛失效，对于某些端口，稍后将使能。

--hold-down，防止洪泛控制在一个完整的发现回路完成前被改变

因此该组件最安全的的使用方法是

openflow.spanning\_tree --no-flood --hold-down .

openflow.webservice

Openflow的一个简单JSON-RPC-ish web service交互式接口，由of\_service信息服务派生而来，依赖于webcore组件。可以使用HTTP POST方式发送JSON进行访问。

目前支持的方法有：

method

get\_flow\_stats，获取流表的表项

get\_switch\_desc，获取指定交换机详细信息

get\_switches，获取交换机列表和基本信息

set\_table , 设置指定交换机的流表

web.webcore

在Pox进程中启动一个web服务，其他组件可以通过它提供静态或动态内容。

messenger

该组件通过双向JSON消息为POX在进程间提供了一个交互接口。该组件本质上是API，通过TCP Socket和HTTP进行通信。具体的功能通过Services实现。messenger.log\_service允许远程操作log（读log信息，配置log等）。openflow.of\_service 允许一下Openflow的操作（如显示交换机列表，设置流表表项等）。./tools/pox-log.py是一个独立的Python应用，可以通过TCP同log服务进行交互。

openflow.of\_01

该组件同openflow 1.0协议版本的交换机 进行通讯，默认启动。

openflow.discovery

该组件在交换机之间使用特制的LLDP报文来发现整个网络的拓扑结构。当链路生效或者失效时，该组件都会产生一个事件（Raise Events）。

openflow.debug

加载该组件将导致POX创建pcap追踪（进行抓包），包括openflow报文，可导入wireshark进行分析。该工具并不能完全代替wireshark或tcpdump，不过有一个比较好的特性是每一个openflow报文都一个完整的帧中。

openflow.keepalive

该组件令POX向已经连接的交换机周期性的发送echo请求。但这会解决两个问题：

第一，有些交换机（包括推荐交换机）会认为空闲连接意味着同控制器连接丢失，将会在一段silence时间后断开

连接。

第二，如果网络与交换机断开，控制器将不会立即获得一个FIN或RST，所以将会很难确定一个交换机失效。通过周期行发送echo请求，并分析交换机的响应，即可解决该问题。

proto.pong

该组件是一个简单的检测ICMP echo请求和应答的样例组件

proto.arp\_responder

该组件为一个ARP应用，可以学习和代理ARP请求，也可以通过查询静态的表项来回复ARP请求。该组件提供了一个控制台交互界面来查询和修改arp表。

info.packet\_dump

该组件将packet\_in信息保存至log中，有点类似于在交换机中运行tcpdump

proto.dns\_spy

检测DNS应答并存储应答结果，其他组件可以通过DNSSpy检测这些信息。

proto.dhcp\_client

DHCP客户端，在同其他组件进行联合时有用（？？）

proto.dhcpd

简单的DHCP服务器端，服务器本身的默认地址为192.168.0.254，下发的地址域为192.168.0.1~192.168.0.253，同时宣称自身为网关和DNS服务器。

misc.of\_tutorial

配合openflow tutorial使用的组件，类似于简单的hub，但可以修改成L2 learning的交换机

misc.full\_payload

默认情况下，当一个数据包在交换机流表中没有命中时，交换机只向控制器发送数据包的前128bytes，使用该组件可以将每一个交换机配置成发送整个数据包

misc.mac\_blocker

具有Tkinter-based界面，可以阻塞Mac地址

misc.nat

实现网络地址转换的组件（木有详细介绍）

misc.ip\_loadbalancer

由carp branch（不理解是啥）启用的TCP负载均衡器

misc.gephi\_topo

检测拓扑结构，并将其导入到gephi中进行分析

### 不同交换机不能ping通的问题（pox）

解决方法：

xuxing@xux-v-m:~/pox$ ./pox.py lab\_controller forwarding.l2\_learning forwarding.hub

在运行控制器的时候加上组件 forwarding.hub

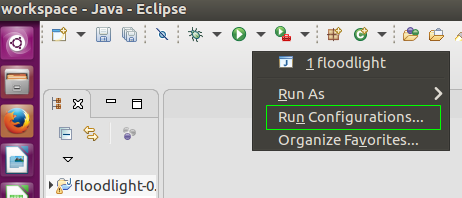
## Mininet+floodlight

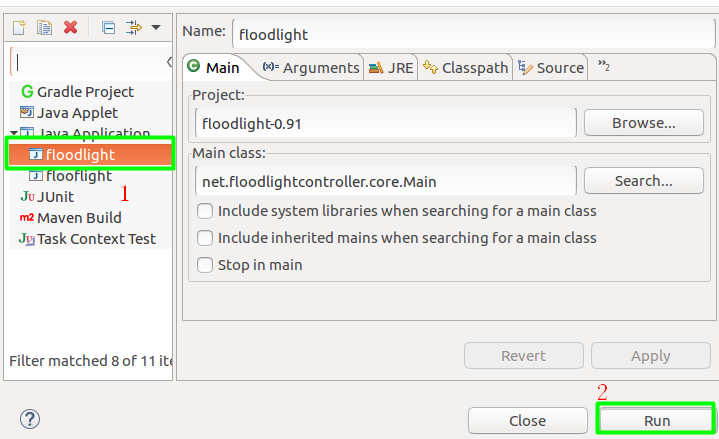
### 搭建环境+启动过程

#### 1.安装jdk+eclipse

<http://www.cnblogs.com/jxldjsn/p/5682390.html>

#### 2.启动控制器





##### 命令启动floodlight

java -jar target/floodlight.jar

#### 3.启动mininet topo

xuxing@xux-v-m:~/mininet$ sudo mn --custom ./custom/topo-2sw-2host.py --topo=mytopo --controller=remote --ip=127.0.0.1

\*\*\* Creating network

\*\*\* Adding controller

Unable to contact the remote controller at 127.0.0.1:6653

Connecting to remote controller at 127.0.0.1:6633

\*\*\* Adding hosts:

h1 h2

\*\*\* Adding switches:

s3 s4

\*\*\* Adding links:

(h1, s3) (s3, s4) (s4, h2)

\*\*\* Configuring hosts

h1 h2

\*\*\* Starting controller

c0

\*\*\* Starting 2 switches

s3 s4 ...

\*\*\* Starting CLI:

mininet> dump

<Host h1: h1-eth0:10.0.0.1 pid=24243>

<Host h2: h2-eth0:10.0.0.2 pid=24246>

<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=24252>

<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None pid=24255>

<RemoteController c0: 127.0.0.1:6633 pid=24235>

mininet>

mininet> pingall

\*\*\* Ping: testing ping reachability

h1 -> h2

h2 -> h1

\*\*\* Results: 0% dropped (2/2 received)

mininet> net

h1 h1-eth0:s3-eth1

h2 h2-eth0:s4-eth2

s3 lo: s3-eth1:h1-eth0 s3-eth2:s4-eth1

s4 lo: s4-eth1:s3-eth2 s4-eth2:h2-eth0

c0

mininet> dpctl dump-flows

\*\*\* s3 ------------------------------------------------------------------------

NXST\_FLOW reply (xid=0x4):

\*\*\* s4 ------------------------------------------------------------------------

NXST\_FLOW reply (xid=0x4):

mininet> pingall

\*\*\* Ping: testing ping reachability

h1 -> h2

h2 -> h1

\*\*\* Results: 0% dropped (2/2 received)

mininet> dpctl dump-flows

\*\*\* s3 ------------------------------------------------------------------------

NXST\_FLOW reply (xid=0x4):

cookie=0x20000000000000, duration=19.833s, table=0, n\_packets=8, n\_bytes=672, idle\_timeout=5, idle\_age=4, priority=0,in\_port=1,vlan\_tci=0x0000,dl\_src=7a:d8:6c:74:8d:63,dl\_dst=6e:5d:08:bb:76:b0 actions=output:2

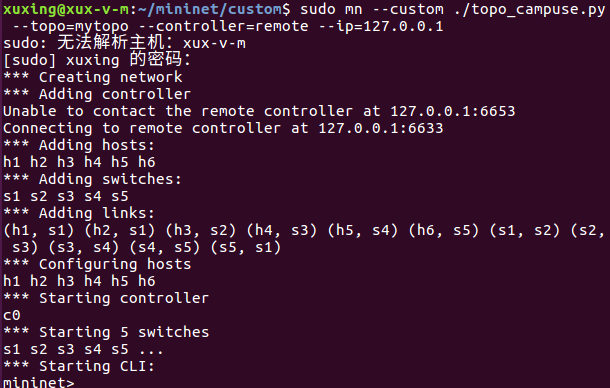
cookie=0x20000000000000, duration=19.810s, table=0, n\_packets=9, n\_bytes=770, idle\_timeout=5, idle\_age=4, priority=0,in\_port=2,vlan\_tci=0x0000,dl\_src=6e:5d:08:bb:76:b0,dl\_dst=7a:d8:6c:74:8d:63 actions=output:1

\*\*\* s4 ------------------------------------------------------------------------

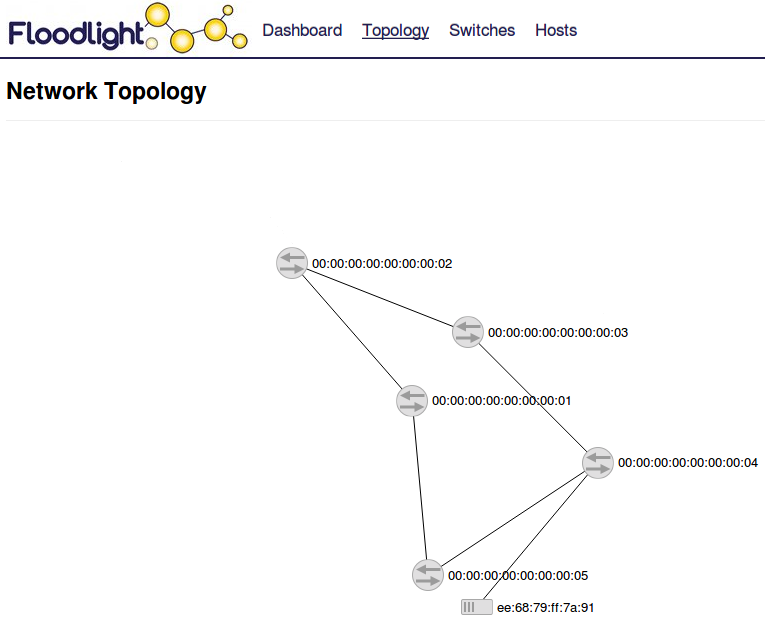
NXST\_FLOW reply (xid=0x4):

cookie=0x20000000000000, duration=20.012s, table=0, n\_packets=8, n\_bytes=672, idle\_timeout=5, idle\_age=4, priority=0,in\_port=1,vlan\_tci=0x0000,dl\_src=7a:d8:6c:74:8d:63,dl\_dst=6e:5d:08:bb:76:b0 actions=output:2

cookie=0x20000000000000, duration=19.990s, table=0, n\_packets=8, n\_bytes=672, idle\_timeout=5, idle\_age=4, priority=0,in\_port=2,vlan\_tci=0x0000,dl\_src=6e:5d:08:bb:76:b0,dl\_dst=7a:d8:6c:74:8d:63 actions=output:1



#### 控制器端查看topo



### Flooglight 流表下发过程简介

<http://blog.csdn.net/xjtuse2014/article/details/51718642>

### [Floodlight 用Static Flow Pusher 改变流表](http://blog.csdn.net/fei_zodiac/article/details/24706313)

<http://blog.csdn.net/fei_zodiac/article/details/24706313>

（重要突破点）

1. 创建topo 1sw\_3hosts



2. 启动mininet

xuxing@xux-v-m:~$ sudo mn --custom ./mininet/custom/topo\_1sw\_3host.py --topo=mytopo --controller=remote --ip=127.0.0.1

1. 启动floodlight 控制器

启动eclipse，运行控制器组件。

1. 创建修改流表的脚本（任意目录），并运行脚本

注意：从windows 复制到linux下的代码需要修改格式，sef ff=unix， 注意在vim中的中文。

import httplib

import json

class StaticFlowPusher(object):

    def \_\_init\_\_(self, server):

        self.server = server

    def get(self, data):

        ret = self.rest\_call({}, 'GET')

        return json.loads(ret[2])

    def set(self, data):

        ret = self.rest\_call(data, 'POST')

        return ret[0] == 200

    def remove(self, objtype, data):

        ret = self.rest\_call(data, 'DELETE')

        return ret[0] == 200

    def rest\_call(self, data, action):

        path = '/wm/staticflowentrypusher/json'

        headers = {

            'Content-type': 'application/json',

            'Accept': 'application/json',

            }

        body = json.dumps(data)

        conn = httplib.HTTPConnection(self.server, 8080)

        conn.request(action, path, body, headers)

        response = conn.getresponse()

        ret = (response.status, response.reason, response.read())

        print ret

        conn.close()

        return ret

pusher = StaticFlowPusher('192.168.131.129')  #控制器ip

flow1 = {

    'switch':"00:00:00:00:00:00:00:01",

    "name":"flow-mod-1",

    "cookie":"0",

    "priority":"32768",

    "ingress-port":"1",

    "active":"true",

    "actions":"output=1"

    }

flow2 = {

    'switch':"00:00:00:00:00:00:00:01",

    "name":"flow-mod-2",

    "cookie":"0",

    "priority":"32768",

    "ingress-port":"2",

    "active":"true",

    "actions":"output=flood"

    }

flow3 = {

    'switch':"00:00:00:00:00:00:00:01",

    "name":"flow-mod-3",

    "cookie":"0",

    "priority":"32768",

    "ingress-port":"3",

    "active":"true",

    "actions":"output=flood"

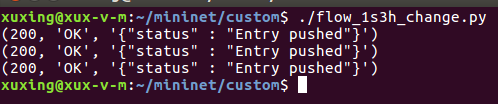
    }

#添加流表flow1，flow2，flow3

pusher.set(flow1)

pusher.set(flow2)

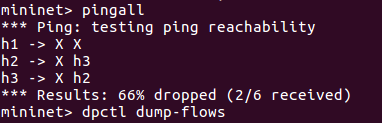
pusher.set(flow3)



说明：

我们使用Static Flow Pusher 改变流表，让h2 ping不通h3,h4。 h3与h4能ping通，转发口使用floodlight控制器，我们设置从端口1的数据从端口1转发回去，也可以在actions域置为空，则丢弃从端口1来的包。

实验结果



#### 流表分析

##### 执行./flow\_1s3h\_change.py之前的流表：

mininet> dpctl dump-flows

\*\*\* s1 ------------------------------------------------------------------------

NXST\_FLOW reply (xid=0x4):

cookie=0x20000000000000,duration=2.671s,table=0, n\_packets=2, n\_bytes=196, idle\_timeout=5, idle\_age=2,priority=0,in\_port=2,vlan\_tci=0x0000,dl\_src=fe:02:e1:af:d5:3e, dl\_dst=de:88:fe:15:9c:31 actions=output:1

cookie=0x20000000000000, duration=2.662s, table=0, n\_packets=1, n\_bytes=98, idle\_timeout=5, idle\_age=2,priority=0,in\_port=1,vlan\_tci=0x0000,dl\_src=de:88:fe:15:9c:31,dl\_dst=fe:02:e1:af:d5:3e actions=output:2

cookie=0x20000000000000,duration=2.614s,table=0, n\_packets=2, n\_bytes=196, idle\_timeout=5, idle\_age=2,priority=0,in\_port=3,vlan\_tci=0x0000,dl\_src=42:f0:97:c5:1b:db,dl\_dst=de:88:fe:15:9c:31 actions=output:1

cookie=0x20000000000000, duration=2.606s, table=0, n\_packets=1, n\_bytes=98, idle\_timeout=5, idle\_age=2,priority=0,in\_port=1,vlan\_tci=0x0000,dl\_src=de:88:fe:15:9c:31,dl\_dst=42:f0:97:c5:1b:db actions=output:3

cookie=0x20000000000000,duration=2.527s,table=0, n\_packets=2, n\_bytes=196, idle\_timeout=5, idle\_age=2,priority=0,in\_port=3,vlan\_tci=0x0000,dl\_src=42:f0:97:c5:1b:db,dl\_dst=fe:02:e1:af:d5:3e actions=output:2

cookie=0x20000000000000, duration=2.522s, table=0, n\_packets=1, n\_bytes=98, idle\_timeout=5, idle\_age=2,priority=0,in\_port=2,vlan\_tci=0x0000,dl\_src=fe:02:e1:af:d5:3e,dl\_dst=42:f0:97:c5:1b:db actions=output:3

##### 执行之后

mininet> pingall

\*\*\* Ping: testing ping reachability

h1 -> X X

h2 -> X h3

h3 -> X h2

\*\*\* Results: 66% dropped (2/6 received)

mininet> dpctl dump-flows

\*\*\* s1 ------------------------------------------------------------------------

NXST\_FLOW reply (xid=0x4):

cookie=0xa000004039d1a6, duration=56.623s, table=0, n\_packets=20, n\_bytes=952, idle\_age=11, in\_port=1 actions=output:1

cookie=0xa000004039d1a7, duration=56.537s, table=0, n\_packets=8, n\_bytes=504, idle\_age=8, in\_port=2 actions=FLOOD

cookie=0xa000004039d1a8, duration=56.493s, table=0, n\_packets=8, n\_bytes=504, idle\_age=8, in\_port=3 actions=FLOOD

## RESTful API 设计指南

<http://www.ruanyifeng.com/blog/2014/05/restful_api.html>

## Floodlight的介绍、安装及使用

<https://wenku.baidu.com/view/1a1a7ebff705cc175427090e.html>

## Floodlight负载均衡

<https://wenku.baidu.com/view/336110690975f46526d3e17f.html>

可以成功实现

## [floodlight路由机制分析](http://blog.csdn.net/tomstrong_369/article/details/42042385)

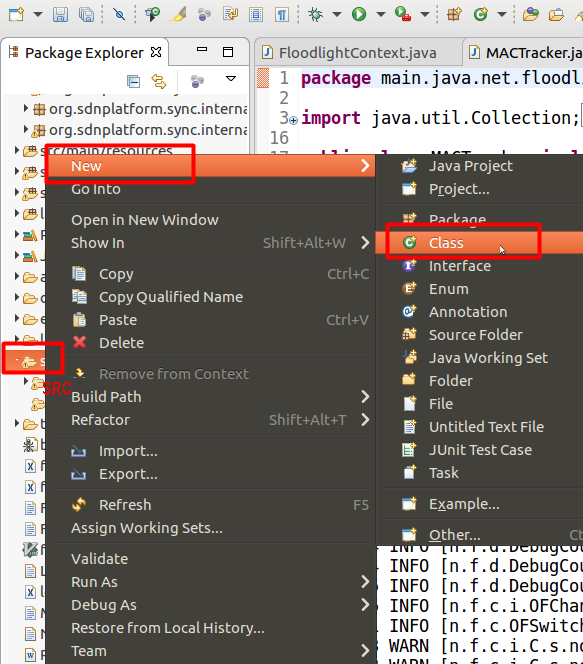
<http://blog.csdn.net/tomstrong_369/article/details/42042385>

## 基于floodlight开发SDN应用

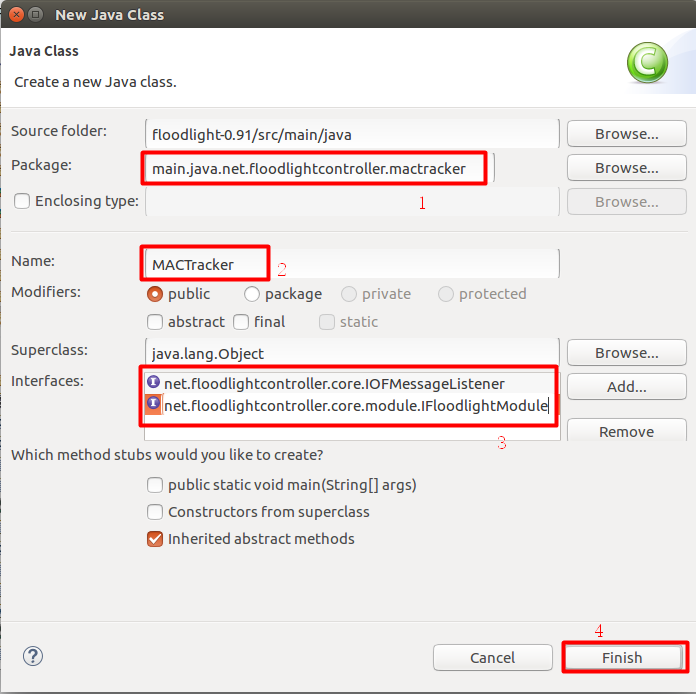
（这里给出一个开发模块实例，它的主要功能是发现并记录新的MAC地址，并将其接入交换机。开发工作在eclipse中完成，在mininet中调试）

在eclipse中相关步骤

1. 在floodlight工程中创建类



创建类的过程中配置如下：



## 常用REST API

### REST API 文档

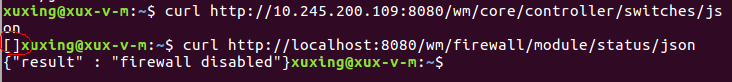
<https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Floodlight+REST+API#FloodlightRESTAPI-FloodlightRESTAPI>

### REST API 的基本用法

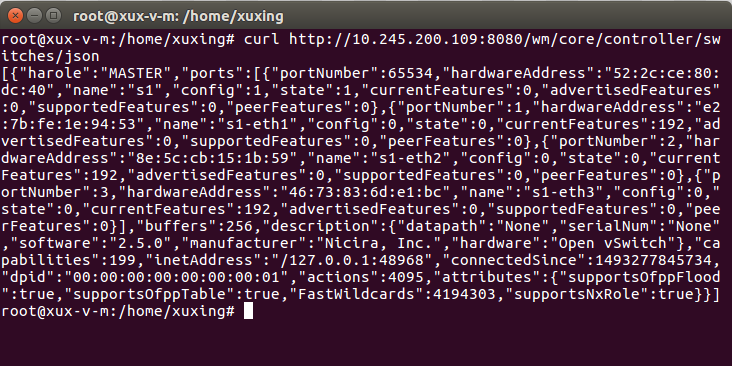
运行控制器，启动mininet之后，新开一个终端，输入一下指令

没有启动mininet之前

xuxing@xux-v-m:~$ curl http://10.245.200.109:8080/wm/core/controller/switches/json



curl <http://10.245.200.109:8080/wm/core/controller/switches/json>



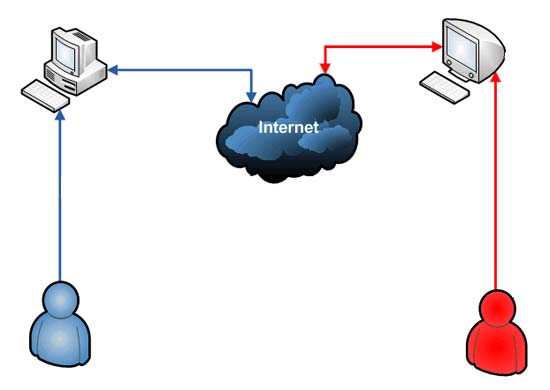
### [理解RESTful架构](http://kb.cnblogs.com/page/114905/)

作者: 阮一峰  发布时间: 2011-09-13 13:35  阅读: 75635 次  推荐: 50   [原文链接](http://www.ruanyifeng.com/blog/2011/09/restful.html" \t "http://kb.cnblogs.com/page/114905/_blank)   [[收藏]](http://kb.cnblogs.com/page/114905/javascript:add_to_wz())

　　越来越多的人开始意识到，**网站即软件**，而且是一种新型的软件。

　　这种"互联网软件"采用客户端/服务器模式，建立在分布式体系上，通过互联网通信，具有高延时（high latency）、高并发等特点。

　　网站开发，完全可以采用软件开发的模式。但是传统上，软件和网络是两个不同的领域，很少有交集；软件开发主要针对单机环境，网络则主要研究系统之间的通信。互联网的兴起，使得这两个领域开始融合，**现在我们必须考虑，如何开发在互联网环境中使用的软件。**



　　RESTful架构，就是目前最流行的一种互联网软件架构。它结构清晰、符合标准、易于理解、扩展方便，所以正得到越来越多网站的采用。

　　但是，到底什么是RESTful架构，并不是一个容易说清楚的问题。下面，我就谈谈我理解的RESTful架构。

**一、起源**

　　REST这个词，是[Roy Thomas Fielding](http://en.wikipedia.org/wiki/Roy_Fielding" \t "http://kb.cnblogs.com/page/114905/_blank)在他2000年的[博士论文](http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm" \t "http://kb.cnblogs.com/page/114905/_blank)中提出的。



　　Fielding是一个非常重要的人，他是HTTP协议（1.0版和1.1版）的主要设计者、Apache服务器软件的作者之一、Apache基金会的第一任主席。所以，他的这篇论文一经发表，就引起了关注，并且立即对互联网开发产生了深远的影响。

　　他这样介绍论文的写作目的：

"本文研究计算机科学两大前沿----软件和网络----的交叉点。长期以来，软件研究主要关注软件设计的分类、设计方法的演化，很少客观地评估不同的设计选择对系统行为的影响。而相反地，网络研究主要关注系统之间通信行为的细节、如何改进特定通信机制的表现，常常忽视了一个事实，那就是改变应用程序的互动风格比改变互动协议，对整体表现有更大的影响。**我这篇文章的写作目的，就是想在符合架构原理的前提下，理解和评估以网络为基础的应用软件的架构设计，得到一个功能强、性能好、适宜通信的架构。**"

(This dissertation explores a junction on the frontiers of two research disciplines in computer science: software and networking. Software research has long been concerned with the categorization of software designs and the development of design methodologies, but has rarely been able to objectively evaluate the impact of various design choices on system behavior. Networking research, in contrast, is focused on the details of generic communication behavior between systems and improving the performance of particular communication techniques, often ignoring the fact that changing the interaction style of an application can have more impact on performance than the communication protocols used for that interaction. My work is motivated by the desire to understand and evaluate the architectural design of network-based application software through principled use of architectural constraints, thereby obtaining the functional, performance, and social properties desired of an architecture. )

**二、名称**

　　Fielding将他对互联网软件的架构原则，定名为REST，即Representational State Transfer的缩写。我对这个词组的翻译是"表现层状态转化"。

　　如果一个架构符合REST原则，就称它为RESTful架构。

**要理解RESTful架构，最好的方法就是去理解Representational State Transfer这个词组到底是什么意思，它的每一个词代表了什么涵义。**如果你把这个名称搞懂了，也就不难体会REST是一种什么样的设计。

**三、资源（Resources）**

　　REST的名称"表现层状态转化"中，省略了主语。"表现层"其实指的是"资源"（Resources）的"表现层"。

**所谓"资源"，就是网络上的一个实体，或者说是网络上的一个具体信息。**它可以是一段文本、一张图片、一首歌曲、一种服务，总之就是一个具体的实体。你可以用一个URI（统一资源定位符）指向它，每种资源对应一个特定的URI。要获取这个资源，访问它的URI就可以，因此URI就成了每一个资源的地址或独一无二的识别符。

　　所谓"上网"，就是与互联网上一系列的"资源"互动，调用它的URI。

**四、表现层（Representation）**

　　"资源"是一种信息实体，它可以有多种外在表现形式。**我们把"资源"具体呈现出来的形式，叫做它的"表现层"（Representation）。**

　　比如，文本可以用txt格式表现，也可以用HTML格式、XML格式、JSON格式表现，甚至可以采用二进制格式；图片可以用JPG格式表现，也可以用PNG格式表现。

　　URI只代表资源的实体，不代表它的形式。严格地说，有些网址最后的".html"后缀名是不必要的，因为这个后缀名表示格式，属于"表现层"范畴，而URI应该只代表"资源"的位置。它的具体表现形式，应该在HTTP请求的头信息中用Accept和Content-Type字段指定，这两个字段才是对"表现层"的描述。

**五、状态转化（State Transfer）**

　　访问一个网站，就代表了客户端和服务器的一个互动过程。在这个过程中，势必涉及到数据和状态的变化。

　　互联网通信协议HTTP协议，是一个无状态协议。这意味着，所有的状态都保存在服务器端。因此，**如果客户端想要操作服务器，必须通过某种手段，让服务器端发生"状态转化"（State Transfer）。而这种转化是建立在表现层之上的，所以就是"表现层状态转化"。**

　　客户端用到的手段，只能是HTTP协议。具体来说，就是HTTP协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。它们分别对应四种基本操作：**GET用来获取资源，POST用来新建资源（也可以用于更新资源），PUT用来更新资源，DELETE用来删除资源。**

**六、综述**

　　综合上面的解释，我们总结一下什么是RESTful架构：

　　（1）每一个URI代表一种资源；

　　（2）客户端和服务器之间，传递这种资源的某种表现层（Representation）；

　　（3）客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"。

**七、误区**

　　RESTful架构有一些典型的设计误区。

**最常见的一种设计错误，就是URI包含动词。**因为"资源"表示一种实体，所以应该是名词，URI不应该有动词，动词应该放在HTTP协议中。

　　举例来说，某个URI是/posts/show/1，其中show是动词，这个URI就设计错了，正确的写法应该是/posts/1，然后用GET方法表示show。

　　如果某些动作是HTTP动词表示不了的，你就应该把动作做成一种资源。比如网上汇款，从账户1向账户2汇款500元，错误的URI是：

　　POST /accounts/1/transfer/500/to/2

　　正确的写法是把动词transfer改成名词transaction，资源不能是动词，但是可以是一种服务：

　　POST /transaction HTTP/1.1  
　　Host: 127.0.0.1  
　　  
　　from=1&to=2&amount=500.00

**另一个设计误区，就是在URI中加入版本号**：

　　http://www.example.com/app/1.0/foo

　　http://www.example.com/app/1.1/foo

　　http://www.example.com/app/2.0/foo

　　因为不同的版本，可以理解成同一种资源的不同表现形式，所以应该采用同一个URI。版本号可以在HTTP请求头信息的Accept字段中进行区分（参见[Versioning REST Services](http://www.informit.com/articles/article.aspx?p=1566460" \t "http://kb.cnblogs.com/page/114905/_blank)）：

　　Accept: vnd.example-com.foo+json; version=1.0

　　Accept: vnd.example-com.foo+json; version=1.1

　　Accept: vnd.example-com.foo+json; version=2.0

### 有关REST的论文

<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>  
链接：https://www.zhihu.com/question/28557115/answer/48094438

### [学习 The Floodlight REST API](http://blog.csdn.net/minword/article/details/21221417)

<http://blog.csdn.net/minword/article/details/21221417>

介绍了不同功能的REST API

上一个连接中打开的网站（需要翻墙）<http://docs.projectfloodlight.org/>

### [如何编写Floodlight REST 应用](http://blog.csdn.net/jackieleewelas/article/details/50419880)

<http://blog.csdn.net/jackieleewelas/article/details/50419880>

可以用任何你喜欢的编程语言编写REST应用，参照以下步骤：

1、  确定需求，也就是你编写的REST应用需要哪些网络服务和信息。

2、  检查REST API，看看是否有提供你所需的服务。

1. 如果有，了解其RESTAPI的语法，输入的参数以及可得的选项，这样就可以直接拿来用。
2. 如果没有，也可能是你所需的网络服务和资源信息没有提供REST API，但却可以在floodlight模块中可获得这些信息，只是没通过API暴露出来。这种情况，你可以自己实现REST API来提供你所需的服务。

3) 如果既没有REST API，又在floodlight中找不到，那你可以自己开发floodlight [Java](http://lib.csdn.net/base/javase" \t "http://blog.csdn.net/jackieleewelas/article/details/_blank" \o "Java SE知识库)模块，并且实现自定义的模块的REST API来提供所需的服务。

3、  用所有你需要的REST API方法，设计以及组成你的应用。

4、  [测试](http://lib.csdn.net/base/softwaretest" \t "http://blog.csdn.net/jackieleewelas/article/details/_blank" \o "软件测试知识库)你的应用并且反馈给floodlight。

下面通过在floodlight/apps目录下的 [Python](http://lib.csdn.net/base/python" \t "http://blog.csdn.net/jackieleewelas/article/details/_blank" \o "Python知识库) Circuit Pusher应用说明。

Curcuit Pusher例子给我们展示了如何创建一个在OpenFlow集群中的两个有IP的主机A和B之间的静态单路径线路。

下面是设计方法：

1、  确定所需的网络服务和信息：

  主机A和B的接触点，即用（交换机ID，端口）表示的数据实体，代表A和B的物理位置。

  A和B之间接触点的路由，即从A经过哪个交换机和哪个端口到达B的路径

  在A和B路由上所有交换机安装流量线路的服务

2、  从RESTAPI中查到的可提供的信息：

  从/wm/device/的GET参数获取设备的接触点信息，比如IP地址

  从/wm/topology/route/<switchIdA>/<portA>/<switchIdB>/<portB>/json可以获取A和B接触点之间的路由信息

  用/wm/staticflowentrypusher/json的POST方法给指定的交换机安装流表项

3、  应用设计：

  语言使用Python

  使用os.popen方法发送curl 命令来调用REST API的方法(应该还可以使用os.system)

  熟悉 /wm/device语法特点，然后在命令返回的结果中解析出A和B接触点的交换机

  熟悉 /wm/topology/route的语法，获取交换机和端口用来下发流表项

  对于每个交换机和端口对，可以通过/wm/staticflowentrypusher/json下发流表

### Floodlight REST API 文档

<https://wenku.baidu.com/view/fd53bd4aa58da0116d174909.html>

### Firewall REST API实验

<http://www.sdnlab.com/firewall-rest-api/>

### RESTFUL API 设计开发(ppt)

<https://wenku.baidu.com/view/277d37d484254b35eefd347d.html>

### 传统校园网拓扑示例

<https://wenku.baidu.com/view/ea96a126dd36a32d73758135.html>

### [Floodlight Core RestAPI - part1](http://hwchiu.logdown.com/posts/161792-floodlight-core-restapi)

<http://hwchiu.logdown.com/posts/161792-floodlight-core-restapi>

### 使用curl调用Rest API

30 DECEMBER 2014 on [shell](https://www.yvanhom.com/tag/shell/)

最近在使用FloodLight，接触了不少Rest API的使用，参照官网所说，使用curl来简单调用，查看需要的信息。

下边是我整理的脚本，使用curl调用Rest API，并用python美化输出结果：

#!/bin/bash

CONTROLLER=192.168.56.1

if [ "$2" != "" ]; then

ARGS="-d ${2}"fi

if [ "$3" != "" ]; then

ARGS="$ARGS -X $3"fi

#echo $ARGS

# curl使用-s关闭统计信息输出，使用python美化json输出#curl -s $ARGS http://${CONTROLLER}:8080/${1}

curl -s $ARGS http://${CONTROLLER}:8080/${1} | python -m json.tool

echo

使用格式如下：

./rest.sh RestURL [json参数] [GET/DELETE]

例子如下：

#### 例子1：

./rest.sh wm/core/memory/json

shell解析成：

curl -s http://192.168.56.1:8080/wm/core/memory/json | python -m json.tool

输出：

{

"free": 208824192,

"total": 334495744}

#### 例子2：带参数

./rest.sh wm/staticflowentrypusher/json '{"switch":"00:00:00:00:00:00:00:01","name":"flow-mod-1","priority":"32768","ingress-port":"1","active":"true","actions":"output=2"}'

输出：

{

"status": "Entry pushed"}

#### 例子3：使用DELETE

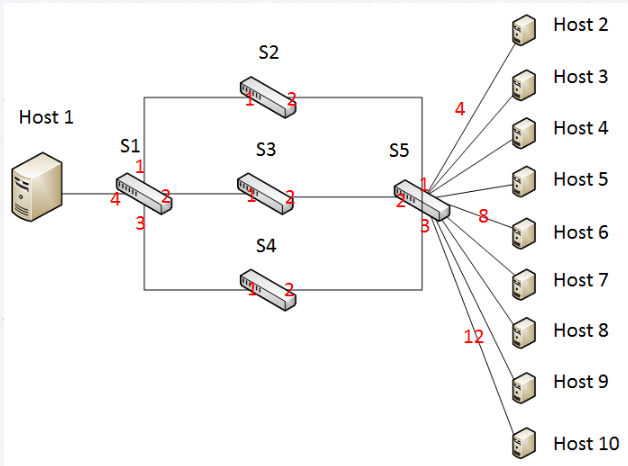
./rest.sh wm/staticflowentrypusher/json '{"name":"flow-mod-1"}' DELETE

输出：

{

"status": "Entry flow-mod-1 deleted"}

使用下列的圖作為我們的網路環境，在此圖中。S1~S5都是支援OpenFlow 1.3的OpenFlow switch，左邊的Host 1則是一個Sender，會對於右邊的九個Host發送資料



from mininet.topo import Topo

class MininetTopo(Topo):

def \_\_init\_\_(self,\*\*opts):

Topo.\_\_init\_\_(self, \*\*opts)

host1 = self.addHost('h1')

host2 = self.addHost('h2')

host3 = self.addHost('h3')

host4 = self.addHost('h4')

host5 = self.addHost('h5')

host6 = self.addHost('h6')

host7 = self.addHost('h7')

host8 = self.addHost('h8')

host9 = self.addHost('h9')

host10 = self.addHost('h10')

self.switch = {}

for s in range(1,6):

self.switch[s-1] = self.addSwitch('s%s' %(s))

self.addLink(self.switch[0], self.switch[1])

self.addLink(self.switch[0], self.switch[2])

self.addLink(self.switch[0], self.switch[3])

self.addLink(self.switch[4], self.switch[1])

self.addLink(self.switch[4], self.switch[2])

self.addLink(self.switch[4], self.switch[3])

#Adding host

self.addLink(self.switch[0], host1)

self.addLink(self.switch[4], host2)

self.addLink(self.switch[4], host3)

self.addLink(self.switch[4], host4)

self.addLink(self.switch[4], host5)

self.addLink(self.switch[4], host6)

self.addLink(self.switch[4], host7)

self.addLink(self.switch[4], host8)

self.addLink(self.switch[4], host9)

self.addLink(self.switch[4], host10)

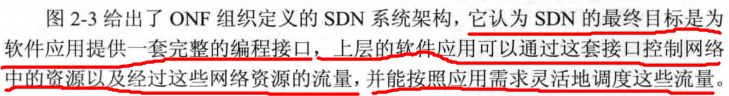
topos = {'group':(lambda:MininetTopo())}

### [Multipath routing with Group table at mininet](http://hwchiu.logdown.com/posts/207387-multipath-routing-with-group-table-at-mininet)

<http://hwchiu.logdown.com/posts/207387-multipath-routing-with-group-table-at-mininet>

# 理论部分

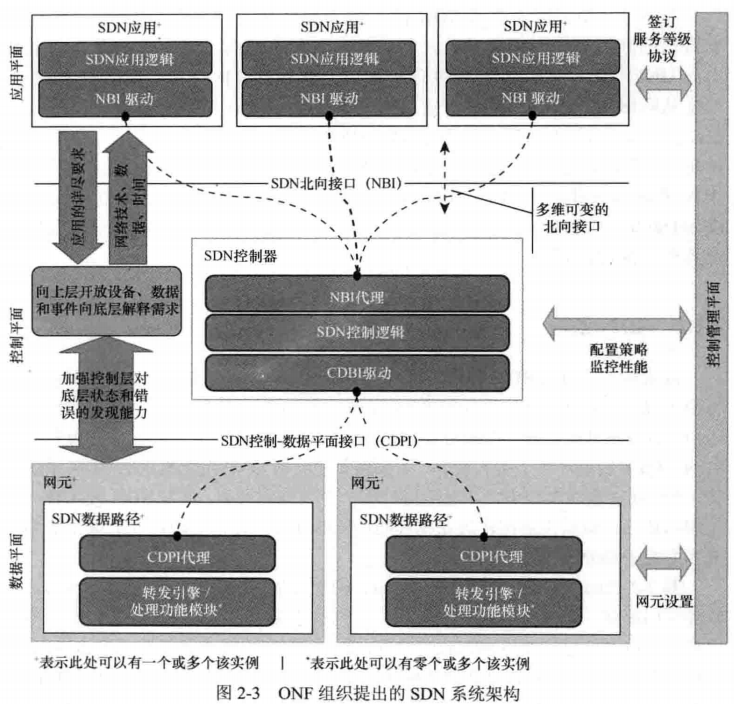
## SDN的最终目标



Page 11

## ONF定义的SDN架构分为几个平面？

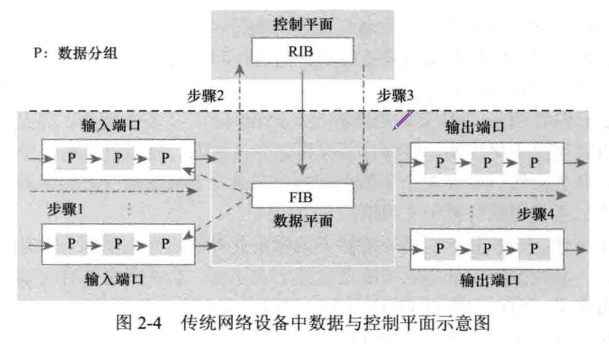
数据平面、控制平面、应用平面、控制管理平面



## SDN技术的特点？

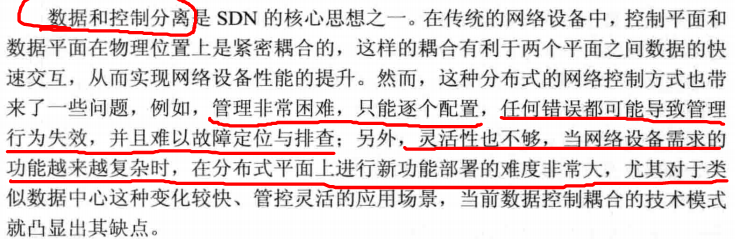
1. 数据控制分离以实现逻辑集中式控制
2. 网络具有可编程性以实现灵活可定义。

## 传统网络控制与数据平面

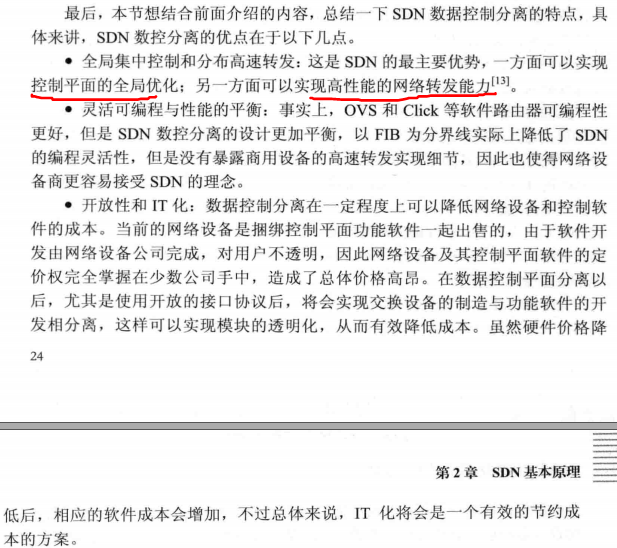


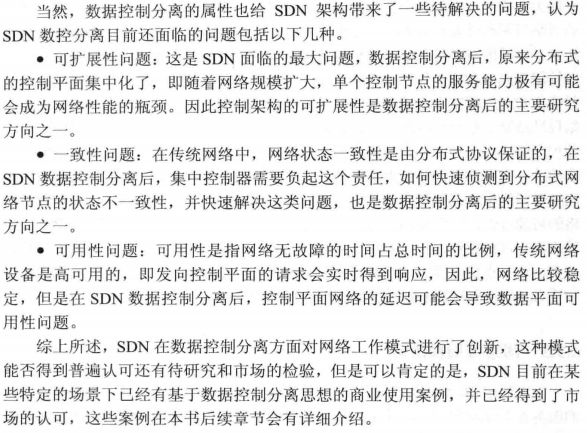
Page 16

## 传统网络的特点



## SDN数据控制分离的优缺点





Page 25

## OF-CONFIG 协议与 openflow的关系？

如前文所述，OpenFlow定义的是SDN网络架构中的一种南向接口，提出了由控制器向OpenFlow交换机发送流表以控制数据流通过网络所经过的路径的方式，但是并没有规定怎样管理和配置这些网络设备，而OF-CONFIG就是为解决这一问题而提出的。

OF-CONFIG的本质是提供一个开放接口用于远程配置和控制OpenFlow交换机，但是它并不会影响到流表的内容和数据转发行为，对实时性也没有太高的要求。具体地说，诸如构建流表和确定数据流走向等事项将由OpenFlow规范进行规定，而诸如如何在OpenFlow交换机上配置控制器IP地址、如何对交换机的各个端口进行enable/disable操作则由OF-CONFIG协议完成。

OpenFlow交换机上所有参与数据转发的软硬件(例如端口、队列等)都可被视为网络资源，而OF-CONFIG的作用就是对这些资源进行管理。

