



西南民族大学
Southwest University for Nationalities

第三届 全国高校 SDN 网络 应用创新开发大赛 报告书

团队名称：网麟

指导老师：陈曦

目录

第 1 题：搭建基于 SDN 的实验网络	2
1.1 SDN 网络环境的搭建	2
1.2 构建网络拓扑图	7
1.3 在终端进行两两相互 Ping	10
第 2 小题：初步分析基于 OpenFlow 的 SDN 网络的控制功能	11
2.1 下发流表实现控制	11
2.2 Openflow 协议报文结构分析及其作用	16
2.3 传统分布式网络与 SDN	18
第 3 题：简单应用的开发与实现——Web 访问用户控制	20
3.1 Web 服务器搭建和页面测试	20
3.2 拓扑视图	25
3.3 北向 API	26
3.4 访问限制测试	27
第 4 题：简单的路由控制应用开发	31
4.1 部署网络环境	31
4.2 北向 API	33
4.3 应用程序和验证程序	33
设计题	42
设计名称	42
具体问题描述	42
研究现状和意义	43
解决方案和实现思路	44
（一）整体思路	44
（二）解决方案	44
（三）抓包和控制器模块示例	45
（四）实现代码	47
参考文献	50

第 1 题：搭建基于 SDN 的实验网络

搭建一个基于 SDN 架构的网络环境,简要说明实验网络环境的搭建思路以及其中所到的软硬件的作用。给出搭建出来的结构拓扑图,可查看网络的拓扑视图信息视图。并且南向接口采用 OpenFlow 协议, H1、H2、H3、H4 任意两两可互通。在实验中给出具体的操作步骤和实验数据。

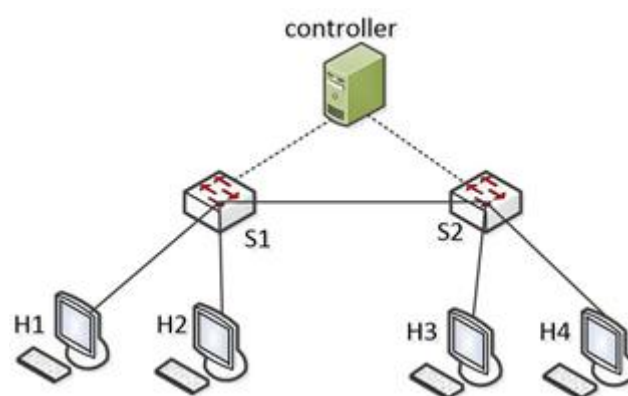


图 1- 1

1.1 SDN 网络环境的搭建

本题采用 Ubuntu+Floodlight+Mininet 部署 SDN 网络环境。Mininet 是基于 LinuxContainer 构架开发的虚拟化平台, 我们通过 Mininet 来创建实验所需的网络拓扑。Floodlight 是基于 Java 的开源控制器, 提供用于实现 SDN 核心网络服务的控制器模块和针对不同业务应用实现解决方案的应用模块。实验中 Mininet 和 Floodlight 都运行在装有 Ubuntu 的真机上。后面的所有实验都是基于此网络环境。网络体系结构如图 1-2。

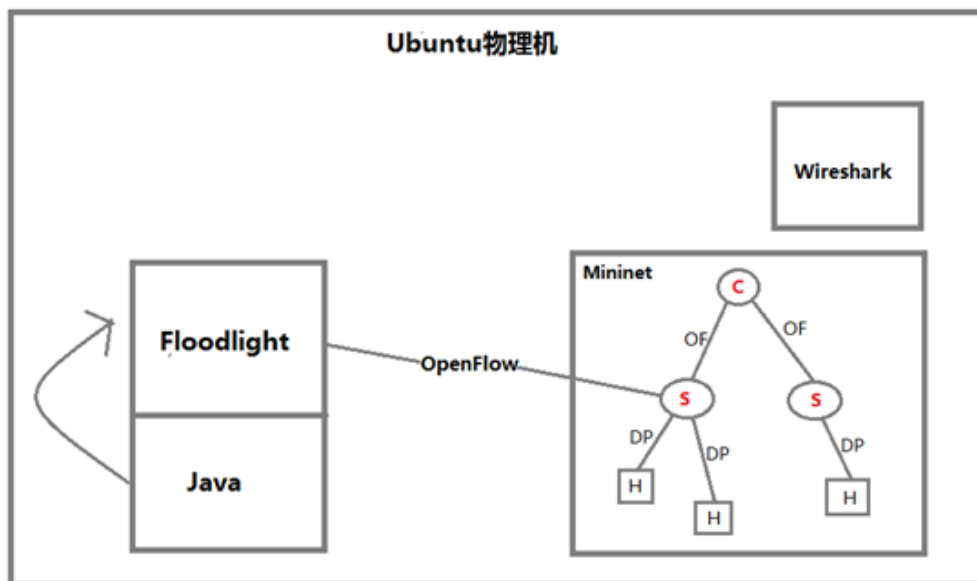


图 1-2 网络体系结构

下面详细描述部署网络环境过程：

1) 安装 OpenJDK

打开终端输入 `sudo apt-get install openjdk-8-jdk`

```

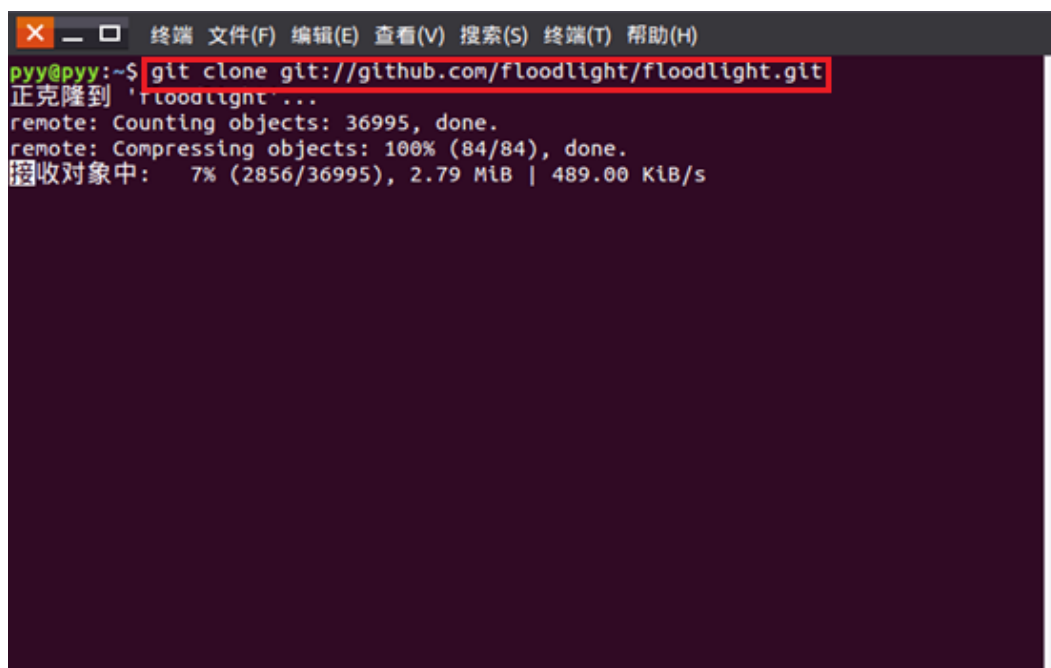
x  _  终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
Total time: 0 seconds
pvy@pvy:~/floodlight$ sudo apt-get install openjdk-8-jdk
[sudo] pvy 的密码:
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
下列软件包是自动安装的并且现在不需要了:
  linux-image-generic
使用 'apt-get autoremove' 来卸载它(它们)。
将会安装下列额外的软件包:
  libgconf2-4 openjdk-8-jre openjdk-8-jre-headless
建议安装的软件包:
  openjdk-8-demo openjdk-8-source visualvm icedtea-8-plugin
  openjdk-8-jre-jamvm fonts-ipafont-gothic fonts-ipafont-mincho fonts-indic
下列【新】软件包将被安装:
  libgconf2-4 openjdk-8-jdk openjdk-8-jre openjdk-8-jre-headless
升级了 0 个软件包，新安装了 4 个软件包，要卸载 0 个软件包，有 0 个软件包未被升级。
需要下载 35.5 MB 的软件包。
解压缩后会消耗掉 139 MB 的额外空间。
您希望继续执行吗？ [Y/n]
获取：1 http://cn.archive.ubuntu.com/ubuntu/ wily-updates/universe openjdk-8-jre
-headless amd64 8u66-b17-1 [26.9 MB]
获取：2 http://cn.archive.ubuntu.com/ubuntu/ wily/universe libgconf2-4 amd64 3.2

```

图 1-3 安装 openJDK

2) 安装和编译 Floodlight

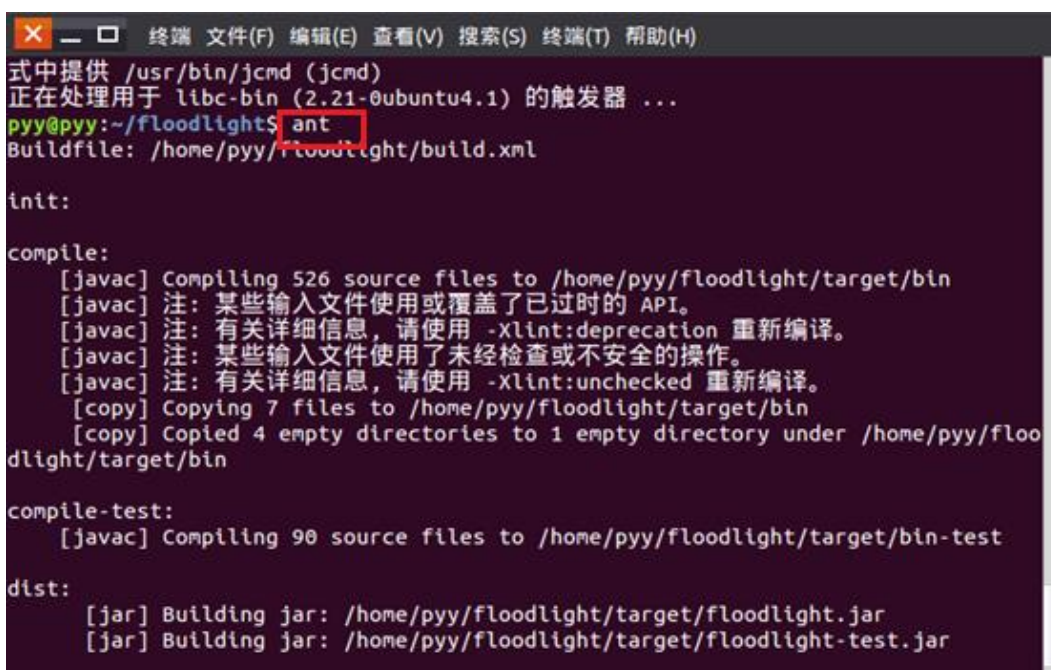
在终端输入 `git clone github.com/floodlight/floodlight.git`



```
pyy@pyy:~$ git clone git://github.com/floodlight/floodlight.git
正克隆到 'floodlight'...
remote: Counting objects: 36995, done.
remote: Compressing objects: 100% (84/84), done.
接收对象中: 7% (2856/36995), 2.79 MiB | 489.00 KiB/s
```

图 1-4 下载 floodlight

下载完成后，进入 floodlight 文件夹，打开终端输入 `ant` 进行编译。



```
pyy@pyy:~/floodlight$ ant
Buildfile: /home/pyy/floodlight/build.xml

init:

compile:
[javac] Compiling 526 source files to /home/pyy/floodlight/target/bin
[javac] 注: 某些输入文件使用或覆盖了已过时的 API。
[javac] 注: 有关详细信息, 请使用 -Xlint:deprecation 重新编译。
[javac] 注: 某些输入文件使用了未经检查或不安全的操作。
[javac] 注: 有关详细信息, 请使用 -Xlint:unchecked 重新编译。
[copy] Copying 7 files to /home/pyy/floodlight/target/bin
[copy] Copied 4 empty directories to 1 empty directory under /home/pyy/floodlight/target/bin

compile-test:
[javac] Compiling 90 source files to /home/pyy/floodlight/target/bin-test

dist:
[jar] Building jar: /home/pyy/floodlight/target/floodlight.jar
[jar] Building jar: /home/pyy/floodlight/target/floodlight-test.jar
```

图 1-5 编译 floodlight

如图 1-6 显示的 BUILD SUCCESSFUL 则编译成功。

```
Buildfile: /home/pyy/floodlight/build.xml

init:

compile:
[javac] Compiling 526 source files to /home/pyy/floodlight/target/bin
[javac] 注: 某些输入文件使用或覆盖了已过时的 API。
[javac] 注: 有关详细信息, 请使用 -Xlint:deprecation 重新编译。
[javac] 注: 某些输入文件使用了未经检查或不安全的操作。
[javac] 注: 有关详细信息, 请使用 -Xlint:unchecked 重新编译。
[copy] Copying 7 files to /home/pyy/floodlight/target/bin
[copy] Copied 4 empty directories to 1 empty directory under /home/pyy/floodlight/target/bin

compile-test:
[javac] Compiling 90 source files to /home/pyy/floodlight/target/bin-test

dist:
[jar] Building jar: /home/pyy/floodlight/target/floodlight.jar
[jar] Building jar: /home/pyy/floodlight/target/floodlight-test.jar

BUILD SUCCESSFUL
total time: 26 seconds
pyy@pyy:~/floodlight$
```

图 1-6 编译 Floodlight 成功

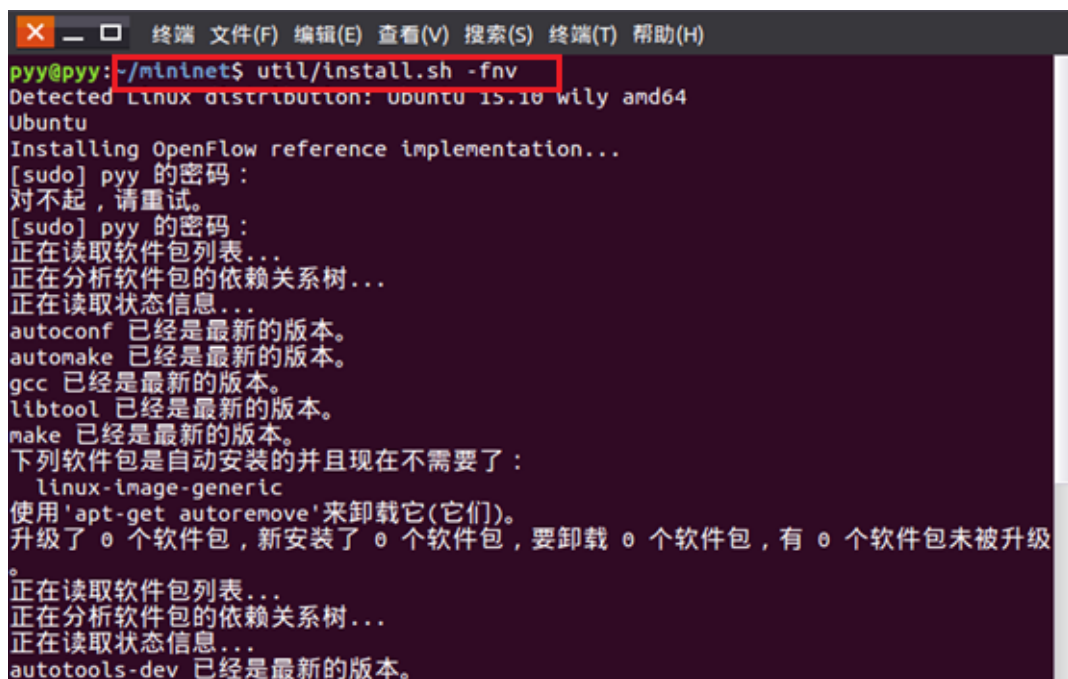
3) 下载和安装 Mininet

在终端输入 `git clone git://github.com/mininet/mininet.git` 进行下载。

```
pyy@pyy:~$ git clone git://github.com/mininet/mininet.git
正克隆到 'mininet'...
remote: Counting objects: 8575, done.
remote: Total 8575 (delta 0), reused 0 (delta 0), pack-reused 8575
接收对象中: 100% (8575/8575), 2.67 MiB | 402.00 KiB/s, 完成。
处理 delta 中: 100% (5579/5579), 完成。
检查连接... 完成。
pyy@pyy:~$
```

图 1-7 下载 Mininet

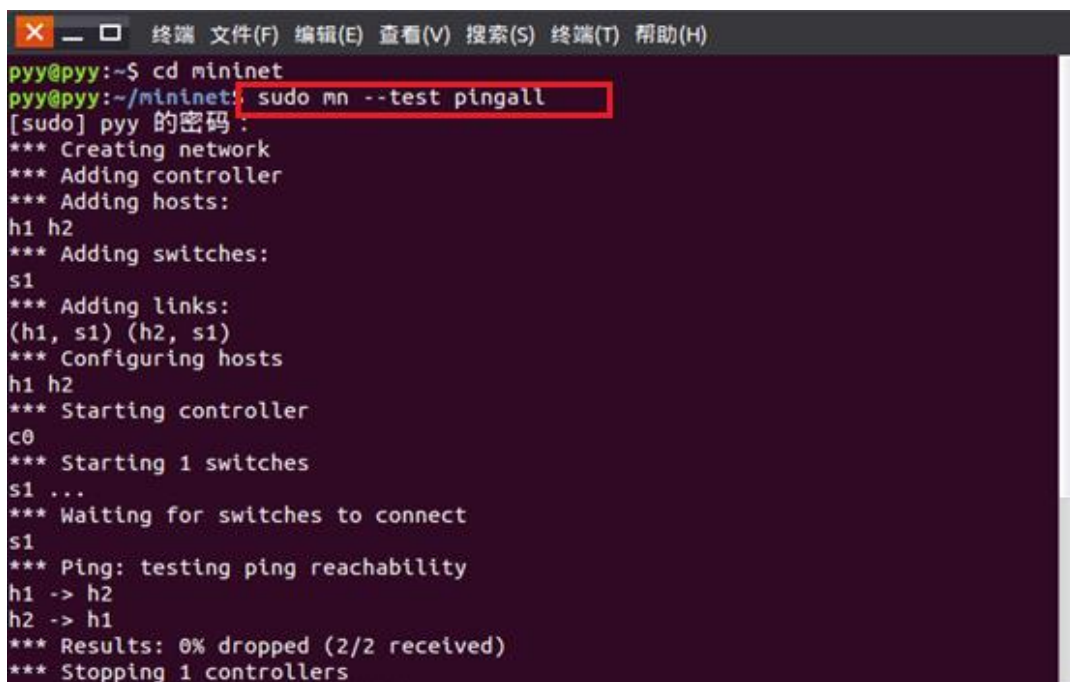
下载完成后进入 mininet 文件夹，打开终端输入 `util/install.sh -f` 开始安装。



```
pyy@pyy: ~/mininet$ util/install.sh -fnv
Detected Linux distribution: Ubuntu 15.10 wily amd64
Ubuntu
Installing OpenFlow reference implementation...
[sudo] pyy 的密码:
对不起, 请重试。
[sudo] pyy 的密码:
正在读取软件包列表...
正在分析软件包的依赖关系树...
正在读取状态信息...
autoconf 已经是最新的版本。
automake 已经是最新的版本。
gcc 已经是最新的版本。
libtool 已经是最新的版本。
make 已经是最新的版本。
下列软件包是自动安装的并且现在不需要了:
  linux-image-generic
使用 'apt-get autoremove' 来卸载它(它们)。
升级了 0 个软件包, 新安装了 0 个软件包, 要卸载 0 个软件包, 有 0 个软件包未被升级。
正在读取软件包列表...
正在分析软件包的依赖关系树...
正在读取状态信息...
autotools-dev 已经是最新的版本。
```

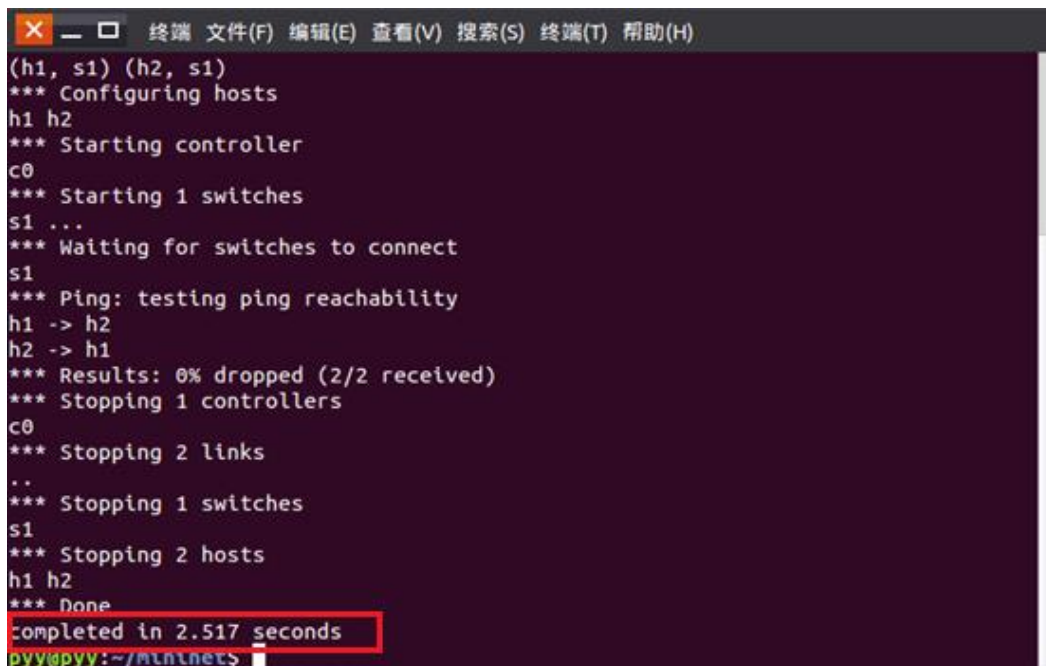
图 1-8 安装 Mininet

在终端输入 `sudo mn --test pingall` 测试 Mininet 是否安装成功, 测试成功如图 1-10。



```
pyy@pyy:~$ cd mininet
pyy@pyy:~/mininet$ sudo mn --test pingall
[sudo] pyy 的密码:
*** 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
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
```

图 1-9 测试

A terminal window with a dark background and light text. The window title bar shows standard Linux window controls and a menu with options: 终端 (Terminal), 文件(F) (File), 编辑(E) (Edit), 查看(V) (View), 搜索(S) (Search), 终端(T) (Terminal), and 帮助(H) (Help). The terminal output shows the execution of Mininet commands: (h1, s1) (h2, s1), *** Configuring hosts, h1 h2, *** Starting controller, c0, *** Starting 1 switches, s1 ..., *** Waiting for switches to connect, s1, *** Ping: testing ping reachability, 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, and finally, completed in 2.517 seconds. The last line shows the prompt pyy@pyy:~/mininet\$.

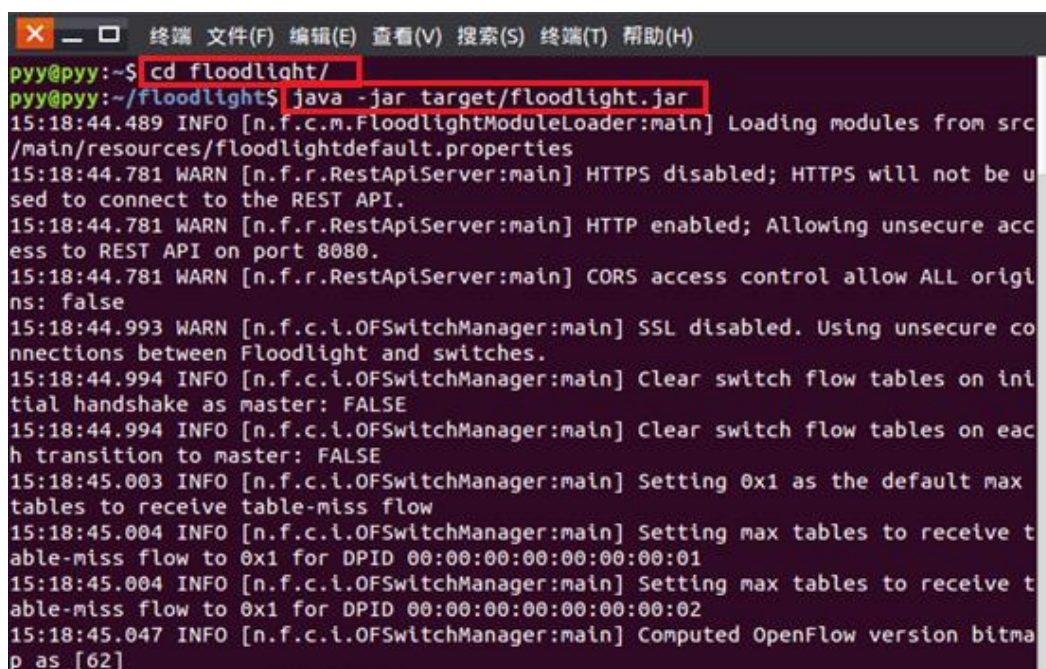
```
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 2.517 seconds
pyy@pyy:~/mininet$
```

图 1- 10 测试 Mininet 成功

1.2 构建网络拓扑图

1) 运行 Floodlight（本题使用的是 1.2 版本的 Floodlight）

终端输入 `cd floodlight` 进入 Floodlight 根目录, 接着输入 `java -jar target/floodlight.jar`

A terminal window with a dark background and light text. The window title bar shows standard Linux window controls and a menu with options: 终端 (Terminal), 文件(F) (File), 编辑(E) (Edit), 查看(V) (View), 搜索(S) (Search), 终端(T) (Terminal), and 帮助(H) (Help). The terminal output shows the execution of Floodlight startup commands: cd floodlight/ and java -jar target/floodlight.jar. The output includes various log messages from the Floodlight modules, such as Loading modules from src/main/resources/floodlightdefault.properties, HTTPS disabled, HTTP enabled, CORS access control allow ALL origins: false, SSL disabled, and setting max tables to receive table-miss flow to 0x1 for DPID 00:00:00:00:00:00:00:01 and 00:00:00:00:00:00:00:02. The last line shows the OpenFlow version bitmap as [62].

```
pyy@pyy:~$ cd floodlight/
pyy@pyy:~/floodlight$ java -jar target/floodlight.jar
15:18:44.489 INFO [n.f.c.m.FloodlightModuleLoader:main] Loading modules from src
/main/resources/floodlightdefault.properties
15:18:44.781 WARN [n.f.r.RestApiServer:main] HTTPS disabled; HTTPS will not be u
sed to connect to the REST API.
15:18:44.781 WARN [n.f.r.RestApiServer:main] HTTP enabled; Allowing unsecure acc
ess to REST API on port 8080.
15:18:44.781 WARN [n.f.r.RestApiServer:main] CORS access control allow ALL origi
ns: false
15:18:44.993 WARN [n.f.c.i.OFSwitchManager:main] SSL disabled. Using unsecure co
nnections between Floodlight and switches.
15:18:44.994 INFO [n.f.c.i.OFSwitchManager:main] Clear switch flow tables on ini
tial handshake as master: FALSE
15:18:44.994 INFO [n.f.c.i.OFSwitchManager:main] Clear switch flow tables on eac
h transition to master: FALSE
15:18:45.003 INFO [n.f.c.i.OFSwitchManager:main] Setting 0x1 as the default max
tables to receive table-miss flow
15:18:45.004 INFO [n.f.c.i.OFSwitchManager:main] Setting max tables to receive t
able-miss flow to 0x1 for DPID 00:00:00:00:00:00:00:01
15:18:45.004 INFO [n.f.c.i.OFSwitchManager:main] Setting max tables to receive t
able-miss flow to 0x1 for DPID 00:00:00:00:00:00:00:02
15:18:45.047 INFO [n.f.c.i.OFSwitchManager:main] Computed OpenFlow version bitma
p as [62]
```

图 1- 11 运行 Floodlight

2)创建网络拓扑

进入拓扑文件夹，打开终端输入 `sudo mn --custom topo1.py --topo mytopo --controller remote,127.0.0.1 --mac`

`--custom topo1.py` 表示载入自定义的拓扑文件，`--topo mytopo` 表示使用 `mytopo.py` 这个文件中定义的名为 `mytopo` 的网络拓扑结构。
`--controller remote` 设置使用远端控制器，Floodlight 和 Mininet 运行在同一个主机上，远端控制器使用环回地址 `127.0.0.1`。`--mac` 将主机和交换机的 MAC 地址设置为一个较小的、唯一的、易读的 ID。



```
pyy@pyy:~/SDNdata/first$ sudo mn --custom topo1.py --topo mytopo --controller remote,127.0.0.1 --mac
[sudo] pyy 的密码:
*** 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 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (s1, s2)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet>
```

图 1- 12 创建拓扑

打开浏览器输入 `http://localhost:8080/ui/index.html` 查看拓扑。

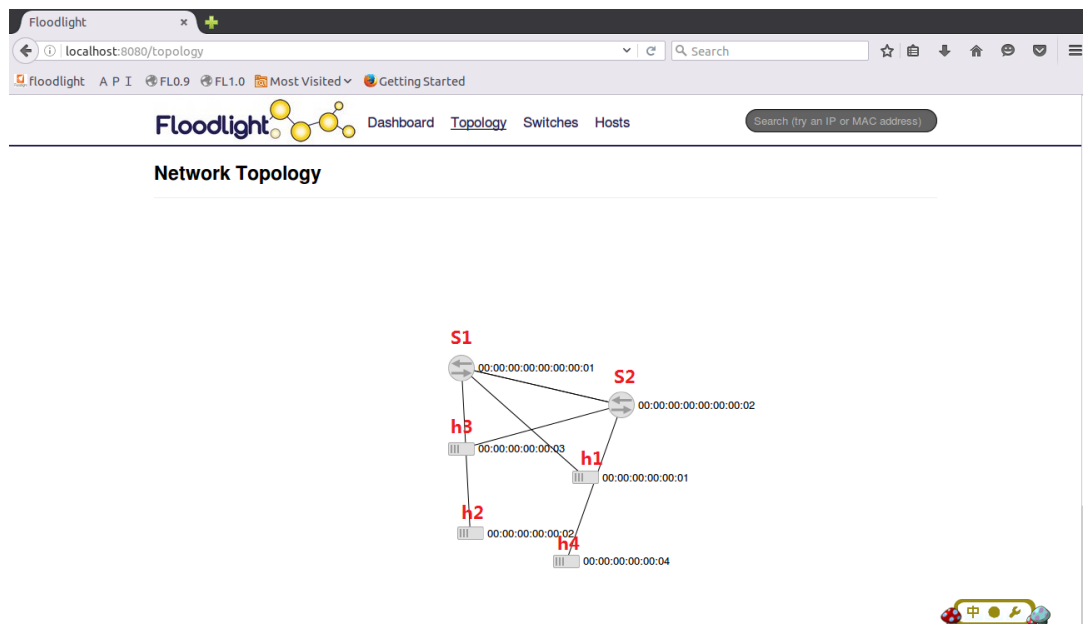


图 1- 13 拓扑结构

topo1.py 文件中的代码图 1-14 所示。

```
class MyTopo(Topo):
    def __init__(self):
        Topo.__init__(self)

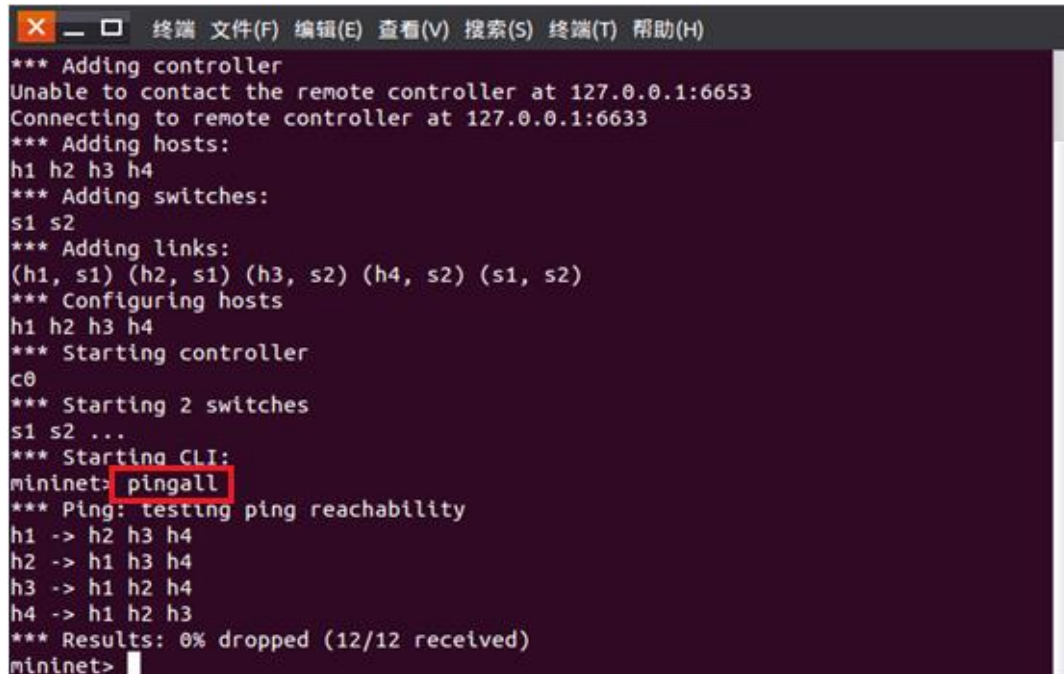
        #add Hosts and Switchs
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        H1 = self.addHost('H1')
        H2 = self.addHost('H2')
        H3 = self.addHost('H3')
        H4 = self.addHost('H4')
        #add Links
        self.addLink(H1,s1)
        self.addLink(H2,s1)
        self.addLink(s1,s2)
        self.addLink(H3,s2)
        self.addLink(H4,s2)

topos = { 'mytopo' : ( lambda : MyTopo() ) }
```

图 1- 14 拓扑结构代码

1.3 在终端进行两两相互 Ping

输入pingall命令进行两两互Ping，结果如图1-15。



```
终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
*** 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 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (s1, s2)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

图 1- 15 pingall 结果

第 2 小题 :初步分析基于 OpenFlow 的 SDN 网络的控制功能

完成第 1 题的基础上，结合具体报文，了解 SDN 的基本运行流程和相关协议的功能。

2.1 下发流表实现控制

1) 运行 Floodlight 控制器(floodlight 版本为 1.2)

在终端输入 `cd floodlight/target`，接着在终端输入 `java -jar floodlight.jar`。

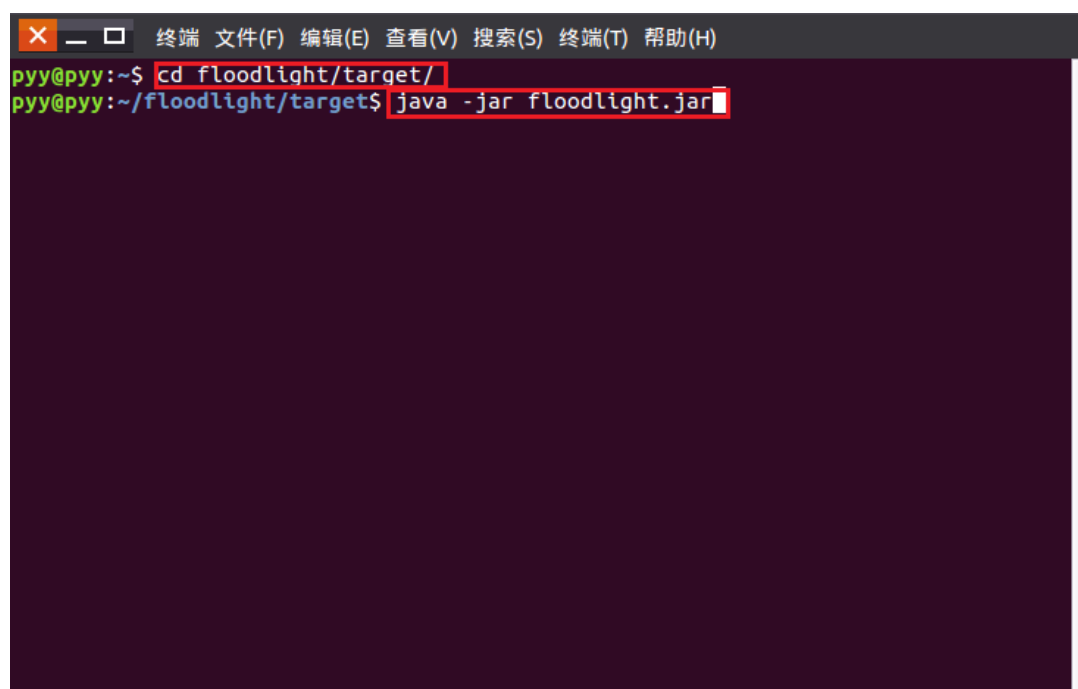
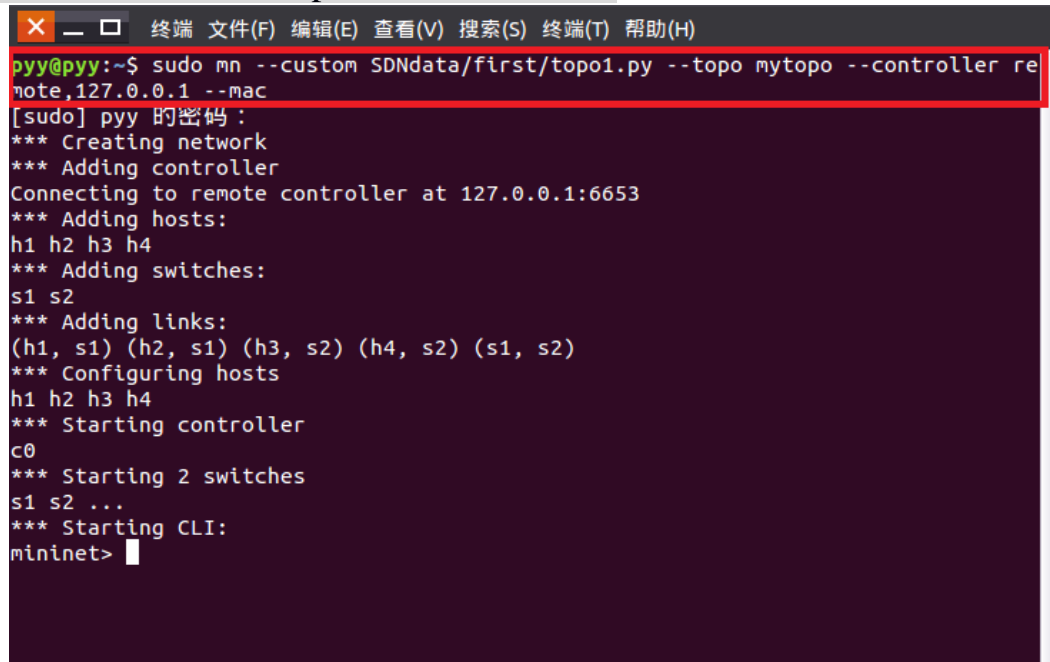
A screenshot of a terminal window with a dark background. The title bar at the top shows standard window controls and a menu with options: 终端 (Terminal), 文件(F) (File), 编辑(E) (Edit), 查看(V) (View), 搜索(S) (Search), 终端(T) (Terminal), and 帮助(H) (Help). The terminal content shows two commands entered at the prompt 'p yy@p yy:~\$'. The first command is 'cd floodlight/target/' and the second is 'java -jar floodlight.jar'. Both commands and the prompt are highlighted with a red rectangular box. The cursor is at the end of the second command.

图 2-1 启动 floodlight 控制器

2) 在 Mininet 中建立简单拓扑结构

打开新终端输入以下代码：

```
sudo mn --custom SDNdata/topo/topo1.py --topo mytopo  
--controller remote,ip=127.0.0.1 --mac
```



```
pyy@pyy:~$ sudo mn --custom SDNdata/first/topo1.py --topo mytopo --controller re  
mote,127.0.0.1 --mac  
[sudo] pyy 的密码 :  
*** Creating network  
*** Adding controller  
Connecting to remote controller at 127.0.0.1:6653  
*** Adding hosts:  
h1 h2 h3 h4  
*** Adding switches:  
s1 s2  
*** Adding links:  
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (s1, s2)  
*** Configuring hosts  
h1 h2 h3 h4  
*** Starting controller  
c0  
*** Starting 2 switches  
s1 s2 ...  
*** Starting CLI:  
mininet>
```

图 2-2 建立网络拓扑

具体代码见图 2-3，此拓扑结构由两台交换机 s1,s2，四台主机 h1、h2、h3 和 h4 组成，图形化拓扑结构如图 2-4 所示。

```
class MyTopo(Topo):  
    def __init__(self):  
        Topo.__init__(self)  
  
        #add Hosts and Switches  
        s1 = self.addSwitch('s1')  
        s2 = self.addSwitch('s2')  
        H1 = self.addHost('H1')  
        H2 = self.addHost('H2')  
        H3 = self.addHost('H3')  
        H4 = self.addHost('H4')  
  
        #add Links  
        self.addLink(H1,s1)  
        self.addLink(H2,s1)  
        self.addLink(s1,s2)  
        self.addLink(H3,s2)  
        self.addLink(H4,s2)  
  
topos = { 'mytopo' : ( lambda : MyTopo() ) }
```

图 2-3 网络拓扑代码

打开浏览器输入 `http://localhost:8080/ui/index.html`，进入 Floodlight Web GUI，如图 2-4 所示。

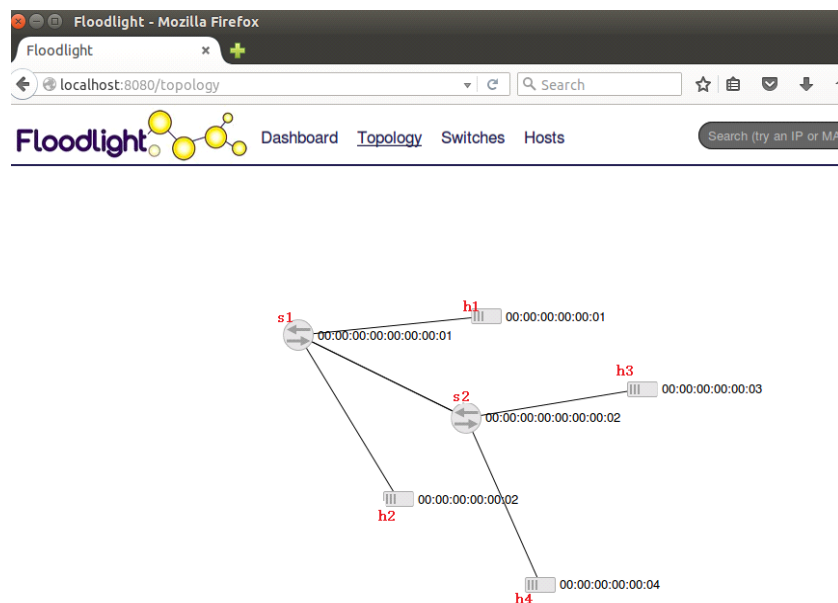


图 2-4 拓扑界面

输入 `pingall` 命令检查各主机之间的连通性，此时各个主机之间可以相互 Ping 通，如图 2-5 所示。

```
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (s1, s2)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

图 2-5 测试各主机间连通性

3) 添加流表

下发流表使得 h1 和 h2, h1 和 h4 互相 ping 不通, h1 和 h3 可以互通。这里我们给交换机 s1 下发两条流表, 使得 s1 丢弃 h1 发给 h2 和 h4 的 IP 数据包。给交换机 s1 下放流表如图 2-6, 流表具体代码如下:

```
curl -d '{"switch": "00:00:00:00:00:00:00:01",  
"actions": "",  
"priority": "32768",  
"name": "H1H2-deny",  
"in_port": "1",  
"eth_type": "0x0800",  
"nw_proto": "1",  
"ipv4_src": "10.0.0.1/32",  
"ipv4_dst": "10.0.0.2/32",  
"active": "true"}' http://127.0.0.1:8080/wm/staticflowpusher/json
```

```
curl -d '{"switch": "00:00:00:00:00:00:00:01",  
"actions": "",  
"priority": "32768",  
"name": "H1H4-deny",  
"in_port": "1",  
"eth_type": "0x0800",  
"nw_proto": "1",  
"ipv4_src": "10.0.0.1/32",  
"ipv4_dst": "10.0.0.4/32",  
"active": "true"}' http://127.0.0.1:8080/wm/staticflowpusher/json
```

```
pyy@pyy:~$ curl -d '{"switch":"00:00:00:00:00:00:00:01","name":"H1H2_deny","in_port":1,"eth_type":"0x0800","ipv4_src":"10.0.0.1/32","ipv4_dst":"10.0.0.2/32","active":"true"}' http://127.0.0.1:8080/wm/staticflowpusher/json
{"status": "Entry pushed"}pyy@pyy:~$ curl -d '{"switch":"00:00:00:00:00:00:00:00","name":"H1H3_deny","in_port":1,"eth_type":"0x0800","ipv4_src":"10.0.0.1/32","ipv4_dst":"10.0.0.4/32","active":"true"}' http://127.0.0.1:8080/wm/staticflowpusher/json
{"status": "Entry pushed"}pyy@pyy:~$
```

图 2-6 添加流表成功

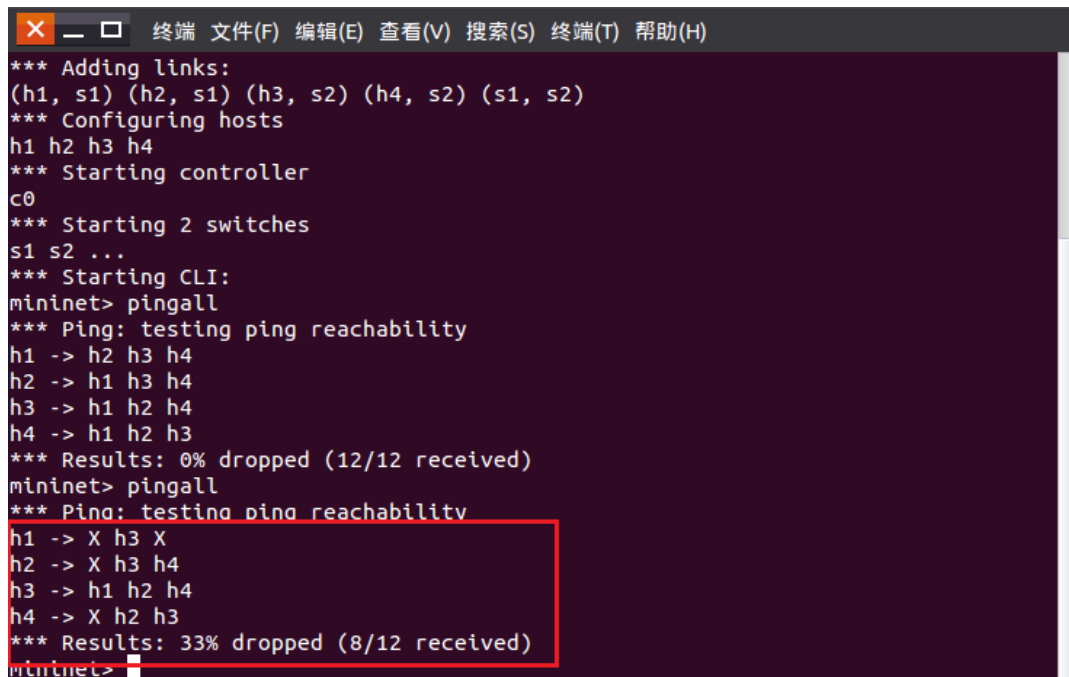
在终端中输入 `sh ovs-ofctl dump-flows s1` 查看已经添加的两条流表。

```
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 X
h2 -> X h3 h4
h3 -> h1 h2 h4
h4 -> X h2 h3
*** Results: 33% dropped (8/12 received)
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0xa000003eda946f, duration=66.803s, table=0, n_packets=2, n_bytes=196,
 idle_age=27, ip,in_port=1,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=drop
 cookie=0xa00000ff6be475, duration=65.826s, table=0, n_packets=2, n_bytes=196,
 idle_age=17, ip,in_port=1,nw_src=10.0.0.1,nw_dst=10.0.0.4 actions=drop
 cookie=0x200000000000000, duration=2.441s, table=0, n_packets=1, n_bytes=42, id
 le_timeout=5, idle_age=2, priority=1,arp,in_port=3,dl_src=00:00:00:00:00:04,dl_
 dst=00:00:00:00:00:02 actions=output:2
 cookie=0x200000000000000, duration=2.437s, table=0, n_packets=0, n_bytes=0, idl
 e_timeout=5, idle_age=2, priority=1,arp,in_port=2,dl_src=00:00:00:00:00:02,dl_d
 st=00:00:00:00:00:04 actions=output:3
 cookie=0x0, duration=74.307s, table=0, n_packets=86, n_bytes=8148, idle_age=2,
 priority=0 actions=CONTROLLER:65535
mininet>
```

图 2-7 查看添加的流表

4) Ping 测试

主机之间互相 Ping，发现 h1 和 h2,h1 和 h4 之间不能互相 Ping 通，如图 2-8 所示。



```
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (s1, s2)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 X
h2 -> X h3 h4
h3 -> h1 h2 h4
h4 -> X h2 h3
*** Results: 33% dropped (8/12 received)
mininet>
```

图 2-8 测试下发流表后各主机连通性

2.2 Openflow 协议报文结构分析及其作用

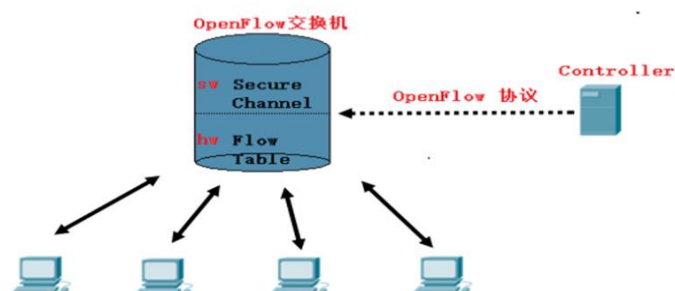


图 2-9

如图 2-9，OpenFlow 交换机通过安全通道上传输的 OpenFlow 协

议与外部的控制器通信，流表则控制数据包的高速转发。OpenFlow 协议支持三种消息类型：controller-to-switch, Asynchronous, symmetric。每一类消息类型又包含多个子消息。Packet_out 消息由控制器发起用来执行链路发现，其流程为：控制器把包含 LLDP 的 Packet_out 发给 Openflow 交换机，Openflow 交换机接收到 Packet_out 消息，就会把 LLDP 数据包进行泛洪。其他 Openflow 交换机收到 LLDP，进行流表匹配，若失败，则发送 Packet_in 给控制器。控制器收到 Packet_in 消息后，创建两台交换机之间的链接记录，从而控制器能够创建完备的网络拓扑结构数据库。捕获到的含有 LLDP 报文的 Packet_out 消息如图 2-10，其报文结构图如图 2-11。

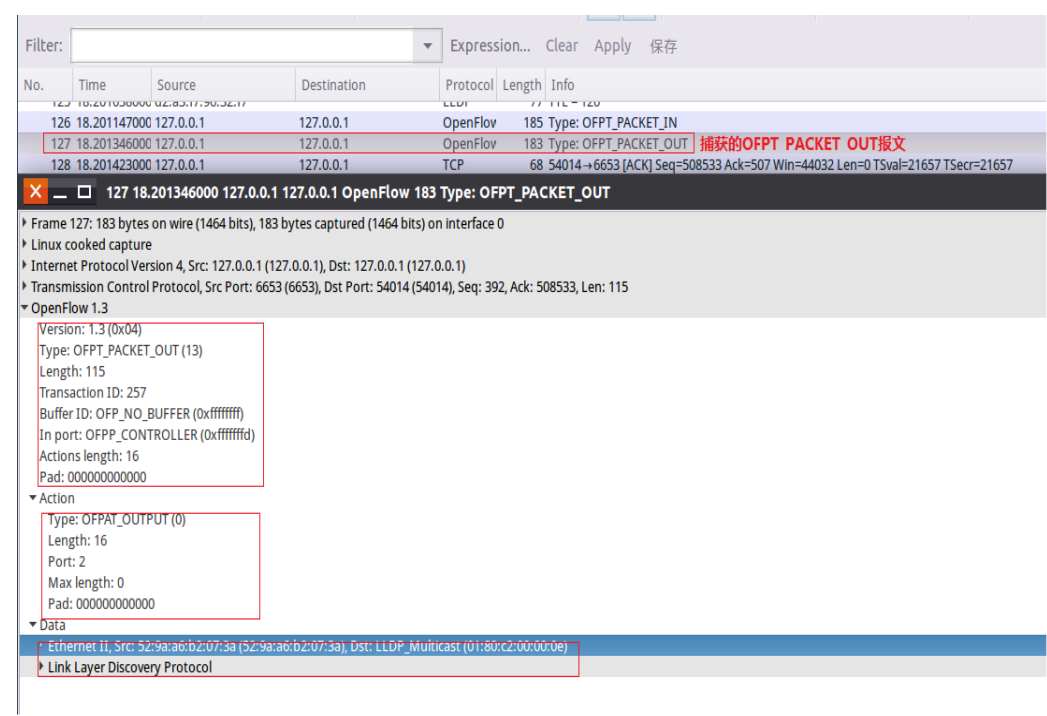


图 2-10 捕获的 Openflow 协议报文

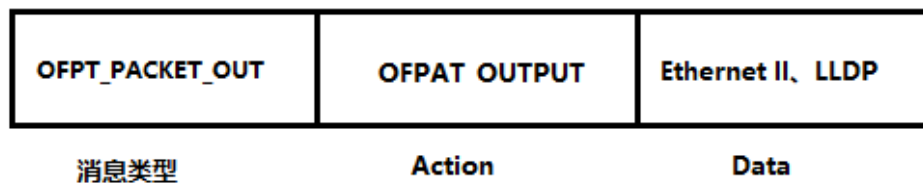


图 2-11 报文结构图

2.3 传统分布式网络与 SDN

1) 传统式网络实现过程

- 搭建如图 2-12 的拓扑结构图
- 配置交换机，路由器，主机，主机之间能够互通
- 给 s1 配置 ACL 访问控制列表,使 h1 与 h2 不能互通

```
s1(config)#access-list 101 deny ip host 10.0.0.2 host 20.0.0.2
```

```
s1(config)#access-list 101 permit ip any any
```

```
s1(config)#int fa0/1
```

```
s1(config-if)#ip access-group 101 in
```

- 给 r1 配置 ACL 访问控制列表,使 h1 与 h4 不能互通

```
r1(config)#access-list 100 deny ip host 10.0.0.2 host 40.0.0.3
```

```
r1(config)#access-list 100 permit ip any any
```

```
r1(config)#int fa0/0
```

```
r1(config-if)#ip access-group 100 in
```

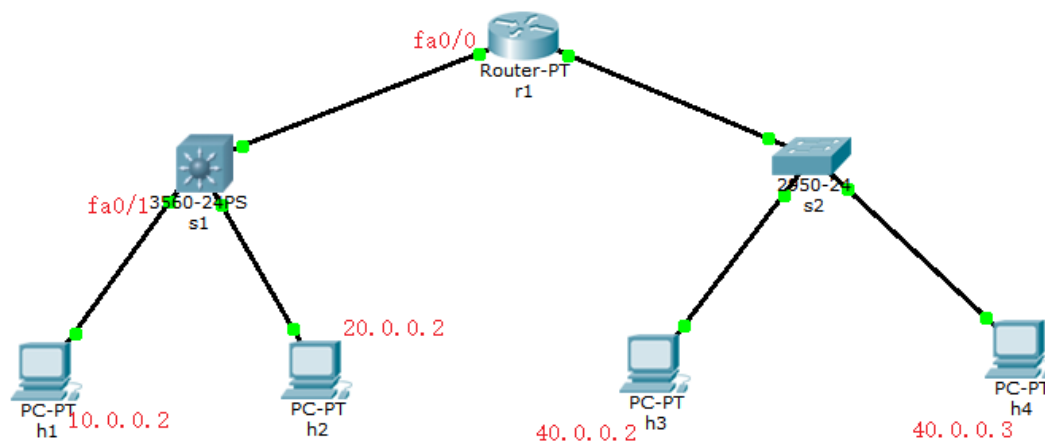


图 2-12 传统式网络拓扑

2) 差别

在传统网络中，网络配置需要通过命令手动配置，传统网络数据的转发根据配置策略进行，但是网络具有动态性，一旦网络业务变化，相关联设备需要手工的进行修改，尤其是在一个大型网络中，效率会非常低。例如在实现要求 1 时，需要在交换机（s1），路由器（r1）分别添加访问控制列表，同时 s1 还需启动路由功能。

而在 SDN 中，只需要在控制器中加上流表即可实现要求。SDN 支持控制平面与转发平面的分离，实现了网络设备的集中性控制。逻辑上的集中控制能够支持获得网络资源的全局信息，并且能根据业务需求进行资源调配和优化。集中控制使得网络在逻辑上视为一台设备进行运行与维护，无须对物理设备进行现场配置，提高网络效率。

第 3 题：简单应用的开发与实现——Web 访问用户控制

开发一个简单的 web 业务系统，对外提供网页访问服务，并在服务端网络基于控制器提供的北向 API 来实现控制具体用户访问权限的功能，开发出配置程序能够使管理员随时更改配置，并进行测试。

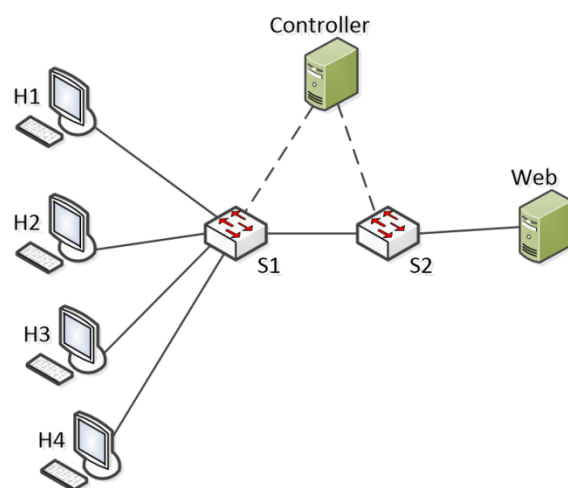


图 3- 1

基本思路

本题将搭建 Web 服务器、部署网络环境以及实现访问控制的过程集成在应用程序中实现。运行应用程序将自动生成本题所需的拓扑，在应用程序界面对“web”主机开启 Web 服务功能，实现 H1~H4 都能对其进行 Web 访问，通过点击程序界面的按钮实现对主机 ACL 增删，从而实现主机对 Web 访问的限制，并将访问结果显示在应用界面中。本题的所有代码位于 webcontrol 文件夹中。

3.1 Web 服务器搭建和页面测试

1) 运行应用程序（本题使用 1.2 版本的 Floodlight）

进入 webcontrol 文件夹终端打开输入 `sudo python accesscontrol.py` 开启应用，`accesscontrol.py` 文件就是应用的主程序。

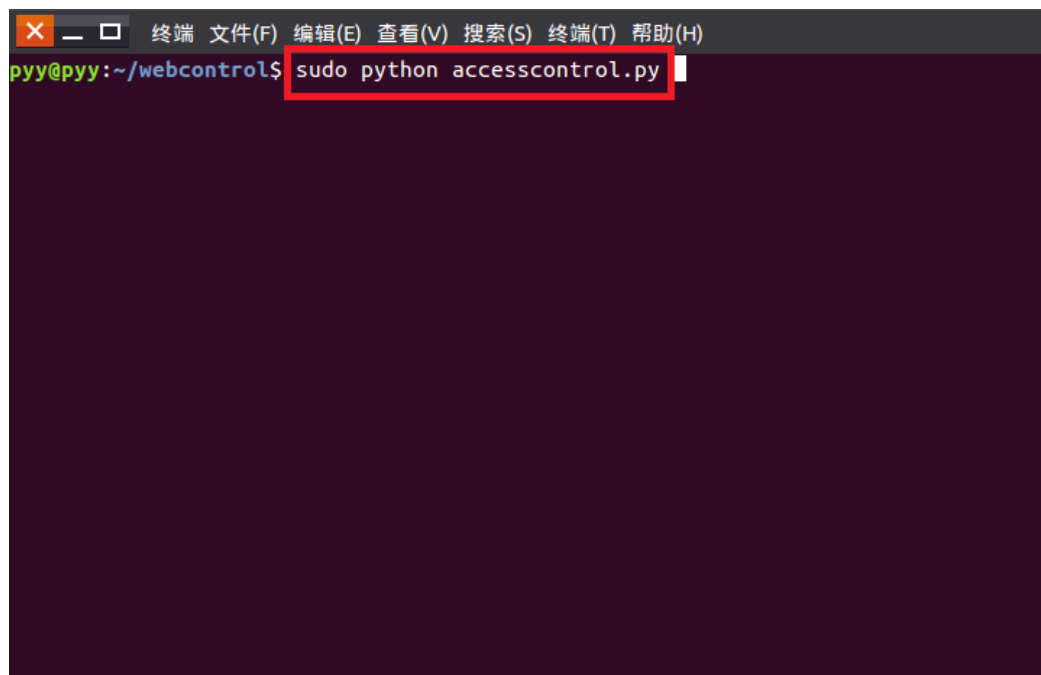


图 3-2

2) 按照提示输入 Floodlight 的绝对路径，打开 Floodlight 控制器同时也打开应用主界面。应用主界面如图 3-4。

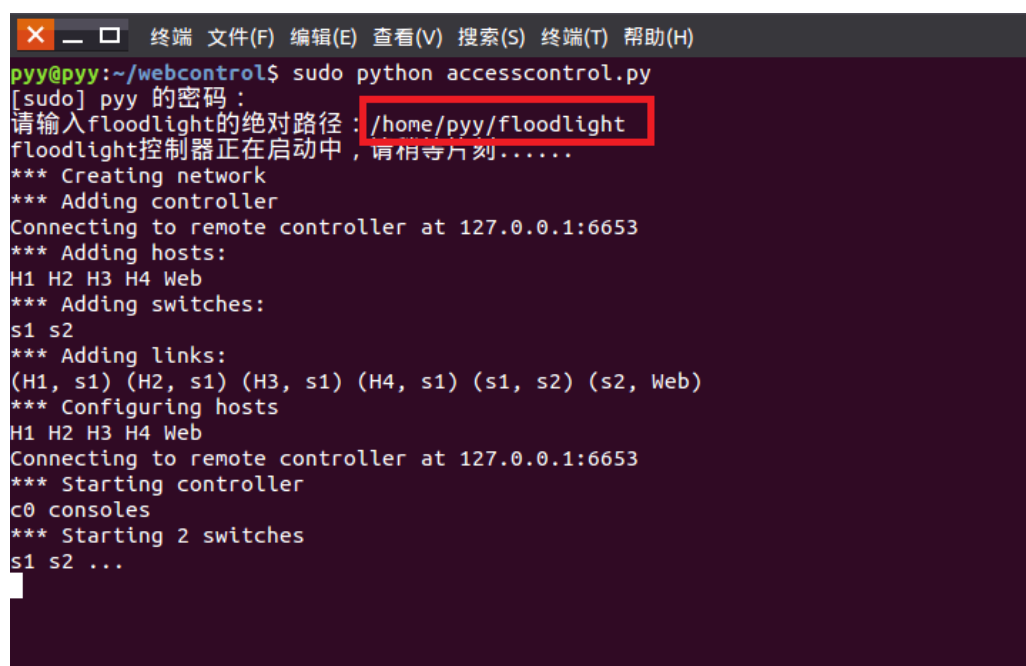


图 3-3



图 3-4 应用主界面

3) 点击图 3-5 所示的应用界面“Web 服务器”按钮，然后进入图 3-6 所示界面，然后点击“开启 Web 服务器”按钮。

这一步相当于在名为“web”的主机终端输入 `python -m`

`SimpleHTTPServer 80 &` 开启其 Web 服务。



图 3-5 点击“Web 服务器”按钮



图 3-6 点击按钮开启 Web 服务

4) 点击“主机”按钮回到应用主界面



图 3-7 点击“主机” 按钮

5) 在应用主界面点击按钮“使用 firefox 访问 web 主页”，即可使用 H1 访问 Web 主页并弹出浏览器显示主页。主页如图 3-9，这里 Web 服务器的 IP 为 10.0.0.5。



图 3-8 使用 firedox 访问 Web 服务器主页



图 3-9 Web 服务器主页

主机 H2、H3、H4 的访问 Web 服务器主页过程类似，这里不一

一赘述。

3.2 拓扑视图

在浏览器输入 `http://localhost:8080/ui/index.html` 查看图形化拓扑如图 3-10。

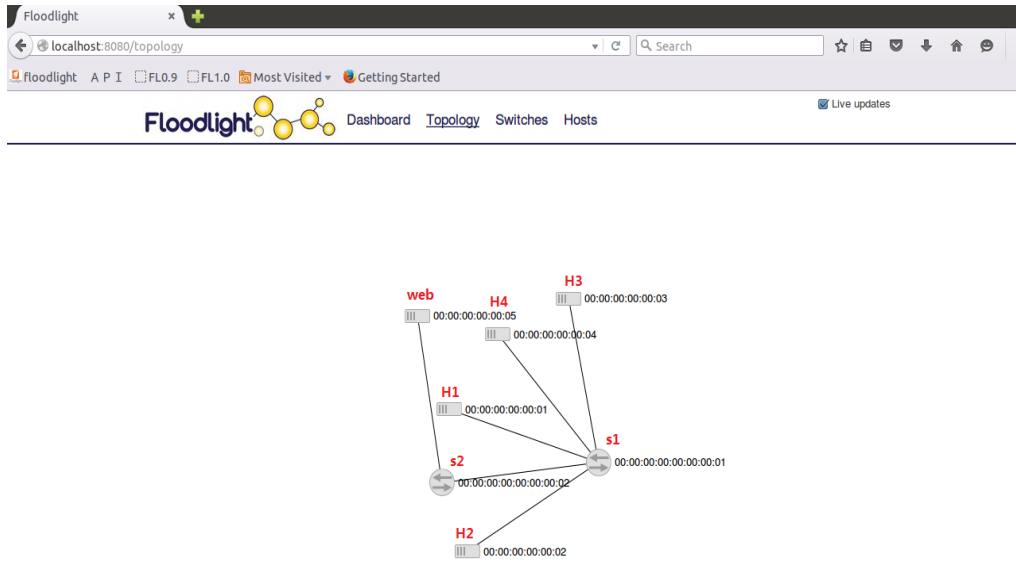


图 3-10

拓扑结构代码如图 3-11 所示

```
__author__ = 'wjl'
# encoding utf-8
from mininet.topo import Topo

class MyTopo(Topo):
    def __init__(self):
        Topo.__init__(self)
        H1 = self.addHost('H1')
        H2 = self.addHost('H2')
        H3 = self.addHost('H3')
        H4 = self.addHost('H4')
        Web = self.addHost('Web')
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        self.addLink(H1,s1)
        self.addLink(H2,s1)
        self.addLink(H3,s1)
        self.addLink(H4,s1)
        self.addLink(s1,s2)
        self.addLink(s2,Web)

topos = {'mytopo': (lambda : MyTopo()) }
```

图 3-11

3.3 北向 API

本题中访问限制的操作都是通过应用对 ACL 的添加和删除，给出用于添加和删除 ACL 的代码如图 3-12。

```
def hostACL():
    if(self.host_boole):
        pusher = ACL('127.0.0.1')
        acl = {
            "ruleid":self.ruleid
        }
        pusher.remove(acl)
        print '我删除了'+self.node.name+'的acl'
        self.host_boole = False
        self.hostname.set('把'+self.node.name+'添加到黑名单')
        label['bg'] = 'green'
    else:
        pusher = ACL('127.0.0.1')
        acl = {
            "src-ip":self.node.IP()+'/32',
            "dst-ip":"10.0.0.5/32",
            "nw-proto":"TCP",
            "tp-dst":80,
            "action":"deny"
        }
        pusher.set(acl)
        print '我添加了'+self.node.name+'的acl'
        self.host_boole = True
        self.hostname.set('从黑名单删除'+self.node.name)
        label['bg'] = 'red'
        f = open('count.txt')
        count = f.read()
        icount = int(count)
        f.close()
        self.ruleid = icount;
        os.system('echo '+str(icount + 1)+' > count.txt')
```

图 3-12

用到的北向 API: http://<controller_ip>:8080/wm/acl/rules/json（用于添加 ACL），http://<controller_ip>:8080/wm/acl/clear/json（用于删除 ACL）。

3.4 访问限制测试

1) 在搭建好 Web 服务器基础之上，应用主界面点击“wget”按钮，此时并未进行访问限制，故四台主机都能够获取主页，结果显示在四个主机对应的终端框中，如图 3-13。



图 3-13 四台主机访问 Web 服务器

2) 点击“中断”按钮断开主机与服务器连接，再点击“清空”按钮清空终端框内容。



图 3- 14 依次点击“中断”和“清空”

3) 点击绿色按钮“把 H1 添加到黑名单”和“把 H2 添加到黑名单”，按钮变红色，将 H1 和 H2 主机加入访问黑名单，如图 3-15。终端显示添加的 ACL 表项成功，如图 3-16。



图 3-15 将 H1 和 H2 加入黑名单

```

X 终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
[sudo] ppy 的密码 :
请输入floodlight的绝对路径 : /home/pyy/floodlight
floodlight控制器正在启动中, 请稍等片刻.....
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
H1 H2 H3 H4 Web
*** Adding switches:
s1 s2
*** Adding links:
(H1, s1) (H2, s1) (H3, s1) (H4, s1) (s1, s2) (s2, Web)
*** Configuring hosts
H1 H2 H3 H4 Web
Connecting to remote controller at 127.0.0.1:6653
*** Starting controller
c0 consoles
*** Starting 2 switches
s1 s2 ...
(200, 'OK', '{"status": "Success! New rule added."}')
我添加了H1的acl
(200, 'OK', '{"status": "Success! New rule added."}')
我添加了H2的acl

```

图 3-16 成功添加 ACL 表项

4) 点击“wget”按钮，四台主机开始访问 Web 服务器，此时对 H1 和 H2 做了访问限制，无法获取 web 服务器主页，而未做限制的 H3 和 H4 仍可获取服务器主页，如图 3-17。

主机	Web 服务器	wget	中断	清空	退出
从黑名单删除H1	从黑名单删除H2	把H3添加到黑名单	把H4添加到黑名单		
使用firefox访问web主页	使用firefox访问web主页	使用firefox访问web主页	使用firefox访问web主页		
H1# --2016-05-25 20:20:23-- http://10.0.0.5/↓ 正在连接 10.0.0.5:80...	--2016-05-25 20:20:23-- http://10.0.0.5/↓ 正在连接 10.0.0.5:80...	content="text/html; charset=utf-8" />↓ <body bgcolor=ADADAD>↓ <div align=center>↓ 西南民族大学主页 ↓ ↓ </div>↓ ↓ <div>↓ 团队成员：周成 吴俊磊 彭延阳 邱夏 谢梅 徐昊宸↓ </div>↓ </body>↓ </html>↓ ↓ 100%[=====>]↓ 370 --.-KB/s 用时 0s ↓ ↓ 2016-05-25 20:20:23 (36.7 MB/s) - 已写入至标准输出 [370/370]↓ ↓ H3#	content="text/html; charset=utf-8" />↓ <body bgcolor=ADADAD>↓ <div align=center>↓ 西南民族大学主页 ↓ ↓ </div>↓ ↓ <div>↓ 团队成员：周成 吴俊磊 彭延阳 邱夏 谢梅 徐昊宸↓ </div>↓ </body>↓ </html>↓ ↓ 100%[=====>]↓ 370 --.-KB/s 用时 0s ↓ ↓ 2016-05-25 20:20:23 (34.5 MB/s) - 已写入至标准输出 [370/370]↓ ↓ H4#		

图 3-17 测试结果

第 4 题：简单的路由控制应用开发

内容：基于北向 API 开发一个简单的路由控制应用，能够根据访问的源地址与目的地址选择不同的路径。

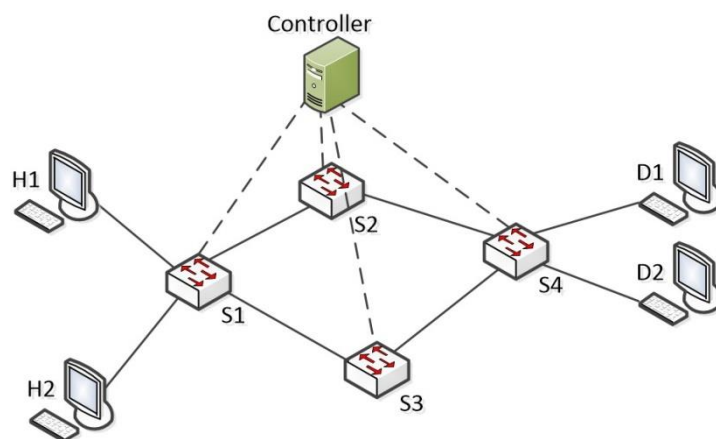


图 4- 1

基本思路

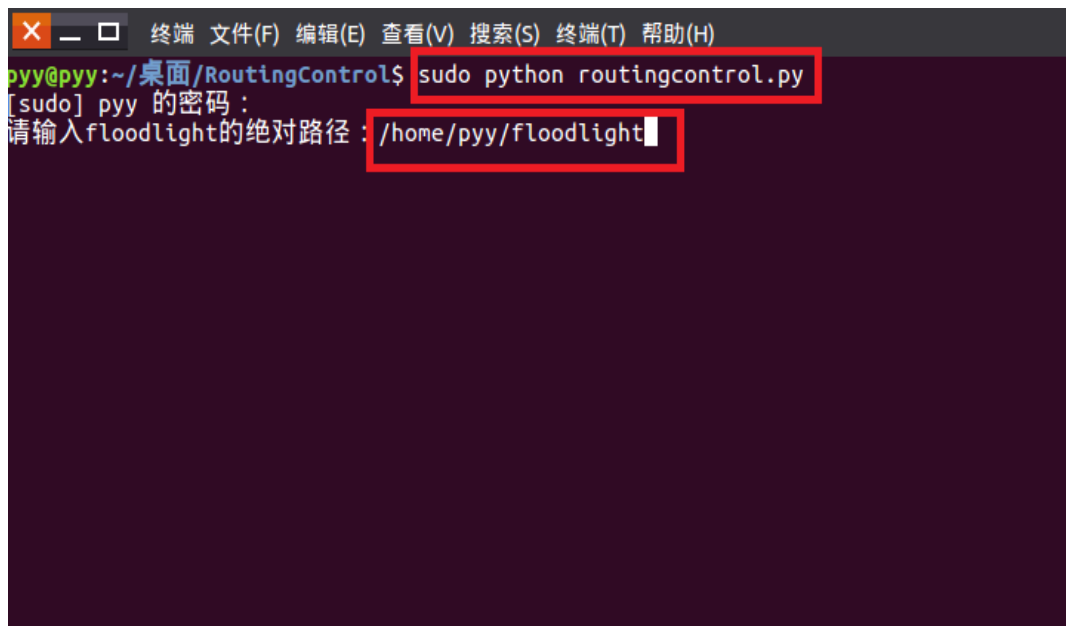
本题使用的是 FloodLight1.2 版本的控制器，使用的配置文件不是默认的,而是删除了 forwarding 模块后的配置文件 no_fwd.properties。通过 Floodlight 控制器添加静态流表实现数据包路径的设置。之后再通过数据包经过各个交换机之后反馈的流表信息，对数据包走向的验证。本题所有代码在 RoutingControl 文件夹下。

4.1 部署网络环境

1) 运行应用程序

在 RoutingControl 文件夹下终端打开输入 `sudo python routing`

control.py，这里 routingcontrol.py 文件是应用主程序。然后按提示输入 Floodlight 控制器的绝对路径。部署网络环境过程集成在路由控制程序中，开启应用同时也打开了 Floodlight 控制器和建立本题所需的拓扑。



```
pyy@pyy:~/桌面/RoutingControl$ sudo python routingcontrol.py
[sudo] pyy 的密码:
请输入floodlight的绝对路径: /home/pyy/floodlight
```

图 4-2 运行程序

2) 打开浏览器输入 <http://localhost:8080/ui/index.html> 查看拓扑，如图 4-3。

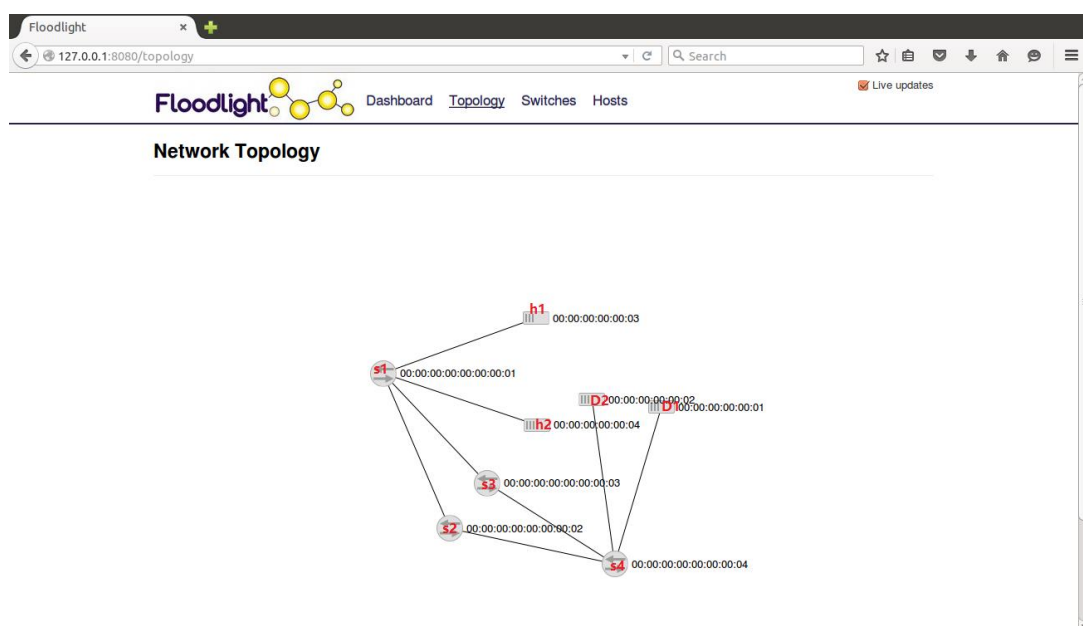


图 4-3 拓扑界面

3) 网络拓扑代码如图 4-4:

```
from mininet.topo import Topo

class MyTopo(Topo):
    def __init__(self):
        Topo.__init__(self)
        H1 = self.addHost('H1')
        H2 = self.addHost('H2')
        D1 = self.addHost('D1')
        D2 = self.addHost('D2')
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        self.addLink(s1,H1)
        self.addLink(s1,H2)
        self.addLink(s1,s2)
        self.addLink(s1,s3)
        self.addLink(s4,s2)
        self.addLink(s4,s3)
        self.addLink(s4,D1)
        self.addLink(s4,D2)

topos = {'mytopo' : ( lambda : MyTopo() ) }
```

图 4-4 拓扑结构代码

4.2 北向 API

本题目使用到的北向 API 为 `http://<controller_ip>:8080/wm/staticflowpusher/json`，用于添加和删除静态流表。

4.3 应用程序和验证程序

路由控制程序

1) 已经打开的路由控制程序界面如图 4-5。



图 4-5 路由控制应用界面

在打开应用的过程中已经自动下发了所有交换机所有端口广播 ARP 报文的流表，并进行 Pingall。如图 4-6

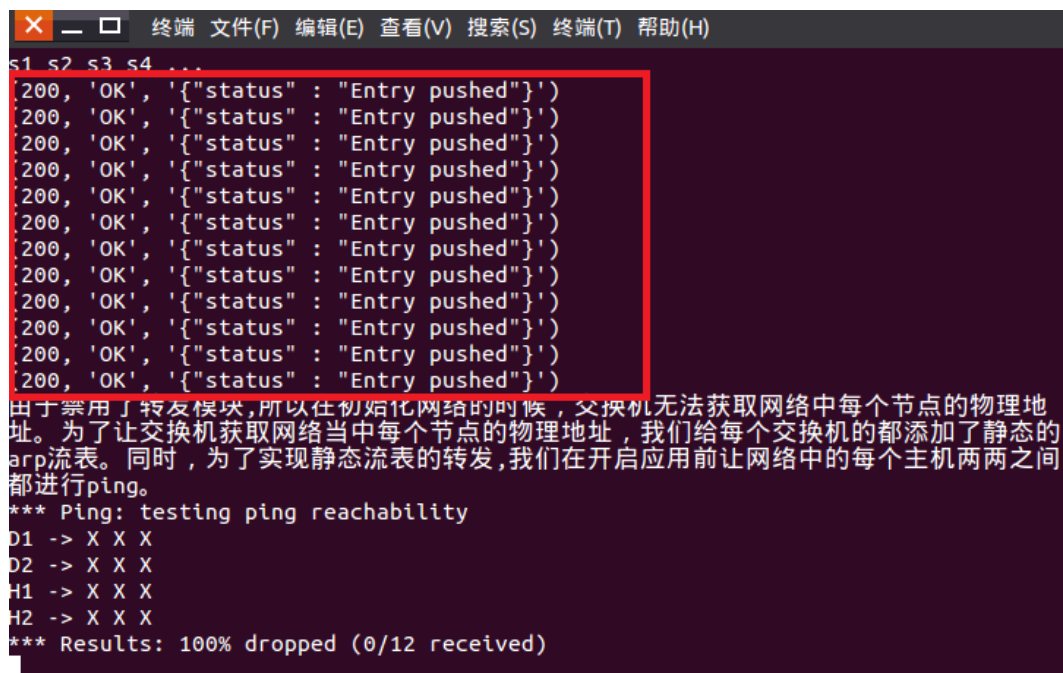


图 4-6 自动添加 ARP 的流表

2) 点击“ping”按钮，进行 H1 Ping D1、H1 Ping D2、H2 Ping D1 和

H2 Ping D2 的操作。此时未添加流表，都无法无法 ping 通。结果显示在对应的终端框中，如图 4-7。

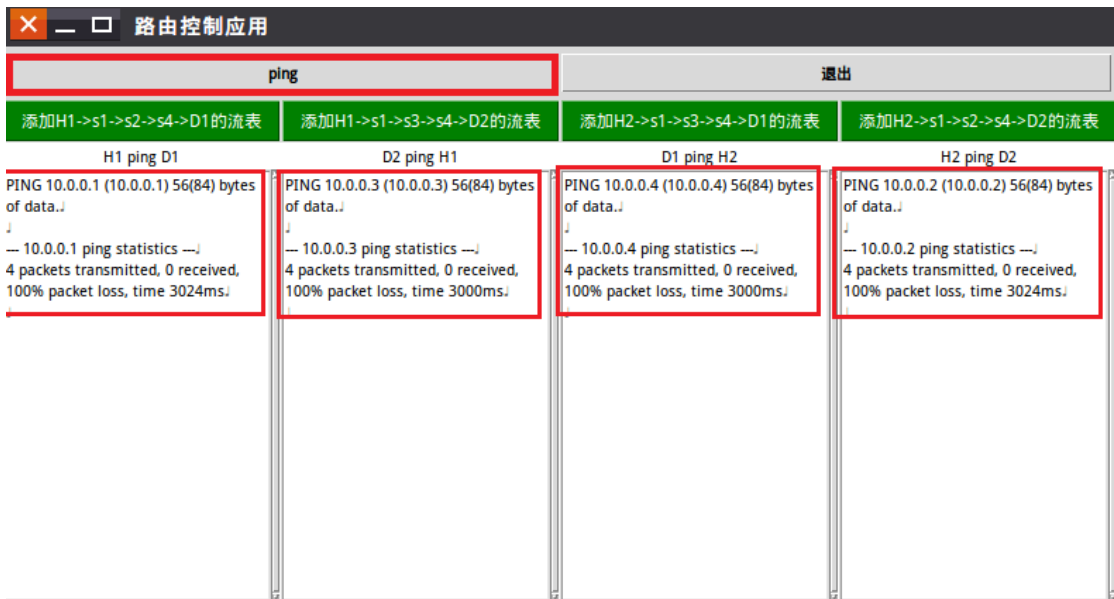


图 4-7 添加流表前 ping 的结果

3) 点击四个绿色按钮添加对应的流表，如图 4-8。按钮变红色，再点击“ping”按钮，使得能够按照规定路径 Ping 通。Ping 结果显示在对应的终端框中，如图 4-9。



图 4-8 点击按钮添加相应的流表

路由控制应用			
ping		退出	
删除H1->s1->s2->s4->D1的流表	删除H1->s1->s3->s4->D2的流表	删除H2->s1->s3->s4->D1的流表	删除H2->s1->s2->s4->D2的流表
H1 ping D1	D2 ping H1	D1 ping H2	H2 ping D2
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.: 64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.944 ms. 64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.045 ms. 64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.023 ms. 64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.043 ms. . -- 10.0.0.1 ping statistics -- 4 packets transmitted, 4 received, 0% packet loss, time 2999ms. rtt min/avg/max/mdev = 0.023/0.263/0.944/0.393 ms.	PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.: 64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.882 ms. 64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.046 ms. 64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.038 ms. 64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.045 ms. . -- 10.0.0.3 ping statistics -- 4 packets transmitted, 4 received, 0% packet loss, time 2999ms. rtt min/avg/max/mdev = 0.038/0.252/0.882/0.363 ms.	PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.: 64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=1.00 ms. 64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=12.8 ms. 64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.031 ms. 64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.032 ms. . -- 10.0.0.4 ping statistics -- 4 packets transmitted, 4 received, 0% packet loss, time 3001ms. rtt min/avg/max/mdev = 0.031/3.474/12.827/5.414 ms.	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=4.34 ms. 64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.042 ms. 64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.041 ms. 64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.039 ms. . -- 10.0.0.2 ping statistics -- 4 packets transmitted, 4 received, 0% packet loss, time 2999ms. rtt min/avg/max/mdev = 0.039/1.116/4.342/1.862 ms.

图 4-9 添加流表后 ping 的结果

4) 这里给出添加静态流表的结构，如图 4-10。

```
def addstaticFlow(dpid,output,name,in_port,ip):
    flow = {"switch":dpid,
            "actions":"output="+output,
            "priority":"32768",
            "name":name,
            "in_port":in_port,
            "eth_type":"0x0800",
            "nw_proto":"1",
            "ipv4_src":ip[0]+"/32",
            "ipv4_dst":ip[1]+"/32",
            "active":"true"
            }
```

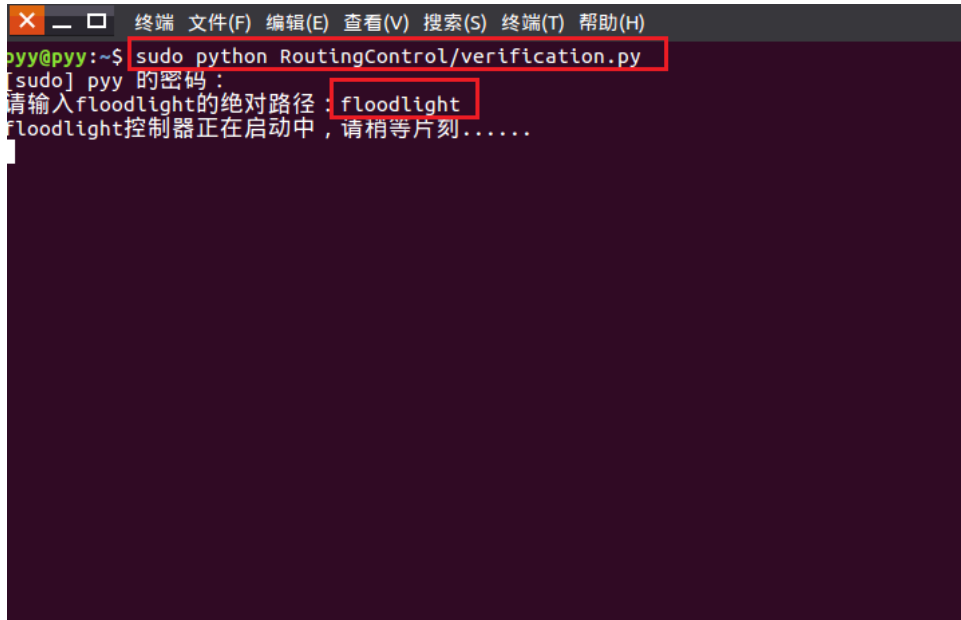
图 4-10 流表结构代码

验证程序

算法原理：下发题意要求的所有路径的流表，输入源 IP 和目的 IP 进行 Ping 测试,对每个交换机流表进行分析，筛选出转发过 Ping 数据包的流表即可知道哪些交换机进行了转发，由于整个网络的拓扑是知道的故可以确定主机访问的路径。

1) 运行验证程序

在终端输入 `sudo python RoutingControl/verification.py`, 按照提示输入 Floodlight 的绝对路径, `verification.py` 就是验证程序, 如图 4-11。

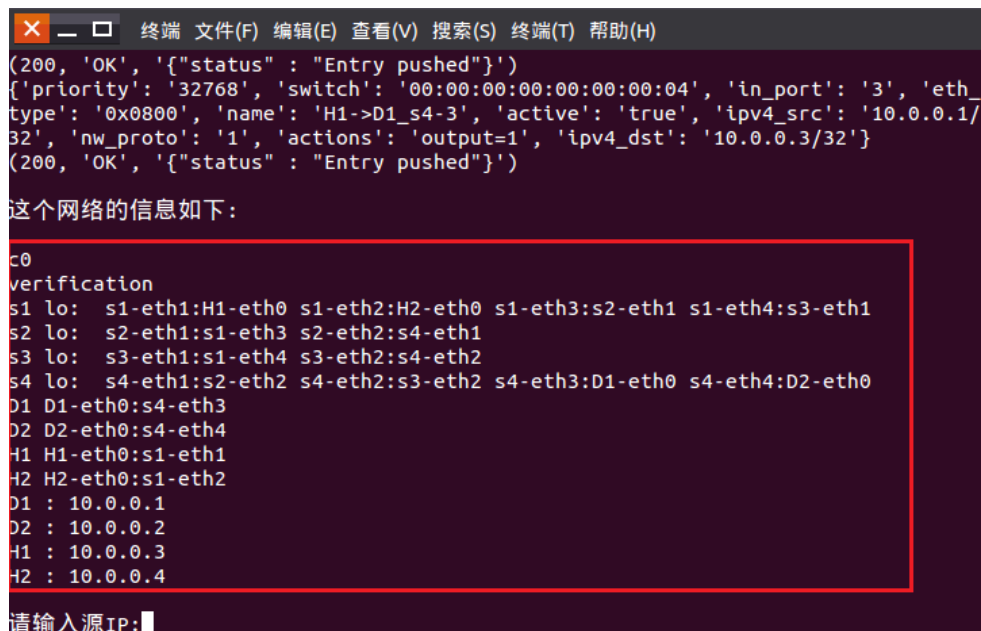


```
终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
pyy@pyy:~$ sudo python RoutingControl/verification.py
[sudo] pyy 的密码:
请输入floodlight的绝对路径: floodlight
Floodlight控制器正在启动中, 请稍等片刻.....
```

图 4-11 运行验证程序

验证程序先会打印出网络的链路信息和主机对应的 IP, 如图 4-12。

然后按照提示输入源 IP 和目的 IP。



```
终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(200, 'OK', '{"status": "Entry pushed"}')
{'priority': '32768', 'switch': '00:00:00:00:00:00:00:04', 'in_port': '3', 'eth_type': '0x0800', 'name': 'H1->D1_s4-3', 'active': 'true', 'ipv4_src': '10.0.0.1/32', 'nw_proto': '1', 'actions': 'output=1', 'ipv4_dst': '10.0.0.3/32'}
(200, 'OK', '{"status": "Entry pushed"}')

这个网络的信息如下:

c0
verification
s1 lo:  s1-eth1:H1-eth0 s1-eth2:H2-eth0 s1-eth3:s2-eth1 s1-eth4:s3-eth1
s2 lo:  s2-eth1:s1-eth3 s2-eth2:s4-eth1
s3 lo:  s3-eth1:s1-eth4 s3-eth2:s4-eth2
s4 lo:  s4-eth1:s2-eth2 s4-eth2:s3-eth2 s4-eth3:D1-eth0 s4-eth4:D2-eth0
D1 D1-eth0:s4-eth3
D2 D2-eth0:s4-eth4
H1 H1-eth0:s1-eth1
H2 H2-eth0:s1-eth2
D1 : 10.0.0.1
D2 : 10.0.0.2
H1 : 10.0.0.3
H2 : 10.0.0.4

请输入源IP: 
```

图 4-12 验证程序显示网络信息

2) 不同 IP 地址访问打印出结果

H1 访问 D1，打印出路径。

```
请输入源IP:10.0.0.3
请输入目的IP:10.0.0.1
```

图 4- 13 H1 访问 D1

```
根据上面每个交换机的流表信息我们可以看出源主机到目的主机的路径为：
路线:H1->s1->s2->s4->D1
回车,删除静态流表以免对下面的测试造成影响
```

图 4- 14 H1 访问 D1 打印路径

H1 访问 D2，打印出路径。

```
请输入源IP:10.0.0.3
请输入目的IP:10.0.0.2
```

图 4- 15 H1 访问 D2

```
根据上面每个交换机的流表信息我们可以看出源主机到目的主机的路径为：
路线:H1->s1->s3->s4->D2
回车,删除静态流表以免对下面的测试造成影响
```

图 4- 16 H1 访问 D2 结果

H2 访问 D1，打印出路径。

```
请输入源IP:10.0.0.4
请输入目的IP:10.0.0.1
```

图 4- 17 H2 访问 D1

根据上面每个交换机的流表信息我们可以看出源主机到目的主机的路径为：

路线:H2->s1->s3->s4->D1

回车,删除静态流表以免对下面的测试造成影响

图 4- 18 H2 访问 D1 打印路径

H2 访问 D2，打印出路径。

请输入源IP:10.0.0.4

请输入目的IP:10.0.0.2

图 4- 19 H2 访问 D2

根据上面每个交换机的流表信息我们可以看出源主机到目的主机的路径为：

路线:H2->s1->s2->s4->D2

回车,删除静态流表以免对下面的测试造成影响

图 4- 20 H2 访问 D2 打印路径

3) 根据流表打印出路径的过程（验证本题具体要求 2），以 H1→S1→S2→S4→D1 路径为例。

输入 H1 和 D1 的 IP 地址进行 Ping 前，程序会清空所有流表项，然后下发本题所有路由路径的流表。Ping 完之后，我们查看交换机 S1 流表，发现只有有两个流表有 IP 数据包转发记录，并且地址是 10.0.0.1（D1）和 10.0.0.3（H1），说明 S1 对 Ping 数据包进行了转发。

```
终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
puy@puy:~$ sudo ovs-ofctl dump-flows s1
[sudo] puy 的密码:
NXST_FLOW reply (xid=0x4):
 cookie=0xa0000004a6635d, duration=918.980s, table=0, n_packets=0, n_bytes=0, idle_age=918, ip,in_port=4,nw_src=10.0.0.1,nw_dst=10.0.0.4 actions=output:2
 cookie=0xa00000064ef0b5e, duration=918.980s, table=0, n_packets=0, n_bytes=0, idle_age=918, ip,in_port=2,nw_src=10.0.0.4,nw_dst=10.0.0.2 actions=output:3
 cookie=0xa00000064ef0b5f, duration=918.980s, table=0, n_packets=0, n_bytes=0, idle_age=918, ip,in_port=3,nw_src=10.0.0.2,nw_dst=10.0.0.4 actions=output:2
 cookie=0xa00000010fda92a, duration=918.917s, table=0, n_packets=0, n_bytes=0, idle_age=918, ip,in_port=1,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=output:4
 cookie=0xa00000010fda92d, duration=918.898s, table=0, n_packets=0, n_bytes=0, idle_age=918, ip,in_port=4,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=output:1
 cookie=0xa000000b0b50127, duration=918.828s, table=0, n_packets=4, n_bytes=392, idle_age=81, ip,in_port=1,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=output:3
 cookie=0xa000000b0b50129, duration=918.817s, table=0, n_packets=4, n_bytes=392, idle_age=81, ip,in_port=3,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=output:1
 cookie=0x0, duration=918.993s, table=0, n_packets=339, n_bytes=33542, idle_age=11, priority=0 actions=CONTROLLER:65535
puy@puy:~$
```

图 4-21 交换机 S1 的流表

同理查看 S2, S3, S4 的流表, 发现 S2 和 S4 都有和 S1 一样的 IP 数据包的转发记录, 而交换机 S3 没有 IP 数据包的转发记录。故可以知道此次 Ping 的数据包经过了交换机 S1、S2 和 S4, 而整个网络的拓扑是知道的, 所以可以确定路径 H1→S1→S2→S4→D1

```
puy@puy:~$ sudo ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
 cookie=0xa00000064efb946, duration=967.982s, table=0, n_packets=0, n_bytes=0, idle_age=968, ip,in_port=1,nw_src=10.0.0.4,nw_dst=10.0.0.2 actions=output:2
 cookie=0xa00000064efb947, duration=967.979s, table=0, n_packets=0, n_bytes=0, idle_age=968, ip,in_port=2,nw_src=10.0.0.2,nw_dst=10.0.0.4 actions=output:1
 cookie=0xa000000b0b5af10, duration=967.877s, table=0, n_packets=4, n_bytes=392, idle_age=131, ip,in_port=1,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=output:2
 cookie=0xa000000b0b5af11, duration=967.864s, table=0, n_packets=4, n_bytes=392, idle_age=131, ip,in_port=2,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=output:1
 cookie=0x0, duration=968.046s, table=0, n_packets=372, n_bytes=38152, idle_age=0, priority=0 actions=CONTROLLER:65535
```

图 4-22 交换机 S2 的流表

```

0, priority=0 actions=CONTROLLER:65535
pyy@pyy:~$ sudo ovs-ofctl dump-flows s3
NXST_FLOW reply (xid=0x4):
 cookie=0xa0000004a7bf2d, duration=1039.122s, table=0, n_packets=0, n_bytes=0, idle_age=1039, ip,in_port=2,nw_src=10.0.0.1,nw_dst=10.0.0.4 actions=output:1
 cookie=0xa0000004a7bf2c, duration=1039.121s, table=0, n_packets=0, n_bytes=0, idle_age=1039, ip,in_port=1,nw_src=10.0.0.4,nw_dst=10.0.0.1 actions=output:2
 cookie=0xa00000010ff04fc, duration=1039.092s, table=0, n_packets=0, n_bytes=0, idle_age=1039, ip,in_port=1,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=output:2
 cookie=0xa00000010ff04fd, duration=1039.079s, table=0, n_packets=0, n_bytes=0, idle_age=1039, ip,in_port=2,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=output:1
 cookie=0x0, duration=1039.196s, table=0, n_packets=357, n_bytes=35048, idle_age=11, priority=0 actions=CONTROLLER:65535

```

图 4-23 交换机 S3 的流表

```

=11, priority=0 actions=CONTROLLER:65535
pyy@pyy:~$ sudo ovs-ofctl dump-flows s4
NXST_FLOW reply (xid=0x4):
 cookie=0xa0000004a86d17, duration=1079.168s, table=0, n_packets=0, n_bytes=0, idle_age=1079, ip,in_port=3,nw_src=10.0.0.1,nw_dst=10.0.0.4 actions=output:2
 cookie=0xa0000004a86d16, duration=1079.168s, table=0, n_packets=0, n_bytes=0, idle_age=1079, ip,in_port=2,nw_src=10.0.0.4,nw_dst=10.0.0.1 actions=output:3
 cookie=0xa00000064f11518, duration=1079.144s, table=0, n_packets=0, n_bytes=0, idle_age=1079, ip,in_port=1,nw_src=10.0.0.4,nw_dst=10.0.0.2 actions=output:4
 cookie=0xa00000064f1151b, duration=1079.113s, table=0, n_packets=0, n_bytes=0, idle_age=1079, ip,in_port=4,nw_src=10.0.0.2,nw_dst=10.0.0.4 actions=output:1
 cookie=0xa00000010ffb2e6, duration=1079.049s, table=0, n_packets=0, n_bytes=0, idle_age=1079, ip,in_port=2,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=output:4
 cookie=0xa00000010ffb2e8, duration=1079.039s, table=0, n_packets=0, n_bytes=0, idle_age=1079, ip,in_port=4,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=output:2
 cookie=0xa000000b0b70ae2, duration=1078.971s, table=0, n_packets=4, n_bytes=392, idle_age=242, ip,in_port=1,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=output:3
 cookie=0xa000000b0b70ae4, duration=1078.949s, table=0, n_packets=4, n_bytes=392, idle_age=242, ip,in_port=3,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=output:1
 cookie=0x0, duration=1079.176s, table=0, n_packets=420, n_bytes=43962, idle_age=6, priority=0 actions=CONTROLLER:65535

```

图 4-24 交换机 S4 的流表

设计题

设计名称

QoS over LLDP: 基于 LLDP 协议的捎带式服务质量采集机制

具体问题描述

QoS 在网络中利用各种技术为网络提供更好的服务能力, 解决网络延迟和阻塞等问题, 确保重要的业务量不受延迟或者丢弃。而要部署 QoS 策略之前要先对网络中 QoS 信息进行采集, 目前广泛使用的 Floodlight 控制器上并没有采集服务质量信息的机制。在 SDN 网络中, 通过 LLDP 报文来进行链路发现, 控制器向交换机下发包含 LLDP 报文的 packet_out 消息, 交换机对收到的 LLDP 报文在相应的端口进行转发, 相邻的交换机收到 LLDP 报文后由于没有相应的流表项, 就将报文装在 packet_in 消息中发给控制器, 实现交换机邻接状态信息的获取, 即链路发现。基于以上现状, 我们设想在交换机收到控制器的 LLDP 报文时将自身的 QoS 信息夹带在报文里, 下一跳交换机会将带有 QoS 信息的 LLDP 报文发回给控制器, 这样在链路发现的过程中控制器就可以获得网络的 QoS 信息。现亟需解决的问题: 一是在交换机收到含 LLDP 报文的 packet_out 消息时, 将 QoS 信息夹带到 LLDP 报文中; 二是在 Floodlight 中添加查看 QoS 信息的模块。由于采用了 LLDP 作为 QoS 采集的承载协议, 我们将实现上述两方面功能的方案称之为 QoS over LLDP。

研究现状和意义

目前,在广泛使用的 Floodlight 控制其中并没有 QoS 信息采集的机制,开发和研究人员在控制器上基于当前的信息部署 QoS 策略。若控制器之前就可以知道整个网络任意局部的丢包、延迟、抖动和带宽等信息,在对不同的数据流提供不同服务质量、对相应设备提供转发优先级策略和实现各种拥塞控制的机制的时候,就有实时数据可以进行参考,这样一来为 SDN 网络部署 QoS 策略显得更简便,更有效率。同时,在使用 LLDP 报文传递 QoS 信息时,仅仅增加了原来的 LLDP 包对 QoS 信息描述的几个字节,网络开销也比较小。任何实现 OpenFlow 的协议都包含 LLDP 数据包处理模块,从而 QoS over LLDP 可方便地移植到不限于 Floodlight 各种控制器的环境中。如果没有在找到控制器模块对 QoS 处理,最坏的情况下仅是缺乏的 QoS 的 TLV 的处理机制,但不会导致与控制器的基本工作流程的任何冲突,因为所有控制器都支持 LLDP 处理。我们只需要添加一个处理 QoS TLV 的模块,使该控制器可以获取从定制的 LLDP 报文中的 QoS 信息。这样一来需要修改的源代码比较少,大大节省了开发者的工作量。

解决方案和实现思路

（一）整体思路

控制器向交换机下发带有 LLDP 报文的 packet_out 消息，交换机将收到的 LLDP 报文填入自身的 QoS 信息，再将其发送到相邻的交换机，相邻的交换机收到 LLDP 报文由于没有相应的流表进行匹配，就向控制器发送装有此 LLDP 报文的 packet_in 消息，这样控制器就获得了交换机的 QoS，在整个拓扑发现过程中控制器就可以获得所有交换机的 QoS 信息。整体框架如图 5-1。

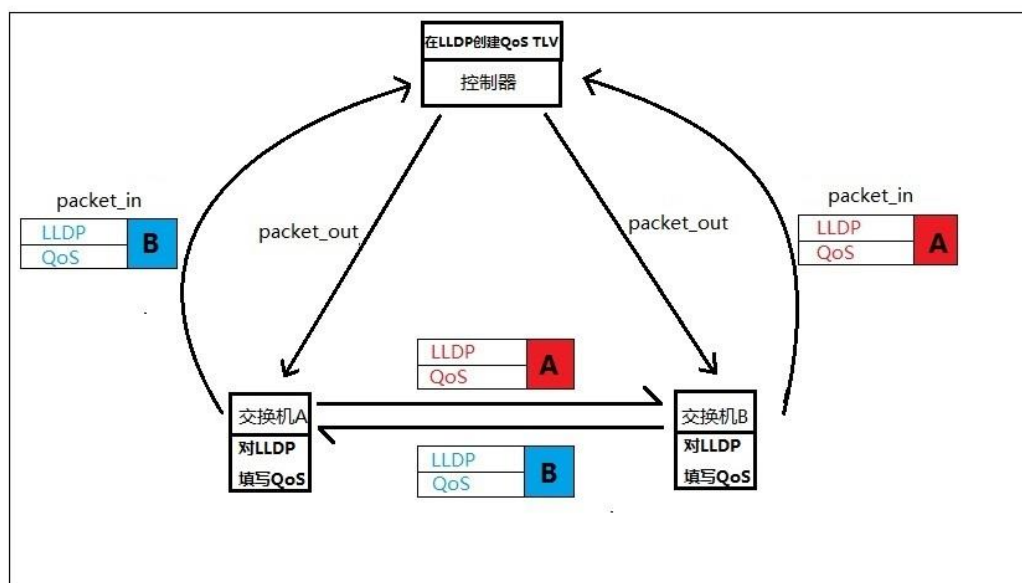


图 5- 1 QoS over LLDP 整体思路

（二）解决方案

LLDP 不同的信息放在 TLV(Type/Length/Value)中，我们可以将

QoS 信息作为 LLDP 数据包中自定义的 TLV。在 TLV Type 字段中，127 表示这是一个自定义的 TLV。TLV Length 字段指定该 TLV 可变长度值。Organization Code 字段表示这是设计者定制的 TLV。Subtype 字段指定了详细的子类型，这里设定子类型有 Bandwidth、DropRate、Delay 和 Jitter。Value String 字段给出了真正的值。根据不同的 Subtype，其 Value String 值的含义又有所不同。为了让接收器可以方便地解析不同的 QoS 信息，我们使用预定义的属性顺序和长度，如图 5-2。

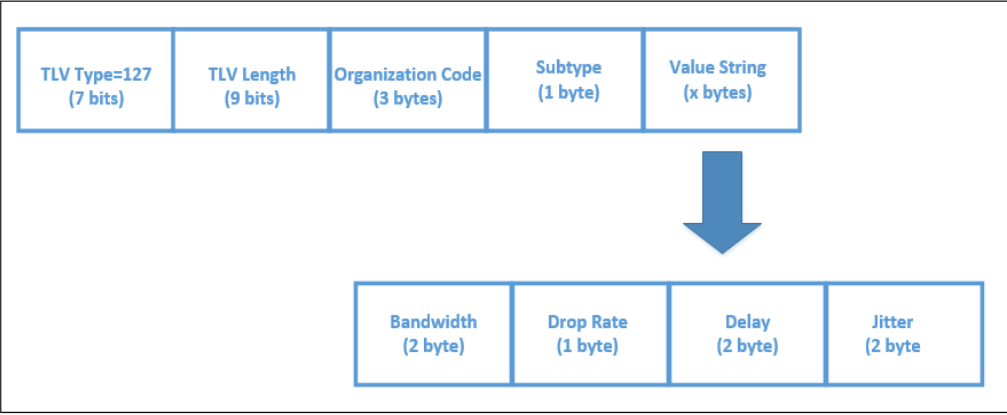


图 5- 2 QoS 的 TLV 字段

（三）抓包和控制器模块示例

捕获采集了交换机 QoS 信息的 LLDP 数据包，如图 5-3 所示，组织机构代码设置为 0xabcdef。这里子类型 Bandwidth 代码设为 100，值为 0x0064（100mbps）。子类型 Delay 代码设为 101，值为 0x000a（10ms）。子类型 Jitter 代码设为 102，值为 0x000a。DropRate 代码设为 103，值为 0x00。

1585	46.673561000	127.0.0.1	127.0.0.1	OpenFlow	214	Type: OFPT_PACKET_OUT
1587	46.673636000	5e:56:3d:cf:95:df		LLDP	108	TTL = 120
1588	46.673637000	5e:56:3d:cf:95:df		LLDP	108	TTL = 120
<div> <div>Unknown - Unknown (100) 带宽信息</div> <div> 1111 111. = TLV Type: Organization Specific (127) TLV类型 0000 0110 = TLV Length: 6 TLV长度 Organization Unique Code: Unknown (0xabcdef) 组织机构代码 Unknown Subtype: 100 子类型 Unknown Subtype Content: 0064 链路带宽值 </div> </div> <div> <div>Unknown - Unknown (101) 时延信息</div> <div> 1111 111. = TLV Type: Organization Specific (127) 0000 0110 = TLV Length: 6 Organization Unique Code: Unknown (0xabcdef) Unknown Subtype: 101 Unknown Subtype Content: 000a 链路时延值 </div> </div> <div> <div>Unknown - Unknown (102) 抖动信息</div> <div> 1111 111. = TLV Type: Organization Specific (127) 0000 0110 = TLV Length: 6 Organization Unique Code: Unknown (0xabcdef) Unknown Subtype: 102 Unknown Subtype Content: 000a 链路抖动值 </div> </div> <div> <div>Unknown - Unknown (103) 丢包率</div> <div> 1111 111. = TLV Type: Organization Specific (127) 0000 0101 = TLV Length: 5 Organization Unique Code: Unknown (0xabcdef) Unknown Subtype: 103 Unknown Subtype Content: 00 链路丢包率 </div> </div>						
0000 00 02 00 01 00 06 5e 56 3d cf 95 df 00 00 88 cc^V =..... 0010 02 07 04 00 00 00 00 02 04 03 02 00 02 06 02 0020 00 78 fe 06 ab cd ef 64 00 64 fe 06 ab cd ef 65 .x.....d .d.....e 0030 00 0a fe 06 ab cd ef 66 00 0a fe 05 ab cd ef 67f.....g						

图 5-3 捕获到含 LLDP 报文的 packet_out 消息

在控制器模块上通过自定义的 Rest API 查看采集到的 QoS 信息。

如图 5-4

```

X  _  终端 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
pyy@pyy:~$ curl http://127.0.0.1:8080/wm/lldp/qos
当前网络的各个接口qos的信息如下:
pyy@pyy:~$ curl http://127.0.0.1:8080/wm/lldp/qos
当前网络的各个接口qos的信息如下:
{ 机箱的ID:00:00:00:00:00:01,端口的ID:3,带宽:100 MB,延迟:10 s,抖动:10 s^2,丢包率:0% }
{ 机箱的ID:00:00:00:00:00:01,端口的ID:4,带宽:100 MB,延迟:10 s,抖动:10 s^2,丢包率:0% }
{ 机箱的ID:00:00:00:00:00:03,端口的ID:1,带宽:100 MB,延迟:10 s,抖动:10 s^2,丢包率:0% }
{ 机箱的ID:00:00:00:00:00:02,端口的ID:1,带宽:100 MB,延迟:10 s,抖动:10 s^2,丢包率:0% }
pyy@pyy:~$

```

图 5-4 rest API 显示 Qos 信息

（四）实现代码

(1) Floodlight 创建带有 QoS TLV 的 LLDP 报文

```
byte[] bandwidth = new byte[] { (byte) 0xab, (byte) 0xcd, (byte) 0xef, 0x64, 0x00, 0x64 }; //带宽100MB
byte[] delay = new byte[] { (byte) 0xab, (byte) 0xcd, (byte) 0xef, 0x65, 0x00, 0x0a }; //时延10s
byte[] jitter = new byte[] { (byte) 0xab, (byte) 0xcd, (byte) 0xef, 0x66, 0x00, 0x0a }; //抖动10
byte[] loss = new byte[] { (byte) 0xab, (byte) 0xcd, (byte) 0xef, 0x67, 0x00 }; //丢包率0%

LLDPTLV dpidtlv_bw = new LLDPTLV().setType((byte) 127)
    .setLength((short) bandwidth.length)
    .setValue(bandwidth);
LLDPTLV dpidtlv_delay = new LLDPTLV().setType((byte) 127)
    .setLength((short) delay.length)
    .setValue(delay);
LLDPTLV dpidtlv_jitter = new LLDPTLV().setType((byte) 127)
    .setLength((short) jitter.length)
    .setValue(jitter);
LLDPTLV dpidtlv_loss = new LLDPTLV().setType((byte) 127)
    .setLength((short) loss.length)
    .setValue(loss);
lldp.getOptionalTLVList().add(dpidtlv_bw);
lldp.getOptionalTLVList().add(dpidtlv_delay);
lldp.getOptionalTLVList().add(dpidtlv_jitter);
lldp.getOptionalTLVList().add(dpidtlv_loss);
```

(2) Floodlight 显示 LLDP 报文的 QoS 信息

创建全局变量 qosdatabase，用于存取网络中的 QoS 信息

```
/**
 * qos数据库
 * @author wjl
 */
public static Map<String, Qos> qosdatabase = new HashMap<String, Qos>();

/**
 * wjl modify
 */
Qos qos = new Qos();
//取出机箱的ID以String的形式存放
byte[] chassisid = lldp.getChassisId().getValue();
String strchassisid = new String();
int i;
String front, behind;
byte num = 15;
String[] arr = new String[]{"0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "a", "b", "c", "d", "e", "f"};
for(i = 1; i < 6; i++){
    behind = arr[chassisid[i]&num];
    front = arr[(chassisid[i]>>4&num)];
    strchassisid += front+behind+":";
}
behind = arr[chassisid[6]&num];
front = arr[(chassisid[6]>>4&num)];
strchassisid += front+behind;
qos.chassisid = strchassisid;
log.info("机箱的ID:"+strchassisid);
//取出对应机箱的端口的ID
int portid = lldp.getPortId().getValue()[1]+lldp.getPortId().getValue()[2];
qos.portid = portid;
log.info("端口的ID:"+portid);
```

在 Floodlight 控制器的 `net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager` 的 `handleLldp` 方法中, 为了获取每个 LLDP 数据包的 QoS 信息, 添加如下代码:

```

} else if( lldptlv.getLength() == 6 || lldptlv.getLength() == 5){
    if(lldptlv.getType() == 127
        && lldptlv.getValue()[0] == (byte)0xab
        && lldptlv.getValue()[1] == (byte)0xcd
        && lldptlv.getValue()[2] == (byte)0xef){
        if(lldptlv.getValue()[3] == 0x64){
            int bandwidth = lldptlv.getValue()[4] + lldptlv.getValue()[5];
            qos.bandwidth = bandwidth;
            log.info("带宽:"+bandwidth+" MB");
        }else if(lldptlv.getValue()[3] == 0x65){
            int delay = lldptlv.getValue()[4] + lldptlv.getValue()[5];
            qos.delay = delay;
            log.info("延迟:"+delay+" S");
        }else if(lldptlv.getValue()[3] == 0x66){
            int jitter = lldptlv.getValue()[4] + lldptlv.getValue()[5];
            qos.jitter = jitter;
            log.info("抖动:"+jitter+" S^2");
        }else if(lldptlv.getValue()[3] == 0x67){
            int loss = 0x00+lldptlv.getValue()[4];
            qos.loss = loss;
            log.info("丢包率:"+loss+" %");
        }
    }
}

qosdatabase.put(qos.chassisid+qos.portid, qos);

```

附件中 `displaylldp` 文件夹是一个 floodlight 的模块, `qos.java` 放在 `net.floodlightcontroller.linkdiscovery.internal` 下面供 Floodlight 控制器加载, 这个类就是 QoS 信息在控制器里面的存储结构。

(3) OVS 交换机在收到的 LLDP 报文中填入 QoS 信息

OVS 交换机对收到的 LLDP 进行解码, 然后获取 LLDP TLV 的 QoS 信息, 具体修改在 `ovswitch` 的源码根目录的 `lib/lldp/lldp.c` 的文件的 `decode_lldp` 的方法, 添加的代码如下:


```

switch(tlv_subtype){
case LLDP_TLV_BANDWIDTH_SUBTYPE:
CHECK_TLV_SIZE(2, "bandwidth");
chassis->bandwidth = PEEK_UINT16;
break;
case LLDP_TLV_DELAY_SUBTYPE:
CHECK_TLV_SIZE(2, "delay");
chassis->delay = PEEK_UINT16;
break;
case LLDP_TLV_JITTER_SUBTYPE:
CHECK_TLV_SIZE(2, "jitter");
chassis->jitter = PEEK_UINT16;
break;
case LLDP_TLV_LOSS_SUBTYPE:
CHECK_TLV_SIZE(1, "loss");
chassis->loss = PEEK_UINT8;
break;

```

在发送 LLDP 的时候，依次把 QoS 信息填充到指定的 TLV 字段里面，具体修改在 ovsch 的源码根目录的/lib/lldp/lldp.c 的文件的 send_lldp 的方法，添加的代码如下：

```

/* bandwidth */
lldp_tlv_start(p,LLDP_TLV_ORG, &start);
lldp_tlv_put_u16(p, 0xabcd);
lldp_tlv_put_u8(p, 0xef);
lldp_tlv_put_u8(p, LLDP_TLV_BANDWIDTH_SUBTYPE);
lldp_tlv_put_u16(p, chassis->bandwidth);
lldp_tlv_end(p, start);

/* delay */
lldp_tlv_start(p,LLDP_TLV_ORG, &start);
lldp_tlv_put_u16(p, 0xabcd);
lldp_tlv_put_u8(p, 0xef);
lldp_tlv_put_u8(p, LLDP_TLV_DELAY_SUBTYPE);
lldp_tlv_put_u16(p, chassis->delay);
lldp_tlv_end(p, start);

/* jitter */
lldp_tlv_start(p,LLDP_TLV_ORG, &start);
lldp_tlv_put_u16(p, 0xabcd);
lldp_tlv_put_u8(p, 0xef);
lldp_tlv_put_u8(p, LLDP_TLV_JITTER_SUBTYPE);
lldp_tlv_put_u16(p, chassis->jitter);
lldp_tlv_end(p, start);

/* loss */
lldp_tlv_start(p, LLDP_TLV_ORG, &start);
lldp_tlv_put_u16(p, 0xabcd);
lldp_tlv_put_u8(p, 0xef);
lldp_tlv_put_u8(p, LLDP_TLV_LOSS_SUBTYPE);
lldp_tlv_put_u8(p, chassis->loss);
lldp_tlv_end(p, start);

```

参考文献

- [1]Thomas D.Nadeau&Ken Gray.SDN 与 OpenFlow 解析[M].北京:人民邮电出版社, 2014.5
- [2]Magnus Lie Hetland.Python 基础教程(第 2 版.修订版) [M].北京:人民邮电出版社, 2014
- [3]黄韬,刘江,魏亮,张娇,刘韵洁. 软件定义网络核心原理与应用实践[M]. 北京:人民邮电出版社,2014.9
- [4]Siamak Azodolmolky.软件定义网络基于 OpenFlow 的 SDN 技术揭秘[M].北京:机械工业出版社, 2014.6
- [5]华为技术有限公司. LLDP 技术白皮书[Z]. 01, 2012.10.31
- [6]雷葆华, 王峰, 王茜, 王和宇. SDN 核心技术剖析和实战指南[M].北京: 电子工业出版社, 2013.9
- [7] Weslly J.Chun. Python 核心编程(第二版)[M]. 北京: 人民邮电出版社, 2008.7