实验 2、Open vSwitch 交换机的安装与配置

电子科技大学信通学院　　杨宁

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

实验原理

Open vSwitch 是一个开源软件，简称为 OVS，是由软件实现的多层虚拟交换机，使用开源 Apache2.0 许可协议。OVS 的主要目的是通过编程扩展来实现大规模网络自动化，同时仍然支持标准的管理接口和协议：NetFlow、sFlow、VLAN、STP 等），并且还支持跨多个物理机的分布式环境。

OVS 非常适合在 VM 环境中充当虚拟交换机，支持多种基于 Linux 的虚拟化技术，包括 Xen/XenServer、KVM 和 VirtualBox。图 1 是使用 OVS 进行 SDN 组网的拓扑示例。OVS 的工作原理与物理交换机类似，主要有两个作用：

- 传递虚拟机之间的流量；
- 实现虚拟机和外界网络的通信。

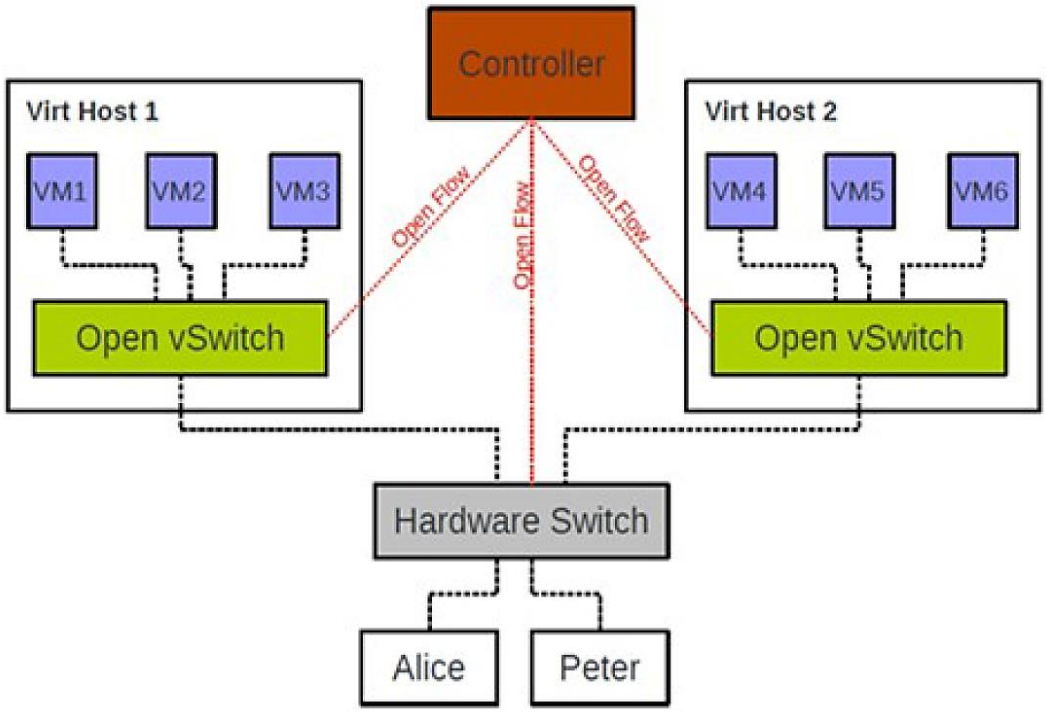


图 1 OVS 组网拓扑示例

1、内部架构

OVS 的目标是多服务器虚拟化部署，内部架构如图 2 所示。OVS 包含以下三个核心模块：

- openvswitch：OVS 的内核模块，实现了多个 datapath，主要负责与底层物理设备进行数据交换，如果在内核的缓存中找到转发规则即转发数据，否则将数据发给用户空间的 ovs-vswitchd 模块去处理。
- ovs-vswitchd：OVS 的核心模块，是一个守护进程。ovs-vswitchd 通过 netlink

与下层的内核模块一起实现基于流的数据交换，可以支持多个独立的 datapath。通过 OVSDB 协议与 ovsdb-server 模块通信，将交换机的配置、数据流信息及其他变化保存到 OVS 数据库（ovsdb）中。通过 OpenFlow 协议与 SDN 控制器通信。

- ovsdb-sever：OVS 的轻量级数据库服务器。ovs-vswitchd 可以通过 ovsdb-sever 来查询和获取虚拟交换机的配置等信息。OVS 数据库（ovsdb）其实只是一个 JSON 文件，本实验中默认安装在/usr/local/etc/openvswitch 目录中，文件名为 conf.db。

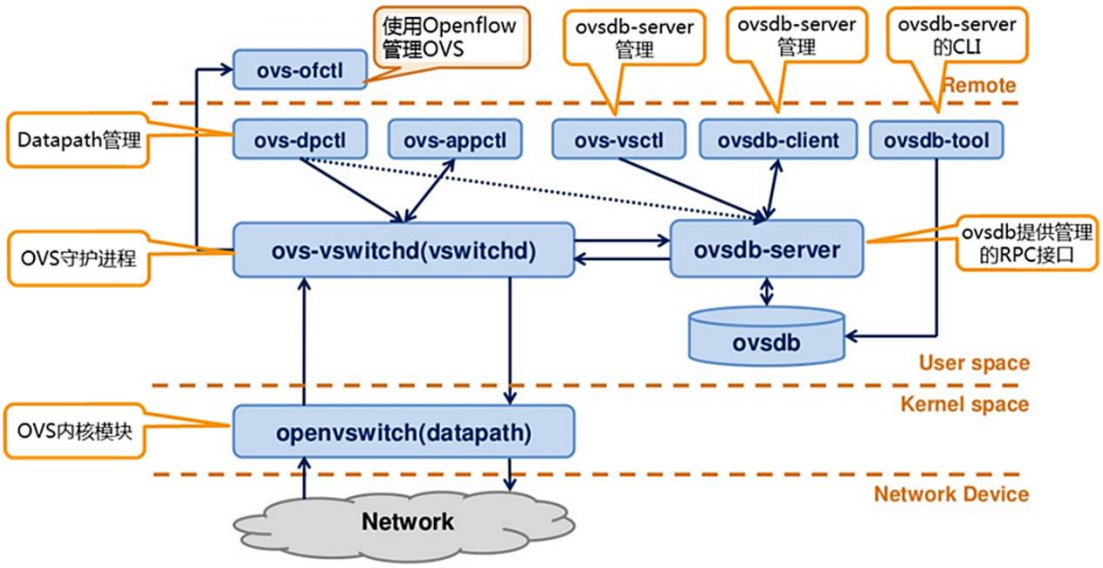


图 2 OVS 内部结构

此外，OVS 还包含一组重要的管理工具模块：

- ovs-dpctl：用来配置管理 datapath 的工具，可以控制转发规则，常用于调试。
- ovs-appctl：用来向 ovs-vswitchd 发送命令的工具，常用于调试。
- ovs-vsctl：用来查询和更改 ovs-vswitchd 配置的实用工具，操作时会更新 ovsdb。
- ovsdb-client：用于管理 ovsdb-server 的工具，侧重于查询。
- ovsdb-tool：管理 OVS 数据库的 CLI 工具，侧重于新建 ovsdb。
- ovs-ofctl：使用 OpenFlow 协议管理 OVS 的工具，主要用于 OVS 流表的查询和控制。

2、工作原理

（1）Port 和 Interface

OVS 内核模块实现的每个 datapath 就是一个虚拟网桥（Bridge），可以有多个

用于设备连接和数据交换的虚拟端口（Port）。Interface 是连接到 Port 的网络接口设备，即安装 OVS 的主机系统（HostOS）的网卡。通常情况下，Port 和 Interface 是一对一关系。

OVS 的 Port 有以下四种类型：

- Normal：将 HostOS 中已有的网卡挂载到 OVS 的网桥上时，OVS 会在网桥上生成一个与该网卡同名的 Normal 端口处理进出这个网卡的分组。Normal 端口是一个以混杂模式工作的二层口，因此对应的网卡不需要配置 IP 地址；已配置的 IP 地址将无效。
- Internal：当在 OVS 中创建一个新网桥时，默认会创建一个与该网桥同名的 Internal 端口。每创建一个 Internal 类型的 Port，OVS 就会在 HostOS 上自动创建一个同名的 Internal 类型的虚拟网卡。只有 Internal 类型的网卡才可以配置 IP 地址，可以用作 OVS 网桥的管理接口。
- Patch：当主机中有多个 datapath 网桥时，可以使用 Patch 类型的 Port 两两个网桥连接起来，使两个网桥逻辑上成为一体，Patch 类型的 Port 通常成对出现；
- Tunnel：添加隧道端口，又分为 vxlan 和 gre 两种。

（2）分组转发流程

OVS 内核模块中的 datapath 从一个 Port 收到一个分组后，首先会根据分组中的字段查询内核模块的流缓存（flow cache）中是否已有处理这个分组的规则缓存。如果存在匹配，就执行对应的操作。否则 datapath 就将该分组送到用户空间的 ovs-vswitchd。

ovs-vswitchd 根据保存在用户空间的流表（flow table）对该分组进行匹配。如果 ovs-vswitchd 匹配成功，则将对应流表项中的执行动作（action）通告给 datapath，并自动生成一条流表项注入到内核模块的流缓存中。datapath 收到的后续同类型分组都会因为匹配到流缓存中的流表项而直接处理。

如果 ovs-vswitchd 匹配不成功，则通过 OpenFlow 协议将该分组发送给 SDN 控制器，由控制器决定如何处理。控制器决定处理规则后，生成流表项发送给 ovs-vswitchd。ovs-vswitchd 将来自控制器的流表规则存入用户态流表中，并更新内核态的流缓存，然后将该分组重新注入到 datapath。

datapath 再次查询内核模块的流缓存（flow cache）找到该分组的匹配项，并执行对应的操作。

从上述流程中，可以看到 datapath 和 ovs-vswitchd 的相互配合中包含了两种分组处理方式：

- Fast Path：datapatch 加载到内核后，会在网卡上注册一个钩子函数，每当有分组到达网卡时，这个函数就会被调用，将分组进行层层解封（MAC

层、IP 层，TCP 层等），然后与内核中的流缓存表项匹配，如果找到匹配的流表项则根据既定策略来处理分组（例如从指定网卡转发、丢弃等）。这个处理过程全在内核完成，所以非常快，称之为 Fast Path。

- Slow Path：内核态没有被分配太多内存，所以内核态的流缓存中能够保存的流表项很少，往往有新的流表项到来后，老的流表项就被丢弃。而用户态的 ovs-vswitchd 被分配了大量内存，所以包含了所有流表项，这些流表项可以是 OpenFlow 控制器通过 OpenFlow 协议下发的，也可以是 OVS 命令行工具 ovs-ofctl 设定的。因此如果 datapath 在内核态的流缓存中找不到匹配的流表项，就会将分组通过 netlink（一种内核态与用户态交互的机制）发送给 ovs-vswitchd。ovs-vswitchd 有一个监听线程，当发现有从内核态发过来的分组，就进入自己的处理流程，即与用户空间的流表项匹配，然后再次将分组重新注入到 datapath。显然，在用户态处理是相对较慢的，故称之为 Slow Path。

当 ovs-vswitchd 最终匹配到了一个流表项之后，会根据“局部性原理（局部数据在一段时间都会被频繁访问，是缓存设计的基础原理）”，通过 netlink 协议将这条流表项下发到内核态。如果此时内核的内存空间不足，则会开始淘汰部分老的流表项。这样可以保证下一个相同类型的分组能够直接从内核匹配到，以此加快执行效率。由于近因效应，接下来的分组应该大概率能够匹配这条流表项的。例如：传输一个文件，同类型的分组会源源不断的到来。

（3）失败模式

OVS 在无法连接到控制器时可以选择两种失败（fail）状态：standalone 和 secure。

如果失败模式设置为 standalone 或没有设置失败模式（OVS 默认是没有设置失败模式），那么在三次探测控制器连接不成功后，OVS 会接管转发逻辑，作为一个 MAC 地址学习的传统二层交换机。但后台仍尝试连接控制器，一旦连接成功则退出 fail 状态。

如果将失败模式设置为 secure，那么 OVS 的初始流表是空的；否则会有一条执行 MAC 地址学习的传统二层交换机转发机制（actions=NORMAL）的流表项。在 secure 模式下，OVS 将不会自动配置新的转发流表，会严格按照原先有的流表转发。

（4）流表匹配过程

支持 OpenFlow 1.0 协议的 OVS，只有一张流表，编号为 0。支持 OpenFlow 1.3 协议的 OVS 允许有多张流表，但在仍然能从流表 0 开始进行查询匹配。一个流表中可以包含多条流表项，基于优先级进行匹配。管理员或控制器可以在一个流表中设置一条特殊流表项——table miss 流表项，该流表项匹配分组的所有字

段且优先级为 0，动作可能是 drop，也可能是其他。OpenDaylight 控制器会默认下发一条动作为 drop 的 table miss 流表项。

一条流表项由三部分构成：基本域、匹配与和动作域。

基本域主要包括优先级（priority）、所属流表（table）、生效时间（duration）、失效时间。失效时间有以下两种：

- 空闲超时时间（idle_timeout）：单位为秒，值为 0 表示永不过期。仅在该流表项空闲时开始计时，如果计时过程中有匹配则清零重新计时，如果计时达到 idle_timeout 值则自动删除该流表项。
- 硬超时时间（hard_timeout）：单位为秒，值为 0 表示永不过期。从流表项创建开始计时，无论有无匹配，达到 hard_timeout 值时就自动删除该流表项。

匹配域涵盖了链路层、网络层和传输层首部中的各个字段，可以匹配一个确定值（例如 10.1.1.1）或是一组范围值（例如 10.1.1.0/24）。

每个流表项可以对应 0 或多个动作，例如转发给控制器、转发到指定的出端口、修改分组中的指定字段值、丢弃等。如果没有定义动作，则默认为丢弃。

分组传送到 OVS 时，首先会从 table 0 开始对分组进行查询匹配，如果匹配到某条流表项，则执行此流表项指定的动作。如果指定动作是匹配另一张指定流表，那么 OVS 就会跳转到指定流表进行查询匹配，并执行匹配流表项指定的动作。如果没有匹配到任何一条流表项，则直接丢弃该分组。

3、ovs-vsctl 命令

ovs-vsctl 命令是获取或更改 OVS 配置信息的重要工具和手段之一，**需要 root 用户或 root 权限才能执行**。

常用命令如表 1 所示（**其中：全大写单词表示命令参数，[ ]表示可选参数**），详细的命令信息可以在安装了 OVS 的 HostOS 上通过“man ovs-vsctl”命令获得。ovs-vsctl 命令中的所有设置操作（含有 add、del、set 的命令字）都会实时生效并保存在 OVS 数据库（conf.db 文件）中。

表 1 ovs-vsctl 常用命令

ovs-vsctl 命令	**作用**
ovs-vsctl show	显示 OVS 中所有网桥的概要信息
ovs-vsctl list br [BRIDGE]	列出所有【或指定】网桥的信息
ovs-vsctl list port [PORT]	列出所有【或指定】端口的信息
ovs-vsctl add-br BRIDGE	添加一个名为 BRIDGE 的新网桥
ovs-vsctl del-br BRIDGE	删除名为 BRIDGE 的网桥及其所有端口
ovs-vsctl add-port BRIDGE PORT	将名为 PORT 的网卡添加到 BRIDGE 网桥中
ovs-vsctl del-port BRIDGE PORT	删除 BRIDGE 网桥的 PORT 端口
ovs-vsctl get-controller BRIDGE	获取网桥的控制器信息

ovs-vsctl 命令	作用
ovs-vsctl del-controller BRIDGE	删除网桥的控制器信息
ovs-vsctl set-controller BRIDGE TRAGET	向网桥添加控制器
ovs-vsctl set-fail-mode BRIDGE MODE	设置网桥的失败模式
ovs-vsctl set bridge BRIDGE protocol=OpenFlow13	设置网桥支持 OpenFlow 1.3 协议
ovs-vsctl set bridge BRIDGE protocol=OpenFlow10	设置网桥支持 OpenFlow 1.0 协议

ovs-vsctl 命令示例：

- 添加一个名为 br-sw 的网桥

```
# ovs-vsctl add-br br-sw
```

- 将虚拟机系统的 eth2 加入到名为 br-sw 的网桥中

```
# ovs-vsctl add-port br-sw eth2
```

注意：命令中的“eth2”必须是 HostOS 已有网卡的网卡名，可以在 HostOS 中使用 ifconfig 命令查看其已有网卡的网卡名。

- 查看 eth2 端口的详细信息

```
# ovs-vsctl list port eth2
```

- 将网桥 br-sw 连接到控制器 192.168.1.1:6633 上

```
# ovs-vsctl set-controller br-sw tcp:192.168.1.1:6633
```

- 将网桥 br-sw 设置为 secure 模式，让其严格按照原有流表进行数据转发

```
# ovs-vsctl set-fail-mode br-sw secure
```

4、ovs-ofctl 命令

ovs-ofctl 命令是获取或更改 OVS 流表信息的重要工具和手段之一，**需要 root 用户或 root 权限才能执行**。

常用命令如表 2 所示（**其中：全大写单词表示命令参数，[ ]表示可选参数**）。ovs-ofctl 命令中的所有设置操作（含有 add、del、mod 的命令字）都会实时生效并保存在 OVS 数据库（conf.db 文件）中。

表 2 ovs-ofctl 常用命令

ovs-ofctl 命令	作用
ovs-ofctl show SWITCH	显示名为 SWITCH 的交换机信息
ovs-ofctl dump-ports SWITCH [PORT]	显示名为 SWITCH 的交换机的所有【或指定】端口的流量统计信息
ovs-ofctl dump-flows SWITCH [FLOW] [-O OpenFLow13]	显示名为 SWITCH 的交换机的所有【或指定】流表项
ovs-ofctl add-flow SWITCH FLOW [-O OpenFLow13]	向 SWITCH 添加流表项
ovs-ofctl mod-flows SWITCH FLOW [-O OpenFLow13]	修改 SWITCH 的匹配流表项
ovs-ofctl del-flows SWITCH FLOW [-O OpenFLow13]	删除 SWITCH 的匹配流表项

OVS 默认使用 OpenFlow 1.0 协议。如要使用 OpenFlow 1.3 协议，则需要在

表 2 的流表相关命令中加上“-O OpenFlow13”选项。

表 2 中的命令参数“FLOW”代表流表项。一条流表规则可以包含多个匹配参数和多个动作参数，使用“,”分隔开。常用的匹配参数如表 3 和表 4 所示，常用的动作参数如表 5 所示，详细信息可在安装了 OVS 的 HostOS 中通过“man ovs-ofctl”命令获得。

表 3 ovs-ofctl 命令中流表规则的常用匹配参数

匹配参数	含义
in_port=PORT	匹配从交换机特定端口进来的分组。PORT 参数是 OpenFlow 的端口号或关键字（如 LOCAL），可通过 ovs-ofctl show 命令获取。
dl_src=MAC	匹配以太帧首部的源 MAC 地址，采用十六进制冒号表示，例如 00:0A:E4:25:6B:B0。
dl_dst=MAC	匹配以太帧首部的目的 MAC 地址。
dl_type=TYPE	匹配以太帧首部的协议类型值，例如 0x0800，即 IP。
nw_src=IP[/NETMASK]	匹配 IP 分组首部的源 IP 地址，例如 10.1.1.1 或 10.1.1.0/24。
nw_dst= IP]/NETMASK]	匹配 IP 分组首部的目的 MAC 地址。
nw_proto=PROTOCOL	匹配 IP 分组首部的协议值。
tcp_src=PORTNUMBER	匹配 TCP 数据段首部的源端口号。
tcp_dst=PORTNUMBER	匹配 TCP 数据段首部的目的端口号。
udp_src=PORTNUMBER	匹配 UDP 数据报首部的源端口号。
udp_dst=PORTNUMBER	匹配 UDP 数据报首部的目的端口号。

表 4 匹配参数中的常用速记符

速记符	等效的匹配参数	速记符	等效的匹配参数
ip	dl_type=0x0800	icmp	dl_type=0x0800,nw_protocol=1
ipv6	dl_type=0x86dd	icmpv6	dl_type=0x08dd,nw_protocol=58
arp	dl_type=0x0806	tcp	dl_type=0x0800,nw_protocol=6
mpls	dl_type=0x8847	udp	dl_type=0x0800,nw_protocol=17

表 5 ovs-ofctl 命令中流表规则的常用动作参数

动作参数	含义
actions=output:PORT	将分组输出到特定端口。PORT 参数含义见表 1-3。
actions=drop	丢弃分组
actions=mod_vlan_vid:VID	修改 VLAN 的 ID
actions=strip_vlan	移除 VLAN ID
actions=mod_dl_src:MAC	修改源 MAC 地址
actions=mod_dl_dst:MAC	修改目的 MAC 地址
actions=mod_nw_src:IP	修改源 IP 地址
actions=mod_nw_dst:IP	修改目的 IP 地址
actions=mod_tp_src:PORT	修改源 TCP/UDP 端口号
actions=mod_tp_dst:PORT	修改目的 TCP/UDP 端口号

ovs-ofctl 命令示例：

- 向网桥 br-sw 下发一条流表项，要求该流表项的优先级为 201，空闲时间为 60 秒，将匹配源 10.2.2.129 到目的 10.2.2.130 的 TCP 数据段输出到端口 3。

```
# ovs-ofctl add-flow br-sw priority=201,idle_timeout=60,tcp,nw_src
  =10.2.2.129,nw_dst=10.2.2.130,actions=output:3
```

- 向网桥 br-sw 下发一条流表项，要求该流表项的优先级为 200，空闲时间为 100 秒，将匹配输入端口 2 的分组丢弃。

```
# ovs-ofctl add-flow br-sw priority=200,idle_timeout=100,in_port=2,
  actions=drop
```

- 删除网桥 br-sw 中 in_port=2 的流表项。

```
# ovs-ofctl del-flows br-sw in_port=2
```

实验拓扑

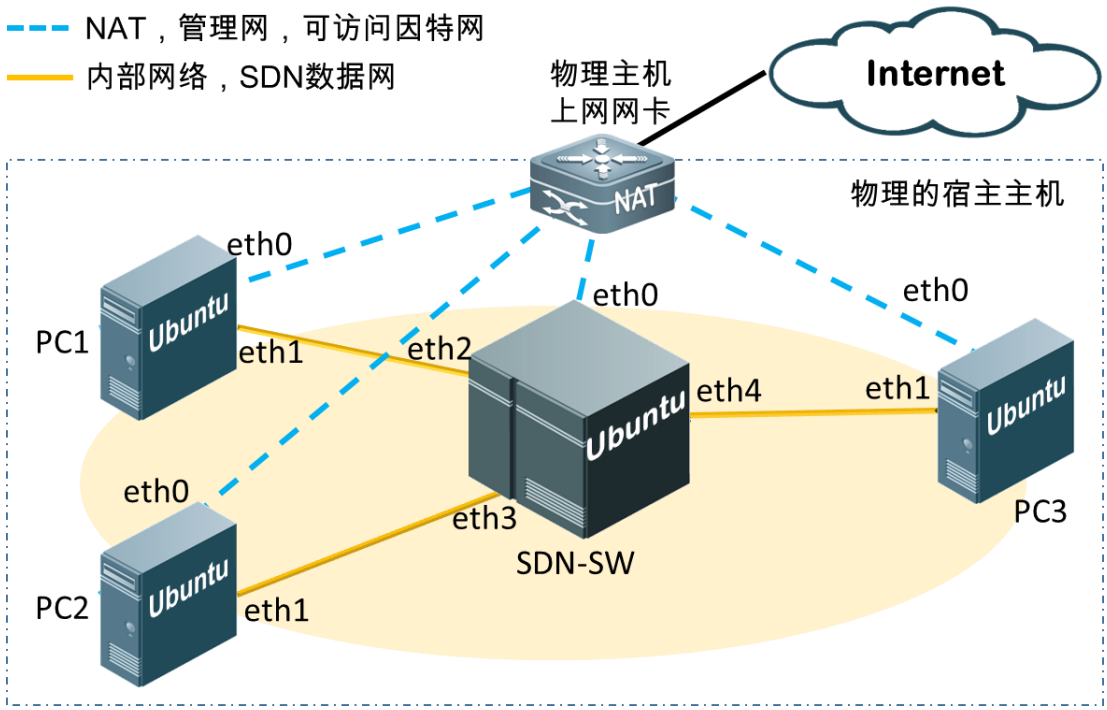


图 3 实验 2 拓扑

实验 2 拓扑中各个虚拟机的主机名、网络连接模式和 IP 地址如表 6 所示。

表 6 实验 2 的虚拟机设置

虚拟机主机名	网卡	网络连接模式			网络接口配置（IP 地址/掩码/GW）
		模式	VirtualBox	VMware	
SDN-SW	网卡 1	NAT	NAT 网络	VMnet8	eth0：DHCP（172.16.1.0/24）
	网卡 3	内部网络	intnet2-1	LAN2-1	eth2：无 IP 地址/掩码/GW
	网卡 4	内部网络	intnet2-2	LAN2-2	eth3：无 IP 地址/掩码/GW
	网卡 2	内部网络	intnet2-3		eth1：无 IP 地址/掩码/GW
	网卡 5	内部网络		LAN2-3	eth4：无 IP 地址/掩码/GW

虚拟机主机名	网卡	网络连接模式			网络接口配置（IP 地址/掩码/GW）
		模式	VirtualBox	VMware	
PC1	网卡 1	NAT	NAT 网络	VMnet8	eth0：DHCP（172.16.1.0/24）
	网卡 2	内部网络	intnet2-1	LAN2-1	eth1：10.2.2.128/24，无 GW
PC2	网卡 1	NAT	NAT 网络	VMnet8	eth0：DHCP（172.16.1.0/24）
	网卡 2	内部网络	intnet2-2	LAN2-2	eth1：10.2.2.129/24，无 GW
PC3	网卡 1	NAT	NAT 网络	VMnet8	eth0：DHCP（172.16.1.0/24）
	网卡 2	内部网络	intnet2-3	LAN2-3	eth1：10.2.2.130/24，无 GW

特别说明：

（1）可以通过复制虚拟机 PC1 来生成 PC3。

（2）如果使用 VMware，则为 SDN-SW 添加一个新的虚拟网卡“网络适配器 5”，并添加一个新的内部网络（即 LAN 区段）“LAN2-3”，将 PC3 的虚拟网卡 2 和 SDN-SW 的虚拟网卡 5 都连接在“LAN2-3”。

（3）如果使用 VirtualBox，因默认只有 4 张虚拟网卡，则添加一个新的内部网络“intnet2-3”来连接 PC3 的虚拟网卡 2 接口和 SDN-SW 的虚拟网卡 2（对应的 eth1 一旦成为 OVS 的 Normal Port，其 IP 地址/掩码自动失效，故无需修改）。

（4）因 SDN-SW 的虚拟机网卡 3、4、2/5 将作为二层转发端口，所以必须工作在混杂模式下。VMware 无需特殊设置，但 VirtualBox 需要将网卡 2、3、4 的“混杂模式”设置为“允许虚拟电脑”，如图 4 所示。

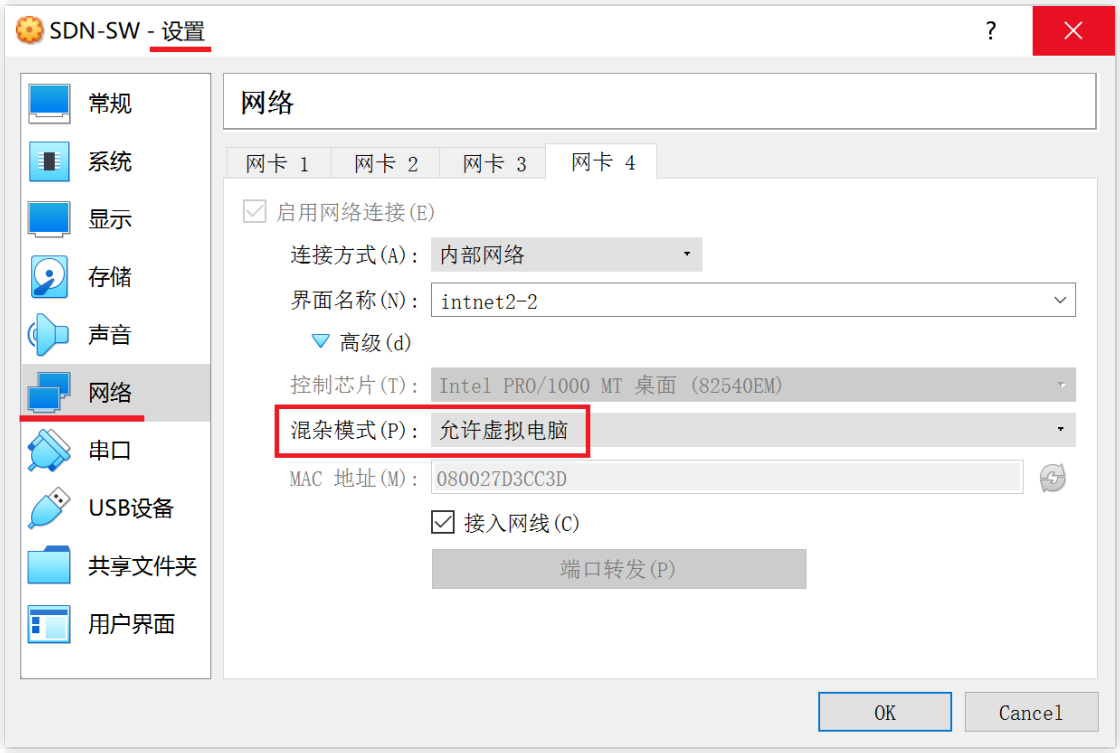


图 4 VirtualBox 虚拟网卡的混杂模式设置

请参考实验 1 中的“五、组建 SDN 基本实验网络”，组建图 3 和表 6 所示的实验 2 网络。

实验 2.1 OVS 的安装和启动

OVS 中负责数据交换和处理的 datapath 模块位于 Linux 内核空间，因此不同版本的 OVS 有其支持对应的 Linux 系统内核版本。OVS 用户空间对 Linux 内核版本不敏感。目前 OVS 的最新版本是 v2.12.0，长期支持版本为 2.5.9。OVS 各版本对应的 Linux 系统内核如表 7 所示。

表 7 OVS 版本对应的 Linux 内核

OVS	Linux 内核	OVS	Linux 内核	OVS	Linux 内核
1.4.x	2.6.18~3.2	1.7.x	2.6.18~3.3	1.9.x	2.6.18~3.8
1.5.x		1.8.x	2.6.18~3.4	1.10.x	
1.6.x				1.11.x	
2.0.x	2.6.32~3.10	2.5.x	2.6.32~4.3	2.9.x	3.10~4.13
2.1.x	2.6.32~3.11	2.6.x	3.10~4.7	2.10.x	3.10~4.17
2.3.x	2.6.32~3.14	2.7.x	3.10~4.9	2.11.x	3.10~4.18
2.4.x	2.6.32~4.0	2.8.x	3.10~4.12	2.12.x	3.10~5.0

（引自 http://docs.openvswitch.org/en/latest/faq/releases/）

本实验中的所有操作仅在 64-bit Ubuntu 14.04.6 桌面版系统和 OVS 2.6.6 软件版本中验证过可以正常工作。（可以尝试基于表 7 安装其他版本的 Linux 系统和 OVS 软件）。

OVS 作为一个开源项目，用户可以自行访问 OVS 官网下载所需版本的 OVS 安装包，官网下载地址为 http://www.openvswitch.org/download。

Vmware 的 VMware Tools 直接提供物理机操作系统与虚拟机操作系统之间的文字拷贝/粘贴和文件拖放，VirtualBox 可以在虚拟机的“设置→常规→高级”中将“共享粘贴板”和“拖放”选择为“双向”（如图 5 所示）来实现相同功能。

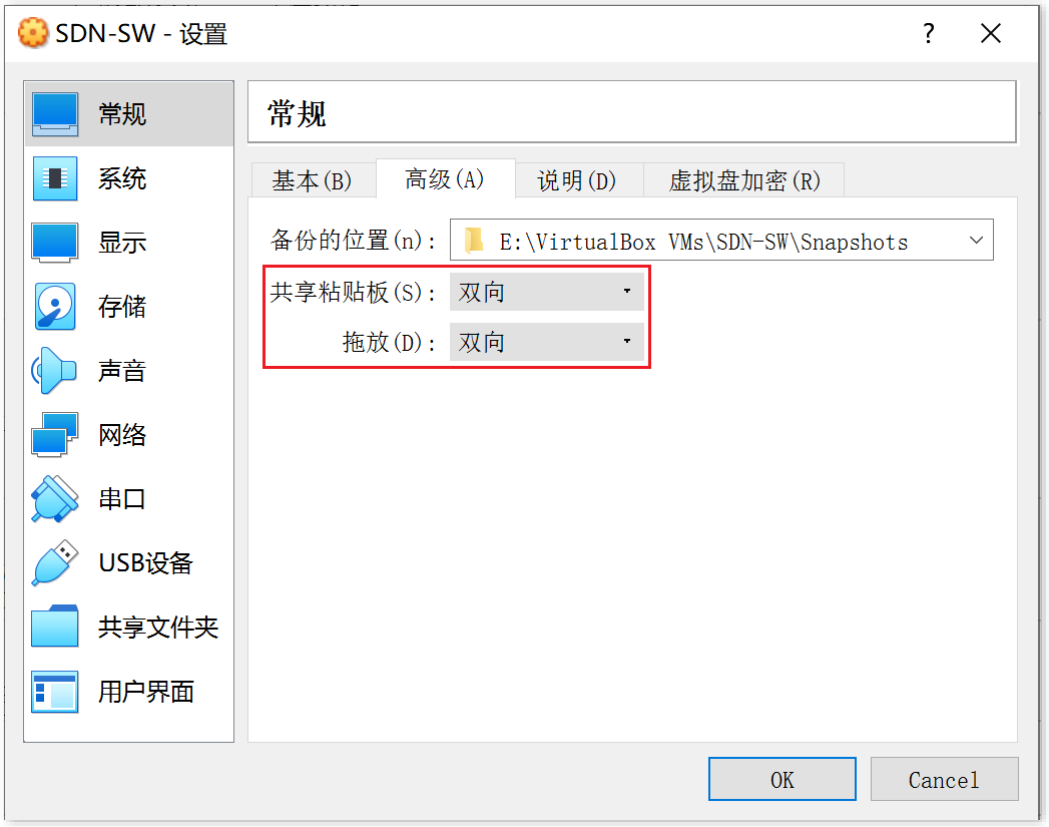


图 5 VirtualBox 的共享粘贴板和拖放设置

下面是在 SDN-SW 虚拟机中安装 OVS 2.6.6 的具体步骤。

步骤 1、获取并解压源代码包

获取 OVS 2.6.6 源代码包（openvswitch-2.6.6.tar.gz）后，进入存放目录进行解压（例如解压存放在/mnt 目录下）：

```
$ sudo tar -xvf openvswitch-2.6.6.tar.gz -C /mnt
```

步骤 2、预编译源代码包

因为 OVS 安装采用的是源码编译安装方式，所以需要先进入 OVS 源代码包解压目录，使用如下命令预编译源代码包：

```
$ cd /mnt/openvswitch-2.6.6
$ sudo apt-get update
$ sudo ./configure --with-linux=/lib/modules/`uname -r`/build
```

注意：命令中用于 uname -r 的 ` 符号不是单引号，而是在标准美国键盘上位于 ESC 键下面、字符 1 左边、TAB 键上面的那个按键，和~符号在一起，请特别留心。

预编译命令执行过程中如无 Error 错误信息，即可进行下一步。否则，请根据 Error 信息自行解决或上网查找解决。

步骤 3、编译和安装 OVS

使用如下命令编译和安装 OVS：

```
$ sudo make
$ sudo make install
```

编译和安装过程中如无 Error 错误信息，即说明 OVS 安装成功。否则，请根据 Error 信息自行解决或上网查找解决。

步骤 4、加载 OVS 启动所需环境变量

OVS 安装完毕后，默认还不可以使用命令启动，需要添加并加载其环境变量。

首先，分别编辑/etc/profile 文件和/root/.bashrc 文件：

```
$ sudo vim /etc/profile
$ sudo vim /root/.bashrc
```

在这两份文件的末尾处都添加以下一行信息，然后保存文件：

```
export PATH=$PATH:/usr/local/share/openvswitch/scripts
```

然后切换为 root 用户，并加载环境变量：

```
$ su root
# source /etc/profile
```

步骤 5、启动 OVS

加载完环境变量后，即可使用 ovs-ctl 命令启动 OVS。（**注意：必须切换为 root 用户才能执行 ovs-ctl 命令。**）

```
# ovs-ctl start
```

可以使用如下命令查看所安装的 OVS 版本：

```
# ovs-vsctl --version
```

记录：ovs-ctl start 和 ovs-vsctl --version 命令及其结果截图。

实验 2.2 使用 ovs-vsctl 命令创建网桥

说明：ovs-vsctl 命令配置的网桥及其端口信息会保存在 OVS 数据库（conf.db 文件）中，但是 ovs-ofctl 配置的流表项不会被保存。因此本实验过程中如果重新启动 SDN-SW，则需使用 ovs-ctl start 命令启动 OVS，即可从 OVS 数据库中自动读取重启前的配置以生成网桥及其端口，但重启前配置的所有流表项则需重新配置。

步骤 1、测试 PC1 与 PC2 的通信

启动 PC1 和 PC2，在 PC1 上 ping PC2 的数据网接口（eth1）IP 地址（10.2.2.129），请使用-c 选项只发送 4 个 ping 报文。

记录：ping 命令及其结果截图。

步骤 2、创建网桥并接入 PC1 和 PC2

使用 ovs-vsctl 命令创建一个名为 br0 的网桥，将 SDN-SW 虚拟机的 eth2 和 eth3 网络接口加入到 br0 网桥中，然后使用 ovs-vsctl show 命令查看网桥。

记录：创建网桥、添加端口、查看网桥的 ovs-vsctl 命令及其结果截图。

步骤 3、查看流表项

使用 ovs-ofctl 命令查看 br0 的所有流表项。

记录：查看流表项的 ovs-ofctl 命令及其结果截图。

步骤 4、再次测试 PC1 与 PC2 的通信

在 PC1 上 ping PC2 的数据网接口（eth1）IP 地址（10.2.2.129），请使用-c 选项只发送 4 个 ping 报文。

记录：ping 命令及其结果截图。

步骤 5、再次查看流表项

在 SDN-SW 上使用 ovs-ofctl 命令查看 br0 的所有流表项。

记录：查看流表项的 ovs-ofctl 命令及其结果截图。

实验 2.3 使用 ovs-ofctl 命令下发流表项

步骤 1、将 PC3 接入网桥

使用 ovs-vsctl 命令将 SDN-SW 虚拟机连接 PC3 的网络接口加入到 br0 网桥中，然后使用 ovs-vsctl show 命令查看网桥。

记录：添加端口、查看网桥的 ovs-vsctl 命令及其结果截图。

步骤 2、测试 PC3 与 PC1、PC2 的通信

启动 PC3，在 PC3 上分别 ping PC1 和 PC2 的数据网接口（eth1）IP 地址，使用-c 选项发送 4 个 ping 报文。

记录：ping 命令及其结果截图。

步骤 3、查看 3 台主机的地址和拓扑连接信息

查看 PC1、PC2 和 PC3 的 eth1 接口的 MAC 地址和 IP 地址，并使用 ovs-ofctl show br0 命令查看 br0 连接 3 台 PC 的 OpenFlow 端口号。

记录：将查看的信息填入表 8 中。

表 8 实验 2.3 的步骤 3 记录

主机	主机 eth1 口的 MAC 地址	主机 eth1 口的 IP 地址	OpenFlow 端口号
PC1			
PC2			
PC3			

步骤 4、下发 L2 流表项

首先在 SDN-SW 上使用下面这个命令添加一条流表项：

```
# ovs-ofctl add-flow br0 priority=205,idle_timeout=0,arp,actions=normal
```

接着，再使用 ovs-ofctl add-flow 命令下发**一条**匹配以太帧首部目的 MAC 地址字段的 L2 流表项，要求：

- 将目的 MAC 地址为 PC3 eth1 接口 MAC 地址的分组全都丢弃，但 PC1 和 PC2 能正常通信；
- 流表项的优先级为 200，idle_timeout 为 0 秒。

记录：下发 L2 流表项的 ovs-ofctl add-flow 命令。

配置完成后，请在 PC1 和 PC3 上使用如下命令启动 Wireshark：

```
$ sudo wireshark
```

启动 Wireshark 时如出现图 6 所示的错误提出，请直接点击提示窗口中的“OK”按钮即可，该错误并不影响 Wireshark 的正常使用！

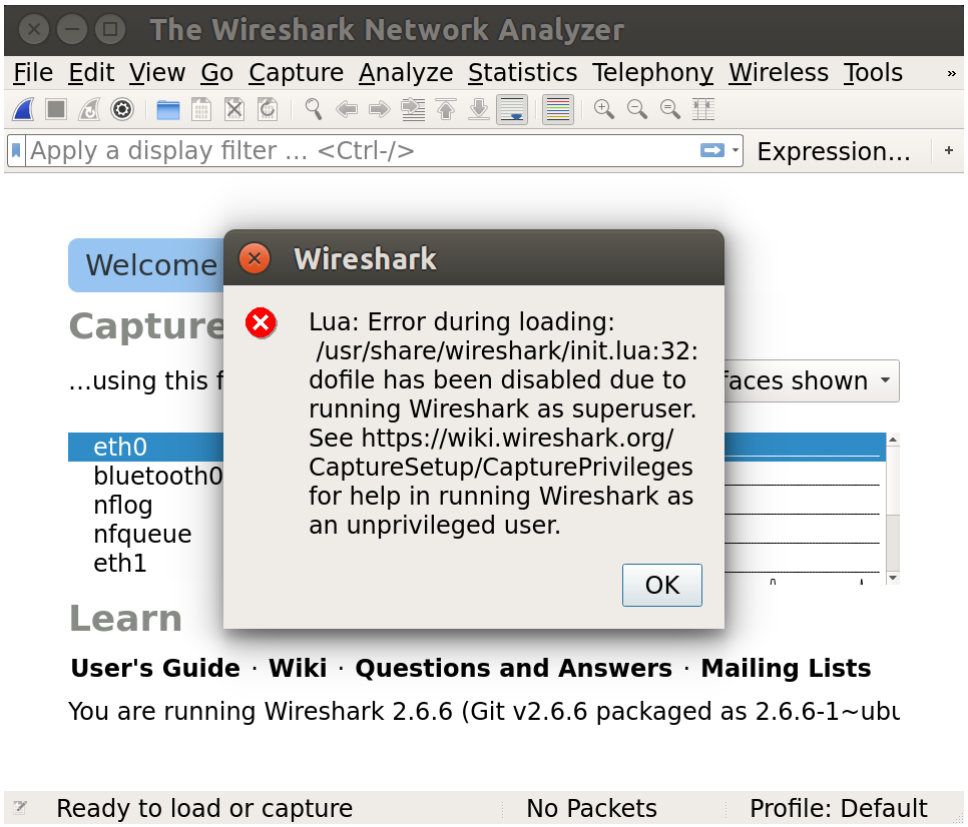


图 6 运行 Wireshark 时的错误提示

然后在 PC1 和 PC3 上启动对其 eth1 接口的 Wireshark 抓包，直至如下测试完成后停止 Wireshark 抓包：

- PC1 ping PC2，即在 PC1 上 ping -c 4 10.2.2.129
- PC2 ping PC3，即在 PC2 上 ping -c 4 10.2.2.130

- PC3 ping PC1，即在 PC3 上 ping -c 4 10.2.2.128

记录：在 PC1、PC2 和 PC3 上运行的 ping 命令及其结果截图。

请根据 br0 的所有流表项，以及在 PC1 和 PC3 上使用 Wireshark 抓取 ping 测试期间的 ping 请求报文和 ping 响应报文，回答以下问题（Q1~Q4）：

Q1、PC1 ping PC2 时，PC1 发出的 ping 请求报文匹配 br0 上的哪一条流表项？br0 如何处理这些 ping 请求报文？请给出依据截图。

PC2 返回的 ping 响应报文匹配 br0 上的哪一条流表项？br0 如何处理这些 ping 响应报文？请给出依据截图。

Q2、PC2 ping PC3 时，PC2 发出的 ping 请求报文匹配 br0 上的哪一条流表项？br0 如何处理这些 ping 请求报文？请给出依据截图。

这次测试中是否有 ping 响应报文？如果没有，请说明理由；如果有，则匹配 br0 上的哪一条流表项？br0 如何处理这些 ping 响应报文？请给出依据截图。

Q3、PC3 ping PC1 时，PC3 发出的 ping 请求报文匹配 br0 上的哪一条流表项？br0 如何处理这些 ping 请求报文？请给出依据截图。

这次测试中是否有 ping 响应报文？如果没有，请说明理由；如果有，则匹配 br0 上的哪一条流表项？br0 如何处理这些 ping 响应报文？请给出依据截图。

Q4、步骤 4 开始时配置的那条匹配 ARP、动作为 normal 的流表项对步骤 4 中的 ping 测试有什么影响？请详细说明。

步骤 5、下发 L3 流表项

步骤 4 下发的流表项仍继续有效的情况下，使用 ovs-ofctl add-flow 命令下发匹配 IP 分组首部源和目的 IP 地址对的 L3 流表项，要求：

- PC2 和 PC3 之间可以相互通信，但 PC2 不能 ping 通 PC1；
- 流表项的优先级为 210，idle_timeout 为 0 秒。

记录：下发 L3 流表项的 ovs-ofctl add-flow 命令。

配置完成后，请在 PC1 和 PC2 上使用启动 Wireshark 对其 eth1 接口进行抓包，直至如下测试完成后停止 Wireshark 抓包：

- PC2 ping PC3，即在 PC2 上 ping –c 4 10.2.2.130
- PC2 ping PC1，即在 PC2 上 ping –c 4 10.2.2.128

记录：在 PC2 上运行的 ping 命令及其结果截图。

请根据 br0 的所有流表项，以及在 PC1 和 PC2 上使用 Wireshark 抓取 ping 测试期间的 ping 请求报文和 ping 响应报文，回答以下问题（Q5~Q8）：

Q5、PC2 ping PC3 时，PC2 发出的 ping 请求报文匹配 br0 上的哪一条流表项？br0 如何处理这些 ping 请求报文？请给出依据截图。

PC3 返回的 ping 响应报文匹配 br0 上的哪一条流表项？br0 如何处理这些 ping 响应报文？请给出依据截图。

Q6、PC2 ping PC1 时，PC2 发出的 ping 请求报文匹配 br0 上的哪一条流表项？br0 如何处理这些 ping 请求报文？请给出依据截图。

Q7、如果 PC1 向 PC2 发送分组，该分组会匹配 br0 上的哪一条流表项？br0 将如何处理这些分组？请给出依据截图。

Q8、步骤 5 中下发的 L3 流表项是否会影响 PC1 和 PC3 之间的通信？请详细说明。

步骤 6、下发 L4 流表项

在步骤 4 和步骤 5 下发的流表项仍继续有效的情况下（即 PC1 不能 ping 通 PC2 和 PC3），使用 ovs-ofctl add-flow 命令下发匹配 TCP 报文段和 UDP 数据报的 L4 流表项，要求：

- PC1 能作为 iperf 客户端向 PC3 正常发起 TCP 性能测试；
- PC1 能作为 iperf 客户端向 PC2 正常发起 UDP 性能测试；
- 流表项的优先级为 220，idle_timeout 为 0 秒。

记录：下发 L4 流表项的 ovs-ofctl add-flow 命令。

iperf 是一个网络性能测试工具，可以运行在 Linux 和 Windows 等操作系统上，可以测试 TCP 和 UDP 性能，可以报告带宽、时延抖动和丢包等测试结果。默认情况下，iperf 客户端向 iperf 服务器 TCP 5001 端口以 8kbps 的速度传输 10 秒测试数据，iperf 服务器在此过程中统计并记录带宽、延时抖动等信息。等客户端将数据发送完毕后，服务器会将测试统计数据反馈给客户端。

启动 iperf 服务器的示例命令如下：

```
# iperf -s                  //启动 iperf 服务器，监听在 TCP 5001 端口
# iperf -s -u               //启动 iperf 服务器，监听在 UDP 5001 端口
```

启动 iperf 客户端的示例命令如下：

```
# iperf -c 10.2.2.130       //启动 iperf 客户端，连接 10.2.2.130 服务器测试 TCP 性能
# iperf -c 10.2.2.129 -u    //启动 iperf 服务器，连接 10.2.2.129 服务器测试 UDP 性能
```

关于 iperf 的详细命令信息可以通过“man iperf”命令获得。

配置完成后进行如下测试：

- 在 PC3 上启动 iperf 服务器端，监听在 TCP 5001 端口
- 在 PC2 上启动 iperf 服务器端，监听在 UDP 5001 端口
- 在 PC1 上启动 iperf 客户端连接 PC3 的 ipfer 服务器进行 TCP 性能测试
- 在 PC1 上 ping –c 4 10.2.2.130
- 在 PC1 上启动 iperf 客户端连接 PC2 的 ipfer 服务器进行 UDP 性能测试

- 在 PC1 上 ping –c 4 10.2.2.129

记录：在 PC1、PC2 和 PC3 上运行的 ipfer 命令和 ping 命令，以及这些命令的运行结果截图。

请根据 br0 的所有流表项，回答以下问题（Q9~Q11）：

Q9、PC1 的 iperf 客户端发送给 PC3 的 iperf 服务器端的 TCP 性能测试数据匹配 br0 上的哪一条流表项？br0 如何处理这些测试数据？请给出依据截图。

PC3 的 iperf 服务器端返回的测试统计反馈数据匹配 br0 上的哪一条流表项？br0 如何处理这些测试统计反馈数据？请给出依据截图。

Q10、PC1 的 iperf 客户端发送给 PC2 的 iperf 服务器端的 UDP 性能测试数据匹配 br0 上的哪一条流表项？br0 如何处理这些测试数据？请给出依据截图。

PC2 的 iperf 服务器端返回的测试统计反馈数据匹配 br0 上的哪一条流表项？br0 如何处理这些测试统计反馈数据？请给出依据截图。

Q11、为什么 PC1 和 PC3 之间能进行正常的 iperf 测试，却不能进行 ping 通信？请详细说明。

实验报告任务

实验报告使用“课程实验报告模板.doc”，模板中的“二、实验步骤、数据及分析结果”请填写前述实验 2.1、实验 2.2 和实验 2.3 中要求的“记录”和“问题”。

附录：轻量版实验方案

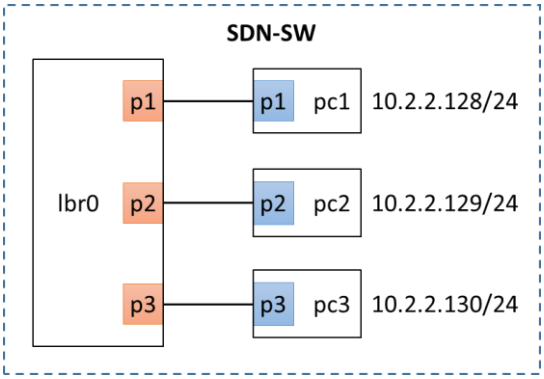
实验 2.2 需要使用 3 台虚拟机，实验 2.3 需要使用 4 台虚拟机，这对物理计算机的性能（主要是内存）是一个巨大挑战！因此，本附录提供的轻量版实验方案将使用 Linux 的 Namespace 技术，仅开启 SDN-SW 这一台虚拟机来完成实验 2.2 和实验 2.3。

Namespace 是 Linux 提供的一种内核级别的环境隔离方法，其主要目标就是实现轻量级虚拟化服务。处于不同 Namespace 中的进程拥有独立的全局系统资源。Namespace 技术为 Linux 容器技术（docker）的出现和发展提供了基础条件。

Namespace 提供了针对文件系统、网络、进程、用户等多种隔离能力。轻量版实验方案使用 Network Namespace，为实验拓扑中的 PC1、PC2、PC3 创建不同的 Namespace，每个 Namespace 都有自己的网络接口；OVS 使用 Internal Port 连接每个 Namespace 的网络接口。

轻量版拓扑

轻量版实验方案的实验拓扑如附图 1 所示。



附图 1 轻量版实验 2 拓扑

图中的 lbr0 是在轻量版实验方案中创建的 OVS 网桥，pc1、pc2 和 pc3 是使用 Network Namespace 创建的三个虚拟网络空间，等效于图 3 所示实验 2 拓扑中的 PC1、PC2 和 PC3。

OVS 除了可以将 HostOS（例如附图 1 中的 SDN-SW 虚拟机）的已有网卡作为 Normal Port 挂载到所创建的网桥中，还可以创建 Internal Port（例如附图 1 中位于 lbr0 内的 p1、p2、p3 端口）。每创建一个 Internal Port，OVS 就会在 HostOS 上自动创建一个同名的 Internal 类型的虚拟网卡（例如附图 1 中位于 pc1、pc2、pc3 中的 p1、p2 和 p3 接口）。因此，只需将 p1、p2、p3 接口移入相应的虚拟网络空间 pc1、pc2、pc3，配置 IP 地址，然后再以混杂模式打开，即可构建与图 3 所示实验 2 拓扑完全等效的轻量版实验 2 拓扑。

轻量版 2.2 使用 ovs-vsctl 命令创建网桥

说明：ovs-vsctl 命令配置的网桥及其端口信息会保存在 OVS 数据库（conf.db 文件）中，但是 ovs-ofctl 配置的流表项不会被保存。因此本实验过程中如果重新启动 SDN-SW，则需使用 ovs-ctl start 命令启动 OVS，即可从 OVS 数据库中自动读取重启前的配置以生成网桥及其端口，但重启前配置的所有流表项则需重新配置。

步骤 1、创建网桥及其 Internal 端口

使用 ovs-vsctl 命令创建一个名为 lbr0 的网桥，并添加名为 p1、p2、p3 的三个 Internal 端口，然后使用 ovs-vsctl show 命令查看网桥。

添加 Internal 端口的 ovs-vsctl add-port 命令语法如下：

```
# ovs-vsctl add-port BRIDGE PORT -- set Interface PORT type=internal
```

其中，“BRIDGE”为网桥名，“PORT”为端口名。

记录：创建网桥、添加端口、查看网桥的 ovs-vsctl 命令及其结果截图。

步骤 2、查看流表项

使用 ovs-ofctl 命令查看 br0 的所有流表项。

记录：查看流表项的 ovs-ofctl 命令及其结果截图。

步骤 3、创建三个虚拟网络空间并连接网桥

三个虚拟空间的名字分为为 pc1、pc2、pc3。

创建 pc1 并连接网桥的命令如下：

```
# ip netns add pc1                                  //创建网络空间 pc1
# ip link set p1 netns pc1                          //将 p1 接口移入 pc1
# ip netns exec pc1 ip addr add 10.2.2.128/24 dev p1  //为 p1 配 IP 地址/掩码
# ip netns exec pc1 ifconfig p1 promisc up          //混杂模式打开 p1 接口
```

使用同样的方法创建 pc2 和 pc3，并将它们接入网桥。

记录：创建 pc2 和 pc3 并连接网桥的命令。

步骤 4、测试 pc1 与 pc2 的通信

使用如下命令进行 pc1 ping pc2 的通信测试：

```
# ip netns exec pc1 ping -c 4 10.2.2.129
```

记录：ping 命令及其结果截图。

步骤 5、再次查看流表项

使用 ovs-ofctl 命令查看 br0 的所有流表项。

记录：查看流表项的 ovs-ofctl 命令及其结果截图。

轻量版 2.3 使用 ovs-ofctl 命令下发流表项

步骤 1、测试 pc3 与 pc1、pc2 的通信

使用如下命令进行 pc3 ping pc1 和 pc3 ping pc2 的通信测试：

```
# ip netns exec pc3 ping -c 4 10.2.2.128
# ip netns exec pc3 ping -c 4 10.2.2.129
```

记录：ping 命令及其结果截图。

步骤 2、查看 pc1、pc2、pc3 的地址和拓扑连接信息

使用如下命令查看 p1、p2 和 p3 接口的 MAC 地址和 IP 地址：

```
# ip netns exec pc1 ifconfig
# ip netns exec pc2 ifconfig
# ip netns exec pc3 ifconfig
```

使用 ovs-ofctl show br0 命令查看 p1、p2、p3 端口的 OpenFlow 端口号。

记录：将查看的信息填入表 8 中。

表 8 实验 2.3 的步骤 3 记录

网络接口	MAC 地址	IP 地址	OpenFlow 端口号
p1			
p2			
p3			

步骤 3、下发 L2 流表项

首先在 SDN-SW 上使用下面这个命令添加一条流表项：

```
# ovs-ofctl add-flow lbr0 priority=205,idle_timeout=0,arp,actions=normal
```

接着，再使用 ovs-ofctl add-flow 命令下发**一条**匹配以太帧首部目的 MAC 地址字段的 L2 流表项，要求：

- 将目的 MAC 地址为 pc3 的 p3 接口 MAC 地址的分组全都丢弃，但 pc1 和 pc2 能正常通信；
- 流表项的优先级为 200，idle_timeout 为 0 秒。

记录：下发 L2 流表项的 ovs-ofctl add-flow 命令。

配置完成后，打开一个 Terminal 窗口输入以下命令启动对 p1 接口的 tcpdump 抓包：

```
# ip netns exec pc1 tcpdump -i p1 -w /tmp/2-3-3-pc1.pcap icmp
```

接着，再打开另一个 Terminal 窗口输入以下命令启动对 p3 接口的 tcpdump 抓包：

```
# ip netns exec pc3 tcpdump -i p3 -w /tmp/2-3-3-pc3.pcap icmp
```

直至如下测试完成后，在这两个 tcpdump 抓包的 Terminal 窗口中按“ctrl-c”停止抓包：

- pc1 ping pc2

```
# ip netns exec pc1 ping -c 4 10.2.2.129
```

- pc2 ping pc3

```
# ip netns exec pc2 ping -c 4 10.2.2.130
```

- pc3 ping pc1

```
# ip netns exec pc3 ping -c 4 10.2.2.128
```

记录：ping 命令及其结果截图。

请根据 lbr0 的所有流表项，以及在 ping 测试期间使用 tcpdump 抓取的 p1 和 p3 接口上的 ping 请求报文和 ping 响应报文（使用“tcpdump -r /tmp/2-3-3-pc1.pcap”命令和“tcpdump -r /tmp/2-3-3-pc3.pcap”命令读取），回答以下问题（Q1~Q4）：

Q1、pc1 ping pc2 时，PC1 发出的 ping 请求报文匹配 lbr0 上的哪一条流表项？lbr0 如何处理这些 ping 请求报文？请给出依据截图。

pc2 返回的 ping 响应报文匹配 lbr0 上的哪一条流表项？lbr0 如何处理这些 ping 响应报文？请给出依据截图。

Q2、pc2 ping pc3 时，pc2 发出的 ping 请求报文匹配 lbr0 上的哪一条流表项？lbr0 如何处理这些 ping 请求报文？请给出依据截图。

这次测试中是否有 ping 响应报文？如果没有，请说明理由；如果有，则匹配 lbr0 上的哪一条流表项？lbr0 如何处理这些 ping 响应报文？请给出依据截图。

Q3、pc3 ping pc1 时，pc3 发出的 ping 请求报文匹配 lbr0 上的哪一条流表项？lbr0 如何处理这些 ping 请求报文？请给出依据截图。

这次测试中是否有 ping 响应报文？如果没有，请说明理由；如果有，则匹配 lbr0 上的哪一条流表项？lbr0 如何处理这些 ping 响应报文？请给出依据截图。

Q4、步骤 3 开始时配置的那条匹配 ARP、动作为 normal 的流表项对步骤 3 中的 ping 测试有什么影响？请详细说明。

步骤 4、下发 L3 流表项

步骤 3 下发的流表项仍继续有效的情况下，使用 ovs-ofctl add-flow 命令下发匹配 IP 分组首部源和目的 IP 地址对的 L3 流表项，要求：

- pc2 和 pc3 之间可以相互通信，但 pc2 不能 ping 通 pc1；
- 流表项的优先级为 210，idle_timeout 为 0 秒。

记录：下发 L3 流表项的 ovs-ofctl add-flow 命令。

配置完成后，请对 p1 和 p2 接口进行 tcpdump 抓包，直至如下测试完成后

停止抓包：

- pc2 ping pc3

```
# ip netns exec pc2 ping -c 4 10.2.2.130
```

- pc2 ping pc1

```
# ip netns exec pc2 ping -c 4 10.2.2.128
```

记录：ping 命令及其结果截图。

请根据 lbr0 的所有流表项，以及在 ping 测试期间使用 tcpdump 抓取的 p1 和 p2 接口上的 ping 请求报文和 ping 响应报文，回答以下问题（Q5~Q8）：

Q5、pc2 ping pc3 时，pc2 发出的 ping 请求报文匹配 lbr0 上的哪一条流表项？lbr0 如何处理这些 ping 请求报文？请给出依据截图。

pc3 返回的 ping 响应报文匹配 lbr0 上的哪一条流表项？lbr0 如何处理这些 ping 响应报文？请给出依据截图。

Q6、pc2 ping pc1 时，pc2 发出的 ping 请求报文匹配 lbr0 上的哪一条流表项？lbr0 如何处理这些 ping 请求报文？请给出依据截图。

Q7、如果 pc1 向 pc2 发送分组，该分组会匹配 lbr0 上的哪一条流表项？lbr0 将如何处理这些分组？请给出依据截图。

Q8、步骤 4 中下发的 L3 流表项是否会影响 pc1 和 pc3 之间的通信？请详细说明。

步骤 5、下发 L4 流表项

在步骤 3 和步骤 4 下发的流表项仍继续有效的情况下（即 pc1 不能 ping 通 pc2 和 pc3），使用 ovs-ofctl add-flow 命令下发匹配 TCP 报文段和 UDP 数据报的 L4 流表项，要求：

- pc1 能作为 iperf 客户端向 pc3 正常发起 TCP 性能测试；
- pc1 能作为 iperf 客户端向 pc2 正常发起 UDP 性能测试；
- 流表项的优先级为 220，idle_timeout 为 0 秒。

记录：下发 L4 流表项的 ovs-ofctl add-flow 命令。

配置完成后进行如下测试：

- pc3 启动 iperf 服务器端，监听在 TCP 5001 端口

```
# ip netns exec pc3 iperf -s
```

- pc2 启动 iperf 服务器端，监听在 UDP 5001 端口

```
# ip netns exec pc2 iperf -s -u
```

- pc1 启动 iperf 客户端连接 PC3 的 ipfer 服务器进行 TCP 性能测试

```
# ip netns exec pc1 iperf -c 10.2.2.130
```

- pc1 ping pc3

```
# ip netns exec pc1 ping -c 4 10.2.2.130
```

- 在 PC1 上启动 iperf 客户端连接 PC2 的 ipfer 服务器进行 UDP 性能测试

```
# ip netns exec pc1 iperf -c 10.2.2.129 -u
```

- pc1 ping pc2

```
# ip netns exec pc1 ping -c 4 10.2.2.129
```

记录：ipfer 命令和 ping 命令，以及这些命令的运行结果截图。

请根据 lbr0 的所有流表项，回答以下问题（Q9~Q11）：

Q9、pc1 的 iperf 客户端发送给 pc3 的 iperf 服务器端的 TCP 性能测试数据匹配 lbr0 上的哪一条流表项？lbr0 如何处理这些测试数据？请给出依据截图。

pc3 的 iperf 服务器端返回的测试统计反馈数据匹配 lbr0 上的哪一条流表项？lbr0 如何处理这些测试统计反馈数据？请给出依据截图。

Q10、pc1 的 iperf 客户端发送给 pc2 的 iperf 服务器端的 UDP 性能测试数据匹配 lbr0 上的哪一条流表项？lbr0 如何处理这些测试数据？请给出依据截图。

pc2 的 iperf 服务器端返回的测试统计反馈数据匹配 lbr0 上的哪一条流表项？lbr0 如何处理这些测试统计反馈数据？请给出依据截图。

Q11、为什么 pc1 和 pc3 之间能进行正常的 iperf 测试，却不能进行 ping 通信？请详细说明。