Open Daylight Controller 简易入门

最新版: yeasy@github

更新历史:
V0.1: 2013-08-08
完成所有内容。

第1章 基本配置

官方网站为 http://www.openflow.org/wk/index.php?title=OpenDayLight_Tutorial。参考了这里的介绍和其他材料，补充上了个人理解。

1.1 环境配置

下载已经预装好 ODL 和其它多个控制器的虚拟机：http://yuba.stanford.edu/~srini/tutorial/OpenFlow_tutorial_64bit.ova，用户名和密码均为 ubuntu。

如果想要自己编译和运行，可以自行下载代码后通过 CLI 或者 Eclipse。

1.1.1 获取源码

匿名用户下载代码，可以通过：

```
git clone https://git.Open Daylight.org/gerrit/p/controller.git
```

1.1.2 CLI

编译代码

```
cd opendaylight/distribution/opendaylight/
mvn clean install [-DskipTests]
```

运行编译好的程序

```
cd  target/distribution.opendaylight-0.1.0-SNAPSHOT-osgipackage/opendaylight/
./run.sh
```

执行后会打印输出各种调试信息，并且提供一个 osgi 的控制台。

1.1.3 Eclipse

使用 Eclipse 进行操作更为简便，主要是要先安装 maven 插件，然后导入项目。

导入项目选择 Maven->Existing Maven Projects，下一步之后定位到 opendaylight\distribution\opendaylight（主目录），会自动找到所有 project 相关的 bundle，全选。下一步后运行自动解析依赖，然后完成。

导入成功后，在包浏览器中能看到各个 bundle。

导入到 eclipse 之后在 run 配置中，执行 opendaylight-assembleit 来干净编译整个项目。

几个 target 的含义如下。

opendaylight-application.launch => 运行控制器。

opendaylight-assembleit-fast.launch => 仅编译所选资源 (Project / Bundle)。

opendaylight-assembleit-noclean.launch =>编译所有 bundle，但不执行 clean。

opendaylight-assembleit-skiput.launch =>编译所有 bundle，但不进行 Unit Tests。

opendaylight-assembleit-sonar.launch =>编译所有 bundle 并运行 Sonar (Code-Coverage, Static-Analysis tool)。

opendaylight-assembleit.launch => 干净编译所有 bundle。

opendaylight-sonar-fast.launch =>仅对所选的资源运行 Sonar 任务。

opendaylight-sonar.launch => 执行所有的 Sonar 任务。

1.2 测试运行

1.2.1 登录 web GUI

运行成功后可以打开浏览器访问本地地址，并登陆控制器 web UI，默认用户名和密码都是 admin。

图表 1 控制器 Web 登陆界面

1.2.2 常见运行问题

如果运行时报错：java.lang.OutOfMemoryError: PermGen space。可以通过修改 maven 配置为

```
export MAVEN_OPTS="-Xmx512m -XX:MaxPermSize=256m"
```

1.2.3 使用 Mininet 测试

Mininet 可以用来模拟大规模的虚拟网络。可以从 mininet.github.org 下载代码。

Open Daylight Controller 可以跟 Mininet 很好的联动。

在已经安装 Mininet 环境的机器（如果跟 controller 在同一个机器，可以用 127.0.0.1）中，启动 Mininet

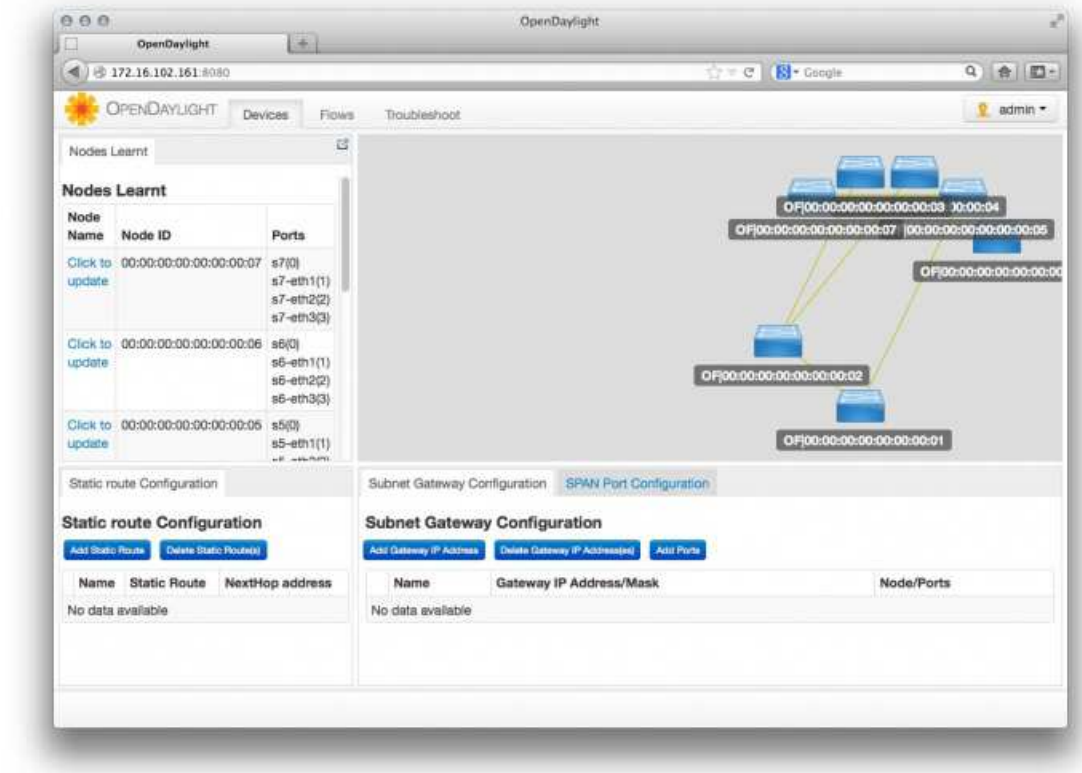
```
mininet@mininet-vm:~$ sudo mn --controller=remote,ip=127.0.0.1 --topo tree
,3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(h1, s3) (h2, s3) (h3, s4) (h4, s4) (h5, s6) (h6, s6) (h7, s7) (h8, s7) (s1, s2) (s1, s5) (s
2, s3) (s2, s4) (s5, s6) (s5, s7)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7
*** Starting CLI:
mininet>
```

连接成功后可以测试控制器的功能。

1.2.3.1 Simple Forwarding Application

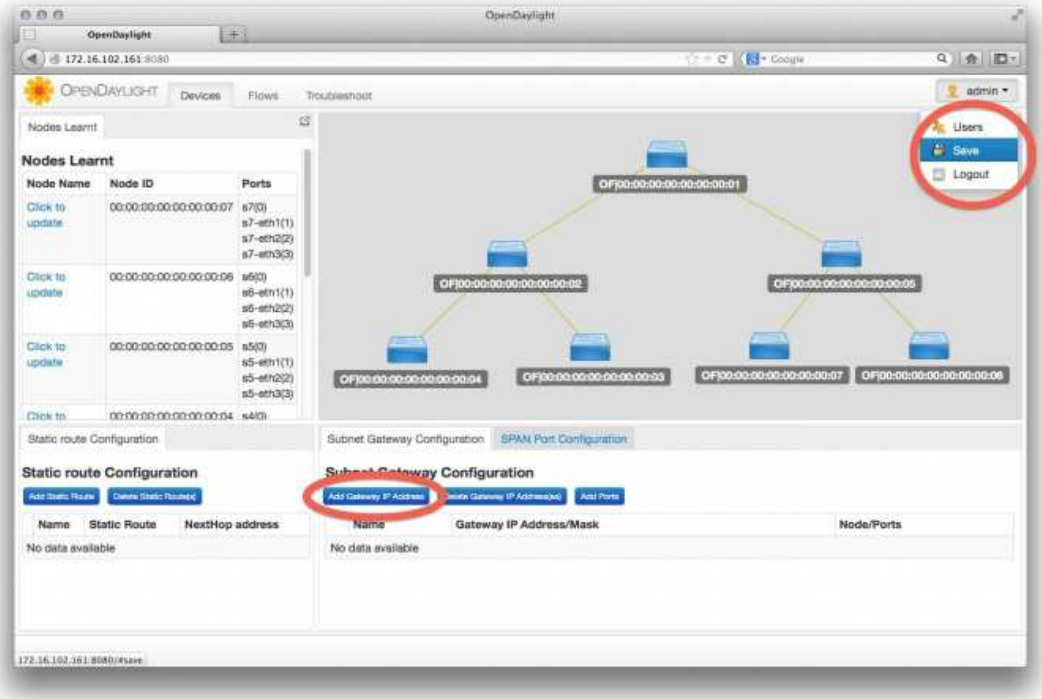
Simple Forwarding 是 Open Daylight Controller 上的一个应用，它通过 arp 包来探测连接到网络的每个主机，并给交换机安装规则（该规则指定到某个主机地址的网包路径），让网包能顺利转到各个主机。

首先，登录到 Controller 界面。



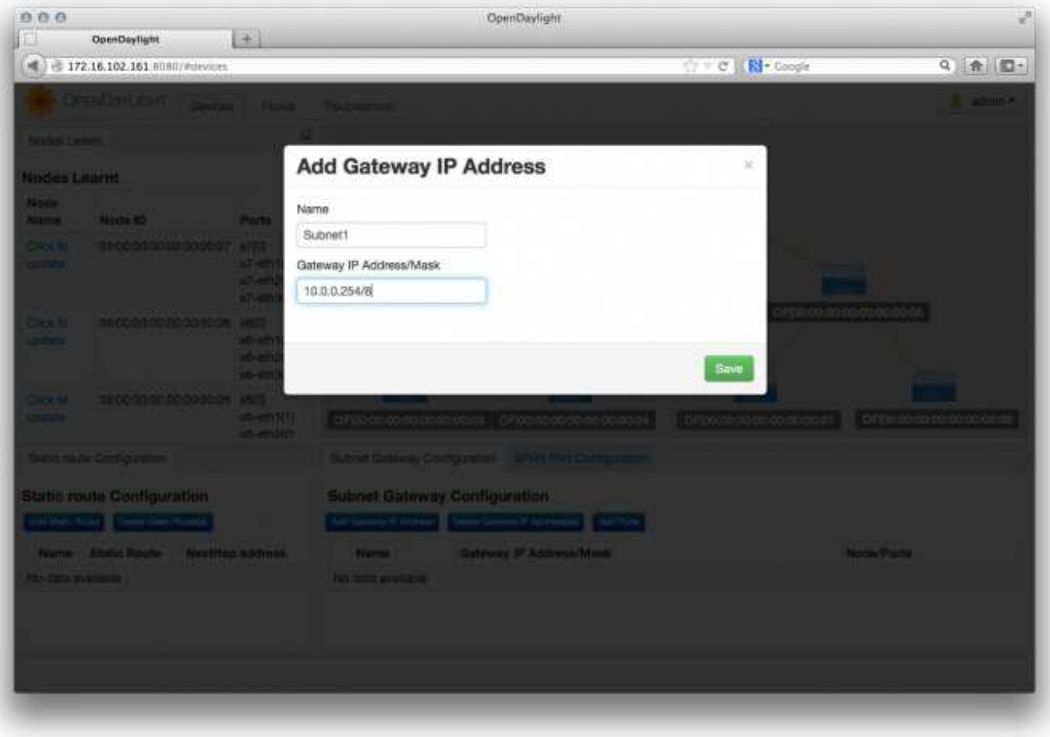
图表 2 Simple Forwarding 界面

通过拖动设备，形成逻辑拓扑，并保存配置。



图表 3 Simple Forwarding 拖动成逻辑拓扑

点击 Add Gateway IP Address 按钮，添加 IP 和 10.0.0.254/8 子网。

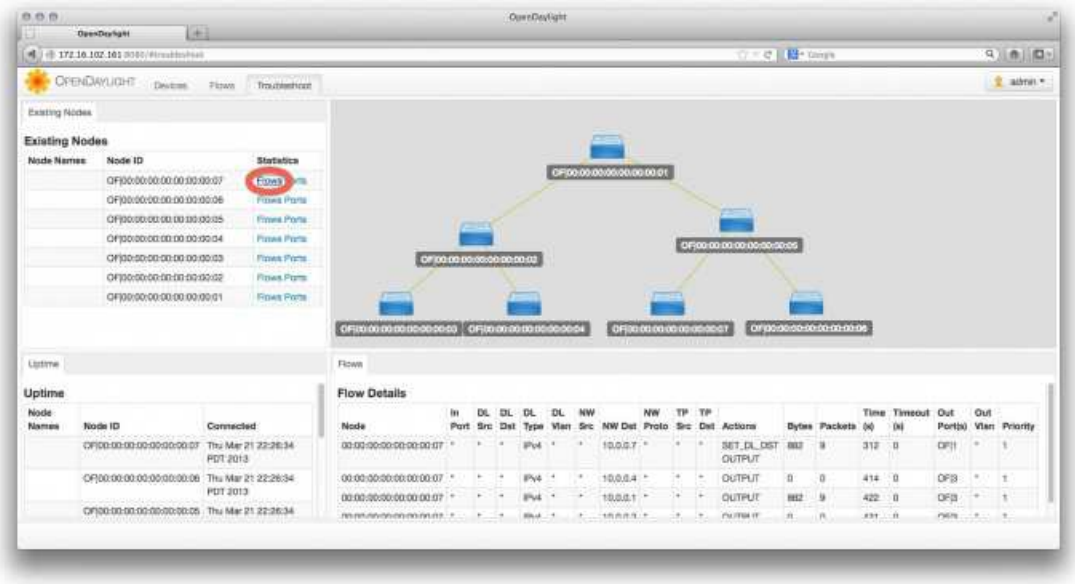


图表 4 添加网关 IP

在 Mininet 中确认主机可以相互连通。

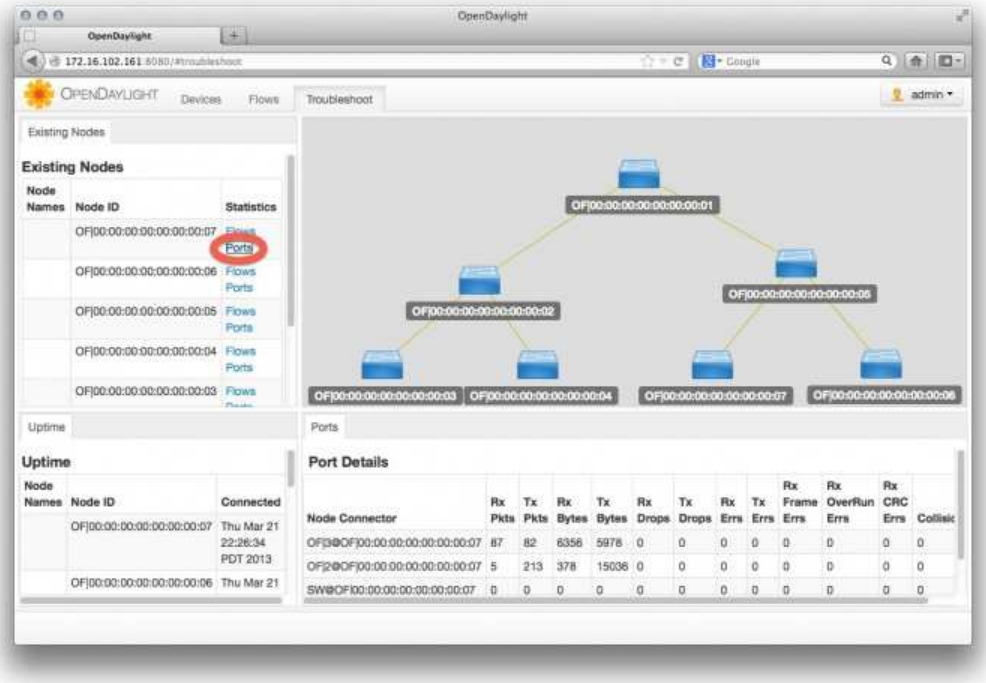
```
mininet> h1 ping h7
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
64 bytes from 10.0.0.7: icmp_req=1 ttl=64 time=1.52 ms
64 bytes from 10.0.0.7: icmp_req=2 ttl=64 time=0.054 ms
64 bytes from 10.0.0.7: icmp_req=3 ttl=64 time=0.060 ms
64 bytes from 10.0.0.7: icmp_req=4 ttl=64 time=0.052 ms
--- 10.0.0.7 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.052/0.422/1.523/0.635 ms
mininet>
```

点击 Troubleshooting 标签页，查看交换机的流表细节。



图表 5 查看交换机流表细节

查看端口细节。



图表 6 查看端口细节

在 osgi 的控制台，输入 ss simple，可以看到 Simple Forwarding 应用被激活。

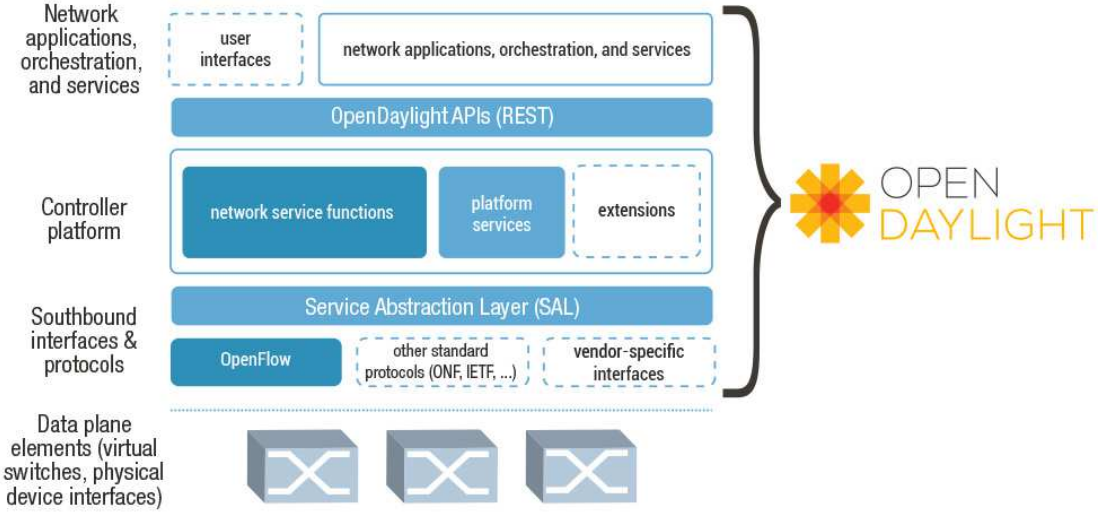
```
osgi> ss simple
"Framework is launched."
id    State       Bundle
```

```
45 ACTIVE org.Open Daylight.controller.samples.simpleforwarding_0.4.0.S
NAPSHOT
```

第2章 基本概念

2.1 整体架构

ODL 架构主要分为三层，从上到下分别是 app、SAL、plugins。



图表 7 整体架构

App 层就是核心的逻辑功能实现，用户自己定义的各种应用。

SAL 是核心抽象层，连接上下层服务，为上层提供一个逻辑的网络 view 和各种对网络的操作接口。

Plugin 是直接跟各种 SDN 设备打交道，支持多种 SDN 协议，类似于驱动层。

2.2 基本工具

要理解 ODL 项目，必须要掌握几个工具，包括 Maven、OSGi 和 Java 接口。

Maven：类似于 ant，是一套管理和组织项目的自动化软件，通过在项目中的 pom.xml 文件来描述项目的组织和各种依赖。ODL 项目的主 pom.xml 就位于 opendaylight\distribution\opendaylight（主目录）下。Eclipse 也是根据 pom.xml 在导入时分析组织关系，添加相关的依赖，生成 OSGi 需要的辅助文件 META-INF/MANIFEST.MF。

OSGi：Java 动态加载模块的事实标准。提供了一个框架，维护的成员以 bundle（类似于模块/包）的形式，在框架上进行服务的注册和获取，并可以相互调用和交互信息。

Java 接口：接口是服务编程的基础。一个接口可以对应到多个实现。Bundle 可以声明接口，告诉 OSGi 自己提供某个服务，其他的 bundle 可以获取这些接口直接使用这些服务。同时，在实现接口时，可以实现某些回调函数，这样发生特定事件的时候回自动被调用运行。

2.3 核心 bundle

ODL 提供的模块和对应的接口可以参考 https://wiki.opendaylight.org/view/Controller_Project's_Modules/Bundles_and_Interfaces。

核心的 bundle 和实现的接口为：

表格 1 核心 Bundle 和实现接口

Bundle	实现/对外服务接口	描述

arphandler	**IHostFinder** IListenDataPacket	通过查看主机的 ARP 会话学习主机位置。
hosttracker	ISwitchManagerAware IInventoryListener **IfIptoHost** IfHostListener ITopologyManagerAware	跟踪主机的位置信息。
switchmanager	IListenInventoryUpdates **ISwitchManager** ICacheUpdateAware IConfigurationContainerAware	维护交换机上的清单信息，包括名称、端口等。
topologymanager	IListenTopoUpdates **ITopologyManager** IConfigurationContainerAware	维护整个网络的拓扑信息。
usermanager	ICacheUpdateAware **IUserManager** IConfigurationAware	用户管理。
statisticsmanager	**IStatisticsManager**	利用 SAL 的 ReadService 采集整个网络中的统计信息。
sal	IReadService	获取网络节点中的 flow/port/queue hardware 信息的接口。
sal	ITopologyService	SAL 提供给应用的拓扑服务接口，提供监听拓扑更新事件的回调方法。

sal	IFlowProgrammerService	在网络节点上安装、卸载、修改流的接口。
sal	IDataPacketService	SAL 提供给应用的 Data Packet 服务。
web	IDaylightWeb	根据系统中启动的 Bundle 来跟踪 Web UI。

第3章 开发应用

3.1 基本功能

SDN 控制器一般从工作过程上来看，至少要实现三个基本功能：

对流事件的响应。包括 PACKET_IN, FLOW_REMOVED 等，对这些事件提供监听的回调函数，例如 receiveDataPacket。

对数据包的处理。例如解析 ARP、TCP、ICMP 等等，并能完成响应的封包。

发出 SDN 消息给节点。例如发送 PACKET_OUT, FLOW_MOD, STATS_REQUEST 等消息。

3.2 基本工作过程

3.2.1 实现 IListenDataPacket 接口

在 ODL 中，SAL 提供了大量的类、（监听者）接口来实现这些操作，例如应用如果希望监听到交换机发送给控制器的数据包，只需要实现 IListenDataPacket 接口即可。

```
public class TutorialL2Forwarding implements IListenDataPacket
```

一旦某个类实现了 IListenDataPacket 接口，则当数据包发送到控制器的时候，会自动调用接口中的回调函数 receiveDataPacket。因此重写这个函数就可以实现获取网包的一份 copy 并进行自定义的处理。

```
public PacketResult receiveDataPacket(RawPacket inPkt) { ... }
```

3.2.2 解析网包

需要注意，收到的网包格式是原始包（物理网络上的包），需要利用 dataPacketService 来进行解析，首先解析为 Packet 通用格式包。

```
Packet formattedPak = this.dataPacketService.decodeDataPacket(inPkt);
```

另外，从原始包的包头中还携带了一些上下文信息，例如到达的交换机节点和接口（在 ODL 中为各种 connector）信息。

```
NodeConnector incoming_connector = inPkt.getIncomingNodeConnector();
```

另外，基于解析后的通用包格式还可以进行一步提取各层信息。例如获取源 mac 地址。

```
byte[] srcMAC = ((Ethernet)formattedPak).getSourceMACAddress();
long srcMAC_val = BitBufferHelper.toNumber(srcMAC);
```

3.2.3 发出网包

发出网包的时候需要构建一个原始包格式。可以利用 dataPacketService 来从一个指定的交换机端口发出，例如将收到的包从端口 p 发出。

```
RawPacket destPkt = new RawPacket(inPkt);
destPkt.setOutgoingNodeConnector(p);
this.dataPacketService.transmitDataPacket(destPkt);
```

3.2.4 发出流消息

除了 PACKET_OUT 之外，还可以创建流消息并添加到指定交换机上。

一条流消息需要指定 Match、Action 和消息类型。

3.2.4.1 Match

创建一条 Match，匹配 IN_PORT 为到达网包的入口。

```
Match match = new Match();
match.setField(MatchType.IN_PORT, incoming_connector);
```

其他匹配域还包括"inPort", "dlSrc", "dlDst", "dlVlan", "dlVlanPriority", "dlOuterVlan", "dlOuterVlanPriority", "dlType", "nwTOS", "nwProto", "nwSrc", "nwDst", "tpSrc", "tpDst"等。所有这些匹配域的定义都在 org.opendaylight.controller.sal.match 包中。

3.2.4.2 Action

Action 的定义在 org.opendaylight.controller.sal.action 包中，包括"drop", "loopback", "flood", "floodAll", "controller", "interface", "software path", "harware path", "output", "enqueue", "setDlSrc", "setDlDst", "setVlan", "setVlanPcp", "setVlanCif", "stripVlan", "pushVlan", "setDlType", "setNwSrc", "setNwDst", "setNwTos", "setTpSrc", "setTpDst", "setNextHop"。

例如创建一个 action 列表，添加 OUTPUT action。

```
List<Action> actions = new ArrayList<Action>();
actions.add(new Output(outgoing_connector));
```

3.2.4.3 发出流

创建好流之后，可以进行添加、删除等操作，例如添加流到入口交换机上。

```
Flow f = new Flow(match, actions);
Status status = programmer.addFlow(incoming_node, f);
if (status.isSuccess())
   logger.trace("Installed flow {} in node {}", f, incoming_node);
else
   //What to do in case of flow insertion failure?
```

3.2.4.4 返回值

处理结束后，应用需要通过返回值告诉控制器进行下一步的操作，包括三种情况。

PacketResult.CONSUME: 网包已经被处理完成了，后续不需要进行任何处理。

PacketResult.KEEP_PROCESSING: 网包已被处理过, 仍需要发送给其它相关 bundle 进行处理。

PacketResult.IGNORED: 网包未被处理, 需要发送给其它相关 bundle 进行处理。

3.2.4.5 查询交换机端口

可以同过使用 Switch Manager 类来列出交换机所有的端口。

```
Set<NodeConnector> nodeConnectors =
this.switchManager.getUpNodeConnectors(incoming_node);
```

3.2.5 依赖和配置管理

通过 pom.xml 来管理 bundle 之间的相互依赖。

每个 bundle 中经常还有一个 Activator.java 文件，继承自 OSGi 框架，并重载了方法来跟 OSGi 框架进行注册。利用 setInterface()方法来告诉 OSGi 框架自己实现了哪些接口；利用 add()方法告诉 OSGi 框架自己使用了哪些其他的 bundle。

3.3 添加新模块

3.3.1 复制已有模块

最简单的创建新模块的办法是复制一个已经存在的模块目录（比如 arphandler），并改名为自己的模块名（比如 tutorial_L2_forwarding），修改新模块目录中所有 java 文件和 pom.xml 文件中的 ARPHandler 名称为 TutorialL2Forwarding，所有的 arphandler 修改为 tutorial_L2_forwarding。

也可以下载文件 http://yuba.stanford.edu/~srini/tutorial/opendaylight_tutorial_srcfiles.tar，解压后放到 opendaylight/opendaylight 目录下，并导入到项目中。

3.3.2 修改配置文件

首先修改整个项目的 pom.xml 文件，位于 opendaylight/opendaylight/distribution/opendaylight/pom.xml，在 arphandler 下面，添加

```
<module>../../tutorial_L2_forwarding</module>
```

修改 logger 的配置文件，位于 opendaylight/opendaylight/distribution/opendaylight/src/main/resources/configuration/logback.xml，添加

```
<logger name="org.opendaylight.controller.tutorial_L2_forwarding" level="INFO"/>
```

如果是通过 eclipse 来运行，还需要在 launch 文件中添加新的模块。在 run configuration 中，选添加 java 项目，选择 org.opendaylight.controller.tutorial_L2_forwarding。

所有的修改在 http://yuba.stanford.edu/~srini/tutorial/opendaylight_changes_for_adding_module.patch 中可以找到。

第4章 示例应用

基于上面的介绍，下面编写一个 hub 和 L2 的普通交换机。

4.1 Hub

一个 hub 将把收到的网包从其他的端口广播出去，因此需要执行如下操作：
监听 packet_in 消息；
列出交换机上所有的端口；
将收到的网包从非入口发出；

4.2 L2 交换机

对于 L2 普通交换机，需要学习 mac，因此需要执行如下操作：
创建一个 HashMap mac_to_port 来记录 mac 到所在端口的映射。
监听 packet_in 消息；
解析网包，获取源 mac 地址和目标 mac 地址；
存储源 mac 和入口到 mac_to_port 表中；
查找目的 mac 是否已经存在，如果存在则生成 flow_mod 消息并发出，否则广播。

4.3 POM.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"  xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.opendaylight.controller</groupId>
    <artifactId>commons.opendaylight</artifactId>
    <version>1.4.0-SNAPSHOT</version>
    <relativePath>../commons/opendaylight</relativePath>
  </parent>

  <artifactId>tutorial_L2_forwarding</artifactId>
  <version>0.4.0-SNAPSHOT</version>
  <packaging>bundle</packaging>

  <build>
```

```
  <plugins>
   <plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
    <version>2.3.6</version>
    <extensions>true</extensions>
    <configuration>
     <instructions>
      <Import-Package>
       org.opendaylight.controller.sal.core,
       org.opendaylight.controller.sal.utils,
       org.opendaylight.controller.sal.packet,
       org.opendaylight.controller.sal.match,
       org.opendaylight.controller.sal.action,
       org.opendaylight.controller.sal.flowprogrammer,
       org.opendaylight.controller.switchmanager,
       org.apache.felix.dm,
       org.osgi.service.component,
       org.slf4j
      </Import-Package>
      <Bundle-Activator>
       org.opendaylight.controller.tutorial_L2_forwarding.internal.Activator
      </Bundle-Activator>
     </instructions>
     <manifestLocation>${project.basedir}/META-INF</manifestLocation>
    </configuration>
   </plugin>
  </plugins>
 </build>
 <dependencies>
  <dependency>
   <groupId>org.opendaylight.controller</groupId>
   <artifactId>switchmanager</artifactId>
   <version>0.4.0-SNAPSHOT</version>
  </dependency>
```

```
    <dependency>
      <groupId>org.opendaylight.controller</groupId>
      <artifactId>sal</artifactId>
      <version>0.5.0-SNAPSHOT</version>
    </dependency>
  </dependencies>
</project>
```

4.4 MANIFEST.MF

```
Manifest-Version: 1.0
Bnd-LastModified: 1371179321226
Build-Jdk: 1.7.0_21
Built-By: ubuntu
Bundle-Activator: org.opendaylight.controller.tutorial_L2_forwarding.int
 ernal.Activator
Bundle-ManifestVersion: 2
Bundle-Name: tutorial_L2_forwarding
Bundle-SymbolicName: org.opendaylight.controller.tutorial_L2_forwarding
Bundle-Version: 0.4.0.SNAPSHOT
Created-By: Apache Maven Bundle Plugin
Import-Package: org.apache.felix.dm;version="[3.0,4)",org.opendaylight.c
 ontroller.sal.action,org.opendaylight.controller.sal.core,org.opendayli
 ght.controller.sal.flowprogrammer,org.opendaylight.controller.sal.match
 ,org.opendaylight.controller.sal.packet,org.opendaylight.controller.sal
 .utils,org.opendaylight.controller.switchmanager,org.osgi.service.compo
 nent;version="[1.2,2)",org.slf4j;version="[1.7,2)"
Tool: Bnd-1.50.0
```

4.5 org.opendaylight.controller.tutorial_L2_forwarding.internal 包

4.5.1 Activator

通过 configureInstance()函数中调用 setInterface 来告诉 OSGi 框架，自己挂载到了 IListenDataPacket 服务接口上。

4.5.2 TutorialL2Forwarding

定义了 TutorialL2Forwarding 类，实现了 IListenDataPacket 接口，重载了 receiveDataPacket

()函数。

```
@Override
    public PacketResult receiveDataPacket(RawPacket inPkt) {
        if (inPkt == null) {
            return PacketResult.IGNORED;
        }
        logger.trace("Received a frame of size: {}",
                     inPkt.getPacketData().length);

        Packet formattedPak = this.dataPacketService.decodeDataPacket(inPkt);
        NodeConnector incoming_connector = inPkt.getIncomingNodeConnector(
);
        Node incoming_node = incoming_connector.getNode();

        if (formattedPak instanceof Ethernet) {
            byte[] srcMAC = ((Ethernet)formattedPak).getSourceMACAddress();
            byte[] dstMAC = ((Ethernet)formattedPak).getDestinationMACAddress();

            // Hub implementation
            if (function.equals("hub")) {
                floodPacket(inPkt);
                return PacketResult.CONSUME;
            }

            // Switch
            else {
                long srcMAC_val = BitBufferHelper.toNumber(srcMAC);
                long dstMAC_val = BitBufferHelper.toNumber(dstMAC);

                this.mac_to_port.put(srcMAC_val, incoming_connector);

                Match match = new Match();
                match.setField( new MatchField(MatchType.IN_PORT, incoming_conn
ector) );
                match.setField( new MatchField(MatchType.DL_DST, dstMAC.clone()) )
;
```

```
            NodeConnector dst_connector;

            // Do I know the destination MAC?
            if ((dst_connector = this.mac_to_port.get(dstMAC_val)) != null) {
                List<Action> actions = new ArrayList<Action>();
                actions.add(new Output(dst_connector));

                Flow f = new Flow(match, actions);

                // Modify the flow on the network node
                Status status = programmer.addFlow(incoming_node, f);
                if (!status.isSuccess()) {
                    logger.warn(
                            "SDN Plugin failed to program the flow: {}. The failure is: {}",
                            f, status.getDescription());
                    return PacketResult.IGNORED;
                }
                logger.info("Installed flow {} in node {}",
                        f, incoming_node);
            }
            else
                floodPacket(inPkt);
        }
    }
    return PacketResult.IGNORED;
}
```

4.5.3 TutorialL2Forwarding_multiswitch

跟 TutorialL2Forwarding 类似，定义了 TutorialL2Forwarding 类，实现了 IListenDataPacket 接口，重载了 receiveDataPacket()函数。所不同的是，在进行 mac 学习的时候区分交换机，所以可以同时管理多个 L2 交换机。