实验 3、OpenDaylight 控制器的安装与运行

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》和《SDN 培训课程-3-控制器 v3.1.pdf》

实验原理

SDN 组网架构实现了数据平面和控制平面的分离，SDN 控制平面大体划分为两个部分：SDN 控制器和 SDN 网络控制应用程序。一个通用的 SDN 控制器可大体组织为三个层次：

- **通信层**：完成 SDN 控制器和受控网络设备（如 SDN 交换机）之间的通信。该通信接口被称为“南向接口”，典型协议是 OpenFlow。
- **网络范围状态管理层**：负责管理维护有关网络的主机、链路、交换机、其他 SDN 控制设备的状态信息。
- **与网络控制应用程序的接口**：称为“北向接口”，控制器通过北向接口与网络控制应用程序交互。

目前主流的控制器由 ONOS、OpenDaylight、Ryu、Floodlight、POX、NOX 等。

1、ODL 简介

Open Daylight（简称 ODL）是一个高度可用、模块化、可扩展、支持多协议的 SDN 控制器平台，是部署最广泛的开源 SDN 控制器平台。

2013 年，Linux Foundation 联合 Cisco、Juniper 和 Broadcom 等多家网络设备商创立了开源项目 OpenDaylight，它的发起者和赞助商多为设备厂商而非运营商等网络设备消费者。ODL 项目的发展目标在于推出一个通用的 SDN 控制平台、网络操作系统，从而管理不同的网络设备，正如 Linux 和 Windows 等操作系统可以在不同的底层设备上运行一样。ODL 支持多种南向协议，包括 OpenFlow、Netconf 和 OVSDB 等，是一个广义的 SDN 控制平台，而不是仅支持 Open Flow 的狭义 SDN 控制器。

ODL 以元素周期表中的元素名称作为版本号，并每 6 个月更新一个版本。从第一个版本“氢（Hydrogen）”发布至今，已经发布了 12 个版本，当前版本为“镁（Magnesium）”。中间的第 2~11 个版本依次为：氦（Helium）、锂（Lithium）、铍（Beryllium）、硼（Boron）、碳（Carbon）、氮（Nitrogen）、氧（Oxygen）、氟（Fluorine）、氖（Neon）、铝（Sodium）

本实验使用的 ODL 版本是 2015 年 6 月 29 日发布的第 3 版“锂（Lithium）”。锂版本增加了对 OpenStack 的支持，并针对之前的安全漏洞加强了安全方面的工作，可拓展性和性能也得到了提升。此外，该版本加大了对 NFV 方面的开发投入。

ODL 锂版本的架构如图 1 所示（https://www.opendaylight.org/what-we-do/current-release/lithium），由许多不同的模块组成，可以根据需要组合这些模块以满足给定方案的要求。

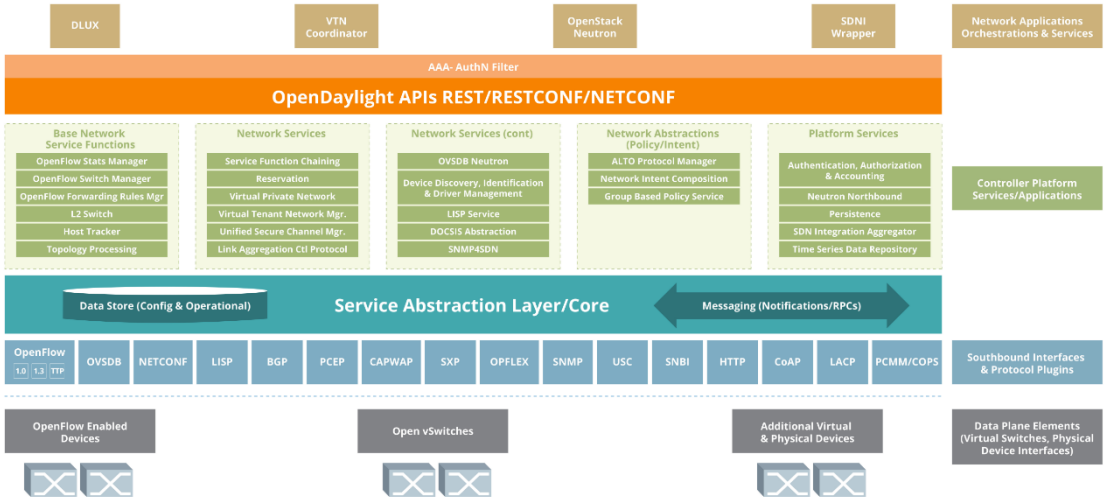


图 1 锂版本的 ODL 架构图

图 1 中的 ODL 架构从下之上分别为：

- **南向接口与协议插件（Southbound Interfaces & Protocol Plugins）**：包含了如 OpenFlow、OVSDB、NETCONF、SNMP 等多种南向协议的实现。
- **SAL/Core**：模型驱动的服务抽象层（Model-Driver Service Abstraction Layer，MD-SAL）是 ODL 平台的核心。在 ODL 中，基础网络设备和网络应用程序均表示为对象或模型，其交互和适配在 SAL 中进行处理。YANG 模型提供了设备或应用程序功能的通用描述，无需知道彼此的具体实现细节。ODL 的所有模块都需要在 MD-SAL 中注册才能正常工作。MD-SAL 也是逻辑上的信息容器，负责数据存储、请求路由、消息订阅和发布等内容。
- **控制器平台服务/应用（Controller Platform Services/Applications）**：提供基础网络服务功能，以及网络服务、网络抽象和平台服务等内部应用程序。
- **ODL APIs**：为外部应用程序提供的北向接口，包含了开放的 REST、RESTCONF、NETCONF 等北向接口和 AAA 认证部分。
- **网络应用程序编排与服务（Network Applications Orchestrations & Services）**：基于北向接口开发的外部应用程序。

从第 10 版“氖（Neon）”开始直至最新的镁版本，ODL 将其架构概括为图 2 所示（https://www.opendaylight.org/what-we-do/current-release/neon）。

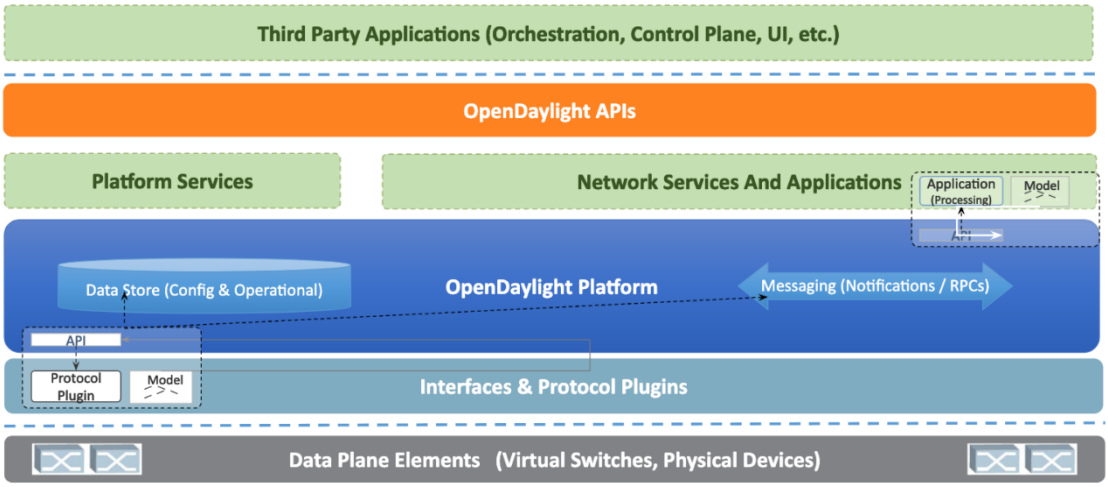


图 2 氖、钠、镁版本的 ODL 架构图

ODL 基于 Java 开发，采用 OSGi（Open Service Gateway Initiative）框架。OSGi 是一个 Java 框架，其中定义了应用程序的生命周期模式和服务注册等规范。OSGi 的优点是支持模块动态加载、卸载、启动和停止等行为，尤其适合需要热插拔的模块化大型项目。ODL 作为一个网络操作系统平台，基于 OSGi 框架开发可以实现灵活的模块加载和卸载等操作，而无须在对模块进行操作时重启整个控制器。基于 ODL 开发模块时，还需要使用 YANG 语言来建模，然后使用 YANG Tools 生成对应的 Java API，并与其他插件代码共同完成服务实现。

最新版 ODL 安装包的官网下载地址为：

https://docs.opendaylight.org/en/latest/downloads.html

由于 ODL 依赖于 Java 平台，理论上可以运行在任何支持 Java 的平台上，因此还需安装 Java-JDK，官网下载地址为：

https://www.oracle.com/java/technologies/javase-downloads.html

2、ODL 模块管理

ODL 是一个模块化的 SDN 控制器平台，模块之间可以由依赖关系，如图 3 所示。ODL 中也将模块称为“项目”或“组件”。ODL 中的关键模块如下：

- odlparent：是所有模块的父模块，必须存在。
- DLUX：为控制器的使用者提供新的交互式 Web UI 应用。
- L2Switch：将 L2 的具体处理代码分离出来的一个独立模块，提供基本的 L2 交换机功能并创建一些可重用的服务，如基本生成树协议、地址跟踪等。
- OpenFlow plugin：主要实现 OpenFlow 1.0、1.3 及后续版本的协议规范支持。
- Controller：主要为多厂家网络的 SDN 部署提供一个高可用、模块化、可扩展、支持多协议的控制器基础架构。

- YANG Tools：在 ODL 中主要对 MD-SAL、NETCONF、OFConfig 应用插件提供 YANG 模型化语言支持。

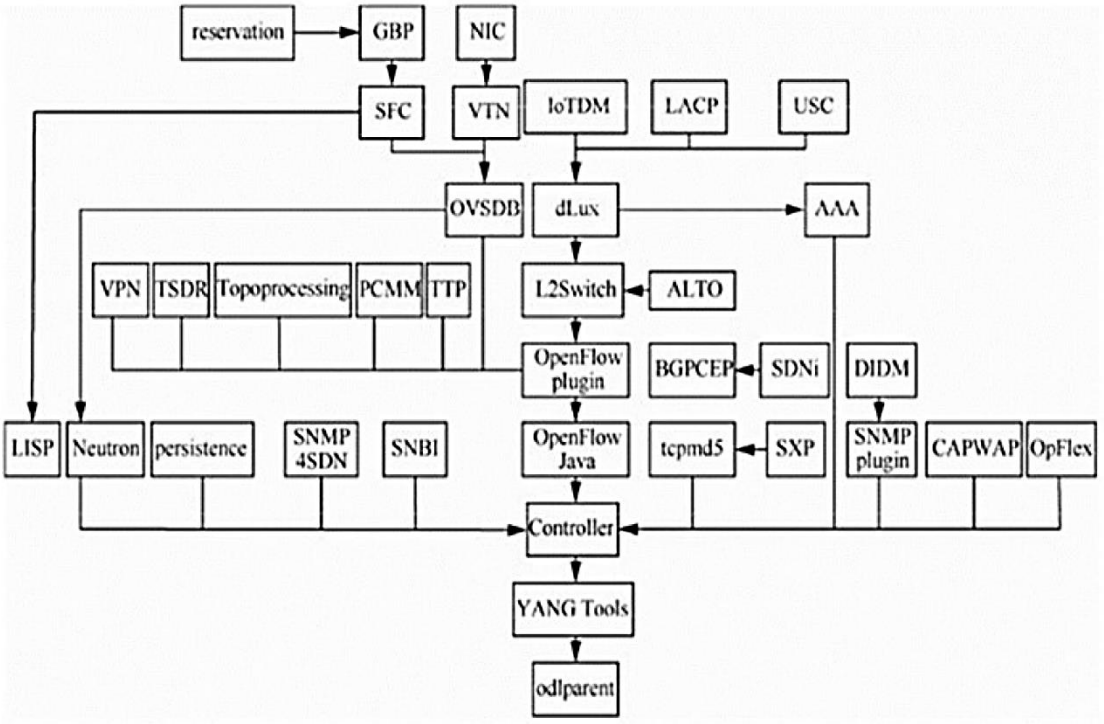


图 3 ODL 中模块的依赖关系

ODL 使用 Karaf 作为与用户交互的控制台。用户可以在 Karaf 控制台中对 ODL 平台的众多功能模块进行管理，通过使用 feature 命令实现对模块的查询、安装、移除功能。

- 安装模块：命令示例中的 xxx 为需要安装的模块名称

```
> feature:install xxx
```

- 查询所有模块

```
> feature:list
```

- 查询已安装的模块

```
> feature:list -i
```

- 移除已安装的模块

建议删除 ODL 安装目录下的 data 目录，然后执行如下命令后重启 ODL 即可

```
# ./bin/karaf clean
```

3、DLUX 模块：Web-UI 界面

DLUX 模块负责提供交互式 Web-UI 用户界面，用户通过这个界面可以查看拓扑、查看或配置流表信息，也可以配置 SDN 交换机的网桥和端口等信息。

在 ODL 中，DLUX 模块包括下面几个小模块：

- odl-dlux-core：Web-UI 的核心部分，负责页面数据的转换与处理；
- odl-dlux-node：Web-UI 用户获取拓扑节点信息进行图形化显示的模块；
- odl-dlux-yangui：启动 YANG UI 的模块。

在安装完 DLUX 模块后即可通过网页浏览器访问 Web-UI。Web-UI 的网址是 http://*{controll-ip-address}*:8181/index.html。

默认的 Web-UI 需要输入用户名和密码才能进一步查看信息或操作配置。默认的用户名和密码均为 admin。

ODL 的 Web-UI 登录页面如图 4 所示。

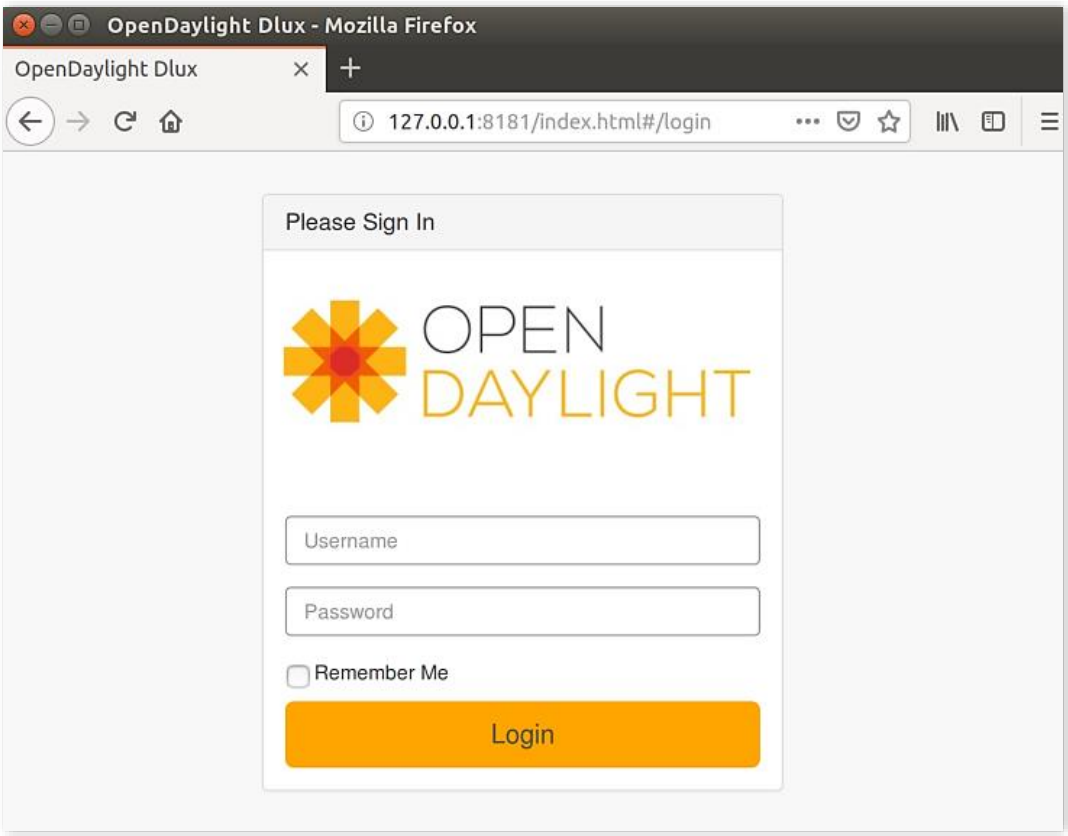


图 4 ODL 的 Web-UI 登录页面

登录后的页面（如图 5 所示）默认显示当前获取到的 SDN 链路拓扑图。页面左侧的导航栏包含以下页面链接：

- Topology：实时拓扑图；
- Nodes：已接入的 OpenFlow 设备的节点信息；
- Yang UI：查询和配置 YANG 模型的界面；
- Yang Visualizer：YANG 模型的可视化界面。

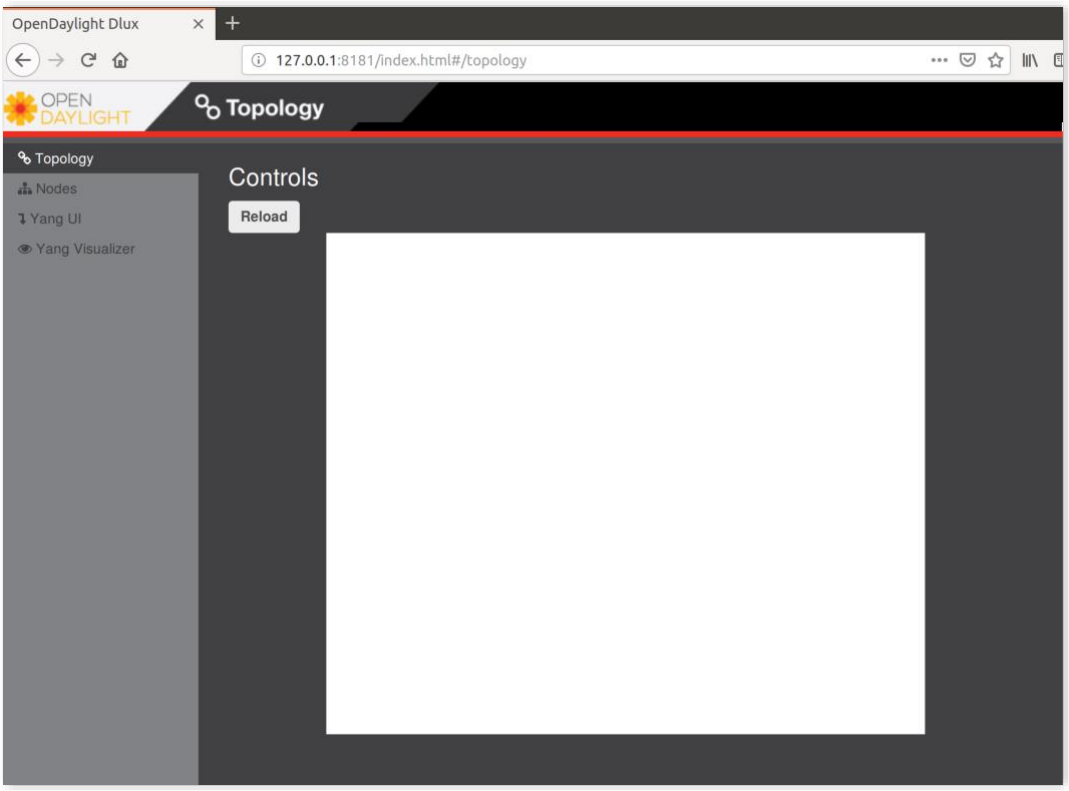


图 5 登录 Web-UI 后的首页

（1）Topology 页面

主要用于显示检测到的拓扑信息，主要是交换机、主机及其连接，控制器不会显示在拓扑中，例如图 6。

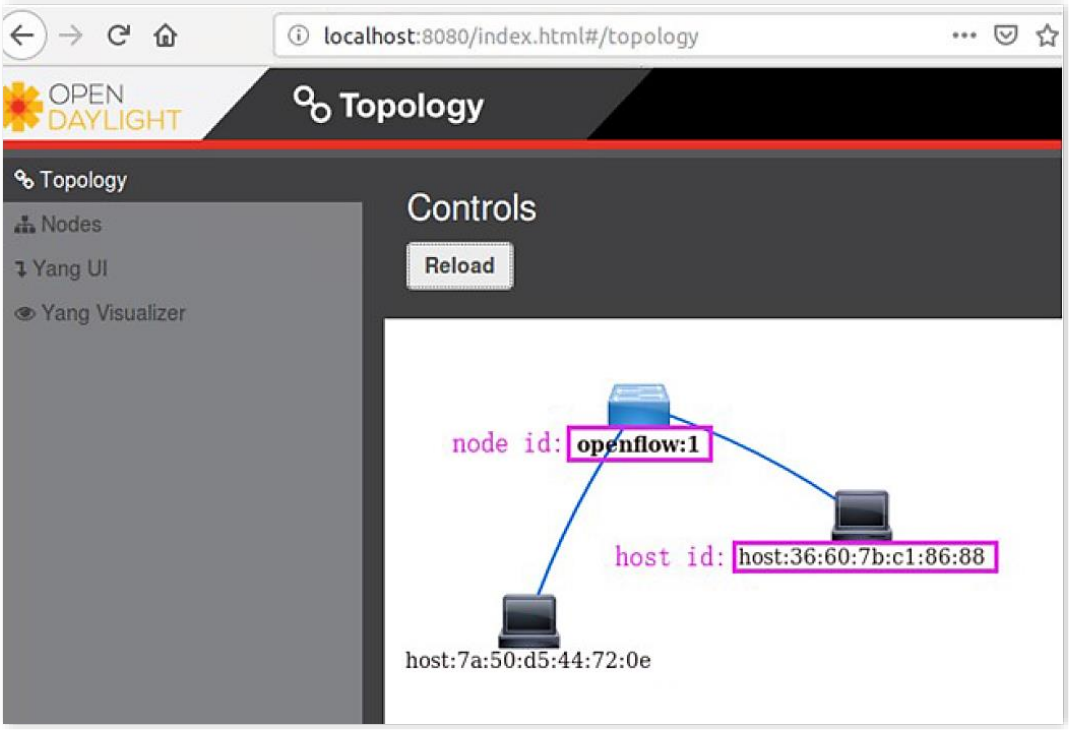


图 6 Topology 页面

如果 OpenFlow 交换机连接主机的链路没有交换流量，那么拓扑图中只会显示连接到控制器的 OpenFlow 交换机。当主机连接交换机的链路上有流量（主机发出或发往主机）经过时，需通过页面上 Controls 下方的“Reload”按钮才可以刷新显示出来。

用户可以在此页面上获取交换机 ID，即 node id。

（2）Nodes 页面

以列表形式存放拓扑中的交换机信息，包括 Node Id（交换机 ID）、Node Name（交换机名）、Node Connectors（交换机当前的连接数量）、Statistics（交换机统计信息，包括流表统计和端口统计数据）。用户可以通过点击表中的超链接（蓝色字体）查看详细信息，如图 7 所示。

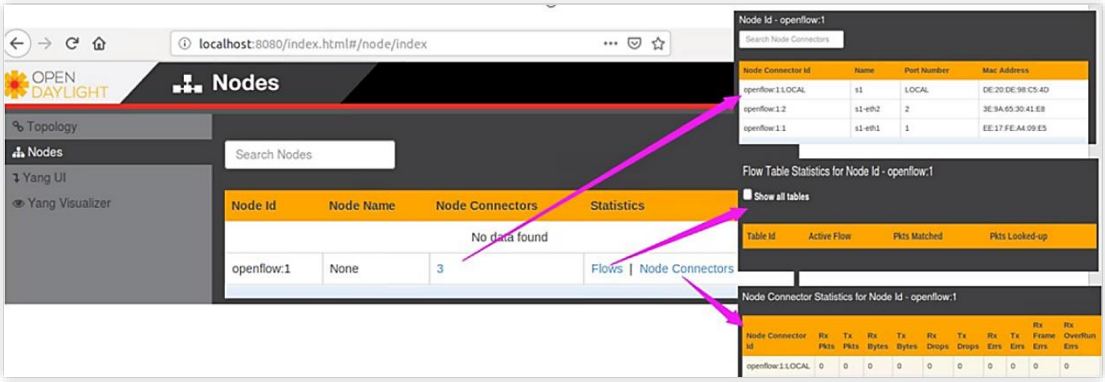


图 7 Nodes 页面

图 8 Yang UI 页面

（3）Yang UI 页面

此页面是 ODL 北向接口的集中操作区。如图 8 所示，Yang UI 页面包含如下区域：

- Module 区域：显示基本 ODL 安装的所有模块的 API 接口；
- 动作区域：显示 API 路径可操作的方法和按钮；
- 状态区域：显示当前操作的状态；
- API 参数填写区域：用户更改数据的集中区域。

（4）Yang Visualizer 页面

Yang UI 的可视化页面，用户可在此页面图形化的查看 Model 信息，如图 9 所示。

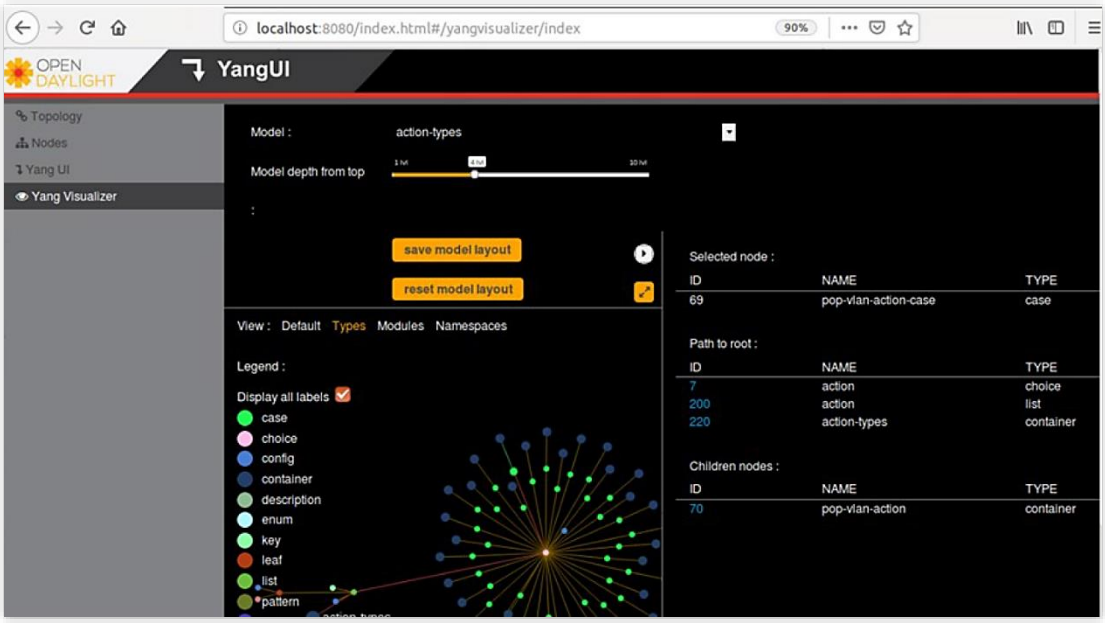


图 9 Yang Visualizer 页面

4、RESTCONF

YANG 模型用于 MD-SAL 和基于 MD-SAL 的应用程序中，用于定义所有 API：组件间 API、插件 API、北向 API 等。YANG 模型在编译时使用 ODL Yang Tools 生成 Java API，根据 RESTCONF 规范在运行时呈现北向接口 REST APIs。简单来说，RESTCONF 是 ODL 的北向 API，是负责为用户提供网络配置与管理的 API 接口。

RESTCONF 基于 HTTP 协议，运行 REST 协议访问 YANG 模型中定义的数据，默认在 8181 端口上监听 HTTP 请求。用户可以通过以下命令查看 RESTCONF 相关组件：

```
> feature:list |grep odl-restconf
```

通过以下命令安装 RESTCONF 所有功能：

```
> feature:install odl-restconf-all
```

RESTCONF 安装完毕后，允许使用的数据存储区域有两种：

- config 数据存储区：包含通过控制器插入的数据，主要用于变更或插入数据。
- operational 数据存储区：包含其他数据信息，主要用于查看已有的数据。

RESTCONF 支持的数据类型为 XML 或 JSON 格式，通过 HTTP 头部的“Content-Type”字段定义请求的输入代码格式，通过 HTTP 头部的“Accept”字段定义响应的输出代码格式。

RESTCONF 支持 OPTIONS、GET、PUT、POST、DELETE、PATCH 等 HTTP 操作。调用 RESTCONF 时，需要使用实例标识符，它必须以<moduleName>:<nodeName>开头，其中<moduleName>是 YANG 模块的名称，<nodeName>是模块中顶级节点的名称。

ODL 锂版本中的 RESTCONF 提供的 Yang UI 模块位于 Yang UI 界面的 API 列表中，其中有一个 OpenDaylight-inventory.rev.2013-08-19，提供了 operational 和 config 两个数据存储区域，分别点开即可看见 Yang UI 下可操作何查看的 API，常用的是 nodes 下的 node{id}下的 table{id}。Operational 区域内存储的数据仅可通过 GET 方法读取；config 区域内存储的数据可通过 GET 方法读取，或通过 PUT、POST 方法写入。

5、Postman

Postman 是 Google 开发的一款功能强大的网页调试与发送网页 HTTP 请求的 Chrome 插件，能运行测试用例，至今已发展成为一款测试和调试网页的强大软件。Postman 可以运行在 Windows 或 Linux 操作系统中。

在 Ubuntu 中安装好 Postman，并更改环境变量来加载 postman 命令后（详见实验 1-SDN 实验环境构建），即可直接在命令行终端输入如下命令启动 Postman：

```
$ sudo postman
```

注册 Postman 账号是免费的，注册并登录 Postman 账号后，可以跨设备同步数据和备份数据，可以分享和管理接口用例集。当然，不注册也可以使用 Postman。具体注册方法参看下面这篇博客文章：

https://www.cnblogs.com/Chamberlain/p/10834699.html

Postman 界面结构和各部分功能的详细信息可参看下面这篇博客文章：

https://www.cnblogs.com/Chamberlain/p/10817198.html

还可以直接在 CSDN 中（https://www.csdn.net/）搜索查阅更多 Postman 使用的相关博客文章。

实验拓扑

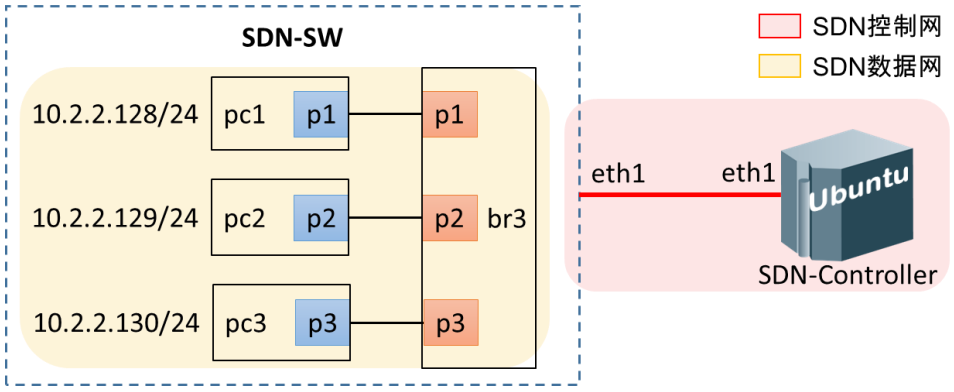


图 10 实验 3 拓扑

实验 3 拓扑中各个虚拟机的主机名、网络连接模式和 IP 地址如表 1 所示。

表 1 实验 3 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-Controller	网卡 1	NAT	NAT 网络	VMnet8	eth0：DHCP（172.16.1.0/24）
	网卡 2	内部网络	intnet1	LAN1	eth1：10.1.1.10/24，无 GW
SDN-SW	网卡 1	NAT	NAT 网络	VMnet8	eth0：DHCP（172.16.1.0/24）
	网卡 2	内部网络	intnet1	LAN1	eth1：10.1.1.20/24，无 GW
PC1 (虚拟网络空间)	Internal 类型的虚拟网卡 p1				p1：10.2.2.128/24，无 GW
PC2 (虚拟网络空间)	Internal 类型的虚拟网卡 p2				p2：10.2.2.129/24，无 GW
PC3 (虚拟网络空间)	Internal 类型的虚拟网卡 p3				P3：10.2.2.130/24，无 GW

考虑物理电脑的性能问题，本实验仅使用 SDN-Controller 和 SDN-SW 这两台虚拟机，在 SDN-SW 虚拟机中创建三个虚拟网络空间作为 PC1、PC2 和 PC3。

实验 3.1 OpenDaylight 控制器的安装和启动

ODL 控制器的安装主要基于 Java 平台的构建，从 ODL 锂版本开始支持 Java-JDK-1.8 或更高的版本。本次实验我们将使用 ODL 锂版本（也可以自行基于网上的相关 ODL 安装教程去尝试安装其他版本的 ODL）。

下面是在 SDN-Controller 虚拟机中安装 ODL 锂版本的具体步骤。

步骤 1、获取并解压源代码包

本实验提供的 Java-JDK 安装包是 jdk-8u211-linux-x64.tar.gz，也可以使用从 Java-JDK 官网下载的安装包。Java-JDK 官网下载地址为：

https://www.oracle.com/java/technologies/javase-downloads.html

ODL 锂版本的安装包是 distribution-karaf-0.3.0-Lithium.tar.gz，也可以使用从 ODL 官网下载的 ODL 安装包。ODL 官网下载地址为：

https://docs.opendaylight.org/en/latest/downloads.html

因为后续 Java-JDK 和 ODL 的安装配置都需要 root 权限，所以先直接切换为 root 用户身份：

```
$ sudo – root
```

进入安装包的存放目录进行解压。本实验中的命令示例是解压存放在/mnt 目录下，也可选择解压到其他目录下：

```
# tar -xvf jdk-8u211-linux-x64.tar.gz -C /mnt
# tar -xvf distribution-karaf-0.3.0-Lithium.tar.gz -C /mnt
```

注意：如果物理主机的存储空间不充足，安装完 Java-JDK 和 ODL 后，请删除压缩的安装包。

步骤 2、配置 Java-JDK 环境变量

解压后的 Java-JDK 不需要编译安装，只需配置环境变量。

Java-JDK 的环境变量主要声明了 Java 平台的 home 目录和 JRE（Java 运行环境）的路径位置。分别编辑/etc/profile 文件和/root/.bashrc 文件：

```
# vim /etc/profile
# vim /root/.bashrc
```

说明：如果不在/root/.bashrc 文件中添加 Java-JDK 环境变量，那么每次重启虚拟机后，通过 su 命令切换为 root 用户身份时，都需要使用 source 命令加载 Java-JDK 环境变量。

在这两份文件的末尾处都输入以下内容，然后保存并退出：

```
# JAVA-SET
JAVA_HOME=/mnt/jdk1.8.0_211
JRE_HOME=$JAVA_HOME/lib
PATH=$JAVA_HOME/bin:$PATH
export JAVA_HOME JRE_HOME PATH
```

步骤 3、加载 Java-JDK 环境变量并验证

使用如下命令加载 Java-JDK 环境变量，并验证 Java 平台是否生效：

```
# source /etc/profile
# java -version
```

显示 Java 平台生效的信息如下所示。

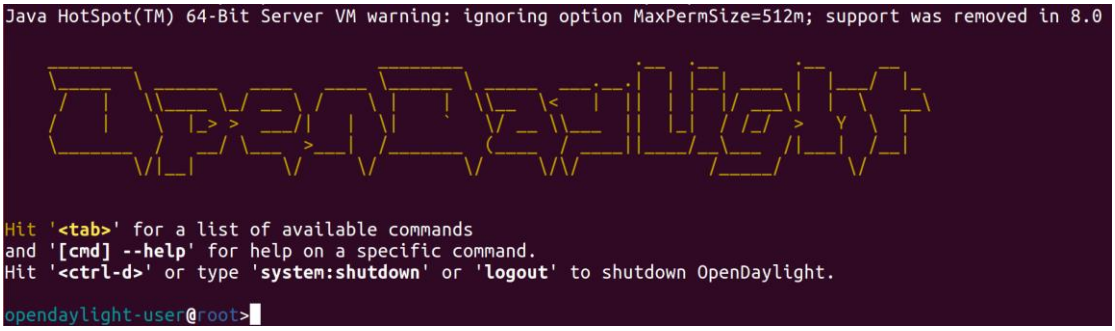
```
java version "1.8.0_211"
Java(TM) SE Runtime Environment (build 1.8.0_211-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.211-b12, mixed mode)
```

步骤 4、启动 ODL

Java 平台生效后，即可执行 ODL 启动程序，命令如下：

```
# cd /mnt/distribution-karaf-0.3.0-Lithium
# ./bin/karaf
```

默认情况下，ODL 启动完成进入如下所示的 karaf 控制台：



注意：如果从 karaf 控制台注销（logout），即会停止 ODL 服务。

步骤 5、安装 ODL 模块

首次启动 karaf 控制台时，ODL 控制器仅包含以下模块，并不具备 SDN 控制器功能：

```
opendaylight-user@root>feature:list -i
Name        | Version | Installed | Repository     | Description
------------------------------------------------------------------------------------------------
standard    | 3.0.3   | x         | standard-3.0.3 | Karaf standard feature
config      | 3.0.3   | x         | standard-3.0.3 | Provide OSGi ConfigAdmin support
region      | 3.0.3   | x         | standard-3.0.3 | Provide Region Support
package     | 3.0.3   | x         | standard-3.0.3 | Package commands and mbeans
kar         | 3.0.3   | x         | standard-3.0.3 | Provide KAR (KARaf archive) support
ssh         | 3.0.3   | x         | standard-3.0.3 | Provide a SSHd server on Karaf
management  | 3.0.3   | x         | standard-3.0.3 | Provide a JMX MBeanServer and a set of MBeans in K
```

因此，需要执行以下命令安装 ODL 的必要模块：

```
> feature:install odl-restconf
> feature:install odl-l2switch-switch
> feature:install odl-openflowplugin-all
> feature:install odl-dlux-all
> feature:install odl-mdsal-all
> feature:install odl-adsal-northbound
```

注意 1：请严格按照上述顺序安装，安装顺序不合理的话会导致无法访问 Web UI 界面。

注意 2：对于不同版本的 ODL 控制器，feature:install 命令安装的模块名会有所不同，可以通过命令 feature:list 查看所有的模块列表。例如：

```
opendaylight-user@root>feature:list |grep odl-adsal
odl-adsal-northbound                          | 0.5.0-Lithium       |
t :: AD-SAL :: Northbound APIs
odl-adsal-all                                 | 0.9.0-Lithium       |
t AD-SAL All Features
odl-adsal-core                                | 0.9.0-Lithium       |
t :: AD-SAL :: Core
odl-adsal-networkconfiguration                | 0.1.0-Lithium       |
t :: AD-SAL :: Network Configuration
odl-adsal-connection                          | 0.2.0-Lithium       |
t :: AD-SAL :: Connection
odl-adsal-clustering                          | 0.6.0-Lithium       |
t :: AD-SAL :: Clustering
odl-adsal-configuration                       | 0.5.0-Lithium       |
t :: AD-SAL :: Configuration
odl-adsal-thirdparty                          | 0.9.0-Lithium       |
t :: AD-SAL :: Third-Party Depenencies
```

每个模块的安装时间长短不一，有些模块在安装过程中没有任何提示或输出信息。因此在执行 **feature:install** 命令后，等到出现 **karaf** 控制台的命令提示符（**opendaylight-user@root>**）时，即表示该模块安装成功。

说明：如果实验中 ODL 出现错误导致不能正常运行时，简单的解决方案是删除 ODL 安装包的解压目录，然后重新解压 ODL 安装包，并重新安装 ODL 模块。

因退出 **karaf** 控制台或重启 **SDN-Controller** 虚拟机而再次启动 ODL 时，需使用如下命令检测 ODL 是否已监听在默认的 TCP 端口 6633 上：

```
$ su root
# netstat –anp|grep 6633
```

如果该命令返回下面的内容，则表示 ODL 启动完毕，否则需继续等待 ODL 完成其启动过程。

```
tcp6       0      0 :::6633                 :::*                    LISTEN
```

实验 3.2 创建 OVS 网桥并连接 PC1、PC2 和 PC3

步骤 1、启动 SDN-SW 虚拟机，删除所有已配置的网桥

```
# ovs-vsctl del-br <已配置的网桥名>
```

步骤 2、创建网桥 br3 及其 Internal 端口

使用 ovs-vsctl 命令创建一个名为 br3 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。

具体命令如下：

```
# ovs-vsctl add-br br3
# ovs-vsctl add-port br3 p1 -- set Interface p1 type=internal
# ovs-vsctl add-port br3 p2 -- set Interface p2 type=internal
```

```
# ovs-vsctl add-port br3 p3 -- set Interface p3 type=internal
# ovs-vsctl show
```

使用 ovs-vsctl 命令创建一个名为 br3 的网桥，创建启动 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：ovs-vsctl show 命令及其结果截图。

步骤 3、查看 br3 的流表项

使用 ovs-ofctl 命令查看 br3 的所有流表项。

记录：查看 br3 流表项的 ovs-ofctl 命令及其结果截图。

步骤 4、创建三个虚拟网络空间并连接网桥 br3

三个虚拟网络空间的名字分为为 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 br3 的命令。

说明：如果实验中重启 SDN-SW 虚拟机，步骤 2 中创建的 br3 网桥配置仍然存在，但必须重新执行步骤 4。

步骤 5、测试 pc1、pc2 和 PC3 之间的通信

执行如下命令进行通信测试：

```
# ip netns exec pc1 ping -c 4 10.2.2.129
# ip netns exec pc1 ping -c 4 10.2.2.130
# ip netns exec pc2 ping -c 4 10.2.2.130
```

记录：以上三个命令及其结果截图。

步骤 6、再次查看 br3 的流表项

在 SDN-SW 上使用 ovs-ofctl 命令查看 br3 的所有流表项。

记录：查看 br3 流表项的 ovs-ofctl 命令及其结果截图。

实验 3.3 配置 br3 连接 ODL 并进行 PC 通信测试

步骤 1、将网桥 br3 连接到 ODL 控制器上

确保 SDN-SW 虚拟机能 ping 通 SDN-Controller 虚拟机的 eth1 接口 IP 地址（10.1.1.10），即本实验的 SDN 控制网时连通的。

在 SDN-SW 虚拟机中输入以下命令，将 br3 连接到 ODL 控制器上，并将网桥 br3 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-controller br3 tcp:10.1.1.10:6633
# ovs-vsctl set-fail-mode br3 secure
```

然后使用 ovs-vsctl show 命令查看 br3。如果命令返回结果中显示 Controller 的连接状态是 true（即 is_connected: true），则说明 br3 与 ODL 之间的 OpenFlow 连接已经建立。此时，使用 ovs-ofctl 命令查看 br3 的所有流表项。

记录：SDN-SW 虚拟机上的 ovs-vsctl show 命令及其结果截图，以及查看 br3 流表项的 ovs-ofctl 命令及其结果截图。

在 SDN-Controller 虚拟机中打开浏览器访问 ODL 的 Web-UI，网址如下：http://127.0.0.1:8181/index.html。

ODL 的 Web-UI 默认用户名和密码均为 admin。

在 ODL 的 Web-UI 界面中查看 Topology 页面和 Nodes 页面，并查看 Nodes 页面所示的 OpenFlow 交换机信息表中 Node Connectors 列和 Statistics 列中的详细信息（即这两列中的所有超链接）。

说明：如果 Topology 页面上没有显示连接到 ODL 控制器的 OpenFlow 交换机，请点击页面上 Controls 下方的“Reload”按钮刷新一下。

记录：SDN-Controller 虚拟机上的 Topology 页面截图、Nodes 页面截图、Node Connectors 列中详细信息截图、Statistics 列中的 Flows 详细信息截图（选中“Show all tables”，第一屏的截图即可）、Statistics 列中的 Node Connectors 详细信息截图。

步骤 2、测试 pc1 与 pc2 之间的通信

在 SDN-SW 虚拟机中执行如下命令进行通信测试：

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

Ping 结束后，使用 ovs-ofctl 命令查看 br3 的所有流表项。

记录：SDN-SW 虚拟机上的 ping 命令及其结果截图、查看 br3 流表项的 ovs-ofctl 命令及其结果截图。

在 SDN-Controller 虚拟机上，通过 ODL Web-UI 界面查看 ODL 当前获得的拓扑，即点击 Topology 页面上的“Reload”按钮。

记录：SDN-Controller 虚拟机上的 Topology 页面截图。

步骤 3、测试 pc1 与 pc3 之间的通信

在 SDN-SW 虚拟机中执行如下命令进行通信测试：

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

Ping 结束后，使用 ovs-ofctl 命令查看 br3 的所有流表项。

记录：SDN-SW 虚拟机上的 ping 命令及其结果截图、查看 br3 流表项的 ovs-ofctl 命令及其结果截图。

在 SDN-Controller 虚拟机上，通过 ODL Web-UI 界面查看 ODL 当前获得的拓扑，即点击 Topology 页面上的“Reload”按钮。

记录：SDN-Controller 虚拟机上的 Topology 页面截图。

步骤 4、测试 pc2 与 pc3 之间的通信

在 SDN-SW 虚拟机中执行如下命令进行通信测试：

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

Ping 结束后，使用 ovs-ofctl 命令查看 br3 的所有流表项。

记录：SDN-SW 虚拟机上的 ping 命令及其结果截图、查看 br3 流表项的 ovs-ofctl 命令及其结果截图。

在 SDN-Controller 虚拟机上，通过 ODL Web-UI 界面查看 ODL 当前获得的拓扑，即点击 Topology 页面上的“Reload”按钮。

记录：SDN-Controller 虚拟机上的 Topology 页面截图。

步骤 5、根据实验记录回答以下问题（Q1~Q6）

Q1、步骤 1 中 br3 的流表项与实验 3.2 步骤 6 中的流表项有哪些不同？请尝试分析本步骤中每条流表项的作用（即匹配哪种特征的分组？做出什么动作？）。

Q2、相比于步骤 1，步骤 2 的 br3 新增了哪几条流表项？这些流表项是通过哪种类型的 OpenFlow 报文生成的？请分析本步骤中新增的每条流表项的作用（即匹配哪种特征的分组？做出什么动作？）。

Q3、相比于步骤 1，步骤 2 的 ODL 拓扑图中新增了哪些拓扑信息？ODL 是通过哪种类型的 OpenFlow 报文获得这些拓扑信息的？

Q4、相比于步骤 2，步骤 3 的 br3 新增了哪几条流表项？这些流表项是通过哪种类型的 OpenFlow 报文生成的？请分析本步骤中新增的每条流表项的作用（即匹配哪种特征的分组？做出什么动作？）。

Q5、相比于步骤 2，步骤 3 的 ODL 拓扑图中新增了哪些拓扑信息？ODL 是通过哪种类型的 OpenFlow 报文获得这些拓扑信息的？

Q6、相比于步骤 3，步骤 4 的 br3 新增了哪几条流表项？这些流表项是通过哪种类型的 OpenFlow 报文生成的？请分析本步骤中新增的每条流表项的作用（即匹配哪种特征的分组？做出什么动作？）。

说明：如果物理主机的性能允许的话，实验 3.3 的步骤 1~步骤 4 中可以在 SDN-Controller 虚拟机上对 eth1 口（SDN 控制网络）进行 tcpdump 抓包并保存为后缀为.pcap 的文件，通过 Wireshark 查看.pcap 文件来帮助理解 OVS 交换机和 ODL 控制器之间 OpenFlow 协议交互。

实验 3.4 使用 Yang UI 工具下发 L3 流表项

L3 流表项要求如下：

- 流 ID 为 1，流表项永不过期，流表项优先级为 25；
- 丢弃（drop）源 IP 地址为 PC1、目的 IP 地址为 PC2 的所有 IP 分组。

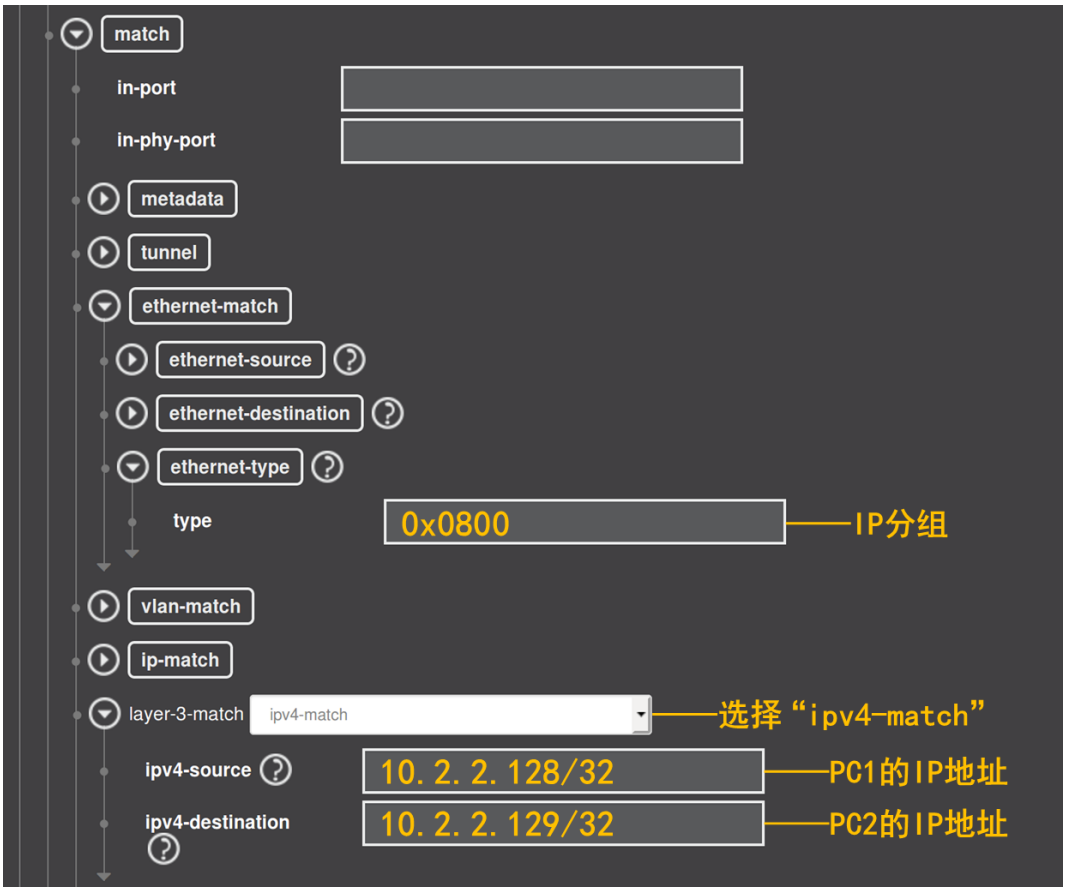
步骤 1、在 Yang UI 中添加一条 L3 流表项

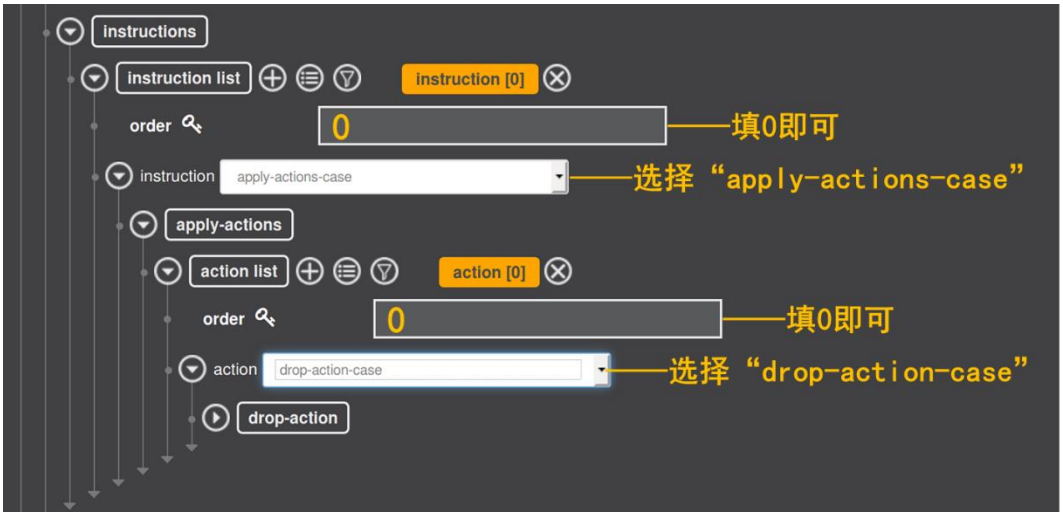
在 SDN-Controller 虚拟机上进入 ODL Yang UI 页面，待模块加载完后依次展开“opendaylight-inventory rev.2013-08-19→config→nodes→node {id}→table {id}”。按照图所示，首先填入 br3 的 Node ID（即实验 3.3 步骤 1 的 Nodes 页面截图中 OVS 交换机的 Node Id 值“openflow:xxx”），然后点击页面下方“table list”旁的⊕符号，即可开始添加流表。

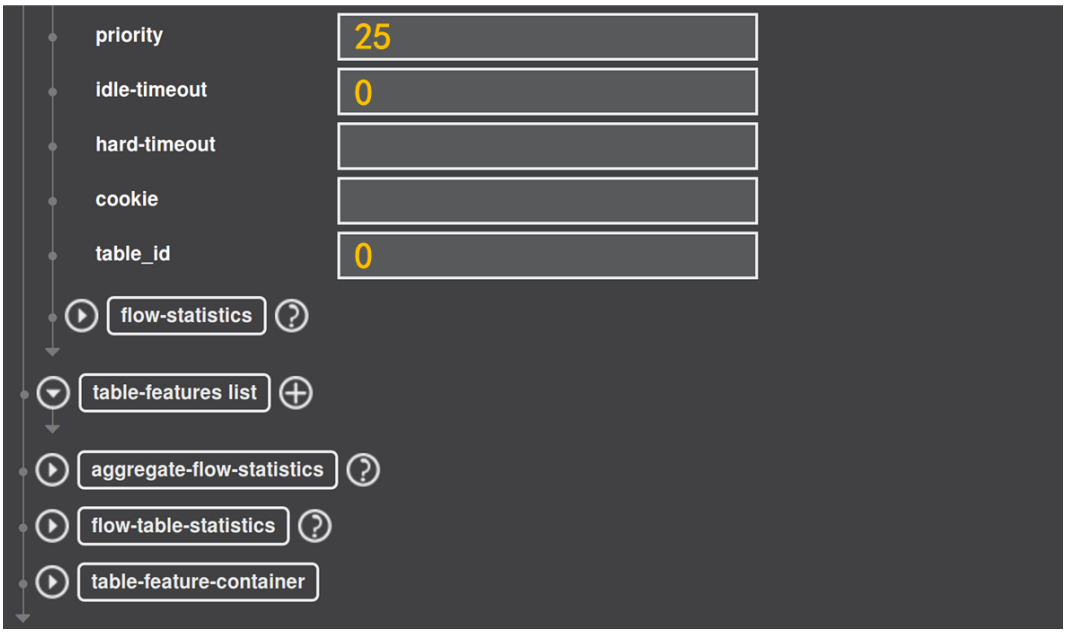
添加本步骤要求的 L3 流表项时，必须填写以下参数：

- 表 ID、流 ID

- match 域
  - ethernet-match
    - ethernet-type
  - layer-3-match→ipv4-match
    - ipv4-source、ipv4-destionation
- instructions 域
  - instructions order
  - instruction→apply-actions-case
    - action order
    - action→drop-action-case
- priority、idle-timeout、table_id







步骤 2、下发 L3 流表项

填写好流表项后，在 Yang UI 的动作区域（Actions）中选择“PUT”方法，点击“Send”按钮即完成流表下发操作。

图 6 运行 Wireshark 时的错误提示

步骤 3、验证 L3 流表项

在 SDN-SW 虚拟机上查看 br3 的所有流表项。

记录：SDN-SW 虚拟机上查看 br3 流表项的 ovs-ofctl 命令及其结果截图。

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：以上命令及其结果截图。

步骤 4、根据实验记录回答以下问题（Q7~Q9）

请对比 ping 测试前后 br3 中各流表项的 n_packets 值：

Q7、步骤 3 的 PC1 ping PC2 时，PC1 发出的 ping 请求报文匹配 br3 上的哪一条流表项？br3 如何处理这些 ping 请求报文？请给出依据截图。

Q8、如果是 PC2 ping PC1，其 ping 报文交互结果与步骤 3 有哪些不同？请说明理由。

Q9、如何能让 PC2 ping PC1 的报文交互结果与步骤 3 一样？

实验 3.5 使用 Postman 工具下发 L4 流表项

L4 流表项要求如下：

- 仅一条流 ID 为 2 的流表项，流表项永不过期，流表项优先级为 30；
- pc1 能作为 iperf 客户端向 pc2 正常发起 TCP 性能测试；
- 实验 3.4 中下发的 L3 流表项仍有效。

步骤 1、在 Yang UI 中编辑一条 L4 流表项

使用 Postman 调用 ODL 的 RESTCONF 进行流表项下发时，发送的是流表项的 JSON 消息。在不熟悉 OpenFlow 流表项的 JSON 消息格式时，可以利用 ODL 的 Yang UI 编辑 OpenFlow 流表项，然后再获取其 JSON 消息。

编辑流表项的具体操作与实验 3.4 的步骤 1 相同，只是需要按照实验 3.5 的 L4 流表项要求填写相应参数。本步骤要求的 L4 流表项必须填写以下参数（其中红字表示与实验 3.4 步骤 1 不同的参数）：

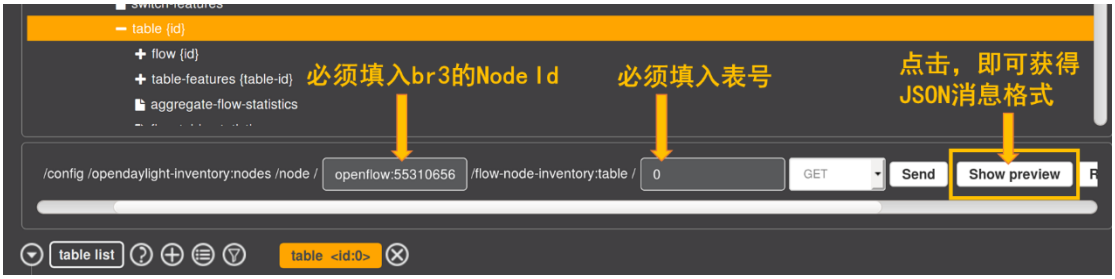
- 表 ID、流 ID
- match 域
  - ethernet-match
    - ethernet-type
  - ip-match
    - ip-protocol = 6 ——表示 TCP 协议
  - layer-3-match→ipv4-match
    - ipv4-source、ipv4-destionation
  - layer-4-match→tcp-match

- tcp-destionation-port = 5001 ——iperf 服务器端口号

- instructions 域
  - instructions order
  - instruction→apply-actions-case
    - action order
    - action→output-action-case
      - output-node-connector = 2 ——br3 连接 PC2 的端口
- priority、idle-timeout、table_id

步骤 2、获取 L4 流表项的 JSON 消息

完成步骤 1 后，在 Yang UI 的动作区域（Actions）中点击“Show preview”按钮以获取步骤 1 中编辑的 L4 流表项的 JSON 消息。



点击“Show preview”按钮后会弹出一个如下所示的 Preview 窗口：

复制 Preview 窗口中的所有信息。

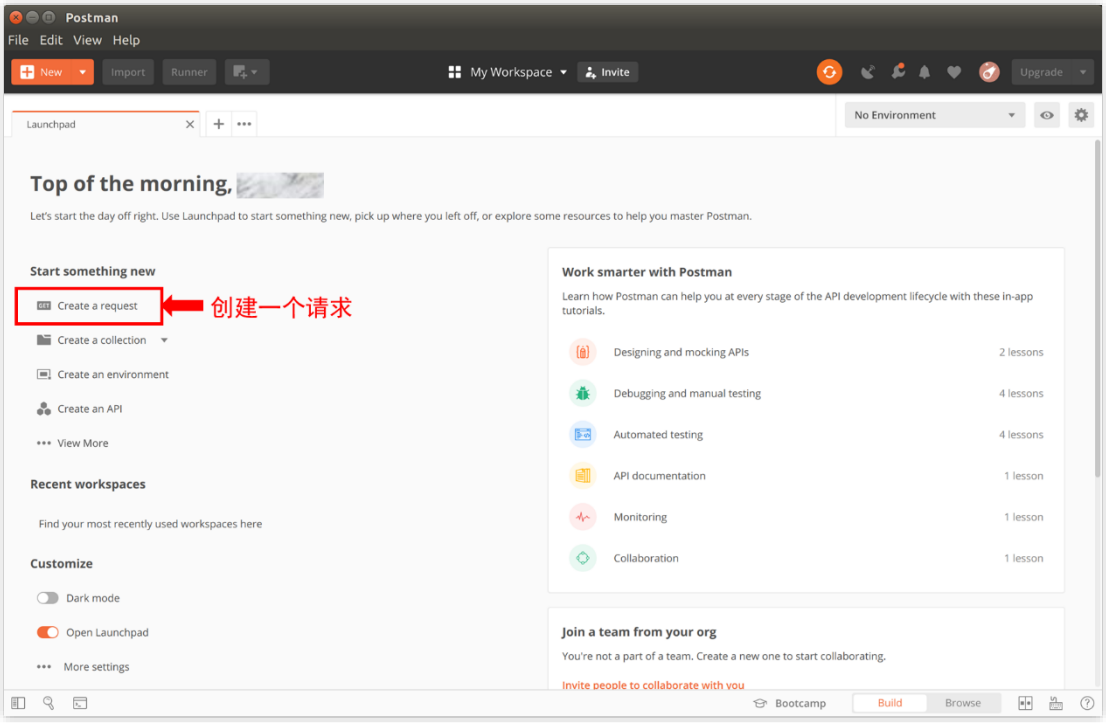
步骤 3、使用 Postman 下发 L4 流表项

在 SDN-Controller 虚拟机中，打开一个命令行终端直接输入如下命令启动 Postman：

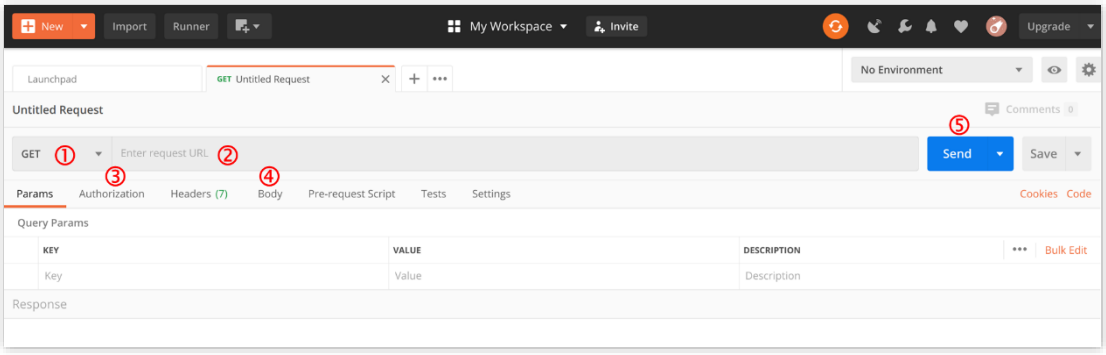
```
$ sudo postman
```

也可以在其他安装有 Postman 工具、且能通过网络访问 ODL 的 Web UI 界面（http://<ODL-ip-address>:8181/）的主机上运行 Postman 工具进行流表项下发操作。

启动 Postman 后的初始页面如下：



直接点击该页面上的“Create a request”链接，进入如下所示的请求创建页面：



使用 Postman 下发实验 3.5 中的 L4 流表项，需要进行以下 5 项操作：

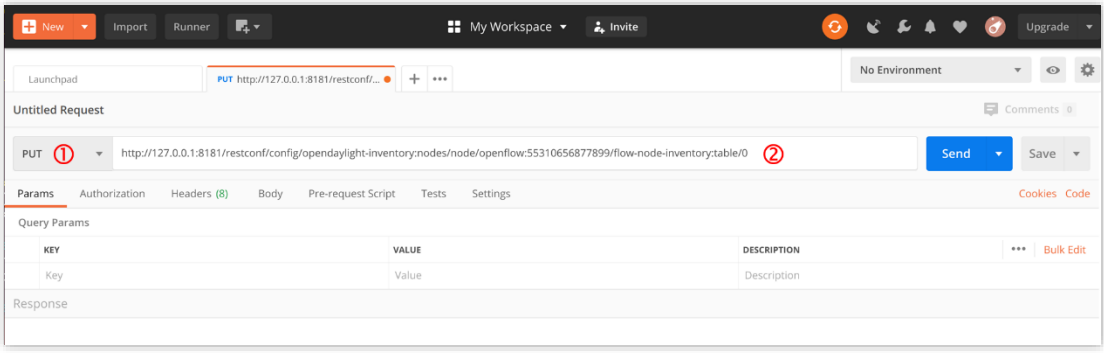
① 选择请求方法

实验 3.5 中的 HTTP 请求方法为“PUT”。

② 填写请求 URL

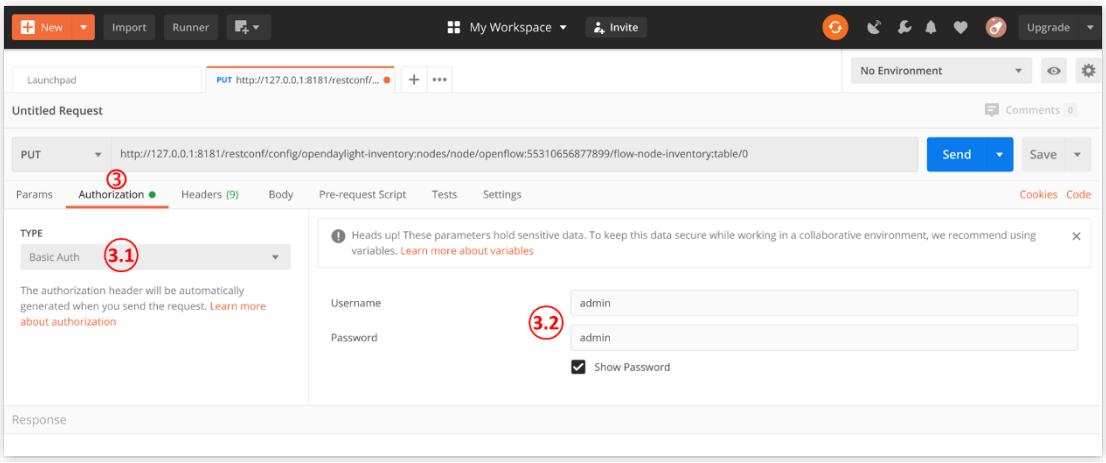
请求 URL 即为实验 3.5 步骤 2 的 Preview 窗口中的第一行内容，即以 http://

开头的那行文字。



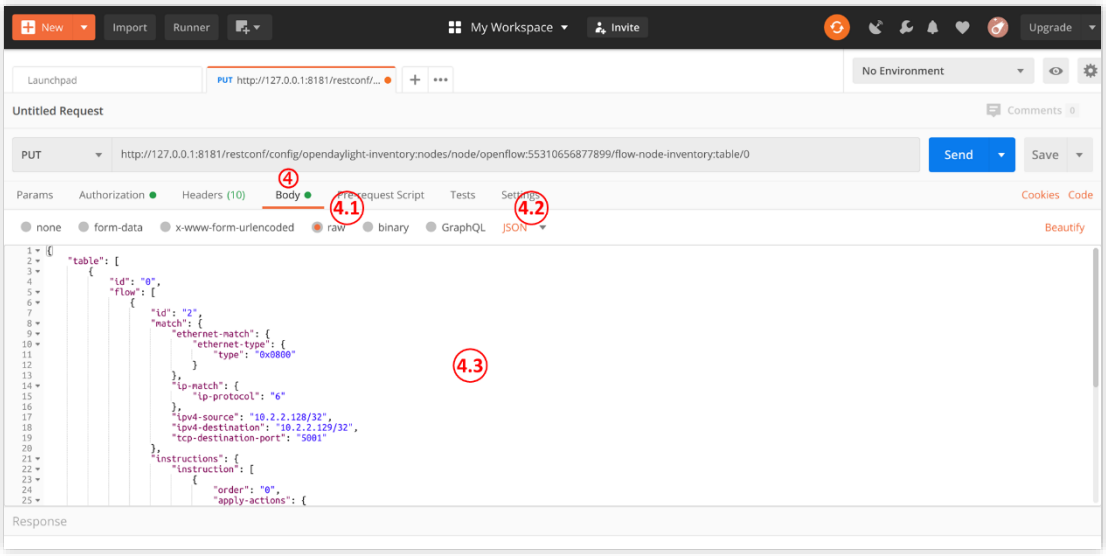
③ 填写 Authorization 标签

（3.1）类型（Type）选择“Basic Auth”;（3.2）用户名和密码均为“admin”。



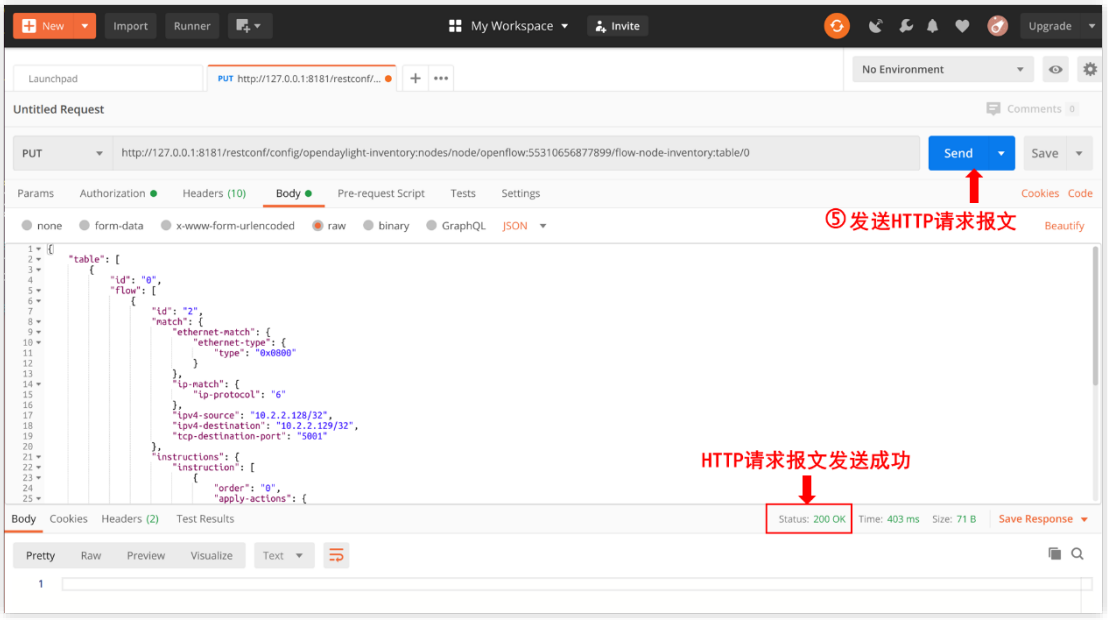
④ 填写 Body 标签

（4.1）点选“raw”;（4.2）选择“JSON”;（4.3）填入实验 3.5 步骤 2 的 Preview 窗口中的第二行开始的所有内容。



⑤ 发送 HTTP 请求报文

点击“Send”按钮。如果回应信息为下图所示“Status: 200 OK”，即表示 HTTP 请求报文发送成功。



步骤 4、验证 L4 流表项

在 SDN-SW 虚拟机上查看 br3 的所有流表项。

记录：SDN-SW 虚拟机上查看 br3 流表项的 ovs-ofctl 命令及其结果截图。

使用如下命令进行通信测试：

- pc2 启动 iperf 服务器端，监听在 TCP 5001 端口

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

- pc1 启动 iperf 客户端连接 PC2 的 ipfer 服务器进行 TCP 性能测试

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

- pc1 ping pc2

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

记录：以上命令及其结果截图。

步骤 5、根据实验记录回答以下问题（Q10~Q12）

请对比 ipfer 测试和 ping 测试前后 br3 中各流表项的 n_packets 值：

Q10、pc1 的 iperf 客户端发送给 pc2 的 iperf 服务器端的 TCP 性能测试数据匹配 br3 上的哪一条流表项？br3 如何处理这些测试数据？请给出依据截图。

Q11、pc2 的 iperf 服务器端返回的测试统计反馈数据匹配 br2 上的哪一条流表项？br3 如何处理这些测试统计反馈数据？请给出依据截图。

Q12、为什么 pc1 和 pc2 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 3.2~实验 3.5 中要求的“记录”和“问题”。