第三章 网络环境搭建

3.1 安装 NOX 控制器及 GUI

3.1.1 Linux 操作系统-Ubuntu10.04 及配置

首先是操作系统的选择，经过反复测试只在 Ubuntu 10.04.4-sever-amd64 上安装成功，在 centos-6.5、centos-6.4、ubuntu-11 上安装均失败（因为一些依赖包的关系），所以建议，在选择 Linux 的版本上尽量采用 Debian Lenny 或者 Ubuntu（官方开发是基于 Debian 系列）。因此本文的 NOX 控制器是安装在 Ubuntu10.04 上的，并且 Ubuntu10.04 是安装在 VMware Workstation 虚拟机软件里的。

为 Ubuntu10.04 的 eth0 配置静态的 IP，命令如下：

vi /etc/network/interfaces

编辑以下内容，然后保存退出：

auto lo

iface lo inet loopback

auto eth0

iface eth0 inet static

address 192.168.100.100

netmask 255.255.255.0

重启 network 使配置生效：

/etc/init.d/networking restart

配置成功后可以查看 eth0，如图 3.1 所示：

```
root@zhangxu:/home/zhangxu# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:03:e6:be
          inet addr:192.168.100.100  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: 2001:da8:9000:a441:20c:29ff:fe03:e6be/64 Scope:Global
          inet6 addr: fe80::20c:29ff:fe03:e6be/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2689 errors:0 dropped:0 overruns:0 frame:0
          TX packets:82 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:256164 (256.1 KB)  TX bytes:12198 (12.1 KB)
```

图 3.1 网卡信息

3.1.2 安装控制器 NOX

控制器 NOX 的安装步骤描述如下：

（1）控制器 NOX 的依赖包的安装。

1）下载 NOX 的开源代码：

git clone git://noxrepo.org/nox

2）依赖包的获取：

wget http://openflowswitch.org/downloads/debian/binary/nox-dependencies.deb

3）解压缩：

dpkg --info nox-dependencies.deb

4）通过下面的命令 apt-get 安装依赖包：

cd /etc/apt/sources.list.d

sudo wget http://openflowswitch.org/downloads/debian/nox.list

sudo apt-get update

sudo apt-get install nox-dependencies

5）编译核心功能具体需要安装的依赖包有：

apt-get install autoconf automake g++ libtool python python-twisted swig libboost1.35-dev libxerces-c2-dev libssl-dev make

如果安装 libboost1.35-dev 这个依赖包有问题可以执行下面的语句：

apt-get install libboost-serialization-dev libboost-all-dev

6）编译全部功能还需要执行：

apt-get install libsqlite3-dev python-simplejson

7）生成文档需要执行：

apt-get install python-sphinx

所需的安装包到此都装完了。

（2）编译，依次执行以下的命令。

cd nox

./boot.sh

mkdir build/

cd build/

../configure --with-python=yes

make（编译时间会很长，可以使用 make -j 5）

（3）完成编译后的 NOX 控制器文件如图 3.2 所示。

```
root@zhangxu:/home/zhangxu/nox# ls
acinclude.m4    boot.sh      config.h.in~     COPYING  Makefile.am  README
aclocal.m4      build        configure        doc      Makefile.in  src
AUTHORS         config       configure.ac     INSTALL  man
autom4te.cache  config.h.in  configure.ac.in  LICENSE  package
```

图 3.2 NOX 文件目录

如果确保无需 C++ STL 的 debuging 检查，可以使用下面的命令来关闭该检查以获取更快的安装速度。

./configure --with-python=’which python2.5’ --enable-ndebug

注意编译完成后的可执行代码是在/build/src 下，不是在原先的源代码文件 src/下，可执行的代码名是 nox_core，如图 3.3 所示。

```
root@zhangxu:/home/zhangxu/nox/build/src# ls
builtin            etc      Makefile  nox.info    nox.xsd.o
components.xsd.cc  include  nox       nox_main.o  tests
components.xsd.o   lib      nox_core  nox.xsd.cc  utilities
```

图 3.3 nox_core

如果 NOX 安装成功，执行./nox_core -h 会显示如图 3.4 所示的帮助信息。

```
root@zhangxu:/home/zhangxu/nox/build/src# ./nox_core -h
nox_core: nox runtime
usage: nox_core [OPTIONS] [APP[=ARG[,ARG]...]] [APP[=ARG[,ARG]...]]...

Interface options (specify any number):
  -i nl:DP_ID             via netlink to local datapath DP_IDX
  -i ptcp:[IP]:[PORT]     listen to TCP PORT on interface specified by IP
                          (default: 0.0.0.0:6633)
  -i pssl:[IP]:[PORT]:KEY:CERT:CONTROLLER_CA_CERT
                          listen to SSL PORT on interface specified by IP
                          (default: 0.0.0.0:6633)
  -i pcap:FILE[:OUTFILE]  via pcap from FILE (for testing) write to OUTFILE
  -i pcapt:FILE[:OUTFILE] same as "pcap", but delay packets based on pcap timest
amps
  -i pgen:                continuously generate packet-in events

Network control options (must also specify an interface):
  -u, --unreliable        do not reconnect to interfaces on error

Leak checking options:
  --check-leaks=FILE      log memory allocation calls to FILE
  --leak-limit=SIZE       log to FILE at most SIZE bytes

Other options:
  -c, --conf=FILE         set configuration file
  -d, --daemon            become a daemon
  -l, --libdir=DIRECTORY  add a directory to the search path for application lib
raries
  -p, --pid=FILE          set pid file
  -n, --info=FILE         set controller info file
  -v, --verbose           make console log verbose (shows INFO messages -- use t
wice for DBG)
  -v, --verbose=CONFIG    configure verbosity
  -h, --help              display this help message
  -V, --version           display version information
```

图 3.4 帮助信息

3.1.3 安装 NOX 的 GUI

GUI 的安装步骤描述如下：

（1）图形界面 gnome 的安装：

1）因为安装的 Linux 是 Ubuntu 10.04.4-sever-amd64，服务器版本默认没有图形界面，所以在运行 NOX 的 GUI 之前，需要安装图形界面 gnome。方法如下：

sudo apt-get install xinit

安装完成，重启，运行 startx，终端由黑色界面变成白底黑字。出现 X 型的鼠标指针。接着还需要安装 gnome 的桌面套件。

2）安装环境管理器：

sudo apt-get install gdm

3）安装桌面环境：

sudo apt-get install ubuntu-desktop（下载会很久）

重启完成 gnome 的安装。

（2）NOX 的 GUI 安装：

1）安装 GUI 首先要安装 QT 依赖：

apt-get install python-qt4 python-simplejson

2）之后安装这个命令库否则不能运行：

apt-get install python-qt4-sql

3）在源代码 nox/src 目录下，启动 GUI：

./nox-gui.py

4）如果在这里提示 Qsql 数据库不能启动，还需要执行：

apt-get install libqt4-sql-sqlite

此时 GUI 仍不能启动还需要另起一个终端，这里启动 NOX 并监听 6633 端口（OpenFlow 协议默认端口）的信息，切换到/home/zhangxu/nox/build/src 下，运行命令：

./nox_core -v -i ptcp:6633 monitoring

运行上述命令后的视图如图 3.5 所示。

```
root@zhangxu:/home/zhangxu/nox/build/src# ./nox_core -v -i ptcp:6633 monitoring
00001|nox|INFO:Starting nox_core (/home/zhangxu/nox/build/src/.libs/lt-nox_core)
00002|nox|INFO:nox bootstrap complete
```

图 3.5 监听

5）启动之后的效果如图 3.6 所示。

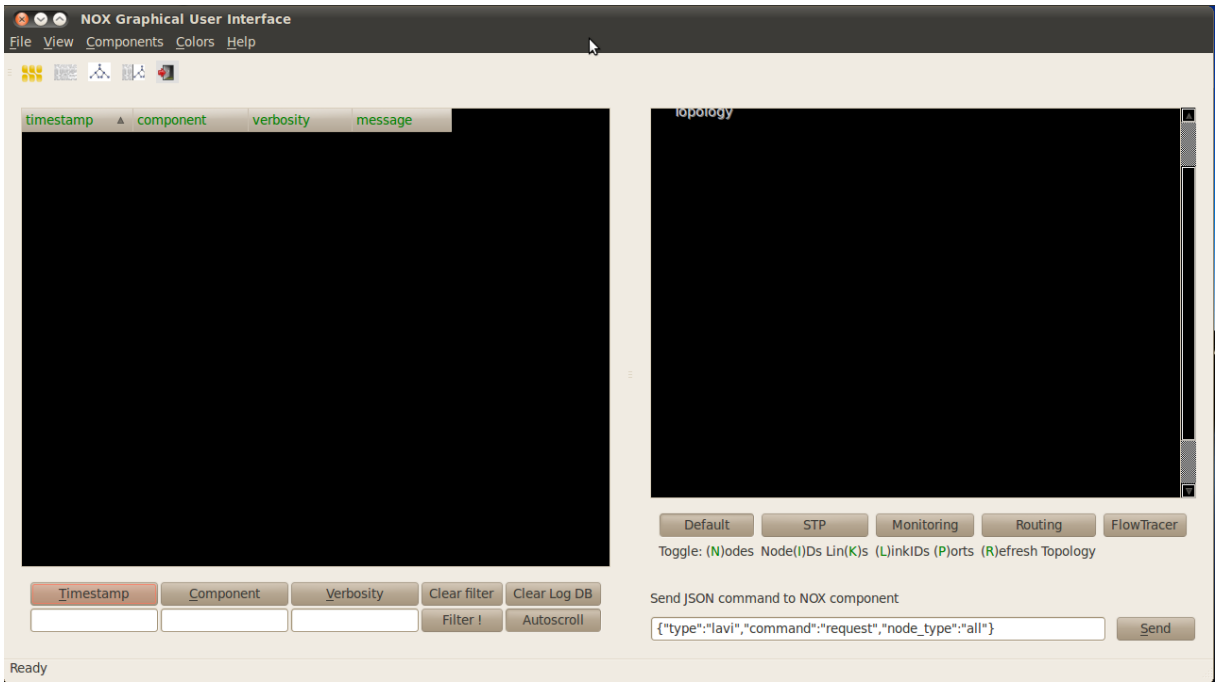

图 3.6 GUI

3.2 安装开放的虚拟交换机 OpenvSwitch

3.2.1 Linux 操作系统-Ubuntu12.04 及配置

经过反复测试，将 OpenvSwitch1.9.0 安装在 Ubuntu12.04 上。安装 OpenvSwitch 的台式机上有 3 个网卡，分别为 3 个网卡配置静态的 IP，即为 Ubuntu12.04 的 eth0、eth1 以及 eth2 配置静态的 IP，命令如下：

vi /etc/network/interfaces

编辑以下内容，然后保存退出：

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
address 192.168.100.10
netmask 255.255.255.0
auto eth1
iface eth1 inet static
address 192.168.100.11
netmask 255.255.255.0
auto eth2
iface eth2 inet static
```

address 192.168.100.12

netmask 255.255.255.0

重启 network 使配置生效：

/etc/init.d/networking restart

配置成功后可以查看 eth0、eth1、eth2，如图 3.7 所示。

```
root@ubuntu:/home/hpc# ifconfig
eth0      Link encap:Ethernet  HWaddr 90:fb:a6:15:d8:4c
          inet addr:192.168.100.10  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::92fb:a6ff:fe15:d84c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2233 errors:0 dropped:0 overruns:0 frame:0
          TX packets:87 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:230182 (230.1 KB)  TX bytes:11146 (11.1 KB)
          Interrupt:17

eth1      Link encap:Ethernet  HWaddr 00:e0:4c:3b:98:d5
          inet addr:192.168.100.11  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::2e0:4cff:fe3b:98d5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4177 errors:0 dropped:0 overruns:0 frame:0
          TX packets:52 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:355249 (355.2 KB)  TX bytes:7832 (7.8 KB)
          Interrupt:20 Base address:0xe800

eth2      Link encap:Ethernet  HWaddr 00:e0:4c:3b:4f:79
          inet addr:192.168.100.12  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::2e0:4cff:fe3b:4f79/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1750 errors:0 dropped:0 overruns:0 frame:0
          TX packets:58 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:167406 (167.4 KB)  TX bytes:8947 (8.9 KB)
          Interrupt:21 Base address:0xe400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:4560 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4560 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:367312 (367.3 KB)  TX bytes:367312 (367.3 KB)
```

图 3.7 网卡信息

3.2.2 安装 OpenvSwitch

OpenvSwitch 的安装步骤描述如下：

（1）依赖包的安装：

apt-get install build-essential

apt-get install openssl

在 Linux 确保/dev/urandom 存在，为了支持 TAP 服务，确保/dev/net/tun 存在，如图 3.8 所示。

执行命令：

ls -l /dev/urandom

ls -l /dev/net/tun

```
root@ubuntu:/home/hpc# ls -l /dev/urandom
crw-rw-rw- 1 root root 1, 9 May 15 21:20 /dev/urandom
root@ubuntu:/home/hpc# ls -l /dev/net/tun
crw-rw-rwT 1 root root 10, 200 May 15 21:20 /dev/net/tun
```

图 3.8 urandom 和 tun

（2）Openvswitch 包获取：

wget http://openvswitch.org/releases/openvswitch-1.9.0.tar.gz

（3）解压：

tar -xzf openvswitch-1.9.0.tar.gz

（4）安装：

cd openvswitch-1.9.0/

./configure --with-linux=/lib/modules/`uname -r`/build

make

make install

insmod datapath/linux/openvswitch.ko

编译成功之后的目录文件如图 3.9 所示。

```
root@ubuntu:/home/hpc# cd openvswitch-1.9.0
root@ubuntu:/home/hpc/openvswitch-1.9.0# ls
acinclude.m4    debian             IntegrationGuide  python
aclocal.m4      DESIGN             lib               README
AUTHORS         distfiles          m4                README-gcov
boot.sh         FAQ                Makefile          REPORTING-BUGS
build-aux       include            Makefile.am       rhel
CodingStyle     INSTALL            Makefile.in       stamp-h1
config.h        INSTALL.bridge     manpage-check     SubmittingPatches
config.h.in     INSTALL.Fedora     manpages.mk       tests
config.log      INSTALL.KVM        NEWS              third-party
config.status   INSTALL.Libvirt    NOTICE            utilities
configure       INSTALL.RHEL       ofproto           vswitchd
configure.ac    INSTALL.SSL        ovsdb             WHY-OVS
COPYING         INSTALL.userspace  package.m4        xenserver
datapath        INSTALL.XenServer  PORTING
```

图 3.9 OpenvSwitch 文件目录

3.2.3 配置 OpenvSwitch

配置 OpenvSwitch 的步骤描述如下：

（1）建立 OpenvSwitch 的配置文件和数据库：

mkdir -p /usr/local/etc/openvswitch

ovsdb-tool create /usr/local/etc/openvswitch/conf.db

/usr/local/share/openvswitch/vswitch.ovsschema

（2）启动配置数据库：

ovsdb-server

--remote=punix:/usr/local/var/run/openvswitch/db.sock

--remote=db:Open_vSwitch,manager_options --private-key=db:SSL,private_key

--certificate=db:SSL,certificate --bootstrap-ca-cert=db:SSL,ca_cert

--pidfile --detach

查看启动情况，如图 3.10 所示为启动成功的情况。

```
root@ubuntu:/home/hpc# ps -ef |grep ovsdb-server
root      2388     1  0 21:42 ?        00:00:01 ovsdb-server --remote=punix:/usr/local/
var/run/openvswitch/db.sock --remote=db:Open_vSwitch,manager_options --private-key=db:S
SL,private_key --certificate=db:SSL,certificate --bootstrap-ca-cert=db:SSL,ca_cert --pi
dfile --detach
root      2789  1967  0 22:41 pts/0    00:00:00 grep --color=auto ovsdb-server
root@ubuntu:/home/hpc# ps -e |grep ovsdb-server
 2388 ?        00:00:01 ovsdb-server
```

图 3.10 数据库启动成功

（3）初始化数据库：

ovs-vsctl --no-wait init

（4）启动 OpenvSwitch daemon，连接到同样的 Unix domain socket 上：

ovs-vswitchd --pidfile --detach

启动成功后，会有三个进程：一个 ovs_workq 进程，一个 ovs-vswitchd 进程，一个 ovsdb-server 进程，如图 3.11 所示。

```
root@ubuntu:/home/hpc# ps -ef |grep ovs
root      2386     2  0 21:42 ?        00:00:00 [ovs_workq]
root      2388     1  0 21:42 ?        00:00:01 ovsdb-server --remote=punix:/usr/local/
var/run/openvswitch/db.sock --remote=db:Open_vSwitch,manager_options --private-key=db:S
SL,private_key --certificate=db:SSL,certificate --bootstrap-ca-cert=db:SSL,ca_cert --pi
dfile --detach
root      2391     1  0 21:42 ?        00:00:00 ovs-vswitchd --pidfile --detach
root      2392  2391  0 21:42 ?        00:00:00 ovs-vswitchd: worker process...
root      2814  1967  0 22:43 pts/0    00:00:00 grep --color=auto ovs
root@ubuntu:/home/hpc# ps -e |grep ovs
 2386 ?        00:00:00 ovs_workq
 2388 ?        00:00:01 ovsdb-server
 2391 ?        00:00:00 ovs-vswitchd
 2392 ?        00:00:00 ovs-vswitchd
```

图 3.11 OpenvSwitch 启动成功

3.3 搭建网络拓扑

3.3.1 总体网络拓扑结构

搭建网络的环境说明如下。使用 3 台台式机，1 台笔记本。在笔记本上的 VMware Workstation（虚拟机）中安装 Ubuntu10.04，并在 Ubuntu10.04 上安装 NOX 控制器。在其中的一台台式机上直接安装 Ubuntu12.04，并在 Ubuntu12.04 上安装 OpenvSwitch。另外在这台台式机上又额外安装了两块网卡，这样这台台式机一共有 3 块网卡。另外两台

台式机直接充当主机。网络结构示意图如图 3.12 所示。

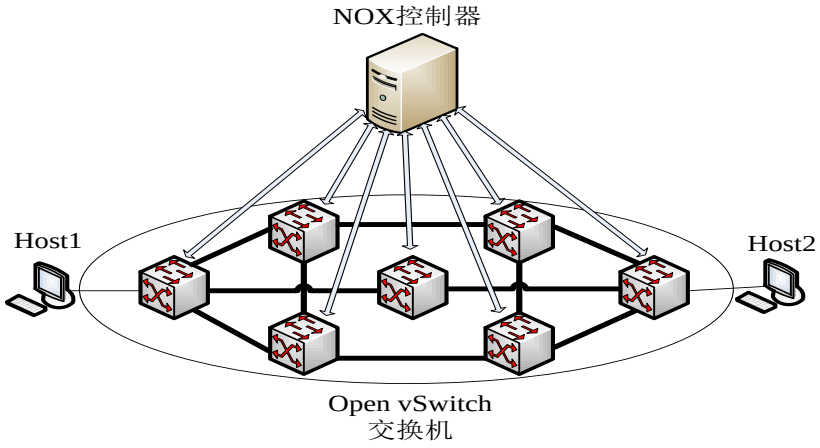

图 3.12 网络结构示意图

3.3.2 虚拟网络构建

在有 3 个网卡的台式机上用 Open vSwitch 构建出一个虚拟的网络拓扑。首先用一根网线连接网卡 eth0 口和笔记本上的 NOX 控制器端网卡，再用一根网线连接网卡 eth1 和主机 host1，最后用一根网线连接网卡 eth2 和主机 host2。实际搭建出来的拓扑如图 3.13 所示。

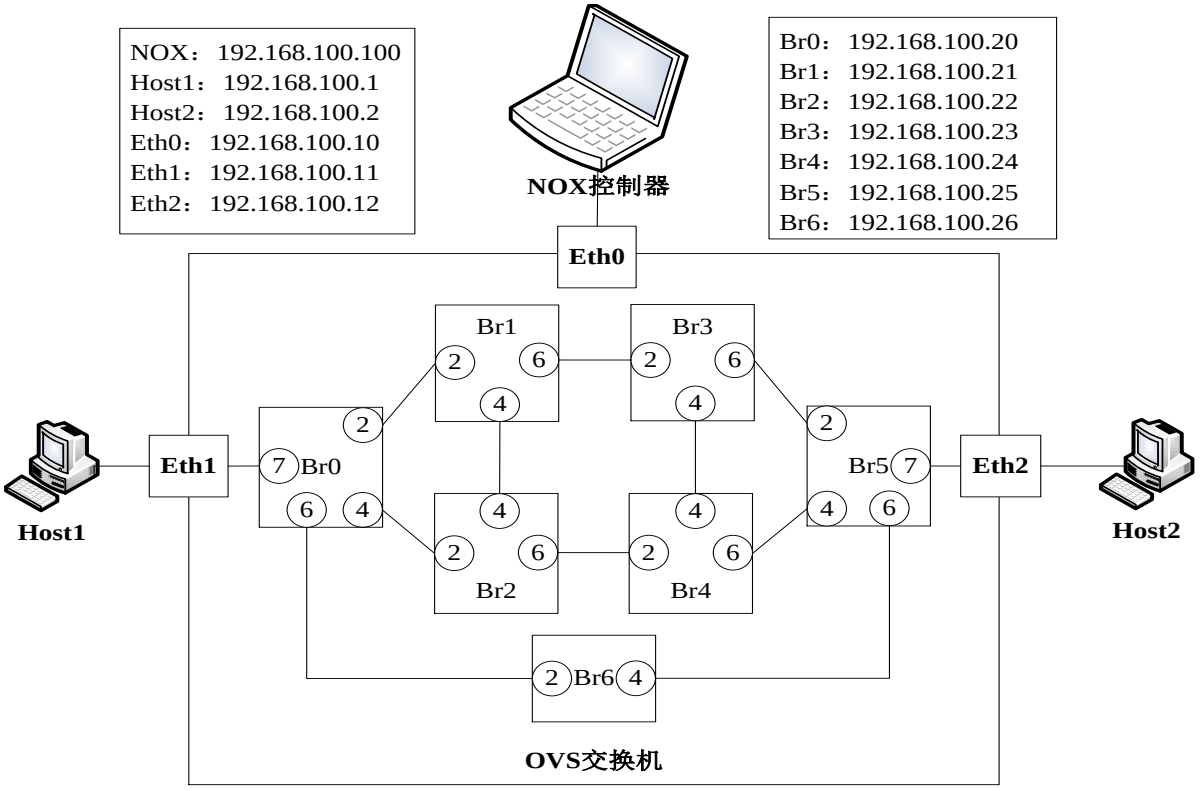

图 3.13 网络拓扑

网络搭建的步骤描述如下：

（1）首先启动 OpenvSwitch：

```
cd openvswitch-1.9.0
insmod datapath/linux/openvswitch.ko
cd ..
ovsdb-server
--remote=punix:/usr/local/var/run/openvswitch/db.sock
--remote=db:Open_vSwitch,manager_options --private-key=db:SSL,private_key
--certificate=db:SSL,certificate --bootstrap-ca-cert=db:SSL,ca_cert
--pidfile --detach
ovs-vsctl --no-wait init
ovs-vswitchd --pidfile --detach
```

（2）创建 7 个网桥，分别为 br0，br1，br2，br3，br4，br5，br6：

```
ovs-vsctl add-br br0
ovs-vsctl add-br br1
ovs-vsctl add-br br2
ovs-vsctl add-br br3
ovs-vsctl add-br br4
ovs-vsctl add-br br5
ovs-vsctl add-br br6
```

（3）分别为每个网桥分配 IP 地址：

```
ifconfig br0 192.168.100.20/24
ifconfig br1 192.168.100.21/24
ifconfig br2 192.168.100.22/24
ifconfig br3 192.168.100.23/24
ifconfig br4 192.168.100.24/24
ifconfig br5 192.168.100.25/24
ifconfig br6 192.168.100.26/24
```

（4）创建 br0 的端口：

```
ovs-vsctl add-port br0 patch-0-1
ovs-vsctl set interface patch-0-1 type=patch
ovs-vsctl set interface patch-0-1 options:peer=patch-1-0
ovs-vsctl add-port br0 patch-0-2
```

```
ovs-vsctl set interface patch-0-2 type=patch
ovs-vsctl set interface patch-0-2 options:peer=patch-2-0
ovs-vsctl add-port br0 patch-0-6
ovs-vsctl set interface patch-0-6 type=patch
ovs-vsctl set interface patch-0-6 options:peer=patch-6-0
```

（5）创建 br1 的端口：

```
ovs-vsctl add-port br1 patch-1-0
ovs-vsctl set interface patch-1-0 type=patch
ovs-vsctl set interface patch-1-0 options:peer=patch-0-1
ovs-vsctl add-port br1 patch-1-2
ovs-vsctl set interface patch-1-2 type=patch
ovs-vsctl set interface patch-1-2 options:peer=patch-2-1
ovs-vsctl add-port br1 patch-1-3
ovs-vsctl set interface patch-1-3 type=patch
ovs-vsctl set interface patch-1-3 options:peer=patch-3-1
```

（6）创建 br2 的端口：

```
ovs-vsctl add-port br2 patch-2-0
ovs-vsctl set interface patch-2-0 type=patch
ovs-vsctl set interface patch-2-0 options:peer=patch-0-2
ovs-vsctl add-port br2 patch-2-1
ovs-vsctl set interface patch-2-1 type=patch
ovs-vsctl set interface patch-2-1 options:peer=patch-1-2
ovs-vsctl add-port br2 patch-2-4
ovs-vsctl set interface patch-2-4 type=patch
ovs-vsctl set interface patch-2-4 options:peer=patch-4-2
```

（7）创建 br3 的端口：

```
ovs-vsctl add-port br3 patch-3-1
ovs-vsctl set interface patch-3-1 type=patch
ovs-vsctl set interface patch-3-1 options:peer=patch-1-3
ovs-vsctl add-port br3 patch-3-4
ovs-vsctl set interface patch-3-4 type=patch
ovs-vsctl set interface patch-3-4 options:peer=patch-4-3
ovs-vsctl add-port br3 patch-3-5
```

ovs-vsctl set interface patch-3-5 type=patch

ovs-vsctl set interface patch-3-5 options:peer=patch-5-3

（8）创建 br4 的端口：

ovs-vsctl add-port br4 patch-4-2

ovs-vsctl set interface patch-4-2 type=patch

ovs-vsctl set interface patch-4-2 options:peer=patch-2-4

ovs-vsctl add-port br4 patch-4-3

ovs-vsctl set interface patch-4-3 type=patch

ovs-vsctl set interface patch-4-3 options:peer=patch-3-4

ovs-vsctl add-port br4 patch-4-5

ovs-vsctl set interface patch-4-5 type=patch

ovs-vsctl set interface patch-4-5 options:peer=patch-5-4

（9）创建 br5 的端口：

ovs-vsctl add-port br5 patch-5-3

ovs-vsctl set interface patch-5-3 type=patch

ovs-vsctl set interface patch-5-3 options:peer=patch-3-5

ovs-vsctl add-port br5 patch-5-4

ovs-vsctl set interface patch-5-4 type=patch

ovs-vsctl set interface patch-5-4 options:peer=patch-4-5

ovs-vsctl add-port br5 patch-5-6

ovs-vsctl set interface patch-5-6 type=patch

ovs-vsctl set interface patch-5-6 options:peer=patch-6-5

（10）创建 br6 的端口：

ovs-vsctl add-port br6 patch-6-0

ovs-vsctl set interface patch-6-0 type=patch

ovs-vsctl set interface patch-6-0 options:peer=patch-0-6

ovs-vsctl add-port br6 patch-6-5

ovs-vsctl set interface patch-6-5 type=patch

ovs-vsctl set interface patch-6-5 options:peer=patch-5-6

（11）把 eth1 加到 br0 下，把 eth2 加到 br5 下：

ovs-vsctl add-port br0 eth1

ovs-vsctl add-port br5 eth2

（12）连接 NOX 控制器。在连接 NOX 控制器之前，NOX 控制器要处于监听 6633 端口

的状态：

ovs-vsctl set-controller br0 tcp:192.168.100.100:6633

ovs-vsctl set-controller br1 tcp:192.168.100.100:6633

ovs-vsctl set-controller br2 tcp:192.168.100.100:6633

ovs-vsctl set-controller br3 tcp:192.168.100.100:6633

ovs-vsctl set-controller br4 tcp:192.168.100.100:6633

ovs-vsctl set-controller br5 tcp:192.168.100.100:6633

ovs-vsctl set-controller br6 tcp:192.168.100.100:6633

当连接上 NOX 控制器时，执行以下命令会显示如图 3.14 的结果。

```
root@ubuntu:/home/hpc# netstat -npt |grep 6633
tcp        0      0 192.168.100.10:45607    192.168.100.100:6633    ESTABLISHED 2391/ovs-vswitchd
tcp        0      0 192.168.100.10:45605    192.168.100.100:6633    ESTABLISHED 2391/ovs-vswitchd
tcp        0      0 192.168.100.10:45608    192.168.100.100:6633    ESTABLISHED 2391/ovs-vswitchd
tcp        0      0 192.168.100.10:45603    192.168.100.100:6633    ESTABLISHED 2391/ovs-vswitchd
tcp        0      0 192.168.100.10:45604    192.168.100.100:6633    ESTABLISHED 2391/ovs-vswitchd
tcp        0      0 192.168.100.10:45606    192.168.100.100:6633    ESTABLISHED 2391/ovs-vswitchd
tcp        0      0 192.168.100.10:45609    192.168.100.100:6633    ESTABLISHED 2391/ovs-vswitchd
```

图 3.14 连接状态

3.3.3 NOX 控制器启动

首先在 NOX 端监听 6633 端口，并且加载 NOX 的路由模块和生成树模块，同时启动 NOX 的 GUI。命令如下：

cd nox/build/src

./nox_core -v -i ptcp:6633 monitoring routing spanning_tree

启动之后，如图 3.15 所示。

```
root@zhangxu:/home/zhangxu/nox/build/src# ./nox_core -v -i ptcp:6633 monitoring
routing spanning_tree
00001|nox|INFO:Starting nox_core (/home/zhangxu/nox/build/src/.libs/lt-nox_core)
00002|nox|INFO:nox bootstrap complete
```

图 3.15 加载组件

再另起一个终端，启动 GUI 执行以下命令：

cd nox/src

./nox-gui.py

NOX 的 GUI 会显示如图 3.16 的信息。

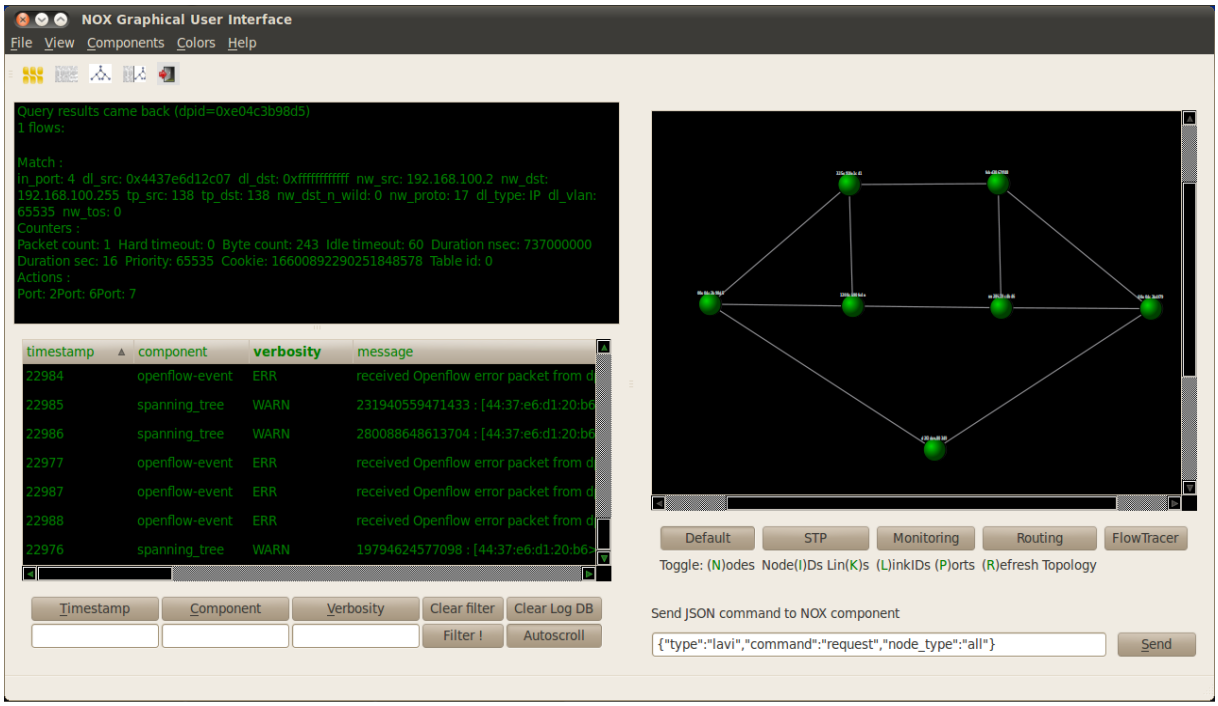

图 3.16 GUI 信息

3.3.4 网络连通性测试

测试网络是否通，用 host1 ping host2 看是否连通。host1 的 IP 为 192.168.100.1，host2 的 IP 为 192.168.100.2，用 host1 ping host2 结果如图 3.17 所示，可以看出是通的。

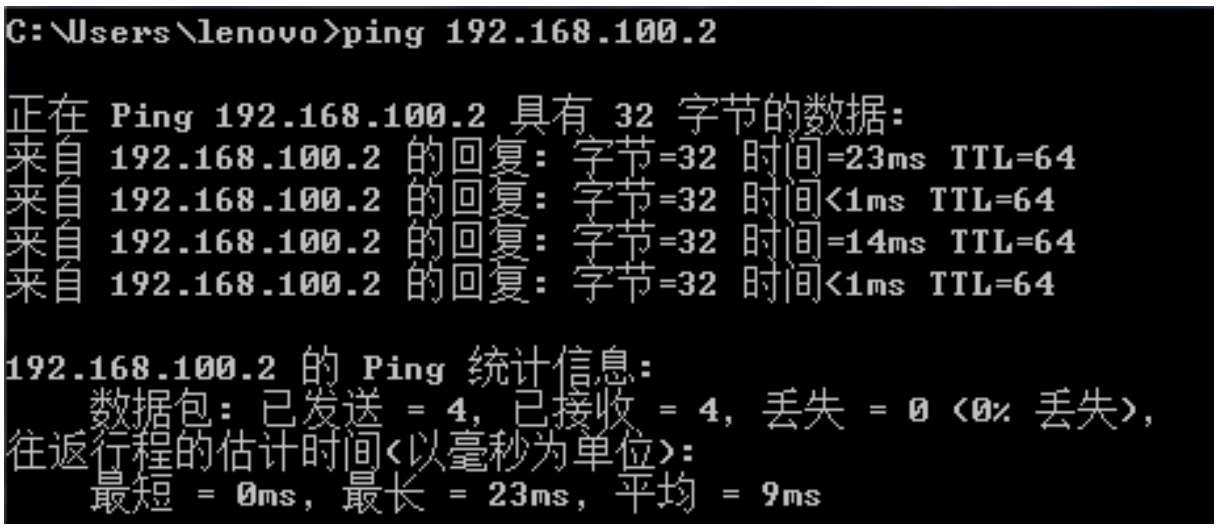

图 3.17 发包成功

第四章 路由功能的实现与仿真分析

4.1 路由模块分析

路由模块是 NOX 控制器中一个非常实用的组件，它可以检索出一个最短路径的路由信息并且在网络中建立流表项，用 OpenFlow 协议中的 actions 来为网络中的流量路由。

路由模块主要是对链路事件（Link_event）进行处理，链路事件的数据结构包括源数据通路（dpsrc）、目的数据通路（dpdst）、源端口（sport）、目的端口（dport）、动作（action）。其中动作包括两种：一个是链路增加（add），另一个是链路删除（remove）。

路由模块中存储路由信息（Route）的数据结构包括：

（1）数据通路的 ID（RouteId）：一个路由的开始数据通路（Start datapath，Sdp）和结束数据通路（End datapath，Edp）；

（2）路径（Path）：一个连接数据通路对的链路列表（list<Link>），其中 Link 结构体包括当前链路的目的数据通路（dpdst）、链路连接的当前数据通路的输出端口（outport）、链路连接的目的数据通路的输入端口（inport）。

路由模块主要维护几个存放路由信息的容器，其中最主要的是存放所有路由信息的容器（local_routes）和存放最短路径的容器（shortest）。local_routes 和 shortest 容器的检索关键字（key 值）都是 RouteId，可以通过查找 RouteId 来找到这对节点对之间的路径信息。local_routes 这个容器中的路由信息对于某个 RouteId 会有不同的 Path，正如从一个源节点到另一个目的节点之间路径有多条一样。而 shortest 这个容器存储的路由信息，对于一个 RouteId 来说，只有一个 Path，就是 local_routes 中不同 Path 中长度最小的那一个，即为最短路径的那一条路由信息。

如图 3.13 中的拓扑所示，以 Br3 到 Br4 为例来说明 local_routes 容器中存放的路由信息，其如表 4.1 所示。

表 4.1 local_routes 路由信息

路由信息（例如:br3→br4）	Sdp	Edp	dpdst	outport	inport
br3→br4	br3	br4	br4	4	4
br3→br5→br4	br3	br4	br5	6	2
			br4	4	6
br3→br1→br2→br4	br3	br4	br1	2	6

表 4.1 local_routes 路由信息（续）

路由信息（例如:br3→br4）	Sdp	Edp	dpdst	outport	inport
			br2	4	4
			br4	6	2
br3→br1→br0→br2→br4	br3	br4	br1	2	6
			br0	2	2
			br2	4	2
			br4	6	2
等等…			等等…		

如表 4.1 所示，从 Br3 到 Br4 的路由信息有多条大小不等的路径即它们之间的节点跳数不同，正如图 3.13 的拓扑所示，从 Br3 到 Br4 有多条路径可走。接下来就是最短路径的问题，如何计算的最短路径，其实从表 4.1 中可以很容易的看出来，只要计算 Path 的长度大小即可，最小的 Path 即为这条路由请求的最短路径。对于从 Br3 到 Br4 这条路由请求而言，第一条路径信息就是最短的路径，接着它会把这条最短路径的信息放到 shortest 这个容器中，这样 shortest 这个容器所维护的就是所有最短路径的路由信息，如表 4.2 所示。

表 4.2 shortest 路由信息

路由信息（例如:br3→br4）	Sdp	Edp	dpdst	outport	inport
br3→br4	br3	br4	br4	4	4

这样如果有路由请求信息，例如 Br3 到 Br4，首先直接查找 shortest 这个容器，如果有相应的路径信息，则直接用 shortest 容器中的路由信息。

在有了请求的路由信息之后，接下来就是给相应的数据通路（Br）下发流表，使用的是 OpenFlow 协议中的 OFPT_FLOW_MOD 消息。主要是通过 OFPT_FLOW_MOD 消息中的 action 来实现下一个端口转发，以及一些匹配域的修改。action 动作有几个针对不同流表项操作的消息，可以实现对流表的修改，这样网络中的流量可以根据下发的或者修改之后的流表项信息进行路由。

通过 OFPT_FLOW_MOD 消息生成下发流表的过程是首先利用 init_openflow 函数初始化 ofp_flow_mod 中的 version、type、xid、wildcards、pad、cookie、command、priority 以及 flags 位。再通过 set_openflow 函数设置 ofp_flow_mod 中的 in_port、dl_src、dl_dst、

dl_vlan、dl_vlan_pcp、dl_type、nw_src、nw_dst、nw_tos、nw_proto、tp_src、tp_dst、buffer_id、idle_timeout、hard_timeout 以及 cookie 位。最后通过 set_openflow_actions 函数设置 ofp_flow_mod 中的 action，其 action 中主要用的消息类型是 OFPAT_OUTPUT，即下一个要转发的输出端口，当然也可以通过 modify_match 函数改变流表项的匹配域中的各个位。

在设置好 ofp_flow_mod 消息的各个域之后，setup_route 函数根据 Route 信息中的输出端口 outport，为相应的数据通路下发相应的 ofp_flow_mod 消息，进而为网络中的流量信息来路由。如果 ofp_flow_mod 消息中的 action 的类型不是 OFPAT_OUTPUT 类型，也可以根据相应的 action 类型，对流进行相应的操作，比如修改流的匹配域的源和目的 MAC 地址。接下来对一条路由信息说明 outport 端口的下发过程，以表 4.3 所示的路由信息为例：

表 4.3 一条路由信息

路由信息（例如:br3→br4）	Sdp	Edp	dpdst	outport	inport
br3→br1→br0→br2→br4	br3	br4	br1	2	6
			br0	2	2
			br2	4	2
			br4	6	2

对表 4.3 中的这条 br3 到 br4 的路由信息，会把第一条 link 信息的 2 号端口通过 ofp_flow_mod 消息下发给数据通路 br3，把第二条 link 信息的 2 号端口通过 ofp_flow_mod 消息下发给数据通路 br1，把第三条 link 信息的 4 号端口通过 ofp_flow_mod 消息下发给数据通路 br0，把第四条 link 信息的 6 号端口通过 ofp_flow_mod 消息下发给数据通路 br2，这样就完成了一条 br3 到 br4 的路由流表下发过程，当网络中的流和这条路由信息相匹配时，会沿着这条路径走。

4.2 仿真结果分析

从前面的路由模块分析中可以看出，NOX 控制器中路由模块这个应用使用的是最短路径的方法，下面通过显示拓扑中各个 Br 的详细信息和其内部的流表来进一步分析基于整个 SDN 环境所实现的路由功能。

显示 br0 的详细信息，执行命令 ovs-ofctl show br0 后如图 4.1 所示。

```
OFPT_FEATURES_REPLY (xid=0x1): dpid:000000e04c3b98d5
n_tables:255, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST EN
QUEUE
 2(patch-0-1): addr:a6:5c:d5:0f:aa:a5
     config:     NO_FLOOD
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 4(patch-0-2): addr:1e:3e:0e:85:3a:0d
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 6(patch-0-6): addr:92:10:f8:f8:2d:94
     config:     NO_FLOOD
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 7(eth1): addr:00:e0:4c:3b:98:d5
     config:     0
     state:      0
     current:    100MB-FD AUTO_NEG
     advertised: 10MB-HD 10MB-FD 100MB-HD 100MB-FD COPPER AUTO_NEG
     supported:  10MB-HD 10MB-FD 100MB-HD 100MB-FD COPPER AUTO_NEG
     speed: 100 Mbps now, 100 Mbps max
 LOCAL(br0): addr:00:e0:4c:3b:98:d5
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x3): frags=normal miss_send_len=0
```

图 4.1 br0 的详细信息

从图 4.1 中可以看出，br0 的数据通路 ID（dpid）是 000000e04c3b98d5，流表数为 255，缓存的最大包数目为 256，br0 支持的功能为流统计、流表统计、端口统计、队列统计、在 ARP 包中匹配 IP 地址，并支持所有的 actions 类型。和 br1 相连的端口号是 2，和 br2 相连的端口号是 4，和 br6 相连的端口号是 6，和 eth1 相连的端口号是 7，正如图 3.13 中的拓扑所示。

显示 br0 的流表，执行命令 ovs-ofctl dump-flows br0 后如图 4.2 所示。

```
NXST_FLOW reply (xid=0x4):
 cookie=0x6d4098c946f4a409, duration=1.438s, table=0, n_packets=2, n_bytes=148, idle_timeout=5, idle_age=0, priority=65535,icmp,in_p
ort=7,vlan_tci=0x0000,dl_src=44:37:e6:d1:20:b6,dl_dst=44:37:e6:d1:2c:07,nw_src=192.168.100.1,nw_dst=192.168.100.2,nw_tos=0,icmp_type
=8,icmp_code=0 actions=output:6
 cookie=0x9d4cbbe74577e5c9, duration=1.428s, table=0, n_packets=2, n_bytes=148, idle_timeout=5, idle_age=0, priority=65535,icmp,in_p
ort=6,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=44:37:e6:d1:20:b6,nw_src=192.168.100.2,nw_dst=192.168.100.1,nw_tos=0,icmp_type
=0,icmp_code=0 actions=output:7
 cookie=0x2819f67581ec44fa, duration=53.623s, table=0, n_packets=2, n_bytes=164, idle_timeout=60, idle_age=13, priority=65535,udp,in
_port=4,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.2,nw_dst=192.168.100.255,nw_tos=0,tp_sr
c=55558,tp_dst=1947 actions=output:2,output:6,output:7
 cookie=0xdf13b6b6c04ae052, duration=0.91s, table=0, n_packets=2, n_bytes=184, idle_timeout=60, idle_age=0, priority=65535,udp,in_po
rt=7,vlan_tci=0x0000,dl_src=44:37:e6:d1:20:b6,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.1,nw_dst=192.168.100.255,nw_tos=0,tp_src=1
37,tp_dst=137 actions=output:2,output:6,output:4
 cookie=0x99f9d212b38bd123, duration=57.624s, table=0, n_packets=2, n_bytes=164, idle_timeout=60, idle_age=17, priority=65535,udp,in
_port=4,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.2,nw_dst=255.255.255.255,nw_tos=0,tp_sr
c=55558,tp_dst=1947 actions=output:2,output:6,output:7
```

图 4.2 br0 的流表

从图 4.2 中可以看出，br0 的流表信息可总结为表 4.4。

表 4.4 br0 流表

Br	进入端口	源 IP 地址	目的 IP 地址	出去端口
br0	7	192.168.100.1	192.168.100.2	6
	6	192.168.100.2	192.168.100.1	7
	7	192.168.100.1	192.168.100.255	2、4、6

表 4.4 br0 流表（续）

Br	进入端口	源 IP 地址	目的 IP 地址	出去端口
br0	4	192.168.100.2	192.168.100.255	2、6、7
	4	192.168.100.2	255.255.255.255	2、6、7

显示 br1 的详细信息，执行命令 ovs-ofctl show br1 后如图 4.3 所示。

```
OFPT_FEATURES_REPLY (xid=0x1): dpid:0000325e91fa1c41
n_tables:255, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST EN
QUEUE
 2(patch-1-0): addr:52:3c:be:d9:42:cd
     config:     NO_FLOOD
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 4(patch-1-2): addr:e6:2a:48:bc:2e:fb
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 6(patch-1-3): addr:c2:19:d6:79:bf:74
     config:     NO_FLOOD
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 LOCAL(br1): addr:32:5e:91:fa:1c:41
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x3): frags=normal miss_send_len=0
```

图 4.3 br1 的详细信息

从图 4.3 中可以看出，br1 的数据通路 ID（dpid）是 0000325e91fa1c41，和 br0 相连的端口号是 2，和 br2 相连的端口号是 4，和 br3 相连的端口号是 6，正如图 3.13 中的拓扑所示。其它信息和 br0 相同。

显示 br1 的流表，执行命令 ovs-ofctl dump-flows br1 后如图 4.4 所示。

```
NXST_FLOW reply (xid=0x4):
 cookie=0xedce0fc7e4e39671, duration=1.653s, table=0, n_packets=2, n_bytes=184, idle_timeout=60, idle_age=0, priority=65535,udp,in_p
ort=4,vlan_tci=0x0000,dl_src=44:37:e6:d1:20:b6,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.1,nw_dst=192.168.100.255,nw_tos=0,tp_src=
137,tp_dst=137 actions=output:2,output:6
 cookie=0x2819f67581ec44fa, duration=13.646s, table=0, n_packets=1, n_bytes=82, idle_timeout=60, idle_age=13, priority=65535,udp,in_
port=4,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.2,nw_dst=192.168.100.255,nw_tos=0,tp_src
=55558,tp_dst=1947 actions=output:2,output:6
```

图 4.4 br1 的流表

从图 4.4 中可以看出，br1 的流表信息可总结为表 4.5。

表 4.5 br1 流表

Br	进入端口	源 IP 地址	目的 IP 地址	出去端口
br1	4	192.168.100.1	192.168.100.255	2、6
	4	192.168.100.2	192.168.100.255	2、6

显示 br2 的详细信息，执行命令 ovs-ofctl show br2 后如图 4.5 所示。

```
OFPT_FEATURES_REPLY (xid=0x1): dpid:00001200cb90fe4a
n_tables:255, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST EN
QUEUE
 2(patch-2-0): addr:fe:8a:b8:bf:4e:38
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 4(patch-2-1): addr:0e:f6:ba:75:0e:9c
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 6(patch-2-4): addr:be:63:41:6b:5d:91
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 LOCAL(br2): addr:12:00:cb:90:fe:4a
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x3): frags=normal miss_send_len=0
```

图 4.5 br2 的详细信息

从图 4.5 中可以看出，br2 的数据通路 ID（dpid）是 00001200cb90fe4a，和 br0 相连的端口号是 2，和 br1 相连的端口号是 4，和 br4 相连的端口号是 6，正如图 3.13 中的拓扑所示。其它信息和 br0 相同。

显示 br2 的流表，执行命令 ovs-ofctl dump-flows br2 后如图 4.6 所示。

```
NXST_FLOW reply (xid=0x4):
 cookie=0xe8824c90c390b3fc, duration=53.635s, table=0, n_packets=2, n_bytes=164, idle_timeout=60, idle_age=13, priority=65535,udp,in
_port=6,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.2,nw_dst=192.168.100.255,nw_tos=0,tp_sr
c=55558,tp_dst=1947 actions=output:2,output:4
 cookie=0x531515db210cd0b4, duration=1.678s, table=0, n_packets=3, n_bytes=276, idle_timeout=60, idle_age=0, priority=65535,udp,in_p
ort=2,vlan_tci=0x0000,dl_src=44:37:e6:d1:20:b6,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.1,nw_dst=192.168.100.255,nw_tos=0,tp_src=
137,tp_dst=137 actions=output:6,output:4
 cookie=0x10fa5fc40b519545, duration=57.637s, table=0, n_packets=2, n_bytes=164, idle_timeout=60, idle_age=17, priority=65535,udp,in
_port=6,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.2,nw_dst=255.255.255.255,nw_tos=0,tp_sr
c=55558,tp_dst=1947 actions=output:2,output:4
```

图 4.6 br2 的流表

从图 4.6 中可以看出，br2 的流表信息可总结为表 4.6。

表 4.6 br2 流表

Br	进入端口	源 IP 地址	目的 IP 地址	出去端口
br2	2	192.168.100.1	192.168.100.255	4、6
	6	192.168.100.2	192.168.100.255	2、4
	6	192.168.100.2	255.255.255.255	2、4

显示 br3 的详细信息，执行命令 ovs-ofctl show br3 后如图 4.7 所示。

```
OFPT_FEATURES_REPLY (xid=0x1): dpid:0000febd3867ff48
n_tables:255, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST EN
QUEUE
 2(patch-3-1): addr:0a:53:73:12:88:e5
     config:     NO_FLOOD
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 4(patch-3-4): addr:06:3b:86:9e:db:c1
     config:     NO_FLOOD
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 6(patch-3-5): addr:5e:07:b5:68:1a:7a
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 LOCAL(br3): addr:fe:bd:38:67:ff:48
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x3): frags=normal miss_send_len=0
```

图 4.7 br3 的详细信息

从图 4.7 中可以看出，br3 的数据通路 ID（dpid）是 0000febd3867ff48，和 br1 相连的端口号是 2，和 br4 相连的端口号是 4，和 br5 相连的端口号是 6，正如图 3.13 中的拓扑所示。其它信息和 br0 相同。

显示 br3 的流表，执行命令 ovs-ofctl dump-flows br3 后如图 4.8 所示。

```
NXST_FLOW reply (xid=0x4):
 cookie=0xe8824c90c390b3fc, duration=53.659s, table=0, n_packets=2, n_bytes=164, idle_timeout=60, idle_age=13, priority=65535,udp,in
_port=6,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.2,nw_dst=192.168.100.255,nw_tos=0,tp_sr
c=55558,tp_dst=1947 actions=output:2,output:4
 cookie=0xa5cbb76656a18c6f, duration=1.656s, table=0, n_packets=2, n_bytes=184, idle_timeout=60, idle_age=0, priority=65535,udp,in_p
ort=6,vlan_tci=0x0000,dl_src=44:37:e6:d1:20:b6,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.1,nw_dst=192.168.100.255,nw_tos=0,tp_src=
137,tp_dst=137 actions=output:2,output:4
 cookie=0x10fa5fc40b519545, duration=57.664s, table=0, n_packets=2, n_bytes=164, idle_timeout=60, idle_age=17, priority=65535,udp,in
_port=6,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.2,nw_dst=255.255.255.255,nw_tos=0,tp_sr
c=55558,tp_dst=1947 actions=output:2,output:4
```

图 4.8 br3 的流表

从图 4.8 中可以看出，br3 的流表信息可总结为表 4.7。

表 4.7 br3 流表

Br	进入端口	源 IP 地址	目的 IP 地址	出去端口
br3	6	192.168.100.1	192.168.100.255	2、4
	6	192.168.100.2	192.168.100.255	2、4
	6	192.168.100.2	255.255.255.255	2、4

显示 br4 的详细信息，执行命令 ovs-ofctl show br4 后如图 4.9 所示。

```
OFPT_FEATURES_REPLY (xid=0x1): dpid:0000aa28619c4b46
n_tables:255, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST EN
QUEUE
 2(patch-4-2): addr:1a:5c:3b:b6:3d:f1
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 4(patch-4-3): addr:c2:72:6f:7f:2b:38
     config:     NO_FLOOD
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 6(patch-4-5): addr:46:53:99:7e:ba:5b
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 LOCAL(br4): addr:aa:28:61:9c:4b:46
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x3): frags=normal miss_send_len=0
```

图 4.9 br4 的详细信息

从图 4.9 中可以看出，br4 的数据通路 ID（dpid）是 0000aa28619c4b46，和 br2 相连的端口号是 2，和 br3 相连的端口号是 4，和 br5 相连的端口号是 6，正如图 3.13 中的拓扑所示。其它信息和 br0 相同。

显示 br4 的流表，执行命令 ovs-ofctl dump-flows br4 后如图 4.10 所示。

```
NXST_FLOW reply (xid=0x4):
 cookie=0x531515db210cd0b4, duration=61.681s, table=0, n_packets=4, n_bytes=368, idle_timeout=60, idle_age=0, priority=65535,udp,in_
port=2,vlan_tci=0x0000,dl_src=44:37:e6:d1:20:b6,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.1,nw_dst=192.168.100.255,nw_tos=0,tp_src
=137,tp_dst=137 actions=output:6,output:4
 cookie=0xe8824c90c390b3fc, duration=53.664s, table=0, n_packets=2, n_bytes=164, idle_timeout=60, idle_age=13, priority=65535,udp,in
_port=6,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.2,nw_dst=192.168.100.255,nw_tos=0,tp_sr
c=55558,tp_dst=1947 actions=output:2,output:4
 cookie=0x10fa5fc40b519545, duration=57.67s, table=0, n_packets=2, n_bytes=164, idle_timeout=60, idle_age=17, priority=65535,udp,in_
port=6,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.2,nw_dst=255.255.255.255,nw_tos=0,tp_src
=55558,tp_dst=1947 actions=output:2,output:4
```

图 4.10 br4 的流表

从图 4.10 中可以看出，br4 的流表信息可总结为表 4.8。

表 4.8 br4 流表

Br	进入端口	源 IP 地址	目的 IP 地址	出去端口
br4	2	192.168.100.1	192.168.100.255	4、6
	6	192.168.100.2	192.168.100.255	2、4
	6	192.168.100.2	255.255.255.255	2、4

显示 br5 的详细信息，执行命令 ovs-ofctl show br5 后如图 4.11 所示。

```
OFPT_FEATURES_REPLY (xid=0x1): dpid:000000e04c3b4f79
n_tables:255, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST EN
QUEUE
 2(patch-5-3): addr:ea:5f:46:29:0a:41
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 4(patch-5-4): addr:8a:7b:2f:6b:8d:b9
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 6(patch-5-6): addr:06:87:cb:fd:65:09
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 7(eth2): addr:00:e0:4c:3b:4f:79
     config:     0
     state:      0
     current:    100MB-FD AUTO_NEG
     advertised: 10MB-HD 10MB-FD 100MB-HD 100MB-FD COPPER AUTO_NEG
     supported:  10MB-HD 10MB-FD 100MB-HD 100MB-FD COPPER AUTO_NEG
     speed: 100 Mbps now, 100 Mbps max
 LOCAL(br5): addr:00:e0:4c:3b:4f:79
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x3): frags=normal miss_send_len=0
```

图 4.11 br5 的详细信息

从图 4.11 中可以看出，br5 的数据通路 ID（dpid）是 000000e04c3b4f79，和 br3 相连的端口号是 2，和 br4 相连的端口号是 4，和 br6 相连的端口号是 6，和 eth2 相连的端口号是 7，正如图 3.13 中的拓扑所示。其它信息和 br0 相同。

显示 br5 的流表，执行命令 ovs-ofctl dump-flows br5 后如图 4.12 所示。

```
NXST_FLOW reply (xid=0x4):
 cookie=0xedce0fc7e4e39671, duration=1.67s, table=0, n_packets=2, n_bytes=184, idle_timeout=60, idle_age=0, priority=65535,udp,in_po
rt=4,vlan_tci=0x0000,dl_src=44:37:e6:d1:20:b6,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.1,nw_dst=192.168.100.255,nw_tos=0,tp_src=1
37,tp_dst=137 actions=output:2,output:6,output:7
 cookie=0x6d4098c946f4a409, duration=1.459s, table=0, n_packets=2, n_bytes=148, idle_timeout=5, idle_age=0, priority=65535,icmp,in_p
ort=6,vlan_tci=0x0000,dl_src=44:37:e6:d1:20:b6,dl_dst=44:37:e6:d1:2c:07,nw_src=192.168.100.1,nw_dst=192.168.100.2,nw_tos=0,icmp_type
=8,icmp_code=0 actions=output:7
 cookie=0x9d4cbbe74577e5c9, duration=1.451s, table=0, n_packets=2, n_bytes=148, idle_timeout=5, idle_age=0, priority=65535,icmp,in_p
ort=7,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=44:37:e6:d1:20:b6,nw_src=192.168.100.2,nw_dst=192.168.100.1,nw_tos=0,icmp_type
=0,icmp_code=0 actions=output:6
```

图 4.12 br5 的流表

从图 4.12 中可以看出，br5 的流表信息可总结为表 4.9。

表 4.9 br5 流表

Br	进入端口	源 IP 地址	目的 IP 地址	出去端口
br5	6	192.168.100.1	192.168.100.2	7
	7	192.168.100.2	192.168.100.1	6
	4	192.168.100.1	192.168.100.255	2、6、7

显示 br6 的详细信息，执行命令 ovs-ofctl show br6 后如图 4.13 所示。

```
OFPT_FEATURES_REPLY (xid=0x1): dpid:0000d2f2dea80349
n_tables:255, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST EN
QUEUE
 2(patch-6-0): addr:b2:85:eb:bb:eb:53
     config:     NO_FLOOD
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 4(patch-6-5): addr:4e:3c:84:3e:ac:02
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
 LOCAL(br6): addr:d2:f2:de:a8:03:49
     config:     0
     state:      0
     speed: 100 Mbps now, 100 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x3): frags=normal miss_send_len=0
```

图 4.13 br6 的详细信息

从图 4.13 中可以看出，br6 的数据通路 ID（dpid）是 0000d2f2dea80349，和 br0 相连的端口号是 2，和 br5 相连的端口号是 4，正如图 3.13 中的拓扑所示。其它信息和 br0 相同。

显示 br6 的流表，执行命令 ovs-ofctl dump-flows br6 后如图 4.14 所示。

```
NXST_FLOW reply (xid=0x4):
 cookie=0xedce0fc7e4e39671, duration=1.668s, table=0, n_packets=2, n_bytes=184, idle_timeout=60, idle_age=0, priority=65535,udp,in_p
ort=4,vlan_tci=0x0000,dl_src=44:37:e6:d1:20:b6,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.1,nw_dst=192.168.100.255,nw_tos=0,tp_src=
137,tp_dst=137 actions=output:2
 cookie=0x6d4098c946f4a409, duration=1.463s, table=0, n_packets=2, n_bytes=148, idle_timeout=5, idle_age=0, priority=65535,icmp,in_p
ort=2,vlan_tci=0x0000,dl_src=44:37:e6:d1:20:b6,dl_dst=44:37:e6:d1:2c:07,nw_src=192.168.100.1,nw_dst=192.168.100.2,nw_tos=0,icmp_type
=8,icmp_code=0 actions=output:4
 cookie=0x2819f67581ec44fa, duration=53.671s, table=0, n_packets=2, n_bytes=164, idle_timeout=60, idle_age=13, priority=65535,udp,in
_port=4,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.2,nw_dst=192.168.100.255,nw_tos=0,tp_sr
c=55558,tp_dst=1947 actions=output:2
 cookie=0x9d4cbbe74577e5c9, duration=1.454s, table=0, n_packets=2, n_bytes=148, idle_timeout=5, idle_age=0, priority=65535,icmp,in_p
ort=4,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=44:37:e6:d1:20:b6,nw_src=192.168.100.2,nw_dst=192.168.100.1,nw_tos=0,icmp_type
=0,icmp_code=0 actions=output:2
 cookie=0x99f9d212b38bd123, duration=57.677s, table=0, n_packets=2, n_bytes=164, idle_timeout=60, idle_age=17, priority=65535,udp,in
_port=4,vlan_tci=0x0000,dl_src=44:37:e6:d1:2c:07,dl_dst=ff:ff:ff:ff:ff:ff,nw_src=192.168.100.2,nw_dst=255.255.255.255,nw_tos=0,tp_sr
c=55558,tp_dst=1947 actions=output:2
```

图 4.14 br6 的流表

从图 4.14 中可以看出，br6 的流表信息可总结为表 4.10。

表 4.10 br6 流表

Br	进入端口	源 IP 地址	目的 IP 地址	出去端口
br6	2	192.168.100.1	192.168.100.2	4
	4	192.168.100.2	192.168.100.1	2
	4	192.168.100.1	192.168.100.255	2
	4	192.168.100.2	192.168.100.255	2
	4	192.168.100.2	255.255.255.255	2

最后对 br0，br1，br2，br3，br4，br5，br6 的各个流表进行汇总，如表 4.11 所示。

表 4.11 流表汇总

Br	进入端口	源 IP 地址	目的 IP 地址	出去端口
br0	**7**	**192.168.100.1**	**192.168.100.2**	**6**
	6	192.168.100.2	192.168.100.1	7
	7	192.168.100.1	192.168.100.255	2、4、6
	4	192.168.100.2	192.168.100.255	2、6、7
	4	192.168.100.2	255.255.255.255	2、6、7
br1	4	192.168.100.1	192.168.100.255	2、6
	4	192.168.100.2	192.168.100.255	2、6
br2	2	192.168.100.1	192.168.100.255	4、6
	6	192.168.100.2	192.168.100.255	2、4
	6	192.168.100.2	255.255.255.255	2、4
br3	6	192.168.100.1	192.168.100.255	2、4
	6	192.168.100.2	192.168.100.255	2、4
	6	192.168.100.2	255.255.255.255	2、4
br4	2	192.168.100.1	192.168.100.255	4、6
	6	192.168.100.2	192.168.100.255	2、4
	6	192.168.100.2	255.255.255.255	2、4
br5	**6**	**192.168.100.1**	**192.168.100.2**	**7**
	7	192.168.100.2	192.168.100.1	6
	4	192.168.100.1	192.168.100.255	2、6、7
br6	**2**	**192.168.100.1**	**192.168.100.2**	**4**
	4	192.168.100.2	192.168.100.1	2
	4	192.168.100.1	192.168.100.255	2
	4	192.168.100.2	192.168.100.255	2
	4	192.168.100.2	255.255.255.255	2

从表 4.11 中可以清晰的看到拓扑中所有交换机（br）的流表。主机 1（Host1）的 IP 地址为 192.168.100.1，主机 2（Host2）的 IP 地址为 192.168.100.2，当 Host1 向 Host2 发包时，br0 刚开始是没有任何流表信息的。当这个包到达时，br0 会发送

OFPT_PACKET_IN 消息给 NOX 控制器，NOX 控制器收到这个消息后，会根据整个网络拓扑的状态，为这个路由请求建立相应的流表，并相应的下发到各个 br 中，下发之后的各个 br 的流表信息就如表 4.11 所示。接下来这个包会沿着表 4.11 中加粗的路径信息来走，首先 br0 从 7 号端口接收到这个包，接着 br0 把它从 6 号端口转发出去到 br6，br6 从 2 号端口接到这个数据包，再把它从 4 号端口转发出去到 br5，br5 从 6 号端口接收到这个包，再把它从 7 号端口转发出去到 eth2 即到了 Host2。从流表的路径可以看出，Host1 到 Host2 走的路径是 Host1→br0→br6→br5→Host2，是最短路径，如图 4.15 拓扑上显示的路径。

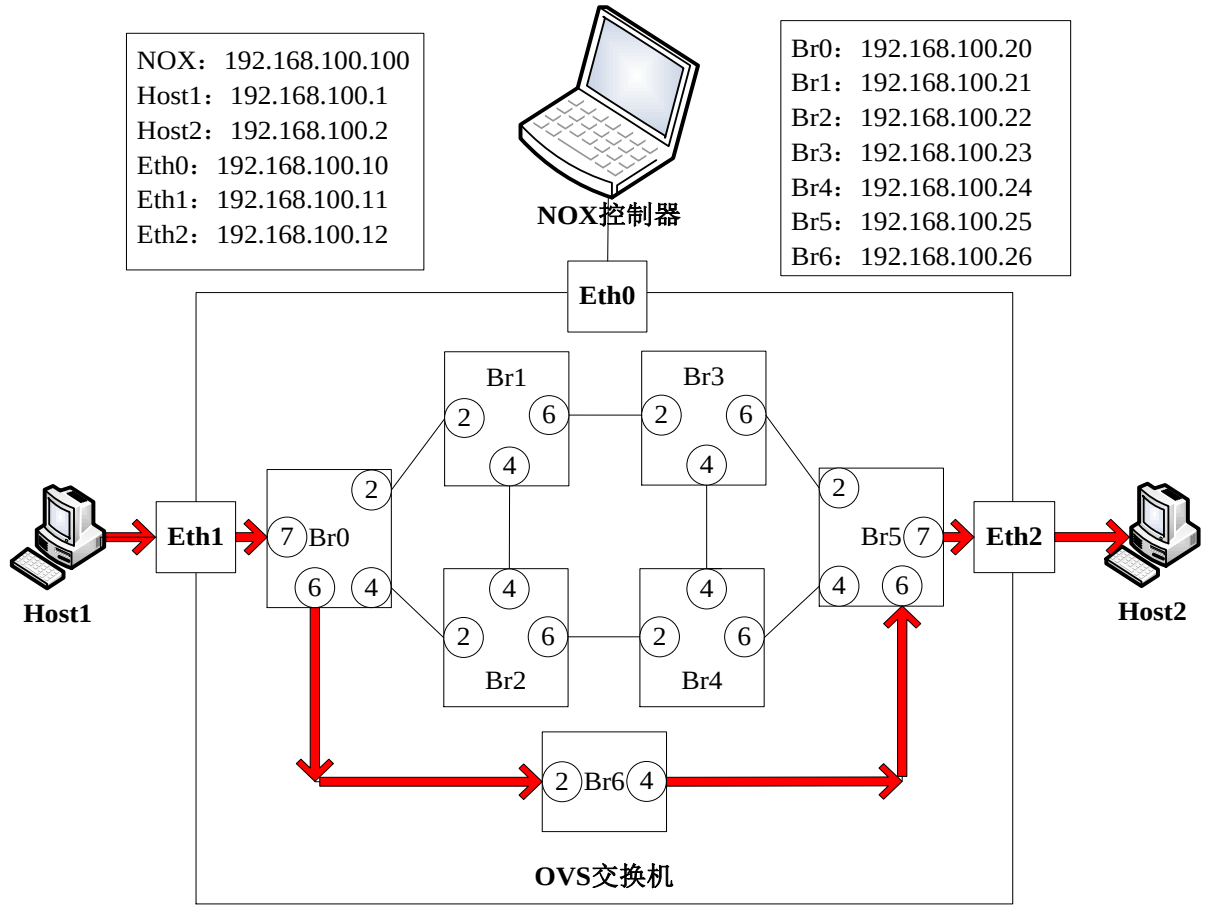

图 4.15 最短路径示意图