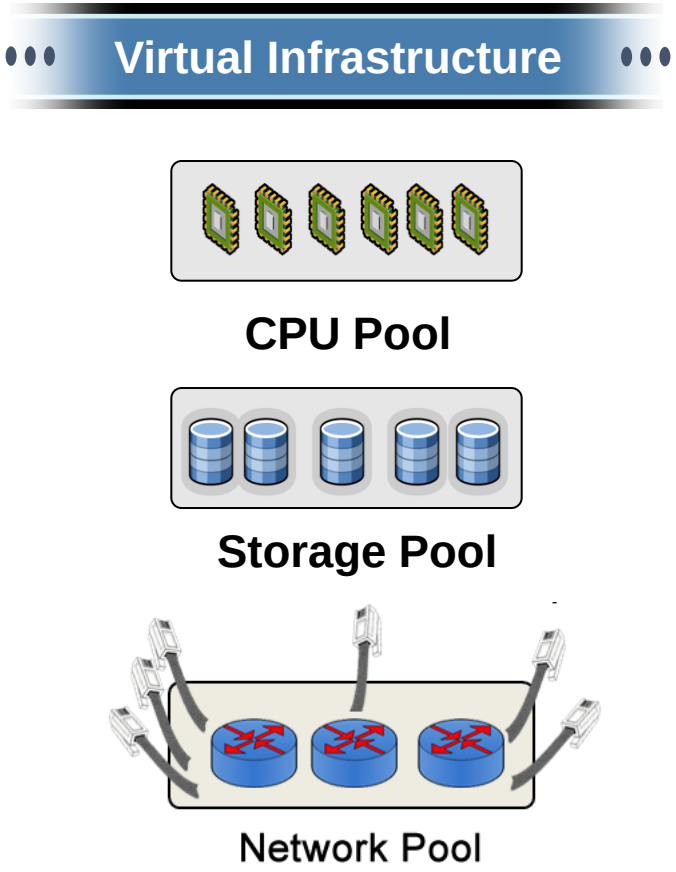
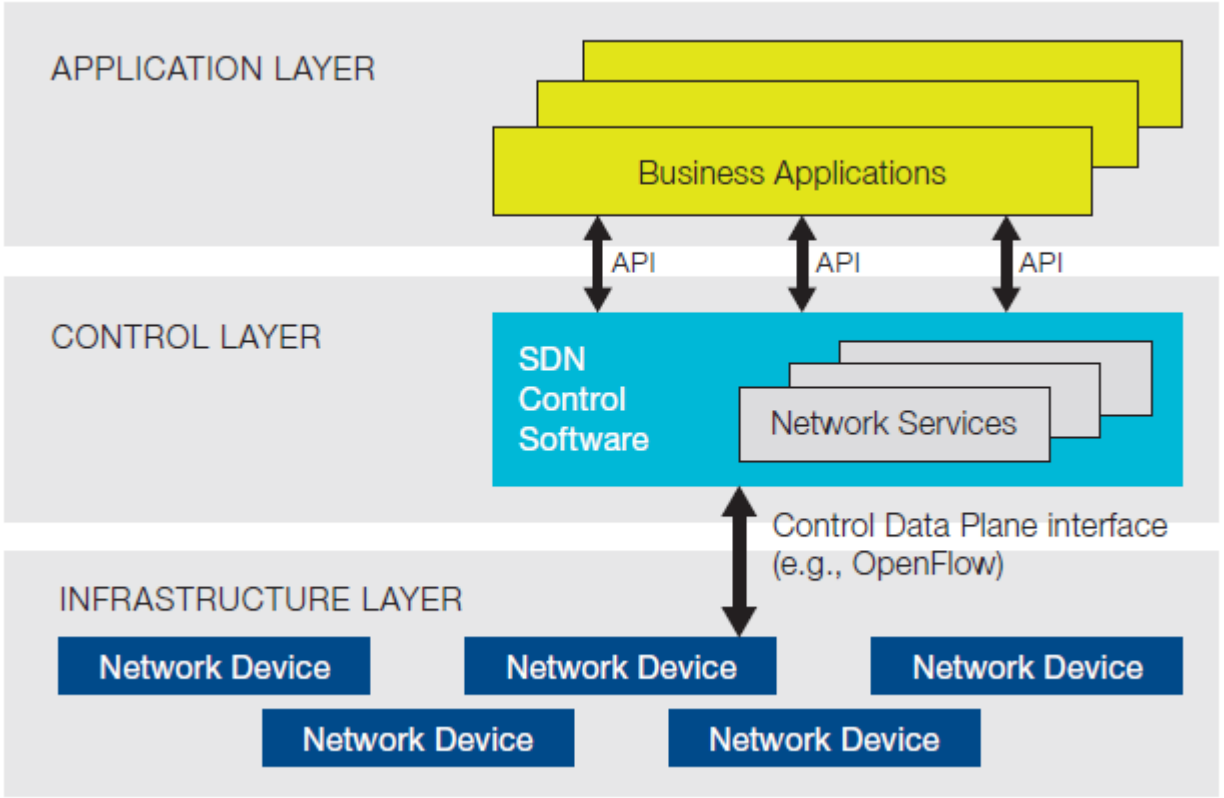
Openvswitch

Popsuper

Software Defined Network





OpenFlow Switch Components

- OpenFlow Channel负责同Controller的交互
- Flow Table包含许多entry，每个entry是对packet进行处理的规则

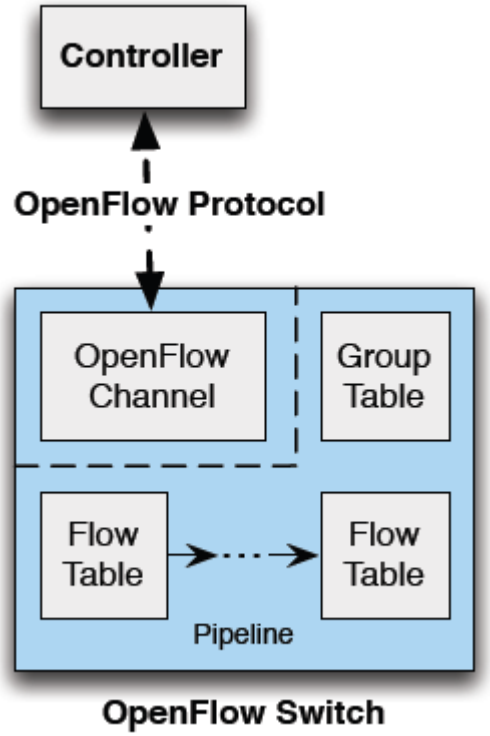
Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Table 1: Main components of a flow entry in a flow table.

Match packets:
- ingress port
- Headers
- metadata

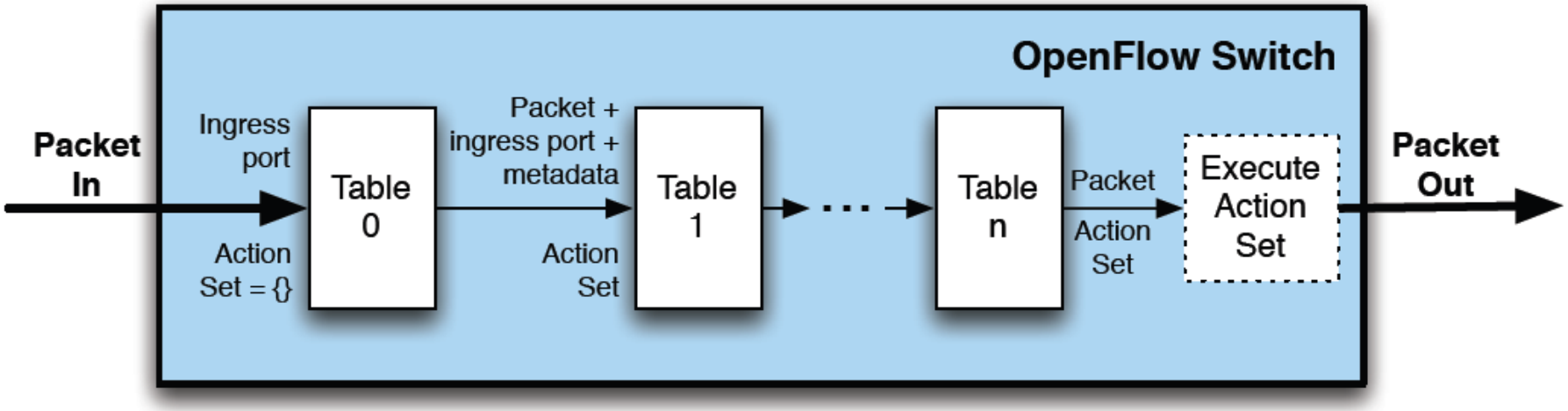
Flow entries match packets in priority order

对packet处理:
- 转发
- 修改
- 交给Group Table
- 交给下个Table

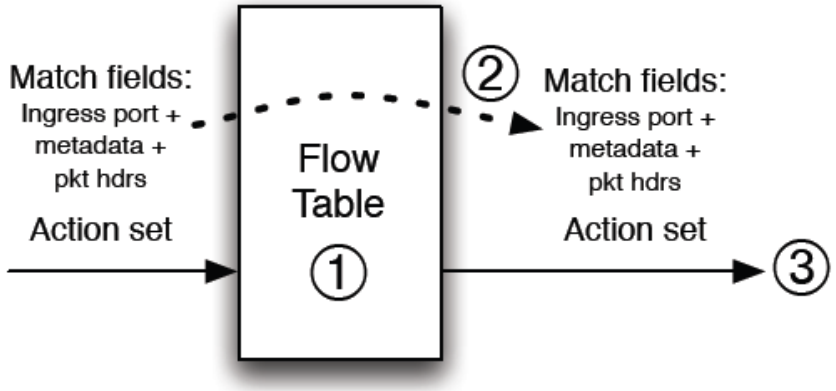


- Group Table：处理更复杂的转发规则
  - 包含一系列Group Entry
  - 每个Entry包含一系列操作集合(action buckets)
  - 每个操作集合包含一系列action，以及参数

OpenFlow Packet Processing



(a) Packets are matched against multiple tables in the pipeline



① Find highest-priority matching flow entry

② Apply instructions:
- i. Modify packet & update match fields (apply actions instruction)
- ii. Update action set (clear actions and/or write actions instructions)
- iii. Update metadata

③ Send match data and action set to next table

(b) Per-table packet processing

OpenFlow Packet Processing

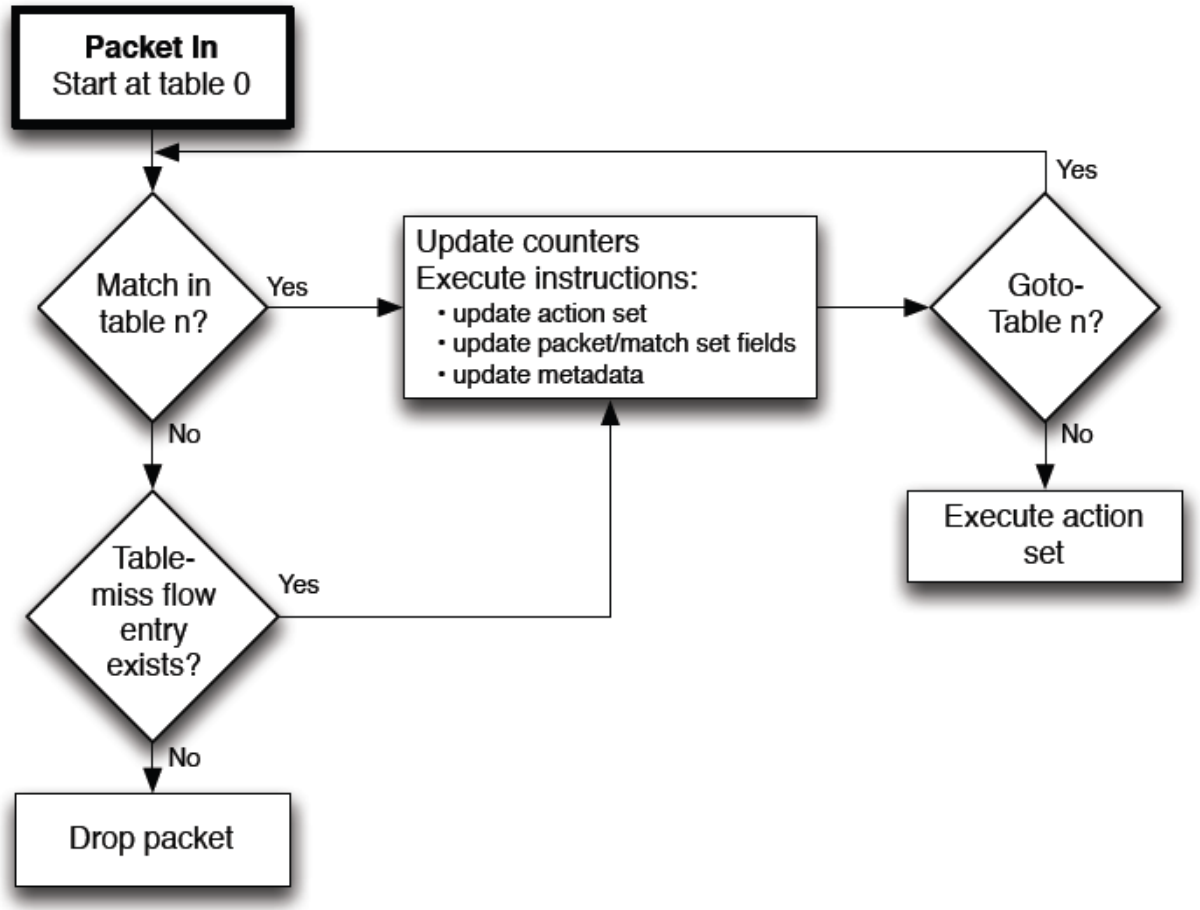


Figure 3: Flowchart detailing packet flow through an OpenFlow switch.

Actions:

- Output: 转发
- Set-Queue: QoS
- Drop
- Group
- Push/Pop tags

OpenFlow Packet Processing

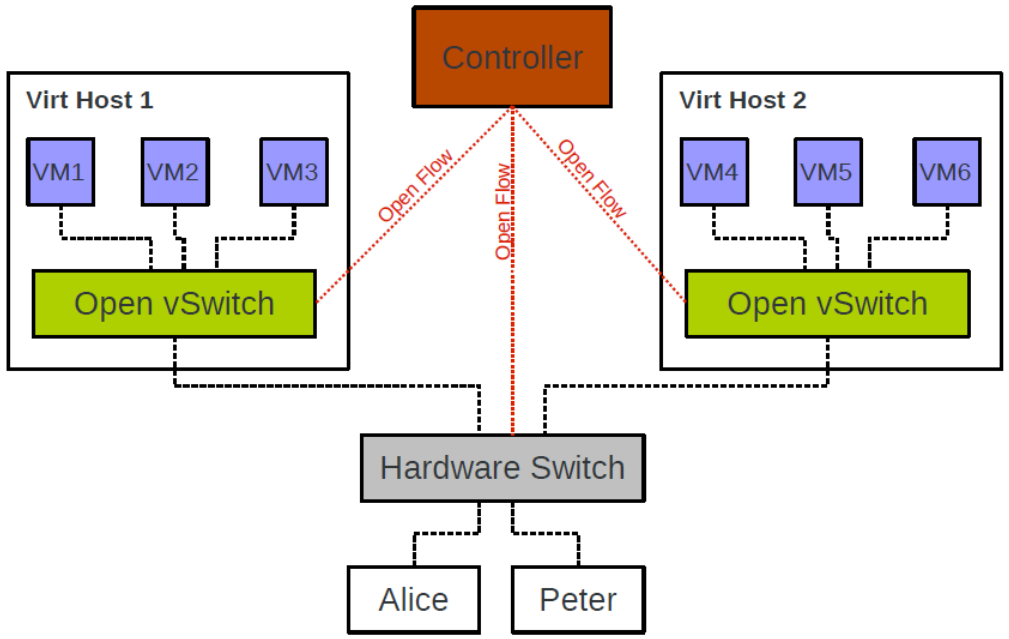
- Actions:
  - Output: 转发
  - Set-Queue: QoS
  - Drop
  - Group
  - Push/Pop tags

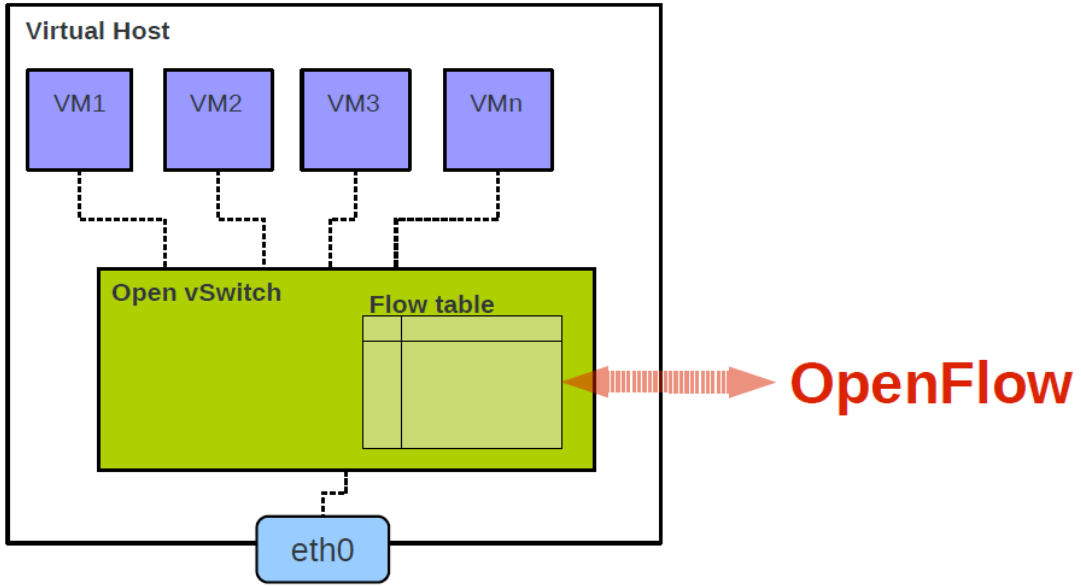
Action	Associated Data	Description
Push VLAN header	Ethertype	Push a new VLAN header onto the packet. The Ethertype is used as the Ethertype for the tag. Only Ethertype 0x8100 and 0x88a8 should be used.
Pop VLAN header	-	Pop the outer-most VLAN header from the packet.
Push MPLS header	Ethertype	Push a new MPLS shim header onto the packet. The Ethertype is used as the Ethertype for the tag. Only Ethertype 0x8847 and 0x8848 should be used.
Pop MPLS header	Ethertype	Pop the outer-most MPLS tag or shim header from the packet. The Ethertype is used as the Ethertype for the resulting packet (Ethertype for the MPLS payload).
Push PBB header	Ethertype	Push a new PBB service instance header (I-TAG TCI) onto the packet (see 7.2.4). The Ethertype is used as the Ethertype for the tag. Only Ethertype 0x88E7 should be used.
Pop PBB header	-	Pop the outer-most PBB service instance header (I-TAG TCI) from the packet (see 7.2.4).

Table 6: Push/pop tag actions.

Openvswitch简介

- Openvswitch是一个virutal swtich， 支持Open Flow协议，当然也有一些硬件Switch也支持Open Flow协议，他们都可以被统一的Controller管理，从而实现物理机和虚拟机的网络联通。





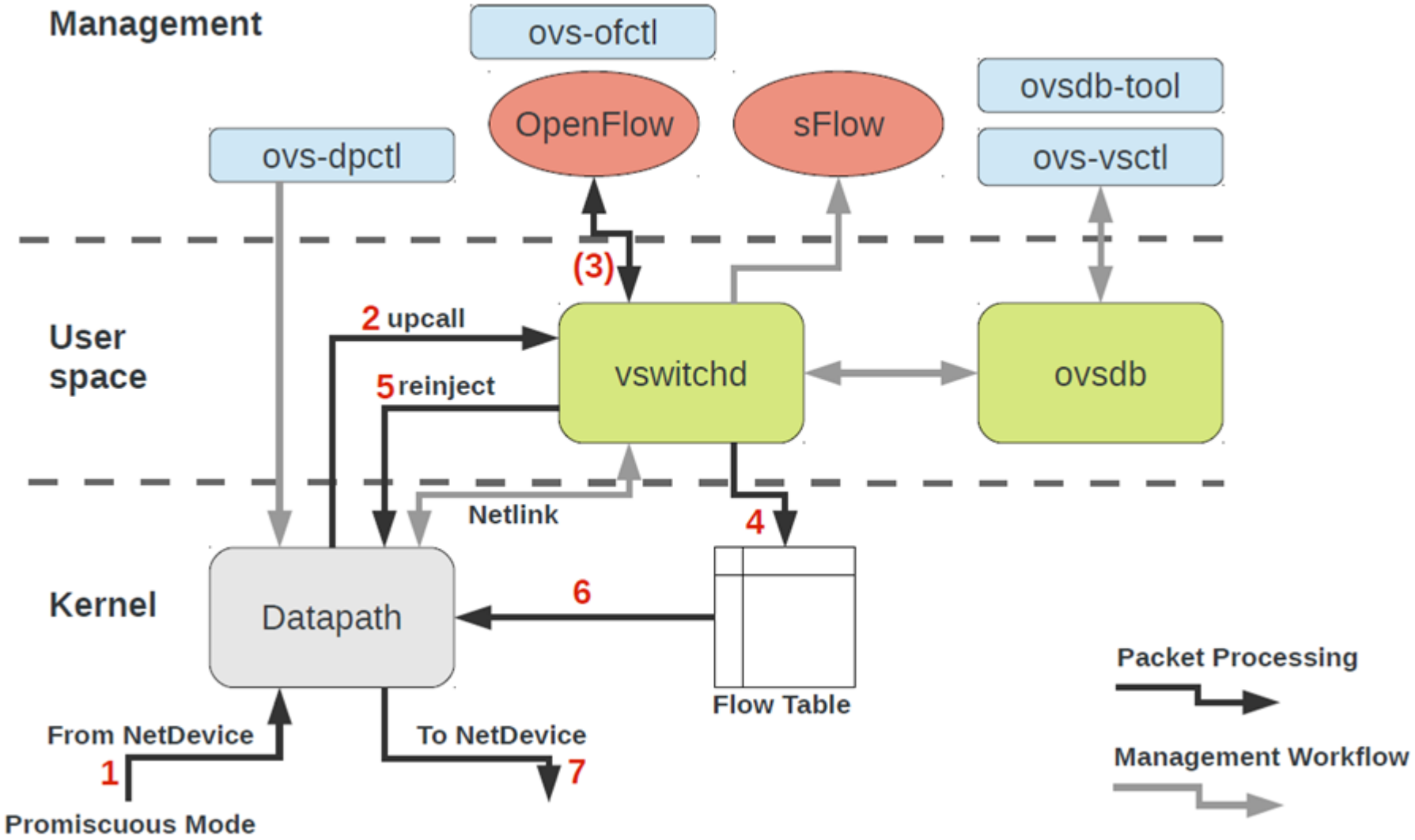
Openvswitch简介

- Match Field涵盖TCP/IP协议各层：
  - Layer 1 – Tunnel ID, In Port, QoS priority, skb mark
  - Layer 2 – MAC address, VLAN ID, Ethernet type
  - Layer 3 – IPv4/IPv6 fields, ARP
  - Layer 4 – TCP/UDP, ICMP, ND
- Action也主要包含下面的操作：
  - Output to port (port range, flood, mirror)
  - Discard, Resubmit to table x
  - Packet Mangling (Push/Pop VLAN header, TOS, ...)
  - Send to controller, Learn

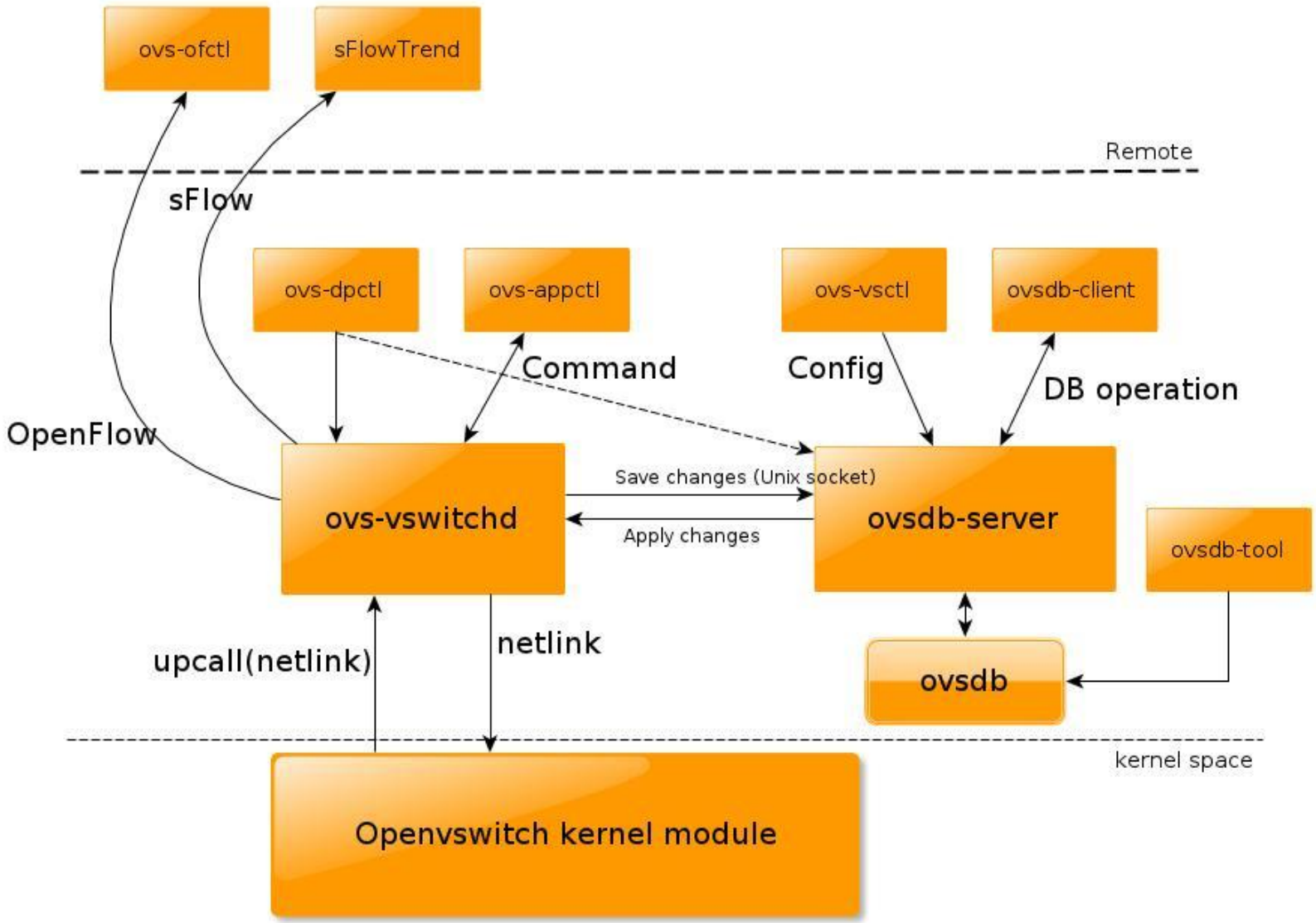
Openvswitch简介

- 可以设置Tunnel
- 可以支持下列的框架来监控流量。
  - sFlow
  - NetFlow
  - Port Mirroring
    - SPAN
    - RSPAN
    - ERSPAN
- 支持QoS
  - Uses existing Traffic Control Layer
    - Policer (Ingress rate limiter)
    - HTB, HFSC (Egress traffic classes)
  - Controller (Open Flow) can select Traffic Class

Openvswitch架构



Openvswitch架构



实验一：查看Openvswitch的架构

```
root@popsuper1982:~# ps aux | grep openvswitch
root      985  0.0  0.0  21172  2120 ?        S<   Aug06   1:20 ovsdb-server /etc/openvswitch/conf.db -vconsole:emer -
vsyslog:err -vfile:info --remote=punix:/var/run/openvswitch/db.sock --private-key=db:Open_vSwitch,SSL,private_key --
certificate=db:Open_vSwitch,SSL,certificate --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert --no-chdir --log-
file=/var/log/openvswitch/ovsdb-server.log --pidfile=/var/run/openvswitch/ovsdb-server.pid --detach --monitor
root     1008  0.1  0.8 242948 31712 ?        S<Ll Aug06  32:17 ovs-vswitchd unix:/var/run/openvswitch/db.sock -
vconsole:emer -vsyslog:err -vfile:info --mlockall --no-chdir --log-file=/var/log/openvswitch/ovs-vswitchd.log --
pidfile=/var/run/openvswitch/ovs-vswitchd.pid --detach --monitor


root@popsuper1982:~# lsmod  | grep openvswitch
openvswitch           66901  0
gre                   13808  1 openvswitch
vxlan                 37619  1 openvswitch
libcrc32c             12644  2 btrfs,openvswitch
```

实验一：查看Openvswitch的架构

```
root@popsuper1982:~# lsof -p 985
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/107/gvfs
      Output information may be incomplete.
COMMAND   PID USER   FD   TYPE             DEVICE SIZE/OFF     NODE NAME
ovsdb-ser 985 root  cwd    DIR                8,1     4096        2 /
ovsdb-ser 985 root  rtd    DIR                8,1     4096        2 /
ovsdb-ser 985 root  txt    REG                8,1   834416 14690596 /usr/sbin/ovsdb-server
ovsdb-ser 985 root  mem    REG                8,1    14664  4718776 /lib/x86_64-linux-gnu/libdl-2.19.so
ovsdb-ser 985 root  mem    REG                8,1  1845024  4718760 /lib/x86_64-linux-gnu/libc-2.19.so
ovsdb-ser 985 root  mem    REG                8,1  1071552  4718798 /lib/x86_64-linux-gnu/libm-2.19.so
ovsdb-ser 985 root  mem    REG                8,1    31792  4718856 /lib/x86_64-linux-gnu/librt-2.19.so
ovsdb-ser 985 root  mem    REG                8,1   141574  4718850 /lib/x86_64-linux-gnu/libpthread-2.19.so
ovsdb-ser 985 root  mem    REG                8,1  1926432  4718772 /lib/x86_64-linux-gnu/libcrypto.so.1.0.0
ovsdb-ser 985 root  mem    REG                8,1   382984  4718864 /lib/x86_64-linux-gnu/libssl.so.1.0.0
ovsdb-ser 985 root  mem    REG                8,1   149120  4718740 /lib/x86_64-linux-gnu/ld-2.19.so
ovsdb-ser 985 root    0u   CHR                1,3      0t0     2051 /dev/null
ovsdb-ser 985 root    1u   CHR                1,3      0t0     2051 /dev/null
ovsdb-ser 985 root    2u   CHR                1,3      0t0     2051 /dev/null
ovsdb-ser 985 root    3r  FIFO                0,8      0t0    11919 pipe
ovsdb-ser 985 root    4w  FIFO                0,8      0t0    11919 pipe
ovsdb-ser 985 root    6u  unix 0xffff8800b31dfa80      0t0    11920 socket
ovsdb-ser 985 root    7u   REG                8,1      141  4194311 /tmp/tmpfRMzvId (deleted)
ovsdb-ser 985 root    8uW  REG               0,16        4     9945 /run/openvswitch/ovsdb-server.pid
ovsdb-ser 985 root    9w  FIFO                0,8      0t0    10572 pipe
ovsdb-ser 985 root   10u   CHR                1,3      0t0     2051 /dev/null
ovsdb-ser 985 root   11r  FIFO                0,8      0t0     9946 pipe
ovsdb-ser 985 root   12w  FIFO                0,8      0t0     9946 pipe
ovsdb-ser 985 root   13uW  REG                8,1        0  9702031 /etc/openvswitch/.conf.db.~lock~
ovsdb-ser 985 root   14u   REG                8,1    29782  9702032 /etc/openvswitch/conf.db
ovsdb-ser 985 root   15u  unix 0xffff88012c635400      0t0     9947 /var/run/openvswitch/db.sock
ovsdb-ser 985 root   16u  unix 0xffff88012c637700      0t0     9949 /var/run/openvswitch/ovsdb-server.985.ctl
ovsdb-ser 985 root   17u  unix 0xffff880036063480      0t0     9975 /var/run/openvswitch/db.sock
ovsdb-ser 985 root   19w   REG                8,1       95 11798904 /var/log/openvswitch/ovsdb-server.log
```

实验一： 查看Openvswitch的架构

```
root@popsuper1982:~# lsof -p 1008
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/107/gvfs
      Output information may be incomplete.
COMMAND    PID USER   FD      TYPE             DEVICE SIZE/OFF     NODE NAME
ovs-vswit 1008 root  cwd       DIR                8,1     4096        2 /
ovs-vswit 1008 root  rtd       DIR                8,1     4096        2 /
ovs-vswit 1008 root  txt       REG                8,1  1339776 14690597 /usr/sbin/ovs-vswitchd
ovs-vswit 1008 root  mem       REG                8,1    14664  4718776 /lib/x86_64-linux-gnu/libdl-2.19.so
ovs-vswit 1008 root  mem       REG                8,1  1845024  4718760 /lib/x86_64-linux-gnu/libc-2.19.so
ovs-vswit 1008 root  mem       REG                8,1  1071552  4718798 /lib/x86_64-linux-gnu/libm-2.19.so
ovs-vswit 1008 root  mem       REG                8,1    31792  4718856 /lib/x86_64-linux-gnu/librt-2.19.so
ovs-vswit 1008 root  mem       REG                8,1   141574  4718850 /lib/x86_64-linux-gnu/libpthread-2.19.so
ovs-vswit 1008 root  mem       REG                8,1  1926432  4718772 /lib/x86_64-linux-gnu/libcrypto.so.1.0.0
ovs-vswit 1008 root  mem       REG                8,1   382984  4718864 /lib/x86_64-linux-gnu/libssl.so.1.0.0
ovs-vswit 1008 root  mem       REG                8,1   149120  4718740 /lib/x86_64-linux-gnu/ld-2.19.so
ovs-vswit 1008 root    0u      CHR                1,3      0t0     2051 /dev/null
ovs-vswit 1008 root    1u      CHR                1,3      0t0     2051 /dev/null
ovs-vswit 1008 root    2u      CHR                1,3      0t0     2051 /dev/null
ovs-vswit 1008 root    4u     unix 0xffff880036065e80      0t0    13321 socket
ovs-vswit 1008 root    5r     FIFO               0,8      0t0     9971 pipe
ovs-vswit 1008 root    6w     FIFO               0,8      0t0     9971 pipe
ovs-vswit 1008 root    7r     FIFO               0,8      0t0     9972 pipe
ovs-vswit 1008 root    8w     FIFO               0,8      0t0     9972 pipe
ovs-vswit 1008 root    9uW     REG              0,16        5    11945 /run/openvswitch/ovs-vswitchd.pid
ovs-vswit 1008 root   10w     FIFO               0,8      0t0    13325 pipe
ovs-vswit 1008 root   11u      CHR               1,3      0t0     2051 /dev/null
ovs-vswit 1008 root   12r     FIFO               0,8      0t0    11946 pipe
ovs-vswit 1008 root   13w     FIFO               0,8      0t0    11946 pipe
ovs-vswit 1008 root   14u     unix 0xffff8800360646 00      0t0    13326 /var/run/openvswitch/ovs-vswitchd.1008.ctl
ovs-vswit 1008 root   15u     unix 0xffff880036066c80      0t0    13328 socket
ovs-vswit 1008 root   16u  netlink                         0t0    13336 GENERIC
ovs-vswit 1008 root   17r     FIFO               0,8      0t0    13337 pipe
ovs-vswit 1008 root   18w     FIFO               0,8      0t0    13337 pipe
ovs-vswit 1008 root   19u     0000               0,9        0     7353 anon_inode
ovs-vswit 1008 root   20u  netlink                         0t0    13338 GENERIC
ovs-vswit 1008 root   21u  netlink                         0t0    13339 GENERIC
ovs-vswit 1008 root   22u     sock               0,7      0t0    10592 can't identify protocol
ovs-vswit 1008 root   23u  netlink                         0t0    10595 GENERIC
ovs-vswit 1008 root   24u  netlink                         0t0    10596 ROUTE
ovs-vswit 1008 root   25u     unix 0xffff880131585e80      0t0    10597 /var/run/openvswitch/ubuntu_br.mgmt
ovs-vswit 1008 root   26u     unix 0xffff880131584d00      0t0    10599 /var/run/openvswitch/ubuntu_br.snoop
ovs-vswit 1008 root   27u  netlink                         0t0    10602 ROUTE
ovs-vswit 1008 root   28r     FIFO               0,8      0t0    10603 pipe
ovs-vswit 1008 root   29w     FIFO               0,8      0t0    10603 pipe
ovs-vswit 1008 root   30u  netlink                         0t0    10604 GENERIC
ovs-vswit 1008 root   32u  netlink                         0t0 30060146 GENERIC
ovs-vswit 1008 root   33w      REG                8,1       95 11797655 /var/log/openvswitch/ovs-vswitchd.log
```

实验一：查看Openvswitch的架构

- strace ovs-vsctl show

```
//建立unix socket，和ovs-dbserver进行通信
socket(PF_LOCAL, SOCK_STREAM, 0)       = 3
fcntl(3, F_GETFL)                      = 0x2 (flags O_RDWR)
fcntl(3, F_SETFL, O_RDWR|O_NONBLOCK)   = 0
connect(3, {sa_family=AF_LOCAL, sun_path="/var/run/openvswitch/db.sock"}, 31) = 0
//写入命令
write(3, "{\"method\":\"monitor\",\"id\":0,\"para"..., 409) = 409
//读取结果
read(3, "{\"id\":0,\"result\":{\"Port\":{\"8afee"..., 512) = 512
read(3, "7a8\"],[\"uuid\",\"8afee51e-6e71-4d4"..., 512) = 501
read(3, 0x24b4d05, 11)                 = -1 EAGAIN (Resource temporarily unavailable)
向终端输出结果
write(1, "c1fe4192-ae6a-457f-a2e1-dfc67284"..., 37c1fe4192-ae6a-457f-a2e1-dfc6728470eb) = 37
write(1, "    Bridge ubuntu_br\n", 21    Bridge ubuntu_br)  = 21
write(1, "        Port ubuntu_br\n", 23        Port ubuntu_br) = 23
write(1, "            Interface ubuntu_br\n", 32            Interface ubuntu_br) = 32
write(1, "                type: internal\n", 31                type: internal) = 31
write(1, "        Port \"vnet0\"\n", 21        Port "vnet0") = 21
write(1, "            Interface \"vnet0\"\n", 30            Interface "vnet0") = 30
write(1, "    ovs_version: \"2.0.1\"\n", 25    ovs_version: "2.0.1") = 25
```

实验一：查看Openvswitch的架构

- strace ovs-dpctl show

```
socket(PF_NETLINK, SOCK_RAW, 16)        = 3
setsockopt(3, SOL_SOCKET, 0x21 /* SO_??? */, [1048576], 4) = 0
getsockopt(3, SOL_SOCKET, SO_RCVBUF, [2097152], [4]) = 0
connect(3, {sa_family=AF_NETLINK, pid=0, groups=00000000}, 12) = 0
getsockname(3, {sa_family=AF_NETLINK, pid=11980, groups=00000000}, [12]) = 0
sendmsg(3, {msg_name(0)=NULL, msg_iov(1)=[{"(\0\0\0\20\0\1\0\1\0\0\0\314.\0\0\3\1\0\0\21\0\2\0ovs_data"..., 40}], msg_controllen=0, msg_flags=0}, 0) = 40
recvmsg(3, {msg_name(0)=NULL, msg_iov(2)=[{"\300\0\0\0\20\0\0\0\1\0\0\0\314.\0\0\1\2\0\0\21\0\2\0ovs_data"..., 1024}, {"", 65536}], msg_controllen=0, msg_flags=0}, MSG_DONTWAIT) = 192
close(3)                                = 0
```

```
write(1, "system@ovs-system:\n", 19system@ovs-system:
)     = 19
sendmsg(3, {msg_name(0)=NULL, msg_iov(1)=[{"\30\0\0\0\33\0\t\0\3\0\0\0\314.\0\0\3\1\0\0\4\0\0\0", 24}], msg_controllen=0, msg_flags=0}, 0) = 24
recvmsg(3, {msg_name(0)=NULL, msg_iov(2)=[{"p\0\0\0\33\0\0\0\3\0\0\0\314.\0\0\1\1\0\0\4\0\0\0\17\0\1\0ovs-"..., 1024}, {"", 65536}], msg_controllen=0, msg_flags=0}, MSG_DONTWAIT) = 112
write(1, "\tlookups: hit:3046 missed:40687 "..., 39     lookups: hit:3046 missed:40687 lost:0
) = 39
write(1, "\tflows: 1\n", 10     flows: 1
)            = 10
```

```
socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) = 6
ioctl(6, SIOCGIFFLAGS, {ifr_name="ovs-system", ifr_flags=IFF_BROADCAST|IFF_MULTICAST}) = 0
write(1, "\tport 0: ovs-system (internal)\n", 31        port 0: ovs-system (internal)
) = 31
ioctl(6, SIOCGIFFLAGS, {ifr_name="ubuntu_br", ifr_flags=IFF_UP|IFF_BROADCAST|IFF_RUNNING}) = 0
write(1, "\tport 1: ubuntu_br (internal)\n", 30 port 1: ubuntu_br (internal)
) = 30
write(1, "\tport 2: vnet0\n", 15        port 2: vnet0
)       = 15
```

实验一：查看Openvswitch的架构

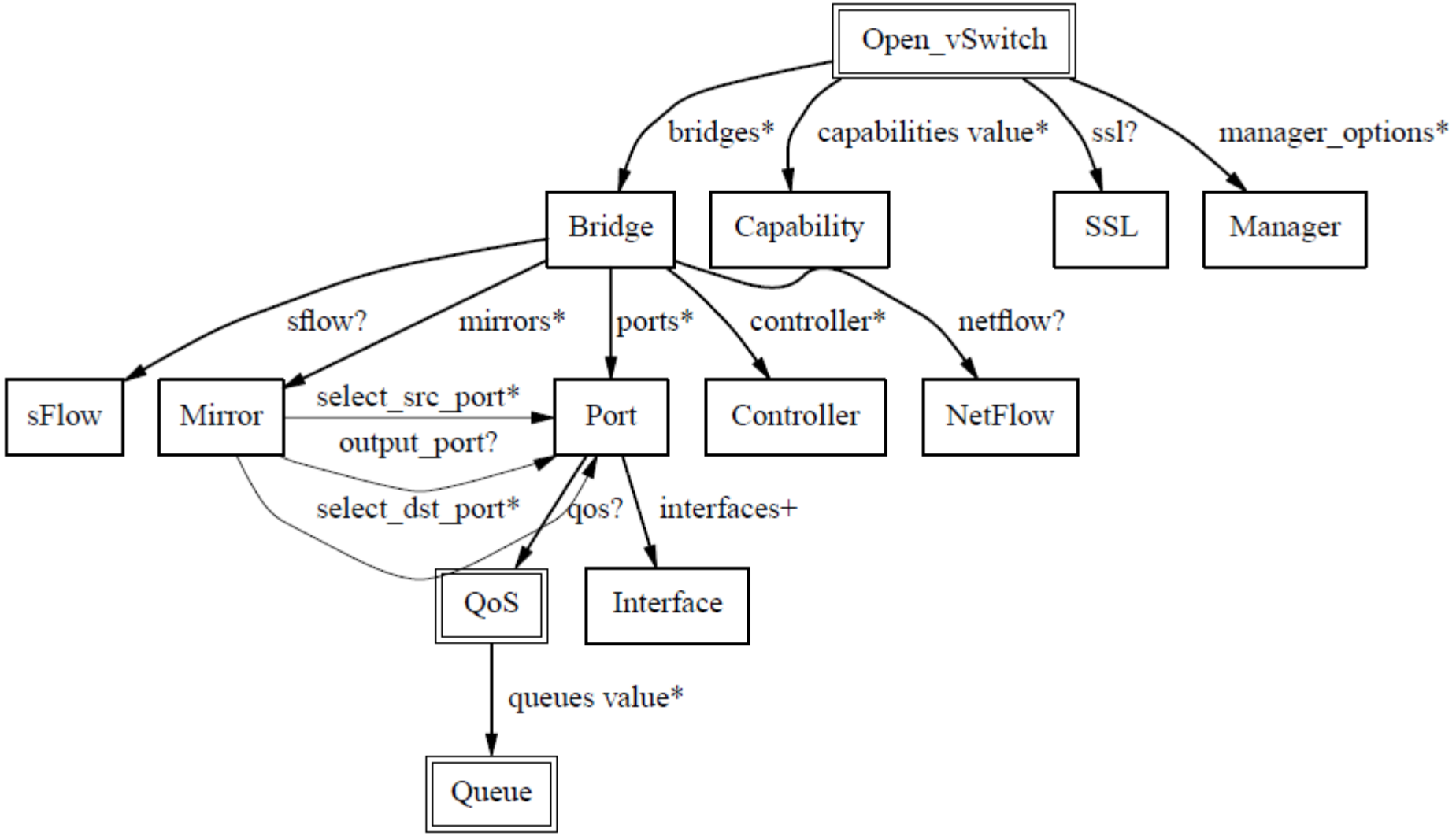
- # strace ovs-ofctl show ubuntu_br

socket(PF_LOCAL, SOCK_STREAM, 0)　　= 3
connect(3, {sa_family=AF_LOCAL, sun_path="/var/run/openvswitch/ubuntu_br"}, 33) = -1 ENOENT (No such file or directory)
close(3)　　= 0

socket(PF_LOCAL, SOCK_STREAM, 0)　　= 3
connect(3, {sa_family=AF_LOCAL, sun_path="/var/run/openvswitch/ubuntu_br.mgmt"}, 38) = 0

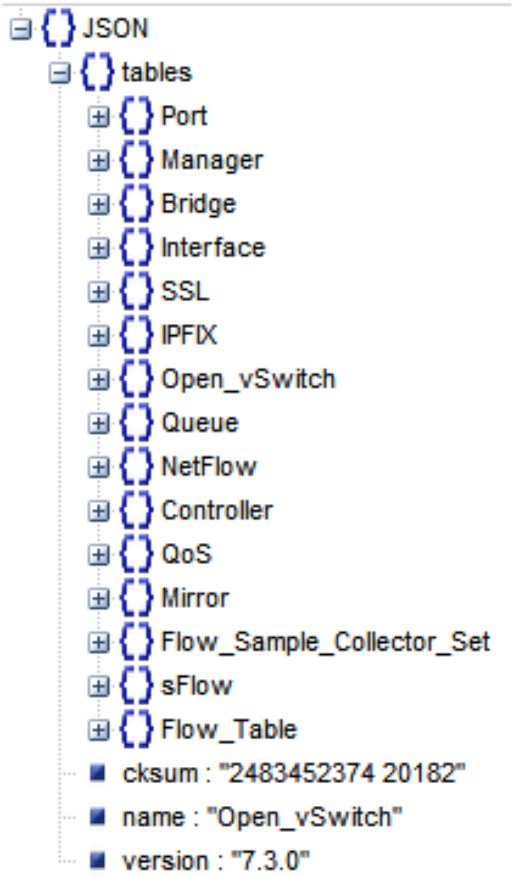
```
write(1, "OFPT_FEATURES_REPLY (xid=0x2): d"..., 53OFPT_FEATURES_REPLY (xid=0x2): dpid:0000f22408ed6741
) = 53
write(1, "n_tables:254, n_buffers:256\n", 28n_tables:254, n_buffers:256
) = 28
write(1, "capabilities: FLOW_STATS TABLE_S"..., 73capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
) = 73
write(1, "actions: OUTPUT SET_VLAN_VID SET"..., 138actions: 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 ENQUEUE
) = 138
write(1, " 2(vnet0): addr:fe:54:11:9b:d5:1"..., 34 2(vnet0): addr:fe:54:11:9b:d5:11
) = 34
write(1, "     config:     0\n", 19     config:     0
)    = 19
write(1, "     state:      0\n", 19     state:      0
)    = 19
write(1, "     current:    10MB-FD COPPER\n", 32     current:    10MB-FD COPPER
) = 32
write(1, "     speed: 10 Mbps now, 0 Mbps "..., 36     speed: 10 Mbps now, 0 Mbps max
) = 36
write(1, " LOCAL(ubuntu_br): addr:82:26:9a"..., 42 LOCAL(ubuntu_br): addr:82:26:9a:94:27:04
) = 42
write(1, "     config:     PORT_DOWN\n", 27     config:     PORT_DOWN
) = 27
write(1, "     state:      LINK_DOWN\n", 27     state:      LINK_DOWN
) = 27
write(1, "     speed: 0 Mbps now, 0 Mbps m"..., 35     speed: 0 Mbps now, 0 Mbps max
) = 35
```

Openvswitch数据库表结构



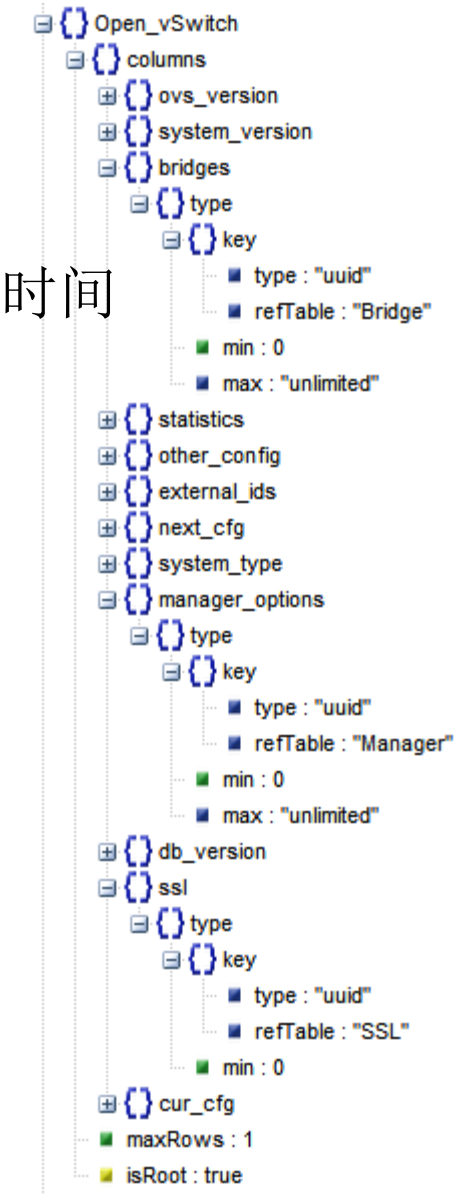
实验二：打印数据库表结构

- cat /etc/openvswitch/conf.db，我们会发现它是json格式的
- 数据库可以通过ovsdb-client dump将数据库内容打印出来



Openvswitch: Open_vSwitch表

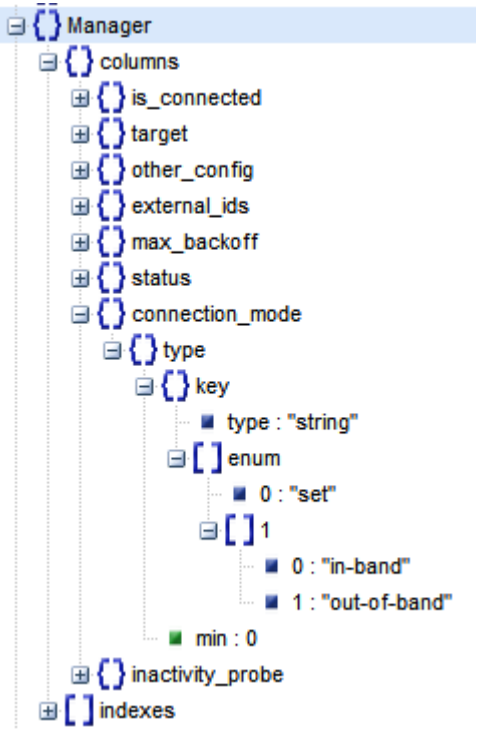
- 数据库的根
- 全局的配置项
  - other_config : stats-update-interval ： 将统计信息写入数据库的间隔时间
  - other_config : flow-limit ： 在flow table中flow entry的数量
  - other_config : n-handler-threads ： 用于处理新flow的线程数
  - other_config : n-revalidator-threads ： 用于验证flow的线程数.
  - other_config : enable-statistics 是否统计
    - statistics : cpu 统计cpu数量，线程
    - statistics : load_average system load
    - statistics : memory 总RAM，swap
    - statistics : process_NAME ： with NAME replaced by a process name， 统计memory size, cpu time等
    - statistics : file_systems： mount point, size, used
- 指向其他表
  - bridge表
  - SSL表
  - Manager表



Openvswitch: Manager

- Manager表配置的是ovsdb-server的
- ovsdb-server使用manager_options中的配置来监听端口，等待client来连接。
  - punix:file: 监听unix socket

```
root@popsupe-1982:~# ps aux | grep openvswitch
root         985  0.0  0.0  21172  2120 ?        S<   Aug06   1:23 ovsdb-server /etc/openvswitch/conf.db -vconsole:emer
emote=punix:/var/run/openvswitch/db.sock --private-key=db:Open_vSwitch,SSL,private_key --certificate=db:Open_vSwitch,S
a-cert=db:Open_vSwitch,SSL,ca_cert --no-chdir --log-file=/var/log/openvswitch/ovsdb-server.log --pidfile=/var/run/open
etach --monitor
root        1008  0.1  0.8 242948 31712 ?        S<Ll Aug06  33:27 ovs-vswitchd unix:/var/run/openvswitch/db.sock -vcons
:info --mlockall --no-chdir --log-file=/var/log/openvswitch/ovs-vswitchd.log --pidfile=/var/run/openvswitch/ovs-vswitcl
root       12238  0.0  0.0  11748   924 pts/0    S+   01:21   0:00 grep --color=auto openvswitch
```



  - ptcp:port[:ip]: 监听TCP连接
  - pssl:port[:ip]: 监听SSL连接

实验三：设置Manager的TCP连接

- ovs-vsctl set-manager ptcp:8881

```
root@popsuper1982:~# ovs-vsctl show
c1fe4192-ae6a-457f-a2e1-dfc6728470eb
    Bridge ubuntu_br
        Port ubuntu_br
            Interface ubuntu_br
                type: internal
        Port "vnet0"
            Interface "vnet0"
    ovs_version: "2.0.1"
```

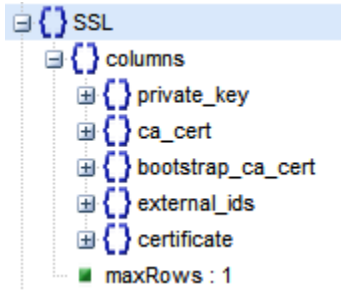
```
root@popsuper1982:~# ovs-vsctl show
c1fe4192-ae6a-457f-a2e1-dfc6728470eb
    Manager "ptcp:8881"
    Bridge ubuntu_br
        Port ubuntu_br
            Interface ubuntu_br
                type: internal
        Port "vnet0"
            Interface "vnet0"
    ovs_version: "2.0.1"
```

在另外一台机器上

```
root@openstackcliu8:~# ovs-vsctl show
71416dda-3f1d-49ee-8caa-c0adc2c8a20f
    ovs_version: "2.0.1"
root@openstackcliu8:~# ovs-vsctl --db=tcp:192.168.100.1:8881 show
c1fe4192-ae6a-457f-a2e1-dfc6728470eb
    Manager "ptcp:8881"
    Bridge ubuntu_br
        Port ubuntu_br
            Interface ubuntu_br
                type: internal
        Port "vnet0"
            Interface "vnet0"
    ovs version: "2.0.1"
```

Openvswitch: SSL

- SSL的配置主要包含几个部分：
  - Private Key: 私钥
  - Certificate: 证书
  - CA Certificate: CA的证书
  - private key和public key对，其中public key放在certificate中，并且需要CA使用自己的private key进行签名，CA来担保这个certificate是合法的，为了验证这个CA签名，当然需要CA的public key，而CA的public key是放在ca cert里面的，当然也需要被签名，被更高级的CA担保，或者自己担保自己。
  - bootstrap_ca_cert是一个boolean，如果是true，则每次启动的时候，都会向controller去拿最新的ca cert。




```
root@popsuper1982:~# ps aux | grep openvswitch
root       985  0.0  0.0  21172  2120 ?        S<   Aug06   1:23 ovsdb-server /etc/openvswitch/conf.db -vconsole:emer -vsyslog:err -vfile:info --r
emote=punix:/var/run/openvswitch/db.sock --private-key=db:Open_vSwitch,SSL,private_key --certificate=db:Open_vSwitch,SSL,certificate --bootstrap-c
a-cert=db:Open_vSwitch,SSL,ca_cert --no-chdir --log-file=/var/log/openvswitch/ovsdb-server.log --pidfile=/var/run/openvswitch/ovsdb-server.pid --d
etach --monitor
root      1008  0.1  0.8 242948 31712 ?        S<Ll Aug06  33:30 ovs-vswitchd unix:/var/run/openvswitch/db.sock -vconsole:emer -vsyslog:err -vfile
:info --mlockall --no-chdir --log-file=/var/log/openvswitch/ovs-vswitchd.log --pidfile=/var/run/openvswitch/ovs-vswitchd.pid --detach --monitor
root     12768  0.0  0.0  11748   924 pts/0    S+   02:05   0:00 grep --color=auto openvswitch
```

默认表是空的

实验四：设置SSL连接

- 生成private key, certificate, CA key, CA certificate
  1. 生成一个CA的private key

openssl genrsa -out caprivate.key 1024

```
root@popsuper1982:~/keys/openvswitch# openssl genrsa -out caprivate.key 1024
Generating RSA private key, 1024 bit long modulus
........++++++
..++++++
e is 65537 (0x10001)
```

  2. CA有一个certificate，里面放着CA的public key，要生成这个certificate，则需要写一个certificate request

openssl req -key caprivate.key -new -out cacertificate.req

```
root@popsuper1982:~/keys/openvswitch# openssl req -key caprivate.key -new -out cacertificate.req
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:CN
Locality Name (eg, city) []:CN
Organization Name (eg, company) [Internet Widgits Pty Ltd]:CN
Organizational Unit Name (eg, section) []:CN
Common Name (e.g. server FQDN or YOUR name) []:popsuper1982
Email Address []:popsuper1982@qq.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

实验四：设置SSL连接

3. 由于这里的CA是root CA，没有更高级的CA了，所以要进行自签发，用自己的private key对自己的certificate请求进行签发

openssl x509 -req -in cacertificate.req -signkey caprivate.key -out cacertificate.pem

```
root@popsuper1982:~/keys/openvswitch# openssl x509 -req -in cacertificate.req -signkey caprivate.key -out cacertificate.pem
Signature ok
subject=/C=CN/ST=CN/L=CN/O=CN/OU=CN/CN=popsuper1982/emailAddress=popsuper1982@qq.com
Getting Private key
```

4. 普通的机构需要有自己的private key

openssl genrsa -out cliu8private.key 1024

```
root@popsuper1982:~/keys/openvswitch# openssl genrsa -out cliu8private.key 1024
Generating RSA private key, 1024 bit long modulus
.....................................++++++
.........++++++
e is 65537 (0x10001)
```

5. 也需要一个证书，里面放自己的public key，需要一个证书请求

openssl req -key cliu8private.key -new -out cliu8certificate.req

```
root@popsuper1982:~/keys/openvswitch# openssl req -key cliu8private.key -new -out cliu8certificate.req
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:CN
Locality Name (eg, city) []:CN
Organization Name (eg, company) [Internet Widgits Pty Ltd]:CN
Organizational Unit Name (eg, section) []:CN
Common Name (e.g. server FQDN or YOUR name) []:cliu8
Email Address []:cliu8@qq.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

实验四：设置SSL连接

6. 要使得这个证书被认可，则需要一个CA对这个证书进行签名，我们用上面的CA的private key对他进行签名

openssl x509 -req -in cliu8certificate.req -CA cacertificate.pem -CAkey caprivate.key -out cliu8certificate.pem -CAcreateserial

```
root@popsuper1982:~/keys/openvswitch# openssl x509 -req -in cliu8certificate.req -CA cacertificate.pem -CAkey caprivate.key -out cliu8certificate.
pem -CAcreateserial
Signature ok
subject=/C=CN/ST=CN/L=CN/O=CN/OU=CN/CN=cliu8/emailAddress=cliu8@qq.com
Getting CA Private Key
```

实验四：设置SSL连接

- 设置manager

ovs-vsctl del-manager
ovs-vsctl set-manager pssl:8881
ovs-vsctl set-ssl /root/keys/openvswitch/cliu8private.key /root/keys/openvswitch/cliu8certificate.pem /root/keys/openvswitch/cacertificate.pem

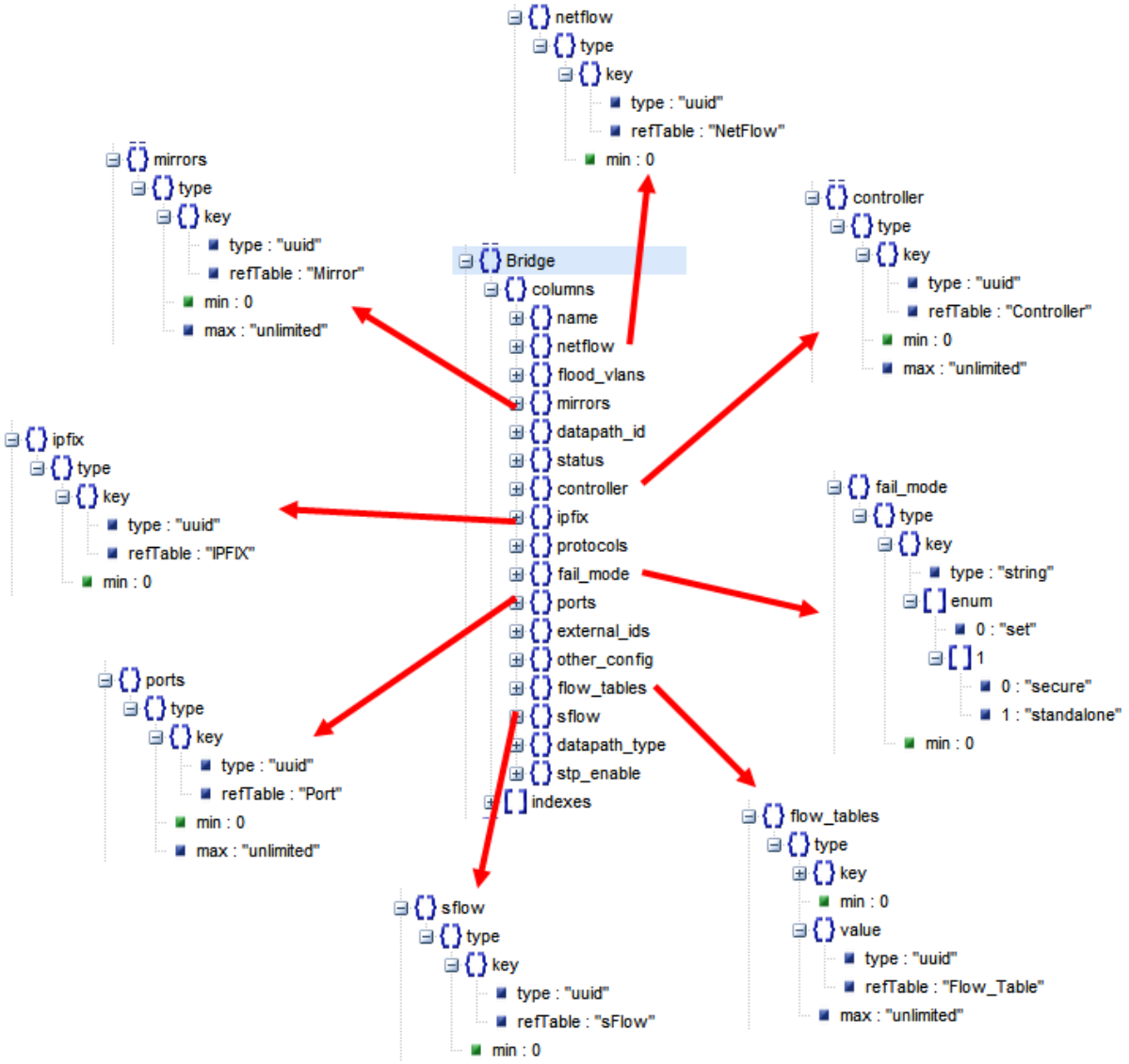
```
SSL table
_uuid                                bootstrap_ca_cert ca_cert                                  certificate                                  ex
ternal_ids private_key
------------------------------------ ----------------- ---------------------------------------- -------------------------------------------- --
---------- ----------------------------------------
5981c32d-0541-4bf2-b32c-415712618891 false             "/root/keys/openvswitch/cacertificate.pem" "/root/keys/openvswitch/cliu8certificate.pem" {}
           "/root/keys/openvswitch/cliu8private.key"
```

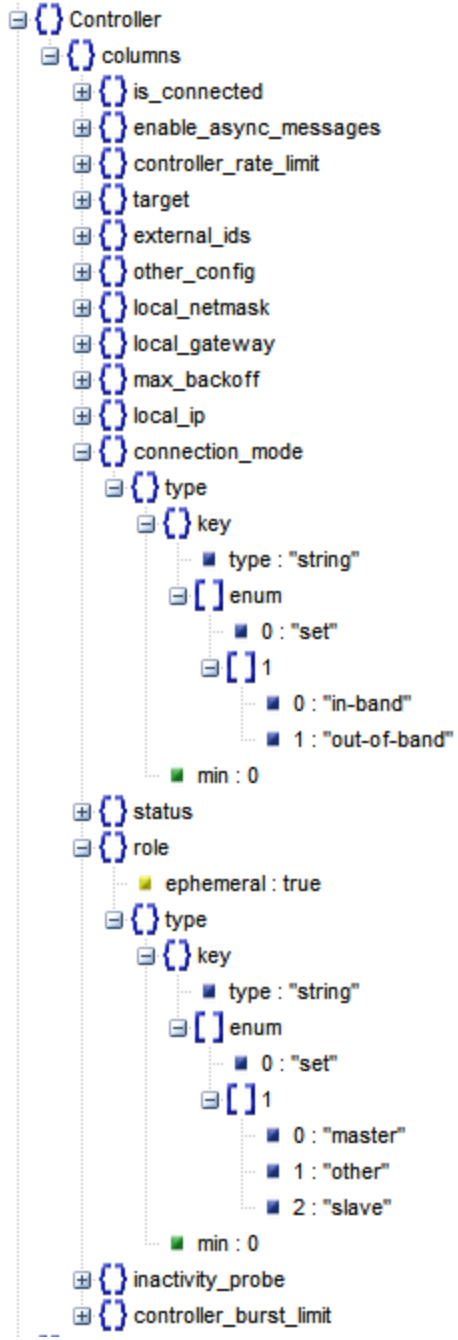
- 在另一台机器上

```
root@openstackcliu8:/home/openstack# ovs-vsctl --db=ssl:192.168.100.1:8881 show
2014-08-27T23:01:25Z|00001|stream_ssl|ERR|Private key must be configured to use SSL
2014-08-27T23:01:25Z|00002|stream_ssl|ERR|Certificate must be configured to use SSL
2014-08-27T23:01:25Z|00003|stream_ssl|ERR|CA certificate must be configured to use SSL
2014-08-27T23:01:25Z|00004|reconnect|WARN|ssl:192.168.100.1:8881: connection attempt failed (Protocol not available)
ovs-vsctl: ssl:192.168.100.1:8881: database connection failed (Protocol not available)
```

```
root@openstackcliu8:~/keys/openvswitch# ovs-vsctl --db=ssl:192.168.100.1:8881 --private-key=cliu8private.key --certificate=cliu8certificate.pem --
ca-cert=cacertificate.pem  show
c1fe4192-ae6a-457f-a2e1-dfc6728470eb
    Manager "pssl:8881"
        is_connected: true
    Bridge ubuntu_br
        Port ubuntu_br
            Interface ubuntu_br
                type: internal
        Port "vnet0"
            Interface "vnet0"
    ovs_version: "2.0.1"
```

Openvswitch: Controller





Openvswitch: Controller

- OpenFlow配置项：从架构图中我们可以看出，openvwitch的一个bridge可以通过openflow协议，被一个统一的controller管理的
  - Controller
  - flow_tables
  - fail_mode:
    - 一旦一个bridge连到一个openflow controller，则flow table就由controller统一管理，如果连接断了
    - secure： 这个bridge会试图一直连接controller，并不自己建立flow table
    - standalone：一旦bridge三次连不上controller，就自己建立和管理flow table
  - datapath_id：

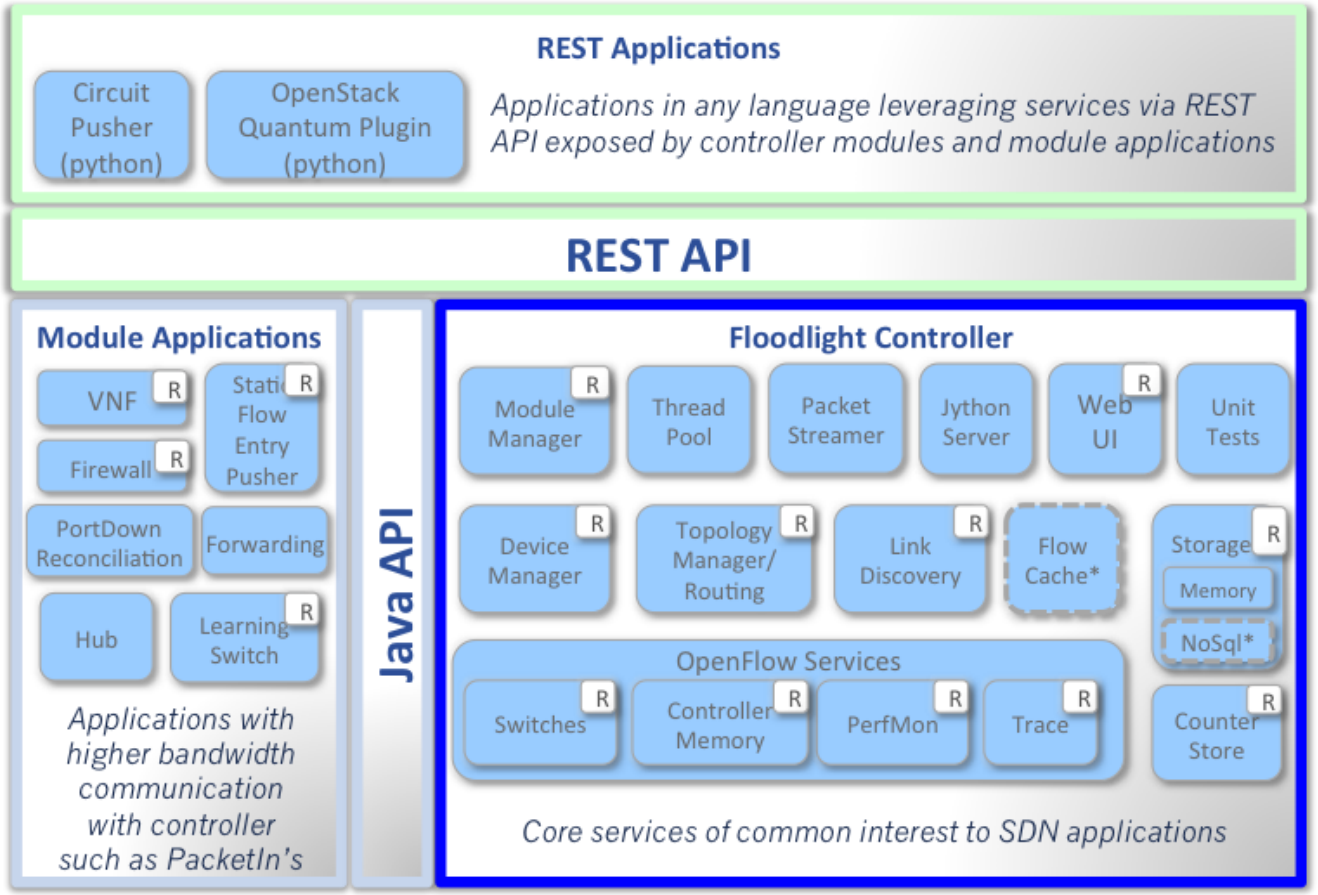
Openvswitch: Controller

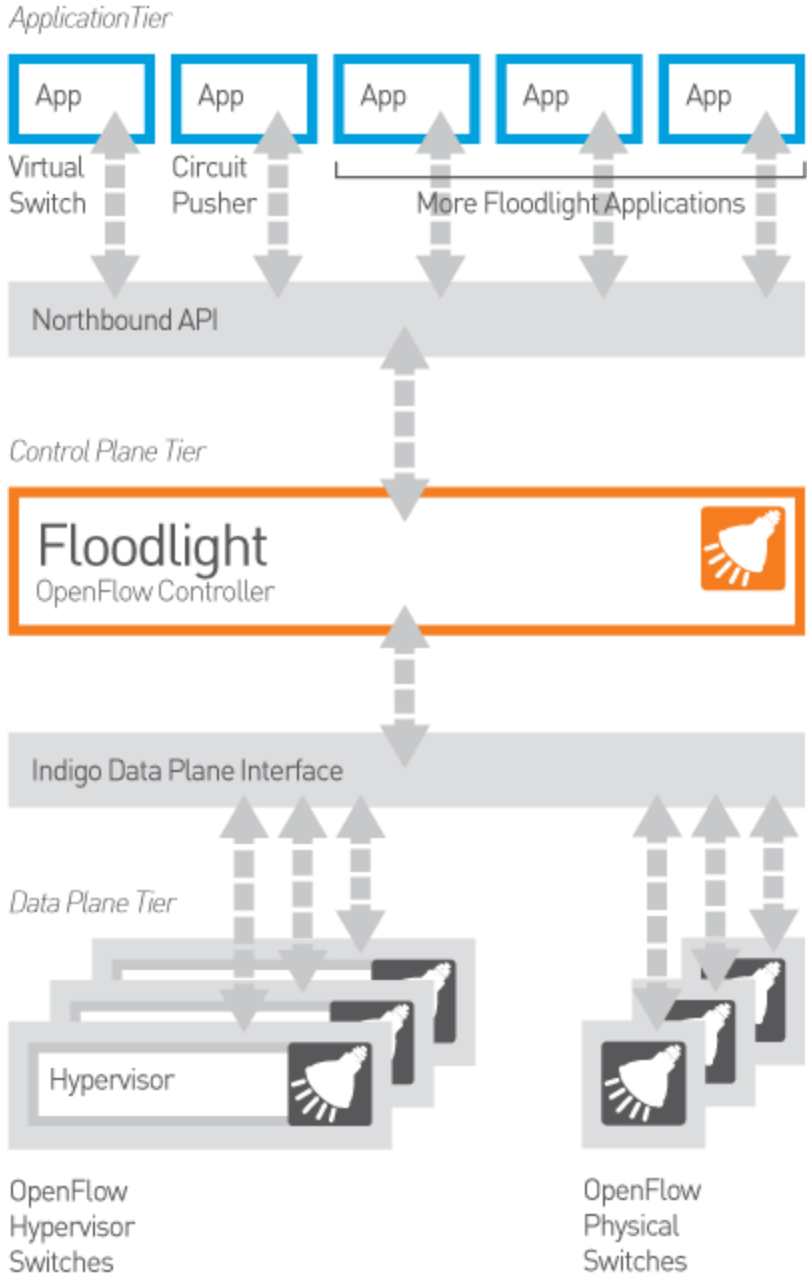
- OpenFlow Controller多种多样
- http://groups.geni.net/geni/wiki/OpenFlow/Controllers

- Beacon is a Java-based controller that supports both event-based and threaded operation. Beacon was developed at Stanford.
- Floodlight is a Java-based controller that was forked from the Beacon controller, and now is supported by a community of developers. Floodlight is released under the Apache License.
- Maestro is a multi-threaded Java-based platform that allows developers to implement new OpenFlow controllers. Maestro was developed at Rice University.
- NodeFlow is an OpenFlow controller written in pure JavaScript for Node.JS. Node.JS provides an asynchronous library over JavaScript for server side programming which is perfect for writing network based applications.
- NOX is a C++ based platform that gives the ability to developers to implement new controllers by writing NOX modules in either C++.
- POX is a Python based platform that gives the ability to developers to implement new controllers by writing NOX modules in either Python. Pox was part of what is now called Nox classic, but it was separated into a different controller platform that only supports Python.
- Trema is a C based platform that allows developers to write new controllers by writing Trema modules in either C or Ruby. Trema was developed by NEC.

Openvswitch: Controller

- 使用Floodlight





实验五：配置使用OpenFlow Controller

- 创建三个虚拟机

```
root@popsuper1982:/home/openstack/images# virsh list
 Id    Name                           State
----------------------------------------------------
 7     Instance01                     running
 8     Instance02                     running
 9     Instance03                     running
```

```
<interface type='bridge'>
  <mac address='52:54:00:9b:d5:11'/>
  <source bridge='ubuntu_br'/>
  <virtualport type='openvswitch'>
    <parameters interfaceid='211adaaf-6731-45fe-b9a6-40fac80b8795'/>
  </virtualport>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

```
<interface type='bridge'>
  <mac address='52:54:00:9b:d5:33'/>
  <source bridge='ubuntu_br'/>
  <virtualport type='openvswitch'>
    <parameters interfaceid='0831f7ee-3f99-4a63-8d51-cd45fe0a510e'/>
  </virtualport>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

```
<interface type='bridge'>
  <mac address='52:54:00:9b:d5:77'/>
  <source bridge='ubuntu_br'/>
  <virtualport type='openvswitch'>
    <parameters interfaceid='d3e82976-0944-4c66-894f-48d31331e1e5'/>
  </virtualport>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

QEMU (Instance01)

```
root@openstackcliu8:/home/openstack# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP grou
    link/ether 52:54:00:9b:d5:11 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.100/22 brd 192.168.100.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe9b:d511/64 scope link
       valid_lft forever preferred_lft forever
root@openstackcliu8:/home/openstack#
```

QEMU (Instance02)

```
root@openstackcliu8:/home/openstack# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP grou
    link/ether 52:54:00:9b:d5:33 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.101/22 brd 192.168.100.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe9b:d533/64 scope link
       valid_lft forever preferred_lft forever
root@openstackcliu8:/home/openstack#
```

QEMU (Instance03)

```
root@openstackcliu8:/home/openstack# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP grou
    link/ether 52:54:00:9b:d5:77 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.102/22 brd 192.168.100.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe9b:d577/64 scope link
       valid_lft forever preferred_lft forever
root@openstackcliu8:/home/openstack#
```

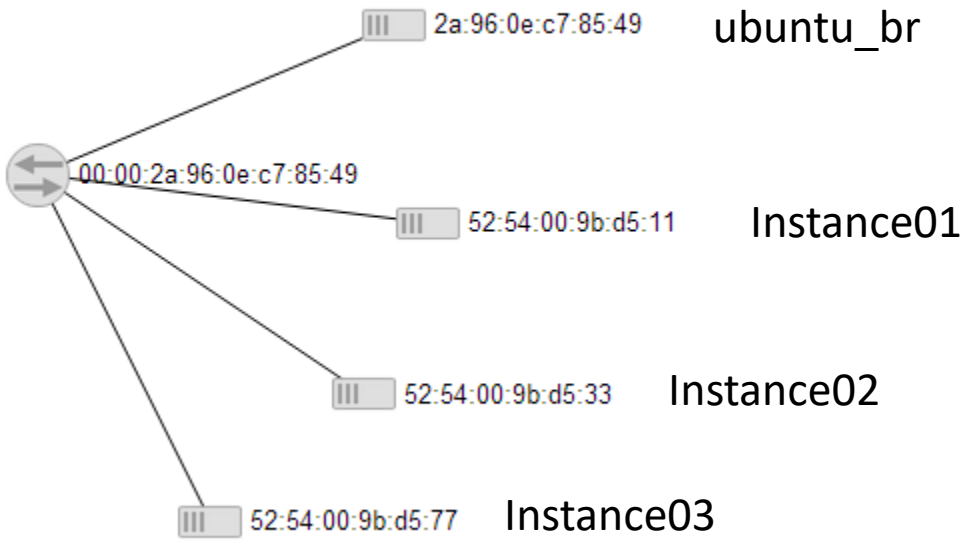
实验五：配置使用OpenFlow Controller

- 安装floodlight
  - http://www.projectfloodlight.org/getting-started/
  - git clone git://github.com/floodlight/floodlight.git
  - cd floodlight/
  - ant
  - nohup java -jar target/floodlight.jar > floodlight.log 2>&1 &
- 设置Controller
  - ovs-vsctl set-controller ubuntu_br tcp:192.168.100.1:6633

```
root@popsuper1982:/home/openstack/floodlight# ovs-vsctl show
d4e05278-c21f-4ed4-be3a-7853ea971931
    Bridge ubuntu_br
        Controller "tcp:16.158.166.150:6633"
            is_connected: true
        Port "vnet0"
            Interface "vnet0"
        Port "vnet2"
            Interface "vnet2"
        Port ubuntu_br
            Interface ubuntu_br
                type: internal
        Port "vnet1"
            Interface "vnet1"
    ovs_version: "2.0.1"
```

实验五：配置使用OpenFlow Controller

- 访问floodlight的界面
  - http://16.158.166.150:8080/ui/index.html
- Floodlight的Rest API
  - http://docs.projectfloodlight.org/display/floodlightcontroller/Floodlight+REST+API
- 默认情况下，三台机器可以相互ping的通




```
root@popsuper1982:/home/openstack/floodlight# ovs-ofctl show ubuntu_br
OFPT_FEATURES_REPLY (xid=0x2): dpid:00002a960ec78549
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_I
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_
 10(vnet0): addr:fe:54:00:9b:d5:11
     config:     0
     state:      0
     current:    10MB-FD COPPER
     speed: 10 Mbps now, 0 Mbps max
 11(vnet1): addr:fe:54:00:9b:d5:33
     config:     0
     state:      0
     current:    10MB-FD COPPER
     speed: 10 Mbps now, 0 Mbps max
 12(vnet2): addr:fe:54:00:9b:d5:77
     config:     0
     state:      0
     current:    10MB-FD COPPER
     speed: 10 Mbps now, 0 Mbps max
 LOCAL(ubuntu_br): addr:2a:96:0e:c7:85:49
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

实验五：配置使用OpenFlow Controller

- 调用Rest API设定规则，只允许Instance01和Instance03之间相互通信

curl -d '{"switch": "00:00:2a:96:0e:c7:85:49", "name":"static-flow1", "cookie":"0", "priority":"32768", "src-mac":"52:54:00:9b:d5:11","active":"true", "actions":"output=12"}' http://16.158.166.150:8080/wm/staticflowentrypusher/json

curl -d '{"switch": "00:00:2a:96:0e:c7:85:49", "name":"static-flow2", "cookie":"0", "priority":"32768", "src-mac":"52:54:00:9b:d5:77","active":"true", "actions":"output=10"}' http://16.158.166.150:8080/wm/staticflowentrypusher/json

```
root@popsuper1982:/home/openstack/floodlight# ovs-ofctl dump-flows ubuntu_br
NXST_FLOW reply (xid=0x4):
 cookie=0xa00000d7014f80, duration=8.393s, table=0, n_packets=0, n_bytes=0, idle_age=8, dl_src=52:54:00:9b:d5:77 actions=output:10
 cookie=0xa00000d7014f7f, duration=20.814s, table=0, n_packets=0, n_bytes=0, idle_age=20, dl_src=52:54:00:9b:d5:11 actions=output:12
```

Ports (4)

#	Link Status	TX Bytes	RX Bytes	TX Pkts	RX Pkts	Dropped	Errors
10 (vnet0)	UP 10 Mbps FDX	3620310	193223	2992	2044	0	0
12 (vnet2)	UP 10 Mbps FDX	3599404	189683	2674	1991	0	0
11 (vnet1)	UP 10 Mbps FDX	3615364	190617	2927	2022	0	0
65534 (ubuntu_br)	UP	566541	10718256	5974	6842	0	0

Flows (2)

Cookie	Priority	Match	Action	Packets	Bytes	Age	Timeout
45035999880892290	-32768	src=52:54:00:9b:d5:77	output 10	0	0	174 s	0 s
45035999880892290	-32768	src=52:54:00:9b:d5:11	output 12	0	0	187 s	0 s

实验五：配置使用OpenFlow Controller

- 用REST API清除所有规则

curl http://16.158.166.150:8080/wm/staticflowentrypusher/clear/00:00:2a:96:0e:c7:85:49/json

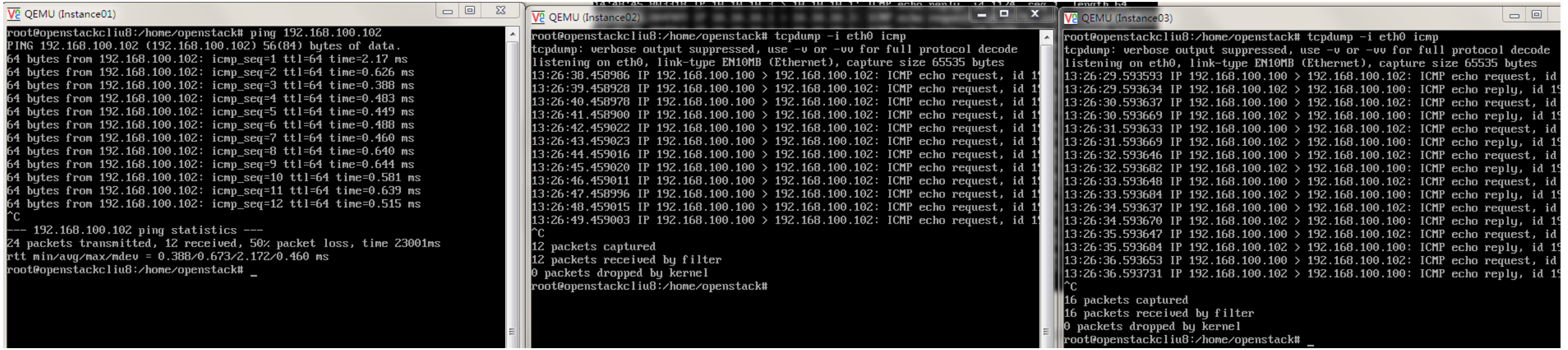
- 将正确的mac导向正确的port

curl -d '{"switch": "00:00:2a:96:0e:c7:85:49", "name":"static-flow1", "cookie":"0", "priority":"32768", "dst-mac":"52:54:00:9b:d5:11","active":"true", "actions":"output=10"}'
http://16.158.166.150:8080/wm/staticflowentrypusher/json
curl -d '{"switch": "00:00:2a:96:0e:c7:85:49", "name":"static-flow2", "cookie":"0", "priority":"32768", "dst-mac":"52:54:00:9b:d5:33","active":"true", "actions":"output=11"}'
http://16.158.166.150:8080/wm/staticflowentrypusher/json
curl -d '{"switch": "00:00:2a:96:0e:c7:85:49", "name":"static-flow3", "cookie":"0", "priority":"32768", "dst-mac":"52:54:00:9b:d5:77","active":"true", "actions":"output=12"}'
http://16.158.166.150:8080/wm/staticflowentrypusher/json

实验五：配置使用OpenFlow Controller

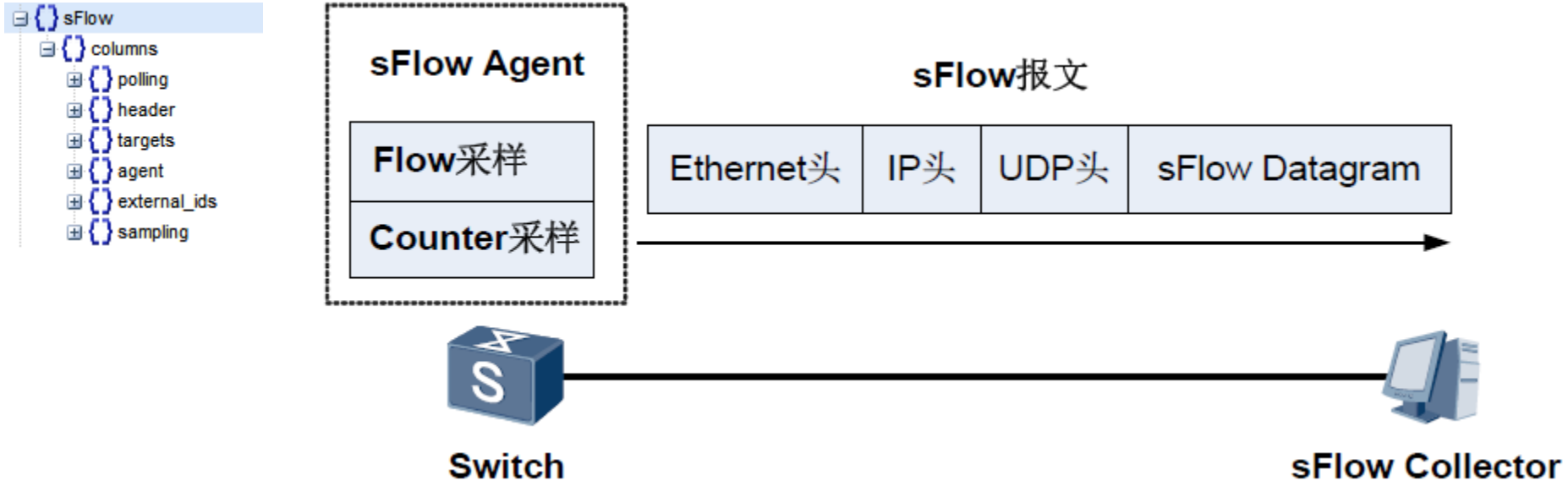
- 从Instance01来ping Instance03，用tcpdump监听Instance02和Instance03，在这个过程中，用REST API将Instance03的包转发给Instance02

curl -d '{"switch": "00:00:2a:96:0e:c7:85:49", "name":"static-flow3", "cookie":"0", "priority":"32768", "dst-mac":"52:54:00:9b:d5:77","active":"true", "actions":"output=11"}'
http://16.158.166.150:8080/wm/staticflowentrypusher/json



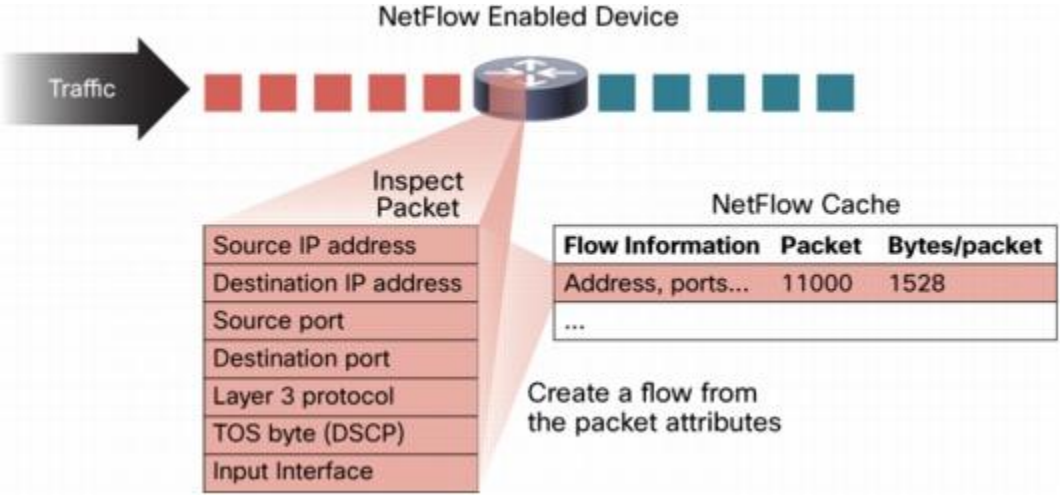
Openvswitch: sFlow, NetFlow/IPFIX

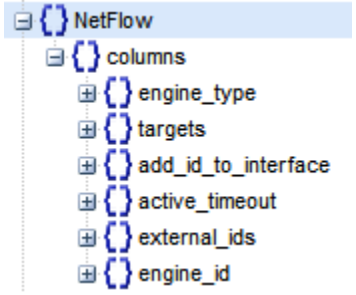
- 采样流sFlow（Sampled Flow）是一种基于报文采样的网络流量监控技术，主要用于对网络流量进行统计分析。
  - Flow 采样是sFlow Agent 设备在指定端口上按照特定的采样方向和采样比对报文进行采样分析, 该采样方式主要是关注流量的细节，这样就可以监控和分析网络上的流行为。
  - Counter采样是sFlow Agent设备周期性的获取接口上的流量统计信息, 只关注接口上流量的量，而不关注流量的详细信息。

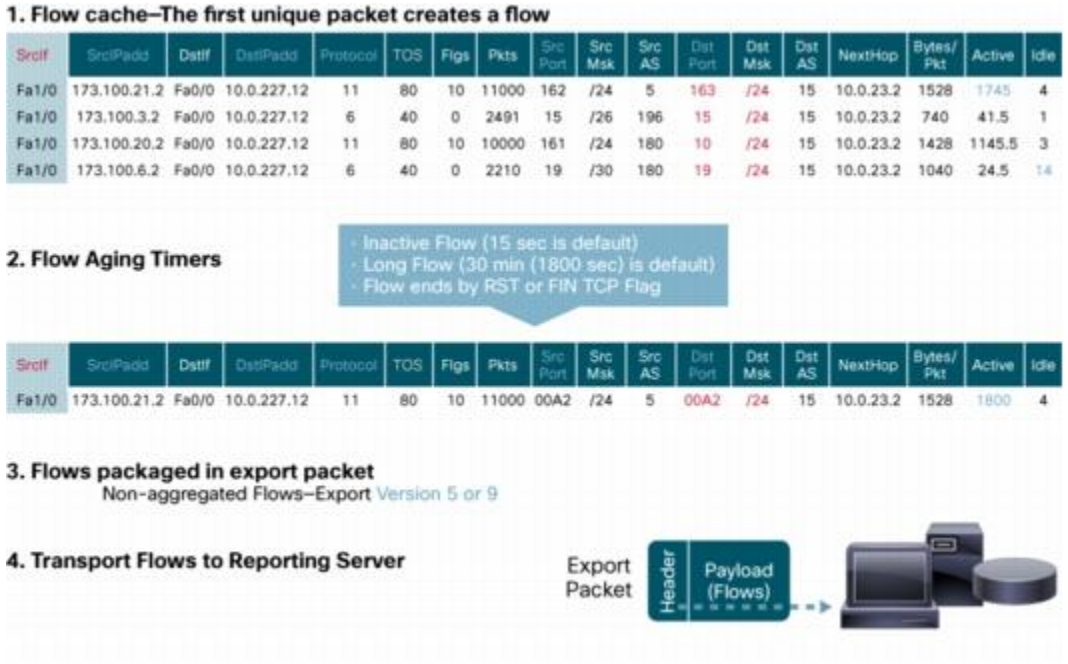


Openvswitch: sFlow, NetFlow/IPFIX

- Cisco NetFlow and IPFIX (the IETF standard based on NetFlow) 也是一个协议，将流量记录发送给服务器

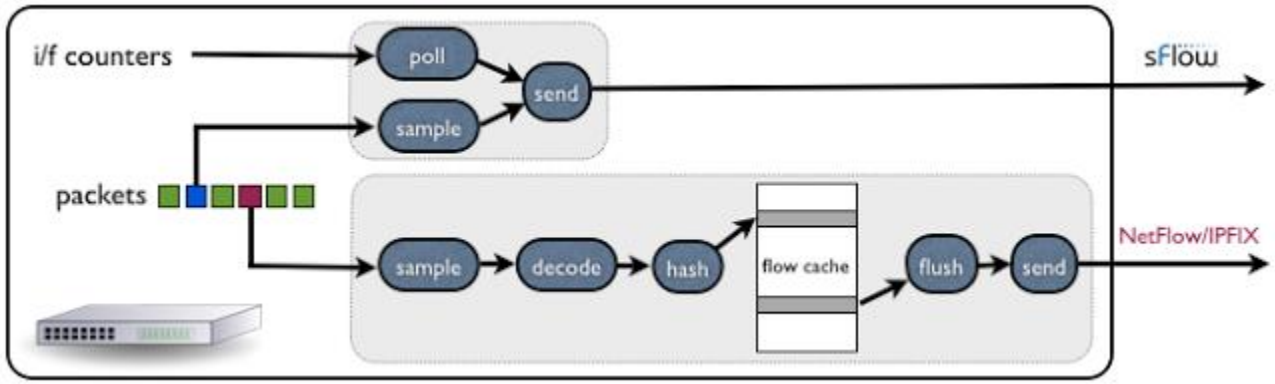




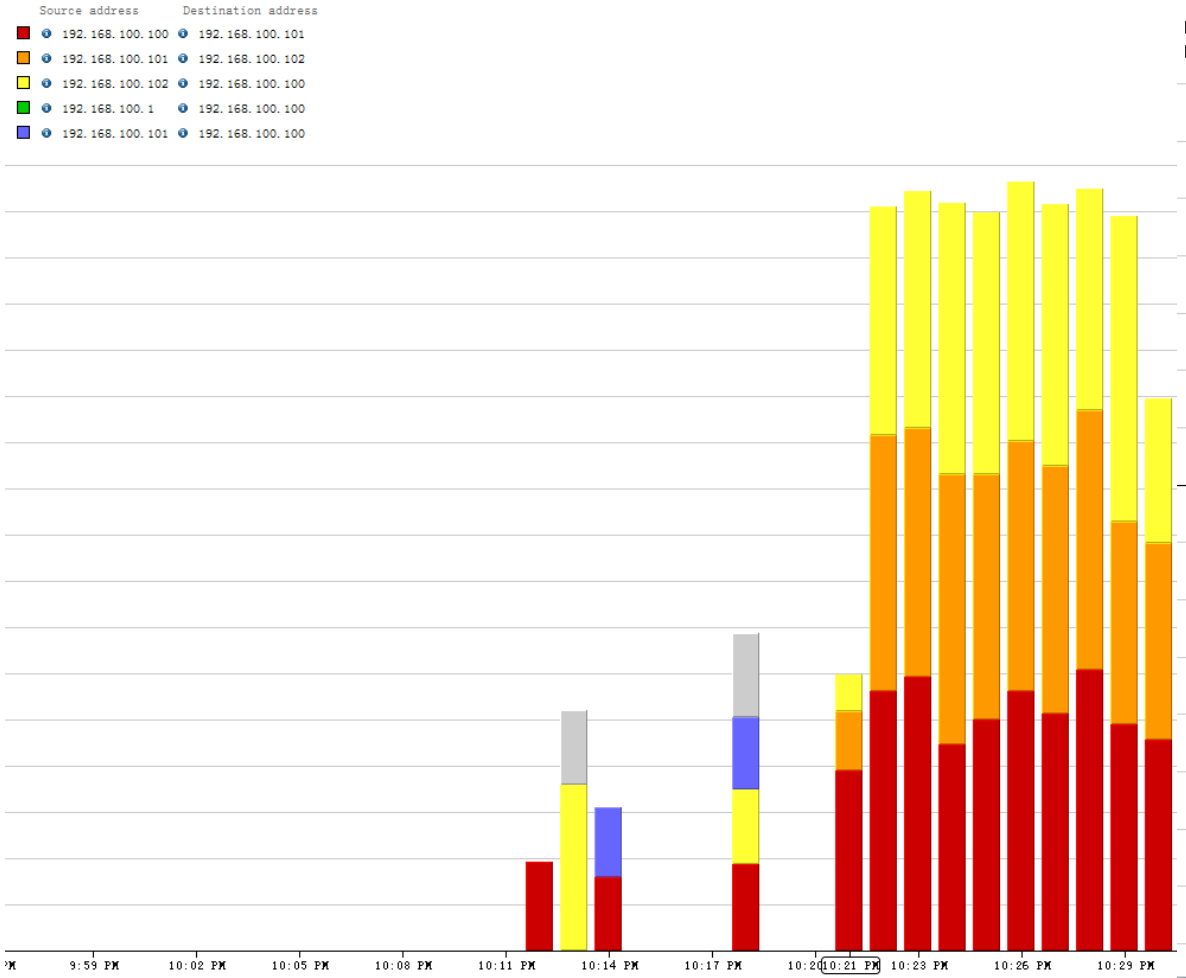


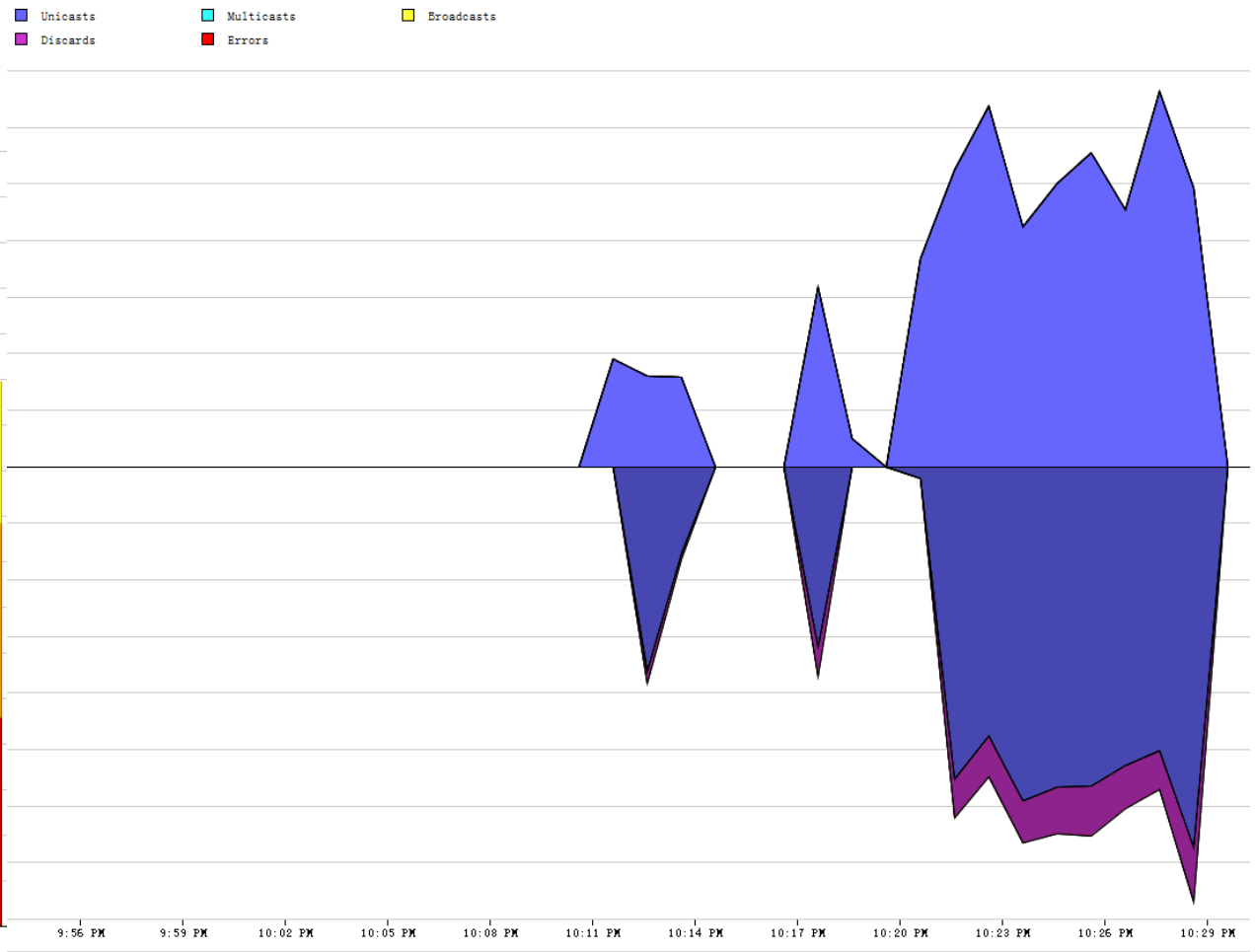
Openvswitch: sFlow, NetFlow/IPFIX

sFlow	NetFlow/IPFIX
InMon sFlowTrend	SolarWinds Real-Time NetFlow Analyzer
流量统计包括L2	仅仅包含L3
No Cache, real-time	With flow cache
monitor all types of traffic: ARP, IPv6, DHCP/BOOTP, STP, LLDP	IPv4 traffic
服务器负责解析包	Switch负责解析包
ovs-vsctl -- --id=@sflow create sflow agent=eth0 target=\"16.158.49.11:6343\" header=128 sampling=512 polling=10 -- set bridge ubuntu_br sflow=@sflow ovs-vsctl list sflow ovs-vsctl -- clear Bridge ubuntu_br sflow	ovs-vsctl -- --id=@nf create NetFlow targets=\"16.158.49.11:2055\" active-timeout=60 -- set Bridge ubuntu_br netflow=@nf ovs-vsctl list NetFlow ovs-vsctl -- clear Bridge ubuntu_br NetFlow



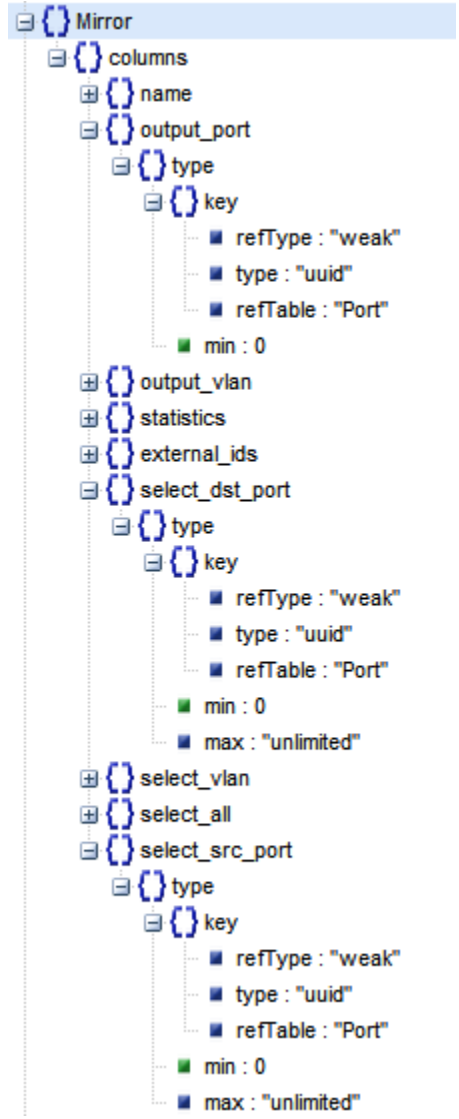
实验六: 使用sFlow和NetFlow





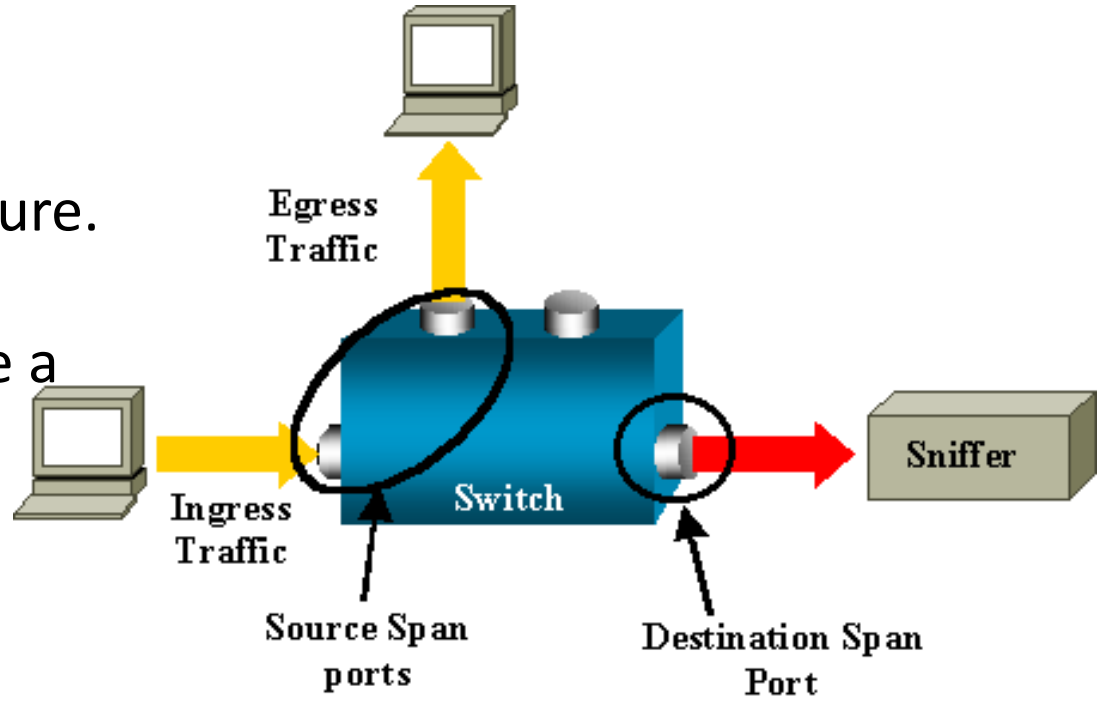
Openvswitch: Mirror

- Mirror就是配置一个bridge，将某些包发给指定的mirrored ports
- 对于包的选择：
  - select_all，所有的包
  - select_dst_port
  - select_src_port
  - select_vlan
- 对于指定的目的：
  - output_port (SPAN Switched Port ANalyzer)
  - output_vlan (RSPAN Remote Switched Port ANalyzer)



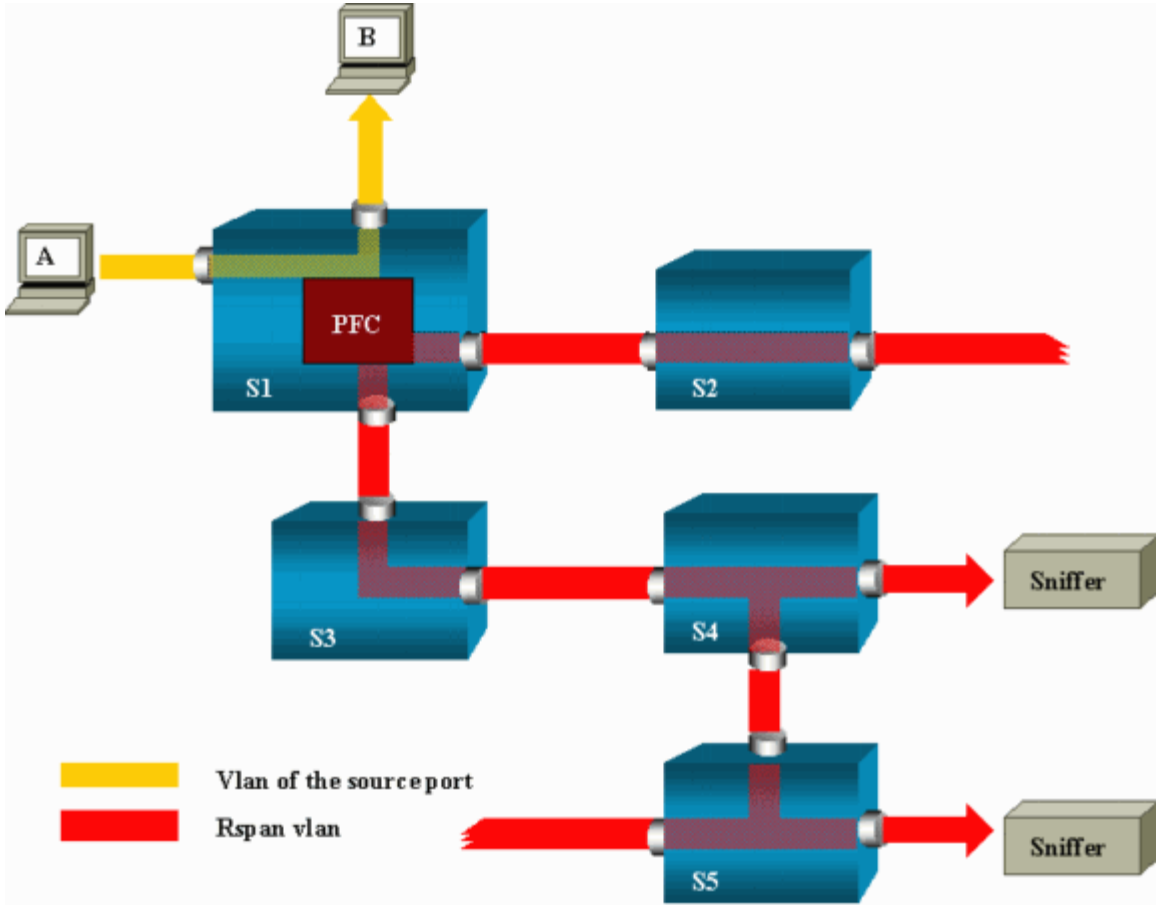
Openvswitch: Mirror

- **SPAN**
  - **Source (SPAN) port** -A port that is monitored with use of the SPAN feature.
  - **Destination (SPAN) port** -A port that monitors source ports, usually where a network analyzer is connected.

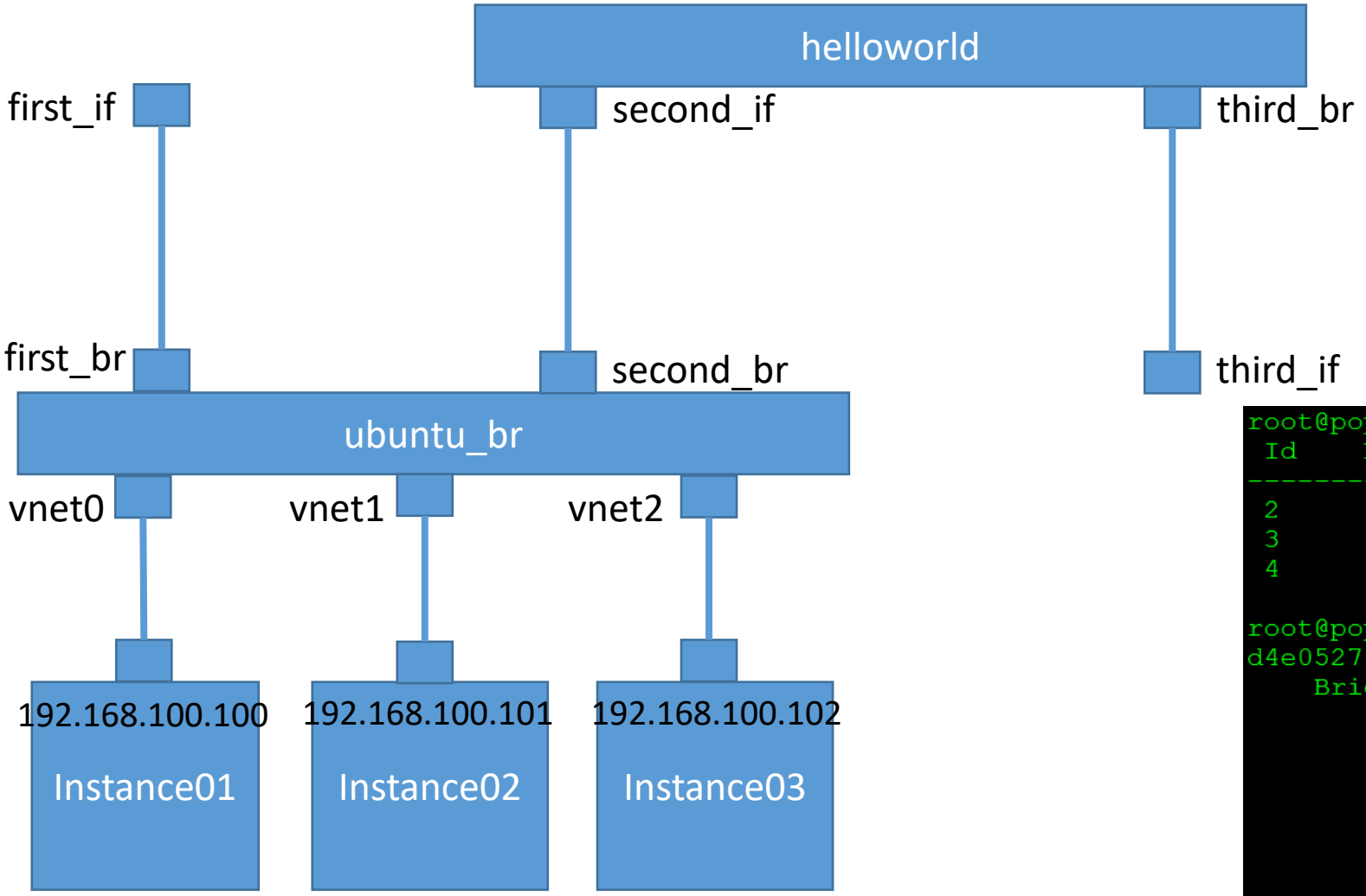


Openvswitch: Mirror

- RSPAN
  - 被监控的流量不是发送到一个指定的端口，而是Flood给指定的VLAN
  - 监听的端口不一定要在本地switch上，可以在指定的VLAN的任意switch上
  - S1 is a source switch
  - S2 and S3 are intermediate switches
  - S4 and S5 are destination switches.
  - learning is disabled to enable flooding



实验七: 测试Mirror的SPAN和RSPAN




```
root@popsuper1982:/home/openstack# virsh list
 Id    Name                           State
----------------------------------------------------
 2     Instance01                     running
 3     Instance02                     running
 4     Instance03                     running

root@popsuper1982:/home/openstack# ovs-vsctl show
d4e05278-c21f-4ed4-be3a-7853ea971931
    Bridge ubuntu_br
        Port "vnet0"
            Interface "vnet0"
        Port "vnet2"
            Interface "vnet2"
        Port ubuntu_br
            Interface ubuntu_br
                type: internal
        Port "vnet1"
            Interface "vnet1"
    ovs_version: "2.0.1"
```

实验七: 测试Mirror的SPAN和RSPAN

- 创建拓扑结构

ovs-vsctl add-br helloworld

ip link add first_br type veth peer name first_if
ip link add second_br type veth peer name second_if
ip link add third_br type veth peer name third_if

ovs-vsctl add-port ubuntu_br first_br
ovs-vsctl add-port ubuntu_br second_br -- set Port second_br tag=110
ovs-vsctl add-port helloworld second_if -- set Port second_if tag=110
ovs-vsctl add-port helloworld third_br -- set Port third_br tag=110

```
root@popsuper1982:/home/openstack# ovs-vsctl show
d4e05278-c21f-4ed4-be3a-7853ea971931
    Bridge ubuntu_br
        Port "vnet0"
            Interface "vnet0"
        Port "vnet2"
            Interface "vnet2"
        Port ubuntu_br
            Interface ubuntu_br
                type: internal
        Port first_br
            Interface first_br
        Port second_br
            tag: 110
            Interface second_br
        Port "vnet1"
            Interface "vnet1"
    Bridge helloworld
        Port third_br
            tag: 110
            Interface third_br
        Port helloworld
            Interface helloworld
                type: internal
        Port second_if
            tag: 110
            Interface second_if
    ovs_version: "2.0.1"
```

实验七: 测试Mirror的SPAN和RSPAN

- 在first_br上面mirror所有进出vnet0的包

ovs-vsctl -- set bridge ubuntu_br mirrors=@m -- --id=@vnet0 get Port vnet0 -- --id=@first_br get Port first_br -- --id=@m create Mirror name=mirrorvnet0 select-dst-port=@vnet0 select-src-port=@vnet0 output-port=@first_br

- 监听first_if，并且从instance01里面ping 192.168.100.102

```
root@popsuper1982:/home/openstack# ovs-vsctl -- set bridge ubuntu_br mirrors=@m -- --id=@vnet0 get Port vnet0 -- --id=@first_br get Port first_br
-- --id=@m create Mirror name=mirrorvnet0 select-dst-port=@vnet0 select-src-port=@vnet0 output-port=@first_br
e63c5a69-7b0b-46fb-a528-c42d406814f6
root@popsuper1982:/home/openstack# tcpdump  -n -i first_if icmp
tcpdump: WARNING: first_if: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on first_if, link-type EN10MB (Ethernet), capture size 65535 bytes
04:35:55.920489 IP 192.168.100.100 > 192.168.100.102: ICMP echo request, id 1
04:35:55.920829 IP 192.168.100.102 > 192.168.100.100: ICMP echo reply, id 175
04:35:56.920403 IP 192.168.100.100 > 192.168.100.102: ICMP echo request, id 1
04:35:56.920695 IP 192.168.100.102 > 192.168.100.100: ICMP echo reply, id 175
04:35:57.919398 IP 192.168.100.100 > 192.168.100.102: ICMP echo request, id 1
04:35:57.919810 IP 192.168.100.102 > 192.168.100.100: ICMP echo reply, id 175
04:35:58.918390 IP 192.168.100.100 > 192.168.100.102: ICMP echo request, id 1
04:35:58.918680 IP 192.168.100.102 > 192.168.100.100: ICMP echo reply, id 175
04:35:59.917500 IP 192.168.100.100 > 192.168.100.102: ICMP echo request, id 1
04:35:59.917856 IP 192.168.100.102 > 192.168.100.100: ICMP echo reply, id 175
04:36:00.917506 IP 192.168.100.100 > 192.168.100.102: ICMP echo request, id 1
04:36:00.917786 IP 192.168.100.102 > 192.168.100.100: ICMP echo reply, id 175
04:36:01.917506 IP 192.168.100.100 > 192.168.100.102: ICMP echo request, id 1
04:36:01.917765 IP 192.168.100.102 > 192.168.100.100: ICMP echo reply, id 175
```

```
QEMU (Instance01)

Ubuntu 14.04 LTS openstackcliu8 tty1

openstackcliu8 login: openstack
Password:
Last login: Sun Aug 31 16:44:14 EDT 2014 from 192.168.100.1 on pts/0
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Mon Sep  1 05:57:08 EDT 2014

  System load:  0.64              Processes:           88
  Usage of /:   45.0% of 2.89GB   Users logged in:     0
  Memory usage: 7%                IP address for eth0: 192.168.100.100
  Swap usage:   0%

  Graph this data and manage this system at:
    https://landscape.canonical.com/

openstack@openstackcliu8:~$ ping 192.168.100.102
PING 192.168.100.102 (192.168.100.102) 56(84) bytes of data.
64 bytes from 192.168.100.102: icmp_seq=1 ttl=64 time=1.74 ms
64 bytes from 192.168.100.102: icmp_seq=2 ttl=64 time=0.493 ms
64 bytes from 192.168.100.102: icmp_seq=3 ttl=64 time=0.643 ms
64 bytes from 192.168.100.102: icmp_seq=4 ttl=64 time=0.471 ms
```

实验七: 测试Mirror的SPAN和RSPAN

- 对进入vnet1的所有进出包，然而ouput到一个vlan 110

ovs-vsctl -- set bridge ubuntu_br mirrors=@m -- --id=@vnet1 get Port vnet1 -- --id=@m create Mirror name=mirrorvnet1 select-dst-port=@vnet1 select-src-port=@vnet1 output-vlan=110

- 在helloworld中也要配置从110来的，都output到vlan 110

ovs-vsctl -- set bridge helloworld mirrors=@m -- --id=@m create Mirror name=mirrorvlan select-vlan=110 output-vlan=110

- Disable mac address learning for vlan 110

ovs-vsctl set bridge ubuntu_br flood-vlans=110
ovs-vsctl set bridge helloworld flood-vlans=110

实验七: 测试Mirror的SPAN和RSPAN

- 监听third_if，并且从instance02里面ping 192.168.100.102

```
root@popsuper1982:/home/openstack# tcpdump  -n -i third_if icmp
tcpdump: WARNING: third_if: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on third_if, link-type EN10MB (Ethernet), capture size 65535 byte
04:44:11.626108 IP 192.168.100.101 > 192.168.100.102: ICMP echo request, id
04:44:11.626497 IP 192.168.100.102 > 192.168.100.101: ICMP echo reply, id 1
04:44:12.625794 IP 192.168.100.101 > 192.168.100.102: ICMP echo request, id
04:44:12.625979 IP 192.168.100.102 > 192.168.100.101: ICMP echo reply, id 1
04:44:13.624798 IP 192.168.100.101 > 192.168.100.102: ICMP echo request, id
04:44:13.625001 IP 192.168.100.102 > 192.168.100.101: ICMP echo reply, id 1
04:44:14.624386 IP 192.168.100.101 > 192.168.100.102: ICMP echo request, id
04:44:14.624624 IP 192.168.100.102 > 192.168.100.101: ICMP echo reply, id 1
04:44:15.624386 IP 192.168.100.101 > 192.168.100.102: ICMP echo request, id
04:44:15.624555 IP 192.168.100.102 > 192.168.100.101: ICMP echo reply, id 1
04:44:16.624394 IP 192.168.100.101 > 192.168.100.102: ICMP echo request, id
04:44:16.624667 IP 192.168.100.102 > 192.168.100.101: ICMP echo reply, id 1
04:44:17.624390 IP 192.168.100.101 > 192.168.100.102: ICMP echo request, id
04:44:17.624544 IP 192.168.100.102 > 192.168.100.101: ICMP echo reply, id 1
04:44:18.624368 IP 192.168.100.101 > 192.168.100.102: ICMP echo request, id
04:44:18.624505 IP 192.168.100.102 > 192.168.100.101: ICMP echo reply, id 1
04:44:19.624436 IP 192.168.100.101 > 192.168.100.102: ICMP echo request, id
04:44:19.624590 IP 192.168.100.102 > 192.168.100.101: ICMP echo reply, id 1
04:44:20.624383 IP 192.168.100.101 > 192.168.100.102: ICMP echo request, id
04:44:20.624628 IP 192.168.100.102 > 192.168.100.101: ICMP echo reply, id 1
04:44:21.624432 IP 192.168.100.101 > 192.168.100.102: ICMP echo request, id
```

QEMU (Instance02)

```
Ubuntu 14.04 LTS openstackcliu8 tty1

openstackcliu8 login: openstack
Password:
Last login: Sun Aug 31 16:44:26 EDT 2014 from 192.168.100.1 on pts/0
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Mon Sep  1 05:57:18 EDT 2014

  System load:  0.88              Processes:           87
  Usage of /:   45.4% of 2.89GB   Users logged in:     0
  Memory usage: 7%                IP address for eth0: 192.168.100.101
  Swap usage:   0%

  Graph this data and manage this system at:
    https://landscape.canonical.com/

openstack@openstackcliu8:~$ ping 192.168.100.102
PING 192.168.100.102 (192.168.100.102) 56(84) bytes of data.
64 bytes from 192.168.100.102: icmp_seq=1 ttl=64 time=2.13 ms
64 bytes from 192.168.100.102: icmp_seq=2 ttl=64 time=0.401 ms
64 bytes from 192.168.100.102: icmp_seq=3 ttl=64 time=0.424 ms
64 bytes from 192.168.100.102: icmp_seq=4 ttl=64 time=0.424 ms
64 bytes from 192.168.100.102: icmp_seq=5 ttl=64 time=0.384 ms
64 bytes from 192.168.100.102: icmp_seq=6 ttl=64 time=0.510 ms
64 bytes from 192.168.100.102: icmp_seq=7 ttl=64 time=0.312 ms
64 bytes from 192.168.100.102: icmp_seq=8 ttl=64 time=0.376 ms
64 bytes from 192.168.100.102: icmp_seq=9 ttl=64 time=0.385 ms
64 bytes from 192.168.100.102: icmp_seq=10 ttl=64 time=0.524 ms
64 bytes from 192.168.100.102: icmp_seq=11 ttl=64 time=0.401 ms
64 bytes from 192.168.100.102: icmp_seq=12 ttl=64 time=0.362 ms
^C
```

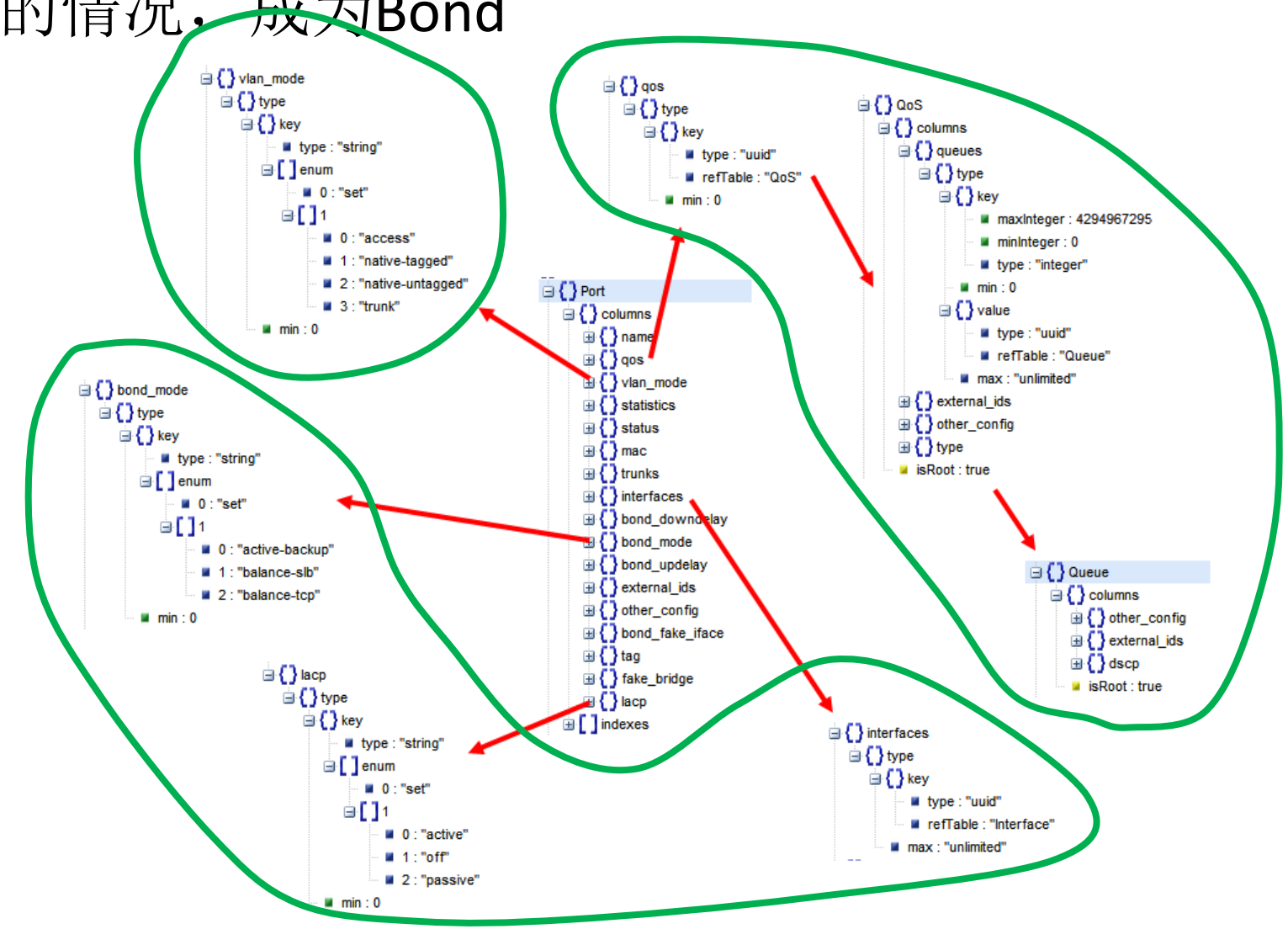
实验七: 测试Mirror的SPAN和RSPAN

- 删除Mirror

查看ubuntu_br
ovs-vsctl list bridge ubuntu_br
清除里面的mirrors
ovs-vsctl clear Bridge ubuntu_br mirrors
清除flood_vlans
ovs-vsctl clear Bridge ubuntu_br flood_vlans
查看所有的Mirror
ovs-vsctl list Mirror
ovs-vsctl clear Bridge helloworld mirrors
ovs-vsctl clear Bridge helloworld flood_vlans

Openvswitch: Port

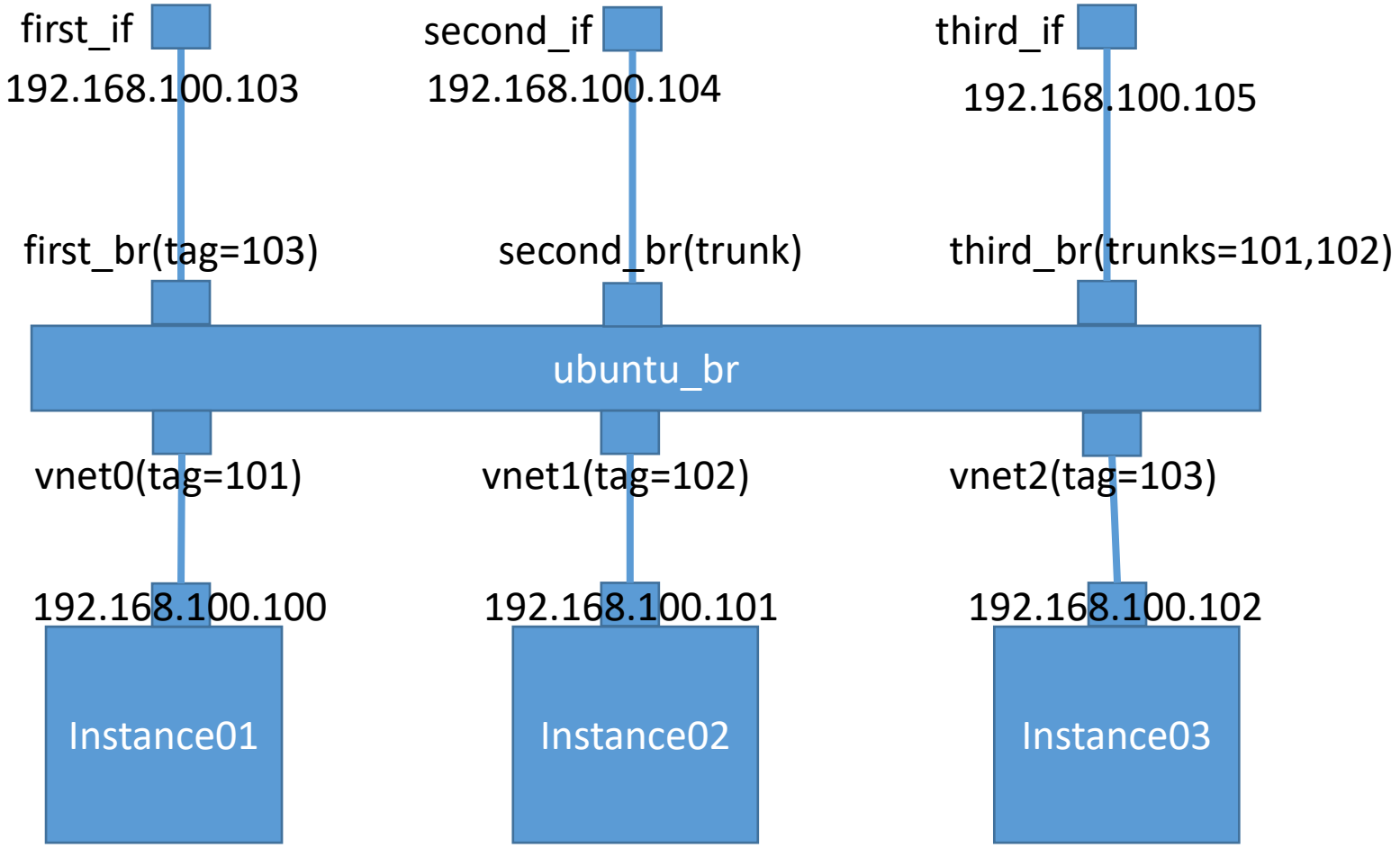
- 一般来说一个Port就是一个Interface，当然也有一个Port对应多个Interface的情况，成为Bond



Openvswitch: Port

- Port的一个重要的方面就是VLAN Configuration，有两种模式：
  - trunk port
    - 这个port不配置tag，配置trunks
    - 如果trunks为空，则所有的VLAN都trunk，也就意味着对于所有的VLAN的包，本身带什么VLAN ID，就是携带者什么VLAN ID，如果没有设置VLAN，就属于VLAN 0，全部允许通过。
    - 如果trunks不为空，则仅仅带着这些VLAN ID的包通过。
  - access port
    - 这个port配置tag，从这个port进来的包会被打上这个tag
    - 如果从其他的trunk port中进来的本身就带有VLAN ID的包，如果VLAN ID等于tag，则会从这个port发出
    - 从其他的access port上来的包，如果tag相同，也会被forward到这个port
    - 从access port发出的包不带VLAN ID
    - 如果一个本身带VLAN ID的包到达access port，即便VLAN ID等于tag，也会被抛弃。

实验八: 测试Port的VLAN功能



实验八: 测试Port的VLAN功能

- 创建拓扑结构

ovs-vsctl add-port ubuntu_br first_br
ovs-vsctl add-port ubuntu_br second_br
ovs-vsctl add-port ubuntu_br third_br
ovs-vsctl set Port vnet0 tag=101
ovs-vsctl set Port vnet1 tag=102
ovs-vsctl set Port vnet2 tag=103
ovs-vsctl set Port first_br tag=103
ovs-vsctl clear Port second_br tag
ovs-vsctl set Port third_br trunks=101,102
需要监听ARP，所以禁止MAC地址学习
ovs-vsctl set bridge ubuntu_br flood-vlans=101,102,103

```
root@popsuper1982:/home/openstack# ovs-vsctl show
d4e05278-c21f-4ed4-be3a-7853ea971931
    Bridge ubuntu_br
        Port "vnet0"
            tag: 101
            Interface "vnet0"
        Port "vnet2"
            tag: 103
            Interface "vnet2"
        Port ubuntu_br
            Interface ubuntu_br
                type: internal
        Port first_br
            tag: 103
            Interface first_br
        Port third_br
            trunks: [101, 102]
            Interface third_br
        Port second_br
            Interface second_br
        Port "vnet1"
            tag: 102
            Interface "vnet1"
    ovs version: "2.0.1"
```

实验八: 测试Port的VLAN功能

- 从192.168.100.102来ping 192.168.100.103， 应该first_if和second_if能够收到包

first_if收到包了，从first_br出来的包头是没有VLAN ID的

```
root@popsuper1982:/home/openstack# tcpdump  -n -e -i first_if arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on first_if, link-type EN10MB (Ethernet), capture size 65535 bytes
06:29:48.415723 52:54:00:9b:d5:77 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 42: Request who-has 192.168.100.103 tell 192.168.100.102, le
ngth 28
06:29:48.415768 92:a7:e0:5e:0f:04 > 52:54:00:9b:d5:77, ethertype ARP (0x0806), length 42: Reply 192.168.100.103 is-at 92:a7:e0:5e:0f:04, length 28
06:29:53.430372 92:a7:e0:5e:0f:04 > 52:54:00:9b:d5:77, ethertype ARP (0x0806), length 42: Request who-has 192.168.100.102 tell 192.168.100.103, le
ngth 28
06:29:53.430744 52:54:00:9b:d5:77 > 92:a7:e0:5e:0f:04, ethertype ARP (0x0806), length 42: Reply 192.168.100.102 is-at 52:54:00:9b:d5:77, length 28
```

second_if也收到包了，由于second_br是trunk port，因而出来的包头是有VLAN ID的，103

```
root@popsuper1982:/home/openstack#  tcpdump  -n -e -i second_if arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on second_if, link-type EN10MB (Ethernet), capture size 65535 bytes
06:29:48.415725 52:54:00:9b:d5:77 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 103, p 0, ethertype ARP, Request who-has 192.168
.100.103 tell 192.168.100.102, length 28
06:29:48.415900 92:a7:e0:5e:0f:04 > 52:54:00:9b:d5:77, ethertype 802.1Q (0x8100), length 46: vlan 103, p 0, ethertype ARP, Reply 192.168.100.103 i
s-at 92:a7:e0:5e:0f:04, length 28
06:29:53.430412 92:a7:e0:5e:0f:04 > 52:54:00:9b:d5:77, ethertype 802.1Q (0x8100), length 46: vlan 103, p 0, ethertype ARP, Request who-has 192.168
.100.102 tell 192.168.100.103, length 28
06:29:53.430745 52:54:00:9b:d5:77 > 92:a7:e0:5e:0f:04, ethertype 802.1Q (0x8100), length 46: vlan 103, p 0, ethertype ARP, Reply 192.168.100.102 i
s-at 52:54:00:9b:d5:77, length 28
```

third_if收不到包

实验八: 测试Port的VLAN功能

- 从192.168.100.100在ping 192.168.100.105, 则second_if和third_if可以收到包(当然ping不通，因为third_if不属于某个VLAN)

first_if收不到包

second_if能够收到包，而且包头里面是VLAN ID = 101

```
root@popsuper1982:/home/openstack# tcpdump  -n -e -i second_if arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on second_if, link-type EN10MB (Ethernet), capture size 65535 bytes
09:07:14.367260 52:54:00:9b:d5:11 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 101, p 0, ethertype ARP, Request who-has 192.168
.100.105 tell 192.168.100.100, length 28
09:07:15.365690 52:54:00:9b:d5:11 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 101, p 0, ethertype ARP, Request who-has 192.168
.100.105 tell 192.168.100.100, length 28
09:07:16.365678 52:54:00:9b:d5:11 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 101, p 0, ethertype ARP, Request who-has 192.168
.100.105 tell 192.168.100.100, length 28
09:07:17.382867 52:54:00:9b:d5:11 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 101, p 0, ethertype ARP, Request who-has 192.168
.100.105 tell 192.168.100.100, length 28
09:07:18.381711 52:54:00:9b:d5:11 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 101, p 0, ethertype ARP, Request who-has 192.168
.100.105 tell 192.168.100.100, length 28
```

third_if也能收到包，而且包头里面是VLAN ID =101

```
root@popsuper1982:/home/openstack# tcpdump  -n -e -i third_if arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on third_if, link-type EN10MB (Ethernet), capture size 65535 bytes
09:07:14.367263 52:54:00:9b:d5:11 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 101, p 0, ethertype ARP, Request who-has 192.168
.100.105 tell 192.168.100.100, length 28
09:07:15.365692 52:54:00:9b:d5:11 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 101, p 0, ethertype ARP, Request who-has 192.168
.100.105 tell 192.168.100.100, length 28
09:07:16.365681 52:54:00:9b:d5:11 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 101, p 0, ethertype ARP, Request who-has 192.168
.100.105 tell 192.168.100.100, length 28
09:07:17.382870 52:54:00:9b:d5:11 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 101, p 0, ethertype ARP, Request who-has 192.168
.100.105 tell 192.168.100.100, length 28
09:07:18.381713 52:54:00:9b:d5:11 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 101, p 0, ethertype ARP, Request who-has 192.168
.100.105 tell 192.168.100.100, length 28
```

实验八: 测试Port的VLAN功能

- 从192.168.100.101来ping 192.168.100.104, 则second_if和third_if可以收到包

first_if收不到包

second_br能够收到包，而且包头里面是VLAN ID = 102

```
root@popsuper1982:/home/openstack# tcpdump  -n -e -i second_if arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on second_if, link-type EN10MB (Ethernet), capture size 65535 bytes
09:15:52.456171 52:54:00:9b:d5:33 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 102, p 0, ethertype ARP, Request who-has 192.168
.100.104 tell 192.168.100.101, length 28
09:15:53.453052 52:54:00:9b:d5:33 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 102, p 0, ethertype ARP, Request who-has 192.168
.100.104 tell 192.168.100.101, length 28
09:15:54.453024 52:54:00:9b:d5:33 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 102, p 0, ethertype ARP, Request who-has 192.168
.100.104 tell 192.168.100.101, length 28
09:15:55.471005 52:54:00:9b:d5:33 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 102, p 0, ethertype ARP, Request who-has 192.168
.100.104 tell 192.168.100.101, length 28
09:15:56.469049 52:54:00:9b:d5:33 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 102, p 0, ethertype ARP, Request who-has 192.168
.100.104 tell 192.168.100.101, length 28
```

third_if也能收到包，而且包头里面是VLAN ID =102

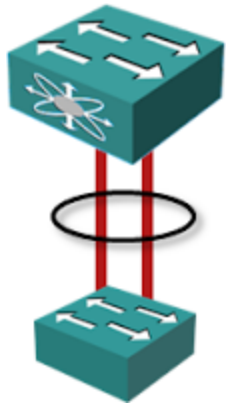
```
root@popsuper1982:/home/openstack# tcpdump  -n -e -i third_if arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on third_if, link-type EN10MB (Ethernet), capture size 65535 bytes
09:15:52.456173 52:54:00:9b:d5:33 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 102, p 0, ethertype ARP, Request who-has 192.168
.100.104 tell 192.168.100.101, length 28
09:15:53.453054 52:54:00:9b:d5:33 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 102, p 0, ethertype ARP, Request who-has 192.168
.100.104 tell 192.168.100.101, length 28
09:15:54.453026 52:54:00:9b:d5:33 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 102, p 0, ethertype ARP, Request who-has 192.168
.100.104 tell 192.168.100.101, length 28
09:15:55.471007 52:54:00:9b:d5:33 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 102, p 0, ethertype ARP, Request who-has 192.168
.100.104 tell 192.168.100.101, length 28
09:15:56.469051 52:54:00:9b:d5:33 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 102, p 0, ethertype ARP, Request who-has 192.168
.100.104 tell 192.168.100.101, length 28
```

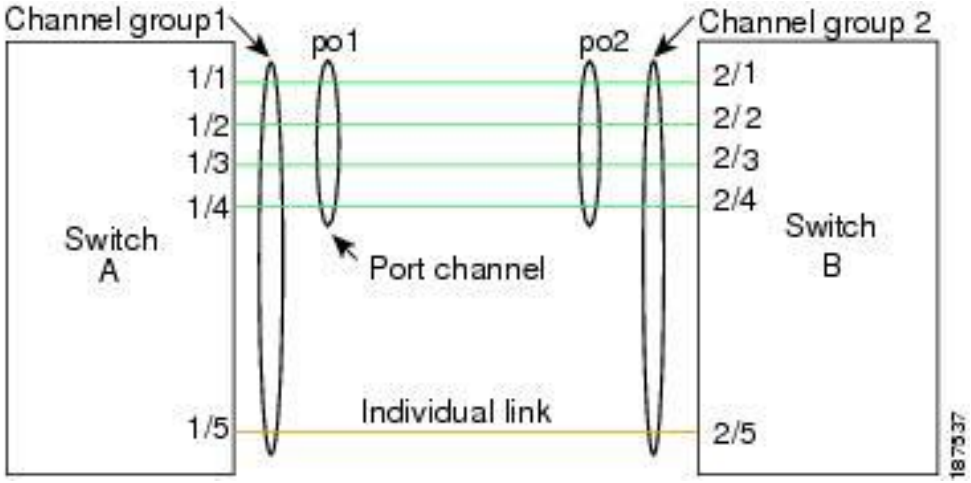
实验八: 测试Port的VLAN功能

- 清理环境
- ovs-vsctl clear Bridge ubuntu_br flood_vlans
- ovs-vsctl list Port
- ovs-vsctl clear Port vnet1 tag
- ovs-vsctl clear Port vnet0 tag
- ovs-vsctl clear Port first_br tag
- ovs-vsctl clear Port third_br trunks

Openvwitch: Bond

- 有关Interface，就不得不提Bond
- Bond将设备用多个连接在一起，形成一个虚拟的连接，从而实现高可用性以及高吞吐量
- 很多别名：LACP Trunk, Bond, Etherchannel
- LACP (Link Aggregation Control Protocol)





Openvwitch: Bond

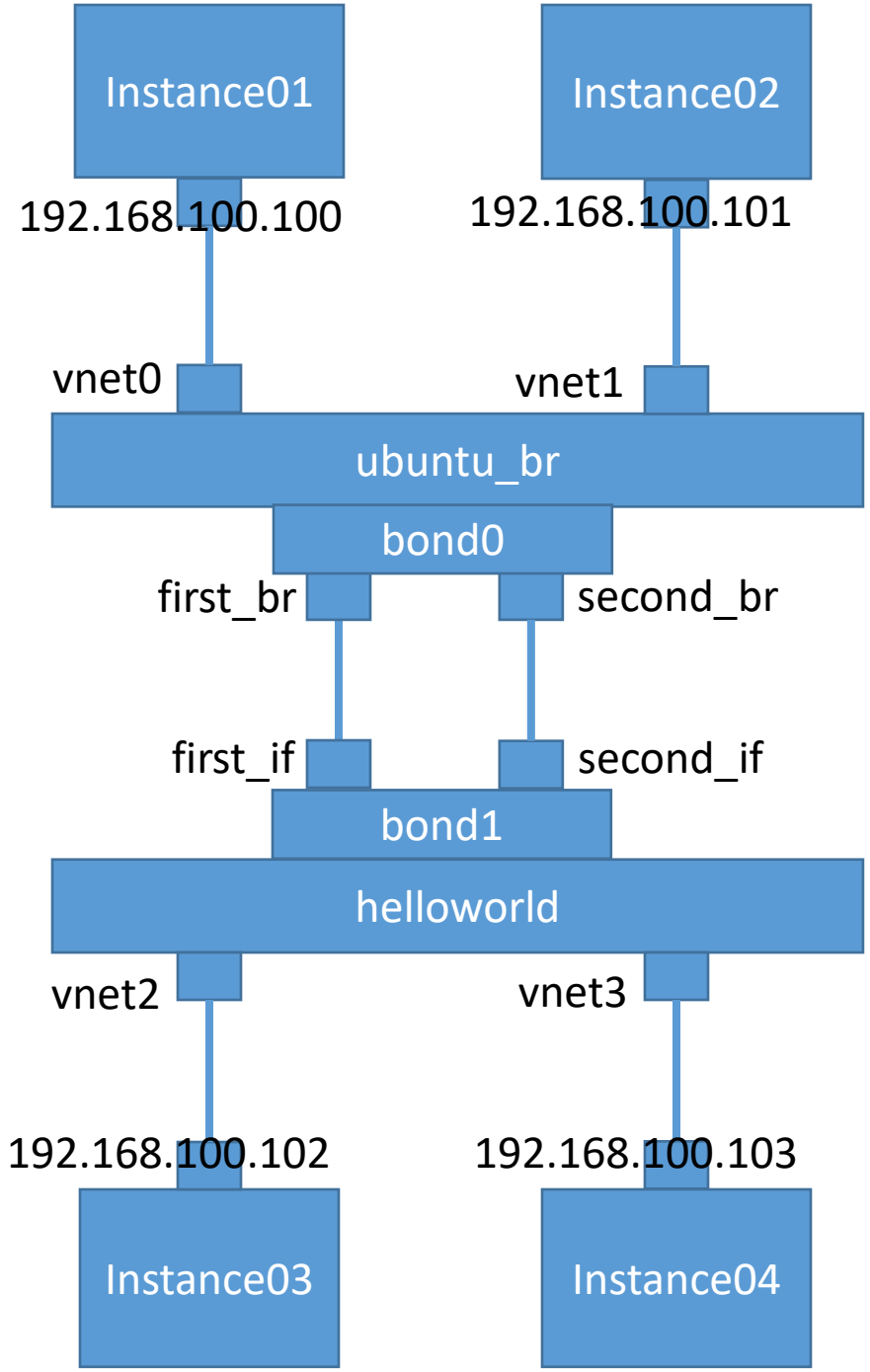
- bond_mode
  - active-backup: 一个连接是active，其他的backup， 当active失效的时候，backup顶上
  - balance-slb: 流量安装源MAC和output VLAN进行负载均衡
  - balance-tcp:必须在支持LACP协议的情况下才可以， 可根据L2, L3, L4进行负载均衡

实验九: 测试Bond功能

ovs-vsctl add-bond ubuntu_br bond0 first_br second_br
ovs-vsctl add-bond helloworld bond1 first_if second_if

ovs-vsctl set Port bond0 lacp=active
ovs-vsctl set Port bond1 lacp=active

```
root@popsuper1982:/home/openstack# ovs-vsctl show
d4e05278-c21f-4ed4-be3a-7853ea971931
    Bridge helloworld
        Port "bond1"
            Interface second_if
            Interface first_if
        Port "vnet2"
            Interface "vnet2"
        Port helloworld
            Interface helloworld
                type: internal
        Port "vnet3"
            Interface "vnet3"
    Bridge ubuntu_br
        Port "vnet0"
            Interface "vnet0"
        Port ubuntu_br
            Interface ubuntu_br
                type: internal
        Port "bond0"
            Interface second_br
            Interface first_br
        Port "vnet1"
            Interface "vnet1"
    ovs_version: "2.0.1"
```



实验九: 测试Bond功能

- 查看Bond

```
root@popsuper1982:/home/openstack# ovs-appctl bond/show
---- bond0 ----
bond_mode: active-backup
bond-hash-basis: 0
updelay: 0 ms
downdelay: 0 ms
lacp_status: negotiated

slave first_br: enabled
        active slave
        may_enable: true

slave second_br: enabled
        may_enable: true

---- bond1 ----
bond_mode: active-backup
bond-hash-basis: 0
updelay: 0 ms
downdelay: 0 ms
lacp_status: negotiated

slave first_if: enabled
        active slave
        may_enable: true

slave second_if: enabled
        may_enable: true
```

- 查看LACP

```
root@popsuper1982:/home/openstack# ovs-appctl lacp/show
---- bond0 ----
        status: active negotiated
        sys_id: 2a:96:0e:c7:85:49
        sys_priority: 65534
        aggregation key: 7
        lacp_time: slow

slave: first_br: current attached
        port_id: 7
        port_priority: 65535
        may_enable: true

        actor sys_id: 2a:96:0e:c7:85:49
        actor sys_priority: 65534
        actor port_id: 7
        actor port_priority: 65535
        actor key: 7
        actor state: activity aggregation synchronized collecting distributing

        partner sys_id: 72:d2:d3:59:8c:41
        partner sys_priority: 65534
        partner port_id: 3
        partner port_priority: 65535
        partner key: 3
```

实验九: 测试Bond功能

- 默认情况下bond_mode是active-backup模式，一开始active的是first_br和first_if
- 从192.168.100.100 ping 192.168.100.102，以及192.168.100.101 ping 192.168.100.103，都是从first_if通过

```
root@popsuper1982:/home/openstack# tcpdump  -n -e -i first_if
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on first_if, link-type EN10MB (Ethernet), capture size 65535 bytes
15:49:40.709377 ee:bd:88:00:00:05 > 01:80:c2:00:00:02, ethertype Slow Protocols (0x8809), length 124: LACPv1, length 110
15:49:41.690989 92:a7:e0:5e:0f:04 > 01:80:c2:00:00:02, ethertype Slow Protocols (0x8809), length 124: LACPv1, length 110
15:49:58.047194 52:54:00:9b:d5:11 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 42: Request who-has 192.168.100.102 tell 192.16
ngth 28
15:49:58.047818 52:54:00:9b:d5:77 > 52:54:00:9b:d5:11, ethertype ARP (0x0806), length 42: Reply 192.168.100.102 is-at 52:54:00:9b:d5:
15:49:58.048209 52:54:00:9b:d5:11 > 52:54:00:9b:d5:77, ethertype IPv4 (0x0800), length 98: 192.168.100.100 > 192.168.100.102: ICMP ec
d 1532, seq 1, length 64
15:49:58.048660 52:54:00:9b:d5:77 > 52:54:00:9b:d5:11, ethertype IPv4 (0x0800), length 98: 192.168.100.102 > 192.168.100.100: ICMP ec
1532, seq 1, length 64
15:49:59.047719 52:54:00:9b:d5:11 > 52:54:00:9b:d5:77, ethertype IPv4 (0x0800), length 98: 192.168.100.100 > 192.168.100.102: ICMP ec
d 1532, seq 2, length 64
15:49:59.047896 52:54:00:9b:d5:77 > 52:54:00:9b:d5:11, ethertype IPv4 (0x0800), length 98: 192.168.100.102 > 192.168.100.100: ICMP ec
1532, seq 2, length 64
15:50:00.049687 52:54:00:9b:d5:11 > 52:54:00:9b:d5:77, ethertype IPv4 (0x0800), length 98: 192.168.100.100 > 192.168.100.102: ICMP ec
d 1532, seq 3, length 64
15:50:00.049911 52:54:00:9b:d5:77 > 52:54:00:9b:d5:11, ethertype IPv4 (0x0800), length 98: 192.168.100.102 > 192.168.100.100: ICMP ec
1532, seq 3, length 64
15:50:01.051587 52:54:00:9b:d5:11 > 52:54:00:9b:d5:77, ethertype IPv4 (0x0800), length 98: 192.168.100.100 > 192.168.100.102: ICMP ec
d 1532, seq 4, length 64
15:50:01.051753 52:54:00:9b:d5:77 > 52:54:00:9b:d5:11, ethertype IPv4 (0x0800), length 98: 192.168.100.102 > 192.168.100.100: ICMP ec
1532, seq 4, length 64
15:50:03.062510 52:54:00:9b:d5:77 > 52:54:00:9b:d5:11, ethertype ARP (0x0806), length 42: Request who-has 192.168.100.100 tell 192.16
ngth 28
15:50:03.062970 52:54:00:9b:d5:11 > 52:54:00:9b:d5:77, ethertype ARP (0x0806), length 42: Reply 192.168.100.100 is-at 52:54:00:9b:d5:
15:50:10.710420 ee:bd:88:00:00:05 > 01:80:c2:00:00:02, ethertype Slow Protocols (0x8809), length 124: LACPv1, length 110
15:50:11.691025 92:a7:e0:5e:0f:04 > 01:80:c2:00:00:02, ethertype Slow Protocols (0x8809), length 124: LACPv1, length 110
15:50:40.711437 ee:bd:88:00:00:05 > 01:80:c2:00:00:02, ethertype Slow Protocols (0x8809), length 124: LACPv1, length 110
15:50:41.691029 92:a7:e0:5e:0f:04 > 01:80:c2:00:00:02, ethertype Slow Protocols (0x8809), length 124: LACPv1, length 110
15:50:52.701660 52:54:00:9b:d5:33 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 42: Request who-has 192.168.100.103 tell 192.16
ngth 28
15:50:52.702168 52:54:00:9b:d5:bf > 52:54:00:9b:d5:33, ethertype ARP (0x0806), length 42: Reply 192.168.100.103 is-at 52:54:00:9b:d5:
15:50:52.702613 52:54:00:9b:d5:33 > 52:54:00:9b:d5:bf, ethertype IPv4 (0x0800), length 98: 192.168.100.101 > 192.168.100.103: ICMP ec
d 1530, seq 1, length 64
15:50:52.703007 52:54:00:9b:d5:bf > 52:54:00:9b:d5:33, ethertype IPv4 (0x0800), length 98: 192.168.100.103 > 192.168.100.101: ICMP ec
1530, seq 1, length 64
15:50:53.702010 52:54:00:9b:d5:33 > 52:54:00:9b:d5:bf, ethertype IPv4 (0x0800), length 98: 192.168.100.101 > 192.168.100.103: ICMP ec
d 1530, seq 2, length 64
```

实验九: 测试Bond功能

- 如果把first_if设成down，则包的走向会变
- ip link set first_if down
- 发现second_if开始有流量，京first_if变成down, 192.168.100.100和192.168.100.101似乎没有收到影响
- second_br和second_if变成active

```
root@popsuper1982:/home/openstack# ovs-appctl bond/show
---- bond0 ----
bond_mode: active-backup
bond-hash-basis: 0
updelay: 0 ms
downdelay: 0 ms
lacp_status: negotiated

slave first_br: disabled
        may_enable: false

slave second_br: enabled
        active slave
        may_enable: true

---- bond1 ----
bond_mode: active-backup
bond-hash-basis: 0
updelay: 0 ms
downdelay: 0 ms
lacp_status: negotiated

slave first_if: disabled
        may_enable: false

slave second_if: enabled
        active slave
        may_enable: true
```

实验九: 测试Bond功能

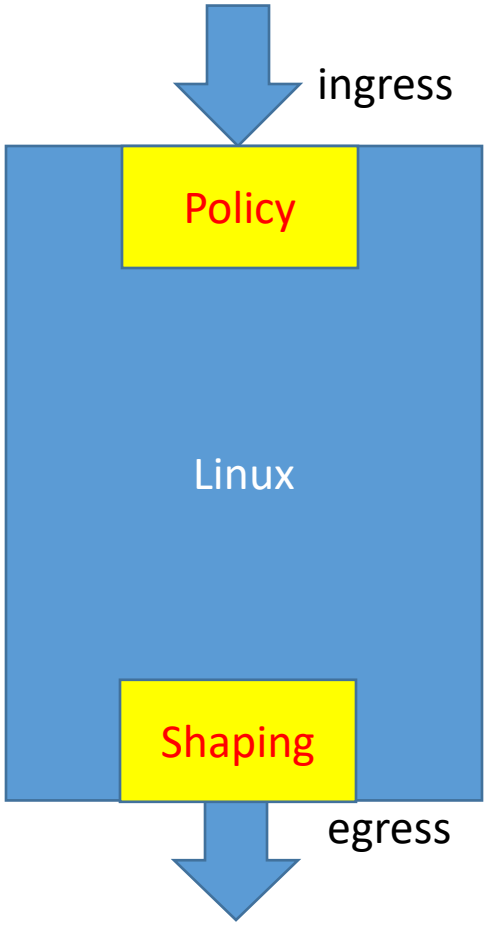
- 重启first_if，但是second_br和second_if仍然是active
- ip link set first_if up

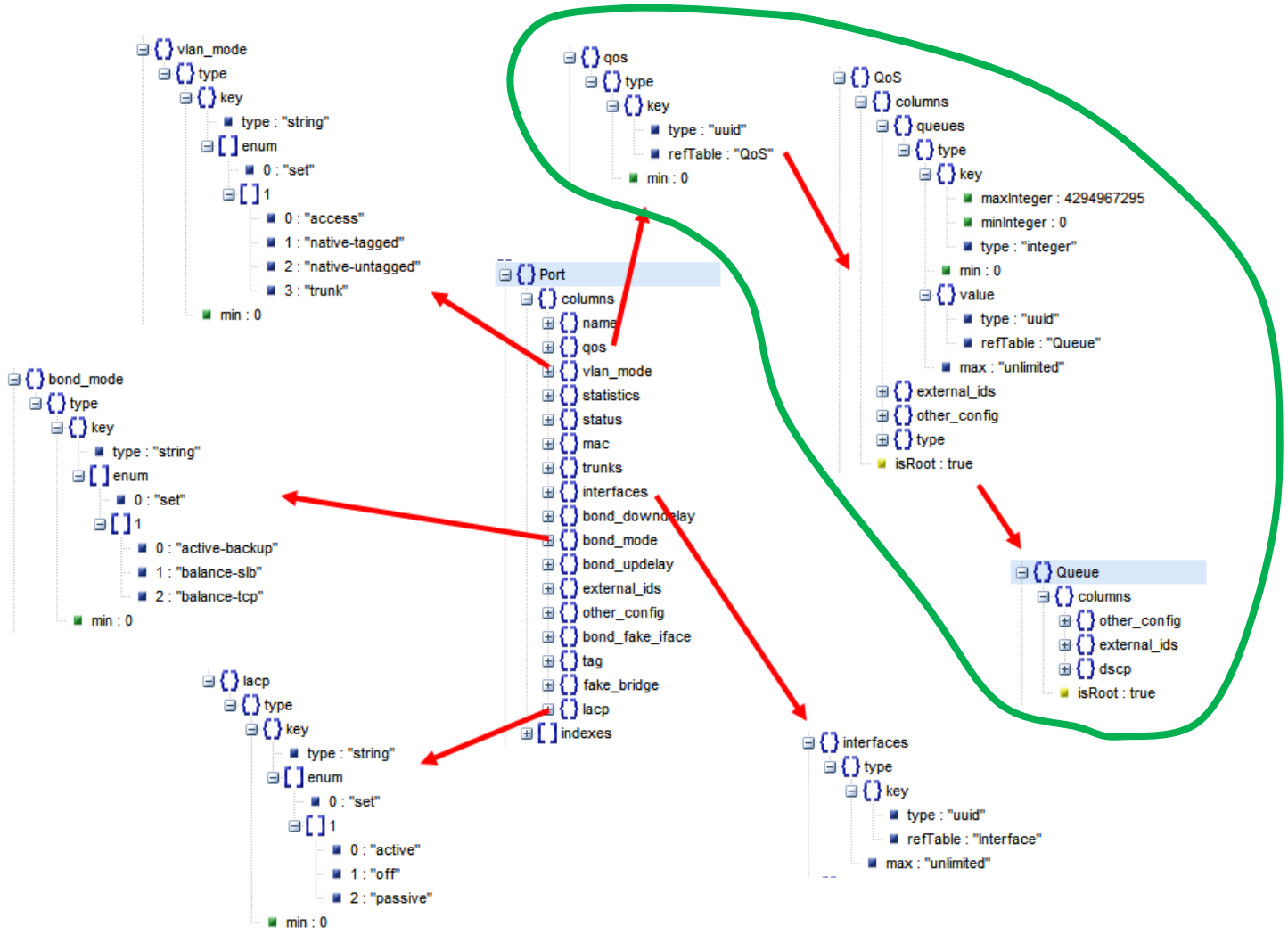
```
root@popsuper1982:/home/openstack# ovs-appctl bond/show
---- bond0 ----
bond_mode: active-backup
bond-hash-basis: 0
updelay: 0 ms
downdelay: 0 ms
lacp_status: negotiated

slave first_br: enabled
        may_enable: true

slave second_br: enabled
        active slave
        may_enable: true

---- bond1 ----
bond_mode: active-backup
bond-hash-basis: 0
updelay: 0 ms
downdelay: 0 ms
lacp_status: negotiated

slave first_if: enabled
        may_enable: true

slave second_if: enabled
        active slave
        may_enable: true
```

实验九: 测试Bond功能

- 把bond_mode设为balance-slb
  - ovs-vsctl set Port bond0 bond_mode=balance-slb
  - ovs-vsctl set Port bond1 bond_mode=balance-slb
  - 同时192.168.100.100 ping 192.168.100.102,192.168.100.101 ping 192.168.100.103, 已经分流了
- 把bond_mode设为balance-tcp
  - ovs-vsctl set Port bond0 bond_mode=balance-tcp
  - ovs-vsctl set Port bond1 bond_mode=balance-tcp
  - 同时在192.168.100.100上： netperf -H 192.168.100.102 -t UDP_STREAM -- -m 1024
  - 在192.168.100.101上： netperf -H 192.168.100.103 -t UDP_STREAM -- -m 1024

Openvswitch: QoS

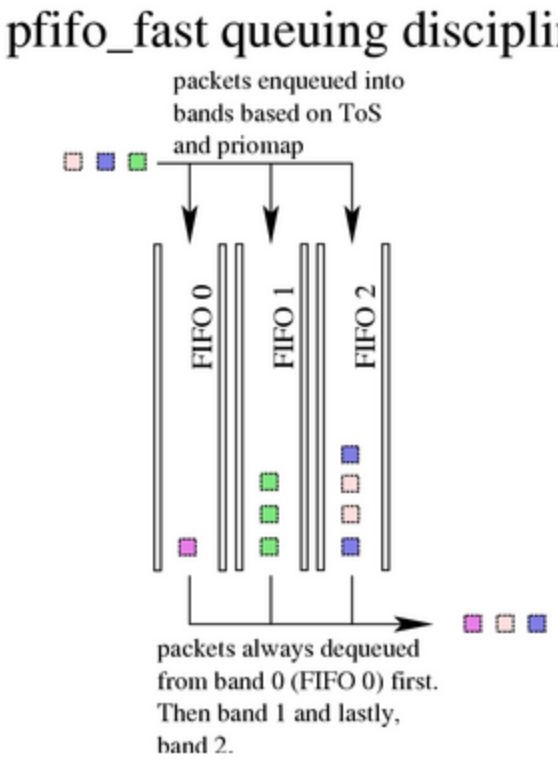




Openvswitch: QoS

- **Classless Queuing Disciplines**
- 默认为pfifo_fast

```
root@popsuper1982:/home/openstack# ip link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 68:b5:99:ef:5c:ac brd ff:ff:ff:ff:ff:ff
```




```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|                 |                       |     |
|   PRECEDENCE    |          TOS          | MBZ |
|                 |                       |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

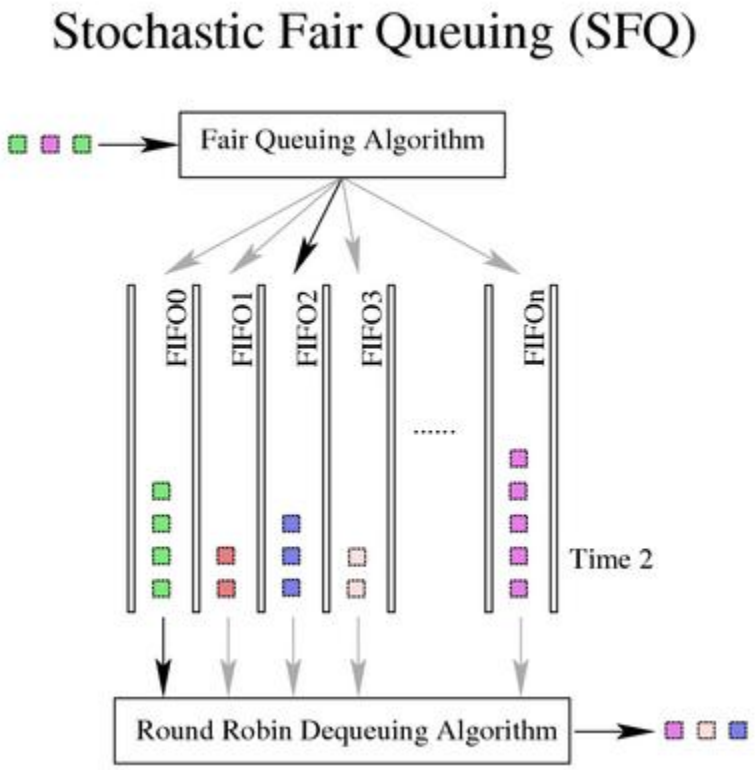
Binary	Decimcal	Meaning
1000	8	Minimize delay (md)
0100	4	Maximize throughput (mt)
0010	2	Maximize reliability (mr)
0001	1	Minimize monetary cost (mmc)
0000	0	Normal Service

TOS	Bits	Means	Linux Priority	Band
0x0	0	Normal Service	0 Best Effort	1
0x2	1	Minimize Monetary Cost	1 Filler	2
0x4	2	Maximize Reliability	0 Best Effort	1
0x6	3	mmc+mr	0 Best Effort	1
0x8	4	Maximize Throughput	2 Bulk	2
0xa	5	mmc+mt	2 Bulk	2
0xc	6	mr+mt	2 Bulk	2
0xe	7	mmc+mr+mt	2 Bulk	2
0x10	8	Minimize Delay	6 Interactive	0
0x12	9	mmc+md	6 Interactive	0
0x14	10	mr+md	6 Interactive	0
0x16	11	mmc+mr+md	6 Interactive	0
0x18	12	mt+md	4 Int. Bulk	1
0x1a	13	mmc+mt+md	4 Int. Bulk	1
0x1c	14	mr+mt+md	4 Int. Bulk	1
0x1e	15	mmc+mr+mt+md	4 Int. Bulk	1

```
root@popsuper1982:/home/openstack# tc qdisc show dev eth0
qdisc pfifo_fast 0: root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
```

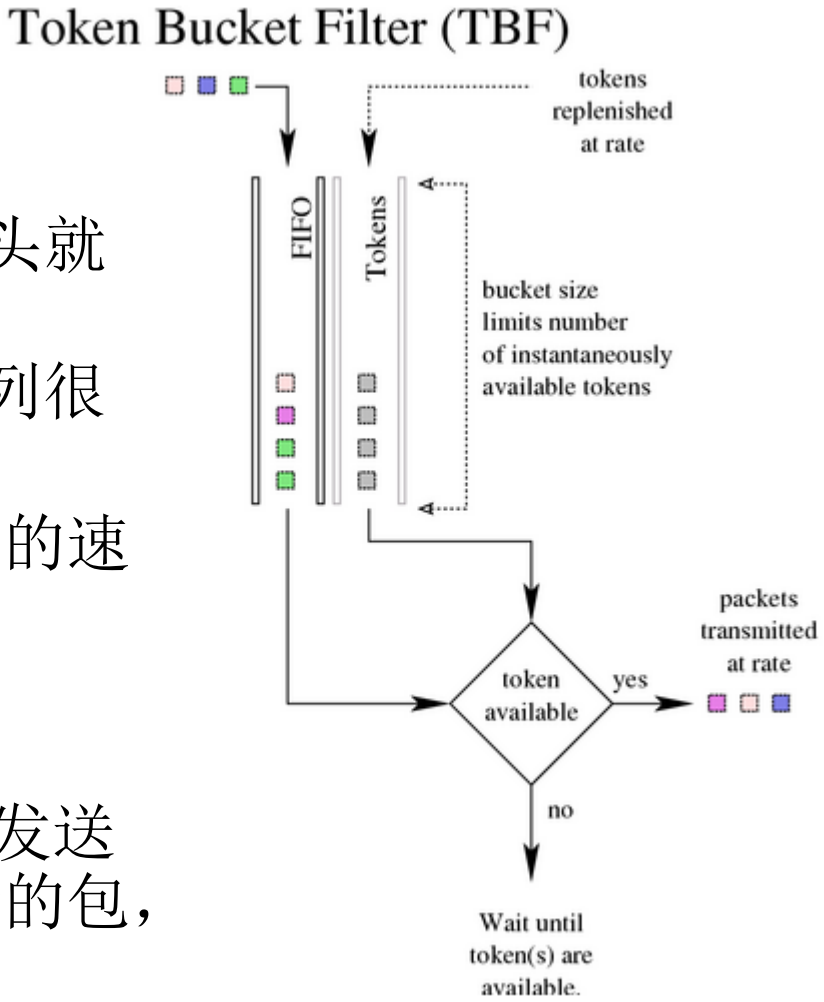
Openvswitch: QoS

- SFQ, Stochastic Fair Queuing
  - 有很多的FIFO的队列，TCP Session或者UDP stream会被分配到某个队列。包会RoundRobin的从各个队列中取出发送。
  - 这样不会一个Session占据所有的流量。
  - 但不是每一个Session都有一个队列，而是有一个Hash算法，将大量的Session分配到有限的队列中。
  - 这样两个Session会共享一个队列，也有可能互相影响。
  - Hash函数会经常改变，从而session不会总是相互影响。



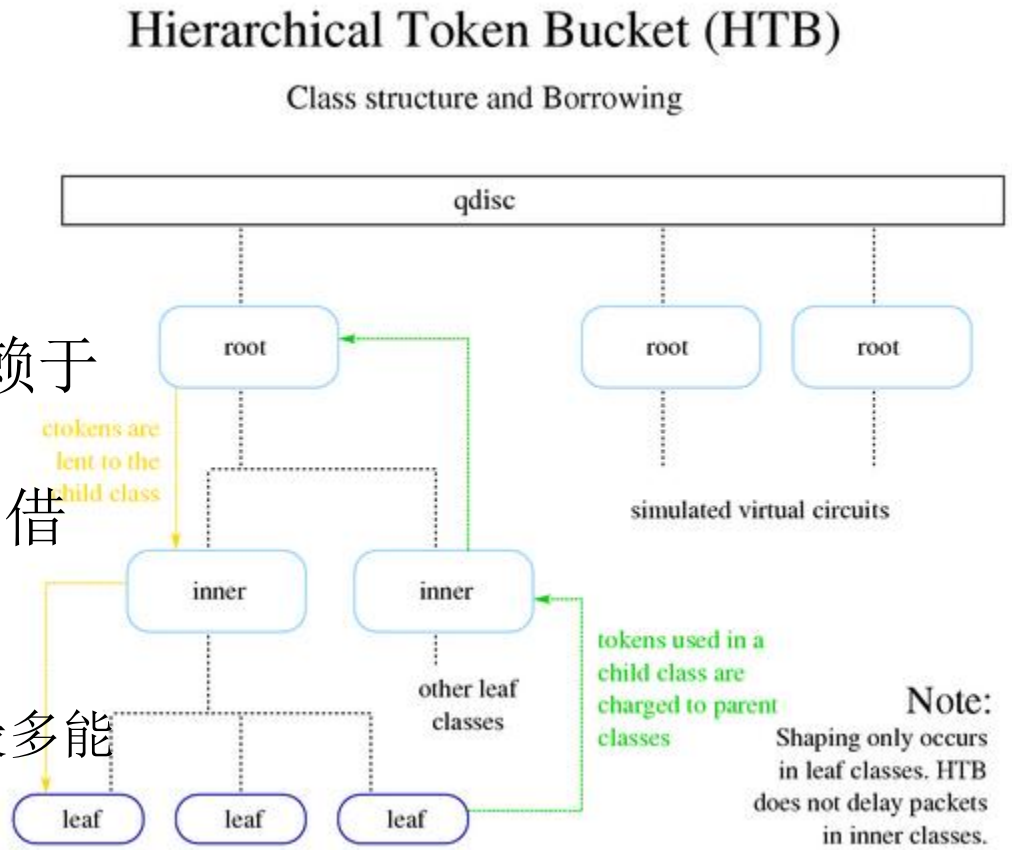
Openvswitch: QoS

- TBF, Token Bucket Filter
  - 两个概念Tokens and buckets
  - 所有的包排成队列进行发送，但不是到了队头就能发送，而是需要拿到Token才能发送
  - Token根据设定的速度rate生成，所以即便队列很长，也是按照rate进行发送的
  - 当没有包在队列中的时候，Token还是以既定的速度生成，但是不是无限累积的，而是放满了buckets为止，篮子的大小常用burst/buffer/maxburst来设定
  - Buckets会避免下面的情况：当长时间没有包发送的时候，积累了大量的Token，突然来了大量的包，每个都能得到Token，造成瞬间流量大增



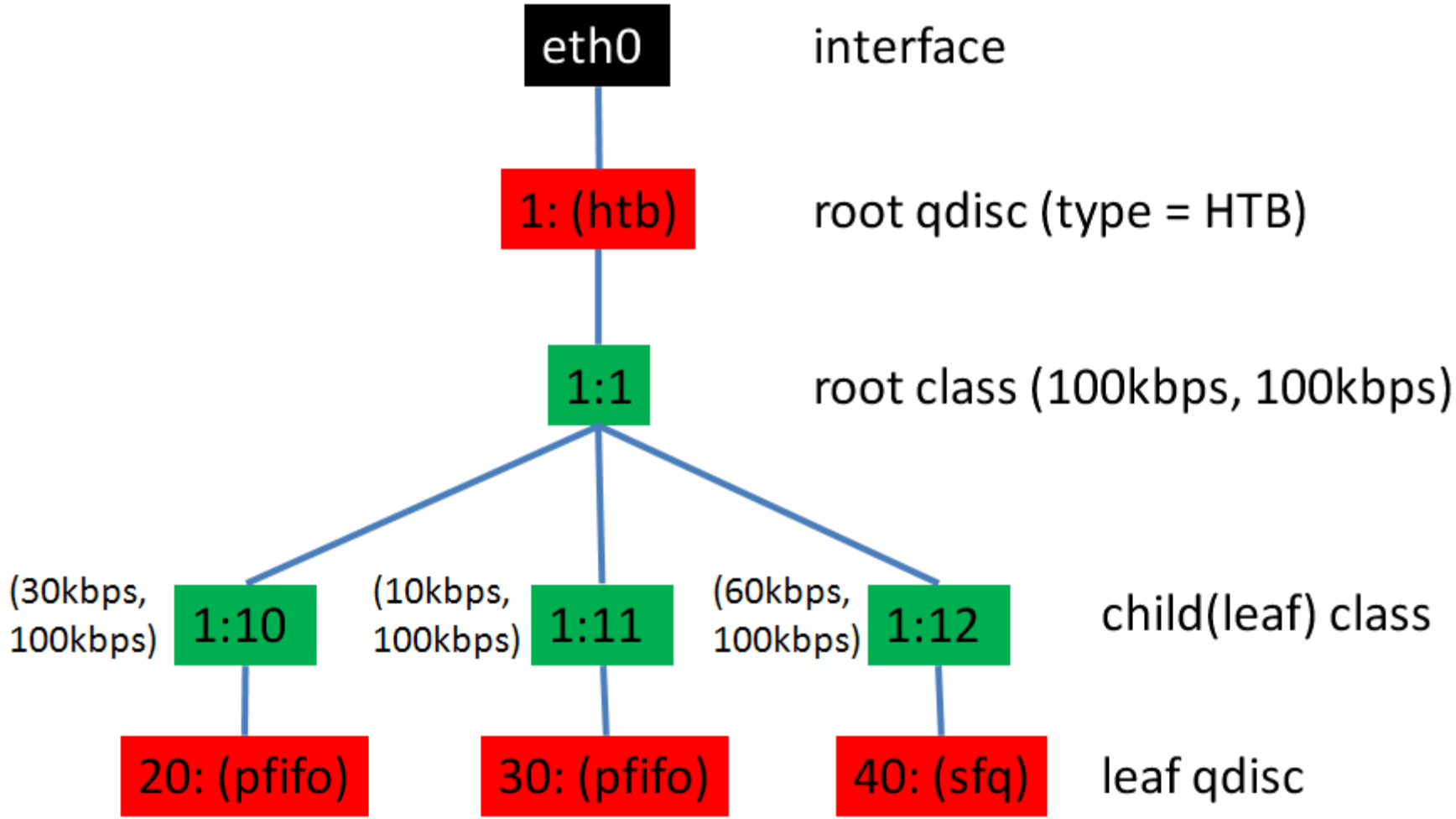
Openvswitch: QoS

- **Classful Queuing Disciplines**
- HTB, Hierarchical Token Bucket
  - Shaping：仅仅发生在叶子节点，依赖于其他的Queue
  - Borrowing: 当网络资源空闲的时候，借点过来为我所用
    - Rate：设定的发送速度
    - Ceil：最大的速度，和rate之间的差是最多能向别人借多少



type of class	class state	HTB internal state	action taken
leaf	< rate	HTB_CAN_SEND	Leaf class will dequeue queued bytes up to available tokens (no more than burst packets)
leaf	> rate, < ceil	HTB_MAY_BORROW	Leaf class will attempt to borrow tokens/ctokens from parent class. If tokens are available, they will be lent in quantum increments and the leaf class will dequeue up to cburst bytes
leaf	> ceil	HTB_CANT_SEND	No packets will be dequeued. This will cause packet delay and will increase latency to meet the desired rate.
inner, root	< rate	HTB_CAN_SEND	Inner class will lend tokens to children.
inner, root	> rate, < ceil	HTB_MAY_BORROW	Inner class will attempt to borrow tokens/ctokens from parent class, lending them to competing children in quantum increments per request.
inner, root	> ceil	HTB_CANT_SEND	Inner class will not attempt to borrow from its parent and will not lend tokens/ctokens to children classes.

Openvswitch: QoS

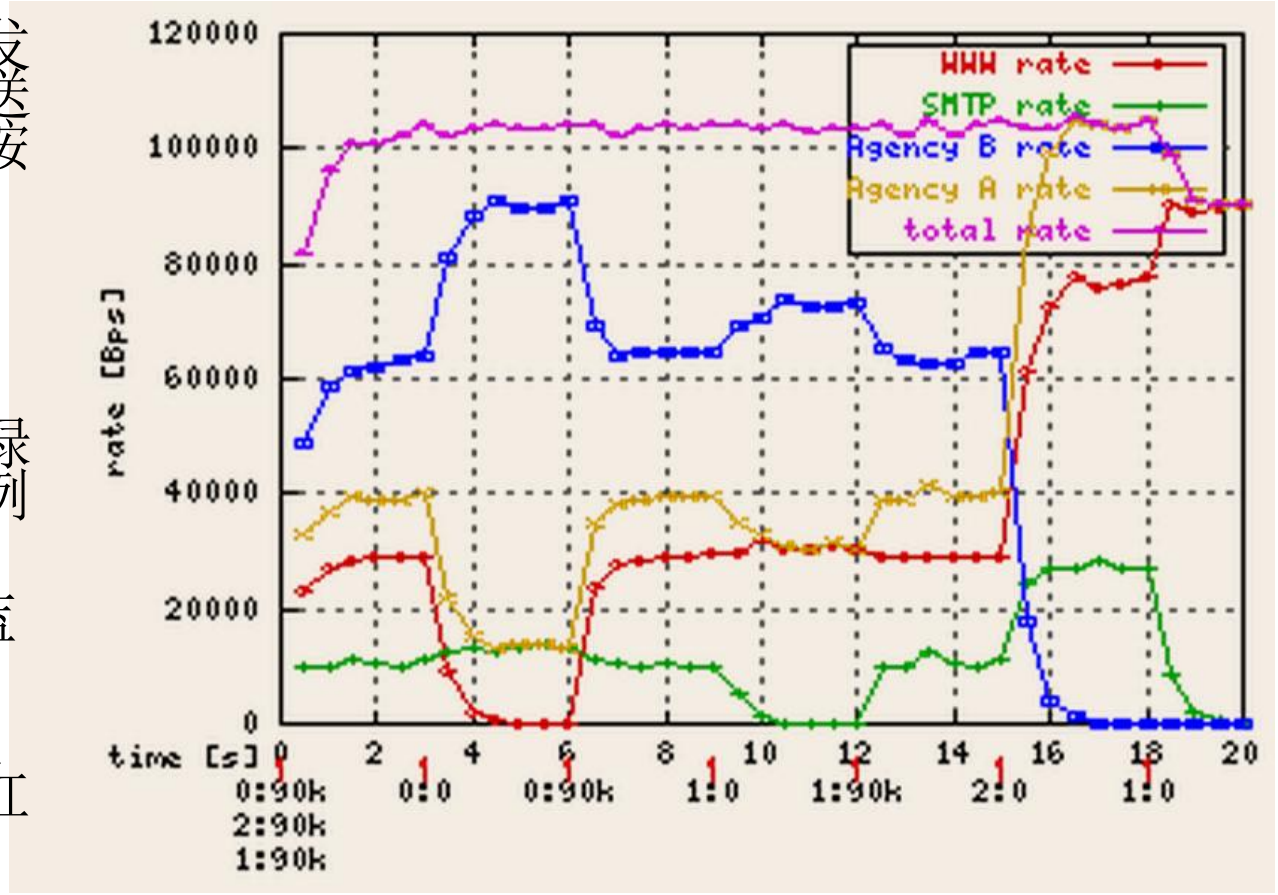


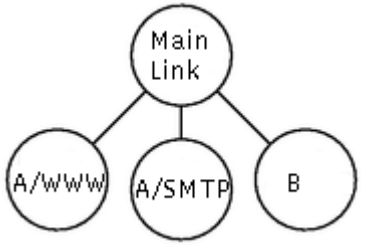
Openvswitch: QoS

创建一个HTB的qdisc在eth0上，句柄为1:，default 12表示默认发送给1:12
tc qdisc add dev eth0 root handle 1: htb default 12
创建一个root class，然后创建几个子class
同一个root class下的子类可以相互借流量，如果直接不在qdisc下面创建一个root class，而是直接创建三个class，他们之间是不能相互借流量的。
tc class add dev eth0 parent 1: classid 1:1 htb rate 100kbps ceil 100kbps
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 30kbps ceil 100kbps
tc class add dev eth0 parent 1:1 classid 1:11 htb rate 10kbps ceil 100kbps
tc class add dev eth0 parent 1:1 classid 1:12 htb rate 60kbps ceil 100kbps
创建叶子qdisc，分别为fifo和sfq
tc qdisc add dev eth0 parent 1:10 handle 20: pfifo limit 5
tc qdisc add dev eth0 parent 1:11 handle 30: pfifo limit 5
tc qdisc add dev eth0 parent 1:12 handle 40: sfq perturb 10
设定规则：从1.2.3.4来的，发送给port 80的包，从1:10走；其他从1.2.3.4发送来的包从1:11走；其他的走默认
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 match ip src 1.2.3.4 match ip dport 80 0xffff flowid 1:10
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 match ip src 1.2.3.4 flowid 1:11

Openvswitch: QoS

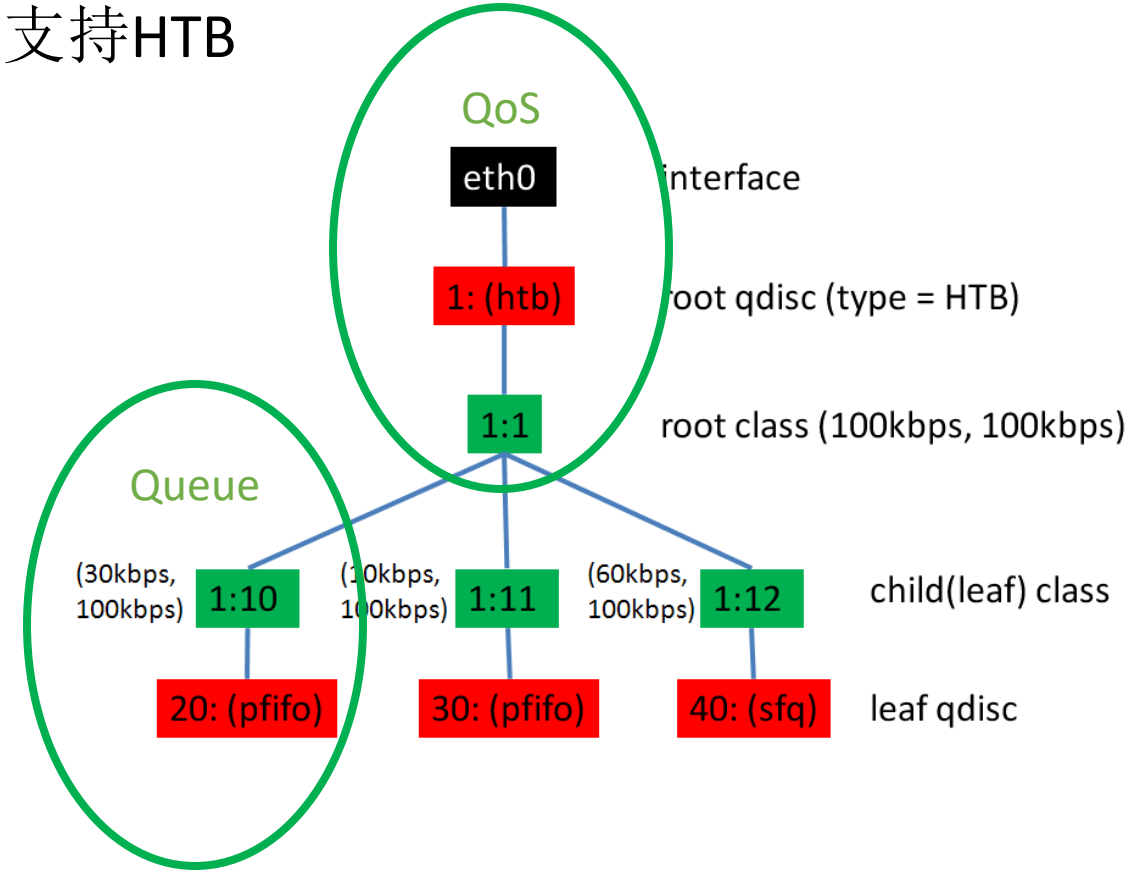
- 时间0的时候，0,1,2都以90k的速度发送数据，在时间3的时候，将0的发送停止，红色的线归零，剩余的流量按照比例分给了蓝色的和绿色的线。
- 在时间6的时候，将0的发送重启为90k，则蓝色和绿色的流量返还给红色的流量。
- 在时间9的时候，将1的发送停止，绿色的流量为零，剩余的流量按照比例分给了蓝色和红色。
- 在时间12,将1的发送恢复，红色和蓝色返还流量。
- 在时间15，将2的发送停止，蓝色流量为零，剩余的流量按照比例分给红色和绿色。
- 在时间19，将1的发送停止，绿色的流量为零，所有的流量都归了红色。





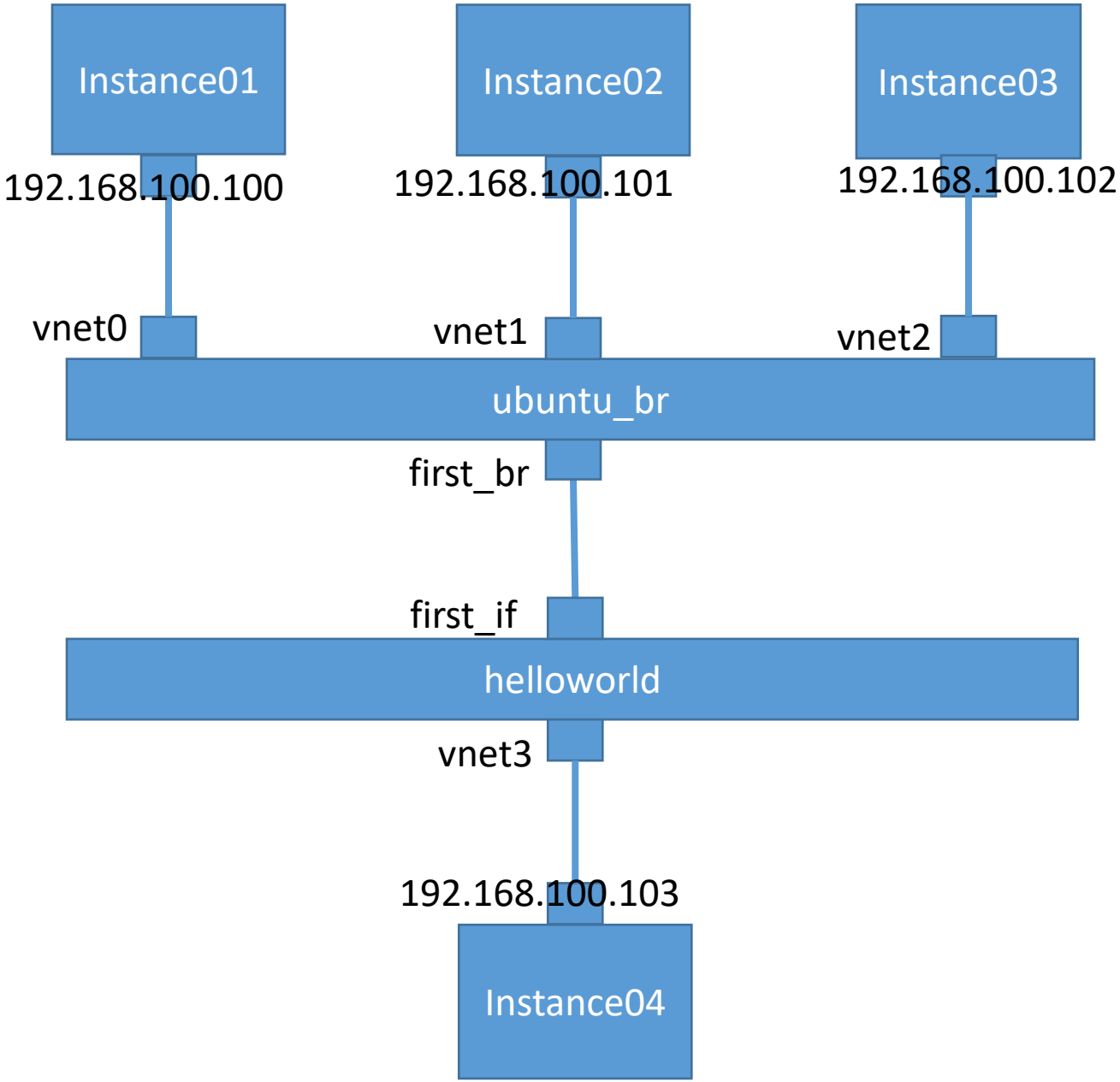
Openvswitch: QoS

- Openvswitch支持两种：
  - Ingress policy
    - ovs-vsctl set Interface tap0 ingress_policing_rate=100000
    - ovs-vsctl set Interface tap0 ingress_policing_burst=10000
  - Egress shaping: Port QoS policy 仅支持HTB
    - 在port上可以创建QoS
    - 一个QoS可以有多个Queue
    - 规则通过Flow设定



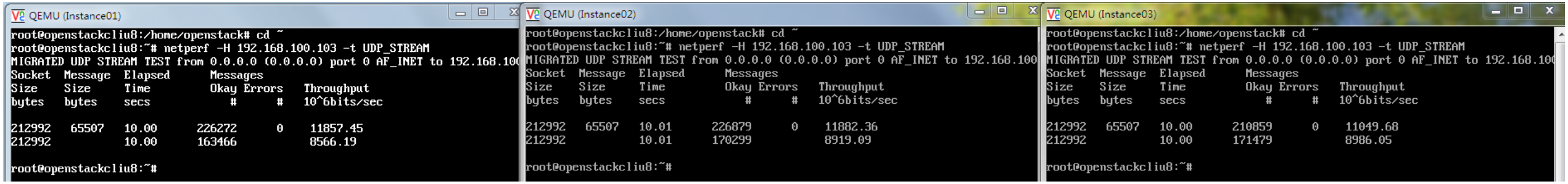
实验十: 测试QoS功能

```
root@popsuper1982:/home/openstack# ovs-vsctl show
d4e05278-c21f-4ed4-be3a-7853ea971931
    Bridge helloworld
        Port "vnet3"
            Interface "vnet3"
        Port first_if
            Interface first_if
        Port helloworld
            Interface helloworld
                type: internal
    Bridge ubuntu_br
        Port first_br
            Interface first_br
        Port "vnet0"
            Interface "vnet0"
        Port ubuntu_br
            Interface ubuntu_br
                type: internal
        Port "vnet1"
            Interface "vnet1"
        Port "vnet2"
            Interface "vnet2"
    ovs_version: "2.0.1"
```



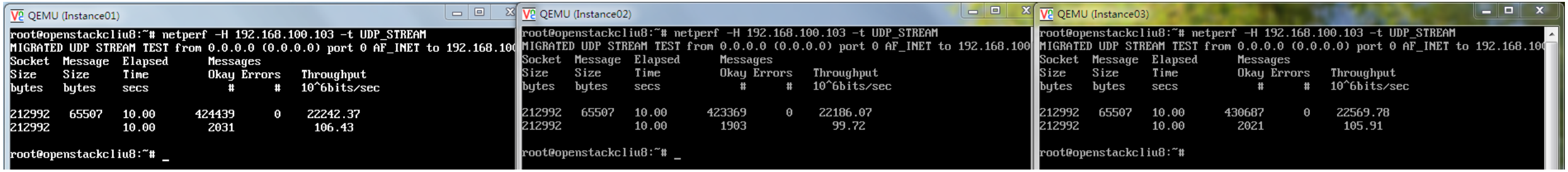
实验十: 测试QoS功能

- 在什么都没有配置的时候，测试一下速度，从192.168.100.100 netperf 192.168.100.103



- 设置 一下first_if

ovs-vsctl set Interface first_if ingress_policing_rate=100000
ovs-vsctl set Interface first_if ingress_policing_burst=10000



实验十: 测试QoS功能

- 清理现场
  - ovs-vsctl set Interface first_if ingress_policing_burst=0
  - ovs-vsctl set Interface first_if ingress_policing_rate=0
  - ovs-vsctl list Interface first_if
- 添加QoS

ovs-vsctl set port first_br qos=@newqos -- --id=@newqos create qos type=linux-htb other-config:max-rate=10000000 queues=0=@q0,1=@q1,2=@q2 -- --id=@q0 create queue other-config:min-rate=3000000 other-config:max-rate=10000000 -- --id=@q1 create queue other-config:min-rate=1000000 other-config:max-rate=10000000 -- --id=@q2 create queue other-config:min-rate=6000000 other-config:max-rate=10000000

- 添加Flow(first_br是ubuntu_br上的port 5)

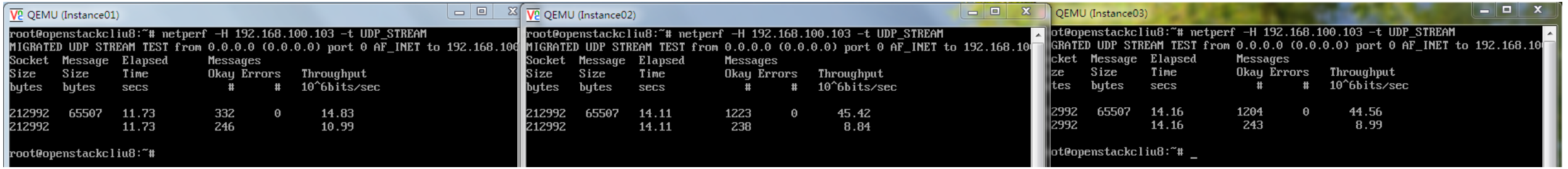
ovs-ofctl add-flow ubuntu_br "in_port=6 nw_src=192.168.100.100 actions=enqueue:5:0"
ovs-ofctl add-flow ubuntu_br "in_port=7 nw_src=192.168.100.101 actions=enqueue:5:1"
ovs-ofctl add-flow ubuntu_br "in_port=8 nw_src=192.168.100.102 actions=enqueue:5:2"

实验十: 测试QoS功能

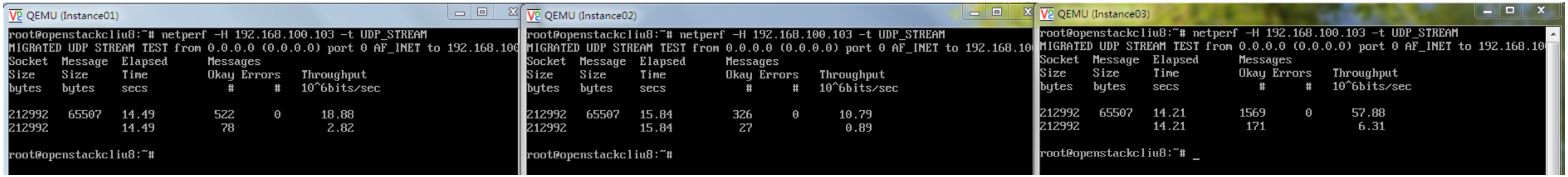
```
root@popsuper1982:/home/openstack# ovs-ofctl show ubuntu_br
OFPT_FEATURES_REPLY (xid=0x2): dpid:00002a960ec78549
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_
 5(first_br): addr:52:c3:37:00:67:20
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 6(vnet0): addr:fe:54:00:9b:d5:11
     config:     0
     state:      0
     current:    10MB-FD COPPER
     speed: 10 Mbps now, 0 Mbps max
 7(vnet1): addr:fe:54:00:9b:d5:33
     config:     0
     state:      0
     current:    10MB-FD COPPER
     speed: 10 Mbps now, 0 Mbps max
 8(vnet2): addr:fe:54:00:9b:d5:77
     config:     0
     state:      0
     current:    10MB-FD COPPER
     speed: 10 Mbps now, 0 Mbps max
 LOCAL(ubuntu_br): addr:2a:96:0e:c7:85:49
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
root@popsuper1982:/home/openstack# ovs-ofctl dump-flows ubuntu_br
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=36.747s, table=0, n_packets=0, n_bytes=0, idle_age=36, in_port=8 actions=enqueue:5q2
 cookie=0x0, duration=36.766s, table=0, n_packets=0, n_bytes=0, idle_age=36, in_port=7 actions=enqueue:5q1
 cookie=0x0, duration=36.776s, table=0, n_packets=0, n_bytes=0, idle_age=36, in_port=6 actions=enqueue:5q0
 cookie=0x0, duration=2819.085s, table=0, n_packets=7988138, n_bytes=211922467650, idle_age=252, priority=0 actions=NORMAL
```

实验十: 测试QoS功能

- 单独测试从192.168.100.100，192.168.100.101，192.168.100.102到192.168.100.103

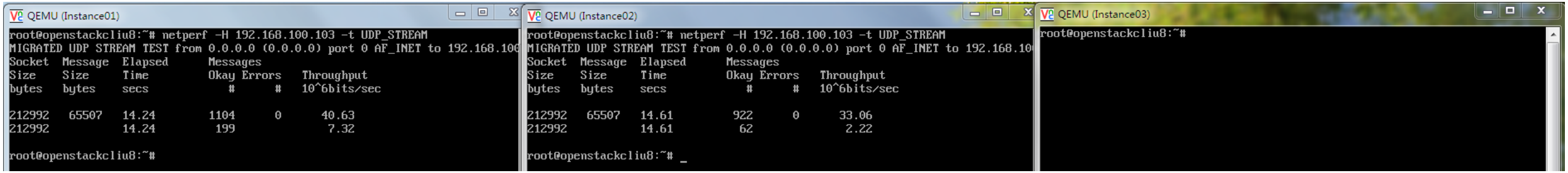


- 如果三个一起测试，发现是按照比例3:1:6进行的

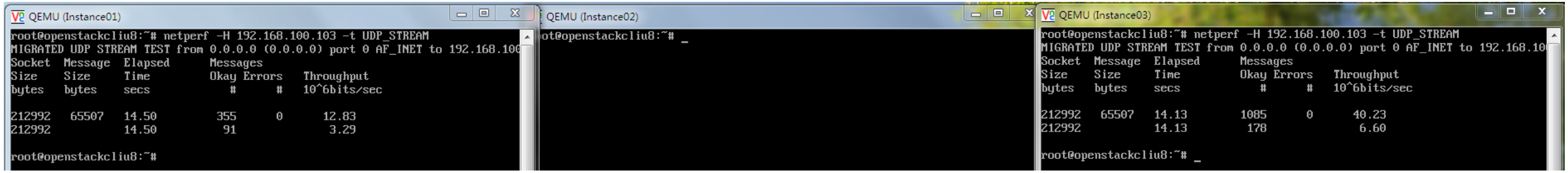


实验十: 测试QoS功能

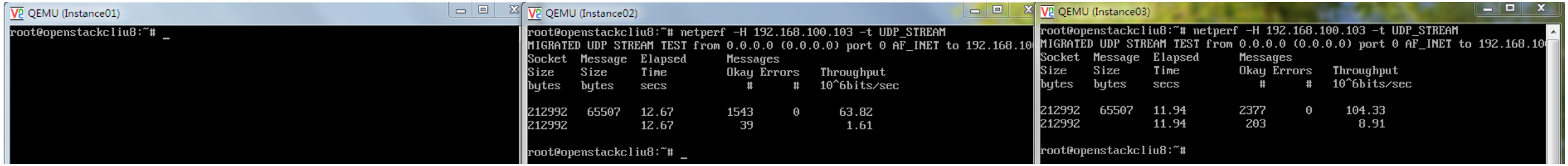
- 如果Instance01和Instance02一起，则3:1



- 如果Instance01和Instance03一起，则1:2



- 如果Instance02和Instance03一起，则1:6



实验十: 测试QoS功能

- 清理环境

```
root@popsuper1982:/home/openstack# ovs-vsctl list Port first_br
_uuid               : 3a527d18-3ab2-49d7-9fc1-07d78a5619e9
bond_downdelay      : 0
bond_fake_iface     : false
bond_mode           : []
bond_updelay        : 0
external_ids        : {}
fake_bridge         : false
interfaces          : [0fbffb46-ce2a-41e7-b463-15bd2f80a4d1]
lacp                : []
mac                 : []
name                : first_br
other_config        : {}
qos                 : 447d3e1b-4668-4d6d-ad6c-c6aa718546c0
statistics          : {}
status              : {}
tag                 : []
trunks              : []
vlan_mode           : []
root@popsuper1982:/home/openstack# ovs-vsctl clear Port first_br qos
```

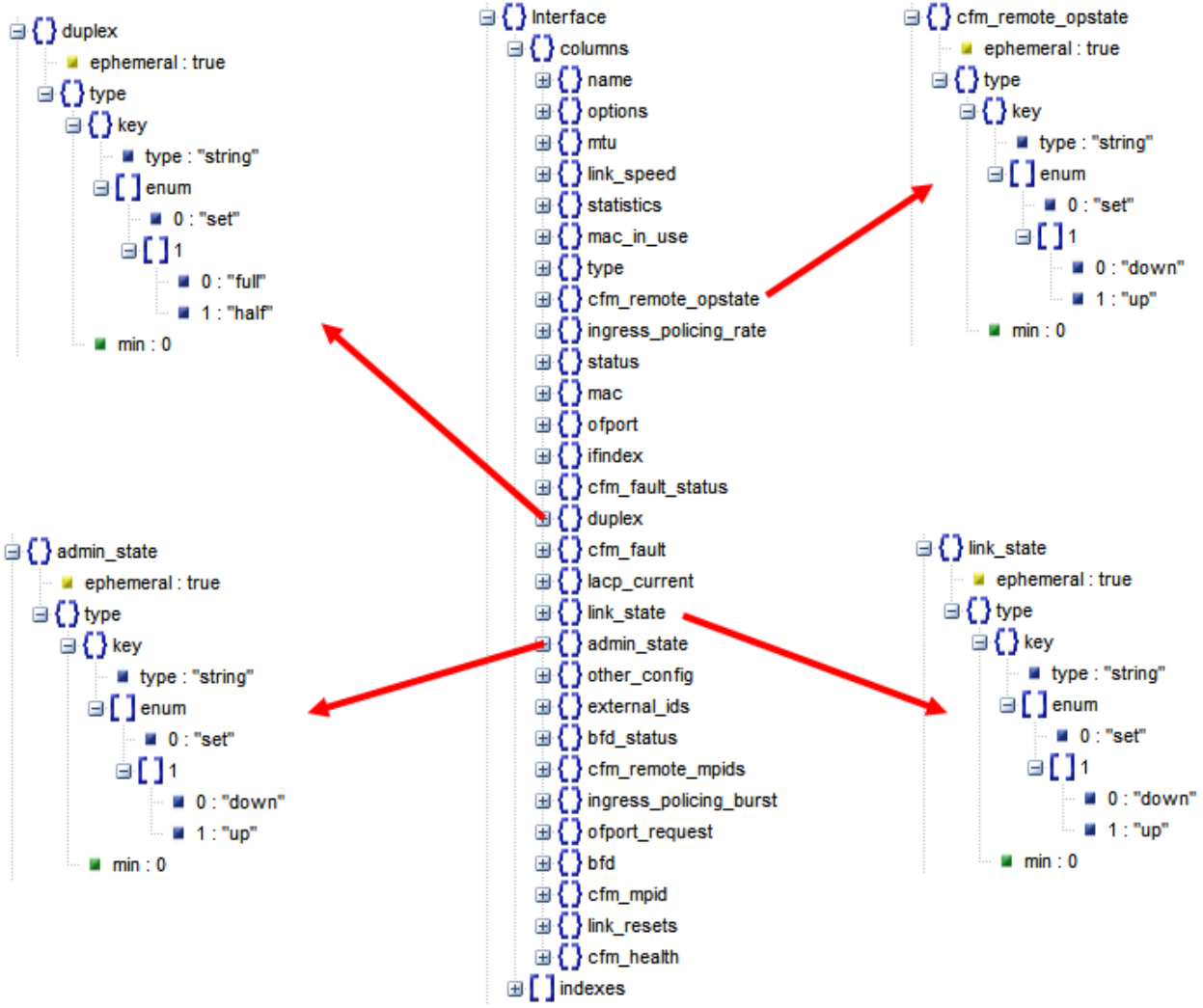
```
root@popsuper1982:/home/openstack# ovs-vsctl list Queue
_uuid               : 2031513c-0478-4a8e-bcf0-85546569c080
dscp                : []
external_ids        : {}
other_config        : {max-rate="10000000", min-rate="3000000"}

_uuid               : 32361b1a-3796-40fa-857f-c745d3af8d6c
dscp                : []
external_ids        : {}
other_config        : {max-rate="10000000", min-rate="6000000"}

_uuid               : 3350435f-a381-4669-9e75-914cee7fd995
dscp                : []
external_ids        : {}
other_config        : {max-rate="10000000", min-rate="1000000"}
root@popsuper1982:/home/openstack# ovs-vsctl destroy Queue 2031513c-0478-4a8e-bcf0-85546569c080
root@popsuper1982:/home/openstack# ovs-vsctl destroy Queue 32361b1a-3796-40fa-857f-c745d3af8d6c
root@popsuper1982:/home/openstack# ovs-vsctl destroy Queue 3350435f-a381-4669-9e75-914cee7fd995
root@popsuper1982:/home/openstack# ovs-vsctl list Queue
```

```
root@popsuper1982:/home/openstack# ovs-vsctl list QoS
_uuid               : 447d3e1b-4668-4d6d-ad6c-c6aa718546c0
external_ids        : {}
other_config        : {max-rate="10000000"}
queues              : {0=2031513c-0478-4a8e-bcf0-85546569c080, 1=3350435f-a381-4669-9e75-914cee7fd995, 2=32361b1a-3796-40fa-857f-c745d3af8d6c}
type                : linux-htb
root@popsuper1982:/home/openstack# ovs-vsctl destroy QoS 447d3e1b-4668-4d6d-ad6c-c6aa718546c0
root@popsuper1982:/home/openstack# ovs-vsctl list QoS
```

```
root@popsuper1982:/home/openstack# ovs-ofctl dump-flows ubuntu_br
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=2493.197s, table=0, n_packets=25618, n_bytes=622886161, idle_age=614, in_port=8 actions=enqueue:5q2
 cookie=0x0, duration=2493.216s, table=0, n_packets=20397, n_bytes=598012919, idle_age=579, in_port=7 actions=enqueue:5q1
 cookie=0x0, duration=2493.226s, table=0, n_packets=14023, n_bytes=348384175, idle_age=695, in_port=6 actions=enqueue:5q0
 cookie=0x0, duration=5275.535s, table=0, n_packets=7989380, n_bytes=211923003426, idle_age=579, priority=0 actions=NORMAL
root@popsuper1982:/home/openstack# ovs-ofctl del-flows ubuntu_br
root@popsuper1982:/home/openstack# ovs-ofctl dump-flows ubuntu_br
NXST_FLOW reply (xid=0x4):
```

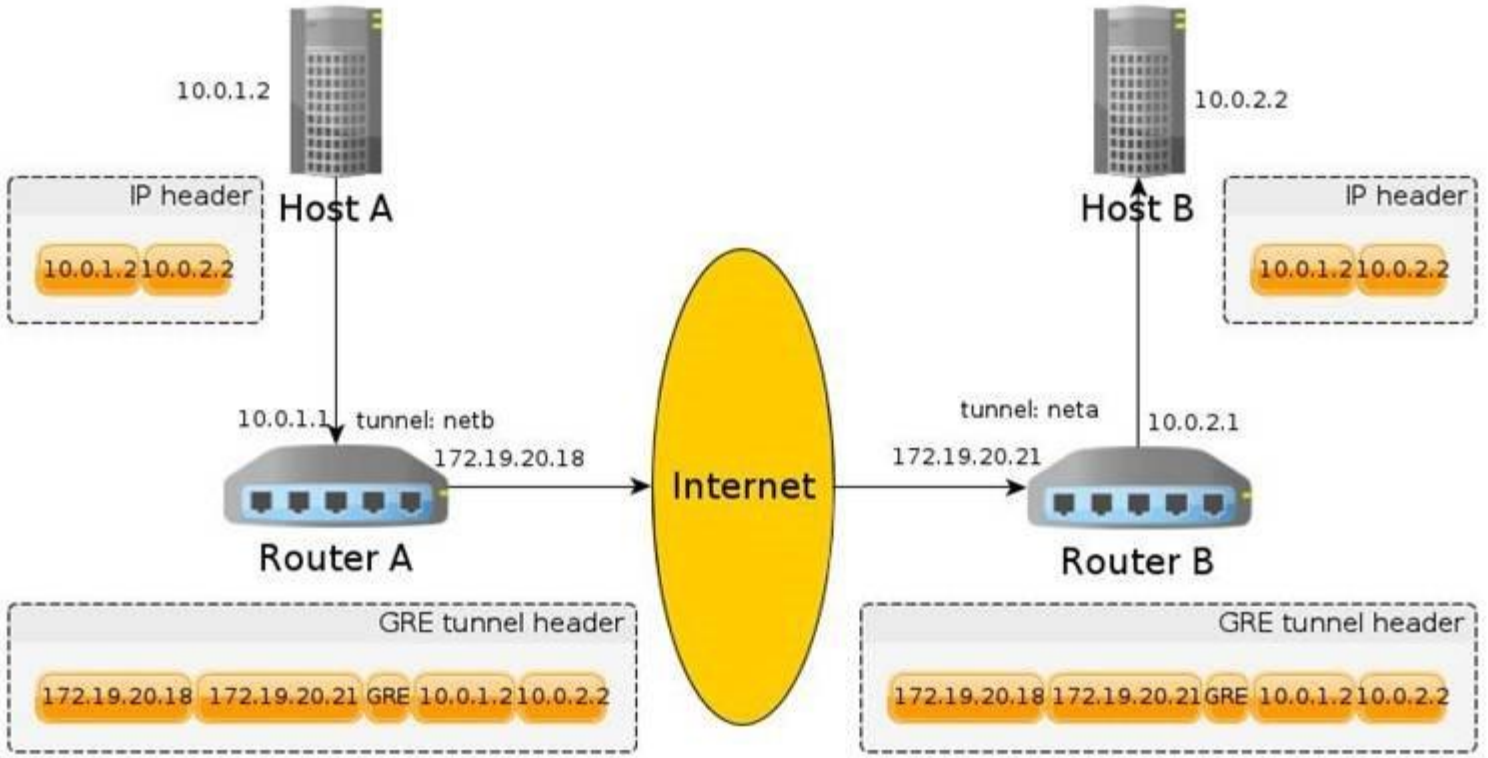
Openvswitch: Tunnel

- gre
- vxlan
- ipsec_gre



Openvswitch: Tunnel GRE

- **Generic Routing Encapsulation** (**GRE**) is a tunneling protocol that can encapsulate a wide variety of network layer protocols inside virtual point-to-point links over an Internet Protocol internetwork.



Openvswitch: Tunnel GRE

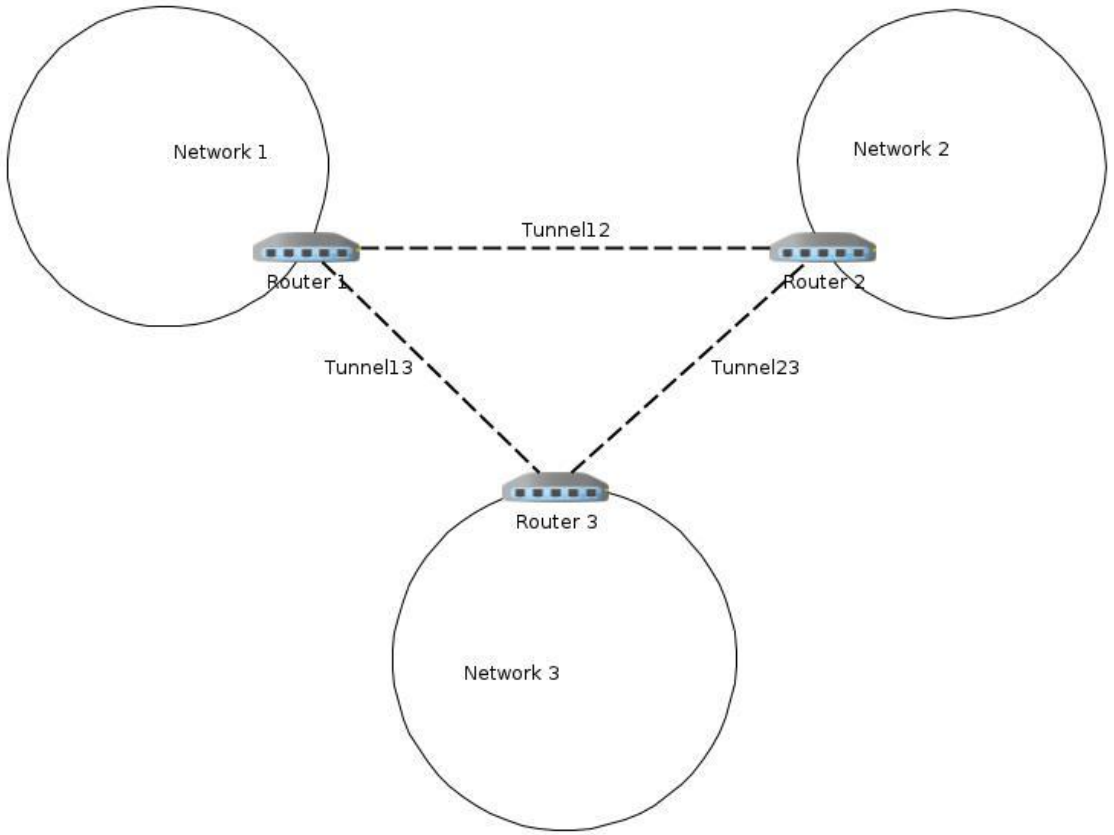
- GRE Header

Bits 0–3				4–12	13–15	16–31
C		K	S	Reserved0	Version	Protocol Type
Checksum *(optional)*						Reserved1 *(optional)*
Key *(optional)*						
Sequence Number *(optional)*						

- 从L2到L3，数据可打包后跨越网关和路由器然后解包称为L2的数据

Openvswitch: Tunnel GRE

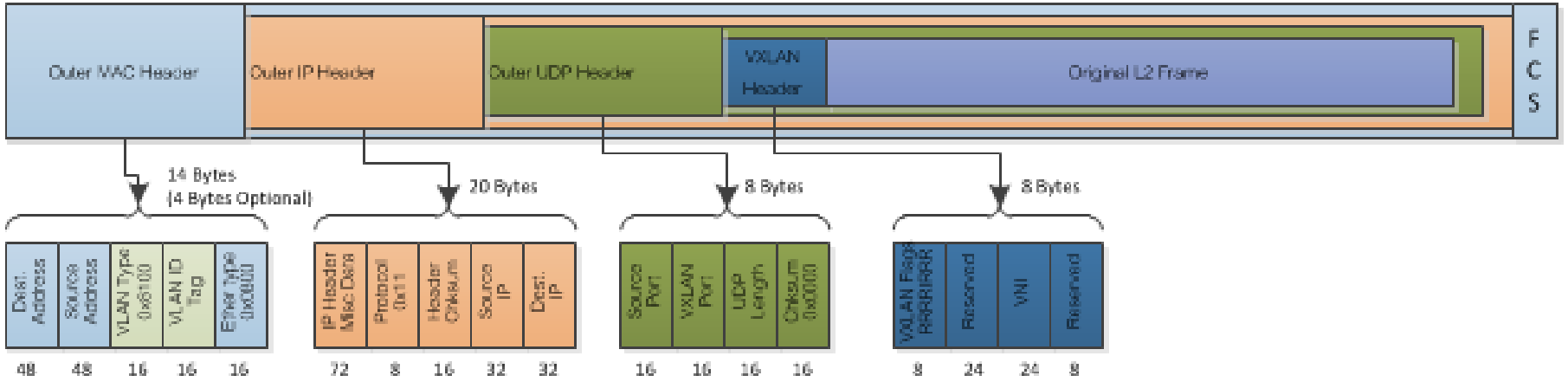
- 缺点：
  - 点对点，扩展性不好
  - 网络设备对GRE包头支持有限，往往负载均衡和防火墙ACL都是根据IP和Port来的。



Openvswitch: Tunnel VXLAN

- VXLAN：通过对L2包的打包和解包实现不同的L2网络感觉在同一个L2网络里面
- Components:
  - Multicast support, IGMP and PIM
  - VXLAN Network Identifier (VNI): 24-bit segment ID
  - VXLAN Gateway
  - VXLAN Tunnel End Point (VTEP)
  - VXLAN Segment/VXLAN Overlay Network

Openvswitch: Tunnel VXLAN



Ethernet Header:
Destination Address MAC address of the destination VTEP if it is local, MAC addr of gateway when the destination VTEP is on a different L3 network.

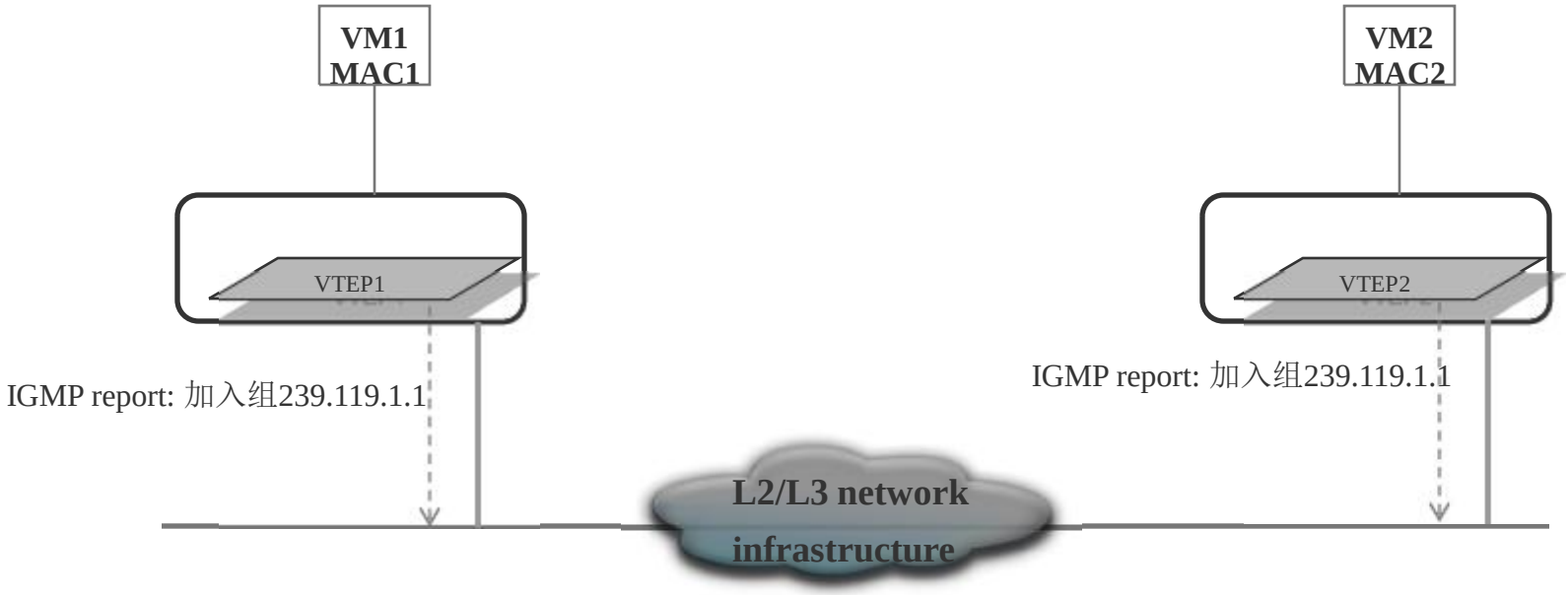
IP Header:
Protocol – Set 0×11 to indicate that the frame contains a UDP packet
Source IP – IP address of originating VTEP
Destination IP – IP address of target VTEP.

UDP Header:
Source Port – Set by transmitting VTEP
VXLAN Port – IANA assigned VXLAN Port.

VXLAN Header:
VNI – 24-bit field that is the VXLAN Network Identifier

Openvswitch: Tunnel VXLAN

- ARP

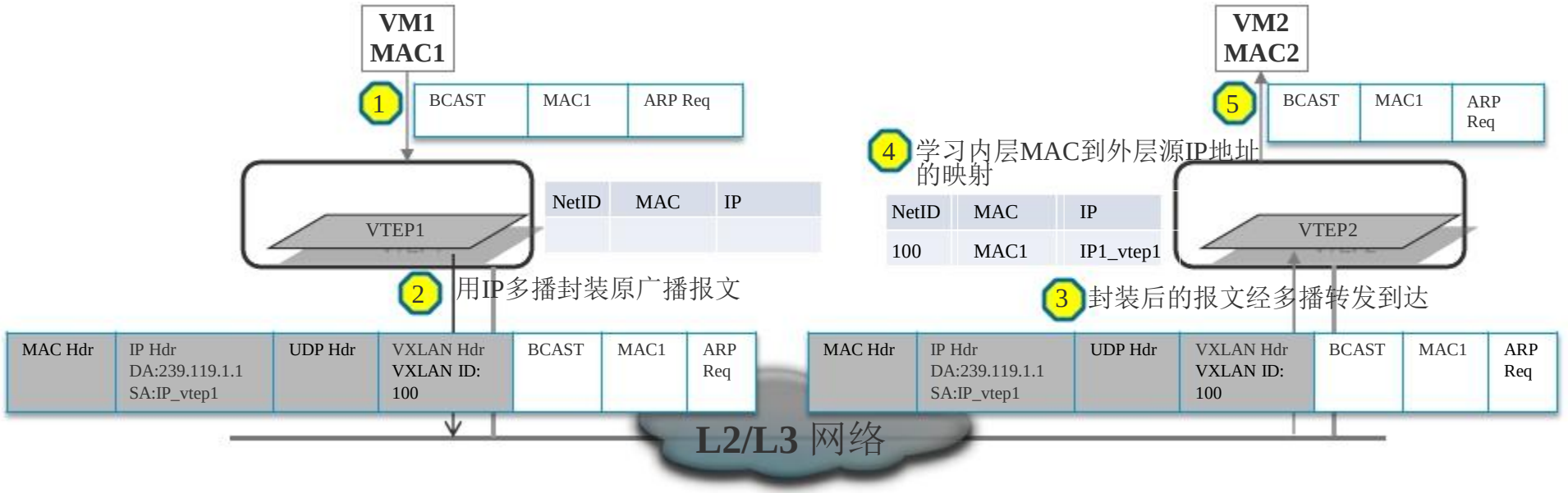


VM1及VM2连接到VXLAN网络100, 两个VXLAN主机加入IP多播组239.119.1.1

VTEP – VXLAN隧道终端 (VXLAN Tunneling End Point)

Openvswitch: Tunnel VXLAN

- ARP

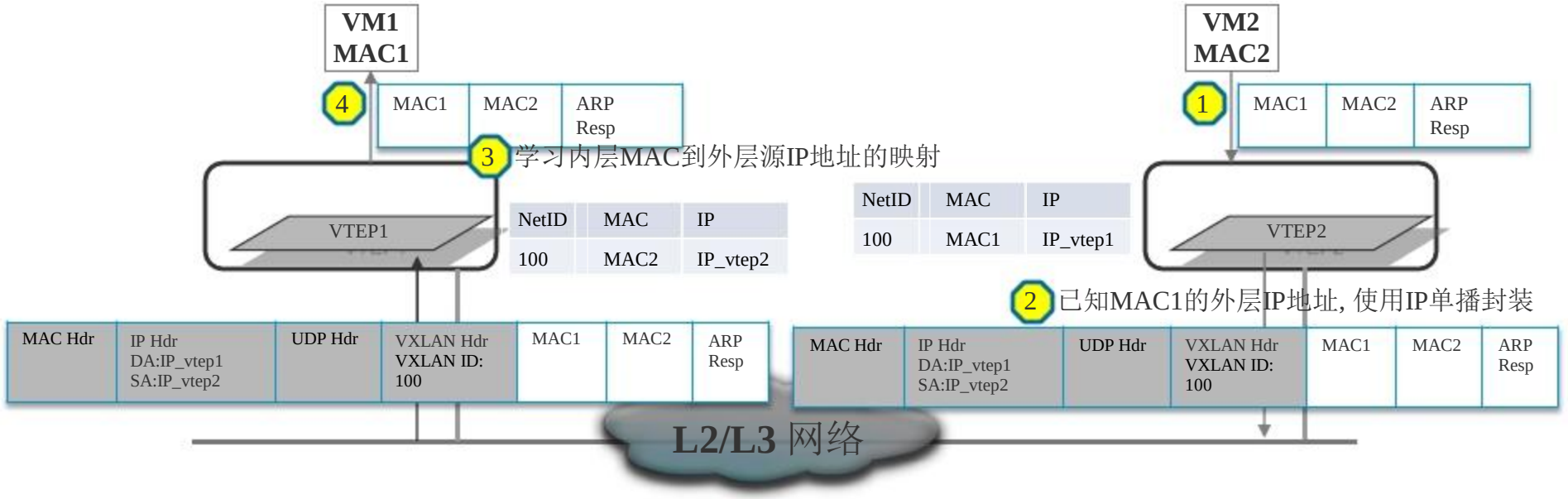


VM1发送ARP请求(广播)以获得VM2的MAC地址，VXLAN ID为100

VTEP – VXLAN隧道终端 (VXLAN Tunneling End Point)

Openvswitch: Tunnel VXLAN

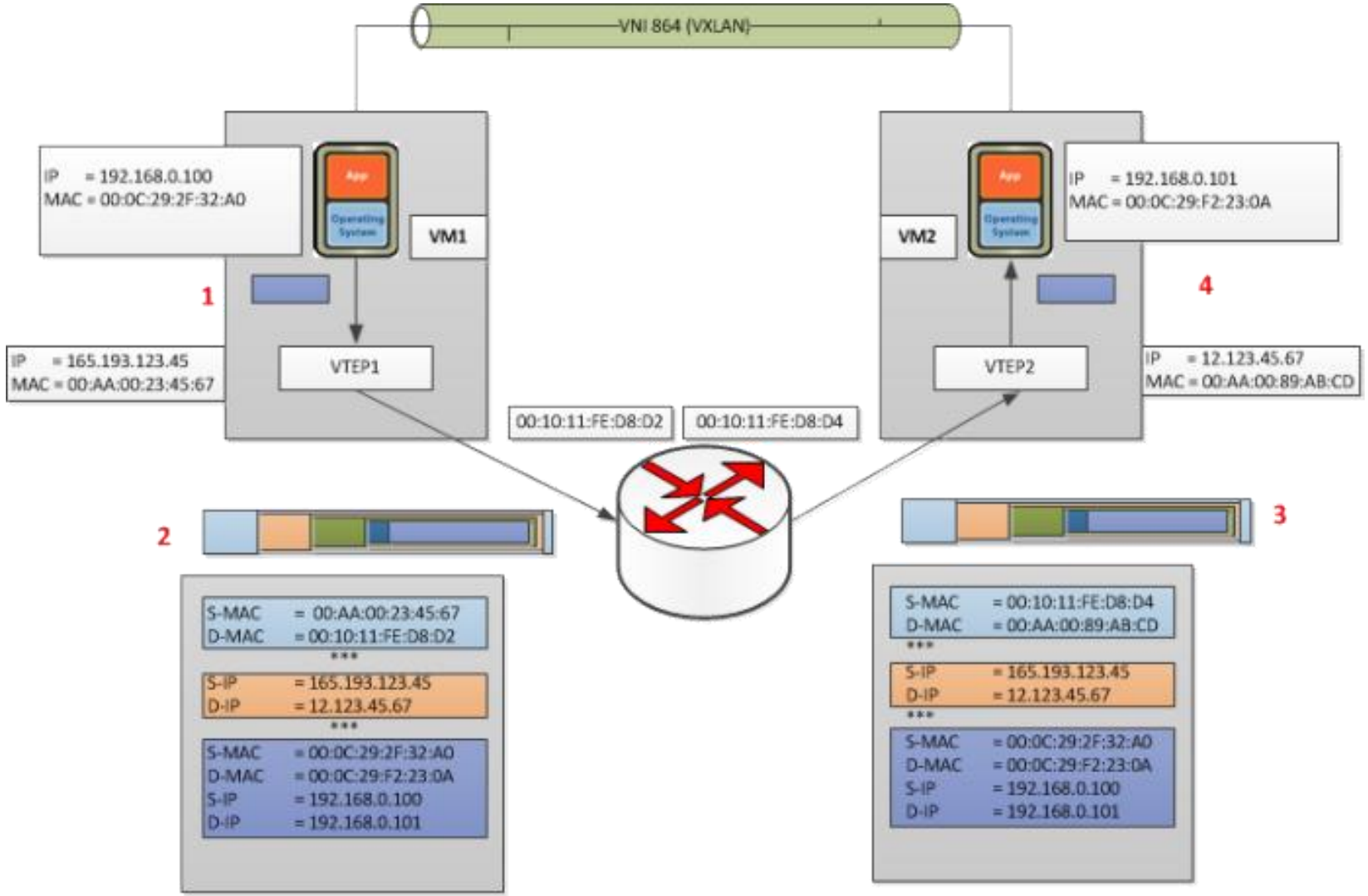
- ARP



VM2发送ARP应答(单播)到VM1

VTEP – VXLAN隧道终端 (VXLAN Tunneling End Point)

Openvswitch: Tunnel VXLAN

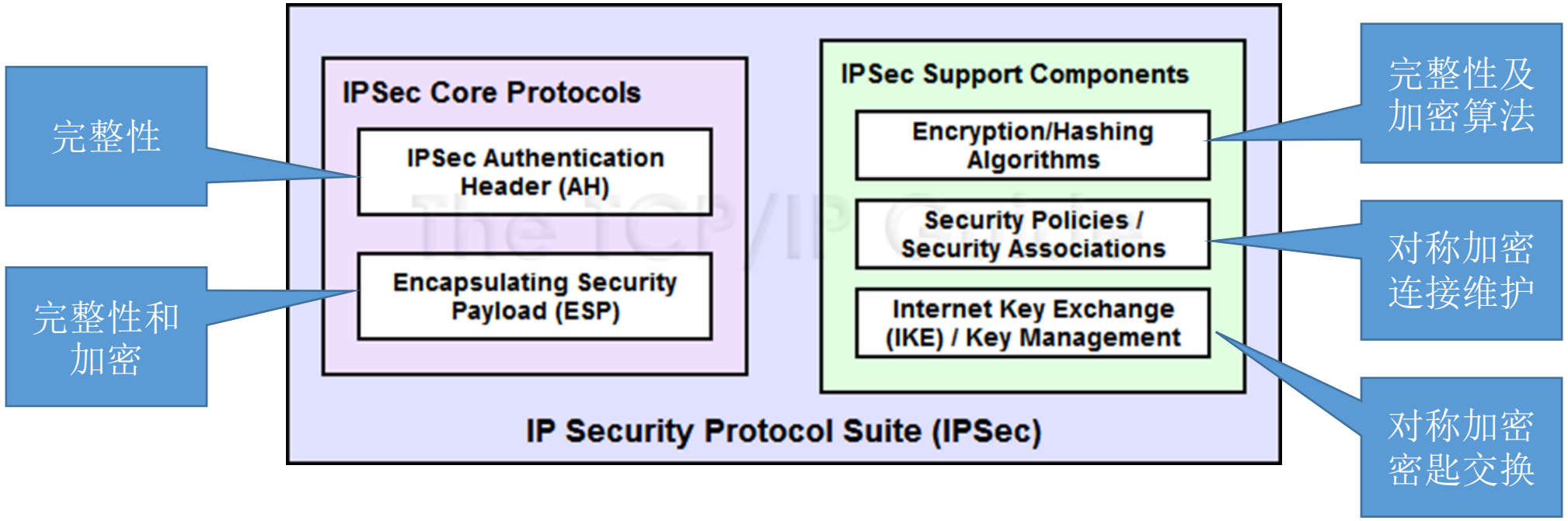


Openvswitch: Tunnel VXLAN

- 可支持Multicast，但是在数据中心常被禁止
- 硬件支持广泛

Openvswitch: Tunnel GRE over IPSec

- 安全通道，安全意味着
  - 信息完整性——认证
  - 信息的私密性——加密
  - 信息不会被Replay



Openvswitch: Tunnel GRE over IPSec

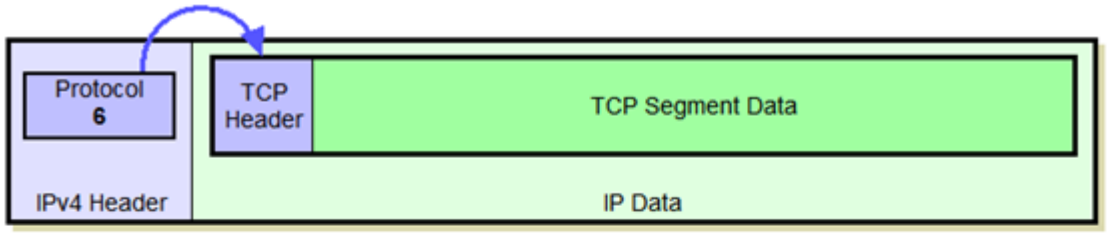
Security Association

ID
- Partner IP
- Ipsec Protocal (ESP/AH)
- Security Parameters Index (SPI)

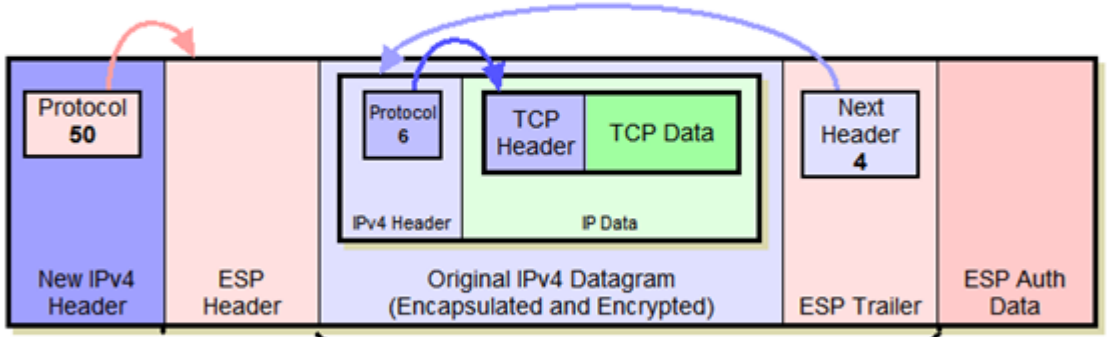
Infos
- ESP encryption algorithm
- ESP encryption secret key
- ESP Authentication
- ……

IPSec Key Exchange (IKE)

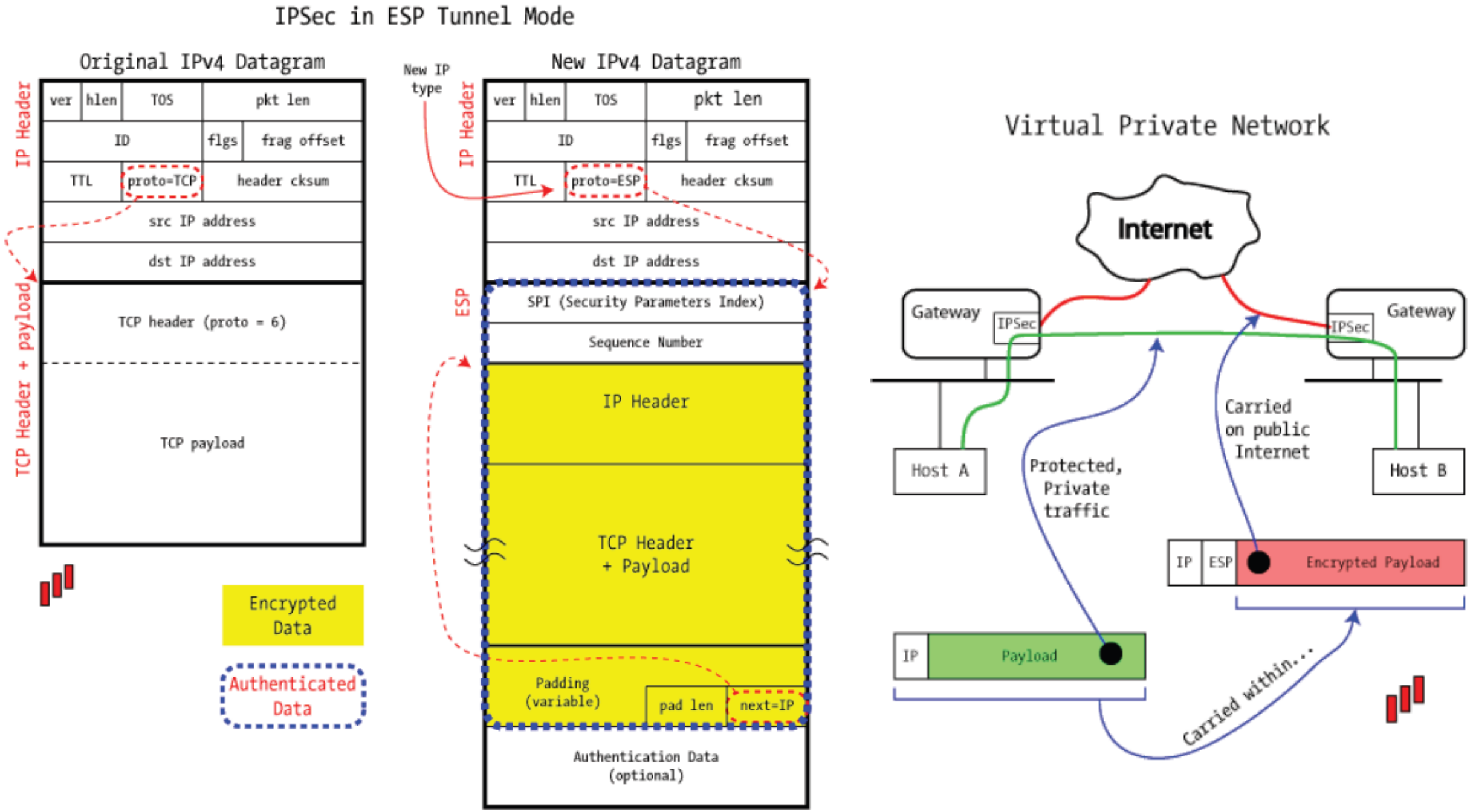
Internet Security Association and Key Management Protocol (ISAKMP)

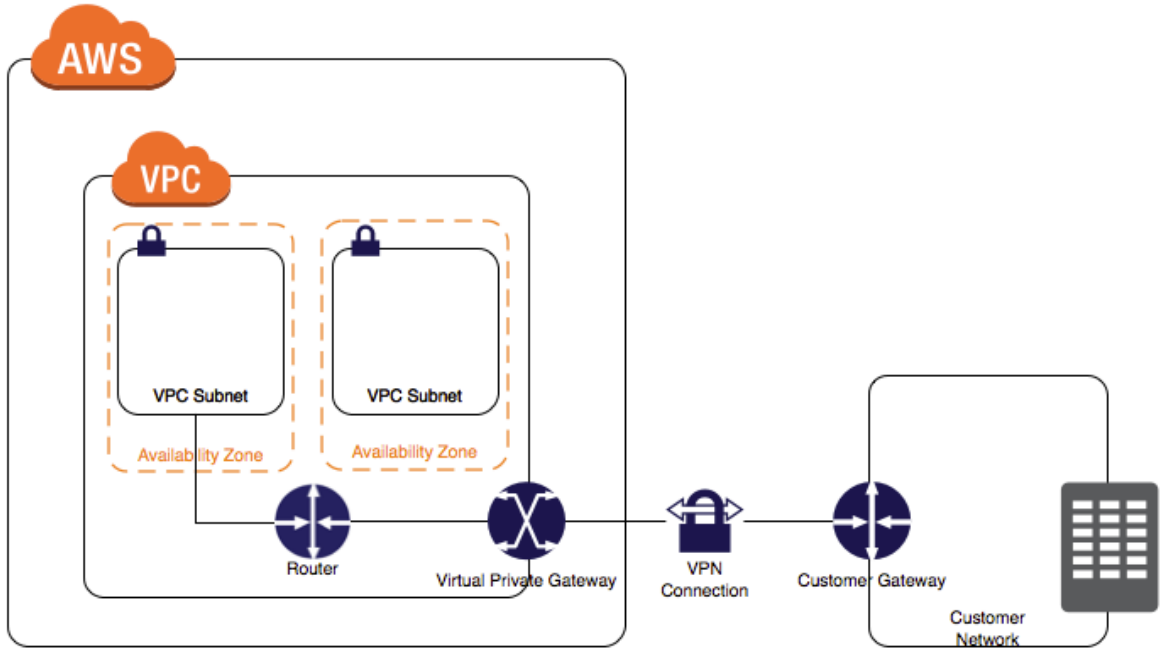


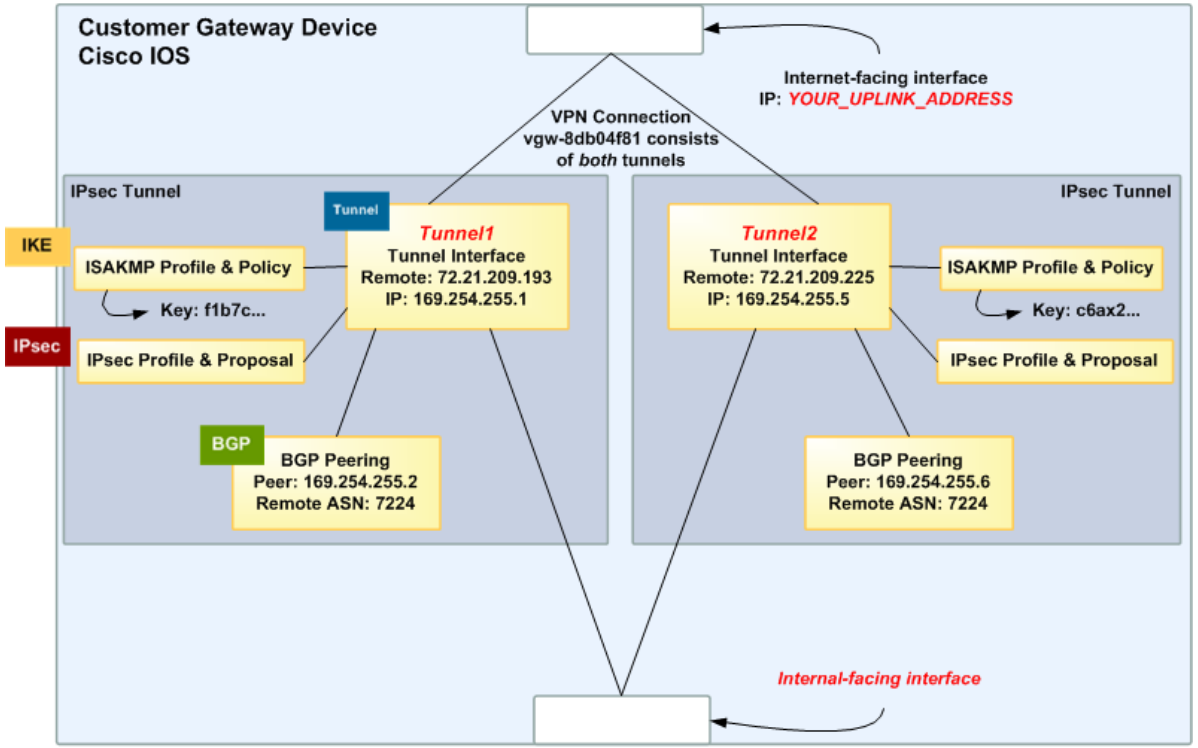
Original IPv4 Datagram Format



IPv4 ESP Datagram Format - IPSec Tunnel Mode


IPSec in ESP Tunnel Mode
Original IPv4 Datagram
IP Header
ver
hlen
TOS
pkt len
ID
flgs
frag offset
TTL
proto=TCP
header cksum
src IP address
dst IP address
TCP Header + payload
TCP header (proto = 6)
TCP payload
Encrypted Data
Authenticated Data
New IP type
New IPv4 Datagram
IP Header
ver
hlen
TOS
pkt len
ID
flgs
frag offset
TTL
proto=ESP
header cksum
src IP address
dst IP address
ESP
SPI (Security Parameters Index)
Sequence Number
IP Header
TCP Header + Payload
Padding (variable)
pad len
next=IP
Authentication Data (optional)
Virtual Private Network
Internet
Gateway
IPSec
IPSec
Gateway
Host A
Host B
Protected, Private traffic
Carried on public Internet
IP
ESP
Encrypted Payload
IP
Payload
Carried within...





IKE

```
crypto isakmp policy 200
  encryption aes 128
  authentication pre-share
  group 2
  lifetime 28800
  hash sha
exit
crypto keyring keyring-vpn-44a8938f-0
  pre-shared-key address 72.21.209.225 key plain-text-password1
exit
crypto isakmp profile isakmp-vpn-44a8938f-0
  match identity address 72.21.209.225
  keyring keyring-vpn-44a8938f-0
exit
```

IPsec

```
crypto ipsec transform-set ipsec-prop-vpn-44a8938f-0 esp-aes 128 esp-sha-hmac
  mode tunnel
exit
crypto ipsec profile ipsec-vpn-44a8938f-0
  set pfs group2
  set security-association lifetime seconds 3600
  set transform-set ipsec-prop-vpn-44a8938f-0
exit
```

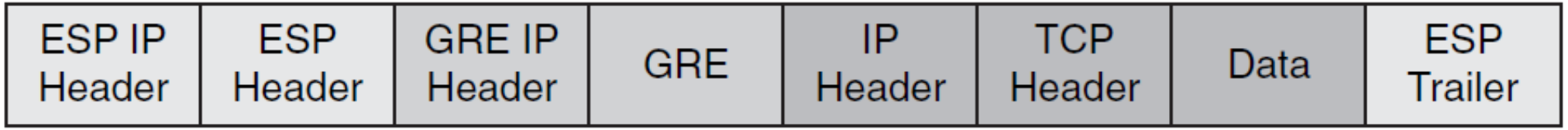
Tunnel

```
interface Tunnel1
  ip address 169.254.255.2 255.255.255.252
  ip virtual-reassembly
  tunnel source YOUR_UPLINK_ADDRESS
  tunnel destination 72.21.209.225
  tunnel mode ipsec ipv4
  tunnel protection ipsec profile ipsec-vpn-44a8938f-0
  ip tcp adjust-mss 1396
  no shutdown
exit
```

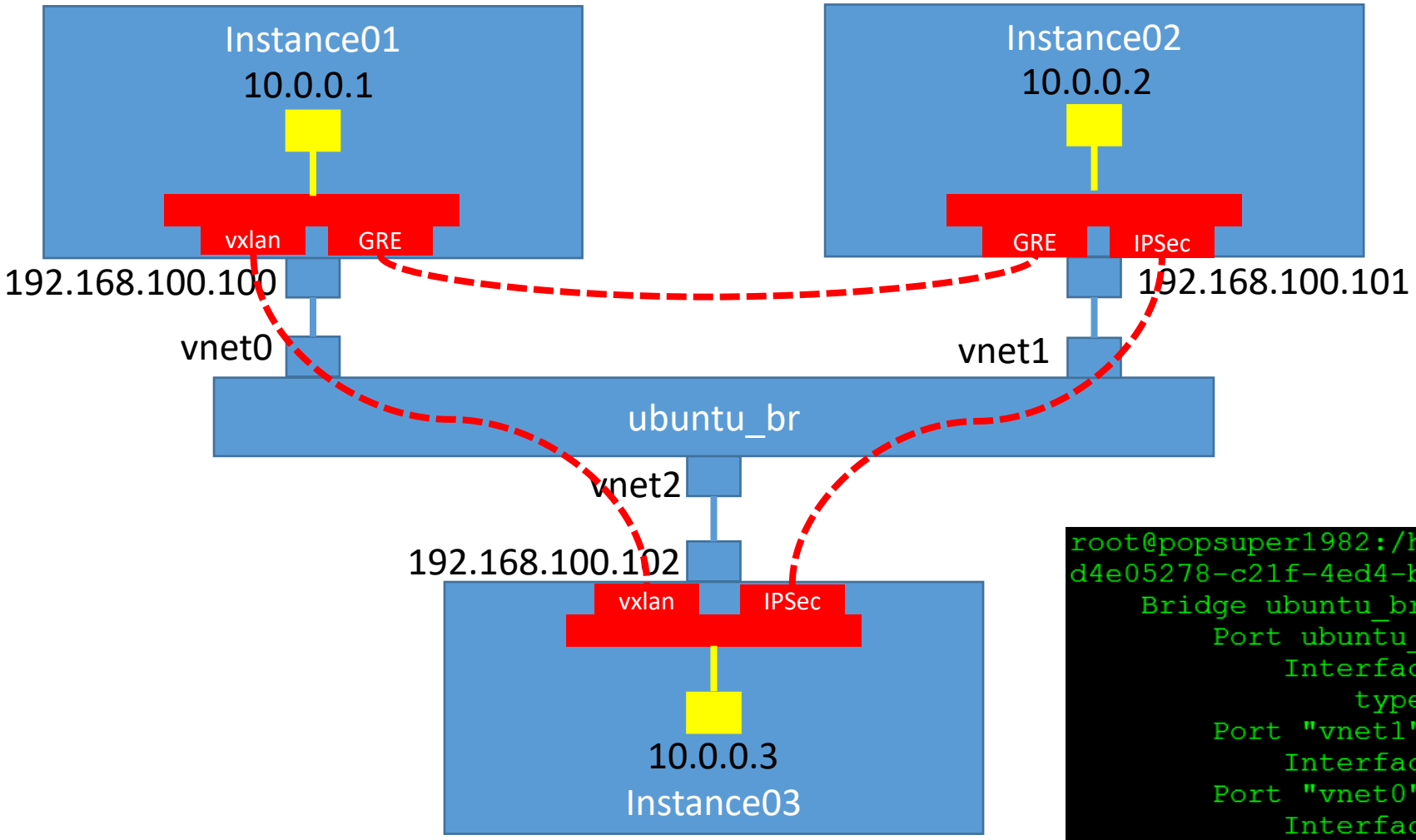
Openvswitch: Tunnel GRE over IPSec

GRE over IPsec Packet Format

Tunnel Mode



实验十一：创建Tunnel




```
root@popsuper1982:/home/openstack# ovs-vsctl show
d4e05278-c21f-4ed4-be3a-7853ea971931
    Bridge ubuntu_br
        Port ubuntu_br
            Interface ubuntu_br
                type: internal
        Port "vnet1"
            Interface "vnet1"
        Port "vnet0"
            Interface "vnet0"
        Port "vnet2"
            Interface "vnet2"
    ovs_version: "2.0.1"
```

实验十一：创建Tunnel

- 在虚拟机Instance01上

ovs-vsctl add-br testbr
ifconfig testbr 10.0.0.1/24
ovs-vsctl add-port testbr gre0 -- set Interface gre0 type=gre options:local_ip=192.168.100.100 options:remote_ip=192.168.100.101
ovs-vsctl add-port testbr vxlan0 -- set Interface vxlan0 type=vxlan options:local_ip=192.168.100.100 options:remote_ip=192.168.100.102

- 在虚拟机Instance02上

ovs-vsctl add-br testbr
ifconfig testbr 10.0.0.2/24
ovs-vsctl add-port testbr gre0 -- set Interface gre0 type=gre options:local_ip=192.168.100.101 options:remote_ip=192.168.100.100
ovs-vsctl add-port testbr ipsec0 -- set Interface ipsec0 type=ipsec_gre options:local_ip=192.168.100.101
options:remote_ip=192.168.100.102 options:psk=password

- 在虚拟机Instance03上

ovs-vsctl add-br testbr
ifconfig testbr 10.0.0.3/24
ovs-vsctl add-port testbr vxlan0 -- set Interface vxlan0 type=vxlan options:local_ip=192.168.100.102 options:remote_ip=192.168.100.100
ovs-vsctl add-port testbr ipsec0 -- set Interface ipsec0 type=ipsec_gre options:local_ip=192.168.100.102 options:remote_ip=192.168.100.101
options:psk=password

实验十一：创建Tunnel

- 不好，出现了环，没关系enable STP

ovs-vsctl set Bridge testbr stp_enable=true

- 监听Instance01的eth0: tcpdump -n -e -i eth0

```
13:19:37.883980 52:54:00:9b:d5:77 > 52:54:00:9b:d5:11, ethertype IPv4 (0x0800), length 102: 192.168.100.102.55628 > 192.168.100.100.4789: VXLAN, f
lags [I] (0x08), vni 0
32:dd:d2:46:eb:82 > 01:80:c2:00:00:00, 802.3, length 52: LLC, dsap STP (0x42) Individual, ssap STP (0x42) Command, ctrl 0x03: STP 802.1d, Config,
Flags [none], bridge-id 8000.2a:18:ca:77:13:4d.8001, length 35
13:19:37.884266 52:54:00:9b:d5:11 > 52:54:00:9b:d5:33, ethertype IPv4 (0x0800), length 90: 192.168.100.100 > 192.168.100.101: GREv0, proto TEB (0x
6558), length 56: 7a:48:fd:b0:80:f4 > 01:80:c2:00:00:00, 802.3, length 52: LLC, dsap STP (0x42) Individual, ssap STP (0x42) Command, ctrl 0x03: ST
P 802.1d, Config, Flags [none], bridge-id 8000.6a:08:bc:38:9c:43.8002, length 35
```

- 监听Instance02的eth0: tcpdump -n -e -i eth0

```
13:23:06.898922 52:54:00:9b:d5:77 > 52:54:00:9b:d5:33, ethertype IPv4 (0x0800), length 134: 192.168.100.102 > 192.168.100.101: ESP(spi=0x0be025cf,
seq=0x441), length 100
13:23:06.899265 52:54:00:9b:d5:11 > 52:54:00:9b:d5:33, ethertype IPv4 (0x0800), length 90: 192.168.100.100 > 192.168.100.101: GREv0, proto TEB (0x
6558), length 56: 7a:48:fd:b0:80:f4 > 01:80:c2:00:00:00, 802.3, length 52: LLC, dsap STP (0x42) Individual, ssap STP (0x42) Command, ctrl 0x03: ST
P 802.1d, Config, Flags [none], bridge-id 8000.6a:08:bc:38:9c:43.8002, length 35
```

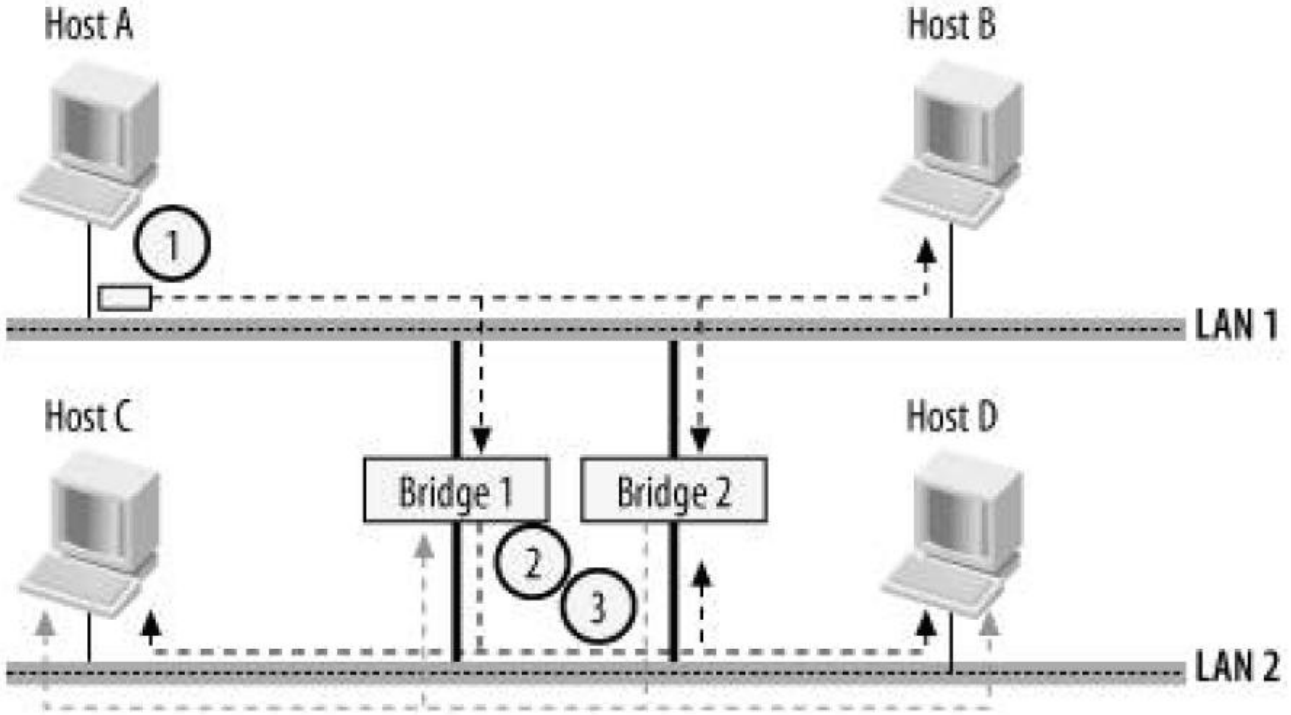
- 监听Instance03的eth0: tcpdump -n -e -i eth0

```
13:24:58.315349 52:54:00:9b:d5:77 > 52:54:00:9b:d5:11, ethertype IPv4 (0x0800), length 102: 192.168.100.102.55628 > 192.168.100.100.4789: VXLAN, f
lags [I] (0x08), vni 0
32:dd:d2:46:eb:82 > 01:80:c2:00:00:00, 802.3, length 52: LLC, dsap STP (0x42) Individual, ssap STP (0x42) Command, ctrl 0x03: STP 802.1d, Config,
Flags [none], bridge-id 8000.2a:18:ca:77:13:4d.8001, length 35
13:24:58.315497 52:54:00:9b:d5:77 > 52:54:00:9b:d5:33, ethertype IPv4 (0x0800), length 134: 192.168.100.102 > 192.168.100.101: ESP(spi=0x0be025cf,
seq=0x479), length 100
```

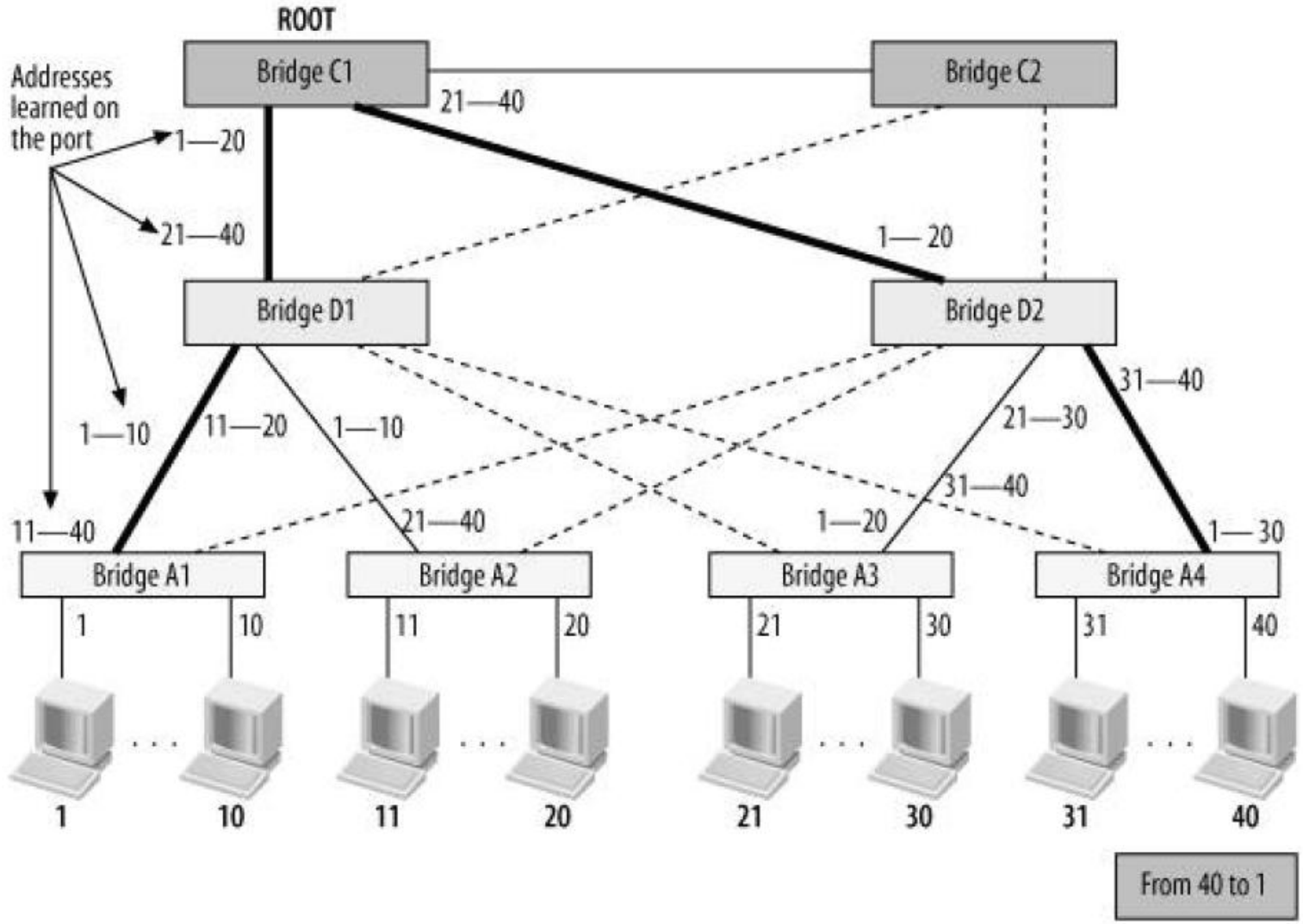
Openvswitch: STP

- The Spanning Tree Protocol (STP)

Figure 14-8. Bridging loop



Openvswitch: STP

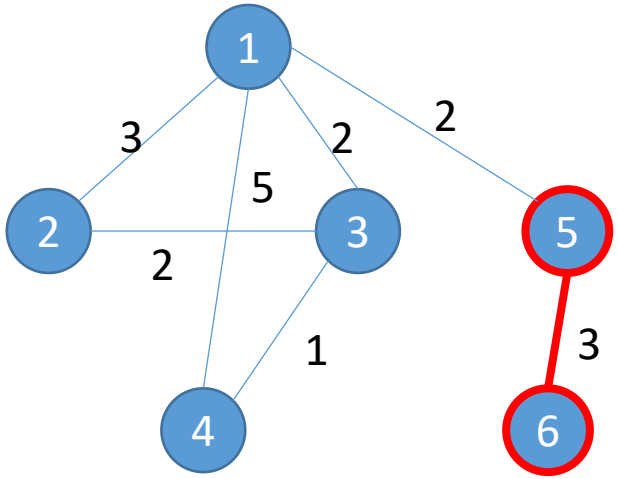


Openvswitch: STP

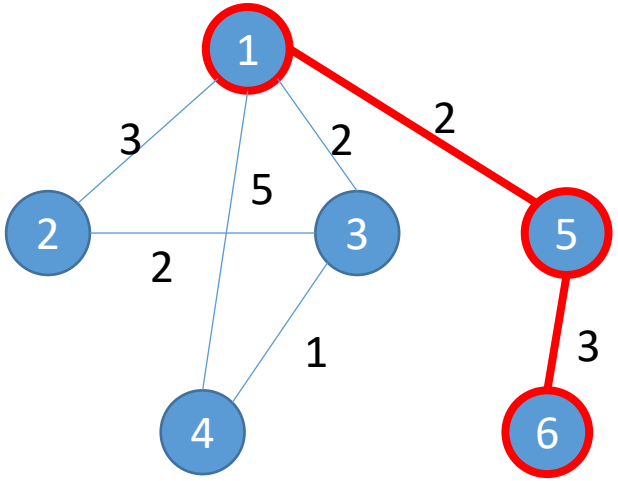
- Root Bridge: 树根，整棵STP树的老大，是掌门，最大的大哥
- Designated Bridges: 我拜谁做大哥
- Bridge Protocol Data Units (BPDUs) 相互比较实力的协议
  - STP (802.1D-1998)
  - RSTP (802.1D-2002 or 802.1w)
  - MSTP (802.1Q-2002 or 802.1s)
- Priority Vector 实力 (值越小越牛)
  - [Root Bridge ID, Root Path Cost, Bridge ID, and Port ID]
  - Root Bridge ID我当前的老大的ID，先拿掌门比
  - Root Path Cost我距离我当前的老大的距离，再拿和掌门关系比
  - Bridge ID我自己的ID，再拿自己的本事比

Openvswitch: STP

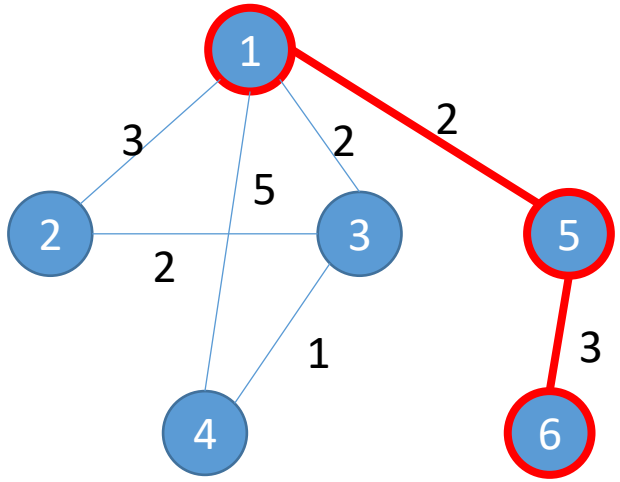
- STP协商过程，选举武林盟主
  - BPDU只有掌门发，已经隶属于某个掌门的bridge不发，仅仅传达掌门指示
  - 开始启动的时候，都认为自己是掌门，向周围展现实力
  - 掌门与掌门相遇：
    - 赢得当掌门，输的当小弟，小弟的小弟也跟随新的掌门
  - 掌门与自己小弟相遇：已经顺从，无需处理
  - 掌门与其他帮派小弟相遇：
    - 小弟拿本帮掌门和这个掌门比较，赢了，这个掌门拜入门来
    - 输了，会拜入新掌门，并且逐渐的将与自己连接的兄弟弃暗投明，最终原来的掌门也拜入门
  - 同门小弟相遇，比较和掌门的关系，近的当大哥
  - 不同门小弟相遇
    - 各自拿掌门比较，输了的拜入赢得掌门门派，并且逐渐的将与自己连接的兄弟弃暗投明，最终原来的掌门也拜入门


1
2
3
4
5
6
3
2
2
5
2
1
3

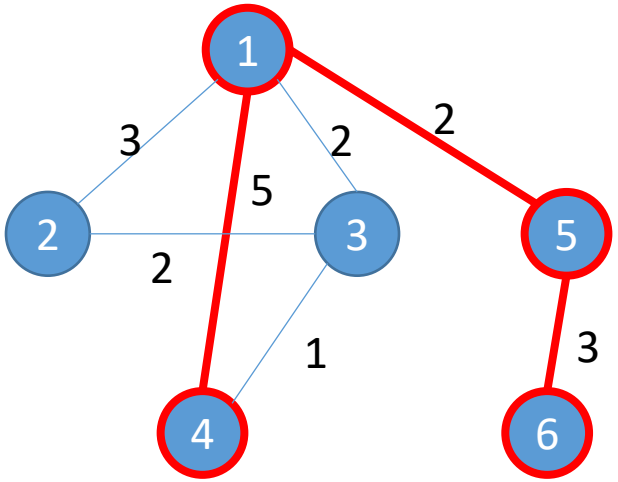
(1)


1
2
3
4
5
6
3
2
2
5
2
1
3

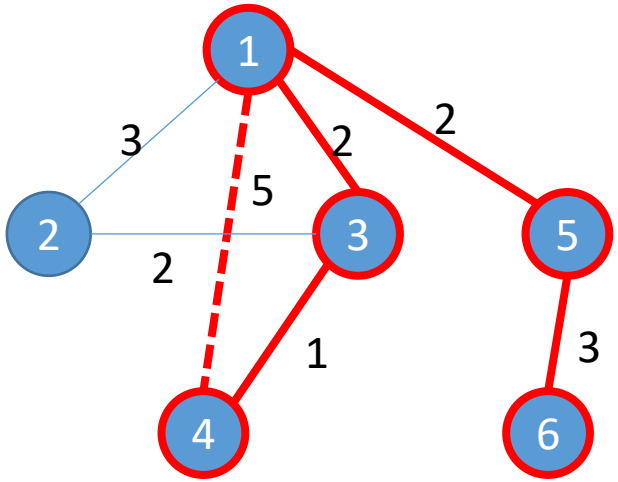
(2)


1
2
3
4
5
6
3
2
2
5
2
1
3

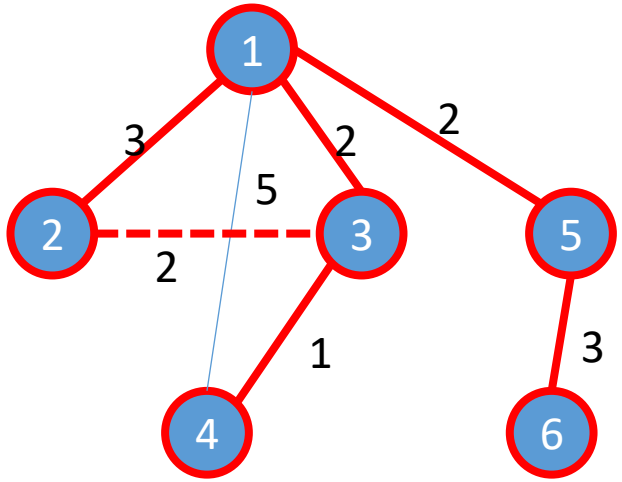
(3)


1
2
3
4
5
6
3
2
2
5
2
1
3

(4)


1
2
3
4
5
6
3
2
2
5
2
1
3

(5)


1
2
3
4
5
6
3
2
2
5
2
1
3

(6)

实验十二：测试STP

- 在Instance01上

```
root@openstackcliu8:/home/openstack# ovs-vsctl list Bridge
_uuid               : 38bc086a-aefb-439c-b75d-1225213215d8
controller          : []
datapath_id         : "00006a08bc389c43"
datapath_type       : ""
external_ids        : {}
fail_mode           : []
flood_vlans         : []
flow_tables         : {}
ipfix               : []
mirrors             : []
name                : testbr
netflow             : []
other_config        : {}
ports               : [c7956cb9-5a7c-494b-a8c4-60d83d96c91a, ccb953a2-443c-4c51-b735-df1e8a75ff02, d664f918-6cbc-42e5-bd69-28a41b29ed94]
protocols           : []
sflow               : []
status              : {stp_bridge_id="8000.6a08bc389c43", stp_designated_root="8000.2a18ca77134d", stp_root_path_cost="19"}
stp_enable          : true
```

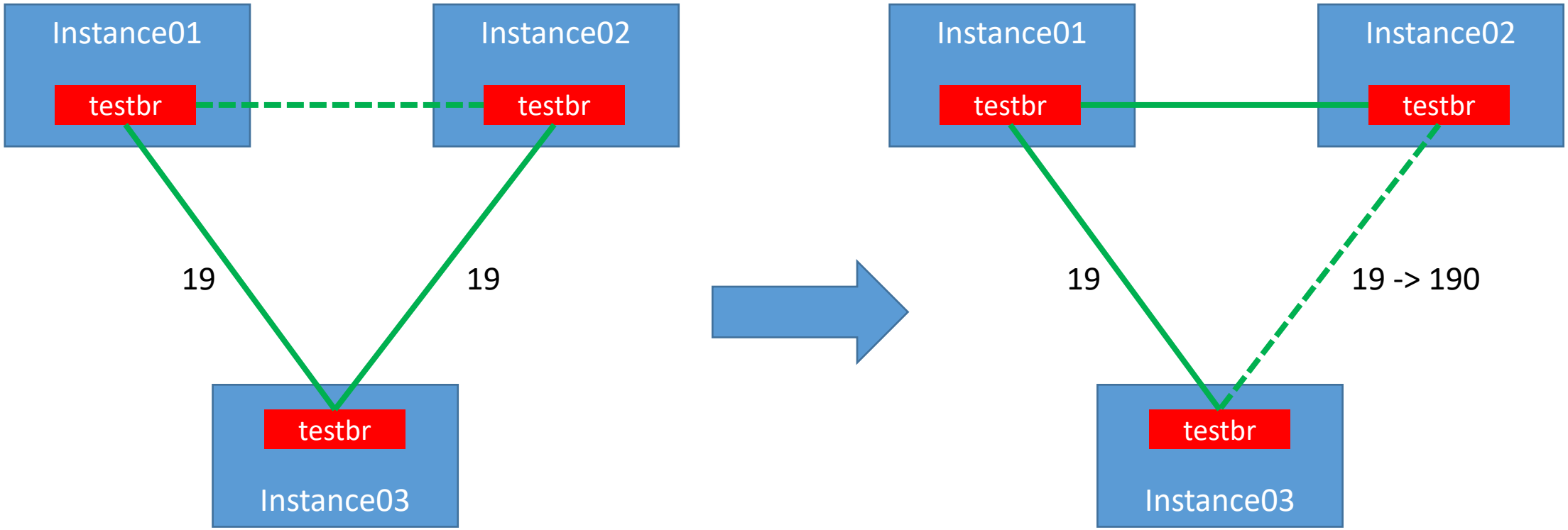
- 在Instance02上

```
status              : {stp_bridge_id="8000.8231a80af44a", stp_designated_root="8000.2a18ca77134d", stp_root_path_cost="19"}
stp_enable          : true
```

- 在Instance03上

```
status              : {stp_bridge_id="8000.2a18ca77134d", stp_designated_root="8000.2a18ca77134d", stp_root_path_cost="0"}
stp_enable          : true
```

实验十二：测试STP



验证：从Instance01 ping Instance02，需要经过Instance03

```
16:43:19.219046 52:54:00:9b:d5:11 > 52:54:00:9b:d5:77, ethertype IPv4 (0x0800), length 148: 192.168.100.100.42907 > 192.168.100.102.4789: VXLAN, f
lags [I] (0x08), vni 0
6a:08:bc:38:9c:43 > 82:31:a8:0a:f4:4a, ethertype IPv4 (0x0800), length 98: 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1739, seq 6, length 64
```

从Instance01 ping Instance03，不需要经过Instance02

实验十二：测试STP

在Instance02和Instance3上运行
ovs-vsctl set Port ipsec0 other_config:stp-path-cost=190

Instance01

```
root@openstackcliu8:/home/openstack# ovs-vsctl list Bridge
_uuid               : 38bc086a-aefb-439c-b75d-1225213215d8
controller          : []
datapath_id         : "00006a08bc389c43"
datapath_type       : ""
external_ids        : {}
fail_mode           : []
flood_vlans         : []
flow_tables         : {}
ipfix               : []
mirrors             : []
name                : testbr
netflow             : []
other_config        : {}
ports               : [c7956cb9-5a7c-494b-a8c4-60d83d96c91a, ccb953a2-443c-4c51-b735-df1e8a75ff02, d664f918-6cbc-42e5-bd69-28a41b29ed94]
protocols           : []
sflow               : []
status              : {stp_bridge_id="8000.6a08bc389c43", stp_designated_root="8000.2a18ca77134d", stp_root_path_cost="19"}
stp_enable          : true
```

Instance02

```
status              : {stp_bridge_id="8000.8231a80af44a", stp_designated_root="8000.2a18ca77134d", stp_root_path_cost="38"}
stp_enable          : true
```

Instance03

```
status              : {stp_bridge_id="8000.2a18ca77134d", stp_designated_root="8000.2a18ca77134d", stp_root_path_cost="0"}
stp_enable          : true
```

Openvswitch: 直接操作DB

- 尽管ovs-vsctl提供了很多命令直接操作Bridge, Port, Interface, Controller, Manager, SSL等，还是有很多的属性，没有直接命令去操作，需要通过修改和查看数据库来进行
- 数据库的schema可以通过ovsdb-client查看
- 修改数据库内容通过ovs-vsctl来进行

实验十三：查看DB的schema

```
root@openstackcliu8:/home/openstack# ovsdb-client list-dbs
Open_vSwitch
root@openstackcliu8:/home/openstack# ovsdb-client list-tables Open_vSwitch
Table
-------------------------
Port
Manager
Bridge
Interface
SSL
IPFIX
Open_vSwitch
Queue
NetFlow
Mirror
QoS
Controller
Flow_Table
sFlow
Flow_Sample_Collector_Set
root@openstackcliu8:/home/openstack# ovsdb-client list-columns Open_vSwitch Port
Column          Type
--------------- ------------------------------------------------------------------------------------------
name            "string"
statistics      {"key":"string","max":"unlimited","min":0,"value":"integer"}
vlan_mode       {"key":{"enum":["set",["access","native-tagged","native-untagged","trunk"]],"type":"string"},"min":0}
qos             {"key":{"refTable":"QoS","type":"uuid"},"min":0}
_uuid           "uuid"
trunks          {"key":{"maxInteger":4095,"minInteger":0,"type":"integer"},"max":4096,"min":0}
mac             {"key":"string","min":0}
status          {"key":"string","max":"unlimited","min":0,"value":"string"}
interfaces      {"key":{"refTable":"Interface","type":"uuid"},"max":"unlimited"}
bond_downdelay  "integer"
_version        "uuid"
bond_mode       {"key":{"enum":["set",["active-backup","balance-slb","balance-tcp"]],"type":"string"},"min":0}
bond_updelay    "integer"
external_ids    {"key":"string","max":"unlimited","min":0,"value":"string"}
other_config    {"key":"string","max":"unlimited","min":0,"value":"string"}
bond_fake_iface "boolean"
tag             {"key":{"maxInteger":4095,"minInteger":0,"type":"integer"},"min":0}
```

实验十三：查看DB的schema

- ovsdb-client get-schema Open_vSwitch
- ovsdb-client dump

实验十四：查看和修改DB的内容

- 准备环境
  - ovs-vsctl add-br helloworld
  - ip link add first_br type veth peer name first_if
  - ip link set first_br up
  - ip link set first_if up
  - ovs-vsctl add-port helloworld first_br
- list table
  - ovs-vsctl list Bridge
- list table [record]
  - ovs-vsctl list Bridge helloworld
- get table record [column[:key]]
  - ovs-vsctl get Bridge helloworld ports
- set table record column[:key]=value
  - ovs-vsctl set Port first_br tag=100
- add table record column [key=]value
  - ovs-vsctl add Port first_br trunks 110

实验十四：查看和修改DB的内容

- remove table record column key
  - ovs-vsctl remove Port first_br trunks 100
- clear table record column
  - ovs-vsctl clear Port first_br trunks

Openvswitch: Flow Table

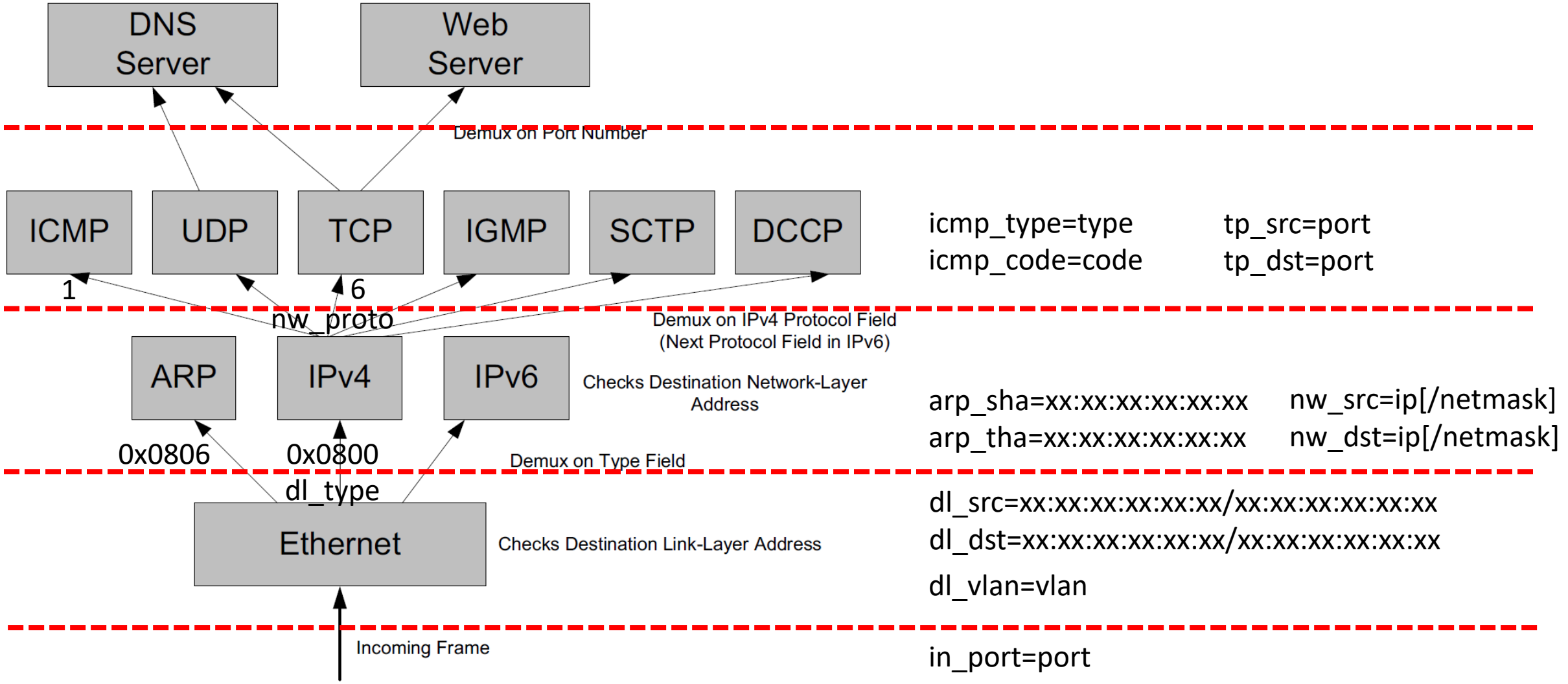
- 对于Flow Table的管理，由ovs-ofctl来控制
  - **add–flow** *switch flow*
  - **mod–flows** *switch flow*
  - **del–flows** *switch* [*flow*]

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Table 1: Main components of a flow entry in a flow table.

- Match Field
- Actions

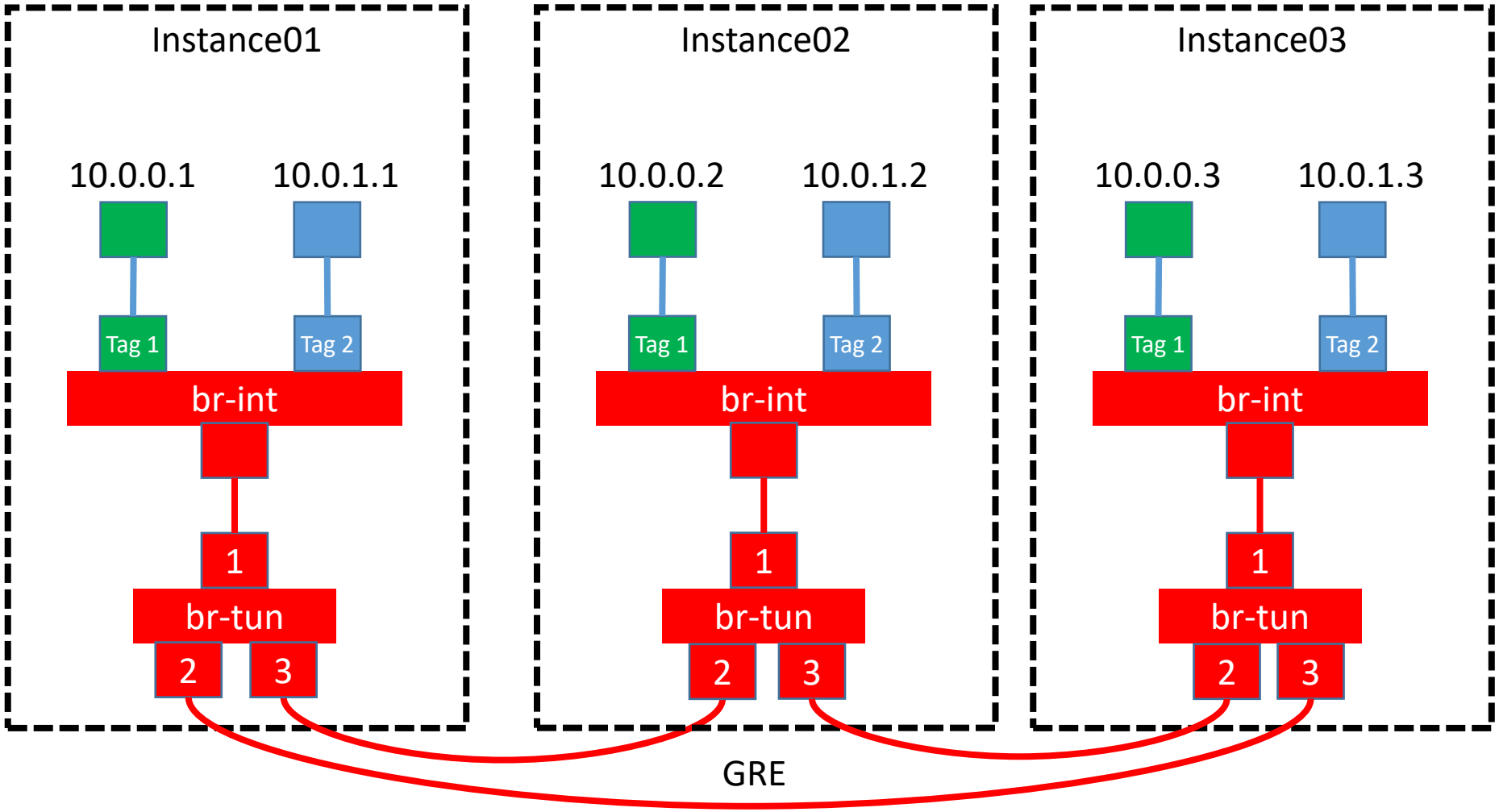
Openvswitch: Flow Table



Openvswitch: Flow Table

- Actions:
  - output:port 和 output:NXM_NX_REG0[16..31]
  - enqueue:port:queue
  - mod_vlan_vid:vlan_vid
  - strip_vlan
  - mod_dl_src:mac 和 mod_dl_dst:mac
  - mod_nw_src:ip 和 mod_nw_dst:ip
  - mod_tp_src:port 和 mod_tp_dst:port
  - set_tunnel:id
  - resubmit([port],[table])
  - move:src[start..end]−>dst[start..end]
  - load:value−>dst[start..end]
  - learn(argument[,argument]...)

实验十五：模拟Openstack中Flow的操作



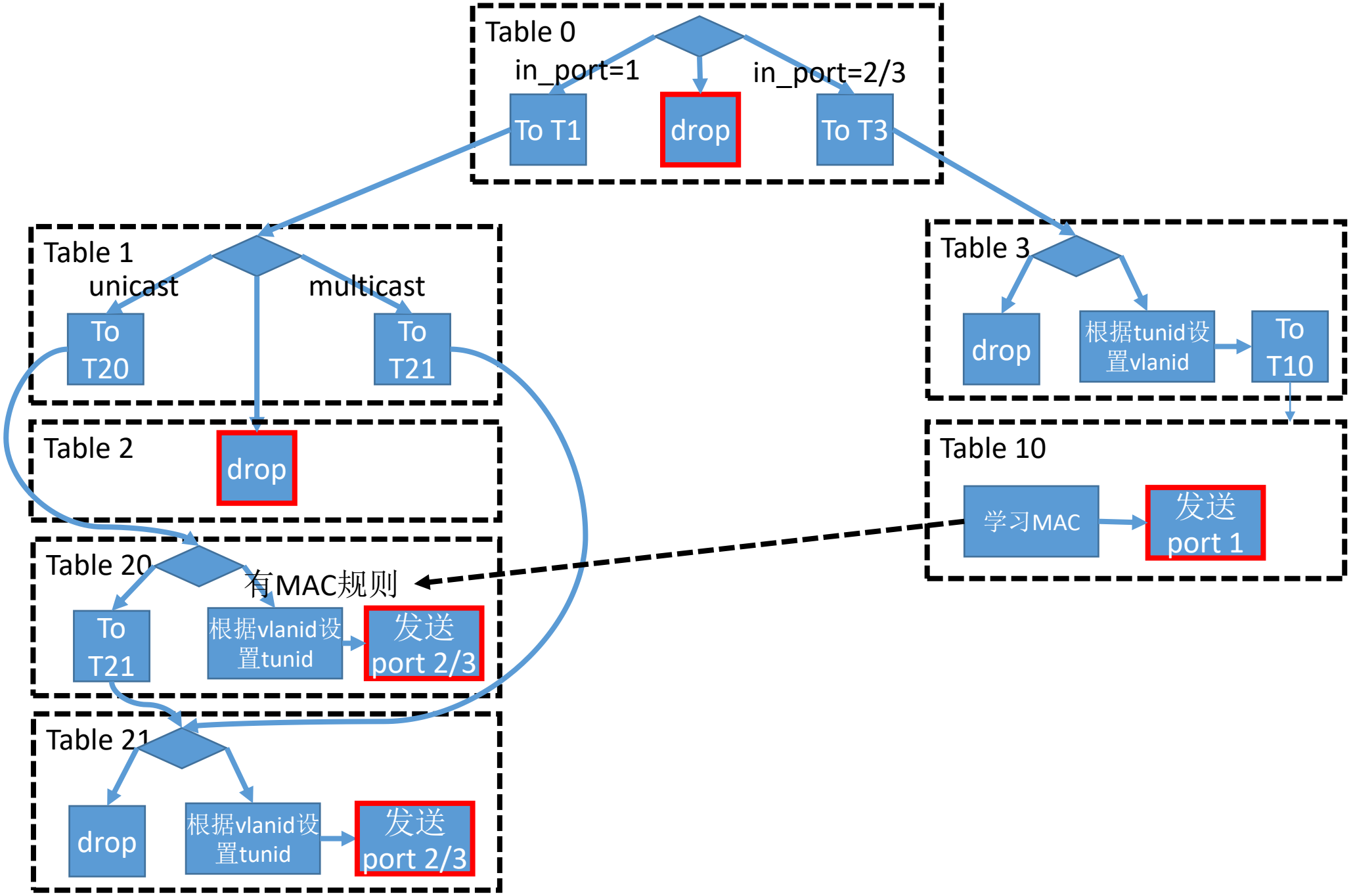


Table 0
in_port=1
in_port=2/3
To T1
drop
To T3
Table 1
unicast
multicast
To T20
To T21
Table 2
drop
Table 3
drop
根据tunid设置vlanid
To T10
Table 10
学习MAC
发送 port 1
Table 20
有MAC规则
To T21
根据vlanid设置tunid
发送 port 2/3
Table 21
drop
根据vlanid设置tunid
发送 port 2/3

实验十五：模拟Openstack中Flow的操作

- 配置拓扑结构：在三台机器上都运行

ovs-vsctl add-br br-int
ovs-vsctl add-br br-tun
ip link add br-int-pair type veth peer name br-tun-pair
ip link set br-int-pair up
ip link set br-tun-pair up
ovs-vsctl add-port br-int br-int-pair
ovs-vsctl add-port br-tun br-tun-pair
ip link add vnic0 type veth peer name vnic0-br-int
ip link set vnic0 up
ip link set vnic0-br-int up
ovs-vsctl add-port br-int vnic0-br-int
ifconfig vnic0 10.0.0.1/24
ip link add vnic1 type veth peer name vnic1-br-int
ip link set vnic1 up
ip link set vnic1-br-int up
ovs-vsctl add-port br-int vnic1-br-int
ifconfig vnic1 10.0.1.1/24
ovs-vsctl set Port vnic0-br-int tag=1
ovs-vsctl set Port vnic1-br-int tag=2
ovs-vsctl add-port br-tun gre0 -- set Interface gre0 type=gre options:local_ip=192.168.100.100 options:in_key=flow options:remote_ip=192.168.100.101 options:out_key=flow
ovs-vsctl add-port br-tun gre1 -- set Interface gre1 type=gre options:local_ip=192.168.100.100 options:in_key=flow options:remote_ip=192.168.100.102 options:out_key=flow

```
root@openstackcliu8:/home/openstack# ovs-ofctl show br-tun
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000928480448d40
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS AR
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SR
 1(br-tun-pair): addr:2e:24:2e:d9:bd:10
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(gre0): addr:d6:b3:a6:0f:19:d8
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 3(gre1): addr:46:dd:1e:10:3d:6b
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 LOCAL(br-tun): addr:92:84:80:44:8d:40
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

实验十五：模拟Openstack中Flow的操作

- 配置Flow
  - 删除所有的Flow

    ovs-ofctl del-flows br-tun
  - Table 0
    - 从port 1进来的，由table 1处理

      ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=1 in_port=1 actions=resubmit(,1)"
    - 从port 2/3进来的，由Table 3处理

      ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=1 in_port=2 actions=resubmit(,3)"
      ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=1 in_port=3 actions=resubmit(,3)"
    - 默认丢弃

      ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=0 actions=drop"

实验十五：模拟Openstack中Flow的操作

- Table 1:
  - 对于单播，由table 20处理

ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=1 table=1 dl_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,20)"

  - 对于多播，由table 21处理

ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=1 table=1 dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,21)"

- Table 2:
  - 默认丢弃

ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=0 table=2 actions=drop"

实验十五：模拟Openstack中Flow的操作

- Table 3:
  - 默认丢弃

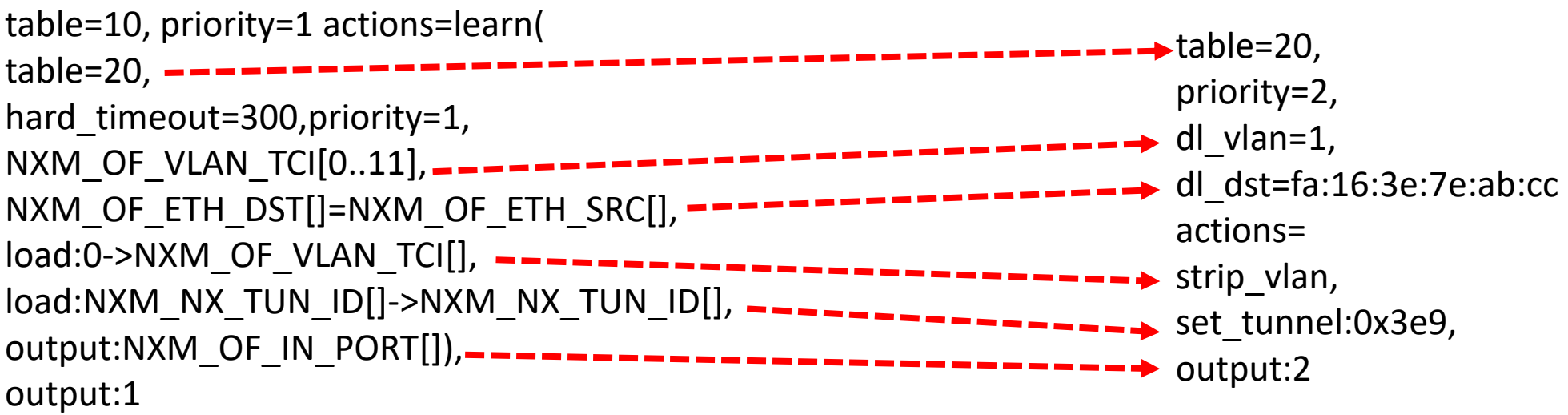
```
ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=0 table=3 actions=drop"
```

  - Tunnel ID -> VLAN ID

```
ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=1 table=3 tun_id=0x1
actions=mod_vlan_vid:1,resubmit(,10)"
ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=1 table=3 tun_id=0x2
actions=mod_vlan_vid:2,resubmit(,10)"
```

- Table 10:
  - MAC地址学习

```
ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=1 table=10
actions=learn(table=20,priority=1,hard_timeout=300,NXM_OF_VLAN_TCI[0..11],NXM_OF_ETH_DST[]=NXM
_OF_ETH_SRC[],load:0->NXM_OF_VLAN_TCI[],load:NXM_NX_TUN_ID[]-
>NXM_NX_TUN_ID[],output:NXM_OF_IN_PORT[]),output:1"
```



- Table 10是用来学习MAC地址的，学习的结果放在Table 20里面，Table20被称为MAC learning table
- NXM_OF_VLAN_TCI这个是VLAN Tag，在MAC Learning table中，每一个entry都是仅仅对某一个VLAN来说的，不同VLAN的learning table是分开的。在学习的结果的entry中，会标出这个entry是对于哪个VLAN的。
- NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[]这个的意思是当前包里面的MAC Source Address会被放在学习结果的entry里面的dl_dst里面。这是因为每个switch都是通过Ingress包来学习，某个MAC从某个port进来，switch就应该记住以后发往这个MAC的包要从这个port出去，因而MAC source address就被放在了Mac destination address里面，因为这是为发送用的。
- load:0->NXM_OF_VLAN_TCI[]意思是发送出去的时候，vlan tag设为0，所以结果中有actions=strip_vlan
- load:NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[]意思是发出去的时候，设置tunnel id，进来的时候是多少，发送的时候就是多少，所以结果中有set_tunnel:0x3e9
- output:NXM_OF_IN_PORT[]意思是发送给哪个port，由于是从port2进来的，因而结果中有output:2

实验十五：模拟Openstack中Flow的操作

- Table 20:
  - 这个是MAC Address Learning Table，如果不空就按照规则处理
  - 如果为空，就使用默认规则，交给Table 21处理

```
ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=0 table=20 actions=resubmit(,21)"
```

- Table 21:
  - 默认丢弃

```
ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=0 table=21 actions=drop"
```

  - VLAN ID -> Tunnel ID

```
ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=1 table=21 dl_vlan=1
actions=strip_vlan,set_tunnel:0x1,output:2,output:3"
ovs-ofctl add-flow br-tun "hard_timeout=0 idle_timeout=0 priority=1 table=21 dl_vlan=2
actions=strip_vlan,set_tunnel:0x2,output:2,output:3"
```

实验十五：模拟Openstack中Flow的操作

- 从10.0.0.1 ping 10.0.0.2
  - 在Instance01上监听br-int上的端口

```
root@openstackcliu8:/home/openstack# tcpdump -n -e -i br-int-pair
tcpdump: WARNING: br-int-pair: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on br-int-pair, link-type EN10MB (Ethernet), capture size 65535 bytes
11:17:14.597292 7e:83:60:7d:1c:8f > 4a:57:54:66:fc:2d, ethertype 802.1Q (0x8100), length 102: vlan 1, p 0, ethertype IPv4, 10.0.0.1 > 10.0.0.2: IC
MP echo request, id 2528, seq 103, length 64
```

  - 在Instance01上监听eth0

```
11:15:39.597315 52:54:00:9b:d5:11 > 52:54:00:9b:d5:33, ethertype IPv4 (0x0800), length 140: 192.168.100.100 > 192.168.100.101: GREv0, key=0x1, pro
to TEB (0x6558), length 106: 7e:83:60:7d:1c:8f > 4a:57:54:66:fc:2d, ethertype IPv4 (0x0800), length 98: 10.0.0.1 > 10.0.0.2: ICMP echo request, id
 2528, seq 8, length 64
```

  - 在Instance02上监听eth0

```
11:21:52.351091 52:54:00:9b:d5:11 > 52:54:00:9b:d5:33, ethertype IPv4 (0x0800), length 140: 192.168.100.100 > 192.168.100.101: GREv0, key=0x1, pro
to TEB (0x6558), length 106: 7e:83:60:7d:1c:8f > 4a:57:54:66:fc:2d, ethertype IPv4 (0x0800), length 98: 10.0.0.1 > 10.0.0.2: ICMP echo request, id
 2534, seq 6, length 64
```

  - 在Instance02上监听br-int上的端口

```
root@openstackcliu8:/home/openstack# tcpdump -n -e -i br-int-pair
tcpdump: WARNING: br-int-pair: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on br-int-pair, link-type EN10MB (Ethernet), capture size 65535 bytes
11:22:59.920447 7e:83:60:7d:1c:8f > 4a:57:54:66:fc:2d, ethertype 802.1Q (0x8100), length 102: vlan 1, p 0, ethertype IPv4, 10.0.0.1 > 10.0.0.2: IC
MP echo request, id 2535, seq 1, length 64
```

Openvswitch: 综合实例

- Open vSwitch Advanced Features Tutorial
  - http://git.openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=tutorial/Tutorial;hb=HEAD
- 创建一个Virtual Switch
- 包含下面四个Port：
  - P1, truck port
  - P2, VLAN 20
  - P3, P4 VLAN 30
- 包含五个flow table:
  - Table 0: Admission control.
  - Table 1: VLAN input processing.
  - Table 2: Learn source MAC and VLAN for ingress port.
  - Table 3: Look up learned port for destination MAC and VLAN.
  - Table 4: Output processing

Openvswitch: 综合实例

- 创建一个bridge

ovs-vsctl add-br helloworld

- 创建四个veth pair

ip link add first_br type veth peer name first_if
ip link add second_br type veth peer name second_if
ip link add third_br type veth peer name third_if
ip link add forth_br type veth peer name forth_if

- 添加四个端口port，ofport_request是指定端口号

ovs-vsctl add-port helloworld first_br -- set Interface first_br ofport_request=1
ovs-vsctl add-port helloworld second_br -- set Interface second_br ofport_request=2
ovs-vsctl add-port helloworld third_br -- set Interface third_br ofport_request=3
ovs-vsctl add-port helloworld forth_br -- set Interface forth_br ofport_request=4

Openvswitch: 综合实例

- 新添加的port都是出于DOWN的状态，设成up

```
ip link set first_if up
ip link set first_br up
ip link set second_br up
ip link set second_if up
ip link set third_if up
ip link set third_br up
ip link set forth_br up
ip link set forth_if up
```

Openvswitch: 综合实例

- 实现第一个Table 0，Admission control
  - 包进入vswitch的时候首先进入Table 0，我们在这里可以设定规则，控制那些包可以进入，那些包不可以进入。
  - multicast的不允许进入

ovs-ofctl add-flow helloworld "table=0, dl_src=01:00:00:00:00:00/01:00:00:00:00:00, actions=drop"

  - STP的也不接受

ovs-ofctl add-flow helloworld "table=0, dl_dst=01:80:c2:00:00:00/ff:ff:ff:ff:ff:f0, actions=drop"

  - 添加最后一个flow，这个flow的priority低于default，如果上面两个不匹配，则我们进入table 1

ovs-ofctl add-flow helloworld "table=0, priority=0, actions=resubmit(,1)"

- 测试Table 0
  - 不满足条件DROP

ovs-appctl ofproto/trace helloworld in_port=1,dl_dst=01:80:c2:00:00:05

  - 满足条件RESUBMIT

ovs-appctl ofproto/trace helloworld in_port=1,dl_dst=01:80:c2:00:00:10

Openvswitch: 综合实例

- 实现第二个Table 1：VLAN Input Processing
  - 添加一个最低优先级的DROP的规则

```
ovs-ofctl add-flow helloworld "table=1, priority=0, actions=drop"
```

  - 对于port 1，是trunk口，无论有没有VLAN Header都接受

```
ovs-ofctl add-flow helloworld "table=1, priority=99, in_port=1, actions=resubmit(,2)"
```

  - 对于port 2, 3, 4, 我们希望没有VLAN Tag，然后我们给打上VLAN Tag

```
ovs-ofctl add-flows helloworld - <<'EOF'
table=1, priority=99, in_port=2, vlan_tci=0, actions=mod_vlan_vid:20, resubmit(,2)
table=1, priority=99, in_port=3, vlan_tci=0, actions=mod_vlan_vid:30, resubmit(,2)
table=1, priority=99, in_port=4, vlan_tci=0, actions=mod_vlan_vid:30, resubmit(,2)
EOF
```

- 测试Table 1
  - 从port 1进入，tag为5 `ovs-appctl ofproto/trace helloworld in_port=1,vlan_tci=5`
  - 从port 2进入，没有打Tag的 `ovs-appctl ofproto/trace helloworld in_port=2`
  - 从port 2进入，带Tag 5的 `ovs-appctl ofproto/trace helloworld in_port=2,vlan_tci=5`

Openvswitch: 综合实例

- 实现第三个Table 2: MAC, VLAN learning for ingress port
  - 对于普通的switch，都会有这个学习的过程，当一个包到来的时候，由于包里面有MAC，VLAN Tag，以及从哪个口进来的这个信息。
  - 于是switch学习后，维护了一个表格port –> MAC –> VLAN Tag。
  - 这样以后如果有需要发给这个MAC的包，不用ARP，switch自然之道应该发给哪个port，应该打什么VLAN Tag。
  - OVS也要学习这个，并维护三个之间的mapping关系。

```
ovs-ofctl add-flow helloworld "table=2 actions=learn(table=10, NXM_OF_VLAN_TCI[0..11],
NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[], load:NXM_OF_IN_PORT[]->NXM_NX_REG0[0..15]), resubmit(,3)"
```

- learn表示这是一个学习的action
- table 10，这是一个MAC learning table，学习的结果会放在这个table中。
- NXM_OF_VLAN_TCI这个是VLAN Tag，在MAC Learning table中，每一个entry都是仅仅对某一个VLAN来说的，不同VLAN的learning table是分开的。在学习的结果的entry中，会标出这个entry是对于哪个VLAN的。
- NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[]这个的意思是当前包里面的MAC Source Address会被放在学习结果的entry里面的dl_dst里面。这是因为每个switch都是通过Ingress包来学习，某个MAC从某个port进来，switch就应该记住以后发往这个MAC的包要从这个port出去，因而MAC source address就被放在了Mac destination address里面，因为这是为发送用的。
- NXM_OF_IN_PORT[]->NXM_NX_REG0将portf放入register.
- 一般对于学习的entry还需要有hard_timeout，这是的每个学习结果都会expire，需要重新学习。

Openvswitch: 综合实例

- 测试Table 2:
  - 从port 1来一个vlan为20的mac为50:00:00:00:00:01的包

ovs-appctl ofproto/trace helloworld in_port=1,vlan_tci=20,dl_src=50:00:00:00:00:01 -generate

ovs-ofctl dump-flows helloworld

table 10多了一条，vlan为20，dl_dst为50:00:00:00:00:01，发送的时候从port 1出去

  - 从port 2进来，被打上了vlan 20，mac为50:00:00:00:00:02

ovs-appctl ofproto/trace helloworld in_port=2,dl_src=50:00:00:00:00:02 -generate

Openvswitch: 综合实例

- 实现第四个table 3: Look Up Destination Port
  - 在table 2中，vswtich通过进入的包，学习了vlanid –> mac –> port的映射后，对于要发送的包，可以根据学习到的table 10里面的内容，根据destination mac和vlan，来找到相应的port发送出去，而不用每次都flood
  - ovs-ofctl add-flow helloworld "table=3 priority=50 actions=resubmit(,10), resubmit(,4)"
  - 添加这条规则，首先到table 10中查找learn table entry，如果找不到则到table 4
  - 如果包本身就是multicast的或者broadcast的，则不用去table 10里面取查找
  - ovs-ofctl add-flow helloworld "table=3 priority=99 dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,4)"

Openvswitch: 综合实例

- 测试Table 3:
  - ovs-appctl ofproto/trace helloworld in_port=1,dl_vlan=20,dl_src=f0:00:00:00:00:01,dl_dst=90:00:00:00:00:01 -generate
  - 由于目标地址f0:00:00:00:00:01没有在table 10中找到，因而到达table 4.
  - 但是这次测试使得table 10中学习到了mac地址90:00:00:00:00:01
  - ovs-appctl ofproto/trace helloworld in_port=2,dl_src=90:00:00:00:00:01,dl_dst=f0:00:00:00:00:01 -generate
  - 因为刚才学习到了mac地址f0:00:00:00:00:01，所以这次在table 10中找到了这条记录，这次同时也学习到了mac地址90:00:00:00:00:01
  - 再发送第一次的包
  - ovs-appctl ofproto/trace helloworld in_port=1,dl_vlan=20,dl_src=f0:00:00:00:00:01,dl_dst=90:00:00:00:00:01 -generate
  - 也在table 10中找到了记录

Openvswitch: 综合实例

- 实现第五个table 4: Output Processing
  - 这个时候，register 0中包含了output port，如果是0则说明是flood
  - 对于port 1来讲，是trunk port，所以携带的vlan tag就让他带着，从port 1出去
  - ovs-ofctl add-flow helloworld "table=4 reg0=1 actions=1"
  - 对于port 2来讲，是vlan 20的，然而出去的时候，vlan tag会被抹掉，从port 2发出去
  - 对于port 3， 4来讲，是vlan 30的，然而出去的时候，vlan tag会被抹掉，从port 3, 4出去

```
ovs-ofctl add-flows helloworld - <<'EOF'
    table=4 reg0=2 actions=strip_vlan,2
    table=4 reg0=3 actions=strip_vlan,3
    table=4 reg0=4 actions=strip_vlan,4
EOF
```

  - 对于broadcast来讲，我们希望一个vlan的broadcast仅仅在这个vlan里面发送，不影响其他的vlan

```
ovs-ofctl add-flows helloworld - <<'EOF'
    table=4 reg0=0 priority=99 dl_vlan=20 actions=1,strip_vlan,2
    table=4 reg0=0 priority=99 dl_vlan=30 actions=1,strip_vlan,3,4
    table=4 reg0=0 priority=50 actions=1
EOF
```

  - 所以对于register = 0的，也即是broadcast的，属于vlan 20的，则从port 1, 2出去，属于vlan 30的，则从port 1, 3, 4出去。

Openvswitch: 综合实例

- 测试Table 4
  - 如果是一个port 1来的vlan 30的broadcast
  - ovs-appctl ofproto/trace helloworld in_port=1,dl_dst=ff:ff:ff:ff:ff:ff,dl_vlan=30
  - 结果是port 1就不发送了，发送给了port 3, 4
  - ovs-appctl ofproto/trace helloworld in_port=3,dl_dst=ff:ff:ff:ff:ff:ff
  - 接着测试mac learning
  - ovs-appctl ofproto/trace helloworld in_port=1,dl_vlan=30,dl_src=10:00:00:00:00:01,dl_dst=20:00:00:00:00:01 -generate
  - 由于这两个地址没有出现过，则除了进行学习以外，广播发送给port 3，4
  - ovs-appctl ofproto/trace helloworld in_port=4,dl_src=20:00:00:00:00:01,dl_dst=10:00:00:00:00:01 -generate
  - 回复的时候，由于学习过了，则仅仅从port 1发送出去
  - ovs-appctl ofproto/trace helloworld in_port=1,dl_vlan=30,dl_src=10:00:00:00:00:01,dl_dst=20:00:00:00:00:01 -generate
  - 由于在回复中进行了学习，因而发送的时候，仅仅发送port 4

Openvswitch: 最后的惊喜