



OpenDaylight

用户手册

OpenDaylight中文社区翻译v0.1版

Lithium(2015.8.20)



OPEN
DAYLIGHT

前言

从 13 年接触 OpenDaylight Hydrogen 版本开始，恐怕对 OpenDaylight 最大的诟病莫过于——“文档不齐全的开源都是耍流氓”。学习资料的匮乏的确在我们和 OpenDaylight 之间竖起了一道壁垒。对于刚接触网络或编程的小伙伴来说，让本来就晦涩难懂的知识显得更加遥不可及。从 Hydrogen 版本的基本无文档，到 helium 版本文档迟迟不露面，新发布的 lithium 版本终于用户手册和软件同步了。但随之而来的是各种专业英语的旁征博引，洋洋洒洒将近 300 页，如此庞大的数据量势必会让人望而生畏。

一场来自民间的自发翻译活动这时低调的展开了，针对与 OpenDaylight Lithium 版本同时发布的《OpenDaylight User Guide Lithium (June 29, 2015)》文档展开了翻译。陆续有 OpenDaylight 中文社区的热心用户参与进来，成立了——OpenDaylight 官方文档翻译小组，从项目招募开始到完成初稿，翻译文档 267 页，累计用时 15 天。并在 SDNLAB.com 网站上经行了为期一周的公测。

参与成员

yoofooyoo：我不是一个人在战斗。不是一个人在战斗的灵魂人物，沉稳霸气。
参与翻译风控工作，智慧与美貌并存灵魂级鼓励师和人生导师。

君子一诺：SDN 测试外加小小编一枚。OpenDaylight 深度用户，霸气女神范，技术达人。

参与翻译 2、5、10、11、15、16、23、27 章节翻译和后期审核工作。

虾米：面朝大海，春暖花开。像名字一样好吃的萌萌哒可爱女神，华丽的翻译文风，棒棒哒。

参与 9、10、17、18、19、22、25 章节翻译和后期审核工作。

dandan：一直不知道这只跳舞小象的真实身份（据说是呵呵哒女神）。果断、干练的翻译文风，好腻害。

参与 3、13、14、24、28 章节翻译和后期审核工作。

云南白药口服：不知道真人是不是像头像一样白（希望是的）。绝对的效率帝，无与伦比的魔幻 $ctrl+C$ 和 $ctrl+V$ 。

参与后期翻译审核和上传工作。

胖欧巴：在 SDN 的路上渐行渐远。本次翻译活动策划组织者，不知道是头像胖还是真人胖的胖欧巴，OpenDaylight 活跃用户，希望技术学习能像听相声一样有趣。

参与 1、4、7、10、12、20、26、28 章节翻译、后期审核工作，附赠鼓励师服务。

大感谢

感谢 OpenDaylight 中文社区热心用户参与我们的活动，并且让整个活动有条不紊的进行着。欢迎更多的对 SDN 和 ODL 感兴趣的小伙伴参与到我们的活动中来，加入 OpenDaylight 官方文档翻译小组，也欢迎在我们的技术社区进行 <http://www.sdnlab.com/odlcommunity/>，让我们一起为小伙伴们更好的在网络创新领域成长做一些事情。对 SDN 和 OpenDaylight 感兴趣的小伙伴也可以经常浏览 www.sdnlab.com 获取相关资讯，也可以加入我们的 OpenDaylight 讨论群（QQ：194240432）进行技术探讨。

活动预告

后续我们会开展《OpenDaylight Developer Guide Lithium (June 29, 2015)》和《OpenDaylight and OpenStack Lithium (June 29, 2015)》翻译活动，欢迎大家参与。

详情求搭讪

SDNLAB : QQ : 506985144

或胖欧巴 : QQ : 353176266

目录

第一部分 开始使用 OpenDaylight

1.OpenDaylight Controller 概述 -----	1
2.OpenDaylight 用户接口--DLUX -----	1
3.运行 XSQL 控制台命令和查询语句 -----	7
4.在 OpenDaylight 上创建集群 -----	11

第二部分 应用程序和插件

5. ALTO 用户向导 -----	16
6.身份认证和授权服务 -----	19
7.BGP 用户指南 -----	26
8. CAPWAP User Guide -----	35
9.DIDM 用户指南 -----	37
10. Group Based Policy 用户指南 -----	37
11.L2Switch 用户指南 -----	76
12.L3VPN 服务：用户指南 -----	82
13.链路聚合控制协议使用指南 -----	88

14.LISP 流映射使用指南 -----	91
15.网络意图组成 (NIC) 用户指南 -----	102
16.ODL-SDNi 用户指南 -----	104
17.OpFlex ovs 代理用户指南 -----	104
18.PCEP 用户指南 -----	110
19.PCMM 用户指南 -----	111
20.Service Function Chaining 用户指南 -----	114
21.SNMP 插件用户指南 -----	142
22.SXP 用户指南 -----	146
23.TCPMD5 用户指南 -----	149
24.TSDR H2 数据存储使用指南 -----	155
25.TSDR HBase 数据存储用户指南 -----	156
26.TTP CLI Tools 用户指南 -----	158
27.统一安全通道 -----	158
28.虚拟租户网络 (VTN) -----	160

OpenDaylight Lithium用户指南v0.1

[TOC]

1 OpenDaylight Controller 概述

OpenDaylight Controller（后面将用控制器代指）基于JVM软件，能够运行在任意支持Java的操作系统或硬件上。控制器是软件定义网络（SDN）的概念的一个实现，并且使用了如下工具：

Maven: OpenDaylight使用Maven进行更为简单的自动化配置。Maven使用pom.xml（Project Object Model）文件管理bundle间的脚本依赖关系，并且描述bundles的加载和启动。

OSGi: 这个属于OpenDaylight后端技术框架，它允许动态加载bundles和JAR包文件，并且绑定bundles以提供bundle间的信息交互。

JAVA接口: Java接口用于事件监听、规范和形成模式。这是特殊bundles实现事件回调函数和表示特殊意识状态的一个主要方式。

REST APIs: 这里有类似于拓扑管理、主机追踪、流编程、静态路由等北向接口。

控制器向应用提供开放的北向APIs。OSGi框架和双向的REST对北向APIs进行支持。OSGi框架用在与控制器在同一地址空间的应用，REST API用在与控制器不在同一地址空间（或在同一系统里）的应用。业务逻辑和算法都驻留在应用程序中。这些应用程序利用控制器收集网络的情报，运行自身算法进行分析，然后编排用于网络的新规则。在南向接口上，多种协议以plugin的形式提供，例如OpenFlow 1.0、OpenFlow 1.3、BGP-LS等等。

OpenDaylight控制器启动默认支持OpenFlow 1.0协议。其他OpenDaylight代码贡献者也开始向控制器添加代码。这些模块动态链接到服务抽象层（SAL）。

SAL向北向的模块提供服务。SAL解决了提供控制器和网络设备间协议无关服务的问题。这种思路让随时间变化的类似于OpenFlow或其他协议对应用程序不会带来影响。控制器需要控制其管辖范围的设备，所以它要对设备的能力、可达性等信息进行了解。这些信息通过拓扑管理模块进行存储和管理。其他部件，诸如ARP管理、主机追踪、设备管理、交换机管理将帮助拓扑管理部件生成拓扑数据库。

更多的OpenDaylight controller概要的详细信息，请参见OpenDaylight开发者指南。

2 OpenDaylight用户接口--DLUX

获取DLUX

DLUX提供很多不同的Karaf features组件，能启动和关闭相分离。在Lithium版本中，与DLUX相关的features主要有：odl-dlux-core、odl-dlux-node、odl-dlux-yangui、odl-dlux-Yangvisualizer。

登录

安装以上应用后，登录DLUX：

1. 打开浏览器并在浏览器中输入登录URL：<http://:8181/index.html>（注：推荐使用chrome浏览器）。

2. 使用admin作为用户ID和密码认证登录应用。

注意：在Lithium版本中，admin是DLUX唯一可用用户类型。

使用DLUX工作

登录DLUX后，如果只启用odl-dlux-core，将在左边只看到可用拓扑应用。（注：为了在所有面板详情中确定拓扑显示，在karaf中启用odl-l2switch-switch）

DLUX还有其他apps，如nodes、yang UI，只有在Karaf分布中，分别启用odl-dlux-node和odl-dlux-yangui功能，其他apps才显示出来。

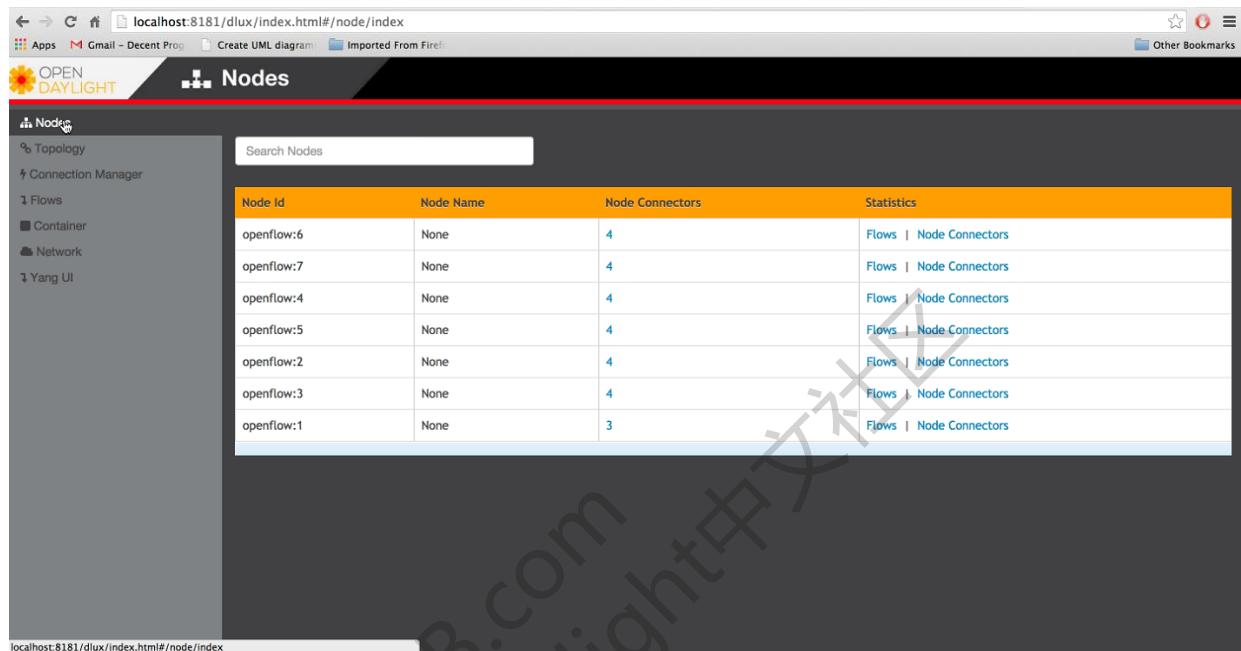


图2.1 DLUX模块

注：如果你在dlux中安装了应用，在刷新浏览器页后，它们将显示在左边导航栏。

网络统计视图

左边面板的Nodes模块将可以查看网络中交换机的网络统计和端口信息。

为使用Nodes模块：

1. 选择左边“Nodes”。右边将显示所有节点、节点连接器和统计的列表信息。
2. 在“Search Nodes”中输入一个节点的ID，将通过节点连接器搜索。
3. 点击“Node Connectors”号查看详细信息，如端口ID、端口名称、每个交换机的端口号、MAC地址等等。
4. 点击统计行中的“Flows”查看具体节点的流表统计信息，如表ID、匹配包、活跃状态的流等等。
5. 点击“Node Connectors”查看具体节点ID的节点连接器的统计信息。

网络拓扑视图

拓扑标签显示被创建的网络拓扑的图形化表示。（注：DLUX UI不提供添加拓扑信息的功能。拓扑由OpenFlow插件创建。当使用OpenFlow协议连接控制器时，控制器在数据库中存储信息且显示在DLUX界面上。）

为查看网络拓扑：

- 1.选择左边面板的“Topology”，可以在右边查看图形化显示。在图中，蓝色方框表示交换机，黑色代表可用主机，线表示交换机之间怎样连接。
- 2.在主机、链路或者交换机上悬停鼠标，显示源和目的端口。
- 3.使用鼠标滚轮放大和缩小图标验证拓扑。

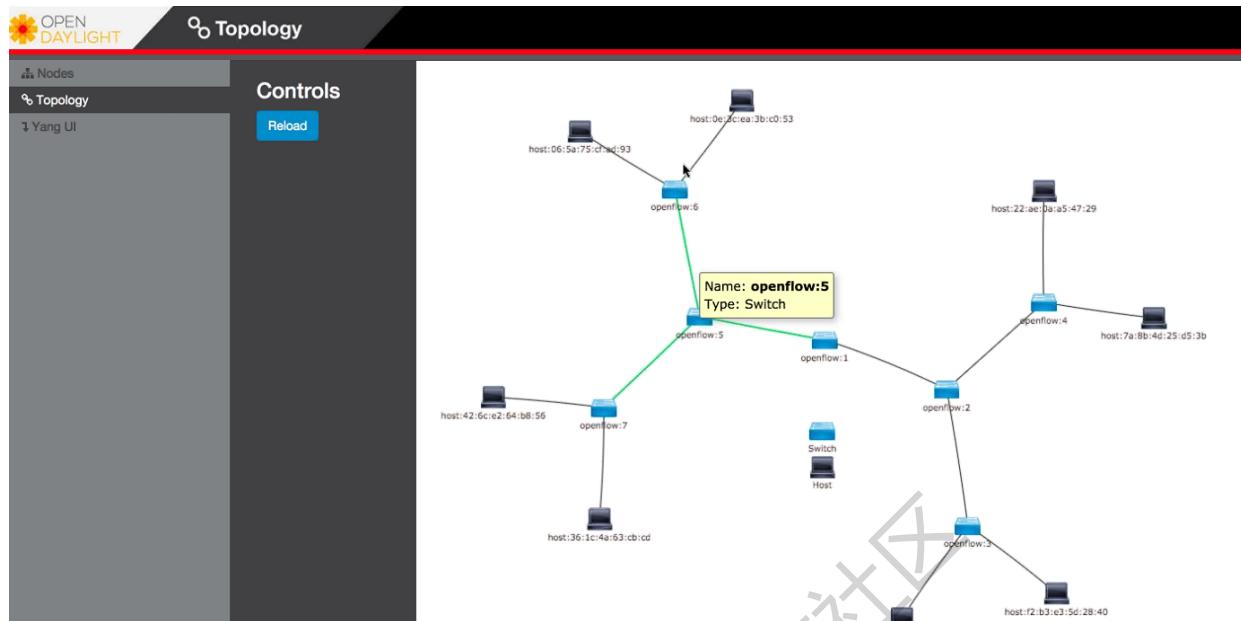


图2.2 拓扑模块

与OpenDaylight交互

Yang UI模块用于与ODL交互。关于Yang Tools的更多详细信息，可查看：
https://wiki.opendaylight.org/view/YANG_Tools>Main [YANG_Tools]。

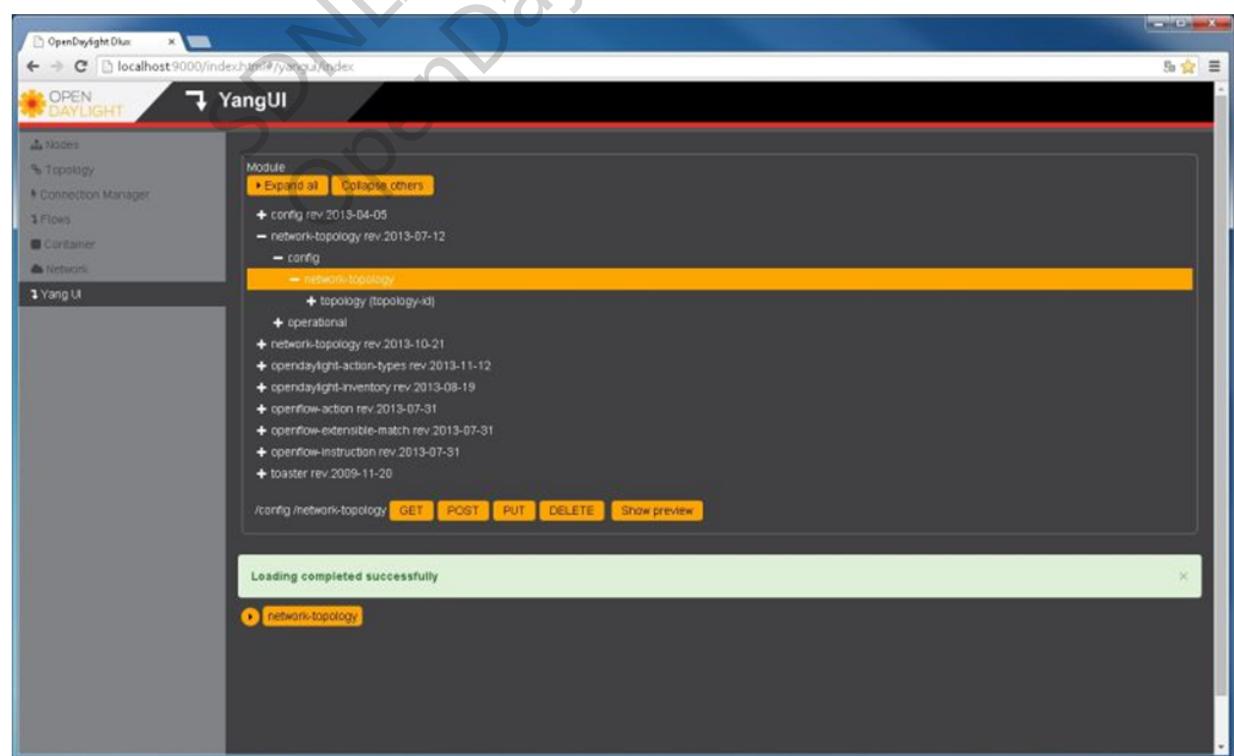


图2.3 Yang UI

为使用Yang UI：

1.选择左边面板的“Yang UI”。右边显示分为两部分。

2.顶部显示APIs、subAPIs和buttons的树形结构，为了调用可能功能，主要包括：GET、POST、PUT、DELETE等等。不是每个subAPIs都能调用每个功能的，如：subAPIs“operational”可选的只有GET功能。当ODL中的现有数据显示或者能被用户在页面上填写并发送给ODL时，可以从ODL填写输入数据。API树下的按钮是可变的，它依赖于subAPIs规范。常见按钮有：

GET：从ODL获取数据；

PUT和POST：为保存配置，发送数据给ODL；

DELETE：为删除配置，发送数据给ODL。为这些操作，必须执行xpath。路径显示在与buttons同一行的前面，为特定路径元素标识符包括文本输入。

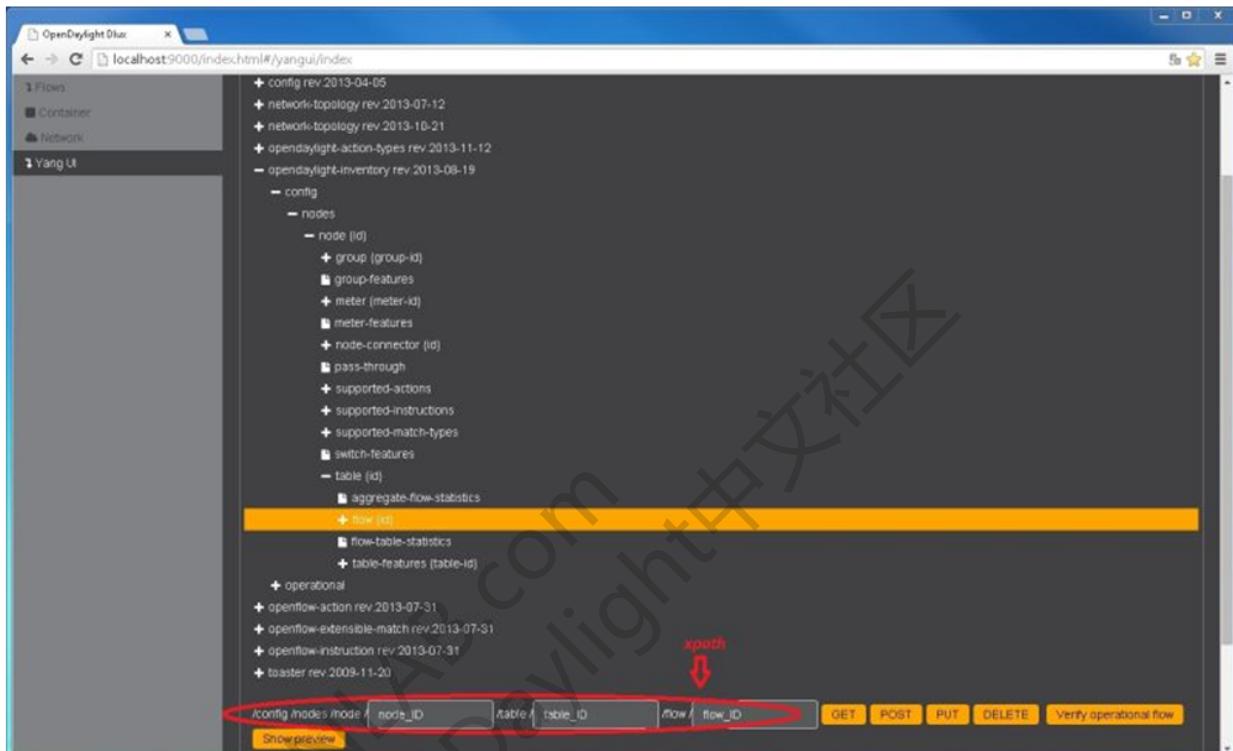


图2.4 Yang API规范

3.右面板底部根据选择的subAPI显示输入。每个subAPI代表列表声明的列表元素，一个列表中可能有多个列表元素，例如：一个设备能够存储多条流。在这个例子中，“flow”是列表的名称，每个列表元素是不同的key值。列表的列表元素中可能包含其他列表，每个列表元素有一个列表名称、key名称及key值、删除列表元素的按钮。通常列表声明的key包含一个ID。从ODL中填写输入并从xpath部分使用GET按钮，或者通过界面上用户填写输入并发送给ODL。

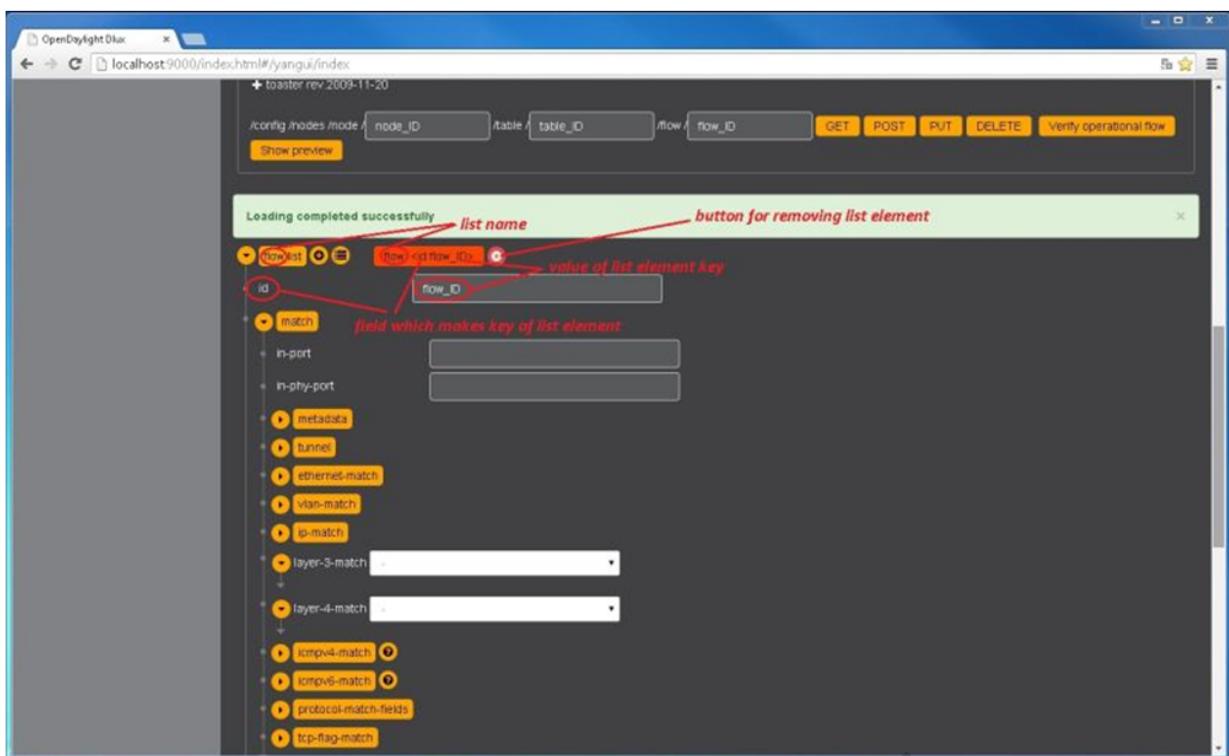


图2.5 Yang UI API规范

4.点击API树下的“Show Preview”按钮显示发送给ODL的请求。当输入被填写时，右边面板显示请求文本。

显示Yang UI的拓扑

为显示拓扑：

- 1.选择subAPI的network-topology → operational → network-Topology。
- 2.通过点击“GET”按钮从ODL获取数据。
- 3.点击“Display Topology”。

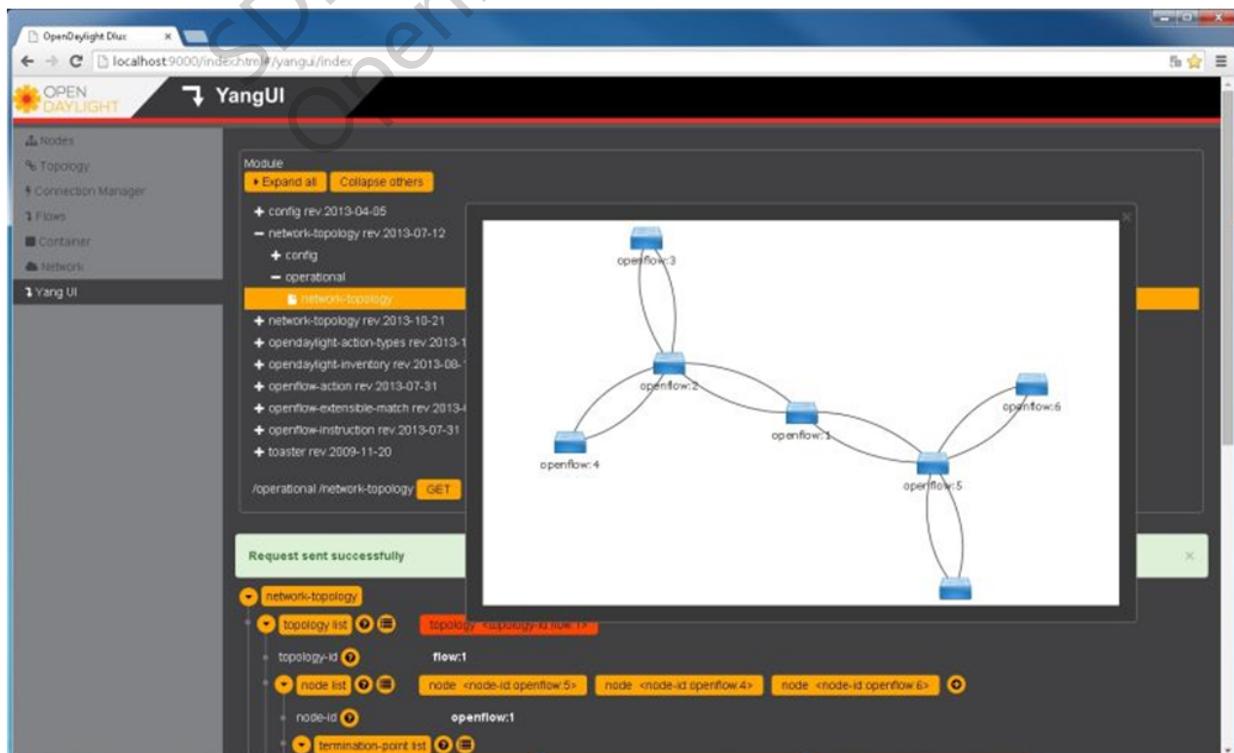


图2.6 DLUX Yang拓扑

配置Yang UI的列表元素

列表显示为树形结构，点击列表名称之前的箭头可以展开或折叠。为了在Yang UI上配置列表元素：

1. 使用列表名称之后提供的加号图标按钮“+”添加一个新的空输入列表元素。当一些列表元素被添加时，名称和key值按钮随之被显示。
2. 为了删除一些列表元素，使用每个列表元素后面提供的“X”按钮： DLUX List Elements. image::dlux-yang-list-elements.png[DLUX list elements,width=500]。
3. 列表值是一个或者多个输入，用来像列表元素的识别。一个列表的所有列表元素必须是不同的key值。如果一些元素存在相同的key值，警告图标“！”在他们的名称按钮附近显示。

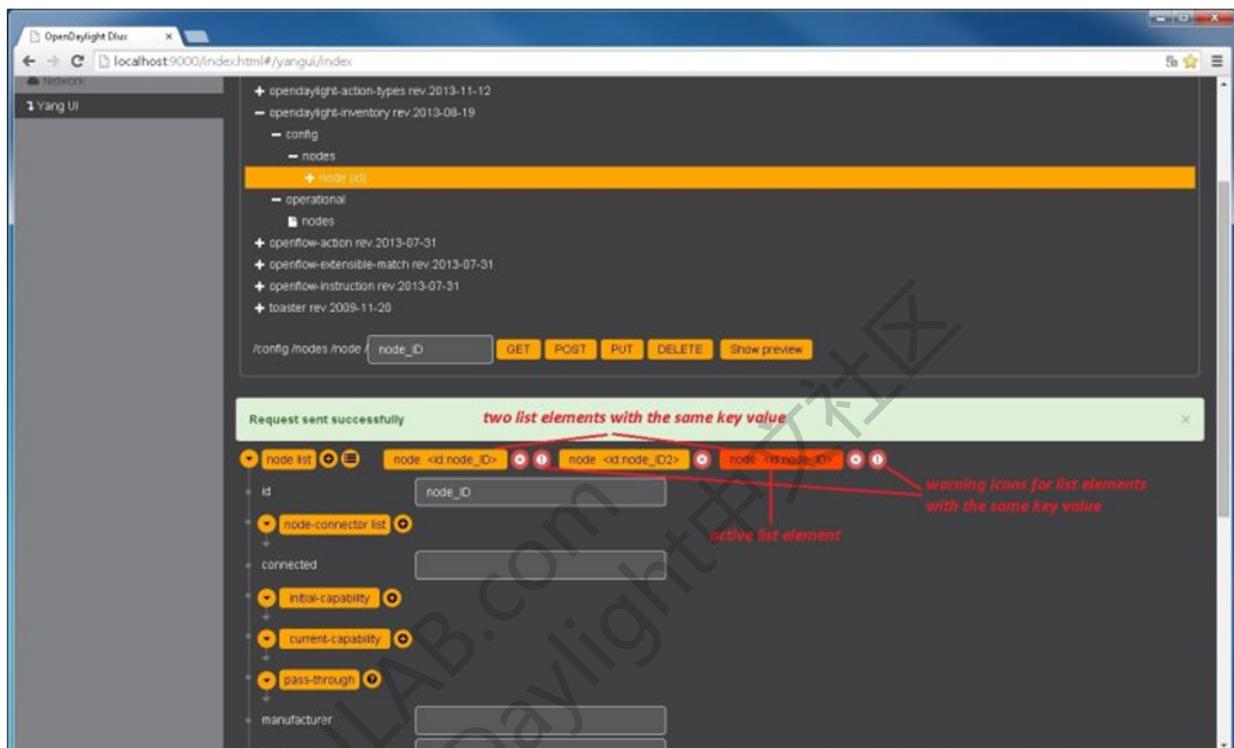


图2.7 DLUX列表警告

4. 当列表包含至少一个列表元素时，“+”图标之后是选择被显示的列表元素图标。通过点击图标，能够选择他们中的一个。相邻列表元素的名称按钮被显示一排列表中。你能通过点击箭头按钮向前或者向后显示一排的列表元素名称按钮。

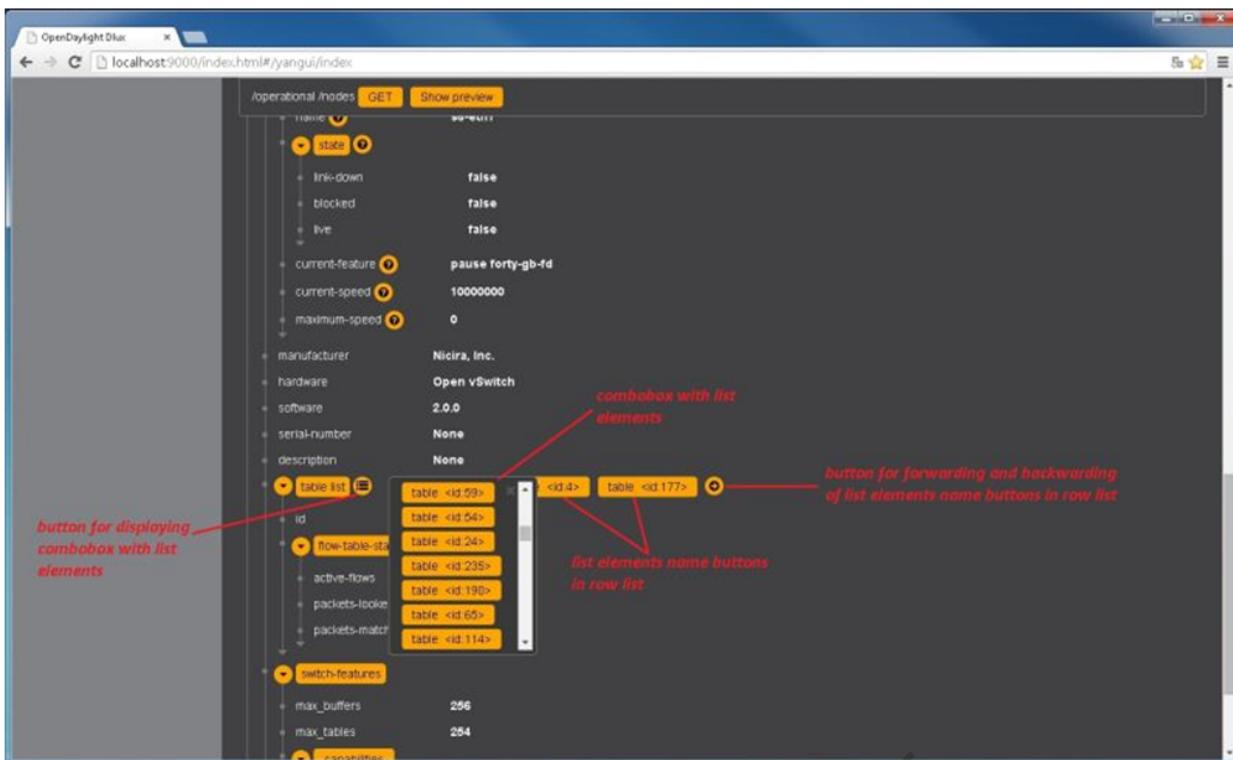


图2.8 DLUX列表按钮

3 运行XSQL控制台命令和查询语句

XSQL概述

XSQL是基于XML的查询语言，描述了简单的存储过程，包括解析XML数据、查询或更新数据库表以及对XML的输出进行排版。XSQL允许你查询树模型，把这些模型当作一个序列数据库。例如，你可以运行一个查询命令，列出所有配置在特定模块上的端口以及它们的属性。

下面章节将介绍XSQL的安装过程、支持的XSQL命令和结构查询的正确方法。

安装XSQL

在XSQL控制台运行命令之前，首先必须在你的系统中安装XSQL：

- 1.进入解压缩OpenDaylight源文件的目录。
- 2.启动Karaf: `./karaf`
- 3.安装XSQL: `feature:install odl-mdsal-xsql`

XSQL控制台命令

在XSQL控制台输入命令的结构如下： `odl:xsql <XSQl command>`

下表描述了本次发布的OpenDaylight版本支持的命令。

表3.1 支持的XSQL控制台命令

SDNLAB.com
OpenDaylight中文社区

命令	描述
r	重复你执行过的最后一个命令。
list vtables	列出目前安装的schema节点容器。无论OpenDaylight module是否安装，它的YANG模块都放在了Schema Context中。在使用这个命令的时候，XSQL接收到一个通知，确认module的YANG模块驻留在Schema Context中，然后通过设置必要的vtables和vfields将模块映射到XSQL中。这条命令用于确认vtable的信息。
list vfields <vtable name>	列出目前在指定的vtable中的vfield。这条命令用于确认vfield的信息。
jdbc <ip address>	当ODL服务器在防火墙之后，并且JDBC客户端不能连接到JDBC服务器，运行这条命令启动客户端建立连接。
exit	关闭控制台。

tocsv	启用/禁用输出.csv文件进行查询转发
filename <filename>	输出的查询数据放到指定的.tocsv文件中。如果你在tocsv建立以后不指定value，文件的filename会自动生成。

XSQL查询语句

使用 `list vtables` 和 `list vfields <vtable name>` 命令提供的信息，你可以运行一个查询语句来提取你所需要的信息。查询语句的结构如下：

```
select <vfields you want to search for, separated by a comma and a space> from <vtables  
you want to search in, separated by a comma and a space> where <criteria> ,<criteria  
operator> ;
```

例如，假设你想在nodes/node-connector表中搜索nodes/node.ID字段，找到所有包含BA的Hardware-Address对象的实例，可以输入如下命令：

```
Select nodes/node. ID from nodes/ node-connector where Hardware-Address like '%BA%' ;
```

以下是支持的条件运算符：

表3.2 支持的XSQL查询条件运算符

SDNLAB.com
OpenDaylight中文社区

条件运算符	描述
=	列出与你指定的value相等的结果。
!=	列出与你指定的value不相等的结果。
like	列出包含你指定的子字符串的结果。例如，如果你指定like %BC%，所有包含BC的字符串都会列出来。
<	列出小于你指定的value的结果。
>	列出大于你指定的value的结果。
and	列出与你指定的两个value匹配的结果。
or	列出与你指定的两个value中至少有一个匹配的结果。
>=	列出大于或等于你指定的value的结果。
#	列出小于或等于你指定的value的结果。
is null	列出还没分配value的结果。
not null	列出所有value都已经匹配的结果。
skip	使用该操作符能够在父节点不匹配指定条件的时候，列出子节点的匹配结果。更多信息请查看下面的例子。

例子：skip条件运算符

假设你看到了如下的结构，并且想确定属于YY类型模块的所有端口：

- Network Element 1
 - Module 1, Type XX**
 - Module 1.1, Type YY
 - Port 1
 - Port 2
 - Module 2, Type YY
 - Port 1
 - Port 2

如果在你的查询条件下指定Module.Type='YY'，与Module 1.1相关的端口将不会返回，因为它的父模块类型是XX。然而，你可以输入Module.Type='YY'或者skip Module!=YY'。这个字段告诉XSQL忽略所有不满足类型是YY的父模块，并且筛选出所有匹配的子模块。在这个例子中，你的查询语句跳过了Module 1，从Module 1.1中挑选相关的数据。

4 在OpenDaylight上创建集群

集群概述

集群是一个实现多个进程和程序在同一个实体中工作的机制。例如，当你用Google搜索时，可能看起来你的搜索请求像是只被一个web服务端所接受。但实际上，你的搜索请求被成千上万的web服务端所接受，只不过他们在在一个集群中，也就是google.com。这样我们可以用同样的思维去类比，你可以在一个实体中运行多个OpenDaylight控制器实例。下面是集群的一些用法：

- 规模化：在集群模式下，如果多个控制器运行在集群模式下，那么可以在控制器上做更多工作或者存储更多数据。你也可以将你的数据拆分成更小的块（这个相当于分片）。-并且在集群中分发数据，或者对集群中某些特定成员进行操作。
- 高可用性：如果在多控制器组成的集群中有一个控制器出现事故，其他的控制器仍然可以继续工作。
- 数据持久性：重启或事故不会造成数据丢失。

下面内容描述了在一个或多个OpenDaylight控制器上创建集群。

单节点集群

在单个OpenDaylight控制器上配置集群，请做如下操作：

1. 下载并且解压一个base controller distribution。你必须使用新的openFlow plugin，所以需要下载一个distribution包，或支持启用新版OpenFlow plugin。
2. 进入本地的 < Karaf-distribution-location >/bin 目录。
3. 运行Karaf: **./karaf**

4. 下载集群的功能组件: **feature:install odl-mdsal-clustering**
5. 如果你使用集成的Karaf分布, 你需要下载openflow plugin flow服务: **feature:install odl-openflowplugin-flow-services**
6. 下载Jolokia bundle: **install -s mvn:org.jolokia/jolokia-osgi/1.1.5**

在一个实例中使用DistributedDataStore feature, 你可以获取如下特性:

- 数据分片: 内存中的MD-SAL树被分拆成若干个更小的子树 (**inventory**、**topology**和**默认值**)。
- 数据持久性: 所有以数据分片定义的数据都可以存储在磁盘上。当重新启动控制器时, 能够利用持续数据将数据分片恢复到之前的状态。。

多节点集群

下面的内容描述了如何在OpenDaylight中设置多节点集群。

部署注意事项

当你进行集群设置时, 需要注意以下信息:

- 当设置一个多节点的集群时, 我们推荐最少使用三台机器。你可以设置一个只有两个节点的进去。但是, 如果两个中有一个宕机的话, 那么剩下的控制器将不再受操控。
- 集群中的每台设备都必须有身份ID。在OpenDaylight中使用节点的角色。你可以在**akka.conf**文件中对第一个节点角色定义为**member-1**, OpenDaylight会使用**member-1**定义那个节点。
- 数据分片用于存储所有数据或者模块的某一段数据。例如, 一个分片能够包含一个模块的所有库存信息, 另一个分片可以包含一个模块的所有拓扑数据。如果你在**modules.conf**文件中未指定一个模块或者在**module-shards.conf**文件中未指定一个分片, 所有的数据将会默认存储在默认的分片中(当然这个分片也必须在**module-shards.conf**文件中被定义)。每个分片会有备份的配置, 当然这个配置也会在**module-shards.conf**中指定。
- 在一个支持HA的三节点组成的集群中。每个节点都会运行默认数据分片的副本。这是因为OpenDaylight集群部署时需要大多数被定义分片副本运行在节点中才能实现相应功能。如果只在两个节点中定义数据分片副本并且其中一个宕机, 那么相应的数据分片功能将不会实现。

给每个集群成员设置**seed**节点时需要注意什么? 当涉及到多个**seed**节点时, 为什么只设置一个IP地址? 能够设置多个**seed**节点进行功能测试吗?

我们建议你对多个**seed**节点进行配置。当一个集群成员启动时, 将会给其所有**seed**节点发送一条消息。随后, 这个集群成员发送一条加入命令给第一个应答的**seed**节点。如果没有**seed**节点回复, 集群成员将重复这个流程直至成功建立一次连接或者直到关机。

当一个节点不可连接后会发生什么情况? 其他两个节点还能正常工作吗? 当第一个节点恢复连接后, 会自动同其他节点进行同步吗?

当一个节点不可连接后, 将在一个配置周期(默认为10秒)内保持下线状态。一旦某个节点宕机, 你需要对其进行重新启动以保证其可以重新加入集群。当重新启动的节点加入集群中时, 该节点将与**lead**节点进行自动同步。

可以运行两个节点进行功能测试吗?

用于功能测试是可以的, 但是进行HA测试则需要运行3个节点。

设置一个多节点集群

在多节点集群中运行OpenDaylight控制器，按照如下步骤进行：

1. 确定三台用于进行集群配置的机器，复制**controller distribution**到每个机器中。
2. 解压**controller distribution**。
3. 定位目录< Karaf-distribution-location >/bin。
4. 运行Karaf: **./karaf**
5. 下载集群feature: **feature:install odl-mdsal-clustering**

注解

关于集群的运行，你必须在每个节点中下载**feature:install odl-mdsal-clustering**。

1. 如果你使用集成的Karaf架构分配，你需要下载**open flow plugin flow services: feature:install odl-openflowplugin-flow-services**
2. 下载Jolokia bundle: **install -s mvn:org.jolokia/jolokia-osgi/1.1.5**
3. 在每个节点上打开下面的.conf文件：
 - configuration/initial/akka.conf
 - configuration/initial/module-shards.conf
4. 在每个配置文件中做下面一些修改：
 - a. 将每个实例中下面代码块中的[127.0.0.1](#)用运行控制器的主机名或IP地址替换：

```
netty.tcp {  
    hostname = "127.0.0.1"
```

注解

集群中的每个节点的指定的值需要是不同的

- a. 将每个实例中下面代码块中的[127.0.0.1](#)用集群中每个机器的主机名或IP地址替换：

```
cluster {  
    seed-nodes = ["akka.tcp://opendaylight-cluster-data@127.0.0.1:2550"]
```

- b. 找到如下内容并且给每个成员节点指定角色。例如，你可以给第一个节点分配角色**member-1**，第二个节点分配角色**member-2**，第三个节点分配角色**member-3**。

```
roles = [  
    "member-1"  
]
```

- c. 打开**configuration/initial/module-shards.conf**文件并且更新下列选项中的条目把副本在主机的**akka.conf**文件中进行定义。

```
replicas = [  
    "member-1"  
]
```

作为参考可以参照一个简单的**akka.conf**文件: <https://gist.github.com/moizr/88f4bd4ac2b03cfa45f0>

- a. 在每个集群节点上运行下面的命令：

- **JAVA_MAX_MEM=4G JAVA_MAX_PERM_MEM=512m ./karaf**
- **JAVA_MAX_MEM=4G JAVA_MAX_PERM_MEM=512m ./karaf**

- JAVA_MAX_MEM=4G JAVA_MAX_PERM_MEM=512m ./karaf

OpenDaylight控制器能够在三节点的集群中运行。可以使用三个成员节点中的任意一个访问数据存储中的数据。

如果想要获取关于member-1的节点分片设计信息，可以访问如下HTTP请求：

```
GET http://< host >:8181/jolokia/read/  
org.opendaylight.controller:Category=Shards,name=member-1-shard-inventoryconfig,  
type=DistributedConfigDatastore
```

注解

如果提示，在username和password栏都输入admin进行鉴权

这个请求将会返回如下信息：

```
{  
    "timestamp": 1410524741,  
    "status": 200,  
    "request": {  
        "mbean": "org.opendaylight.controller:Category=Shards, name=member-1-shardinventory-  
config, type=DistributedConfigDatastore",  
        "type": "read"  
    },  
    "value": {  
        "ReadWriteTransactionCount": 0,  
        "LastLogIndex": -1,  
        "MaxNotificationMgrListenerQueueSize": 1000,  
        "ReadOnlyTransactionCount": 0,  
        "LastLogTerm": -1,  
        "CommitIndex": -1,  
        "CurrentTerm": 1,  
        "FailedReadTransactionsCount": 0,  
        "Leader": "member-1-shard-inventory-config",  
        "ShardName": "member-1-shard-inventory-config",  
        "DataStoreExecutorStats": {  
            "activeThreadCount": 0,  
            "largestQueueSize": 0,  
            "currentThreadPoolSize": 1,  
            "maxThreadPoolSize": 1,  
            "totalTaskCount": 1,  
            "largestThreadPoolSize": 1,  
            "currentQueueSize": 0,  
            "completedTaskCount": 1,  
            "rejectedTaskCount": 0,  
            "maxQueueSize": 5000  
        },  
        "FailedTransactionsCount": 0,  
        "CommittedTransactionsCount": 0,  
        "NotificationMgrExecutorStats": {  
            "activeThreadCount": 0,  
            "largestQueueSize": 0,  
            "currentThreadPoolSize": 0,  
            "maxThreadPoolSize": 20,  
            "totalTaskCount": 0,  
            "largestThreadPoolSize": 0,  
            "currentQueueSize": 0,  
            "completedTaskCount": 0,  
            "rejectedTaskCount": 0,  
            "maxQueueSize": 1000  
        },  
        "LastApplied": -1,  
        "LastSynced": -1  
    }  
}
```

```

    "AbortTransactionsCount": 0,
    "WriteOnlyTransactionCount": 0,
    "LastCommittedTransactionTime": "1969-12-31 16:00:00.000",
    "RaftState": "Leader",
    "CurrentNotificationMgrListenerQueueStats": []
}
}

```

这里有个关键点是分片的命名。分片名称结构如下：

< member-name >-shard-< shard-name-as-per-configuration >-< store-type >

这里有一对简单的数据短名称：

- member-1-shard-topology-config
- member-2-shard-default-operational

多节点集群的HA（High Availability）功能

在一个三节点的集群中实现HA功能：

1. 在每个集群节点上打开configuration/initial/module-shards.conf文件。
2. 给每个data分片增加member-2和member-3的副本。
3. 重新启动所有节点。所有节点将自动同步member-1的配置。稍后，集群需要等待下一步操作。

当增加HA功能后，在每个分片中至少需要三个副本。每个节点的配置文件将像如下所示：

```

module-shards = [
{
    name = "default"
    shards = [
        {
            name="default"
            replicas = [
                "member-1",
                "member-2",
                "member-3"
            ]
        }
    ]
},
{
    name = "topology"
    shards = [
        {
            name="topology"
            replicas = [
                "member-1",
                "member-2",
                "member-3"
            ]
        }
    ]
},
{
    name = "inventory"
    shards = [
        {
            name="inventory"

```

```
replicas = [
    "member-1",
    "member-2",
    "member-3"
]
}
]
},
{
    name = "toaster"
    shards = [
        {
            name="toaster"
            replicas = [
                "member-1",
                "member-2",
                "member-3"
            ]
        }
    ]
}
]
```

当多个节点具备了HA功能后，分片将为那些节点复制数据。无论何时数据分片的lead副本失效，就会有另一个副本顶上来。因此，集群应该保持可用性。为了确定数据分片中的lead副本，可以通过HTTP请求去获得任意节点上的数据分片信息。返回的信息中将表明哪个是lead副本。

5 ALTO用户向导

概述

ALTO项目提供支持RFC 7285定义中的应用层流量优化（ALTO）服务。在Lithium版本中，ALTO使用YANG模型。

（注：YANG模型描述：<https://tools.ietf.org/html/draft-shi-alto-yang-model-03>）

ALTO架构

在OpenDaylight中，ALTO包主要有如下三种：

(1) 核心包。核心包包括：

- a) alto-model: 在MD-SAI里定义ALTO服务的YANG模型；
- b) service-api-rfc7285: 在AD-SAL里定义ALTO服务的端口；
- c) alto-northbound: 实现RFC兼容的RestAPI。

(2) 基础包。基础包包括：

- a) ALTO服务的基础实现：

-
1. alto-provider: 实现alto-model中定义的服务；
 2. simple-impl: 实现service-api-rfc7285中定义的服务。
-

- b) Utilities实用工具

1. alto-manager: 提供karaf命令行工具来操作network-map及cost-map。

(3) 服务包。服务包包括:

- a) alto-hosttracker: 生成网络图、相应的cost-map和基于L2switch端点成本服务。

配置ALTO

需要自己配置文件的有三种包，包括alto-provider、alto-hosttracker和simple-impl。但是，唯一的配置选项是所有三个配置文件中的数据代理类型。

实施和管理ALTO

为启动ALTO功能，以下features功能一定要先安装:

```
karaf > feature:install odl-alto-provider  
karaf > feature:install odl-alto-manager  
karaf > feature:install odl-alto-northbound  
karaf > feature:install odl-alto-hosttracker
```

通过RESTCONF管理数据

在karaf中安装odl-alto-provider以后，使用RESTCONF管理network-maps和cost-maps是可能的。在API服务页：<http://localhost:8181/apidoc/explorer/index.html>通过alto-model查看所有提供的选项。

随着下面的输入例子，通过在API doc页面填写表格，或使用如curl的工具，能将网络图插入到数据存储中。

```
HOST_IP=localhost # IP address of the controller  
CREDENTIAL=admin:admin # username and password for authentication  
BASE_URL=$HOST_IP:8181/restconf/config  
SERVICE_PATH=alto-service:resources/alto-service:network-maps/alto-  
service:network-map  
RESOURCE_ID=test_odl # Should match the one in the input file  
curl -X PUT -H "content-type:application/yang.data+json" \  
-d @example-input.json -u $CREDENTIAL \  
http://$BASE_URL/$SERVICE_PATH/$RESOURCE_ID
```

```
{  
"alto-service:network-map": [  
{  
"alto-service:map": [  
{  
"alto-service:endpoint-address-group": [  
{  
"alto-service:address-type": "ipv4",  
"alto-service:endpoint-prefix": [  
"192.0.2.0/24",
```

```

"198.51.100.0/25"
]
}
],
{
"alto-service:pid": "PID1"
},
{
"alto-service:endpoint-address-group": [
{
"alto-service:address-type": "ipv4",
"alto-service:endpoint-prefix": [
"198.51.100.128/25"
]
}
],
{
"alto-service:pid": "PID2"
},
{
"alto-service:endpoint-address-group": [
{
"alto-service:address-type": "ipv4",
"alto-service:endpoint-prefix": [
"0.0.0.0/0"
]
},
{
"alto-service:address-type": "ipv6",
"alto-service:endpoint-prefix": [
":/:/0"
]
}
],
{
"alto-service:pid": "PID3"
}
],
{
"alto-service:resource-id": "test_odl",
"alto-service:tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
}
]
}

```

使用以下命令查看结果：

```

HOST_IP=localhost # IP address of the controller
CREDENTIAL=admin:admin # username and password for authentication
BASE_URL=$HOST_IP:8181/restconf/config
SERVICE_PATH=alto-service:resources/alto-service:network-maps/alto-
service:network-map
RESOURCE_ID=test_odl
curl -X GET -u $CREDENTIAL http://$BASE_URL/$SERVICE_PATH/$RESOURCE_ID

```

使用****DELETE****方法移除数据存储中的数据：

```

HOST_IP=localhost # IP address of the controller
CREDENTIAL=admin:admin # username and password for authentication
BASE_URL=$HOST_IP:8181/restconf/config
SERVICE_PATH=alto-service:resources/alto-service:network-maps/alto-
service:network-map
RESOURCE_ID=test_odl

```

```
curl -X DELETE -H "content-type:application/yang.data+json" \
-u $CREDENTIAL http://$BASE_URL/$SERVICE_PATH/$RESOURCE_ID
```

使用alto-manager

alto-manager包提供karaf命令行工具，在最后一节中描述的封装功能。

```
karaf > alto-create
karaf > alto-delete
```

现在只有network-map和cost-map是支持的。利用alto-manager的源文件按照RFC7285兼容格式代替RESTCONF格式。

下面的例子演示了如何使用alto-manager将network map映射到数据存储中。

```
karaf > alto-create network-map example-rfc7285-networkmap.json
{
  "meta": {
    "resource-id": "test_odl",
    "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
  },
  "network-map": {
    "PID1": {
      "ipv4": [
        "192.0.2.0/24",
        "192.51.100.0/25"
      ]
    },
    "PID2": {
      "ipv4": [
        "192.51.100.128/25"
      ]
    },
    "PID3": {
      "ipv4": [
        "0.0.0.0/0"
      ],
      "ipv6": [
        "::/0"
      ]
    }
  }
}
```

使用alto-hosttracker

作为ALTO服务的真实实例，alto-hosttracker从l2switch中读取数据并产生带有资源id hosttracker-network-map的网络图network-map和带有资源ID hostracker-cost-map的cost-map。它只能在OpenFlow启用的网络下工作。

安装odl-alto-hosttracker feature功能后，相应的network-map和cost-map将被插入到数据存储中，按照使用alto-manager章节通过RESTCONF读取数据查看内容的步骤进行操作。

6 身份认证和授权服务

身份认证服务

身份认证是使用用户提供的凭证来识别用户。

注：锂版本提供的身份认证用户存储不完全支持集群节点部署。具体地，H2数据库提供的AAA级用户存储需要同步。然而cluster-capable是AAA级的令牌缓存。

身份认证数据模型

用户在已经定义角色的域中请求身份验证，可以选择下列方式：

- 提供认证信息
- 在域中创建一个令牌。OpenDaylight中，域就是具有访问控制目的的资源分组(直接或间接、物理、逻辑或虚拟)。

模型中的术语及其定义

Token 对控制器上一组资源的访问声明。

Domain 一组用于访问控制的资源，直接或间接、物理、逻辑或虚拟。

User 拥有或能够获得控制器上资源的人。

role 一组权限的非透明表示，这仅仅是作为管理员或客户的一个独一无二的字符串。

Credential 用户名和密码、OTP、生物识别技术等身份的证明。

Client 一个请求访问控制器的服务或应用程序。

Claim 关于用户验证数据的集合，如：role、domain、name等。

身份认证方法

在OpenDaylight中用户有三种方式进行身份认证：

- 基于HTTP的身份认证
 - 规律地、不基于令牌的、用户名/密码认证的。
 - 基于令牌的身份认证
 - 直接认证：用户提供用户名/密码和用户希望访问控制器，获得定时的(默认为1小时)的访问令牌的domain。然后用户使用这个令牌访问如RESTCONF。
 - 联合身份验证：用户向受信于控制器的第三方身份提供者(如：SSSD)提供认证信息。身份认证成功之后，控制器返回一个更新的(unscoped)令牌，令牌内包含用户能访问的domain列表。然后用户提供这个更新的新令牌获得一个域内的访问令牌。然后用户使用这个访问令牌访问如RESTCONF。

使用curl令牌认证示例：

(username/password = admin/admin, domain = sdn)

```
**Create a token**
curl -ik -d 'grant_type=password&username=admin&password=admin&scope=sdn'
http://localhost:8181/oauth2/token
```

```
**Use the token** (e.g., ed3e5e05-b5e7-3865-9f63-eb8ed5c87fb9) obtained from  
above (default token validity is 1 hour):  
curl -ik -H 'Authorization:Bearer ed3e5e05-b5e7-3865-9f63-eb8ed5c87fb9' http://  
/localhost:8181/restconf/config/toaster:toaster
```

使用curl进行HTTP认证示例：

```
curl -ik -u 'admin:admin' http://localhost:8181/restconf/config/  
toaster:toaster
```

OpenDaylight身份认证服务是如何工作的

在直接认证方式中，用户和OpenDaylight控制器之间存在着服务关系。用户和控制器间建立信任，让用户使用并验证凭证。用户通过认证信息建立用户身份。

在直接认证方式中，用户请求步骤如下：

1. 用户向控制器管理员请求一个用户账号。

与用户帐户相关联的是管理员最初创建的用户认证信息。OpenDaylight仅支持用户名/密码凭证。默认情况下，管理员账户存在于OpenDaylight的out-of-the-box中，默认的用户名和密码是admin/admin。除了创建用户账户以外，控制器管理员也会给一个或多个domain分配角色。默认情况下，有两个用户角色，admin和user，只有一个domain，就是sdn。

2. 用户提供令牌中的认证信息请求在domain中的令牌服务。

3. 然后请求信息会传递到控制器的令牌终端。

4. 控制器的令牌终端使用认证证书给客户端返回一个claim。

5. 控制器令牌实体将返回的claim（user、domain和roles）传递给令牌，然后提供给用户。

在联合身份认证方式中，用户和用于身份验证的第三方身份提供者(IdP)没有直接的信任关系。联合身份验证依赖于第三方身份提供者(IdP)认证用户。

用户通过受信的IdP和返回给OpenDaylight认证服务的claim进行身份认证。claim被转化为一个OpenDaylight claim并且传递给用户的令牌。

在联合身份认证的设置中，OpenDaylight控制器AAA module提供SSSD claim支持。SSSD可用于映射外部LDAP服务器的用户到OpenDaylight控制器中定义的用户。

配置身份认证服务

AAA配置如下：

通过如下之一的实现身份认证功能：

- Webconsole
- CLI (Karaf shell中的配置命令)
- 直接编辑etc/org.opendaylight.aaa.*.cfg文件

通过如下之一的方式进行令牌数据缓存设置：

- 编辑etc/opendaylight/karaf目录下的08-authn-config.xml配置文件

- 使用RESTCONF

注：AAA的配置都是动态的，不要求重启。

配置身份认证：

在Web控制台配置features：

1.安装Web控制台：

```
feature:install webconsole
```

2.从控制台(<http://localhost:8181/system/console>)（默认Karaf用户名/密码：`karaf/karaf`）进入OSGi > Configuration > OpenDaylight AAA Authentication Configuration

a.授权客户端：列出有权访问OpenDaylight北向API的软件客户端。

b.启用身份认证：启用/禁用身份认证。（默认是启用）

配置令牌存储

1.在文本编辑器中打开`etc/opendaylight/karaf/08-authn-config.xml`，可以编辑的区域如下：

a.**timeToLive**配置最长时间，以毫秒为单位，令牌被缓存。默认是360000。

2.保存文件。

注：令牌过期时会从缓存中移除。

配置AAA federation

1.在控制台点击OpenDaylight AAA Federation Configuration。

2.使用Custom HTTP Headers和Custom HTTP Attributes字段为联合身份认证指定HTTP头部和属性。通常情况下，除了默认的规范之外没有其他需要改动的。

注：当你改动了配置文件保存好后会自动提交，不需要重启。

配置联合身份认证

根据如下步骤设置联合身份认证：

- 1.为OpenDaylight控制器设置Apache前端和插件。
- 2.设置映射规则（从LDAP用户到OpenDaylight用户）。
- 3.使用federation中的ClaimAuthFilter进行claim转换。

映射用户到role和domain中

OpenDaylight身份认证服务从外部联合IdP转换到身份认证服务数据：

- 1.OpenDaylight AAA前面的Apache网站服务器传数据到SssdAuthFilter。
- 2.SssdAuthFilter将数据构造成JSON文件。

3. OpenDaylight身份认证服务使用通用的映射转换JSON文件。

操作模型

操作模型的工作如下：

- 1.IdP中的声明存储在关联矩阵中。
- 2.一系列的规则被应用，成功返回的第一个规则认为是一个匹配。
- 3.一旦成功后，就会返回映射规则的关联矩阵。
 - 映射的值从规则执行中的局部变量中获取。
 - 规则和映射结果的定义用JSON格式。

操作模型：示例代码

```
mapped = null
foreach rule in rules {
    result = null
    initialize rule.variables with pre-defined values
    foreach block in rule.statement_blocks {
        for statement in block.statements {
            if statement.verb is exit {
                result = exit.status
                break
            }
            elif statement.verb is continue {
                break
            }
            if result {
                break
            }
            if result == null {
                result = success
            }
            if result == success {
                mapped = rule.mapping(rule.variables)
            }
        }
    }
    return mapped
```

映射用户

JSON对象充当生成最终name/value对的关联数组的映射模板。name/value中的value可以是常量或者变量。JSON格式的映射模板和规则变量的例子：

模板：

```
{
    "organization": "BigCorp.com",
    "user": "$subject",
    "roles": "$roles"
}
```

局部变量:

```
{
  "subject": "Sally",
  "roles": ["user", "admin"]
}
```

最终的映射结果:

```
{
  "organization": "BigCorp.com",
  "user": "Sally",
  "roles": ["user", "admin"]
}
```

例子：将完整的用户名拆分成用户和域组件

一些IdP返回一个完整的用户名(例如，`principal`或`subject`)。完整的用户名是将用户名、分离器和域名串联起来。下面的例子显示了映射结果分别返回用户和域，而完整的用户名是`bob@example.com`。

JSON格式的映射:

```
{
  "user": "$username",
  "realm": "$domain"
}
```

JSON格式的assertion:

```
{
  "Principal": "bob@example.com"
}
```

应用的规则:

```
[
  [
    [
      "in", "Principal", "assertion"],
      ["exit", "rule_fails", "if_not_success"],
      ["regexp", "$assertion[Principal]", "(?P<username>\w+)@(?P<domain>.+)"],
      ["set", "$username", "$regexp_map[username]"],
      ["set", "$domain", "$regexp_map[domain]"],
      ["exit", "rule_succeeds", "always"]
    ]
]
```

JSON格式的映射结果:

```
{
  "user": "bob",
  "realm": "example.com"
}
```

同时，用户在特定的组成员关系中可能会被授予角色。<同样，会根据用户在组中的成员关系为其分配角色。>

身份认证服务允许拥有特定身份的用户进入白名单，白名单可以确保特定身份的用户无条件的被接受和认可。被无

条件拒绝访问的用户可以放置在黑名单中。

管理OpenDaylight身份认证服务

系统中的参与者

OpenDaylight控制器管理员 OpenDaylight控制器管理员有以下职责:

- 作者身份认证策略使用IdmLight服务API
- 给发出请求的用户提供认证、用户名和密码

OpenDaylight资源拥有者 资源所有者(通过联合或直接提供自己的认证信息给控制器)获得一个访问令牌。这个访问令牌可以用来访问资源控制器。一个OpenDaylight资源所有者享受以下特权:

- 创建、更新、删除访问令牌
- 从Secure Token Service中获取令牌
- 将安全令牌授予资源用户

OpenDaylight资源使用者 资源使用者不需要进行认证: 如果他们得到资源拥有者给的访问令牌, 就能访问资源。获取令牌的默认时间期限是1个小时(这个时间是可以配置的)。OpenDaylight资源用户做以下的事情:

- 从资源拥有者或者控制器管理员那获得令牌
- 使用令牌从北向API访问应用

系统组件

IdmLight Identity manager 存储本地用户身份验证和授权数据, 为CRUD操作提供了一个Admin REST API。

Pluggable authenticators 提供特定domain的认证机制

Authenticator 对用户进行身份认证, 建立claim

Authentication Cache 缓存所有的身份认证状态和令牌

Authentication Filter 认证令牌和提取claim

Authentication Manager 包含会话令牌和认证的claim存储

IdmLight身份管理员

轻量级Identity Manager(IdmLight)存储本地用户的身份认证和授权数据和角色, 并在users/roles/domains数据库提供CRUD操作的Admin REST API。IdmLight REST API默认是通过{controller baseURI:8181}/auth/v1/API终端获取的。API的获取只受制于已认证的客户端或者那些拥有一个令牌的:

例如: 获取用户列表。

```
curl http://admin:admin@localhost:8181/auth/v1/users
```

下面的文档包含API允许支持的CRUD操作的详细列表:

```
https://wiki.opendaylight.org/images/a/ad/AAA_Idmlight_REST_APIs.xlsx
```

OpenDaylight授权服务

目前OpenDaylight中的授权服务是一种实验性的服务，这里只简要记录。授权基于成功的身份验证，仿照基于角色的访问控制(RBAC)方法定义权限，决定访问控制器中API资源的级别。

7 BGP用户指南

Overview

OpenDaylight的Karaf distribution会对基线版本的BGP进行预配置。可以在目录etc/opendaylight/karaf下找到两个文件：

- 31-bgp.xml (定义基本解析和RIB支持)
- 41-bgp-example.xml (包含了一个定制化部署的简单配置)

下面的内容将描述如何手动或者通过RESTCONF配置BGP。

配置BGP

RIB

```
<module>
    <type>prefix:rib-impl</type>
    <name>example-bgp-rib</name>
    <rib-id>example-bgp-rib</rib-id>
    <local-as>64496</local-as>
    <bgp-id>192.0.2.2</bgp-id>
    <cluster-id>192.0.2.3</cluster-id>
    ...
</module>
```

- **rib-id:** BGP RIB Identifier, 在这个配置文件中你可以通过复制粘贴的上方代码的方式指定很多BGP RIB。这些RIB必须有独一无二的rib-id和名称。
- **local-as:** 本地AS编码（部署OpenDaylight的位置），我们把这个用于最佳路径选择
- **bgp-id:** 本地BGP身份（部署OpenDaylight的虚拟机IP），用于最佳路径选择
- **cluster-id:** Cluster Identifier，非强制部署，如无详细说明则只使用BGP Identifier

部分配置不是不必要的：主要取决于你的BGP路由器，你可能需要将链路状态属性类型在99和29中进行切换。检查路由器厂商版本。如果路由器支持类型为29，可以对iana-linkstate-attribute-type域进行更改。这位于本地文件31-bgp.xml中。

```
<module>
    <type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:linkstate">prefix:bgplinkstate</
type>
    <name>bgp-linkstate</name>
    <iana-linkstate-attribute-type>true</iana-linkstate-attribute-type>
</module>
```

- **iana-linkstate-attribute-type:** IANA对BGP链路状态属性类型 (=29) 发布了早期配置。当TYPE=99时设

置值为false： 使用IANA指定类型29，可以将值设置为true或者删除（默认删除为true）。

BGP Peer

客户端启动使用默认配置时会忽略初始化配置。因此第一步是修改包含bgp-peer的模块。

```
<module>
  <type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">prefix:bgppeer</
type>
  <name>example-bgp-peer</name>
  <host>192.0.2.1</host>
  <holdtimer>180</holdtimer>
  <peer-role>ibgp</peer-role>
  <rib>
    <type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:cfg">prefix:rib</type>
      <name>example-bgp-rib</name>
      </rib>
    ...
  </module>
```

- **name:** BGP Peer的名称，在这个配置文件中你可以通过复制粘贴上述模块指定很多的BGP Peer。
- **host:** BGP speaker的IP地址或主机名（OpenDaylight连接全局拓扑的IP）
- **holdtimer:** 单位：秒
- **peer-role:** 如果peer的角色没有声明，将使用默认值"ibgp"（角色的值也可以是"ebgp"或"rr-client"）。这个地方区分大小写。
- **rib:** BGP RIB身份证明

连接属性配置-可选

```
<module>
  <type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:reconnectstrategy">prefix:timedreconnect-
strategy</type>
  <name>example-reconnect-strategy</name>
  <min-sleep>1000</min-sleep>
  <max-sleep>180000</max-sleep>
  <sleep-factor>2.00</sleep-factor>
  <connect-time>5000</connect-time>
  <executor>
    <type xmlns:netty=
"urn:opendaylight:params:xml:ns:yang:controller:netty">netty:netty-eventexecutor</
type>
      <name>global-event-executor</name>
      </executor>
  </module>
```

- **min-sleep-**尝试重新连接的最小睡眠时间（毫秒）
- **max-sleep-**尝试重新连接的最大睡眠时间（毫秒）
- **sleep-factor-**尝试重新连接睡眠时间的功率因素

- connect-time-TCP尝试连接的等待时间， 默认连接时间超时TCP将进行重传

BGP Speaker配置

之前的条目解决了OpenDaylight初始化的BGP连接配置。OpenDaylight同时也支持BGP Speaker功能和接收Incomming BGP连接。

BGP Speaker在本地的配置文件41-bgp-example.xml中：

```
<module>
  <type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">prefix:bgp-peeracceptor</
  type>
    <name>bgp-peer-server</name>
    <!--Default parameters-->
    <!--<binding-address>0.0.0.0</binding-address>-->
    <!--<binding-port>1790</binding-port>-->
    ...
    <!--Drops or accepts incoming BGP connection, every BGP Peer that should
be accepted needs to be added to this registry-->
    <peer-registry>
      <type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">prefix:bgp-peerregistry</
      type>
        <name>global-bgp-peer-registry</name>
        </peer-registry>
    </module>
```

修改speaker配置

修改绑定地址：取消地址绑定的tag并且修改地址(例如修改为[127.0.0.1](#))。默认绑定地址为[0.0.0.0](#)。

修改绑定端口：取消地址绑定的tag并且修改端口(例如修改为1790)。默认绑定端口为179 (详见[BGP RFC](#))。

Incomming BGP连接

BGP Speaker丢弃所有来自未知**BGP Peer**的**BGP**连接。这个由添加到Speaker中的bgp-peer-registry配置(registry在31-bgp.xml中注册)。

如果需要配置BGP peer到注册表中，只需在例如41-bgp-example.xml文件中配置常规的BGP Peer。注意：BGP peer取决于相同的bgp-peer-registry和bgp-speker:

```
<module>
  <type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">prefix:bgppeer</
  type>
    <name>example-bgp-peer</name>
    <host>192.0.2.1</host>
    ...
    <peer-registry>
      <type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">prefix:bgp-peerregistry</
      type>
        <name>global-bgp-peer-registry</name>
        </peer-registry>
    ...
  </module>
```

BGP peer自身可以注册到注册表中，允许进入bgp-speaker处理的BGP连接。（peer-registry的配置属性现在是可选的，并且向后兼容）。通过配置，OpenDaylight将初始化到192.0.2.1的连接并且也会接收到来自192.0.2.1的连接。在实际中两个连接都将被建立，其中只有一个会保持，另一个会被丢弃。低级别bgp的设备在初始化连接时会被注册表丢弃。每个BGP peer必须在自己的模块中进行配置。注意，模块的名字需要有区分，所以在配置很多peer的情况下更改主机时也要将名字一并更改。有一种只用于传入连接的配置peer方法（这个连接不会被OpenDaylight初始化，OpenDaylight只会等待对端peer的传入连接。对端peer用他的IP地址标识id）。配置仅用于传入连接属性initiate-connection添加到对端peer。

```
<module>
  <type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">prefix:bgppeer</
type>
  <name>example-bgp-peer</name>
  <host>192.0.2.1</host> // IP address or hostname
of the speaker
  <holdtimer>180</holdtimer>
  <initiate-connection>false</initiate-connection> // Connection will not
be initiated by ODL
  ...
</module>
```

initiate-connection: 如果将值设为false，OpenDaylight将不会初始化到这个peer的连接，每个peer的默认设置值为true。

BGP Application Peer

一个BGP speaker需要注册所有能连接到他的peer（也就是说一个没配置的BGP peer，OpenDaylight不能与其成功连接）。第一步是配置RIB。然后，用自己的RIB应用配置peer应用以取代peer的固定配置。修改bgp-peer-id的值为本地BGP-ID，该id用于BGP最佳路径选择算法。

```
<module>
  <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:bgpapplication-
peer</type>
  <name>example-bgp-peer-app</name>
  <bgp-peer-id>10.25.1.9</bgp-peer-id>
  <target-rib>
    <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:rib-instance</
type>
    <name>example-bgp-rib</name>
  </target-rib>
  <application-rib-id>example-app-rib</application-rib-id>
  ...
</module>
```

bgp-peer-id: 本地BGP的id（部署OpenDaylight的虚拟机IP），用于最佳路径选择

target-rib: 数据传输所用的已存在RIB的ID

application-rib-id: 本地appplcation RIB的ID（所有在OpenDaylight设置的路由都将在那里显示）

通过RESTCONF配置

另一种配置BGP的方法是通过RESTCONF进行动态配置。在启动前确认你已经完成了安装指南的1-5步骤。这时不用再重启Karaf，而是去加载另一个feature，该feature将提供访问restconf/config/URL的功能。

```
feature:install odl-netconf-connector-all
```

通过下面链接获取已配置的模块: `localhost:8181/restconf/config/network-topology:network-topology/topology/topologynetconf/node/controller-config/yang-ext:mount/config:modules/` 这个URL也用于POST新配置。如果想修改此列表中的任意其他配置, 确定包括了正确的命名空间, 即可。RESTCONF将会反馈命名空间的错误。

使用**PUT**或绝对路径更新已存在的配置。

遵循用户指南中的描述的步骤至关重要。

RIB

首先, 设置RIB。这个模块已经进行了配置, 我们要做的就是去修改我们需要的参数。在这个例子中, 修改**bgp-rib-id**和**local-as**。

URL: `http://127.0.0.1:8181/restconf/config/network-topology:network-topology/topologynetconf/node/controller-config/yang-ext:mount/config:modules/module/odl-bgp-rib-impl-cfg:bgp-rib/example-bgp-rib`

PUT:

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
  <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:rib-impl</
type>
  <name>example-bgp-rib</name>
  <session-reconnect-strategy xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
    <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:protocol:framework">x:reconnectstrategy-
factory</type>
    <name>example-reconnect-strategy-factory</name>
  </session-reconnect-strategy>
  <rib-id xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">example-bgprib</
rib-id>
  <extensions xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
    <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:spi">x:extensions</
type>
    <name>global-rib-extensions</name>
  </extensions>
  <codec-tree-factory xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
    <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding">x:bindingcodec-
tree-factory</type>
    <name>runtime-mapping-singleton</name>
  </codec-tree-factory>
  <tcp-reconnect-strategy xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
    <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:protocol:framework">x:reconnectstrategy-
factory</type>
    <name>example-reconnect-strategy-factory</name>
  </tcp-reconnect-strategy>
  <data-provider xmlns=
```

```
urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
    <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding">x:bindingasync-
data-broker</type>
    <name>pingpong-binding-data-broker</name>
</data-provider>
<local-as xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">64496</local-as>
    <bgp-dispatcher xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
        <type>bgp-dispatcher</type>
        <name>global-bgp-dispatcher</name>
    </bgp-dispatcher>
    <dom-data-provider xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
        <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:md:sal:dom">x:dom-async-databroker</
type>
        <name>pingpong-broker</name>
    </dom-data-provider>
    <local-table xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
        <type>bgp-table-type</type>
        <name>ipv4-unicast</name>
    </local-table>
    <local-table xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
        <type>bgp-table-type</type>
        <name>ipv6-unicast</name>
    </local-table>
    <local-table xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
        <type>bgp-table-type</type>
        <name>linkstate</name>
    </local-table>
    <local-table xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
        <type>bgp-table-type</type>
        <name>flowspec</name>
    </local-table>
    <bgp-rib-id xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">192.0.2.2</bgprib-
id>
</module>
```

Important: 需不需要取决于你的BGP路由器，你可能需要在链路状态属性类型99和29之间进行选择。联系你的路由器厂商。如果路由器支持类型29可以选择为true。

URL: <http://127.0.0.1:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules/module/odl-bgp-linkstate-cfg:bgp-linkstate>

PUT:

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
    <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:linkstate">x:bgplinkstate</
type>
    <name>bgp-linkstate</name>
    <iana-linkstate-attribute-type xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:linkstate">true</ianalinkstate-
attribute-type>
```

```
</module>
```

BGP Peer

我们同样需要对bgp-peer进行配置。在这个例子中，需要对整个模块进行配置。请更改**host**、**holdtimer**和**peer-role**（如果需要的话）的值。

POST:

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
  <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:bgp-peer</
type>
  <name>example-bgp-peer</name>
  <host xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">192.0.2.1</host>
  <holdtimer xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">180</holdtimer>
  <peer-role xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">ibgp</peer-role>
  <rib xmlns="urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:cfg">x:rib</type>
    <name>example-bgp-rib</name>
  </rib>
  <peer-registry xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
    <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:bgp-peerregistry</
type>
      <name>global-bgp-peer-registry</name>
    </peer-registry>
    <advertized-table xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
      <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:bgp-tabletype</
type>
        <name>ipv4-unicast</name>
      </advertized-table>
      <advertized-table xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
        <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:bgp-tabletype</
type>
          <name>ipv6-unicast</name>
        </advertized-table>
        <advertized-table xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
          <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:bgp-tabletype</
type>
            <name>linkstate</name>
          </advertized-table>
          <advertized-table xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
            <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:bgp-tabletype</
type>
              <name>flowspec</name>
            </advertized-table>
```

```
</module>
```

你所需求的信息可以通过连接speaker的ODL获取。

BGP Application Peer

修改**bgp-peer-id**也就是本地BGP ID的值，用于BGP最佳路径选择算法。

POST:

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
  <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:bgpapplication-
peer</type>
  <name>example-bgp-peer-app</name>
  <bgp-peer-id xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">10.25.1.9</bgppeer-
id> <!-- Your local BGP-ID that will be used in BGP Best Path Selection
algorithm -->
  <target-rib xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
    <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:rib-instance</
type>
    <name>example-bgp-rib</name>
  </target-rib>
  <application-rib-id xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">example-apprib</
application-rib-id>
  <data-broker xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
    <type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding">x:bindingasync-
data-broker</type>
    <name>pingpong-binding-data-broker</name>
  </data-broker>
</module>
```

教程

查看BGP拓扑

将简单介绍能够通过RESTCONF查看BGP数据。这也是当前查看这些数据的唯一方式。

Important: 从Helium版本开始端口从8080变为8181.

网络拓扑视图

网络拓扑的基本URL是：<http://localhost:8181/restconf/operational/networktopology:network-topology/>

如果正确的配置了BGP，应该会像下面这样显示：

```
<network-topology>
  <topology>
    <topology-id>pcep-topology</topology-id>
    <topology-types>
```

```

<topology-pcep/>
</topology-types>
</topology>
<topology>
  <server-provided>true</server-provided>
  <topology-id>example-ipv4-topology</topology-id>
  <topology-types/>
</topology>
<topology>
  <server-provided>true</server-provided>
  <topology-id>example-linkstate-topology</topology-id>
  <topology-types/>
</topology>
</network-topology>

```

BGP数据由BGP speaker发送，如果三个拓扑都已配置会展示三个拓扑：

example-linkstate-topology: 通过链路状态更新信息展示链接和节点，链

接：<http://localhost:8181/restconf/operational/network-topology:network-topology/topology/example-linkstate-topology>

example-ipv4-topology: 展示拓扑中节点的IPv4地址，链

接：<http://localhost:8181/restconf/operational/network-topology:network-topology/topology/example-ipv4-topology>

example-ipv6-topology: 展示拓扑中节点的IPv6地址，链

接：<http://localhost:8181/restconf/operational/network-topology:network-topology/topology/example-ipv6-topology>

路由信息库（RIB）视图

另一个视图通过**BGP RIBs**提供BGP数据，URL如下：<http://localhost:8181/restconf/operational/bgp-rib:bgp-rib/>

这里有多个配置的RIB：

AdjRibsIn(per Peer): Adjacency RIB In, 来自BGP Peer的BGP路由信息

EffectiveRib(per Peer): 在导入策略后的BGP路由信息

LocRib(per RIB): Local RIB, 所有BGP Peer的BGP路由信息

AdjRibsOut(per Peer): Adjacency RIB Out, 在使用外部的策略后, 将通知BGP路由

当配置IPv4地址和链路状态时，输出是这样的：

```

<loc-rib>
  <tables>
    </attributes>
    <safi>x:linkstate-subsequent-address-family</safi>
    <afi>x:linkstate-address-family</afi>
    </linkstate-routes>
  </tables>
  <tables>
    </attributes>
    <safi>x:unicast-subsequent-address-family</safi>
    <afi>x:ipv4-address-family</afi>
    </ipv4-routes>
  </tables>
</loc-rib>

```

你可以通过扩展这些RESTCONF连接看到每个AFI的细节；

IPv4: <http://localhost:8181/restconf/operational/bgp-rib:bgp-rib/rib/example-bgp-rib/locrib/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/ipv4-routes>

Linkstate: <http://localhost:8181/restconf/operational/bgp-rib:bgp-rib/rib/example-bgprib/loc-rib/tables/bgp-linkstate:linkstate-address-family/bgp-linkstate:linkstate-subsequentaddress-family/linkstate-routes>

RIB填充

如果已经配置了peer，可以通过调用下面的RESTCONF填充RIB：

URL: <http://localhost:8181/restconf/config/bgp-rib:application-rib/example-app-rib/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/>

example-app-rib就是添加application RIB id（在配置文件中指定的）和指定AFI与SAFI表数据的位置

POST:

Content-Type: application/xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
< ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-rib">
    < ipv4-route>
        < prefix>200.20.160.1/32</prefix>
        < attributes>
            < ipv4-next-hop>
                < global>199.20.160.41</global>
            </ipv4-next-hop><as-path/>
            < multi-exit-dsc>
                < med>0</med>
            </multi-exit-dsc>
            < local-pref>
                < pref>100</pref>
            </local-pref>
            < originator-id>
                < originator>41.41.41.41</originator>
            </originator-id>
            < origin>
                < value>igp</value>
            </origin>
            < cluster-id>
                < cluster>40.40.40.40</cluster>
            </cluster-id>
        </attributes>
    </ipv4-route>
</ipv4-routes>
```

这个请求的响应如预期的，是**204 No content**。

8 CAPWAP User Guide

概述

CAPWAP功能填补了OpenDaylight控制器在管理CAPWAP兼容企业网络中无线终端设备方面的空白，并且可以通过REST APIs利用WTP网络设备的运行状态开发智能应用（如集中式防火墙管理、无线电规划）。

CAPWAP架构

CAPWAP作为一个MD-SAL基础提供者模块，负责帮助发现WTP设备并且更新在MD-SAL operational数据存储器中的设备状态。

CAPWAP项目的适用范围

锂版本中CAPWAP项目只是为了探测WTPs设备并且将设备的基础特征存储到operational数据存储器中，可以通过REST和java APIs进行访问。

安装CAPWAP

下载OpenDaylight，在karaf控制台安装odl-capwap-ac-rest功能组件。

配置CAPWAP

目前锂版本还没有配置要求

管理CAPWAP

通过karaf控制台安装好功能组件odl-capwap-ac-rest后，用户可以利用APIDOCs资源管理器管理CAPWAP。

访问[http://\\${ipaddress}:8181/apidoc/explorer/index.htm](http://${ipaddress}:8181/apidoc/explorer/index.htm)，登录并且扩展capwap-impl面板，用户可以执行多种API调用。

教程

查看发现的WTPs

概述

本教程可以作为一篇参考文档，学习启动CAPWAP功能组件，探测CAPWAP WTPs，访问WTPs的operational状态。

先决条件

用户必须有权访问一个基于CAPWAP兼容WTP的软件/硬件。该设备应该配置OpenDaylight控制器的IP地址，作为CAPWAP访问控制器地址。此外，WTPs和OpenDaylight控制器应该共享同一个以太网广播域。

使用说明

- 1.运行OpenDaylight并且通过karaf控制台安装odl-capwap-ac-rest
- 2.访问[http://\\${ipaddress}:8181/apidoc/explorer/index.html](http://${ipaddress}:8181/apidoc/explorer/index.html)
- 3.扩展capwap-impl

4.点击/operational/capwap-impl:capwap-ac-root/

5.点击"Try it out"

6.以上步骤展示出ODL CAPWAP功能组件发现的WTPs列表

9 DIDM用户指南

概述

设备识别和驱动管理（DIDM）项目解决了提供指定设备功能的需求。指定设备功能是实现某个功能的代码，这个代码了解设备的功能和缺陷。例如，配置VLANs和调整FlowMods特征，以及不同设备类型可能有不同的实现。指定设备功能作为设备驱动实现，设备驱动需要和配套使用的设备相挂钩，决定这种挂钩的是识别设备类型的能力。

DIDM架构

DIDM创建了一个架构，支持以下功能：

发现：检测控制器管理域中的设备并且建立连接。对支持OpenFlow的设备而言，OpenDaylight已有的发现机制可以满足需求。不支持OpenFlow的设备需要采用手动方式，通过GUI或REST API访问设备信息。

识别：识别设备类型。

驱动注册：以路由RPC的形式注册设备驱动。

同步：搜集设备信息、设备配置和链路（连接）信息。

通用功能的数据模型：定义该数据模型来实现如VLAN配置这样的通用功能。例如，应用可以按照通用数据模型指定的方式向数据存储写VLAN数据来配置一个VLAN。

通用功能的RPC：配置VLAN和调整FlowMod都是功能示例，定义RPC就是为了给这些功能指定API。驱动为指定的设备实现功能并且支持RPC定义APIs。不同的设备类型可能会有不同的驱动实现。

10.Group Based Policy用户指南

概述

OpenDaylight组策略允许用户以陈述的方式而不是命令的方式描述网络配置。例如会使用“what you want”而不是“how to do it”。

为了实现组策略需要采用一种意图系统。一个意图系统：

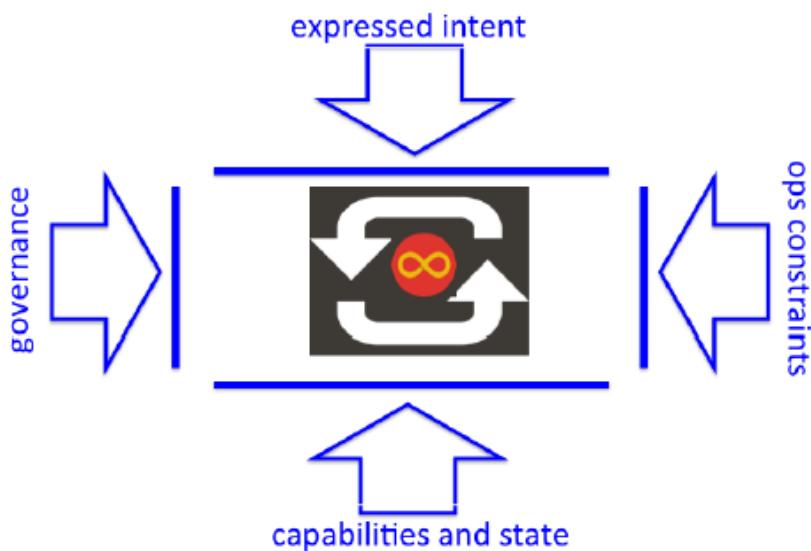
是一个意图驱动数据模型的流程

不包含特定的域

意图是能够处理多个语义定义

为此，BGP策略直观的将意图系统视为：

Figure 10.1. Intent System Process and Policy Surfaces



expressed intent是系统入口。

operational constraints提供使用该系统的方法，例如“所有金融应用必须使用特定的加密标准”。

capabilities and state由**renderers**提供。**renderer**动态地向核心模块提供自己的功能，允许核心模块保持没有特定的域。

governance是交付意图后的反馈。例如“Did we do what you asked us?”

总之，BGP的意图是自动化。

这样思考意图系统，就可以发现它实现了：

意图的自动化：通过专注于模型、处理和自动化，一系列连续的策略解析过程确保**expressed intent**和**renderer**（负责提供实现意图的功能）之间的映射。

递归/意图独立行为：一个人的具体是另一个人的抽象，可以通过不特定的域解析分层实现意图。在一个策略解析实例中，**renderer**提供特定的域，并通过API进行开放。例如：

DNS：域名“www.foo.com”是抽象的，而IPV4地址**10.0.0.10**是具体的。

IP堆栈：**10.0.0.10**是抽象的，MAC地址0: 05: 04: 03: 02: 01是具体的。

以太网交换机：MAC地址08:05:04:03:02:01是抽象的，CAM表中解析为一个端口是具体的。

光网络：端口可能是抽象的，光波长是具体的。

注意：以上是一个很简单的类比，相信大多数读者都能理解。这并不是代表GBP应该在OSI模型中实现。实现一个意图系统，可以将用户从描述意图的繁琐工作中解放出来。

在设定项目方向的时候应该展示出GBP整体的思路。锂版本中，GBP侧重于**expressed intent**和**capabilities**。

GBP架构和价值定位

专业术语

为了介绍**GBP**基本的价值主张，举一个例子。首先需要定义一些专业术语。

访问模型是**GBP**意图系统解析过程的核心。

Figure 10.2. GBP Access Model Terminology - Endpoints, EndpointGroups, Contract

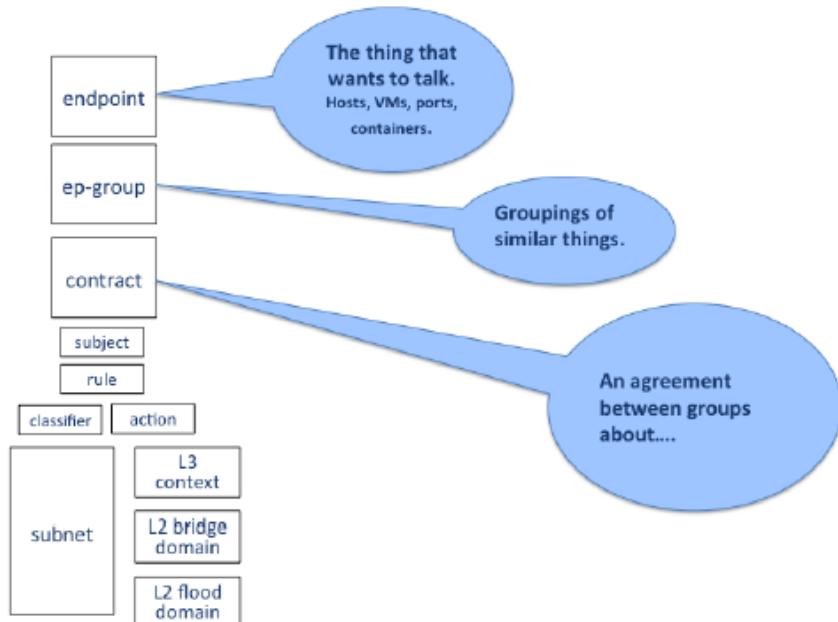


Figure 10.3. GBP Access Model Terminology - Subject, Classifier, Action

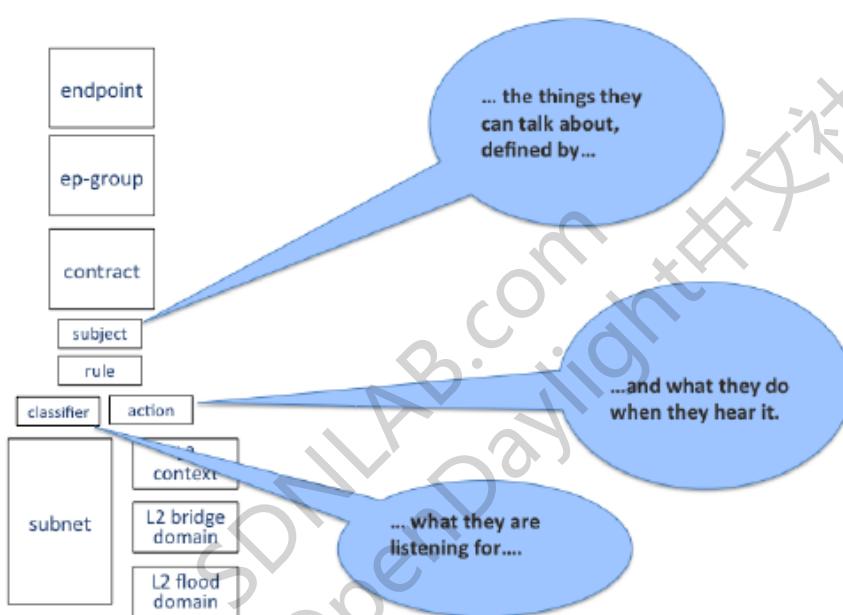
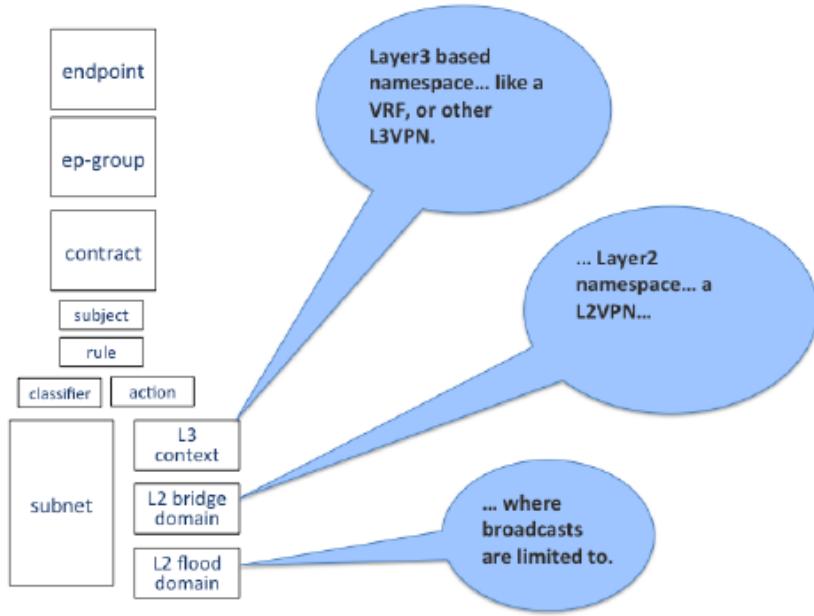


Figure 10.4. GBP Forwarding Model Terminology - L3 Context, L2 Bridge Context, L2 Flood Context/Domain, Subnet



终端：具有唯一标识的实体。锂版本中，可以是Docker容器或Neutron端口

终端组：终端组是共用同一策略的一组终端。终端组参与协议，决定通信类型。终端组消费和提供协议。并且提供需求和性能，决定怎样应用协议。一个终端组可以指定一个父类并继承父类。

协议：协议规定哪个终端采用哪种形式进行通信。终端组定义协议选择器选择不同终端组之间的协议。协议的性质是终端组进行选择的标签。一旦选用一个协议，协议就有匹配终端组的需求、性能以及任何可能情况的规范，为了激活主题可以允许进行某种通信。允许某个协议继承其父类协议。

主题：主题描述允许两个终端通信的几个方面。主题定义了一个有序的规则序列来匹配流量，并且对流量采用必要的动作。如果主题不允许通信则不能进行通信。

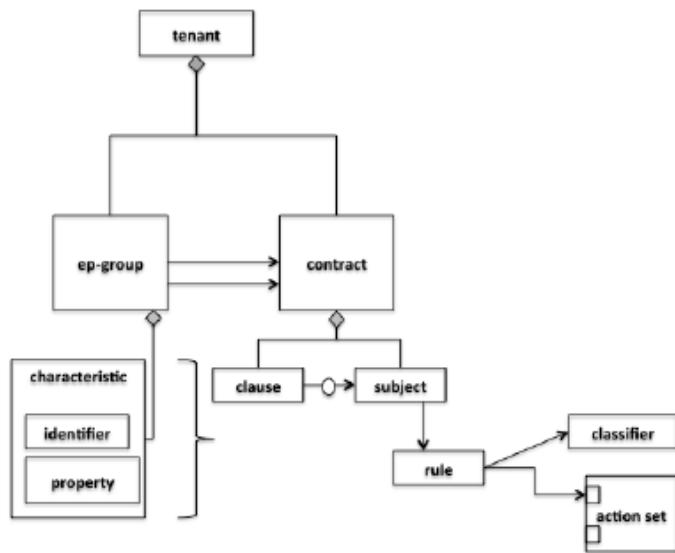
规范：规范是协议中定义的一部分。规范决定某个终端或终端组怎样运用协议。规范匹配终端组的需求和性能，以及终端可能出现的情况。匹配好的规范定义一组主题应用于通信的两个终端。

架构和价值主张

GBP提供一个基于接口的意图，可以通过UX、REST API访问，或者直接通过映射接口使用特定的域语言，例如Neutron。

GBP中有两个模型：访问模型和转发模型

Figure 10.5. GBP Access (or Core) Model



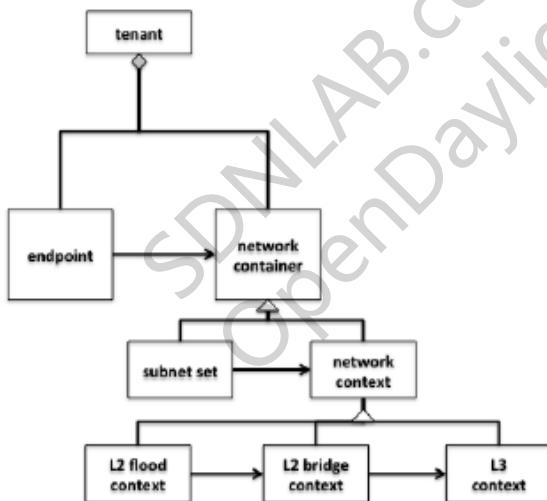
分类器和动作是模型的一部分，可以看成是钩，其定义是renderer提供的。在锂版本GBP中有一个renderer，OpenFlow Overlay renderer (OfOverlay)。

这些钩子具有类型定义功能，是renderer为主题提供的，称为subject-feature-definitions。

这就意味描述意图可以由多个renderer同时实现，GBP消费者无需进行任何资源调配。

自从SDN控制器OpenDaylight中实现了GBP，它就需要处理网络。利用可以运用于不同类型网络中的转发模型进行处理。

Figure 10.6. GBP Forwarding Model



每个终端都配有网络控制，可以是：

子网

一般的IP堆栈行为，ARP在子网中实现，将流量发送到默认网关。

一个子网是以下任何转发模型上下文的子网，但是典型的是泛洪域的子类。

L2泛洪域

允许泛洪行为。

是bridge-domain的n:1子类。

可以有多个子类。

L2桥域

是2层命名空间。

流量可以在2层发送的范围。

是L3 context的一个n: 1子类。

可以有多个子类。

L3 context:

是一个三层的命名空间。

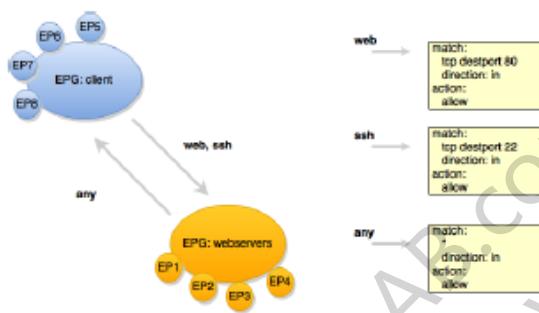
流量在3层发送的范围。

是tenant的一个n:1子类。

可以有多个子类。

访问、转发模型怎样工作，示例如下：

Figure 10.7. GBP Endpoints, EndpointGroups and Contracts



在该示例中，EPG:webservers提供web和ssh协议，而EPG:client使用这些协议。EPG:client提供的任何协议都由EPG:webservers使用。

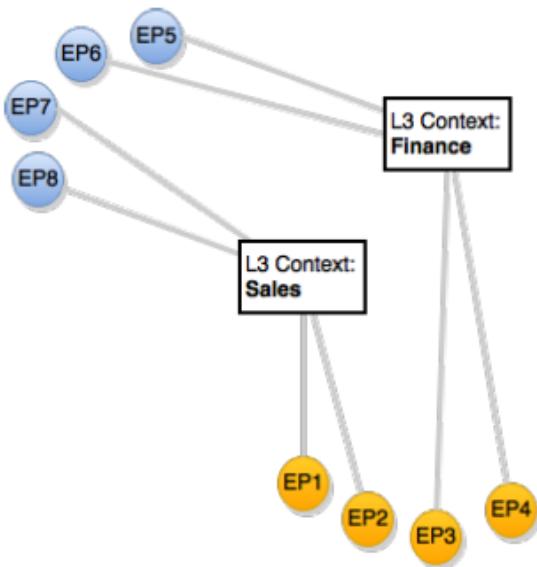
方向关键字始终从协议提供方的角度出发，在本例中，web协议由EPG:webservers提供，用分类器去匹配目标端口80，这意味着：

数据包中TCP端口为80

在EPG:webservers中发送给终端

会被允许

Figure 10.8. GBP Endpoints and the Forwarding Model

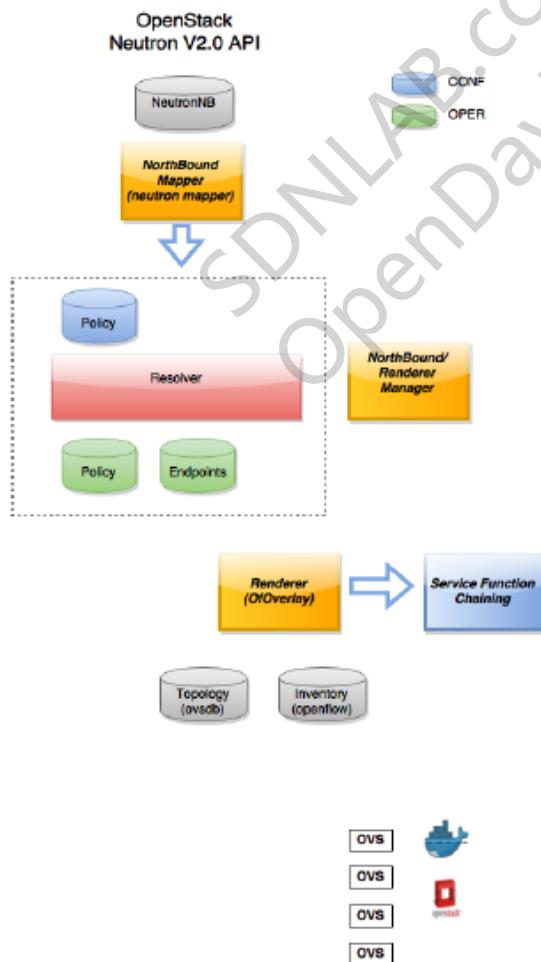


当转发模型如上图所示，你会发现即使所有的终端使用同一种协议，转发依旧会包含在转发模型的上下文或命名空间中。上图中，连接一个网络控制的网络终端有一个L3 Context的终极父类：Sales只能通过这个L3 Context与其他终端通信，采用这种方式实现L3VPN服务可以不对协议意图产生任何影响。

高级架构实现

整体架构包含Neutron domain specific mapping和OpenFlow Overlay renderer，如下所示：

Figure 10.9. GBP High Level Lithium Architecture

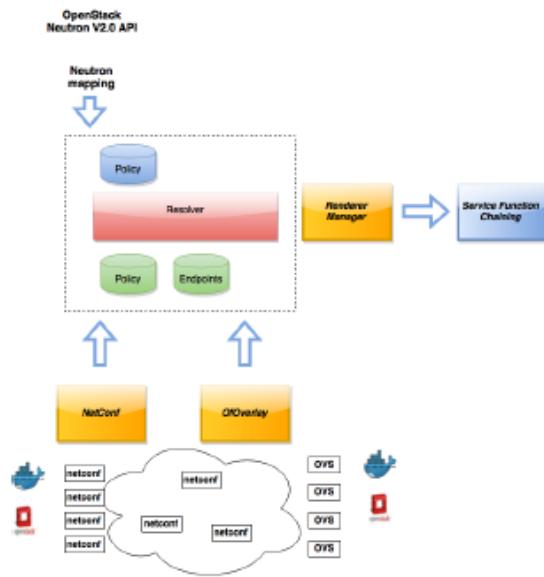


这个架构最大的优点就是特定域语言映射是独立的，独立于底层renderer的实现。

例如，使用Neutron Mapper将Neutron API映射到GBP核心模型，可以通过UX使用服务功能链增强该映射动态的生成某个协议，目前OpenStack还不提供该功能。

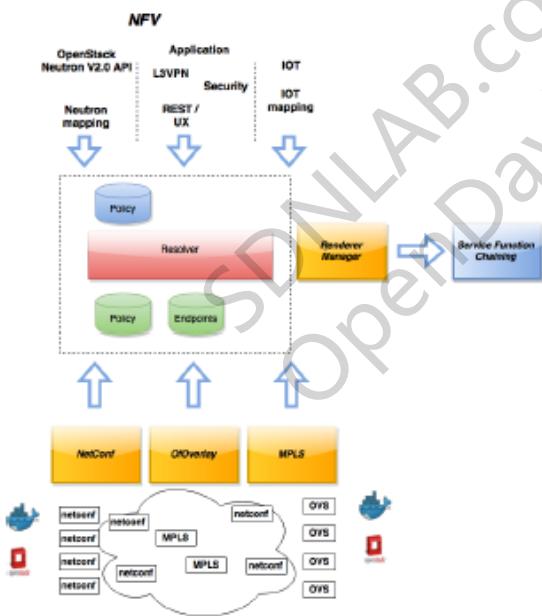
如果添加一个renderer，比如NetConf，那么NetConf设备就可以同时使用相同的策略。

Figure 10.10. GBP High Level Lithium Architecture - adding a renderer



当其他特定域映射也在进行时，如果使用的是同一个renderer，那么这个renderer只需要执行GBP访问和转发模型即可，而特定域映射也只需要管理到访问和转发模型的映射。例如：

Figure 10.11. GBP High Level Lithium Architecture - adding a renderer



总而言之，GBP架构：

隔离：所表达的意图与底层renderers完全隔离。

模块化集成：每个模块负责且只负责自己的部分。

可扩展：可以优化关于模型映射/执行和功能重利用的代码。

策略解决方案

Contract Selection

策略解决方案的第一步是选择适用范围内的contracts。

EndpointGroups以一个provider或者一个contract的消费者角色参与contracts。每个EndpointGroups能够在同一时间参与很多contracts。但是每个contract在一个时间内只能有一个起作用。此外，一个EndpointGroups选择一个contract有两种方法，无论是任何一个：

named selector: 通过contract ID简单的选择一个指定contract。

target selector: 通过对contract目标的质量匹配，允许更多的灵活性。

且contract selector一共有四种：

provider named selector: 通过contract ID选择一个contract并作为一个provider参与。

provider target selector: 针对一个质量匹配的contract目标进行匹配，并作为一个provider参与。

consumer named selector: 通过contract ID选择一个contract，并作为一个消费者参与。

consumer target selector: 针对一个质量匹配的contract目标进行匹配，并作为一个消费者参与。

源EndpointGroups选择一个contract作为provider或者消费者发现contract，目的EndpointGroups在相应的角色中匹配相同的contract，来决定适用范围内的contract。所以如果在EndpointGroups X中endpoint x正在与EndpointGroups Y中的endpoint y通信，且如果X选择C作为一个provider并且Y选择C作为一个消费者，亦或者X选择C作为一个消费者，Y选择C作为一个provider，contract C将是在适用范围内。

质量匹配怎样工作的详情在下一章节Matchers中描述。质量匹配为基于标签的contract选择提供一个灵活机制。

contract选择阶段的最后结果能够作为一组元组代表选择的contract适用范围被考虑到。元组域如以下：

Contract ID

provider EndpointGroup ID

用来选择contract的provider EndpointGroup，叫做matching provider selector，这个选择器的名称

消费者EndpointGroup ID。

用来选择contract的消费者EndpointGroup，叫做matching consumer selector，这个选择器的名称

Subject Selection

策略解决方案的第二个字段是决定哪个subject是在适用范围内。在EndpointGroups中subject定义允许endpoints之间进行哪种通信。subject selection程序应用于contract选择段中每个所选的contract适用范围。

通过将subject带入到适用范围内匹配所谓的标签、能力、需求和条件。当endpoints具备条件时，EndpointGroups有能力的需求。

Requirements和Capabilities（需求和能力）

当作为一个provider，EndpointGroups提供代表特定功能片的标签能力，能够提供给其他EndpointGroups来满足其他EndpointGroups的功能性需求。

当作为一个消费者，EndpointGroups提供需求，这是代表EndpointGroups需求一些特定功能片的标签。

例如一个案例，我们创建一个叫做“user-database”的能力，它表明一个EndpointGroups包含endpoints实现用户的数据库。也可以创建一个需求叫做“user-database”，表明EndpointGroups包含需要与提供这个服务的endpoints进行通信的endpoints。注意：在这个例子中，需求和能力是相同的名字，但是用户不需要遵循这个约

定。

匹配的provider选择器（用于provider EndpointGroups选择contract）被检查来决定provider EndpointGroups为contract适用范围提供的能力。

供应商选择器有一个能力列表直接包括在provider选择器中，也可以从一个父选择器或者一个父EndpointGroups中继承。

类似的，匹配的消费者选择器也将提供一系列的需求。

Conditions (条件)

Endpoints可以有conditions，代表一些与endpoint相关的可选状态。

关于condition的例子是“malware-detected”，或者“authentication-succeeded”。conditions用于影响特定endpoint怎样通信。为了继续这个例子，“malware-detected”condition可能导致一个endpoint的连接被中断，而“authentication-succeeded”是开放endpoint与服务的通信，要求一个endpoint是第一个被认证的，然后转发认证证书。

Clauses

Clauses呈现subjects的真实选择。一个clause在两类中有matchers列表。为了使clause是有效的，所有的matchers必须匹配。一个匹配clause将选择所有clause相关的subjects。注意：一个空的matchers列表计数为一个match。

第一类是消费者匹配器，匹配消费者EndpointGroup和endpoints。消费者匹配器是：

Group Identification Constraint: 需求匹配器，在匹配消费者选择器中匹配相关需求。

Group Identification Constraint: GroupName，匹配组名。

Consumer condition matchers: 在消费者EndpointGroup中匹配相关endpoints的conditions。

Consumer Endpoint Identification Constraint: 基于标签的endpoints匹配标准。在Lithium版本中，用于基于IpPrefix的标签endpoints。

第二类是provider匹配器，匹配provider的EndpointGroup和endpoints。provider匹配器是：

Group Identification Constraint: 功能匹配器，在匹配provider选择器中匹配相关功能。

Group Identification Constraint: GroupName，匹配组名。

Consumer condition matchers: 在provider EndpointGroup中匹配相关endpoints的conditions。

Consumer Endpoint Identification Constraint: 基于标签的endpoints匹配标准。在Lithium版本中，用于基于IpPrefix的标签endpoints。

clauses有subjects列表，应用clause匹配中所有的匹配器。subject选择段的输出逻辑上是适用范围内任何特定endpoints对的一系列subjects。

Rule Application

现在subjects已经被应用于一组特定endpoints间的流量，policy能被应用到允许endpoints间的通信。先前步骤中可用subjects将各自包含一组rules。

Rules由一系列classifiers和actions组成。classifiers匹配两个endpoints间的流量，如：classifier将匹配80端口所有

的TCP流量，或者匹配包含特定cookie的HTTP流量。Actions是到达目的地之前，需要采取流量的特定行为。Actions包括以一些方式标记和封装流量，重定向流量，或者应用服务功能链（见在BGP中使用服务功能链章节）。

Rules、subjects、actions有一个顺序参数，一个较低的顺序值意思是一个特定item首先被应用。一个特定subject的所有rules将在其他subject的rules之前被应用，且来自特定rule的所有actions将在其他rule的actions之前被应用。如果有多个item有相同的顺序参数，将打破名称的字母顺序，逻辑上更早的名称有较低的顺序。

Matchers

matchers指定一系列标签（包括requirements、capabilities、conditions和qualities）来匹配。有几种操作类似的matchers：

Quality matchers: 用于contract选择段期间的target selectors。Quality matchers通过比较一个命名的选择器提供一个更先进、更灵活的方法来选择contracts。

Requirement and capability matchers: 用于subject选择段期间的clauses，匹配EndpointGroups相关的requirements和capabilities。

Condition matchers: 用于subject选择段期间的clauses，匹配endpoints相关的conditions。

在其核心，一个匹配器是相当简单的。它包含一列标签名称，连同一个match type。match type可以是：

all: 匹配器匹配时，匹配所有的标签。

any: 匹配器匹配时，匹配任何一个标签。

none: 匹配器匹配时，不匹配任何标签。

注意：根据匹配类型“all”，所有matchers通过匹配一系列空的标签做一个match。另外，匹配的每个标签可选的包括一个相关的名称域。quality matchers是一个目标名称，capability和requirement matchers是一个选择器名称。如果名称域被指定，匹配器随着名称只匹配目标或者选择器，而不是匹配任一目标或者选择器。

Inheritance

系统中一些对象包括相关的父进程，他们将从父进程中继承，父进程参考图表一定是循环免费的。当解析名称时，解决方案系统必须侦测循环和引起异常。循环部分的对象可能被视为他们根本不会去定义。通常的，继承以简单的将父对象导入到子对象中工作。当在子对象中有相同名称的对象时，子对象将根据特定的对象类型的规则覆盖父对象。我们将为每个对象类型的继承探索详细的规则。

EndpointGroups

EndpointGroups将从父EndpointGroups中继承所有的选择器。相同名称的选择器作为父EndpointGroups的选择器，将继承如下定义的行为。

Selectors

Selectors（选择器）包括provider命名选择器、provider target选择区，消费者命名选择器和消费者target选择器。选择器自己不能有父选择器，但是当选择器与在父EndpointGroup中相同类型的选择器名称相同时，他们将从父EndpointGroup中继承并覆盖选择器的行为。

Named Selectors（命名选择器）

Named Selectors将添加一系列被父命名选择器选择的contract IDs。

Target Selectors

在子EndpointGroup中的Target Selectors与在父EndpointGroup的target selector相同名称时，将从父

EndpointGroup继承quality matchers。如果在子EndpointGroup中有一个quality matcher与父EndpointGroup中的名称相同时，将覆盖以下描述的Matchers。

Contracts

Contracts将从父contracts继承所有的targets、clauses和subjects。当这些对象中有任何一个与父contract里有相同名称时，描述将如以下被定义。

Targets

Targets自己不能有父target，但是可以从父contract中相同名称的target里继承。target里的Qualities将从父target里继承，如果一个quality在子target中被定义，他将没有任何语义上的影响，除非quality的inclusion-rule参数设置为“exclude”。这种情况下，匹配target的目的标签将被忽视。

Subjects

Subjects不能有自己的父Subject，但是可以从父contract相同名称的subject中继承。子subject中的顺序参数，如果存在，将覆盖父subject中的顺序参数。但是不管怎样，rules将不覆盖相同名称的rules。代替的，父subject中的所有rules将被视为比子subject中所有rules更高的顺序，所以子subject中的所有rules将在父subject中的rules之前运行。如果没有合并排序的潜在问题，将对覆盖父subject中的任何rules有影响。

Clauses

Clauses不能有自己的父clause，但是可以从父contract相同名称的clause中继承。在父clause的subject reference列表将被添加到子clause的subject reference列表中。这仅仅是一个集合的操作。一个subject reference指的是父contract中的subject名称，可以有子contract被覆盖的名称。clause中每个matcher都被子clause继承。子clause中和父clause中的名称和类型相同，将继承和覆盖父matcher。以下Matchers查看更多信息。

Matchers

Matchers包括quality matchers、condition matchers、requirement matchers和capability matchers。Matchers不能有自己的父matcher，但是当与父对象有一个相同名称和类型的matcher时，子对象中的matcher将继承和覆盖父对象中的matcher行为。匹配类型如果在子对象中被指定，则覆盖父对象中指定的值。标签也将继承父对象中的标签。如果与在子对象中有相同名称的标签，没有任何语义上的影响，除非标签的inclusion-rule参数被设置为“exclude”。这种情况下，以匹配为目的的标签将被忽视。否则，相同名称的标签将完全覆盖父对象中的标签。

使用GBP UX接口

概述

下面的组件构成了这个应用，下面会进行更详细的描述：

Basic视图

Governance视图

Policy Expression视图

Wizard视图

访问GBP UX:

<http://< odl controller >:8181/index.html>

Basic视图

Basic视图包含5个导航按钮：

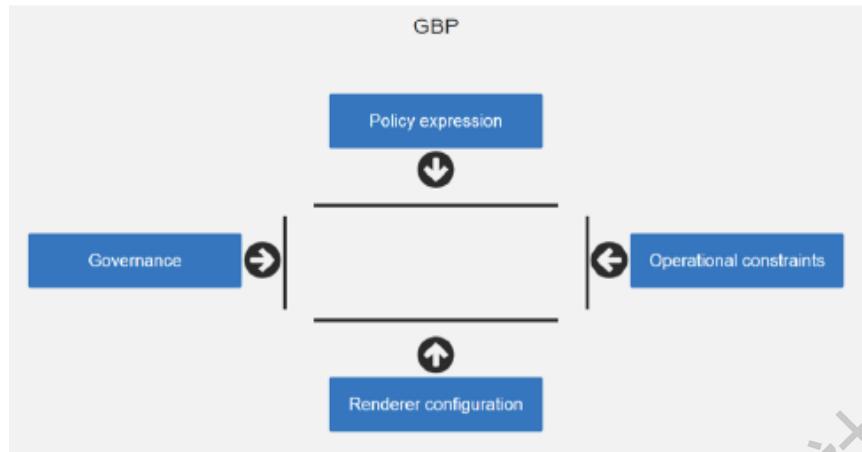
Governance-切换到Governance视图（中间图有相同的功能）

Renderer configuration-展开Renderers区域，切换到Policy expression视图

Policy expression-展开Policy区域，切换到Policy expression区域

Operational constraints-下一版本的预留位

图10.12 Basic view



Governance视图

Governance视图包含三列。

图10.13 Governance视图



Governance视图-Basic视图-最左边一列

最左边是Health区域，区域内Exception和Conflict按钮的功能还没实现。这是为了之后的版本开发预留的位置。

Governance视图-Basic视图-中间一列

这个区域的上半部分是一个可选框，列出了可选租户。选择好租户以后，该应用的子区域就会根据所选的租户操作和显示数据。

在这个可选框下面有两个按钮，用来展示Governance区域的Expressed policy或者Delivered policy。这个区域的下半部分也是一个可选框，列出了可选renderers。目前只有OfOverlay renderer可用。

可选框下面是Renderer configuration按钮，切换app到Renderers区域的Policy expression视图，可以执行CRUD

操作。Renderer state按钮展示Renderer state视图。

Governance视图-Basic视图-右边一列

Governance视图最右边区域的底部是Home按钮，key切换app到Basic视图。

上面部分是导航菜单，有4个主要的区域。

Policy expression按钮可以展开/关闭子菜单——Policy expression的三个主要部分。用户点击子菜单按钮，可以切换到appropriate section区域的Policy expressions视图，可以执行CRUD操作。

Renderer configuration按钮切换到Policy expressions视图。

Governance按钮展开/关闭子菜单——Governance区域的四个主要部分。Governance区域的子菜单按钮列出Governance视图的相应部分。

Operational constraints的功能还没实现，是为之后的版本开发预留的位置。

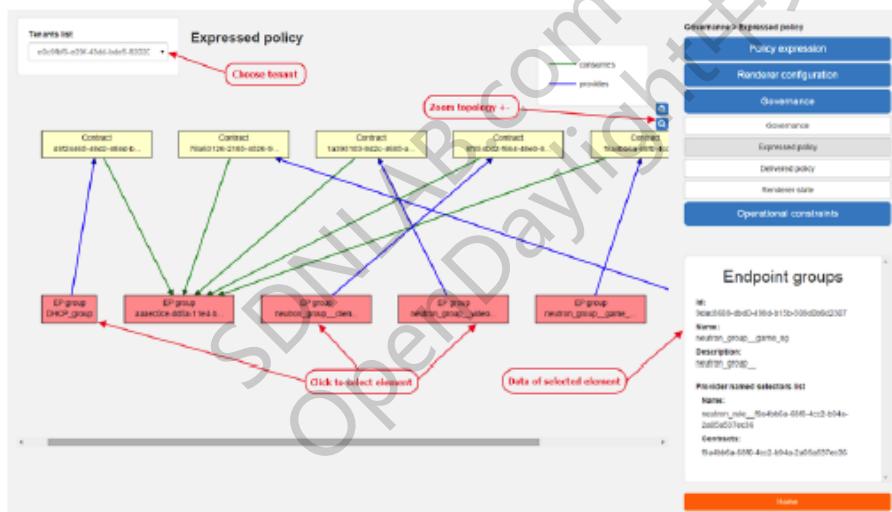
菜单下方是放视图信息的，展示拓扑中实际选择的元素信息（下面会给出解释）。

Governance视图-Expressed policy

这个视图陈列了所选租户提供和消耗EndpointGroups的contracts，在左上角的可选框里面可以进行选择。

单击任何contract或者EPG，被选元素的相关数据将会展示在右侧菜单下面。点击Manage按钮会显示wizard窗口，管理项目的配置，如之后会介绍的Service Function Chaining。

图10.14 Expressed policy



Governance视图-Delivered policy

这个视图陈列了所选租户提供和消耗EndpointGroups的subjects，在左上角的可选框里面可以进行选择。

单击任何subject或者EPG，被选元素的数据将会展示在右侧菜单的下方。

双击subject，subject的详细视图将会陈列所选subject的规则，同样在左上角的可选框内可以改变选项。

单击rule或subject，所选元素的数据会在右侧菜单下方显示。

在Delivered policy视图中双击EPG，将会展示出EPG的详情——EPG终端，这也可以在左上角的可选框内修改。

单击EPG或endpoint，所选元素的数据将会显示在右侧菜单的下方。

图10.15 Delivered policy

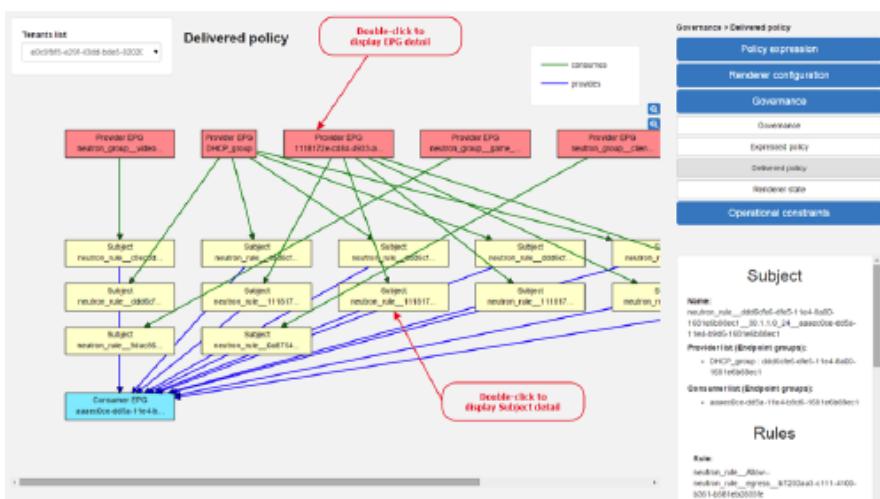


图10.16 Subject detail

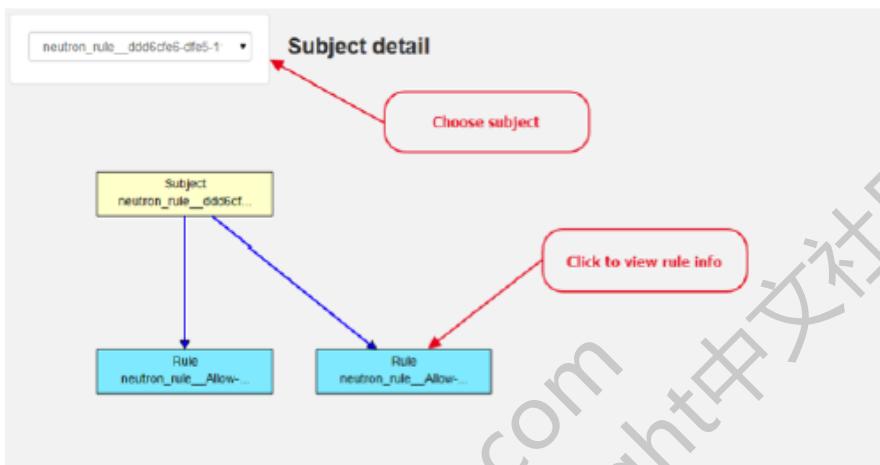
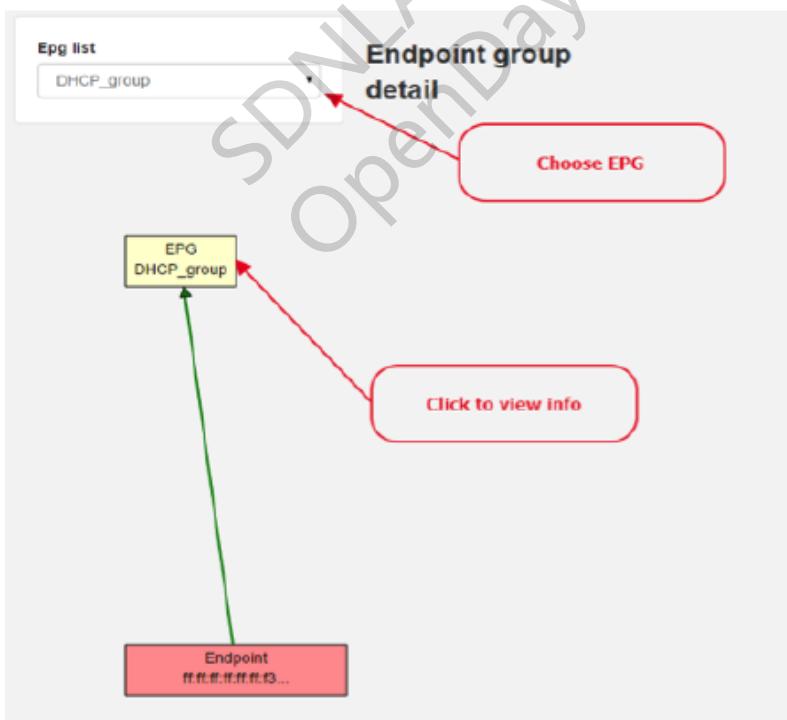


图10.17 EPG detail



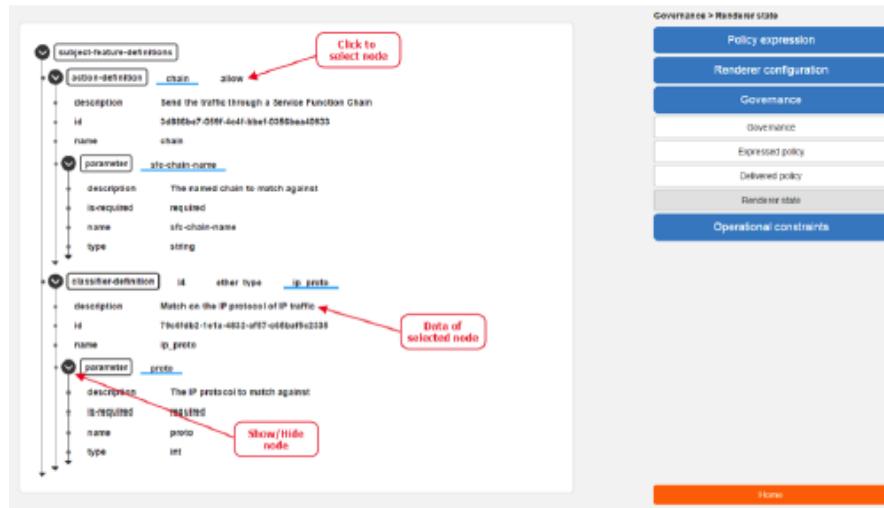
Governance视图-Render state

这部分列出Subject功能定义数据，有两个部分：Action definition和Classifier definition。

单击圆圈中的down/right箭头，可以展开/隐藏相应容器和列表的数据。列表节点的旁边会显示经常被选择的列表元素，并且显示元素的数据（名字下方用蓝线划出）。

点击子节点的名字，可以展示出该节点的数据。

图10.18 Renderer state



Policy expression视图

该视图的左边部分列出了所选元素的拓扑，在底部有一些按钮可以用来转换拓扑的类型。

该视图的右边那列包括4个部分。顶端列出了应用中breadcrumb的实际位置。

在breadcrumb下面是可选框，列出可选的租户。中间部分放的是导航菜单，能够切换到你想要操作的区域，执行CRUD操作。

底部是Access Model Wizard按钮，这是一个快速导航菜单，展示Wizard视图。Home按钮切换app到Basic视图和Back按钮——切换到上一个区域。

Policy expression-Navigation菜单

打开Policy expression，从GBP主屏选择Policy expression。

在navigation框上面，你可以从租户列表选择租户来激活与租户相关的功能。

在右边的菜单中，默认Policy菜单区域是展开的。这个区域的子项目是租户、EndpointGroups、contracts和L2/L3 objects模块化的CRUD（创建、读取、更新和删除）。

Renderers区域包括Classifiers和Actions的CRUD窗口。

Endpoints区域包括Endpoint和L3 prefix endpoint的CRUD窗口。

图10.19 Navigation菜单



Policy expression-拓扑类型

有三种拓扑类型：

Configured拓扑-CONFIG数据存储中的EndpointGroups以及EndpointGroups之间contracts。

Operational拓扑-列出相同的信息，但是基于operational数据。

L2/L3-列出L3Contexts、L2 Bridge domains、L2 Flood domains和Subnets中的关系。

图10.20 CRUD操作

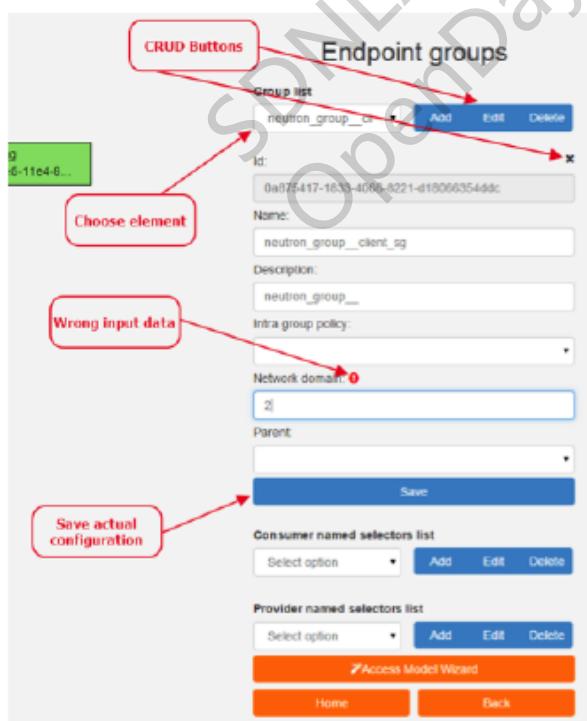


图10.21 L2/L3拓扑

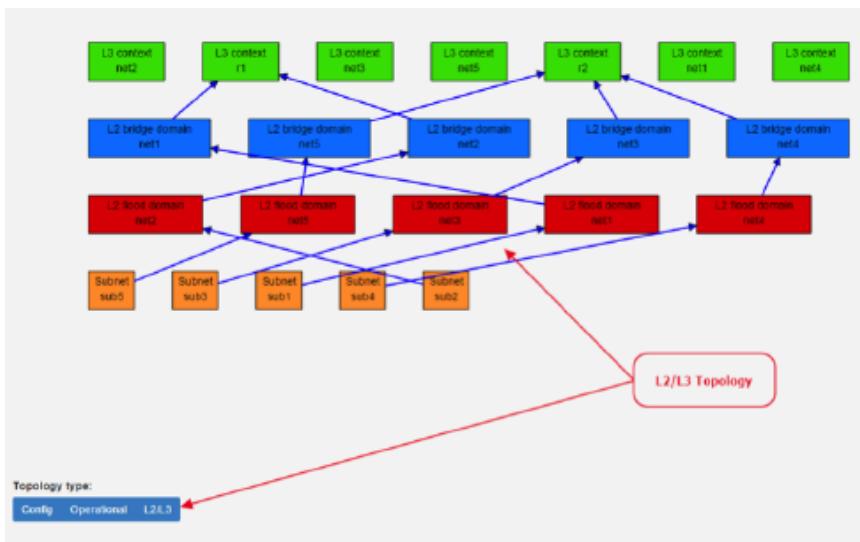
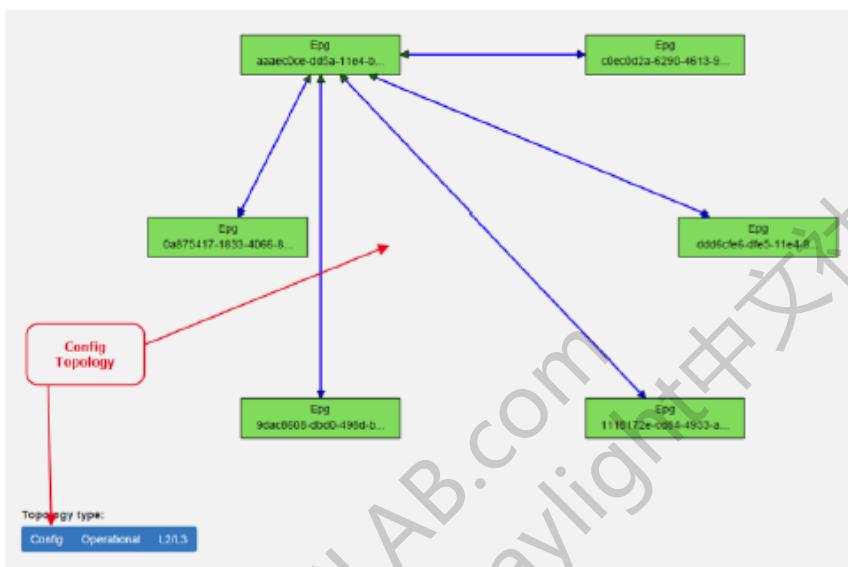


图10.22 Config拓扑



Policy expression-CRUD操作

该部分描述了观察、添加、编辑和删除系统元素（如租户、EndpointGroups等）的基流。

租户

点击右侧菜单的Tenants按钮编辑租户信息。CRUD表格中包含了租户列表和控制按钮。

点击Add按钮可以增加一个新的租户，在表格中将显示增加一个新的租户。在填写完租户的属性名称和描述后点击Save按钮。只有所有相应的属性填写正确保存才有效。当属性填写错误时，将鼠标移至概叹号上会有下一步的提示。在设置保存后表格会

查看现有的租户，从选择框选择租户租户名单。视图的形式是只读的，可以点击右上角的十字标记进行关闭。

编辑选定的租户，单击edit按钮，将显示所选的编辑表单租户。编辑所选租户的名称和描述后点击保存按钮保存选中的租户。在保存租户的编辑表单将被关闭，租户名单将被设置为默认值。

删除租户从租户选择租户列表并点击删除按钮。

回到策略表达式，点击窗口底部的后退按钮。

EndpointGroups

管理EndpointGroups(EPG)租户必须从租户列表顶部选择。

添加新的EPG,单击add按钮,必需属性填后点击保存按钮。在添加EPG之后您可以编辑它并分配给Consumer命名选择器或Provider命名选择器给它。

从Group列表中选择EPG后单击edit按钮编辑EPG。

单击Consumer named selector (CNS) 列表旁边的add按钮添加新的CNS。在CNS编辑时,你可以对目前的CNS按加号按钮并且从contracts列表选择contracts的方式设置一个或多个contacts。可以点击contracts旁边的十字标记删除contracts。可以查看添加的CNS,并且可以通过点击Edit和Delete按钮进行编辑和删除。

单击Provider named selector (PNS) 列表旁边的add按钮添加新的PNS。在PNS编辑时,你可以对目前的PNS按加号按钮并且从contracts列表选择contracts的方式设置一个或多个contacts。可以点击contracts旁边的十字标记删除contracts。可以查看添加的PNS,并且可以通过点击Edit和Delete按钮进行编辑和删除。

可以通过选择selectbox旁边的删除按钮对selectbox里的EPG, CNS或PNS进行选择删除。

Contracts

从租户列表中选择租户用于管理contracts。

点击Add按钮并在填完必要信息后点击Save按钮创建一个新的Contract。

添加Contract之后用户可以在Contract列表中选择并单击编辑按钮编辑它。

在编辑Construct时,可以在Clause列表旁边单击Add按钮以添加新的Clause。当对在Clause列表中选择的Clause进行编辑时,用户可以通过点击Clause subjects标签旁边的加号按钮来指定Clause subjects。增加并编辑action必须点击Save按钮以提交。可以通过对Clause列表进行增删改查来管理Subjects。

L2/L3

在管理L2/L3时,必须在Tenants列表中选择tenant。

点击L3 Context列表旁边的Add按钮添加L3 Context,将会显示窗口表示添加了一个新的L3 Context。填完信息后点击Save按钮,

可以从L3 Context列表中选择一个已有的L3 Context,弹出窗口只能显示Context的一些数据,并不能进行修改,可以点击右上角的叉号关闭窗口。

用户点击Edit按钮对已选择的L3 Context进行编辑。点击保存,保存后窗口会关闭并且L3 Context列表会被设置为默认值。

从L3 Context列表中选择一个已有的L3 Context,点击Delete按钮可以删除它。

点击L2 Bridge Domain列表旁边的Add按钮可以添加L2 Bridge Domain,点击后会有窗口显示添加了一个新的L2 Bridge Domain。填完信息后点击Save按钮,保存后窗口会关闭并且L2 Bridge Domain列表会被设置为默认值。

可以从L2 Bridge Domain列表中选择一个已有的L2 Bridge Domain,弹出窗口只能显示L2 Bridge Domain的一些数据,并不能进行修改,可以点击右上角的叉号关闭窗口。

用户点击Edit按钮对已选择的L2 Bridge Domain进行编辑。点击保存,保存后窗口会关闭并且L2 Bridge Domain列表会被设置为默认值。

从L2 Bridge Domain列表中选择一个已有的L2 Bridge Domain,点击Delete按钮可以删除它。

点击L3 Flood Domain列表旁边的Add按钮可以添加L3 Flood Domain,点击后会有窗口显示添加了一个新的L3 Flood Domain。填完信息后点击Save按钮,保存后窗口会关闭并且L3 Flood Domain会被设置为默认值。

可以从L3 Flood Domain列表中选择一个已有的L3 Flood Domain,弹出窗口只能显示L3 Flood Domain的一些数据,并不能进行修改,可以点击右上角的叉号关闭窗口。

用户点击Edit按钮对已选择的L3 Flood Domain进行编辑。点击保存,保存后窗口会关闭并且L3 Flood Domain列

表会被设置为默认值。

从L3 Flood Domain列表中选择一个已有的L3 Flood Domain，点击Delete按钮可以删除它。

点击Subnet列表旁边的Add按钮可以添加Subnet，点击后会有窗口显示添加了一个新的Subnet。填完信息后点击Save按钮，保存后窗口会关闭并且Subnet列表会被设置为默认值。

可以从Subnet列表中选择一个已有的Subnet,弹出窗口只能显示Subnet的一些数据，并不能进行修改，可以点击右上角的叉号关闭窗口。

用户点击Edit按钮对已选择的Subnet进行编辑。点击保存，保存后窗口会关闭并且Subnet列表会被设置为默认值。

从Subnet列表中选择一个已有的Subnet，点击Delete按钮可以删除它。

Classifiers

点击Classifier列表旁边的Add按钮可以添加Classifier，点击后会有窗口显示添加了一个新的Classifier。填完信息后点击Save按钮，保存后窗口会关闭并且Classifier列表会被设置为默认值。

可以从Classifier列表中选择一个已有的Classifier,弹出窗口只能显示Classifier的一些数据，并不能进行修改，可以点击右上角的叉号关闭窗口。

用户点击Edit按钮对已选择的Classifier进行编辑。点击保存，保存后窗口会关闭并且Classifier列表会被设置为默认值。

从Classifier列表中选择一个已有的Classifier，点击Delete按钮可以删除它。

Actions

点击Action列表旁边的Add按钮可以添加Action，点击后会有窗口显示添加了一个新的Action。填完信息后点击Save按钮，保存后窗口会关闭并且Action列表会被设置为默认值。

可以从Action列表中选择一个已有的Action,弹出窗口只能显示Action的一些数据，并不能进行修改，可以点击右上角的叉号关闭窗口。

用户点击Edit按钮对已选择的Action进行编辑。点击保存，保存后窗口会关闭并且Action列表会被设置为默认值。

从Action列表中选择一个已有的Action，点击Delete按钮可以删除它。

Endpoint

点击Endpoint列表旁边的Add按钮可以添加Endpoint，点击后会有窗口显示添加了一个新的Endpoint。填完信息后点击Save按钮，保存后窗口会关闭并且Endpoint列表会被设置为默认值。

可以从Endpoint列表中选择一个已有的Endpoint,弹出窗口只能显示Endpoint的一些数据，并不能进行修改，可以点击右上角的叉号关闭窗口。

用户点击Edit按钮对已选择的Endpoint进行编辑。点击保存，保存后窗口会关闭并且Endpoint列表会被设置为默认值。

从Endpoint列表中选择一个已有的Endpoint，点击Delete按钮可以删除它。

L3 prefix endpoint

添加L3 prefix endpoint,单击add按钮旁边的L3 prefix endpoint列表。这将显示的形式添加一个新的端点。添加EndpointGroup任务,点击EndpointGroups旁边的加号按钮标签。点击Condition标签旁边的加号按钮添加Condition。点击L2网关标签旁边的加号按钮添加L2网关。单击L3网关标签旁边的加号按钮添加L3网关。填完信息后点击Save按钮，保存后窗口会关闭并且Endpoint列表会被设置为默认值。

可以从L3 prefix endpoint列表中选择一个已有的L3 prefix endpoint,弹出窗口只能显示L3 prefix endpoint的一些数

据，并不能进行修改，可以点击右上角的叉号关闭窗口。

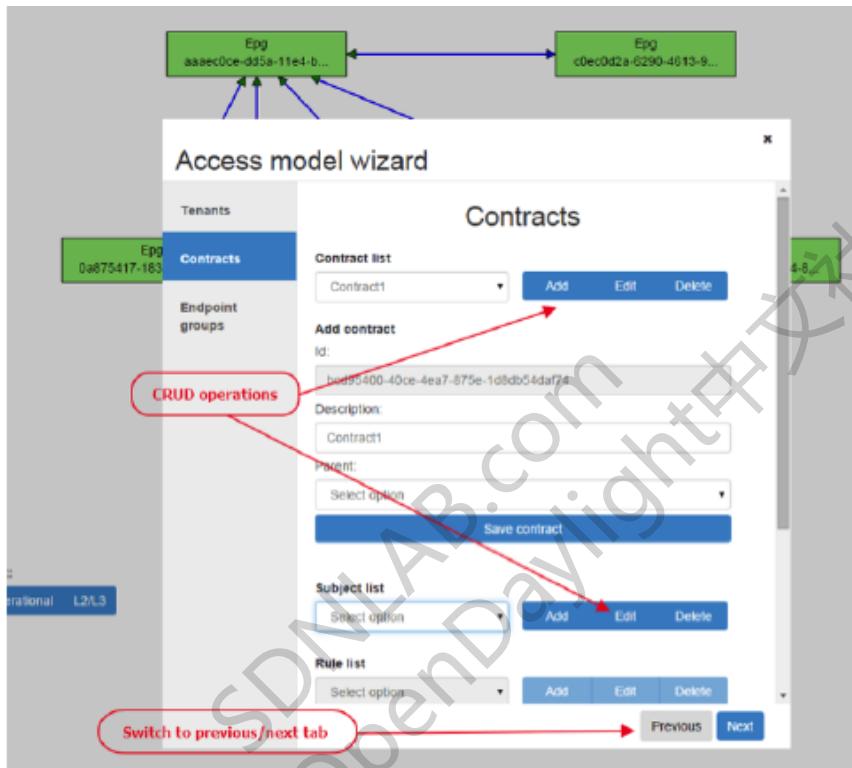
用户点击Edit按钮对已选择的L3 prefix endpoint进行编辑。点击保存，保存后窗口会关闭并且L3 prefix endpoint列表会被设置为默认值。

从L3 prefix endpoint列表中选择一个已有的L3 prefix endpoint，点击Delete按钮可以删除它。

Wizard

Wizard提供了快速方法将所需基本数据发送给控制器，包含所需的GBP应用程序的基本用法。这对于没有任何数据的控制器显得很有用。在第一个选项卡的界面用于创建租户。第二个选项卡用于对contracts同它们的子元素进行CRUD操作，比如subjects、rules,action和classifier。最后一个选项卡用于对EndpointGroups及其CNS和PNS进行CRUD操作。所创建数据的结构可以通过点击提交按钮进行发送。

图10.23 Wizard



使用GBP API

可以查看以下章节：

[使用GBP OpenFlow Overlay \(OfOverlay\) renderer](#)

[Policy Resolution](#)

[Forwarding Model](#)

[GBP demo和开发环境技巧](#)

也可以建议使用：

[Neutron mapper](#)

[UX](#)

如果REST API必须被使用，来探索各种GBP REST选项，上面的资源是不够的：

安装: feature:install odl-mdsal-apidocs or odl-dlux-yangui

浏览: <http://<odl-controller>:8181/apidoc/explorer/index.html>

使用OpenStack结合GBP

概述

这主要是为应用开发者和网络管理者集成GBP（Group Based Policy）到OpenStack中使用。

1.启用GBP Neutron Mapper功能，在Karaf控制台安装:

```
feature:install odl-groupbasedpolicy-neutronmapper
```

2.Neutron Mapper自动加载以下依赖: odl-neutron-service;

3.Neutron Northbound实现REST API用于OpenStack: odl-groupbasedpolicy-base;

4.基于GBP功能设置，如 policy resolution, 数据模型等: odl-groupbasedpolicy-ofovrlay;

在Lithium版本中，GBP有一个渲染器，因此默认被加载。OpenStack Neutron中的REST调用是通过Neutron NorthBound项目，GBP提供 Neutron V2.0 API的实现。

功能组件

已支持的Neutron实体列表:

端口 (Port)

网络 (Network) : 标准内部以及外部提供商L2/3层网络

子网 (Subnet)

安全组 (Security-groups)

路由器 (Routers) :

1.每个计算本地路由的分布式功能;

2.每个计算节点外部网关接入 (需要专用端口)

3.每个租户多台路由器

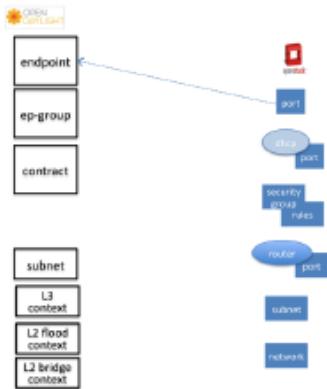
FloatingIP NAT

IPv4/IPv6支持

Neutron实体与GBP实体的映射如下显示:

Neutron Port

图10.24 Neutron Port

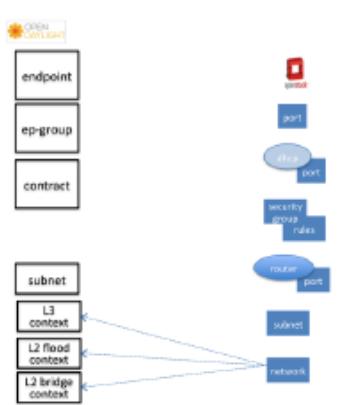


Neutron port被映射到endpoint。当前的实现支持每个Neutron端口一个IP地址。如果Neutron端口是在多个Neutron安全组中，endpoint和L3-endpoint属于多个EndpointGroups中。

endpoint的关键是获得作为代表Neutron网络L2-flood-domain根源的L2-bridge-domain。MAC地址来自Neutron端口。一个L3-endpoint被基于Neutron端口的L3-context (the parent of the L2-bridge-domain)和IP地址创建。

Neutron Network

图10.25 Neutron Network



Neutron network有以下特点：

定义一个广播域

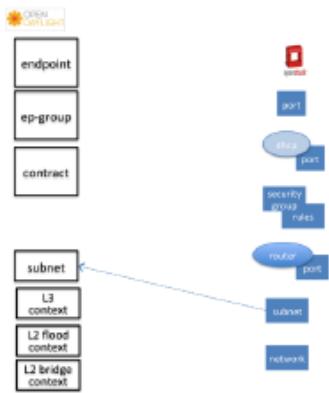
定义一个L2层传输域

定义一个L2层命名空间

为了显示这个，Neutron Network映射多个GBP实体，第一个匹配是一个L2层flood-domain来反应Neutron网络是一个泛洪或者广播域名。一个L2-bridge-domain（二层桥接域）与L2层flood-domain（泛洪域）的根节点相关。这个反应L2层传输域和L2层地址命名空间。第三个映射是到L3-context，代表不同的L3地址空间。L3-context是L2-bridge-domain的根节点。

Neutron Subnet

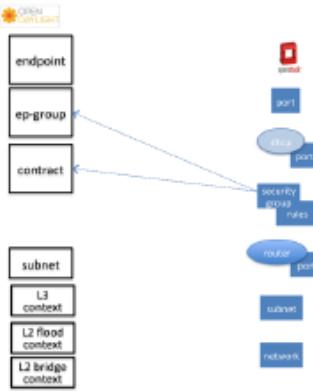
图10.26 Neutron Subnet



Neutron subnet与Neutron Network相关。Neutron subnet被映射到GBP子网， GBP子网的根节点是代表Neutron 网络的L2-flood-domain。

Neutron Security Group

图 10.27 Neutron Security Group



代表Neutron security-group的GBP实体是EndpointGroup。

Infrastructure EndpointGroups

Neutron-mapper自动创建EndpointGroup来管理关键基础设施项目，如：

DHCP EndpointGroup: 包含代表Neutron DHCP端口的endpoints。

Router EndpointGroup: 包含代表Neutron路由端口的endpoints。

External EndpointGroup: 保留代表Neutron路由网关端口、也和FloatingIP端口相关的L3-endpoints。

Neutron Security Group Rules

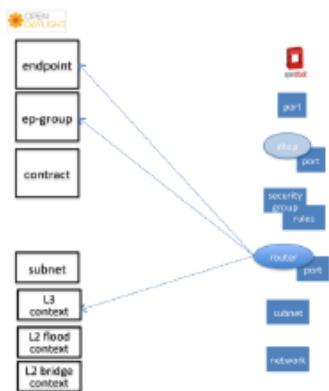
所有映射中，这个是最相关的，因为Neutronecurity-group-rules被映射到带有clauses、subjects、rules、action-refs、classifier-refs等的contracts。contracts用于代表Neutron Security Groups的EndpointGroups之间。为了使简化，需要重点注意的是Neutron security-group-rules和一个GBP rule包含的是相似的，如下显示：

classifier with direction

“allow”操作

Neutron Routers

图10.28 Neutron Routers



Neutron Routers代表一个L3-context。这将一个路由器视为一个L3层的命名空间，因此，每个网络附加它作为L3层命名空间的一部分。

这允许每个租户的多个路由器完全隔离。到EndpointGroup的映射代表GBP Neutron Mapper创建的内部基础设施EndpointGroups。

当一个Neutron路由器接口被添加到一个网络/子网中时，网络/子网和相关的endpoints或Neutron端口无缝的被添加到命名空间中。

Neutron FloatingIP

这与Neutron Port相关，利用了ofOverlay渲染器的NAT功能。

每个Nova计算节点主机上的一个专用的外部接口允许分布式外部访问。如果没有通过Neutron控制器必须进行路由，或者必须启用Neutron分布式路由功能的任何格式，与FloatingIP相关的每个Nova实例能够访问外部网络。

假设Neutron Subnet命令中外部网络预分配的网关是可达的，GBP Neutron Mapper和OfOverlay renderer的结合将自动为默认网关进行ARP，要求无用户干预。

Troubleshooting within GBP

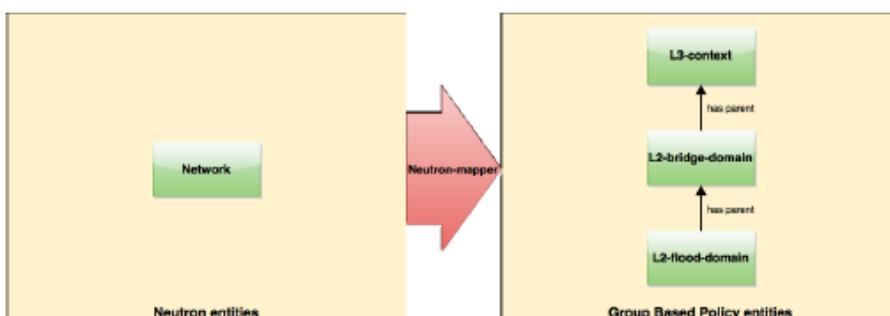
映射功能的日志等级可以在包org.opendaylight.groupbasedpolicy.neutron.mapper中设置。在karaf中启用TRACE日志等级的例子如：

```
log:set TRACE org.opendaylight.groupbasedpolicy.neutron.mapper
```

Neutron mapping example

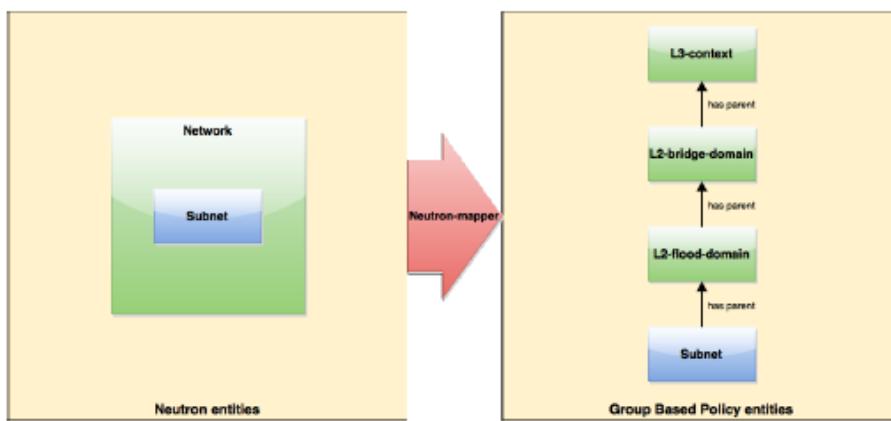
映射的例子可以用来创建Neutron网络、subnet和端口。当一恶搞Neutron玩过被创建，3个GBP实体被创建：l2-flood-domain、l2-bridge-domain、l3-context。

图10.29 Neutron network mapping



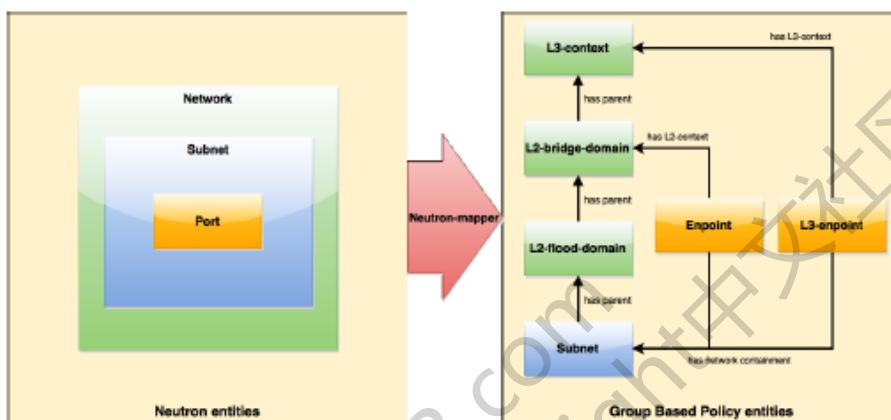
在网络映射中一个subnet被创建后，如下显示：

图10.30 Neutron subnet mapping



如果一个Neutron端口在subnet中被创建，那么一个endpoint和l3-endpoint也被创建。这个endpoint有来自Neutron端口中l2-bridge-domain和MAC address组成的key。一个l3-endpoint的key是由l3-context和IP address组成。包含endpoint和l3-endpoint的网络指向subnet。

图10.31 Neutron port mapping



配置GBP Neutron

图10.31. Neutron port mapping.png

用户需求在通过初始OpenStack建立设置时没有干预。

更多的配置信息可以在GBP wiki ([https://wiki.opendaylight.org/view/Group_Based_Policy_\(GBP\)](https://wiki.opendaylight.org/view/Group_Based_Policy_(GBP))) 上的 DevStack demo environment查看。

实施或管理GBP Neutron

为一致性需求，所有的预留应该通过Neutron API以CLI命令行或者是界面的方式来呈现。

映射策略能够通过GBP UX的方式来扩展：

启用服务功能链（SFC）

从Neutron的外部，添加endpoints，也就是在OpenStack中没有预分配的VMs或者容器

扩展Security Group Rules中的policies/contracts

覆盖额外的contracts或groupings

教程

DevStack demo environment在GBP wiki上，可根据wiki指南进行操作。

[https://wiki.opendaylight.org/view/Group_Based_Policy_\(GBP\)](https://wiki.opendaylight.org/view/Group_Based_Policy_(GBP))

使用GBP OpenFlow Overlay (OfOverlay) renderer

概述

OpenFlow Overlay功能组件实现OpenFlow Overlay renderer，在OpenvSwitch软件交换机节点之间创建一个网络虚拟化解决方案。

安装与依赖

在OpenDaylight的karaf控制台：

```
feature:install odl-groupbasedpolicy-ofoverlay
```

本renderer适合与OpenVSwitch (OVS) 2.1+ (尽管 2.3 是推荐版本)、OpenFlow 1.3共同使用。

当与Neutron Mapper功能组件共同使用时不需要额外的OfOverlay特定设置。

当该功能组件加载“standalone”，用户就应该开始配置架构，如下：

实例化OVS网桥

将主机连接到该网桥

在网桥中创建VXLAN/VXLAN-GPE隧道端口

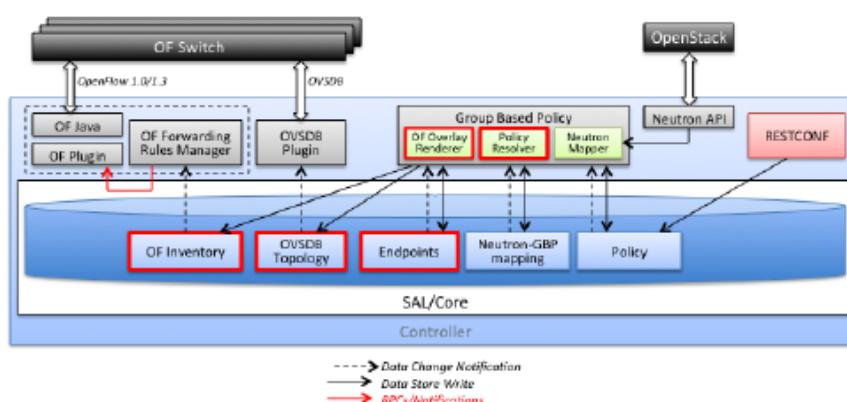
在锂版本中，GBP OfOverlay renderer还支持表偏移选项，流程从table 0开始偏移。

当然，也可以进行设置：修改./distribution-karaf/target/assembly/etc/opendaylight/karaf/15-groupbasedpolicyofoverlay.xml文件中的“< gbp-ofoverlay-table-offset >0< /gbp-ofoverlay-table-offset >”。

OpenFlow Overlay架构

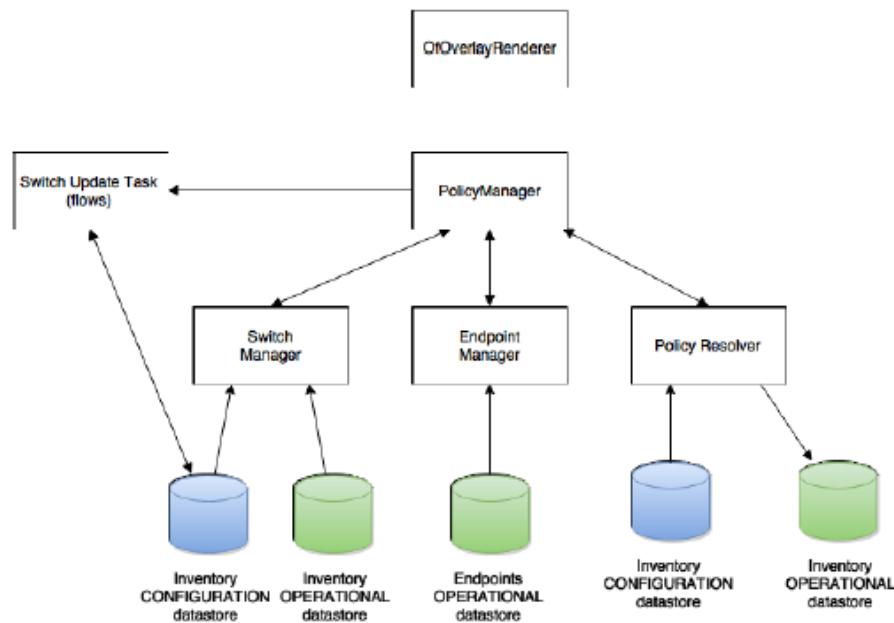
以下是GBP的基本组成部分，其中OfOverlay组件已经标红。

Figure 10.32. OfOverlay within GBP



GBP OfOverlay renderer内部组件：

Figure 10.33. OfOverlay expanded view:



OfOverlay Renderer内部组件如下：

Policy Resolver: 策略解析完全是独立域的，OfOverlay采用内部过程策略信息，见策略解析过程（链接）。

该模块侦听Tenants configuration datastore模块，验证租户输入的内容，然后写入到Tenants operational datastore中。

该模块将生成的内部通知发送给PolicyManager模块。

在下一版本中，将会采用一个non-renderer specific location。

Endpoint Manager: 锂版本中，终端库以编排器模式运行着，这就意味着用户负责通过UX/GUI、REST API提供终端。

注意：当使用Neutron mapper功能组件时，所有东西都是通过Neutron管理。

终端管理负责侦听终端库的更新，并且当一个有效终端成功注册后向交换机管理发送提醒。

还为流水线处理流提供多个功能。

Switch Manager

Switch Manager是在Lithium中重构的一个纯状态管理器。

交换机有三种状态：

DISCONNECTED

PREPARING

READY

ready表示交换机可连接：

有一个隧道接口

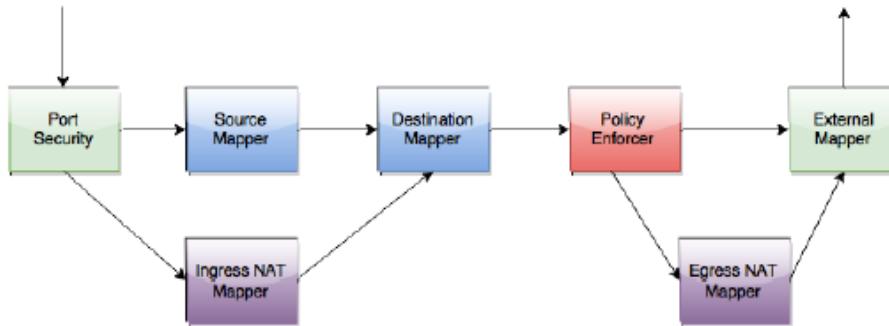
至少有一个连接的终端

这种状态下，GBP没有写入交换机，所以没有任何业务。

preparing仅仅意味着交换机连接了控制器，但是缺少上面说的其中一个条件。

disconnected意味着之前连接的交换机不会再出现在Inventory动态数据库中。

图10.34 OfOverlay流管道



OfOverlay利用Nicira注册如下信息：

REG0 = Source EndpointGroup + Tenant ordinal

REG1 = Source Conditions + Tenant ordinal

REG2 = Destination EndpointGroup + Tenant ordinal

REG3 = Destination Conditions + Tenant ordinal

REG4 = Bridge Domain + Tenant ordinal

REG5 = Flood Domain + Tenant ordinal

REG6 = Layer 3 Context + Tenant ordinal

端口安全

OpenFlow管道的Table 0负责确保只有有效的连接可以发送数据包到管道中：

```

cookie=0x0, <snip>, priority=200, in_port=3 actions=goto_table:2
cookie=0x0, <snip>, priority=200, in_port=1 actions=goto_table:1
cookie=0x0, <snip>,
priority=121, arp, in_port=5, dl_src=fa:16:3e:d5:b9:8d, arp_spa=10.1.1.3
actions=goto_table:2
cookie=0x0, <snip>,
priority=120, ip, in_port=5, dl_src=fa:16:3e:d5:b9:8d, nw_src=10.1.1.3
actions=goto_table:2
cookie=0x0, <snip>,
priority=115, ip, in_port=5, dl_src=fa:16:3e:d5:b9:8d, nw_dst=255.255.255.255
actions=goto_table:2
cookie=0x0, <snip>, priority=112, ipv6 actions=drop
cookie=0x0, <snip>, priority=111, ip actions=drop
cookie=0x0, <snip>, priority=110, arp actions=drop
cookie=0x0, <snip>, in_port=5, dl_src=fa:16:3e:d5:b9:8d actions=goto_table:2
cookie=0x0, <snip>, priority=1 actions=drop
  
```

隧道接口的入口，进入到Table Source Mapper：

```
cookie=0x0, <snip>, priority=200, in_port=3 actions=goto_table:2
```

外部入口，进入Table Ingress NAT Mapper：

```
cookie=0x0, <snip>, priority=200, in_port=1 actions=goto_table:1
```

终端ARP，进入Table Source Mapper：

```
cookie=0x0, <snip> ,
priority=121, arp, in_port=5, dl_src=fa:16:3e:d5:b9:8d, arp_spa=10.1.1.3
actions=goto_table:2
```

终端IPv4，进入Table Source Mapper:

```
cookie=0x0, <snip> ,
priority=120, ip, in_port=5, dl_src=fa:16:3e:d5:b9:8d, nw_src=10.1.1.3
actions=goto_table:2
```

终端DHCP DORA，进入Table Source Mapper:

```
cookie=0x0, <snip> ,
priority=115, ip, in_port=5, dl_src=fa:16:3e:d5:b9:8d, nw_dst=255.255.255.255
actions=goto_table:2
```

设置一些带有优先级的DROP表捕获任何不确定的流，这些流匹配上述条件:

```
cookie=0x0, <snip> , priority=112, ipv6 actions=drop
cookie=0x0, <snip> , priority=111, ip actions=drop
cookie=0x0, <snip> , priority=110, arp actions=drop
```

"L2"抓取所有上面没有识别的流:

```
cookie=0x0, <snip> , in_port=5, dl_src=fa:16:3e:d5:b9:8d actions=goto_table:2
```

Drop Flow:

```
cookie=0x0, <snip> , priority=1 actions=drop
```

进入NAT Mapper

表offset+1。

外部NAT地址的ARP应答器:

```
cookie=0x0, <snip> , priority=150, arp, arp_tpa=192.168.111.51, arp_op=1
actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[], set_field:fa:16:3e:58:c3:dd-
>eth_src, load:0x2->NXM_OF_ARP_OP[], move:NXM_NX_ARP_SHA[]-
>NXM_NX_ARP_THA[], load:0xfa163e58c3dd->NXM_NX_ARP_SHA[], move:NXM_OF_ARP_SPA[]-
>NXM_OF_ARP_TPA[], load:0xc0a86f33->NXM_OF_ARP_SPA[], IN_PORT
```

从外部转换到内部，并且完成和SourceMapper相同的功能:

```
cookie=0x0, <snip> , priority=100, ip, nw_dst=192.168.111.51
actions=set_field:10.1.1.2->ip_dst, set_field:fa:16:3e:58:c3:dd-
>eth_dst, load:0x2->NXM_NX_REG0[], load:0x1->NXM_NX_REG1[], load:0x4-
>NXM_NX_REG4[], load:0x5->NXM_NX_REG5[], load:0x7->NXM_NX_REG6[], load:0x3-
>NXM_NX_TUN_ID[0..31], goto_table:3
```

Source Mapper

表offset+2。

基于入端口特性的决定:

EndpointGroup(s) it belongs to

Forwarding context

Tunnel VNID ordinal

在有效的目的交换机上建立隧道。

带有VNID Ordinal的远程节点的入隧道映射到Source EPG、Forwarding Context等：

```
cookie=0x0, <snip>, priority=150, tun_id=0xd, in_port=3 actions=load:0xc->NXM_NX_REG0[], load:0xfffffff->NXM_NX_REG1[], load:0x4->NXM_NX_REG4[], load:0x5->NXM_NX_REG5[], load:0x7->NXM_NX_REG6[], goto_table:3
```

映射终端到Source EPG、基于入端口的Forwarding Context和MAC：

```
cookie=0x0, <snip>, priority=100, in_port=5, dl_src=fa:16:3e:b4:b4:b1 actions=load:0xc->NXM_NX_REG0[], load:0x1->NXM_NX_REG1[], load:0x4->NXM_NX_REG4[], load:0x5->NXM_NX_REG5[], load:0x7->NXM_NX_REG6[], load:0xd->NXM_NX_TUN_ID[0..31], goto_table:3
```

一般的drop：

```
cookie=0x0, duration=197.622s, table=2, n_packets=0, n_bytes=0, priority=1 actions=drop
```

Destination Mapper

表offset+3。

基于终端特性的决定：

EndpointGroup(s) it belongs to

Forwarding context

Tunnel Destination value

管理基于有效入节点ARP的路由，适用于默认网关，匹配网关MAC或者目的终端MAC。

[10.1.1.0/24子网的默认网关的ARP](#)：

```
cookie=0x0, <snip>, priority=150, arp, reg6=0x7, arp_tpa=10.1.1.1, arp_op=1 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[], set_field:fa:16:3e:28:4c:82->eth_src, load:0x2->NXM_OF_ARP_OP[], move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[], load:0xfa163e284c82->NXM_NX_ARP_SHA[], move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[], load:0xa010101->NXM_OF_ARP_SPA[], IN_PORT
```

通往GroupTable的广播流：

```
cookie=0x0, <snip>, priority=140, reg5=0x5, dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=load:0x5->NXM_NX_TUN_ID[0..31], group:5
```

3层目的匹配流，priority=100+masklength。现在GBP支持L3Prefix终端，我们可以设置默认路由等：

```
cookie=0x0, <snip>, priority=132, ip, reg6=0x7, dl_dst=fa:16:3e:b4:b4:b1, nw_dst=10.1.1.3 actions=load:0xc->NXM_NX_REG2[], load:0x1->NXM_NX_REG3[], load:0x5->NXM_NX_REG7[], set_field:fa:16:3e:b4:b4:b1->eth_dst, dec_ttl, goto_table:4
```

2层目的匹配流，在最后一个IP流之后被捕捉（IP流的最低优先级值为100）：

```
cookie=0x0, duration=323.203s, table=3, n_packets=4, n_bytes=168,
priority=50, reg4=0x4, dl_dst=fa:16:3e:58:c3:dd actions=load:0x2-
>NXM_NX_REG2[], load:0x1->NXM_NX_REG3[], load:0x2->NXM_NX_REG7[], goto_table:4
```

普通的drop流： cookie=0x0, duration=323.207s, table=3, n_packets=6, n_bytes=588,priority=1 actions=drop

Policy Enforcer

表offset+4。

一旦分配了Source和Destination EndpointGroups，政策的实施基于被解决的规则。

在Service Function Chaining案例中，封装流和目的流被发现和执行。

Policy流，允许EndpointGroups间的IP流通过：

```
cookie=0x0, <snip>, priority=64998, ip, reg0=0x8, reg1=0x1, reg2=0xc, reg3=0x1
actions=goto_table:5
```

Egress NAT Mapper

表offset+5。

在Egressing OVS实例到underlay网络中之前执行NAT功能。

在发送到underlay网络之前的NAT内外网转换：

```
cookie=0x0, <snip>, priority=100, ip, reg6=0x7, nw_src=10.1.1.2
actions=set_field:192.168.111.51->ip_src, goto_table:6
```

External Mapper

表offset+6。

管理Service Function Chaining，所以能够支持同步和异步的服务链，分布式的输入输出分类。

一般形式：

```
cookie=0x0, <snip>, priority=100 actions=output:NXM_NX_REG7[]
```

通过REST进行OpenFlow Overlay配置

注：请通过UX内容了解如何通过GUI配置**GBP**

Endpoint

```
POST http://{{controllerIp}}:8181/restconf/operations/endpoint:registerendpoint
{
  "input": {
    "endpoint-group": "<epg0>",
    "endpoint-groups": ["<epg1>", "<epg2>"],
    "network-containment": "<forwarding-model-context1>",
    "12-context": "<bridge-domain1>",
    "mac-address": "<mac1>",
    "13-address": [
      {
        "ip-address": "<ipaddress1>",
        "port": 1
      }
    ]
  }
}
```

```

        "13-context": "<13_context1>"
    }
],
"*ofoverlay:port-name*": "<ovs port name>",
"tenant": "<tenant1>"
}
}

```

OVS拓展详细目录

```

PUT http://{{controllerIp}}:8181/restconf/config/opendaylight-inventory:nodes/
{
    "opendaylight-inventory:nodes": {
        "node": [
            {
                "id": "openflow:123456",
                "ofoverlay:tunnel": [
                    {
                        "tunnel-type": "overlay:tunnel-type-vxlan",
                        "ip": "<ip_address_of_ovs>",
                        "port": 4789,
                        "node-connector-id": "openflow:123456:1"
                    }
                ]
            },
            {
                "id": "openflow:654321",
                "ofoverlay:tunnel": [
                    {
                        "tunnel-type": "overlay:tunnel-type-vxlan",
                        "ip": "<ip_address_of_ovs>",
                        "port": 4789,
                        "node-connector-id": "openflow:654321:1"
                    }
                ]
            }
        ]
    }
}

```

Tenants的详细细节看Policy Resolution和Forwarding Model

```

{
    "policy:tenant": {
        "contract": [
            {
                "clause": [
                    {
                        "name": "allow-http-clause",
                        "subject-refs": [
                            "allow-http-subject",
                            "allow-icmp-subject"
                        ]
                    }
                ],
                "id": "<id>",
                "subject": [
                    {
                        "name": "allow-http-subject",
                        "rule": [
                            {

```

```
"classifier-ref": [
  {
    "direction": "in",
    "name": "http-dest"
  },
  {
    "direction": "out",
    "name": "http-src"
  }
],
"action-ref": [
  {
    "name": "allow1",
    "order": 0
  }
],
"name": "allow-http-rule"
}
]
},
{
  "name": "allow-icmp-subject",
  "rule": [
    {
      "classifier-ref": [
        {
          "name": "icmp"
        }
      ],
      "action-ref": [
        {
          "name": "allow1",
          "order": 0
        }
      ],
      "name": "allow-icmp-rule"
    }
  ]
},
"endpoint-group": [
  {
    "consumer-named-selector": [
      {
        "contract": [
          "<id>"
        ],
        "name": "<name>"
      }
    ],
    "id": "<id>",
    "provider-named-selector": []
  },
  {
    "consumer-named-selector": [],
    "id": "<id>",
    "provider-named-selector": [
      {
        "contract": [
          "<id>"
        ]
      }
    ]
  }
]
```

```
        ],
        "name": "<name>"
    }
]
},
],
"id": "<id>",
"12-bridge-domain": [
{
    "id": "<id>",
    "parent": "<id>"
}
],
"12-flood-domain": [
{
    "id": "<id>",
    "parent": "<id>"
},
{
    "id": "<id>",
    "parent": "<id>"
}
],
"13-context": [
{
    "id": "<id>"
}
],
"name": "GBPOC",
"subject-feature-instances": {
"classifier-instance": [
{
    "classifier-definition-id": "<id>",
    "name": "http-dest",
    "parameter-value": [
{
    "int-value": "6",
    "name": "proto"
},
{
    "int-value": "80",
    "name": "destport"
}
]
},
{
    "classifier-definition-id": "<id>",
    "name": "http-src",
    "parameter-value": [
{
    "int-value": "6",
    "name": "proto"
},
{
    "int-value": "80",
    "name": "sourceport"
}
]
},
{
    "classifier-definition-id": "<id>",
    "name": "icmp",
    "parameter-value": []
}
]
```

```
"parameter-value": [
    {
        "int-value": "1",
        "name": "proto"
    }
],
"action-instance": [
    {
        "name": "allow1",
        "action-definition-id": "<id>"
    }
]
},
"subnet": [
    {
        "id": "<id>",
        "ip-prefix": "<ip_prefix>",
        "parent": "<id>",
        "virtual-router-ip": "<ip address>"
    },
    {
        "id": "<id>",
        "ip-prefix": "<ip prefix>",
        "parent": "<id>",
        "virtual-router-ip": "<ip address>"
    }
]
}
```

指南

所有的详细demo环境可以查看GBP wiki

[https://wiki.opendaylight.org/view/Group_Based_Policy_\(GBP\)](https://wiki.opendaylight.org/view/Group_Based_Policy_(GBP))

功能服务链SFC使用GBP

概述

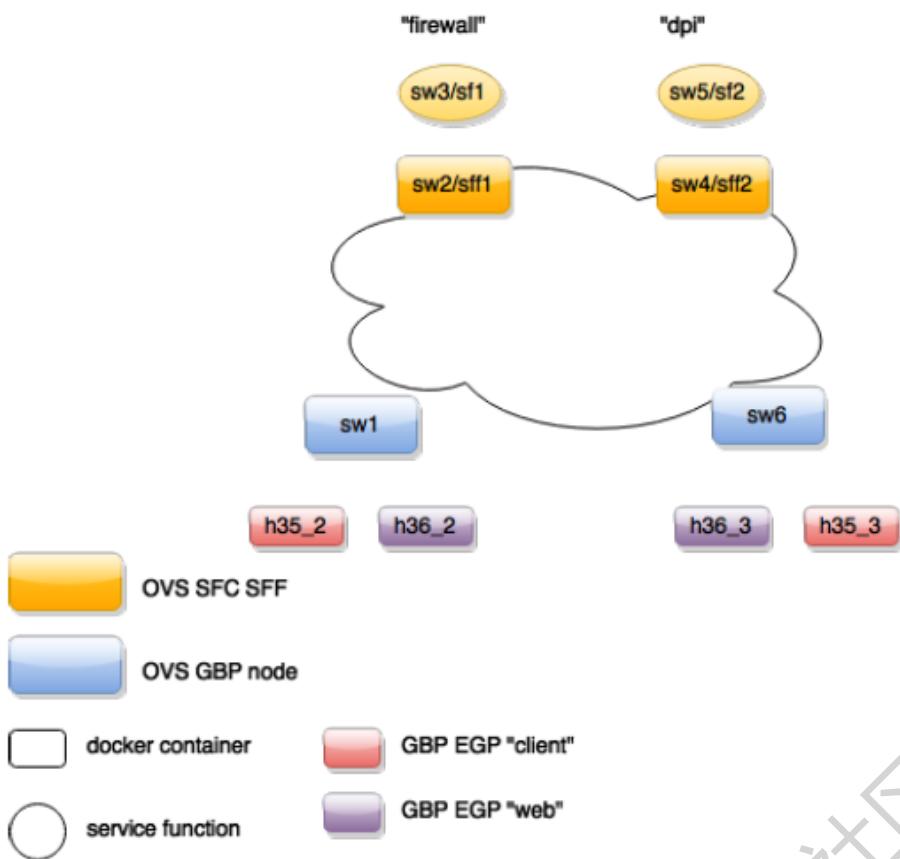
详细的原理和流程请参照Service Function Chaining项目

GBP允许在策略上使用服务链。

这个具体表现为GBP的action。

使用GBP demo和开发环境的例子

图 10.35 GBP和SFC集成环境

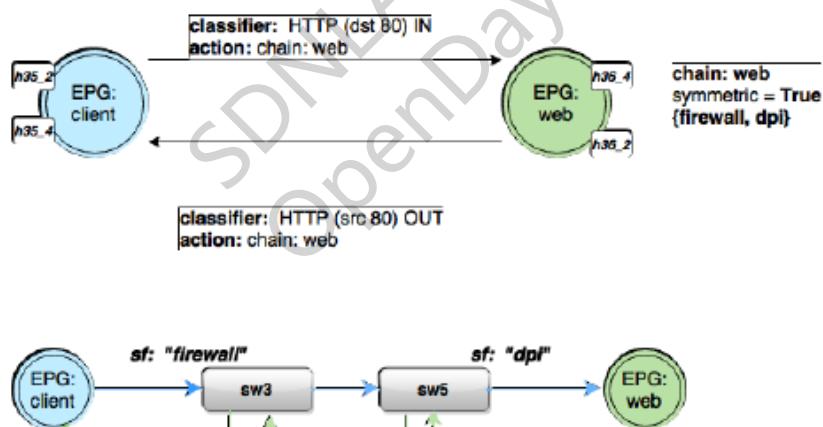


在这个拓扑中，H35_2和H36_3之间有一个对称的服务链路径：

H35_2到sw1到sff1到sf1到sff2到sf2到sff2到sw6到H36_3

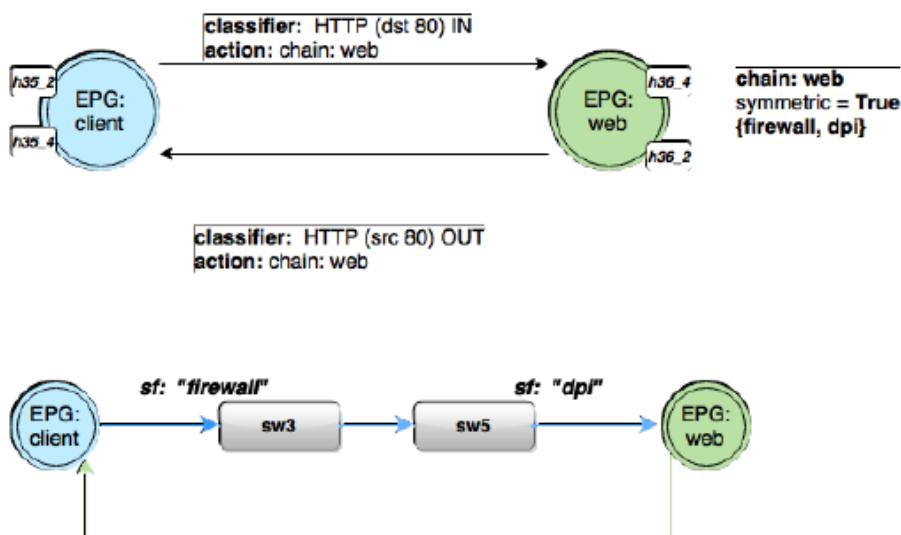
如果期望是对称服务链，那么返回的路径是：

图 10.36 GBP和SFC对称服务链环境



如果期望是非对称的服务链，那么返回的路径可能是直接的或完全不同的服务链。

图 10.37 GBP和SFC的非对称服务链环境



Lithium支持所有的这些特性。

在**Subject Feature Instance**租户配置内容中，我们定义了ICMP和HTTP的实例。

```

"subject-feature-instances": {
    "classifier-instance": [
        {
            "name": "icmp",
            "parameter-value": [
                {
                    "name": "proto",
                    "int-value": 1
                }
            ]
        },
        {
            "name": "http-dest",
            "parameter-value": [
                {
                    "int-value": "6",
                    "name": "proto"
                },
                {
                    "int-value": "80",
                    "name": "destport"
                }
            ]
        },
        {
            "name": "http-src",
            "parameter-value": [
                {
                    "int-value": "6",
                    "name": "proto"
                },
                {
                    "int-value": "80",
                    "name": "sourceport"
                }
            ]
        }
    ],

```

然后action实例将与匹配classifier定义的流量进行交互。

注意，*SFC Chain name*在SFC中必须存在。

```
"action-instance": [
  {
    "name": "chain1",
    "parameter-value": [
      {
        "name": "sfc-chain-name",
        "string-value": "SFCGBP"
      }
    ]
  },
  {
    "name": "allow1",
  }
],
},
```

匹配ICMP包的规则，使流量通过服务链

```
"contract": [
  {
    "subject": [
      {
        "name": "icmp-subject",
        "rule": [
          {
            "name": "allow-icmp-rule",
            "order": 0,
            "classifier-ref": [
              {
                "name": "icmp"
              }
            ],
            "action-ref": [
              {
                "name": "allow1",
                "order": 0
              }
            ]
          }
        ]
      },
      {
        "name": "http-subject",
        "rule": [
          {
            "name": "http-chain-rule-in",
            "classifier-ref": [
              {
                "name": "http-dest",
                "direction": "in"
              }
            ],
            "action-ref": [

```

当匹配了HTTP，对输入的包支持TCP目的端口（80端口）或HTTP请求筛选，对输出的包支持TCP源端口（80端口）或HTTP应答筛选。

```
{
  "name": "http-subject",
  "rule": [
    {
      "name": "http-chain-rule-in",
      "classifier-ref": [
        {
          "name": "http-dest",
          "direction": "in"
        }
      ],
      "action-ref": [
```

```

        {
            "name": "chain1",
            "order": 0
        }
    ],
},
{
    "name": "http-chain-rule-out",
    "classifier-ref": [
        {
            "name": "http-src",
            "direction": "out"
        }
    ],
    "action-ref": [
        {
            "name": "chain1",
            "order": 0
        }
    ]
}
]
}

```

使用非对称的服务链，用户希望只有HTTP请求能够穿过服务链，HTTP应答不穿过。那么将HTTP应答设置为`allow`代替`chain`:

```

{
    "name": "http-chain-rule-out",
    "classifier-ref": [
        {
            "name": "http-src",
            "direction": "out"
        }
    ],
    "action-ref": [
        {
            "name": "allow1",
            "order": 0
        }
    ]
}

```

Demo/环境部署

Lithium版本的GBP项目有两个Demo/环境部署

Docker based GBP and GBP+SFC integration Vagrant environment

DevStack based GBP+Neutron integration Vagrant environment

Demo @ GBP wiki

[https://wiki.opendaylight.org/view/Group_Based_Policy_\(GBP\)/Consumability/Demo](https://wiki.opendaylight.org/view/Group_Based_Policy_(GBP)/Consumability/Demo)

11 L2Switch用户指南

概述

L2Switch项目提供L2交换机功能。

L2Switch架构

包处理器：对发往

控制器的包进行解码，且进行调度；

环删除器：删除网络中的环路

ARP处理器：处理已解码成ARP的包

地址跟踪器：网络中实体地址学习（MAC和IP）

主机追踪器：网络中追踪主机的位置

L2switch Main：在每个基于网络流量的交换机上安装流

配置L2Switch

下面部分将给出有关可配置组件的详细配置信息。

配置Loop删除器

52-loopremover.xml文件中，将对以下参数进行配置：

is-install-lldp-flow: “true”表示在每个交换机上安装发送所有LLDP数据包给控制器的流；“false”表示这条流将不被安装。

lldp-flow-table-id: LLDP流将被安装到每个交换机的指定流表中，但是只有“is-install-lldp-flow”设置为“true”时，才生效。

lldp-flow-priority: LLDP流随着设置指定优先级被安装，同样的，只有“is-install-lldp-flow”设置为“true”时，此参数才生效。

lldp-flow-idle-timeout: 如果流没有在X秒之内转发包，LLDP流将超时（这条流将从交换机中删除），且只有“is-install-lldp-flow”设置为“true”时，才生效。

lldp-flow-hard-timeout: 不管多少数据包正在转发，X秒后LLDP流将超时（这条流将从交换机中删除），且只有“is-install-lldp-flow”设置为“true”时，才生效。

graph-refresh-delay: 当网络元素添加或移除时，维护并更新网络图，如链路和交换机。一个网络元素添加或移除后，重新计算网络图之前，等待 **graph-refresh-delay** 几秒。因为图没有立即更新，有丢包的潜在可能，不过 **higher value** 有利于减少图的更新；在更多的计算成本中，**lower value** 有利于较快的处理网络拓扑改变。

配置ARP处理器

54-arpandler.xml文件中，将对以下参数进行配置：

is-proactive-flood-mode:

“true”表示泛洪流（flood flows）安装在每个交换机上。随着这条泛洪流，每个交换机将泛洪没有匹配其他任何流

的一个包。其中，“Advantage”是指较少的包发送到控制器，因为这些包被泛洪在网络中；“Disadvantage”指很多网络流量被产生。

“false”表示此泛洪流将不被安装，相对的，将会在每个交换机中安装一条ARP流发送所有的ARP数据包给控制器。其中，“Advantage”是指产生较少的流量；“Disadvantage”指控制器处理更多的包（ARP请求&回复）且ARP进程花费比如果有flood流的时间长。

flood-flow-table-id: 泛洪流被安装在每个交换机的指定流表中，但只有“is-proactive-flood-mode”设置为“true”时，才生效。

flood-flow-priority: 安装泛洪流并指定其优先级，同样的，只有“is-proactive-flood-mode”设置为“true”时，才生效。

flood-flow-idle-timeout: 如果流没有在X秒之内转发包，泛洪流将超时（这条流将从交换机中删除），且只有“is-proactive-flood-mode”设置为“true”时，才生效。

flood-flow-hard-timeout: 不管多少数据包正在转发，X秒后泛洪流将超时（这条流将从交换机中删除），且只有“is-proactive-flood-mode”设置为“true”时，才生效。

arp-flow-table-id: 在每个交换机的指定流表中安装ARP流，且只有“is-proactive-flood-mode”设置为“false”时，才生效。

arp-flow-priority: 安装ARP流并指定优先级，只有“is-proactive-flood-mode”设置为“false”时，才生效。

arp-flow-idle-timeout: 如果流没有在X秒之内转发包，ARP流将超时（这条流将从交换机中删除），且只有“is-proactive-flood-mode”设置为“false”时，才生效。

arp-flow-hard-timeout: 不管多少数据包正在转发，arp-flow-hard-timeout一些秒后ARP流将超时（这条流将从交换机中删除），且只有“is-proactive-flood-mode”设置为“false”时，才生效。

配置地址追踪器

56-addressstracker.xml文件中，将对以下参数进行配置：

timestamp-update-interval: 一个最新看到的时间戳与每个地址相关。timestamp-update-interval几十毫秒后，将更新最新看到的时间戳。“higher value”有利于对数据库进行更少的写操作；“lower value”有利于了解怎样刷新一个地址。

observe-addresses-from: 能从ARP、IPv4、IPv6包中观察到或学习到IP和MAC地址。

配置L2Switch Main

58-l2switchmain.xml文件中，将对以下参数进行配置：

is-install-dropall-flow: “true”表示将在每个交换机中安装一个drop-all流，所以默认的action将是丢弃一个包而不是发送包到控制器。“false”表示drop-all流将不被安装。

dropall-flow-table-id: 在每个交换机的指定流表上安装dropall流，但是只有“is-install-dropall-flow”设置为“true”时，才生效。

dropall-flow-priority: 安装dropall流并指定优先级，同样的，只有“is-install-dropall-flow”设置为“true”时，才生效。

dropall-flow-idle-timeout: 如果流没有在X秒之内转发包，dropall流将超时（这条流将从交换机中删除），且只有“is-install-dropall-flow”设置为“true”时，才生效。

dropall-flow-hard-timeout: 不管多少数据包正在转发，X秒后dropall流将超时（这条流将从交换机中删除），

且只有"is-install-dropall-flow"设置为"true"时，才生效。

is-learning-only-mode: "true"表示L2Switch将只学

习地址，不会安装额外的优化网络流量的流。"false"表示L2Switch将响应网络流量并在每个交换机上安装流来优化流量。当前模式下，安装MAC-to-MAC流。

reactive-flow-table-id: 将在每个交换机的指定流表中安装reactive流，但是只有"is-learning-only-mode"设置为"false"时，才生效。

reactive-flow-priority: 安装reactive流并指定优先级，同样的，只有"is-learning-only-mode"设置为"false"时，才生效。

reactive-flow-idle-timeout: 如果流没有在X秒之内转发包，reactive流将超时（这条流将从交换机中删除），且只有"is-learning-only-mode"设置为"false"时，才生效。

reactive-flow-hard-timeout: 不管多少数据包正在转发，X秒后reactive流将超时（这条流将从交换机中删除），且只有"is-learning-only-mode"设置为"false"时，才生效。

运行L2Switch项目

为了运行L2Switch，Lithium OpenDaylight里只需简单安装odl-l2switch-switch-ui功能：

```
feature:install odl-l2switch-switch-ui
```

使用Mininet创建网络

Mininet使用命令：

```
sudo mn --controller=remote, ip= --topo=linear, 3 --switch  
ovsk, protocols=OpenFlow13  
sudo mn --controller=remote, ip=127.0.0.1 --topo=linear, 3 --switch  
ovsk, protocols=OpenFlow13
```

以上的命令将创建一个包含3个交换机，且每个交换机下挂一个主机的虚拟网络，每个交换机将连接到指定的控制器，如本命令中的本机IP地址127.0.0.1。

```
sudo mn --controller=remote, ip=127.0.0.1 --mac --topo=linear, 3 --switch  
ovsk, protocols=OpenFlow13
```

以上命令含有"mac"选项，它使主机MAC地址和交换机MAC地址间更容易区分。

使用Mininet产生网络流量

```
h1 ping h2
```

上面命令指主机host1 (h1) 对主机host2 (h2) 进行ping操作。

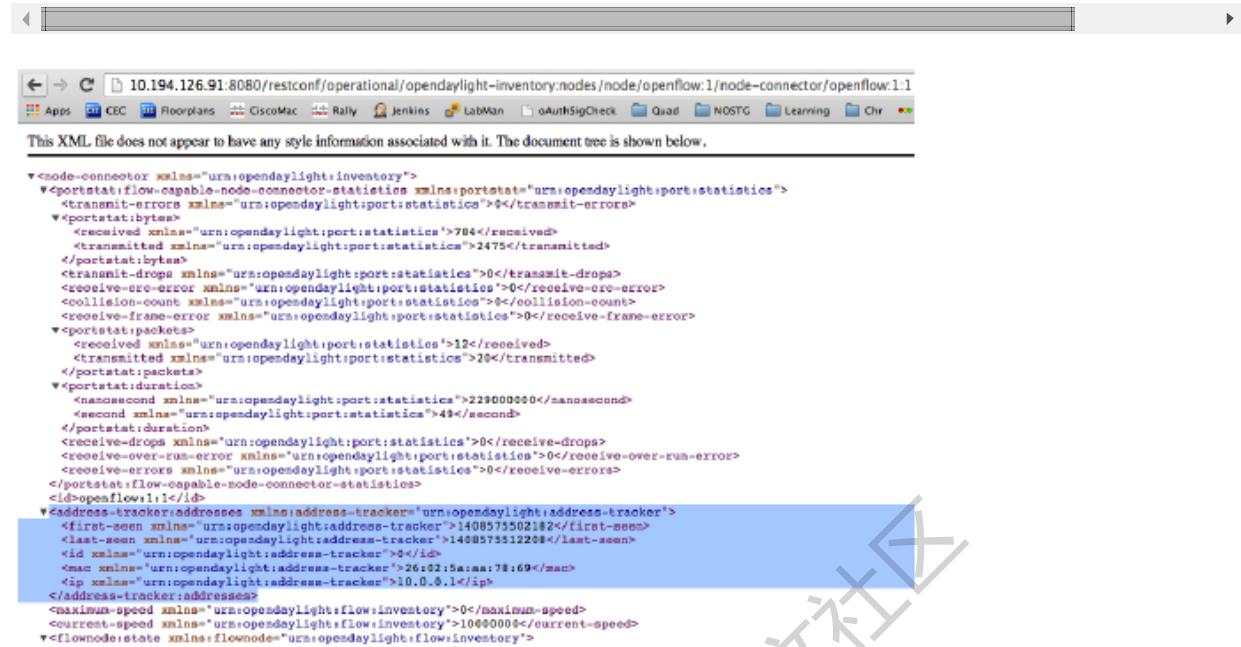
```
pingall
```

pingall将使每个主机和其他每个主机进行ping操作。

检查Address Observations

Address Observations被添加在库数据树中。在一个节点连接器上的Address Observations能够通过浏览器或者REST客户端来检查。

<http://10.194.126.91:8080/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/node-connector>



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```

<node-connector xmlns="urn:opendaylight:inventory">
  <portstat:flow-capable-node-connector-statistics xmlns:portstat="urn:opendaylight:port:statistics">
    <transmit-errors xmlns="urn:opendaylight:port:statistics">0</transmit-errors>
    <portstat:bytes>
      <received xmlns="urn:opendaylight:port:statistics">784</received>
      <transmitted xmlns="urn:opendaylight:port:statistics">2475</transmitted>
    </portstat:bytes>
    <transmit-drops xmlns="urn:opendaylight:port:statistics">0</transmit-drops>
    <receive-crc-error xmlns="urn:opendaylight:port:statistics">0</receive-crc-error>
    <collision-count xmlns="urn:opendaylight:port:statistics">0</collision-count>
    <receive-frame-error xmlns="urn:opendaylight:port:statistics">0</receive-frame-error>
  </portstat:bytes>
  <portstat:packets>
    <received xmlns="urn:opendaylight:port:statistics">12</received>
    <transmitted xmlns="urn:opendaylight:port:statistics">20</transmitted>
  </portstat:packets>
  <portstat:duration>
    <nanosecond xmlns="urn:opendaylight:port:statistics">229000000</nanosecond>
    <second xmlns="urn:opendaylight:port:statistics">49</second>
  </portstat:duration>
  <receive-drops xmlns="urn:opendaylight:port:statistics">0</receive-drops>
  <receive-over-run-error xmlns="urn:opendaylight:port:statistics">0</receive-over-run-error>
  <receive-errors xmlns="urn:opendaylight:port:statistics">0</receive-errors>
</portstat:flow-capable-node-connector-statistics>
<id>openflow:1</id>
<address-tracker:addresses xmlns:address-tracker="urn:opendaylight:address-tracker">
  <first-seen xmlns="urn:opendaylight:address-tracker">1408575502182</first-seen>
  <last-seen xmlns="urn:opendaylight:address-tracker">1408575512208</last-seen>
  <id xmlns="urn:opendaylight:address-tracker">0</id>
  <mac xmlns="urn:opendaylight:address-tracker">26:02:15:aa:78:69</mac>
  <ip xmlns="urn:opendaylight:address-tracker">10.0.0.1</ip>
</address-tracker:addresses>
<maximum-speed xmlns="urn:opendaylight:flow:inventory">0</maximum-speed>
<current-speed xmlns="urn:opendaylight:flow:inventory">1048000</current-speed>
<flownode:state xmlns:flownode="urn:opendaylight:flow:inventory">

```

图11.1 Address Observations

检查主机

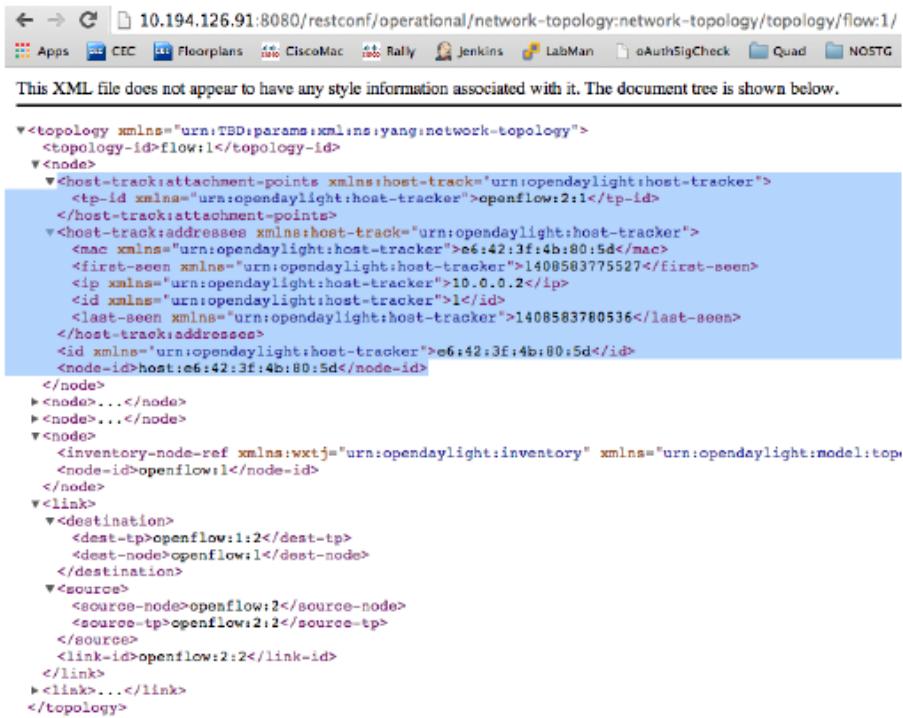
主机信息被添加在拓扑数据树中：

主机地址

到一个节点/交换机的连接点（链路）

主机信息和连接点信息可以通过浏览器或者REST客户端检查：

<http://10.194.126.91:8080/restconf/operational/network-topology:network-topology/topology:flow:1/>



```

<?xml version="1.0" encoding="UTF-8"?>
<topology xmlns="urn:TBD:params:xml:ns:yang:network-topology">
  <topology-id>flow:1</topology-id>
  <node>
    <host-track:attachment-points xmlns="urn:opendaylight:host-tracker">
      <tp-id xmlns="urn:opendaylight:host-tracker">openflow:2:1</tp-id>
    </host-track:attachment-points>
    <host-track:addresses xmlns="urn:opendaylight:host-tracker">
      <mac xmlns="urn:opendaylight:host-tracker">e6:42:3f:4b:80:5d</mac>
      <first-seen xmlns="urn:opendaylight:host-tracker">1408583775527</first-seen>
      <ip xmlns="urn:opendaylight:host-tracker">10.0.0.2</ip>
      <id xmlns="urn:opendaylight:host-tracker">1</id>
      <last-seen xmlns="urn:opendaylight:host-tracker">1408583780536</last-seen>
    </host-track:addresses>
    <id xmlns="urn:opendaylight:host-tracker">e6:42:3f:4b:80:5d</id>
    <node-id>host:e6:42:3f:4b:80:5d</node-id>
  </node>
  <node>...</node>
  <node>...</node>
  <node>
    <inventory-node-ref xmlns="urn:wxtj">urn:opendaylight:inventory</inventory-node-ref>
    <node-id>openflow:1</node-id>
  </node>
  <link>
    <destination>
      <dest-tp>openflow:1:2</dest-tp>
      <dest-node>openflow:1</dest-node>
    </destination>
    <source>
      <source-node>openflow:2</source-node>
      <source-tp>openflow:2:2</source-tp>
    </source>
    <link-id>openflow:2:2</link-id>
  </link>
  <link>...</link>
</topology>

```

图11.2 主机

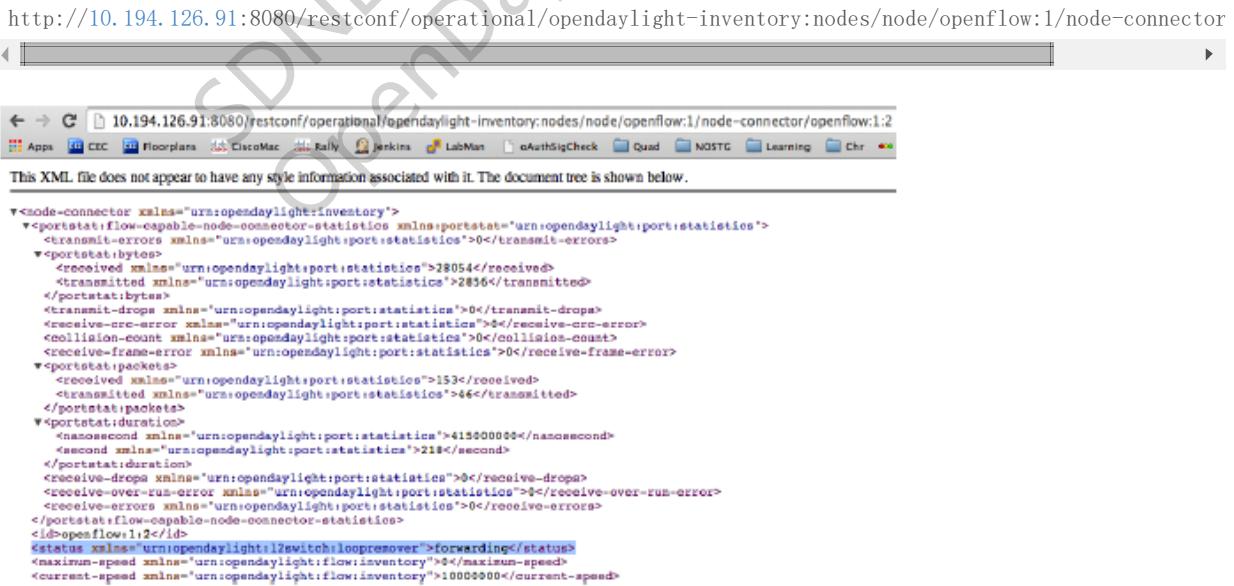
检查每条链路的STP状态

STP状态信息被添加在库数据树中：

“forwarding”状态表示链路是active且数据包正在链路上流动。

“discarding”状态表示链路是inactive且数据包没有通过他被发送。

链路的STP状态能够通过浏览器或者REST客户端来检查：



```

http://10.194.126.91:8080/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/node-connector
<?xml version="1.0" encoding="UTF-8"?>
<node-connector xmlns="urn:opendaylight:inventory">
  <port-stat-flow-capable-node-connector-statistic xmlns="urn:opendaylight:port:statistics">
    <transmit-errors xmlns="urn:opendaylight:port:statistics">0</transmit-errors>
  </port-stat-flow-capable-node-connector-statistic>
  <port-stat-bytes>
    <received xmlns="urn:opendaylight:port:statistics">28054</received>
    <transmitted xmlns="urn:opendaylight:port:statistics">28354</transmitted>
  </port-stat-bytes>
  <transmit-drops xmlns="urn:opendaylight:port:statistics">0</transmit-drops>
  <receive-err-error xmlns="urn:opendaylight:port:statistics">0</receive-err-error>
  <collisions-count xmlns="urn:opendaylight:port:statistics">0</collisions-count>
  <receive-frame-error xmlns="urn:opendaylight:port:statistics">0</receive-frame-error>
  <port-stat-packets>
    <received xmlns="urn:opendaylight:port:statistics">153</received>
    <transmitted xmlns="urn:opendaylight:port:statistics">84</transmitted>
  </port-stat-packets>
  <port-stat-duration>
    <nanossecond xmlns="urn:opendaylight:port:statistics">415600000</nanossecond>
    <second xmlns="urn:opendaylight:port:statistics">218</second>
  </port-stat-duration>
  <receive-drops xmlns="urn:opendaylight:port:statistics">0</receive-drops>
  <receive-over-run-error xmlns="urn:opendaylight:port:statistics">0</receive-over-run-error>
  <receive-error xmlns="urn:opendaylight:port:statistics">0</receive-error>
  </port-stat-flow-capable-node-connector-statistic>
  <id>openflow:1:2</id>
  <status xmlns="urn:opendaylight:l2switch:loopremover">forwarding</status>
  <maximum-speed xmlns="urn:opendaylight:flow:inventory">0</maximum-speed>
  <current-speed xmlns="urn:opendaylight:flow:inventory">10000000</current-speed>

```

图11.3 STP状态

其他Mininet命令

link s1 s2 down

将交换机switch1（s1）和switch2（s2）之间的链路down掉。

```
link s1 s2 up
```

将交换机switch1（s1）和switch2（s2）之间的链路up起来。

```
link s1 h1 down
```

将交换机switch1（s1）和主机host1（h1）之间的链路down掉。

12 L3VPN服务： 用户指南

目录

概述

模块和接口

配置步骤和简单配置

概述

OpenDaylight中的L3VPN服务提供了一个用于创建基于BGP-MP的L3VPN的框架。同样适用于创建用于DC云环境的网络虚拟化。

模块和接口

可通过使用如下模块实现L3VPN服务

VPN服务模块

1. **VPN Manager:** 创建并管理VPN和VPN接口
2. **BGP manager:** 配置BGP路由堆栈和提供用于路由服务的接口
3. **FIB manager:** 向FIB提供接口，创建并管理数据平面的转发规则
4. **Nexthop Manager:** 创建并管理下一条的出口指针，创建数据平面的出口规则
5. **Interface Manager:** 创建并管理不同类型的网络接口，例如VLAN、L3tunnel等
6. **Id Manager:** 对给定的key提供集群的唯一标识。用于给不同条目提供唯一的标识。
7. **MD-SAL Util:** 给MD-SAL提供接口。用于服务模块获取MD-SAL的数据库和服务。

上述的模块功能都是独立的，并且可以被其他服务调用。

配置接口

下面的模块将提供用户可配置L3VPN服务的配置接口。

1. BGP Manager
2. VPN Manager

3. Interface Manager

4. FIB Manager

配置接口详细信息

BGP Manager Interface

1. 数据节点路径:/config/bgp:bgp-router/

a. Fields :

- i. local-as-identifier
- ii. local-as-number

b. REST Methods : GET, PUT, DELETE, POST

2. 数据节点路径:/config/bgp:bgp-neighbors/

a. Fields :

- i. List of bgp-neighbor

b. REST Methods : GET, PUT, DELETE, POST

3. 数据节点路径:/config/bgp:bgp-neighbors/bgp-neighbor/{as-number}/

a. Fields :

- i. as-number
- ii. ip-address

b. REST Methods : GET, PUT, DELETE, POST

VPN Manager Interface

1. 数据节点路径:/config/l3vpn:vpn-instances/

a. Fields :

- i. List of vpn-instance

b. REST Methods : GET, PUT, DELETE, POST

2. 数据节点路径:/config/l3vpn:vpn-interfaces/vpn-instance

a. Fields :

- i. name
- ii. route-distinguisher
- iii. import-route-policy
- iv. export-route-policy

b. REST Methods : GET, PUT, DELETE, POST

3. 数据节点路径:/config/l3vpn:vpn-interfaces/

a. Fields :

- i. List of vpn-interface

b. REST Methods : GET, PUT, DELETE, POST

4. 数据节点路径:/config/l3vpn:vpn-interfaces/vpn-interface

a. Fields :

- i. name
- ii. vpn-instance-name

b. REST Methods : GET, PUT, DELETE, POST

5. 数据节点路径:/config/l3vpn:vpn-interfaces/vpn-interface/{name}/adjacency

a. Fields :

- i. ip-address

- ii. mac-address
- b. REST Methods : GET, PUT, DELETE, POST

Interface Manager Interface

1. 数据节点路径:/config/if:interfaces/interface
 - a. Fields:
 - i. name
 - ii. type
 - iii. enabled
 - iv. of-port-id
 - v. tenant-id
 - vi. base-interface
 - b. type specific fields
 - i. when type = l2vlan
 - A. vlan-id
 - ii. when type = stacked_vlan
 - A. stacked-vlan-id
 - iii. when type = l3tunnel
 - A. tunnel-type
 - B. local-ip
 - C. remote-ip
 - D. gateway-ip
 - iv. when type = mpls
 - A. list labelStack
 - B. num-labels
 - c. REST Methods : GET, PUT, DELETE, POST

FIB Manager Interface

1. 数据节点路径:/config/odl-fib:fibEntries/vrfTables
 - a. Fields :
 - i. List of vrfTables
 - b. REST Methods : GET, PUT, DELETE, POST
2. 数据节点路径:/config/odl-fib:fibEntries/vrfTables/{routeDistinguisher}/
 - a. Fields :
 - i. route-distinguisher
 - ii. list vrfEntries
 - A. destPrefix
 - B. label
 - C. nexthopAddress
 - b. REST Methods : GET, PUT, DELETE, POST
3. 数据节点路径:/config/odl-fib:fibEntries/ipv4Table
 - a. Fields :
 - i. list ipv4Entry
 - A. destPrefix
 - B. nexthopAddress
 - b. REST Methods : GET, PUT, DELETE, POST

配置步骤和简单配置

安装

1. 编辑etc/custom.properties并且设置如下属性：

vpnserver.bgpspeaker.host.name = < bgpserver-ip > < bgpserver-ip > 这里设置为BGP运行的主机IP地址。

1. 运行ODL并且安装VPN服务 `feature:install odl-vpnservice-core`

用REST接口设置L3VPN服务

必备条件

1. VRF支持的BGP堆栈需要安装并配置

在下面的步骤1中设置指定的BGP

1. 在交换机之间和交换机到网关节点之间使用ovsdb/ovs-vsctl创建GRE/VxLAN隧道

在下面的步骤2中创建l3tunnel接口对应的每个隧道接口的指定DS。

步骤1：配置BGP

1. 配置BGP Router

REST API:PUT /config/bgp:bgp-router/

Sample JSON Data

```
{  
    "bgp-router": {  
        "local-as-identifier": "10.10.10.10",  
        "local-as-number": 108  
    }  
}
```

2. 配置BGP Neighbors

REST API:PUT /config/bgp:bgp-neighbors/

Sample JSON Data

```
{  
    "bgp-neighbor" : [  
        {  
            "as-number": 105,  
            "ip-address": "169.144.42.168"  
        }  
    ]  
}
```

步骤2：创建Tunnel接口

创建用于对应所有GRE/VxLAN隧道（ovsdb创建）的l3tunnel接口。使用下面的REST接口

REST API:PUT /config/if:interfaces/if:interfacce

Sample JSON Data

```
{
  "interface": [
    {
      "name": "GRE_192.168.57.101_192.168.57.102",
      "type": "odl-interface:l3tunnel",
      "odl-interface:tunnel-type": "odl-interface:tunnel-type-gre",
      "odl-interface:local-ip": "192.168.57.101",
      "odl-interface:remote-ip": "192.168.57.102",
      "odl-interface:portId": "openflow:1:3",
      "enabled": "true"
    }
  ]
}
```

下面是这些配置的预期结果

1. 会产生唯一的If-index
2. 更新了Interface-state操作DS
3. 创建了对应的下一跳组条目（Nexthop Group Entry）

步骤3：OS创建Neutron端口并且附加VMs

在这一步用户创建虚拟机。

步骤4：创建虚拟机接口

对应步骤3创建的虚拟机创建l2vlan接口

REST API:PUT /config/if:interfaces/if:interface

Sample JSON Data

```
{
  "interface": [
    {
      "name": "dpn1-dp1.2",
      "type": "l2vlan",
      "odl-interface:of-port-id": "openflow:1:2",
      "odl-interface:vlan-id": "1",
      "enabled": "true"
    }
  ]
}
```

步骤5：创建VPN实例

REST API:PUT /config/l3vpn:vpn-instances/l3vpn:vpn-instance/

Sample JSON Data

```
{
    "vpn-instance": [
        {
            "description": "Test VPN Instance 1",
            "vpn-instance-name": "testVpn1",
            "ipv4-family": {
                "route-distinguisher": "4000:1",
                "export-route-policy": "4000:1,5000:1",
                "import-route-policy": "4000:1,5000:1"
            }
        }
    ]
}
```

下面是这些配置的预期结果

1. 分配并且在data-store中更新VPN ID
2. 在BGP中创建了相应的VRF
3. 如果已经配置了针对该VPN的vpn-interface，对应的执行在步骤5中会有定义

步骤5： 创建VPN-Interface和Local Adjacency

这一个将分为两个步骤完成

创建vpn-interface

REST API:PUT /config/l3vpn:vpn-interfaces/l3vpn:vpn-interface/

Sample JSON Data

```
{
    "vpn-interface": [
        {
            "vpn-instance-name": "testVpn1",
            "name": "dpn1-dp1.2"
        }
    ]
}
```

注释

这里的名称是在步骤3、4中创建的虚拟机接口的名称。

在vpn-interface上添加Adjacency

REST API:PUT /config/l3vpn:vpn-interfaces/l3vpn:vpn-interface/dpn1-dp1.3/adjacency

Sample JSON Data

```
{
    "adjacency": [
        {

```

```

        "ip-address" : "169.144.42.168",
        "mac-address" : "11:22:33:44:55:66"
    }
]
}

```

这是个列表，用户可以在一个vpn接口上定义多个adjacency

上面的这些步骤可以通过下面的一个步骤进行执行

```

{
  "vpn-interface": [
    {
      "vpn-instance-name": "testVpn1",
      "name": "dpn1-dp1.3",
      "odl-13vpn:adjacency": [
        {
          "odl-13vpn:mac_address": "11:22:33:44:55:66",
          "odl-13vpn:ip_address": "11.11.11.2",
        }
      ]
    }
  ]
}

```

下面是这些配置的预期结果

1. 会产生前缀标签并且存储在DS中
2. 相应接口的流被编写在Ingress表中
3. Local Egress Group创建完成
4. 前缀被添加到BGP
5. BGP将路由更新添加到FIB YANG Interface
6. FIB条目流被添加到了OF管道中的FIB表中

13 链路聚合控制协议使用指南

概述

本节描述如何使用OpenDaylight中LACP插件项目，包括配置。

链路聚合控制协议（LACP）架构

OpenDaylight LACP项目实现了链路聚合控制协议(LACP)，这作为MD-SAL服务模块用于自动探测和聚合多个OpenDaylight控制的网络和LACP-enabled终端或交换机之间的链路。最终一个逻辑通道的创建表示链路的聚合。链路聚合提供了链路回弹性和带宽聚合的功能。该架构的实现符合IEEE 802.3标准。

配置链路聚合控制协议

LACP功能可以在Karaf控制台启动，命令如下：

```
feature:install odl-lacp-ui
```

注：

- 1.确保传统(non-OpenFlow)交换机配置了LACP模式，能够在长时间超时情况下依旧活跃，允许OpenDaylight LACP插件响应交换机的消息。
- 2.想利用LACP-configured链路聚合组(LAGs)的流必须显式地使用LACP插件创建的一个OpenFlow组表条目。插件只会创建组表条目，不会自己编写任何的流。

管理或经营LACP

LACP-discovered网络资源和网络统计可以使用如下的REST API观察到：

- 1.列出节点的聚合器：

```
http://<ControllerIP>:8181/restconf/operational/opendaylightinventory:  
nodes/node/<node-id>
```

聚合器的信息将出现在 XML tag 中。

- 2.只查看聚合器的信息：

```
http://<ControllerIP>:8181/restconf/operational/opendaylightinventory:  
nodes/node/<node-id>/lacp-aggregators/<agg-id>
```

与聚合器相关的group ID能在 XML tag 中发现。

组表条目信息可以在OpenDaylight-inventory节点数据库中找到。

- 3.观察物理端口信息。

```
http://<ControllerIP>:8181/restconf/operational/opendaylightinventory:  
nodes/node/<node-id>/node-connector/<node-connector-id>
```

与聚合器相关的端口将会有tag，更新有效的聚合器信息。

教程

以下教程示范了如何为mininet拓扑创建LACP LAG。

mininet上的LACP拓扑创建示例

```
sudo mn --controller=remote, ip=<Controller IP> --topo=linear, 1 --switch  
ovsk, protocols=OpenFlow13
```

以上命令将创建有一个交换机和一台主机的虚拟网络。交换机将被连接到控制器。

一旦发现拓扑，使用以下的ovs-ofctl命令验证“dl_type”字段设置为“0x8809”的流条目的存在，该流条目用来处理LACP包：

```
ovs-ofctl -O OpenFlow13 dump-flows s1  
OFPST_FLOW reply (OF1.3) (xid=0x2):
```

```
cookie=0x3000000000000001e, duration=60.067s, table=0, n_packets=0, n_bytes=0,
priority=5, dl_dst=01:80:c2:00:00:02, dl_type=0x8809 actions=CONTROLLER:65535
```

在mininet中使用以下命令给交换机（s1）与主机（h1）间配置额外的两条链路：

```
mininet> py net.addLink(s1, net.get('h1'))
mininet> py s1.attach('s1-eth2')
```

LACP模块将监听传统（non-OpenFlow）交换机生成的LACP控制数据包。在我们的示例中，主机(h1)将作为LACP包生成器。为了生成LACP控制数据包，必须在主机（h1）上创建bond接口，模式类型设置为long-timeout LACP。为了配置bond接口，在/etc/modprobe.d/目录下创建一个bonding.conf文件，并且插入下面几行代码：

```
alias bond0 bonding
options bonding mode=4
```

这边mode=4代表LACP，并且默认long-timeout。

在mininet上使用以下命令让bond接口联合物理接口h1-eth0和h1-eth1作为主机（h1）上bond接口的成员：

```
mininet> py net.get('h1').cmd('modprobe bonding')
mininet> py net.get('h1').cmd('ip link add bond0 type bond')
mininet> py net.get('h1').cmd('ip link set bond0 address <bond-mac-address>')
mininet> py net.get('h1').cmd('ip link set h1-eth0 down')
mininet> py net.get('h1').cmd('ip link set h1-eth0 master bond0')
mininet> py net.get('h1').cmd('ip link set h1-eth1 down')
mininet> py net.get('h1').cmd('ip link set h1-eth1 master bond0')
mininet> py net.get('h1').cmd('ip link set bond0 up')
```

一旦bond0接口启动，主机（h1）将发送LACP包给交换机（s1）。然后，LACP模块通过主机（h1）和交换机（s1）间LACP数据包的交换创建LAG。在主机（h1）端观察bond接口的输出：

```
mininet> py net.get('h1').cmd('cat /proc/net/bonding/bond0')
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)
Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer2 (0)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0
802.3ad info
LACP rate: slow
Min links: 0
Aggregator selection policy (ad_select): stable
Active Aggregator Info:
    Aggregator ID: 1
    Number of ports: 2
    Actor Key: 33
    Partner Key: 27
    Partner Mac Address: 00:00:00:00:01:01
    Slave Interface: h1-eth0
    MII Status: up
    Speed: 10000 Mbps
    Duplex: full
    Link Failure Count: 0
    Permanent HW addr: 00:00:00:00:00:11
    Aggregator ID: 1
    Slave queue ID: 0
    Slave Interface: h1-eth1
    MII Status: up
    Speed: 10000 Mbps
```

```
Duplex: full  
Link Failure Count: 0  
Permanent HW addr: 00:00:00:00:00:12  
Aggregator ID: 1  
Slave queue ID: 0
```

在OpenFlow交换机上创建相关的组表条目，"type"设置为"select"，执行LAG功能。观察组条目：

```
mininet>ovs-ofctl -O Openflow13 dump-groups s1  
OFPST_GROUP_DESC reply (OF1.3) (xid=0x2):  
group_id=60169, type=select, bucket=weight:0, actions=output:1, output:2
```

在交换机上应用LAG功能，流要将action配置成GroupId，而不是输出端口。以下是添加一条流的配置示例：

```
sudo ovs-ofctl -O Openflow13 add-flow s1  
dl_type=0x0806, dl_src=SRC_MAC, dl_dst=DST_MAC, actions=group:60169
```

14 LISP流映射使用指南

概述

定位器/ID分离协议

定位器/ID分离协议(LISP)技术提供了一个灵活的map-and-encap框架，可用于覆盖网络应用，如数据中心网络虚拟化和网络功能虚拟化(NFV)。

LISP提供以下的命名空间：

- Endpoint Identifiers (EIDs)
- Routing Locators (RLOCs)

在虚拟环境中EIDs可以被看作是虚拟地址空间，RLOCs可以被看作是物理网络地址空间。

LISP框架为网络控制平面与转发平面分离提供了：

- 数据平面指定如何从底层物理网络地址封装虚拟网络地址。
- 控制平面存储"虚拟到物理"的地址空间映射、相关转发策略并提供这些信息给数据平面。

网络可编程的实现是通过编写转发策略，如透明迁移、服务链和映射系统中的流量工程等；在新流到达时，数据平面元素可以获取这些策略。这章描述了OpenDaylight中LISP流映射项目以及如何使用它完成先进的SDN和NFV用例。

LISP数据平面隧道路由器在以下平台的开源社区中LISPMob.org上得到：

- Linux
- Android
- OpenWRT

更多详情请参考网址lispmob.org。

LISP流映射服务

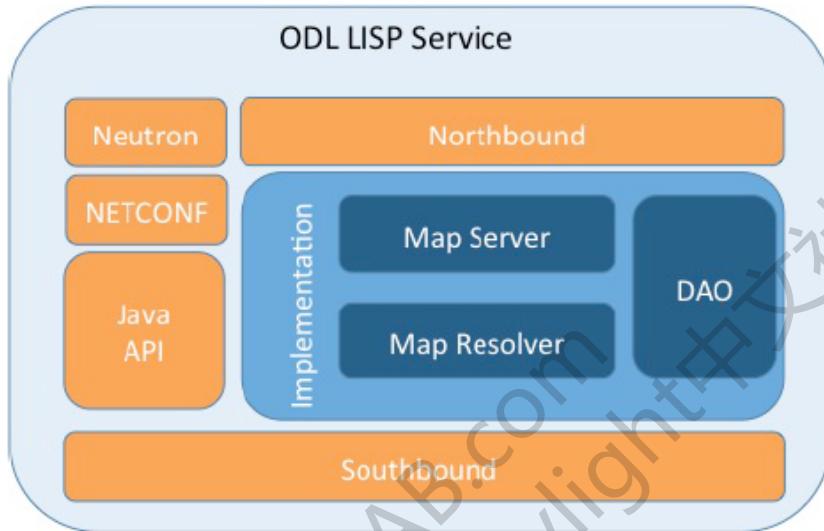
LISP流映射服务提供LISP映射系统服务，包括LISP Map-Server和LISP Map-Resolver服务来存储和将数据映射到数据平面节点以及OpenDaylight应用程序上。映射数据可以包括映射虚拟地址到物理网络地址（虚拟节点是可得到的或者主机可达的），除此以外，还包括多种路由策略，如流量工程和负载均衡等。利用此服务，OpenDaylight应用和服务可以使用北向REST API在LISP映射服务中定义映射和策略。数据平面设备能够允许LISP控制协议通过南向LISP插件利用这个服务。LISP-enabled设备必须配置为使用这个OpenDaylight服务，作为他们的Map-Server和/或 Map-Resolver。

南向LISP插件支持LISP控制协议（Map-Register、Map-Request和Map-Reply消息），也能用来在OpenDaylight映射服务中注册映射。

LISP流映射架构

下图展示了各种LISP流映射模块。

图14.1 LISP映射服务内部架构



对每个模块进行简要描述如下：

- **DAO (Data Access Object)**: 这一层将LISP逻辑从数据库中分离，所以我们可以从映射库特定的实现中分离map server和map resolver。目前我们有这层带有inmemory HashMap的实现，只需要实现ILispDAO接口，就可以切换到任何其他key/value存储中。
- **Map Server**: 该模块处理身份认证令牌（keys）和映射的添加和注册。更多详情请访问网站：<http://tools.ietf.org/search/rfc6830>。
- **Map Resolver**: 该模块接受和处理映射查询，提供映射给请求者。更多详情请访问网站：<http://tools.ietf.org/search/rfc6830>。
- **Northbound API**: OpenDaylight北向API的一部分。该模块能够定义key-EID，还可以通过Map Server添加映射信息。Key-EID也能通过这个API查询到。北向API还提供为EID前缀查询映射信息的功能。
- **Neutron**: 该模块实现了OpenDaylight Neutron Service APIs。提供LISP服务、OpenDaylight Neutron服务、和OpenStack间的集成。
- **NETCONF**: 该模块能够通过OpenDaylight的NETCONF插件让LISP服务和NETCONF-enabled设备交互。
- **Java API**: API模块通过一个java API暴露 Map Server和 Map Resolver。
- **LISP Southbound Plugin**: 该插件使支持LISP控制平面协议的数据平面设备通过LISP控制平面协议注册和查询到LISP Flow Mapping上的映射。

配置LISP流映射

为了使用LISP映射服务从北向或南向注册EID到RLOC映射，**keys**必须首先定义为EID前缀。一旦**key**定义好，可以用来添加映射，EID前缀可以多次使用。如果服务被用来处理来自南向LISP插件的Map-Register消息，必须使用数据平面相同的**key**为相关的EID前缀创建Map-Register中的身份验证数据。

Karaf中的etc/custom.properties文件允许一些OpenDaylight参数的配置。LISP服务有两个属性可以调整：**lisp.mappingOverwrite**和**lisp.smr**。

lisp.mappingOverwrite (默认:

true): 配置更新映射的处理，当设置为**true** (默认) (或者经由Map-Register消息的南向插件或者通过北向API PUT REST调用)，现有的与EID前缀相关的RLOC设置被覆盖。当设置为**false**，更新的RLOCs并入现有的设置。

lisp.smr (默认: **false**): 启用/禁用Solicit-Map-Request (SMR)功能。SMR是EID-to-RLOC映射为"subscribers"的通知改变方法。LISP服务将所有Map-Request的源RLOC作为请求EID前缀的订阅者，如果映射改变了，将会发送SMR控制消息给此RLOC。

lisp.elpPolicy (默认: **default**): 配置如何从一个映射构建一个Map-Reply南向消息，这个映射包含一个明确的定位路径(ELP)RLOC。它用于兼容数据平面不知道ELP LCAF格式的设备。默认设置没有改变映射，返回所有未修改的RLOC。**both**设置意为添加了一个新的RLOC给映射，比ELP的优先级低，这是服务链中的下一个跃点。为了确定下一跳，它搜索ELP中Map-Request的源RLOC，并且选择下一跳，如果不存在，选择第一跳。替换的设置使用和**both**设置相同的算法添加一个新的RLOC，但使用ELP RLOC的原优先级，该优先级已经从映射中删除了。

LISP地址格式的字符型约定

除了常见的IPv4、IPv6和MAC地址数据类型，LISP控制平面除了LISP Canoncal Address Format (LCAF)以外，支持任意IANA指定的Address Family Identifiers。

OpenDaylight中的LISP流映射项目实现了对这些不同地址格式的支持，下表中给出了完整的总结。虽然一些地址格式定义明确并且广泛使用，但是还有很多没有，所以有必要定义一个约定用于log、URL、输入字段等所有的实现地址类型。下表列出了支持的格式，以及AFI数字和LCAF类型，包括用于消除潜在重叠的前缀，已经有例子输出。

表14.1 LISP地址格式

SDNLAB.com
OpenDaylight中文社区

SDNLAB.com
OpenDaylight中文社区

名字	<i>AFI</i>	<i>LCAF</i>	前缀	文本渲染
No Address	0	-	no:	No Address Present
IPv4 Prefix	1	-	ipv4:	192.0.2.0/24
IPv6 Prefix	2	-	ipv6:	2001:db8::/32
MAC Address	16389	-	mac:	00:00:5E:00:53:00
Distinguished Name	17	-	dn:	stringAsIs
AS Number	18	-	as:	AS64500
AFI List	16387	1	list:	{ 192.0.2.1 , 192.0.2.2 ,2001:db8::1}
Instance ID	16387	2	-	[223] 192.0.2.0/24
Application Data	016387/td>	4	appdata:	192.0.2.1!128!17! 80-81!6667-7000
Explicit Locator Path	16387	10	elp:	{ 192.0.2.1#192.0.2.2 192.0.2.3#192.0.2.3 }
Source/Destination Key	16387	12	srcdst:	192.0.2.1/32 192.0.2.2/32

Pair

16387

15

kv:

192.0.2.1#192.0.2.2

请注意斜杠/通常用来分离IPv4和IPv6地址，在URL中转化为%2f。

Karaf命令

在本节我们将讨论Karaf命令中的两种类型：内置命令和LISP特殊命令。一些内置命令很有用，在教程中会用到，所以在这会讨论。所有LISP指定命令的引用，包含被LISP Flow Mapping项目添加的内容，在debugging时很有用。

可用内置命令

help: 列出所有可用的命令，包括一些简短的描述。

help <command_name>: 显示特定命令的详细信息。

feature:list [-i]: 显示所有Karaf容器中可用的功能。-i选项表示只列出目前安装的功能。可以使用| grep过滤输出流（对所有的命令，不仅仅是这个命令）。

feature:install

<feature_name>: 安装功能feature_name。

log:set <level> <class>: 设置日志级别。所有class默认的日志级别是INFO。想debugging或者学习LISP内部构建，可以运行日志，在Karaf启动以后设置

log:set TRACE org.opendaylight.lispflowmapping.

log:display: 在控制台输出日志文件，返回控制给用户。

log:tail: 持续展示日志输出，Ctrl+C返回控制台。

LISP特殊命令

lisp命令可以通过**help lisp**得到。当前的命令有：

lisp:addkey: 为IPv4 EID前缀0.0.0.0/0（所有地址）添加默认密码password。在用南向设备进行实验时该命令将会有用，使用REST接口将会很麻烦。

lisp:mappings: 显示所有存储在内部非持久性数据存储（DAO）的映射，列出完整的数据架构。输出可以用于调试。

教程

本节提供了展示服务各种功能的教程。

创建LISP覆盖

本节提供指令设置三个节点的LISP网络（一个“client”节点和两个“server”节点），使用LISPmob作为数据平面LISP节点和OpenDaylight的LISP Flow Mapping项目作为LISP网络的LISP可编程映射系统。

概述

下面将演示一个客户端和两台服务器组成的LISP网络的设置，然后演示两个“server”节点的故障转移。

前提条件

- **OpenDaylight Lithium**

- **The Postman Chrome App:**推荐使用Postman Chrome App来按照教程编辑和发送请求。该项目的git库在resources/tutorial/Lithium_Tutorial.json.postman_collection文件中集合了所有的请求。你可以通过点击顶部的import按钮将该文件放入Postman中，从link中选择Download，然后进入下面的URL：

https://git.opendaylight.org/gerrit/gitweb?p=lispflowmapping.git;a=blob_plain;f=resources/tutorial/Lithium_Tutorial.json.postman_collection;hb=refs/heads/stable/lithium。或者，你可以在你的机器上保存文件，或者你有库被检验，从库里输入也可以。如果你没有修改默认的控制器设置的话，你将需要创建新的Postman环境，并且定义一些变量包括：controllerHost设置为hostname或者ODL实例上的IP地址，restconfPort设置为8181。

- **LISPmob version 0.5.x**在写这个的时候该版本还没发布，但是在这个项目的git库里面的实验分支key看到。README.md列出了从源代码构建所需的依赖项。

- **A virtualization platform**

目标环境

假设三个LISP数据平面节点和LISP映射系统运行在Linux虚拟机上，拥有NAT模式下的eth0接口允许外部网络访问，eth1连接到只有一台主机的网络，IP地址在下表展示（如果使用别的地址的话，请调整配置文件、JSON实例等）。

表14.2实验节点

节点	节点类型	IP地址
controller	OpenDaylight	192.168.16.11
client	LISPmob	192.168.16.30
server1	LISPmob	192.168.16.31
server2	LISPmob	192.168.16.32
service-node	LISPmob	192.168.16.33

注：当教程使用LISPmob作为数据平面，可以是LISP-enabled硬件和软件路由（商业/开源）。

说明

下面的步骤使用命令行工具curl与LISP Flow Mapping RPC REST API交互。所以，你可以在页面上看到实际请求的url和body的内容。

1.在控制器虚拟机上运行OpenDaylight锂版本。请根据安装指南进行安装。一旦用Karaf命令安装odl-openflowplugin-all后：

```
feature:install odl-lispflowmapping-all
```

需要相当长一段时间来加载和初始化所有特性和它们的依赖项。在Karaf控制台运行命令log:tail看日志输出。

2.在client、server1、server2和服务-node上安装LISPmob，参考
<https://github.com/LISPmob/lispmob#software-prerequisites>。

3.配置LISPmob安装。从lispd.conf.example文件开始，在你的虚拟/LISP网络的IP地址空间中的每个lispd.conf文件中设置EID。在本教程中client的EID设置为1.1.1.1/32，server1和server2设置为2.2.2.2/32。

4.每个lispd.conf文件中的RLOC接口设置为eth1。基于这个接口，LISP将决定RLOC（对应虚拟机的IP地址）。

5.将Map-Resolver地址设置为控制器的IP地址，客户端的Map-Server也一样。在server1和server2上，手动设置Map-Server为其他的地址，它将不干扰控制器上的映射，。

6.在每个lispd.conf文件中修改"key"参数设置为你想要的key/password（本教程中是password）。

注：该项目的git库中的stable/lithium分支下的resources/tutorial目录中有本教程使用的文件
<https://git.opendaylight.org/gerrit/gitweb?>

`p=lispflowmapping.git;a=tree;f=resources/tutorial;hb=refs/heads/stable/lithium`, 可以拷贝该文件到各虚拟机的`/root/lispd.conf`中, 将在相同目录发现JSON参考文件。

7. 使用client EID (1.1.1.1/32)的RPC REST API定义一个key和EID前缀, 允许南向注册。服务器EID映射可以通过REST API配置。在控制器端运行如下命令 (或者任何可以连接到控制器的机器, localhost设置成控制器的IP地址)。

```
curl -u "admin":"admin" -H "Content-type: application/json" -X POST \
http://localhost:8181/restconf/operations/lfm-mapping-database:add-key --data @add-key.json
```

add-key.json文件的内容如下:

```
{
  "input": {
    "LispAddressContainer": {
      "Ipv4Address": {
        "afi": 1,
        "Ipv4Address": "1.1.1.1"
      }
    },
    "mask-length": 32,
    "key-type": 1,
    "authkey": "password"
  }
}
```

8. 通过应答如下的URL验证key已经正确添加:

```
curl -u "admin":"admin" -H "Content-type: application/json" -X POST \
http://localhost:8181/restconf/operations/lfm-mapping-database:get-key \
--data @get1.json
```

get1.json文件的内容可以通过移除key-type和authkey区域从add-key.json文件中生成。以上的调用会输出如下内容:

```
{"output": {"authkey": "password"}}
```

9. 在所有的虚拟机上运行lispd LISPmob进程:

```
lispd -f /root/lispd.conf
```

10. 客户端LISPmob节点应该在OpenDaylight中注册其EID-to-RLOC映射。通过REST API验证你可以查看相应的EID。

```
curl -u "admin":"admin" -H "Content-type: application/json" -X POST \
http://localhost:8181/restconf/operations/lfm-mapping-database:getmapping \
--data @get1.json
```

另一种使用南向接口查询ODL映射是使用lig开源工具。

11. 注册服务器EID 2.2.2.2/32到控制器的EID-to-RLOC映射, 指向server1和server2, server1赋予较高优先级。

```
curl -u "admin":"admin" -H "Content-type: application/json" -X POST \
http://localhost:8181/restconf/operations/lfm-mapping-database:addmapping \
--data @mapping.json
```

mapping.json文件内容如下：

```
{
  "input": {
    "recordTtl": 1440,
    "maskLength": 32,
    "authoritative": true,
    "LispAddressContainer": {
      "Ipv4Address": {
        "afi": 1,
        "Ipv4Address": "2.2.2.2"
      }
    },
    "LocatorRecord": [
      {
        "name": "server1",
        "priority": 1,
        "weight": 1,
        "multicastPriority": 255,
        "multicastWeight": 0,
        "localLocator": true,
        "rlocProbed": false,
        "routed": false,
        "LispAddressContainer": {
          "Ipv4Address": {
            "afi": 1,
            "Ipv4Address": "192.168.16.31"
          }
        }
      },
      {
        "name": "server2",
        "priority": 2,
        "weight": 1,
        "multicastPriority": 255,
        "multicastWeight": 0,
        "localLocator": true,
        "rlocProbed": false,
        "routed": false,
        "LispAddressContainer": {
          "Ipv4Address": {
            "afi": 1,
            "Ipv4Address": "192.168.16.32"
          }
        }
      }
    ]
  }
}
```

第二个RLOC([192.168.16.32](#) - server2)的优先级值是2，高于[192.168.16.31](#)的优先级值1。**server1**要先于**server2**到达EID [2.2.2.2/32](#)。注意在LISP中，值越低，优先级越高。

12.验证[2.2.2.2/32](#) EID正确注册：

```
curl -u "admin":"admin" -H "Content-type: application/json" -X POST \
http://localhost:8181/restconf/operations/lfm-mapping-database:getmapping \
--data @get2.json
```

其中通过改变Ipv4Address区域的内容，将[1.1.1.1](#)改为[2.2.2.2](#)，get2.json就可以从get1.json中生成。

13.现在LISP网络启动了。key登录客户端虚拟机进行验证，ping服务器EID:

```
ping 2.2.2.2
```

14.测试故障转移容错机制。假设在server1上有一个服务不可用，但server1本身仍然是可获取的。LISP不会自动容错，即使映射地址2.2.2.2/32有两个定位器，还是会使用高优先级的定位器。为了强加容错，需要将server2的优先级设置为一个较低的值。使用文件mapping.json，交换两个定位器的优先级值(mapping.json文件中的15-31行)重复步骤11。也可以重复步骤12观察映射是否正确注册。如果停止ping on，使用wireshark监控流，可以看到ping 2.2.2.2，流量将从server1 RLOC转移到server2 RLOC。

使用默认的OpenDaylight配置，容错应该是无缝对接的(我们观察到在最坏的情况下才失去了3个ping包)，因为LISP Solicit-Map-Request (SMR) 机制可以让LISP数据平面元素为特定的EID更新映射(默认启用)。它是由etc/custom.properties中的lisp.smr参数控制。启用时，RPC接口任何映射的改变都会触发一个SMR包给所有最后15分钟请求映射的数据平面元素。如果禁用，ITRs保持映射直到Map-Reply中的TTL到期。

15.添加一个服务链到从客户端到服务器的路径上，我们可以使用显示的定位路径，指定service-node作为第一跳，server1 (或者server2) 作为第二跳。下面将实现：

```
curl -u "admin":"admin" -H "Content-type: application/json" -X POST \
http://localhost:8181/restconf/operations/lfm-mapping-database:addmapping \
--data @elp.json
```

elp.json文件如下：

```
{
  "input": {
    "recordTtl": 1440,
    "maskLength": 32,
    "authoritative": true,
    "LispAddressContainer": {
      "Ipv4Address": {
        "afi": 1,
        "Ipv4Address": "2.2.2.2"
      }
    },
    "LocatorRecord": [
      {
        "name": "ELP",
        "priority": 1,
        "weight": 1,
        "multicastPriority": 255,
        "multicastWeight": 0,
        "localLocator": true,
        "rlocProbed": false,
        "routed": false,
        "LispAddressContainer": {
          "LcafTrafficEngineeringAddr": {
            "afi": 16387,
            "lcafType": 10,
            "Hops": [
              {
                "name": "service-node",
                "lookup": false,
                "RLOCProbe": false,
                "strict": true,
                "hop": {
                  "Ipv4Address": {
                    "afi": 1,
```

根据以上配置，[2.2.2.2/32](#)映射更新好以后，客户端到server1的ICMP流量将流经service-node。可以在LISPmob日志中得到确认，或者通过嗅探service-node或server1中的流来确认。注意，服务链是单向的，除非添加另一个ELP映射用于数据流的返回。数据包直接从server1发送到客户端。

16. 假设service-node是一个防火墙，数据流用于支持访问控制列表（ACLs）。在本教程中，可以使用service-node中的iptables防火墙规则进行仿真。想要阻止上面定义的数据流通过，可以添加以下命令：

```
iptables -A OUTPUT --dst 192.168.16.31 -j DROP
```

客户端的ping现在应该停止通信了。

在这个例子中，ACL在目的RLOC中完成。在LISPmob社区中，在EID上正在做一项工作是进行过滤，这是一个更加符合逻辑部署ACLs的地方。

17. 在服务链上删除规则并且恢复连接，通过以下命令删除ACL：

```
iptables -D OUTPUT --dst 192.168.16.31 -j DROP
```

LISP流映射支持

要想支持lispflowmapping项目，可以邮件联系开发者：lispflowmapping-dev@lists.opendaylight.org，或者在opendaylight-lispflowmapping IRC channel上。

更多信息参见Lisp Flow Mapping
wiki。https://wiki.opendaylight.org/view/OpenDaylight_Lisp_Flow_Mapping:Main

15 网络意图组成 (NIC) 用户指南

概述

网络意图组成 (NIC) 是一个允许客户端在implementation-neutral表格中传递预期状态的一个端口，主要在

OpenDaylight系统控制下通过可用资源的修改来实施。抽象为一个端口，主要包含网络服务、虚拟设备、存储等等。

intent端口意思是一个与控制无关的端口，因此可以跨组件使用，如：OpenDaylight和ONOS。另外，一个**intent**规范不应该包含实施或者技术特性。

随着本地实施规则、策略和/或设置驱动的实施规范，**intent**规范通过分解**intent**和增加**intent**来实施。

NIC架构

NIC架构的核心是**intent**模型，指明预期状态的详细信息。NIC实施主要负责将预期状态转变为OpenDaylight控制下的资源。将**intent**转变为实现的组件典型的是作为一个渲染器（renderer）。

Lithium版本中，不支持多个、同步的渲染器。VTN或者GBP渲染功能能被安装，但是不能同时安装。

且Lithium版本中，唯一支持的**actions**是“ALLOW”和“BLOCK”。“ALLOW”动作表明流量能够在源和目的端点间流动，“BLOCK”禁止流动。随着额外的**actions**，提供的实施会增加可用的**actions**是可能的。

除了将预期状态转换为一个真实状态，渲染器也主要负责更新OpenDaylight中NIC数据模型中的可选状态树来影响渲染器实施的**intent**。

配置NIC

Lithium版本中，没有默认的渲染器，另外，如果没有安装附加的模块，NIC将无法正常工作。

配置或管理NIC

没有额外的与Network Intent Composition功能相关的管理能力的配置。

交互

用户能够通过RESTful接口使用标准的RESTCONF操作和语法或者通过Karaf控制命令行，与NIC交互。

REST

配置

NIC功能针对配置数据存储支持以下REST操作：

POST：在配置存储中创建一个**intent**新实例，触发这个**intent**的实现。一个ID作为这个**intent**的属性被规范为这个请求的一部分。

PUT：在配置存储中创建或更新**intent**的新实例，触发这个**intent**的实现。

GET：获取所有已配置的**intents**的列表或特定的已配置的**intent**。

DELETE：在配置存储中删除一个已配置的**intent**，触发删除网络中的**intent**。

Operational

NIC功能针对可选数据存储支持以下REST操作：

GET: 获取所有可选的intents的列表或特定的可选intent。

Karaf控制命令行CLI

能够通过使用Karaf 控制命令行CLI操作intents。以下的Karaf控制命令行命令是可用的：

intent:add : 创建一个新的intent

intent:update : 更新一个现有的intent

intent:list: 在系统中列出所有的intent

intent:show : 显示特定intent的详细信息

intent:delete : 从系统中删除一个intent

16 ODL-SDNi用户指南

介绍

用户指南帮助通过使用Lithium版本建立ODL-SDNi应用和包含使用ODL-BGPCEP的实例配置。

组件

SDNiAggregator（控制器）、SDNi REST API(控制器)和SDNiWrapper(bgpcep)是ODL-SDNi应用中的三个组件。

SDNiAggregator: 连接控制器的交换机、拓扑、主机跟踪管理来获取拓扑和其他相关数据。

SDNi REST API: 控制器北向的一部分，通过队列SDNiAggregator提供要求的信息。

SDNiWrapper: 这个组件使用SDNi REST API和采集需要的信息在控制器之间共享。

故障排除

为了使用多个控制器工作，需要在config.ini文件中更改一些配置，如：

在/root/controller/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipackage/opendaylight/configuration/config.ini (如of.listenPort=6653)文件中改变一个控制器的监听端口为6653，另一个控制器的监听端口为6663。

OpenFlow相关系统参数：控制器正在监听的TCP端口（默认6633）， of.listenPort=6653。

17 OpFlex ovs代理用户指南

简介

ovs代理是一个适用于OVS的策略代理，用本地连接的虚拟机和容器实现一个基于组策略的网络模型。策略代理旨在和OpenStack这样的编排工具集成。

代理配置

用配置文件进行代理配置，配置文件默认路径是"/etc/opflex-agent-ovs/opflex-agent-ovs.conf"。

以下就是一个配置文件示例，列出了可选项：

```
{  
    // Logging configuration  
    // "log": {  
    //     "level": "info"  
    // },  
    // Configuration related to the OpFlex protocol  
    "opflex": {  
        // The policy domain for this agent.  
        "domain": "openstack",  
        // The unique name in the policy domain for this agent.  
        "name": "example-agent",  
        // a list of peers to connect to, by hostname and port. One  
        // peer, or an anycast pseudo-peer, is sufficient to bootstrap  
        // the connection without needing an exhaustive list of all  
        // peers.  
        "peers": [  
            // EXAMPLE:  
            {"hostname": "10.0.0.30", "port": 8009}  
        ],  
        "ssl": {  
            // SSL mode. Possible values:  
            // disabled: communicate without encryption  
            // encrypted: encrypt but do not verify peers  
            // secure: encrypt and verify peer certificates  
            "mode": "disabled",  
            // The path to a directory containing trusted certificate  
            // authority public certificates, or a file containing a  
            // specific CA certificate.  
            "ca-store": "/etc/ssl/certs/",  
        },  
        "inspector": {  
            // Enable the MODB inspector service, which allows  
            // inspecting the state of the managed object database.  
            // Default: enabled  
            "enabled": true,  
            // Listen on the specified socket for the inspector  
            // Default /var/run/opflex-agent-ovs-inspect.sock  
            "socket-name": "/var/run/opflex-agent-ovs-inspect.sock"  
        }  
    },  
    // Endpoint sources provide metadata about local endpoints  
    "endpoint-sources": {  
        // Filesystem path to monitor for endpoint information  
        "filesystem": ["/var/lib/opflex-agent-ovs/endpoints"]  
    },  
    // Renderers enforce policy obtained via OpFlex.  
    "renderers": {  
        // Stitched-mode renderer for interoperating with a  
        // hardware fabric such as ACI  
        // EXAMPLE:  
        "stitched-mode": {  
            "ovs-bridge-name": "br0",  
            // Set encapsulation type. Must set either vxlan or vlan.  
            "encap": {  
                // Encapsulate traffic with VXLAN.  
                "vxlan": {  
                    // VXLAN options  
                }  
            }  
        }  
    }  
}
```

```
// The name of the tunnel interface in OVS
"encap-iface": "br0_vxlan0",
// The name of the interface whose IP should be used
// as the source IP in encapsulated traffic.
"uplink-iface": "eth0.4093",
// The vlan tag, if any, used on the uplink interface.
// Set to zero or omit if the uplink is untagged.
"uplink-vlan": 4093,
// The IP address used for the destination IP in
// the encapsulated traffic. This should be an
// anycast IP address understood by the upstream
// stiched-mode fabric.
"remote-ip": "10.0.0.32",
// UDP port number of the encapsulated traffic.
"remote-port": 8472
}
// Encapsulate traffic with a locally-significant VLAN
// tag
// EXAMPLE:
// "vlan" : {
// // The name of the uplink interface in OVS
// "encap-iface": "team0"
// }
},
// Configure forwarding policy
"forwarding": {
// Configure the virtual distributed router
"virtual-router": {
// Enable virtual distributed router. Set to true
// to enable or false to disable. Default true.
"enabled": true,
// Override MAC address for virtual router.
// Default is "00:22:bd:f8:19:ff"
"mac": "00:22:bd:f8:19:ff",
// Configure IPv6-related settings for the virtual
// router
"ipv6" : {
// Send router advertisement messages in
// response to router solicitation requests as
// well as unsolicited advertisements. This
// is not required in stitched mode since the
// hardware router will send them.
"router-advertisement": true
}
},
// Configure virtual distributed DHCP server
"virtual-dhcp": {
// Enable virtual distributed DHCP server. Set to
// true to enable or false to disable. Default
// true.
"enabled": true,
// Override MAC address for virtual dhcp server.
// Default is "00:22:bd:f8:19:ff"
"mac": "00:22:bd:f8:19:ff"
},
"endpoint-advertisements": {
// Enable generation of periodic ARP/NDP
// advertisements for endpoints. Default true.
"enabled": "true"
}
},
// Location to store cached IDs for managing flow state
```

```
"flowid-cache-dir": "/var/lib/opflex-agent-ovs/ids"
}
}
}
```

终端注册

代理利用终端元数据文件学习终端，终端元数据文件默认路径为："/var/lib/opflex-agent-ovs/endpoints"。

JSON格式文件如下所示：

```
{
  "uuid": "83f18f0b-80f7-46e2-b06c-4d9487b0c754",
  "policy-space-name": "test",
  "endpoint-group-name": "group1",
  "interface-name": "veth0",
  "ip": [
    "10.0.0.1", "fd8f:69d8:c12c:ca62::1"
  ],
  "dhcp4": {
    "ip": "10.200.44.2",
    "prefix-len": 24,
    "routers": ["10.200.44.1"],
    "dns-servers": ["8.8.8.8", "8.8.4.4"],
    "domain": "example.com",
    "static-routes": [
      {
        "dest": "169.254.169.0",
        "dest-prefix": 24,
        "next-hop": "10.0.0.1"
      }
    ]
  },
  "dhcp6": {
    "dns-servers": ["2001:4860:4860::8888", "2001:4860:4860::8844"],
    "search-list": ["test1.example.com", "example.com"]
  },
  "ip-address-mapping": [
    {
      "uuid": "91c5b217-d244-432c-922d-533c6c036ab4",
      "floating-ip": "5.5.5.1",
      "mapped-ip": "10.0.0.1",
      "policy-space-name": "common",
      "endpoint-group-name": "nat-epg"
    },
    {
      "uuid": "22bfd01-a390-4b6f-9b10-624d4ccb957b",
      "floating-ip": "fdf1:9f86:d1af:6cc9::1",
      "mapped-ip": "fd8f:69d8:c12c:ca62::1",
      "policy-space-name": "common",
      "endpoint-group-name": "nat-epg"
    }
  ],
  "mac": "00:00:00:00:00:01",
  "promiscuous-mode": false
}
```

文档中部分参数解释：

uuid: 终端唯一的ID

endpoint-group-name: 终端组的名称

policy-space-name: 终端组策略空间的名称

interface-name: 连接终端的OVS接口的名称

ip: 一串字符，包含终端可以使用的IPv4或IPv6地址

mac: 终端接口的MAC地址

promiscuous-mode: 允许来自该虚拟机的流量绕过默认安全端口

dhcp4: 一个分布式DHCPv4配置块（见下）

dhcp6: 一个分布式DHCPv6配置块（见下）

ip-address-mapping: IP地址映射配置块（见下）

DHCPv4配置块包含以下参数：

ip: IP地址随着DHCP返回，必须是一个配置好的IPv4地址

prefix: 子网前缀长度

routers: 终端默认网关列表

dns: DNS服务器地址列表

domain: DHCP响应中发送的域名参数

static-routes: 静态路由配置块列表，包含发送给主机的静态路由参数"dest"、"destprefix"和"next-hop"。

DHCPv6配置块包含以下参数：

dns: 终端DNS服务器列表

search-patch: 终端DNS路径搜寻

ip-address-mapping配置块包含以下参数：

uuid: 创建映射的虚拟终端的唯一ID

floating-ip: 使用DNAT映射到浮动IPv4或IPv6地址

mapped-ip: 源IPv4或IPv6地址，必须是分配给终端的IPs之一

endpoint-group-name: NATed IP终端组的名称

policy-space-name: NATed IP策略空间的名称

Inspector

Opflex inspector是一个有效的命令行工具，用来查看被管理的对象数据库，有助于代理进行调试和诊断。

命令名称是“gbp_inspect”，采用以下参数：

```
# gbp_inspect -h
Usage: ./gbp_inspect [options]
Allowed options:
```

```
-h [ --help ] Print this help message
--log arg Log to the specified file (default
standard out)
--level arg (=warning) Use the specified log level (default
info)
--syslog Log to syslog instead of file or
standard out
--socket arg (=/usr/local/var/run/opflex-agent-ovs-inspect.sock)
Connect to the specified UNIX domain
socket (default /usr/local/var/run/
opfl
ex-agent-ovs-inspect.sock)
-q [ --query ] arg Query for a specific object with
subjectname,uri or all objects of a
specific type with subjectname
-r [ --recursive ] Retrieve the whole subtree for each
returned object
-f [ --follow-refs ] Follow references in returned objects
--load arg Load managed objects from the
specified
file into the MODB view
-o [ --output ] arg Output the results to the specified
file (default standard out)
-t [ --type ] arg (=tree) Specify the output format: tree, list,
or dump (default tree)
-p [ --props ] Include object properties in output
```

以下是使用该工具的几个例子。

可以使用一个或多个查询命令获取运行系统的信息，命令由一个对象模型的**class**名称和某个对象的**URL**组成。简单的查询获取单一的对象：

```
# gbp_inspect -q DmtreeRoot
--* DmtreeRoot, /
# gbp_inspect -q GbpEpGroup
--* GbpEpGroup, /PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
--* GbpEpGroup, /PolicyUniverse/PolicySpace/test/GbpEpGroup/group1/
# gbp_inspect -q GbpEpGroup, /PolicyUniverse/PolicySpace/common/GbpEpGroup/natepg/
--* GbpEpGroup, /PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
```

你也可以显示出某个对象所有的性质：

```
# gbp_inspect -p -q GbpeL24Classifier
--* GbpeL24Classifier, /PolicyUniverse/PolicySpace/test/GbpeL24Classifier/
classifier4/
{
connectionTracking : 1 (reflexive)
dFromPort : 80
dToPort : 80
etherT : 2048 (ipv4)
name : classifier4
prot : 6
}
--* GbpeL24Classifier, /PolicyUniverse/PolicySpace/test/GbpeL24Classifier/
classifier3/
{
etherT : 34525 (ipv6)
name : classifier3
order : 100
prot : 58
}
```

```
--* GbpeL24Classifier, /PolicyUniverse/PolicySpace/test/GbpeL24Classifier/
classifier2/
{
etherT : 2048 (ipv4)
name : classifier2
order : 101
prot : 1
}
```

你也可以请求去获取你所查询的对象的所有**children**:

```
# gbp_inspect -r -q GbpEpGroup, /PolicyUniverse/PolicySpace/common/GbpEpGroup/
nat-epg/
--* GbpEpGroup, /PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
|-* GbpeInstContext, /PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
GbpeInstContext/
`-* GbpEpGroupToNetworkRSrc, /PolicyUniverse/PolicySpace/common/GbpEpGroup/
nat-epg/GbpEpGroupToNetworkRSrc/
```

你也可以按照下载对象的参考文档:

```
# gbp_inspect -fr -q GbpEpGroup, /PolicyUniverse/PolicySpace/common/GbpEpGroup/
nat-epg/
--* GbpEpGroup, /PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
|-* GbpeInstContext, /PolicyUniverse/PolicySpace/common/GbpEpGroup/nat-epg/
GbpeInstContext/
`-* GbpEpGroupToNetworkRSrc, /PolicyUniverse/PolicySpace/common/GbpEpGroup/
nat-epg/GbpEpGroupToNetworkRSrc/
--* GbpFloodDomain, /PolicyUniverse/PolicySpace/common/GbpFloodDomain/fd_ext/
`-* GbpFloodDomainToNetworkRSrc, /PolicyUniverse/PolicySpace/common/
GbpFloodDomain/fd_ext/GbpFloodDomainToNetworkRSrc/
--* GbpBridgeDomain, /PolicyUniverse/PolicySpace/common/GbpBridgeDomain/bd_ext/
`-* GbpBridgeDomainToNetworkRSrc, /PolicyUniverse/PolicySpace/common/
GbpBridgeDomain/bd_ext/GbpBridgeDomainToNetworkRSrc/
--* GbpRoutingDomain, /PolicyUniverse/PolicySpace/common/GbpRoutingDomain/
rd_ext/
|-* GbpRoutingDomainToIntSubnetsRSrc, /PolicyUniverse/PolicySpace/common/
GbpRoutingDomain/rd_ext/GbpRoutingDomainToIntSubnetsRSrc/122/%2fPolicyUniverse
%2fPolicySpace%2fcommon%2fGbpSubnets%2fsubnets_ext%2f/
`-* GbpForwardingBehavioralGroupToSubnetsRSrc, /PolicyUniverse/PolicySpace/
common/GbpRoutingDomain/rd_ext/GbpForwardingBehavioralGroupToSubnetsRSrc/
--* GbpSubnets, /PolicyUniverse/PolicySpace/common/GbpSubnets/subnets_ext/
|-* GbpSubnet, /PolicyUniverse/PolicySpace/common/GbpSubnets/subnets_ext/
GbpSubnet/subnet_ext4/
`-* GbpSubnet, /PolicyUniverse/PolicySpace/common/GbpSubnets/subnets_ext/
GbpSubnet/subnet_ext6/
```

18 PCEP用户指南

概述

OpenDaylight karaf预配置了一个基本的PCEP配置。

32-pcep.xml (基本PCEP配置, 包括会话参数)

39-pcep-provider.xml (PCEP provider的配置)

配置PCEP

默认配置是在0.0.0.0: 4189启动PCE服务器，可以修改39-pcep-provider.xml来修改默认配置：

```
<module>
<type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:pcep:topology:provider">prefix:pcep-topology-
provider</type>
<name>pcep-topology</name>
<listen-address>192.168.122.55</listen-address>
<listen-port>4189</listen-port>
...
</module>
```

listen-address: PCE启动和监听的地址

listen-port: 启动和监听地址的端口

PCEP默认配置符合状态PCEP拓展：

draft-ietf-pce-stateful-pce: 在版本02和07中

PCEP部分路由

符合draft-eitf-pce-segment-routing: 部分路由的PCEP拓展

配置文件位于：etc/opendaylight/karaf。

33-pcep-segment-routing.xml: 不需要编辑该文件

19 PCMM用户指南

系统概述

这些组件使用PCMM协议进行DOCSIS QoS Gates管理。驱动组件负责PCMM/COPS/PDP功能，处理来自PacketCable Provider和FlowManager请求。请求消息转换为PCMM Gate控制信息并通过COPS传输到CMTS。CableLabs规范定义了连接PCMM/COPS/PDP功能的插件，PacketCable解决方案是一个兼容MDSAL的组件。

Packetcable组件（PCMM QoS Gates）

packetcable包含三个OpenDaylight bundle

Bundle	Description
packetcable-policy-server	基于CMTS架构和COPS协议实现 PCMM模型的插件
packetcable-policy-model	这个模型提供了到底层CMTS的QoS Gates的直接映射
packetcable-driver	编码译码器负责为流和CMTS连接把模型转换成合适的 PCMM Gate消息

安装功能组件

```
opendaylight-user@root>feature:install odl-restconf odl-l2switch-switch odldlux-core odl-mdsal-apidocs odl-packetcable-policy-all
```

探究学习PacketCable REST API

<http://localhost:8181/apidoc/explorer/index.html>

向OpenDaylight Inventory添加一个CMTS

RESTCONF url使得可以向OpenDaylight添加一个CMTS并且进行连接。

The screenshot shows the Postman interface for interacting with the OpenDaylight Inventory API. It displays two requests:

- GET /config/opendaylight-inventory:nodes/node/{id}/packetable-cmsts:cmts-node/**: This request retrieves the configuration of a CMTS node. The response schema is defined as follows:


```
{
  "packetable-cmsts:port": "integer"
}
```

 The response content type is set to application/json. A parameter 'id' is specified with a value of 1, which is described as the unique identifier for the node.
- PUT /config/opendaylight-inventory:nodes/node/{id}/packetable-cmsts:cmts-node/**: This request updates the configuration of a CMTS node. The body of the request contains the following JSON:


```
{
  "cmts-node": {
    "cmts-node": [
      {
        "ipmtu_size": [
          "address": "10.200.90.3",
          "port": 3918
        ]
      }
    ]
}
```

 The response content type is application/json. The request URL is set to `http://localhost:8181/restconf/config/opendaylight-inventory:nodes/1/packetable-cmsts:cmts-node/`. The response body is labeled 'no content'. The response code is 200.

figure5.1 Add a CMTS to OpenDaylight Inventory

Postman

Configure the Chrome browser ([链接](#))

Download and import sample packetable collection ([链接](#))

Postman

The screenshot shows the Postman interface with the 'Collection' tab selected. The left sidebar lists various API endpoints for the 'packetable-cmsts' collection, including methods like GET, POST, PUT, and DELETE, along with their descriptions. The right panel shows a specific request for creating a CMTS node:

- Method:** PUT
- URL:** `http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/{id}/packetable-cmsts:cmts-node/`
- Body:** (JSON)


```
{
  "cmts-node": {
    "cmts-node": [
      {
        "ipmtu_size": [
          "address": "10.200.90.3",
          "port": 3918
        ]
      }
    ]
}
```
- Status:** 200 OK

Operations.

figure6.1 Postman Operations

20 Service Function Chaining

OpenDaylight功能服务链（SFC）概述

OpenDaylight功能服务链（SFC）提供了定义有序网络服务（例如防火墙、负载均衡）列表的功能。这些网络服务在网络中被“缝”到一起创建一个服务链。这个工程提供了一个基础架构（包含链接逻辑、APIs），需要ODL提供网络服务链和一个定义这些服务链的终端用户应用。

首字母缩写列表

ACE - Access Control Entry

ACL - Access Control List

SCF - Service Classifier Function

SF - Service Function

SFC - Service Function Chain

SFF - Service Function Forwarder

SFG - Service Function Group

SFP - Service Function Path

RSP - Rendered Service Path

NSH - Network Service Header

SFC用户接口

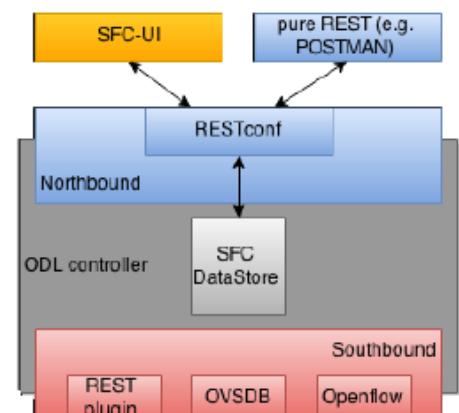
概述

SFC用户接口（SFC-UI）是一个基于DLUX的工程。提供了一种简便的方式对Datastore中的配置进行创建、读取、更新和删除操作。而且，可以展示所有SFC特性（例如安装、卸载）的状态和Karaf的日志信息。

SFC-UI架构

通过RESTCONF对SFC-UI进行操作

图 20.1 集成在ODL中的SFC-UI



配置SFC-UI

配置步骤

1. 运行ODL（运行karaf）
2. 在karaf控制台执行命令：feature:install odl-sfc-ui
3. 浏览SFC-UI：http://:8181/sfc/index.html

SFC南向REST插件

概述

南向REST插件用于从Datastore向网络设备发送配置信息以提供REST API（也就是说他们有一个配置的REST URL）。提供POST/PUT/DELETE操作，这些变更都是在SFC的Data Store中。

在现行状态下，都将通过下面这些SFC的**Data Store**中进行更改

Access Control List (ACL)

Service Classifier Function (SCF)

Service Function (SF)

Service Function Group (SFG)

Service Function Schedule Type (SFST)

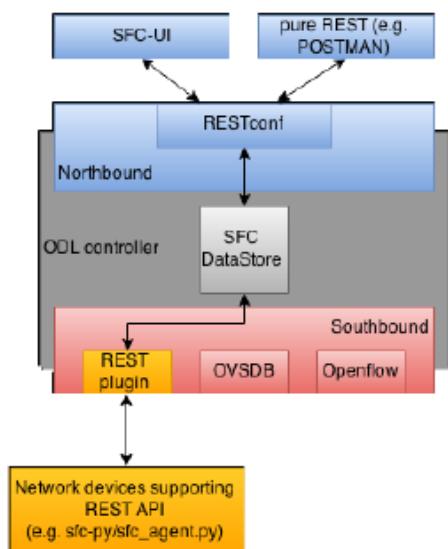
Service Function Forwader (SFF)

Rendered Service Path (RSP)

南向REST插件架构

从用户的视角看，REST插件是另一个用于同网络设备通信的SFC南向插件。

图 20.2 集成在ODL中的南向REST插件



配置南向REST插件

配置步骤

1. 运行ODL（运行karaf）
2. 在karaf控制台执行命令：feature:install odl-sfc-sb-rest
3. 通过SFC用户接口或RESTCONF为SF/SFF配置REST URL（配置步骤可在下面的指南中获取）

指南

怎样使用南向REST插件和如何控制网络设备可以参考如下链接：

https://wiki.opendaylight.org/view/Service_Function_Chaining:Main→SFC_101

SFC-OVS集成

概述

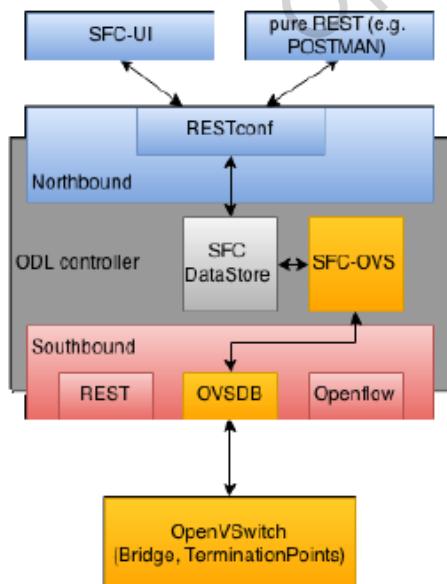
SFC-OVS提供SFC同Open vSwitch(OVS)的集成。集成通过SFC对象（像SF、SFF、Classifier等）到OVS对象（像Bridge、TerminationPoint=Port/Interface）的映射实现。在映射时注意自动实例化（setup）时对应的对象是否创建了副本。例如，当创建了新一个的SFF时，SFC-OVS插件将创建一个新的OVS Bridge。并且当创建了一个新的OVS Bridge时，SFC-OVS将创建一个新的SFF。

这个特性用于SFC用户使用Open vSwitch作为底层部署RSPs（Rendered Service Paths服务路径呈现）网络基础设施的情形。

SFC-OVS架构

SFC-OVS使用OVSDDB MD-SAL南向API从OVS设备中获取信息或者写入信息到OVS设备中。从用户的角度看，SFC-OVS扮演的角色是SFC DataStore和OVSDDB的一个中间层。

图 20.3 在ODL中集成SFC-OVS



配置SFC-OVS

配置步骤

1. 运行ODL（运行karaf）
2. 在karaf控制台执行命令：feature:install odl-sfc-ovs
3. 将ODL作为管理员配置OVS，使用如下命令：ovs-vsctl set-manager tcp::6640

指南

验证从OVS到SFF的映射

概述

这个指南将展示当OVS配置转换为对应SFC对象（这里用SFF）时通常的流程。

必备条件

已经安装好Open vSwitch（支持shell的ovs-vsctl命令）

已配置好如上所述的SFC-OVS特性

说明

shell命令

1. ovs-vsctl set-manager tcp::6640
2. ovs-vsctl add-br br1
3. ovs-vsctl add-port br1 testPort

验证

这里有两个验证SFF是否被创建的方法：

1. 访问SFC用户接口：<http://:8181/sfc/index.html#/sfc/serviceforwarder>
2. 使用RESTCONF并且发送GET请求：<http://:8181/restconf/config/service-functionforwarder:service-function-forwarders>

这里应该会有SFF，在br1最后出现并且SFF应该会包含两个数据平面信息：br1和testPort。

验证从SFF到OVS的映射

概述

这个指南将展示使用SFC API创建OVS Bridge的常用流程。

必备条件

已经安装好Open vSwitch（支持shell的ovs-vsctl命令）

已配置好如上所述的SFC-OVS特性

说明

步骤：

1. shell命令：ovs-vsctl set-manager tcp::6640
2. 发送POST请求：<http://:8181/restconf/operations/service-function-forwarder-ovs:create-ovs-bridge>

使用基础认证：“admin”,“admin”

设置Content-Type: application/json

POST请求数据段应包含如下信息：

```
{  
    "input":  
    {  
        "name": "br-test",  
        "ovs-node": {  
            "ip": "<Open_vSwitch_ip_address>"  
        }  
    }  
}
```

这里的Open_vSwitch_ip_address指安装OVS的机器IP。

认证

shell命令：ovs-vsctl show

将会出现一个名为br-test的Bridge和一个名为br-test的port/interface。

另外，OVS Bridge对应的SFF也应该被配置了，可通过上述SFC用户接口或RESTCONF进行验证。

SFC Classifier用户指南

概述

Classifier的描述详见：<https://datatracker.ietf.org/doc/draft-ietf-sfcarchitecture/>

Classifier管理从发包时的数据包监听启动到相应ip table规则的创建和移除的所有环节，相应的收包过程亦然。其提供NetfilterQueue（只在Linux上实现），过滤出匹配ip table规则的数据包。不过Classifier需要取得ROOT权限才能进行相关操作。

到目前为止，Classifier已经能够处理ACL的特征包括MAC地址、一层的端口、IPv4和IPv6、四层的TCP和UDP协议。

Classifier架构

在sfc-py/common/classifier.py文件中有该项目的Python代码。

注释

当ACL的连接Classifier成功时，假设ODL已经支持Rendered Service Path(RSP)了。

如何工作

1. sfc_agent收到了一个ACL并且pass
2. RSP(SFF的定位器)向ODL请求ACL参照
3. 如果ODL中已经有RSP，那么用于RSP的基于ACL的iptables规则将会被应用

这个流程结束后，每个数据包都将成功的与唯一的iptable规则匹配（也就是说分类成功）。而且会被封装在NSH中转发到相应的SFF（负责穿越RSP）。

通过iptables命令创建规则。如果Access Control Entry(ACE)规则是MAC地址的话iptables和ip6tables都执行，如果是IPv4地址则只执行iptables，IPv6地址就ip6tables

配置Classifier

Classifier不需要任何配置。它只需要NetfilterQueue不被其他进程占用并且对于相应的Classifier是可用的。

管理Classifier

Classifier需要搭配sfc_agent运行，下面是启动命令：

```
sudo python3.4 sfc-py/sfc_agent.py --rest --odl-ip-port localhost:8181 --autosff-name --nfq-class
```

SFC OpenFlow Layer 2 Renderer用户指南

概述

SFC OpenFlow Layer 2 Renderer (SFCOFL2)在OpenFlow交换机上对Service Chaining进行了实现。它对Rendered Service Path (RSP)的创建事件进行监听，并且一旦接收到事件，就对运行在支持OpenFlow的交换机上的Service Function Forwarders (SFF)进行编码以控制数据包能够通过service chain。

通用的缩写列表

SF - Service Function

SFF - Service Function Forwarder

SFC - Service Function Chain

SFP - Service Function Path

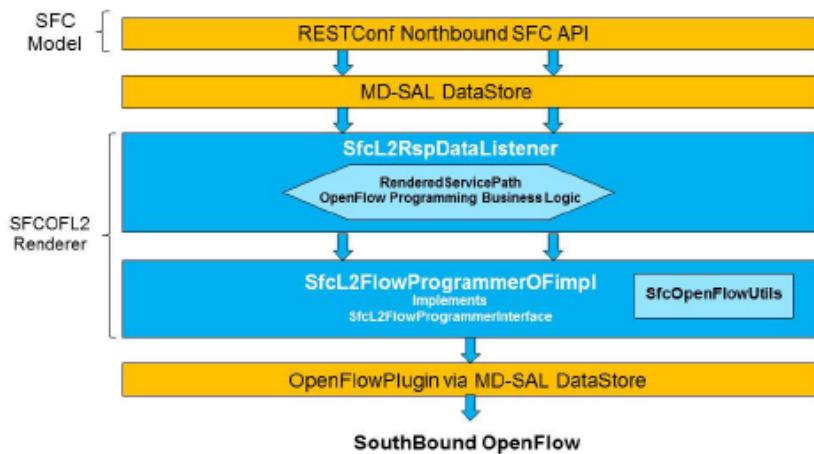
RSP - Rendered Service Path

SFC OpenFlow Renderer架构

在使用SfcL2RspDataListener的MD-SAL监听者创建RSP之后会调用SFCOFL2。在SFCOFL2初始化后，SfcL2RspDataListener会进行注册以监听RSP的变化。SFCOFL2被调用后，SfcL2RspDataListener会继续RSP进程，并且调用SfcL2FlowProgrammerOImpl创建Service Function Forwarders（在RSP中配置）中所需的流。下图中有详细的解释。

图 20.4 SFC OpenFlow Renderer高级架构

SFCOFL2 Architecture



SFC OpenFlow交换机流传输路径

SFC OpenFlow Renderer使用如下的表标识流的传输路径：

Table 0, Transport Ingress

Table 1, Path Mapper

Table 2, Next Hop

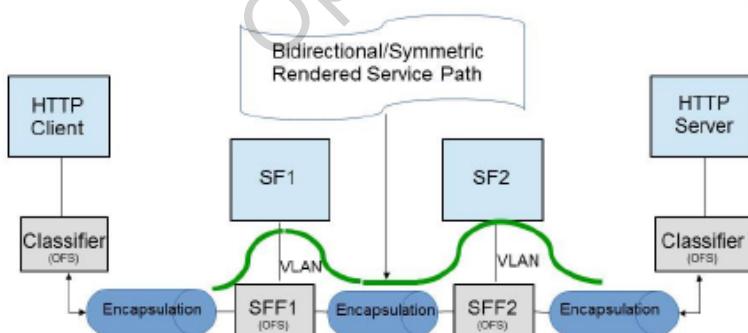
Table 10, Transport Egress

OpenFlow Table Pipeline就所有不同的SFC封装而言规定为一般的工作。

所有table的内容将在下面内容中做详细解释。SFF（包括SFF1和SFF2）、SF（SF1）和table的拓扑都在下面做解释。

图 20.5 SFC OpenFlow Renderer典型网络拓扑

SFCOFL2 Typical Network topology



```
Simple mininet command to create topology
sudo mn --topo linear,4 --mac --switch ovsk --protocols=OpenFlow13 --controller remote,ip=192.168.1.103
```

Transport Ingress Table详细

Transport Ingress table有一个预期隧道传输类型的条目被特殊的SFF接收，在SFC配置中创建。

这里是SFF1上的一个例子，假设SFF-SF使用了VLAN，并且RSP的隧道是MPLS：

表 20.1 Table Transport Ingress

Priority	Match	Action
256	EtherType==0x8847 (MPLS unicast)	Goto Table 1
256	EtherType==0x8100 (VLAN)	Goto Table 1
5	Match Any	Drop

Path Mapper Table 详细

Transport Ingress table有一个预期隧道传输信息的条目被特殊的SFF接收，在SFC配置中创建。这个隧道传输信息用于确定RSP Path ID，并且存储在OpenFlow Metadata中。

SF节点支持隧道，IP头的DSCP字段用于存储RSP Path ID。RSP Path ID写入DSCP字段，存储在用于匹配发送到SF的数据包的Transport Egress table中。

这里有一个SFF1的例子，假设有下面的细节：

VLAN ID 1000用于SFF-SF

RSP Path 1隧道使用MPLS label 100做输入，label 101做输出

RSP Path 2(同步下行路径)使用MPLS label 101做输入，label 100做输出

表 20.2 Table Path Mapper

<i>Priority</i>	<i>Match</i>	<i>Action</i>
256	MPLS Label==100	RSP Path=1, Pop MPLS, Goto Table 2
256	MPLS Label==101	RSP Path=2, Pop MPLS, Goto Table 2
256	VLAN ID==1000, IP DSCP==1	RSP Path=1, Pop VLAN, Goto Table 2
256	VLAN ID==1000, IP DSCP==2	RSP Path=2, Pop VLAN, Goto Table 2
5	Match Any	Drop

Next Hop Table详细

Next Hop Table用RSP Path ID和源MAC地址来确定目的MAC地址。

这里是一个SFF1上的例子，假设SFF1连接上SFF2，RSP Path 1入口数据包来自另外的SFC，这里不包含源MAC地址。

<i>Priority</i>	<i>Match</i>	<i>Action</i>
256	RSP Path==1, MacSrc==SF1	MacDst=SFF2, Goto Table 10
256	RSP Path==2, MacSrc==SF1	Goto Table 10
256	RSP Path==2, MacSrc==SFF2	MacDst=SF1, Goto Table 10
256	RSP Path==1	MacDst=SF1, Goto Table 10
5	Match Any	Drop

Transport Egress Table详细

Transport Egress table准备输出隧道信息并且向外发包。

这里有一个SFF1的例子，假设SFF-SF使用VLAN，并且RSP的隧道是MPLS:

表 20.4 Transport Egress table

Priority	Match	Action
256	RSP Path==1, MacDst==SF1	Push VLAN ID 1000, Port=SF1
256	RSP Path==1, MacDst==SFF2	Push MPLS Label 101, Port=SFF2
256	RSP Path==2, MacDst==SF1	Push VLAN ID 1000, Port=SF1
256	RSP Path==2	Push MPLS Label 100, Port=Ingress
5	Match Any	Drop

Administering SFCOFL2

使用SFC OpenFlow Renderer Karaf，至少下面的Karaf特性需要安装

```
odl-openflowplugin-all  
odl-sfc-core (includes odl-sfc-provider and odl-sfc-model)  
odl-sfcfl2  
odl-sfc-ui (optional)
```

下面的命令用于查看当前安装了那些feature

```
opendaylight-user@root>feature:list -i
```

或者可以使用管道看更精确匹配的feature

```
opendaylight-user@root>feature:list -i | grep sfc
```

安装特殊的feature使用Karaf的feature:install命令。

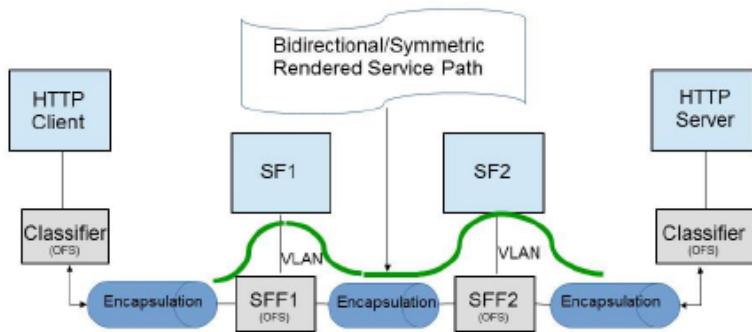
SFCOFL2指南

概述

下面的网络拓扑图展示了如何配置SFC去创建一个Service Chain。

图 20.6 SFC OpenFlow Renderer详细网络拓扑

SFCOFL2 Typical Network topology



```
Simple mininet command to create topology
sudo mn --topo linear,4 --mac --switch ovsk,protocols=OpenFlow13 --controller remote;ip=192.168.1.103
```

必备条件

在这个例子中，SFF OpenFlow交换机必须像上图中一样连接。另外，SF必须和SFF连接。

目标环境

目标环境并不重要，但是这个例子只能在Linux环境中使用。

使用说明

参考配置步骤如下所示。有多种方式发送配置，下列配置章节中会将合适的curl命令也展示出来，包括URL。

配置SFCOFL2的步骤如下：

- 1、发送SF RESTCONF配置
- 2、发送SFF RESTCONF配置
- 3、发送SFC RESTCONF配置
- 4、发送SFP RESTCONF配置

成功创建配置后，就可以通过SFC UI或RESTCONF查询服务通道。需要注意的是RSP是对称的，所以接下来创建2个RSP: `sfc-path1`和`sfc-path1-Reverse`。

此时服务链已经创建了，就可以通过服务链对OpenFlow交换机进行编程来控制流量了，可以将流量从一个客户端注入到服务链。为了调试问题，可以通过以下命令让OpenFlow流表停用，假设SFF1简称s1而SFF2简称s2：

```
sudo ovs-ofctl -0 OpenFlow13 dump-flows s1
sudo ovs-ofctl -0 OpenFlow13 dump-flows s2
```

在下面所有配置章节中，用核实的JSON配置替代\$ {json} 字符串。并且，修改URL中主机的destination。

服务功能配置

用以下命令进行服务功能配置：

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache"
```

```
--data '${JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/
config/service-function:service-functions/
```

SF configuration JSON.

```
{
  "service-functions": {
    "service-function": [
      {
        "name": "sf1",
        "type": "service-function-type:http-header-enrichment",
        "nsh-aware": false,
        "ip-mgmt-address": "10.0.0.2",
        "sf-data-plane-locator": [
          {
            "name": "sf1-sff1",
            "mac": "00:00:08:01:02:01",
            "vlan-id": 1000,
            "transport": "service-locator:mac",
            "service-function-forwarder": "sff1"
          }
        ]
      },
      {
        "name": "sf2",
        "type": "service-function-type:firewall",
        "nsh-aware": false,
        "ip-mgmt-address": "10.0.0.3",
        "sf-data-plane-locator": [
          {
            "name": "sf2-sff2",
            "mac": "00:00:08:01:03:01",
            "vlan-id": 2000,
            "transport": "service-locator:mac",
            "service-function-forwarder": "sff2"
          }
        ]
      }
    ]
  }
}
```

服务功能转发配置

通过以下命令实现服务功能转发配置：

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache"
--data '${JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/
config/service-function-forwarder:service-function-forwarders/
```

SFF configuration JSON.

```
{
  "service-function-forwarders": {
    "service-function-forwarder": [
      {
        "name": "sff1",
        "service-node": "openflow:2",
        "sff-data-plane-locator": [

```

```
{  
  "name": "ulSff1Ingress",  
  "data-plane-locator":  
  {  
    "mpls-label": 100,  
    "transport": "service-locator:mpls"  
  },  
  "service-function-forwarder-ofs:ofs-port":  
  {  
    "mac": "11:11:11:11:11:11",  
    "port-id": "1"  
  }  
},  
,  
{  
  "name": "ulSff1ToSff2",  
  "data-plane-locator":  
  {  
    "mpls-label": 101,  
    "transport": "service-locator:mpls"  
  },  
  "service-function-forwarder-ofs:ofs-port":  
  {  
    "mac": "33:33:33:33:33:33",  
    "port-id": "2"  
  }  
}  
]  
],  
"service-function-dictionary": [  
  {  
    "name": "sf1",  
    "type": "service-function-type:http-header",  
    "sff-sf-data-plane-locator":  
    {  
      "mac": "22:22:22:22:22:22",  
      "vlan-id": 1000,  
      "transport": "service-locator:mac"  
    },  
    "service-function-forwarder-ofs:ofs-port":  
    {  
      "port-id": "3"  
    }  
  }  
]  
},  
{  
  "name": "sff2",  
  "service-node": "openflow:3",  
  "sff-data-plane-locator": [  
    {  
      "name": "ulSff2Ingress",  
      "data-plane-locator":  
      {  
        "mpls-label": 101,  
        "transport": "service-locator:mpls"  
      },  
      "service-function-forwarder-ofs:ofs-port":  
      {  
        "mac": "44:44:44:44:44:44",  
        "port-id": "1"  
      }  
    },  
    {  
      "name": "ulSff2Egress",  
      "data-plane-locator":  
      {  
        "mpls-label": 102,  
        "transport": "service-locator:mpls"  
      },  
      "service-function-forwarder-ofs:ofs-port":  
      {  
        "mac": "55:55:55:55:55:55",  
        "port-id": "2"  
      }  
    }  
  ]  
}
```

```
"name": "ulSff2Egress",
"data-plane-locator":
{
  "mpls-label": 102,
  "transport": "service-locator:mpls"
},
"service-function-forwarder-ofs:ofs-port":
{
  "mac": "66:66:66:66:66:66",
  "port-id" : "2"
}
},
],
"service-function-dictionary": [
{
  "name": "sf2",
  "type": "service-function-type:firewall",
  "sff-sf-data-plane-locator":
  {
    "mac": "55:55:55:55:55:55",
    "vlan-id": 2000,
    "transport": "service-locator:mac"
  },
  "service-function-forwarder-ofs:ofs-port":
  {
    "port-id" : "3"
  }
}
]
}
```

功能服务链配置

可以通过以下功能链配置功能服务链：

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache"  
--data '${JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/config/service-function-chain:service-function-chains/
```

SFC configuration JSON.

```
{  
  "service-function-chains": {  
    "service-function-chain": [  
      {  
        "name": "sfc-chain1",  
        "symmetric": true,  
        "sfc-service-function": [  
          {  
            "name": "hdr-enrich-abstract1",  
            "type": "service-function-type:http-header-enrichment"  
          },  
          {  
            "name": "firewall-abstract1",  
            "type": "service-function-type:firewall"  
          }  
        ]  
      }  
    ]  
  }  
}
```

```
]  
}  
}
```

功能服务链路配置

通过以下命令配置功能服务链路:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache"  
--data '${JSON}' -X PUT --user admin:admin http://localhost:8181/restconf/  
config/service-function-path:service-function-paths/
```

SFP configuration JSON.

```
{  
  "service-function-paths": {  
    "service-function-path": [  
      {  
        "name": "sfc-path1",  
        "service-chain-name": "sfc-chain1",  
        "transport-type": "service-locator:mpls",  
        "symmetric": true  
      }  
    ]  
  }  
}
```

创建**Rendered**服务链路:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --  
data '${JSON}' -X POST --user admin:admin http://localhost:8181/restconf/  
operations/rendered-service-path:create-rendered-path/
```

RSP creation JSON.

```
{  
  "input": {  
    "name": "sfc-path1",  
    "parent-service-function-path": "sfc-path1",  
    "symmetric": true  
  }  
}
```

移除**Rendered**服务链路, 用以下命令移除名为sfc-path1的**Rendered**服务链路:

```
curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --  
data '{"input": {"name": "sfc-path1" }}' -X POST --user admin:admin http://  
localhost:8181/restconf/operations/rendered-service-path:delete-rendered-path/
```

Rendered服务链路查询, 用以下命令查询所有创建的**Rendered**服务链路:

```
curl -H "Content-Type: application/json" -H "Cache-Control: no-cache" -X  
GET --user admin:admin http://localhost:8181/restconf/operational/renderedservice-  
path:rendered-service-paths/
```

功能服务调度算法

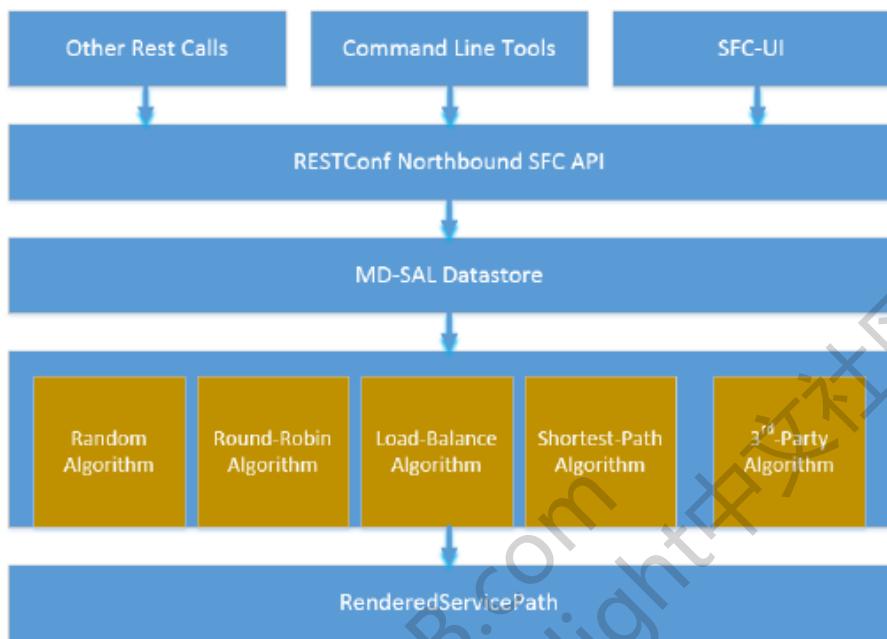
概述

在创建**Rendered**服务链路时，原始的SFC控制器从服务功能名称列表中选择第一个可获得的服务功能，这将可能导致很多问题，例如服务功能过载以及作为SFC的服务链路过长无法获取服务功能和网络拓扑的状态。服务功能选择框架至少支持4种算法（随机、轮询、负载均衡和最短路径），当实例化**Rendered**服务链路时选择最合适的服务功能。此外，这是一个灵活的框架可以支持第三方选择算法。

架构

下图阐明了服务功能选择框架和算法。

Figure 20.7. SF Selection Architecture



一个用户有三种方式选择一个服务功能选择算法：

- 1、集成RESTCONF调用。OpenStack或其他管理系统可以提供插件调用该API选择调度算法。
- 2、命令行工具。`curl`或浏览器插件例如POSTMAN (谷歌浏览器插件)和RESTClient (火狐浏览器插件) 可以通过RESTCONF调用选择调度算法。
- 3、SFC-UI。现在当创建**Rendered**服务链路时SFC-UI提供一个选择算法选项。

RESTCONF北向接口SFC API为选择服务选择算法提供GUI/RESTCONF。MD-SAL数据存储器提供所有支持的服务功能选择算法，并提供API。一旦选用某个服务功能选择算法，该算法就会在创建**Rendered**服务链路时生效。

用调度选择SFs

创建一个**Rendered**服务链路时，管理员可以使用以下方式选择一个选择算法。

命令行工具。命令行工具包括Linux命令`curl`或浏览器插件，例如谷歌浏览器的POSTMAN或火狐浏览器的RESTClient。在这种情况下，需要以下JSON内容：

`Service_function_schedule_type.json`

```
{
  "service-function-scheduler-types": {
    "service-function-scheduler-type": [
      {
        ...
      }
    ]
  }
}
```

```

    "name": "random",
    "type": "service-function-scheduler-type:random",
    "enabled": false
},
{
    "name": "roundrobin",
    "type": "service-function-scheduler-type:round-robin",
    "enabled": true
},
{
    "name": "loadbalance",
    "type": "service-function-scheduler-type:load-balance",
    "enabled": false
},
{
    "name": "shortestpath",
    "type": "service-function-scheduler-type:shortest-path",
    "enabled": false
}
]
}
}

```

如果使用以下curl命令，则：

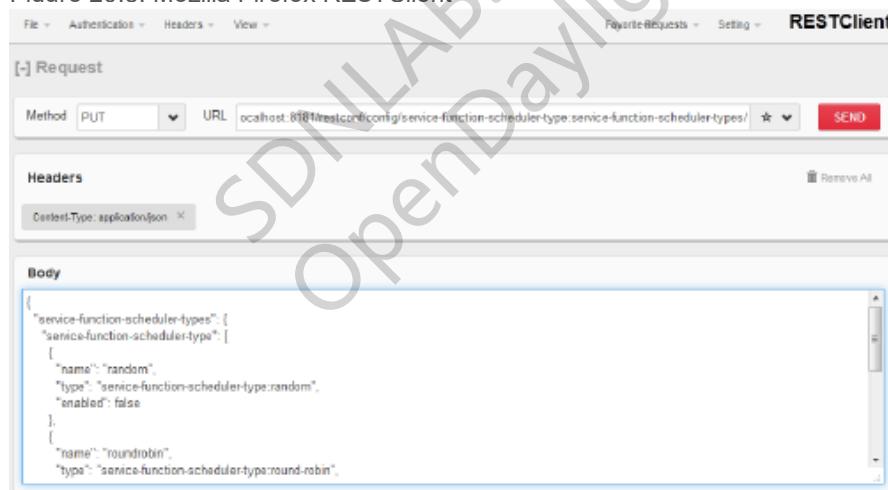
```

curl -i -H "Content-Type: application/json" -H "Cache-Control: no-cache" --data '$${Service_function_schudule_type.json}' -X PUT --user admin:admin http://localhost:8181/restconf/config/servicefunction-scheduler-type:service-function-scheduler-types/

```

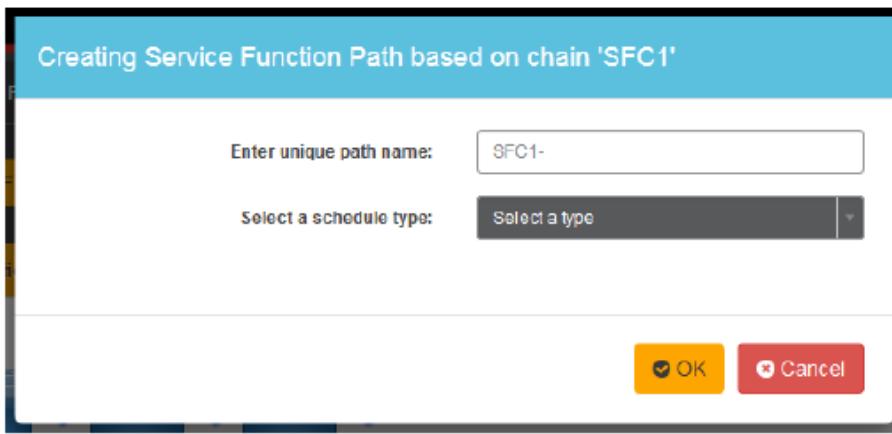
使用RESTClient plugin的快照：

Figure 20.8. Mozilla Firefox RESTClient



SFC-UI。SFC-UI为服务功能选择算法提供一个下拉菜单。以下是一张创建Rendered服务链路时用户使用SFC-UI的快照。

Figure 20.9. Karaf Web UI



注意：下拉菜单中一些服务功能选择算法并不可用，只有前三个可用。

随机算法

从选择服务算法的名称列表中随机选择。

概述

随机算法从选择服务算法的名称列表中随机选择一种服务功能类型。

先决条件

服务功能信息存在数据存储器中。

要么没有算法，要么选择随机算法。

使用说明

在karaf中安装好该插件后，用户可以选择自己最喜欢的方式选择随机调度算法类型，并没有特别的使用指南。

轮询算法

以循环的方式从名称列表中选择服务功能。

概述

采用循环的方式从名称列表中选择一种服务功能。这种方式这可以均衡所有服务功能的负载，但是，不能保证所有服务功能负载相同的工作，因为采用的是基于流的循环。

先决条件

服务功能信息存储在数据存储器中。

选择了循环算法。

目标环境

选用一种循环算法后轮询算法就会生效。

使用说明

在karaf中安装好该插件后，用户可以选择自己最喜欢的方式选择轮询调度算法类型，并没有特别的使用指南。

负载均衡算法

通过实际CPU利用率选择合适的服务功能。

概述

负载均衡算法是通过实际CPU利用率选择合适的服务功能。CPU利用率是通过NETCONF从监控信息中获取的。

先决条件

服务功能的CPU利用率。

NETCONF服务器。

NETCONF客户端。

每个虚拟机有一个NETCONF服务器，可以和NETCONF客户端协同工作。

使用说明

创建虚拟机，在虚拟机中安装NETCONF服务器，确保它们是相互独立的。例如：

1、创建虚拟机

a、创建4个虚拟机，包括2个SF类型的防火墙，其他的是Napt44，将其命名为firewall-1、firewall-2、napt44-1、napt44-2，这四个虚拟机运行在同一服务器或不同服务器上。

b、在每个虚拟机上安装NETCONF服务器，可以在OpenDaylight wiki上查看更多分NETCONF信息：<https://wiki.opendaylight.org/view/>

OpenDaylight_Controller:Config:Examples:Netconf:Manual_netopeer_installation

c、向虚拟机上的NETCONF服务器获取监控数据。下列静态XML数据就是一个例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<service-function-description-monitor-report>
<SF-description>
<number-of-daports>2</number-of-daports>
<capabilities>
<supported-packet-rate>5</supported-packet-rate>
<supported-bandwidth>10</supported-bandwidth>
<supported-ACL-number>2000</supported-ACL-number>
<RIB-size>200</RIB-size>
<FIB-size>100</FIB-size>
<ports-bandwidth>
<port-bandwidth>
<port-id>1</port-id>
<ipaddress>10.0.0.1</ipaddress>
<macaddress>00:1e:67:a2:5f:f4</macaddress>
<supported-bandwidth>20</supported-bandwidth>
```

```
</port-bandwidth>
<port-bandwidth>
<port-id>2</port-id>
<ipaddress>10.0.0.2</ipaddress>
<macaddress>01:le:67:a2:5f:f6</macaddress>
<supported-bandwidth>10</supported-bandwidth>
</port-bandwidth>
</ports-bandwidth>
</capabilities>
</SF-description>
<SF-monitoring-info>
<liveness>true</liveness>
<resource-utilization>
<packet-rate-utilization>10</packet-rate-utilization>
<bandwidth-utilization>15</bandwidth-utilization>
<CPU-utilization>12</CPU-utilization>
<memory-utilization>17</memory-utilization>
<available-memory>8</available-memory>
<RIB-utilization>20</RIB-utilization>
<FIB-utilization>25</FIB-utilization>
<power-utilization>30</power-utilization>
<SF-ports-bandwidth-utilization>
<port-bandwidth-utilization>
<port-id>1</port-id>
<bandwidth-utilization>20</bandwidth-utilization>
</port-bandwidth-utilization>
<port-bandwidth-utilization>
<port-id>2</port-id>
<bandwidth-utilization>30</bandwidth-utilization>
</port-bandwidth-utilization>
</SF-ports-bandwidth-utilization>
</resource-utilization>
</SF-monitoring-info>
</service-function-description-monitor-report>
```

2、启动SFC

a、减压SFC压缩包。

b、运行SFC: \${sfc}/bin/karaf。在OpenDaylight SFC wiki上可以查看更多关于服务功能链的信息。

3、验证负载均衡算法

a、部署SFC2(firewall-abstract2→napt44-abstract2)，点击按钮在SFC UI中创建Rendered服务链路(<http://localhost:8181/sfc/index.html>)。

b、验证Rendered服务链路并确保同类服务功能中所选跳的CPU利用率是其中最小的。正确的RSP是firewall-1→napt44-2。

最短路径算法

利用Dijkstra的算法选出合适的服务功能。Dijkstra算法是选出节点间最短的路径。

概述

最短路径算法用来从实际拓扑中选择出最合适的服务功能。

先决条件

部署拓扑（包括SFFs、SFs和其他链路）。

Dijkstra的算法。更多关于Dijkstra的算法的信息可以参见wiki:http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

使用说明

1、启动SFC

a、减压SFC压缩包。

b、运行SFC: \${sfc}/bin/karaf。

c、部署SFFs和SFs。导入service-function-forwarders.json 和 servicefunctions.json(<http://localhost:8181/sfc/index.html>→/sfc/config)。

service-function-forwarders.json:

```
{  
    "service-function-forwarders": {  
        "service-function-forwarder": [  
            {  
                "name": "SFF-br1",  
                "service-node": "OVSDDB-test01",  
                "rest-uri": "http://localhost:5001",  
                "sff-data-plane-locator": [  
                    {  
                        "name": "eth0",  
                        "service-function-forwarder-ovs:ovs-bridge": {  
                            "uuid": "4c3778e4-840d-47f4-b45e-0988e514d26c",  
                            "bridge-name": "br-tun"  
                        },  
                        "data-plane-locator": {  
                            "port": 5000,  
                            "ip": "192.168.1.1",  
                            "transport": "service-locator:vxlan-gpe"  
                        }  
                    }  
                ],  
                "service-function-dictionary": [  
                    {  
                        "sff-sf-data-plane-locator": {  
                            "port": 10001,  
                            "ip": "10.3.1.103"  
                        },  
                        "name": "napt44-1",  
                        "type": "service-function-type:napt44"  
                    },  
                    {  
                        "sff-sf-data-plane-locator": {  
                            "port": 10003,  
                            "ip": "10.3.1.102"  
                        },  
                        "name": "firewall-1",  
                        "type": "service-function-type:firewall"  
                    }  
                ],  
                "connected-sff-dictionary": [  
                    {  
                        "name": "SFF-br3"  
                    }  
                ]  
            }  
        ]  
    }  
}
```

```
]
},
{
  "name": "SFF-br2",
  "service-node": "OVSDB-test01",
  "rest-uri": "http://localhost:5002",
  "sff-data-plane-locator": [
    {
      "name": "eth0",
      "service-function-forwarder-ovs:ovs-bridge": {
        "uuid": "fd4d849f-5140-48cd-bc60-6ad1f5fc0a1",
        "bridge-name": "br-tun"
      },
      "data-plane-locator": {
        "port": 5000,
        "ip": "192.168.1.2",
        "transport": "service-locator:vxlan-gpe"
      }
    }
  ],
  "service-function-dictionary": [
    {
      "sff-sf-data-plane-locator": {
        "port": 10002,
        "ip": "10.3.1.103"
      },
      "name": "napt44-2",
      "type": "service-function-type:napt44"
    },
    {
      "sff-sf-data-plane-locator": {
        "port": 10004,
        "ip": "10.3.1.101"
      },
      "name": "firewall-2",
      "type": "service-function-type:firewall"
    }
  ],
  "connected-sff-dictionary": [
    {
      "name": "SFF-br3"
    }
  ]
},
{
  "name": "SFF-br3",
  "service-node": "OVSDB-test01",
  "rest-uri": "http://localhost:5005",
  "sff-data-plane-locator": [
    {
      "name": "eth0",
      "service-function-forwarder-ovs:ovs-bridge": {
        "uuid": "fd4d849f-5140-48cd-bc60-6ad1f5fc0a4",
        "bridge-name": "br-tun"
      },
      "data-plane-locator": {
        "port": 5000,
        "ip": "192.168.1.2",
        "transport": "service-locator:vxlan-gpe"
      }
    }
  ],
  "service-function-dictionary": [
    {
      "sff-sf-data-plane-locator": {
        "port": 10002,
        "ip": "10.3.1.103"
      },
      "name": "napt44-2",
      "type": "service-function-type:napt44"
    },
    {
      "sff-sf-data-plane-locator": {
        "port": 10004,
        "ip": "10.3.1.101"
      },
      "name": "firewall-2",
      "type": "service-function-type:firewall"
    }
  ],
  "connected-sff-dictionary": [
    {
      "name": "SFF-br4"
    }
  ]
},
```

```
"service-function-dictionary": [
  {
    "sff-sf-data-plane-locator": {
      "port": 10005,
      "ip": "10.3.1.104"
    },
    {
      "name": "test-server",
      "type": "service-function-type:dpi"
    },
    {
      "sff-sf-data-plane-locator": {
        "port": 10006,
        "ip": "10.3.1.102"
      },
      "name": "test-client",
      "type": "service-function-type:dpi"
    }
  ],
  "connected-sff-dictionary": [
    {
      "name": "SFF-br1"
    },
    {
      "name": "SFF-br2"
    }
  ]
}
```

service-functions.json:

```
{
  "service-functions": {
    "service-function": [
      {
        "rest-uri": "http://localhost:10001",
        "ip-mgmt-address": "10.3.1.103",
        "sf-data-plane-locator": [
          {
            "name": "preferred",
            "port": 10001,
            "ip": "10.3.1.103",
            "service-function-forwarder": "SFF-br1"
          }
        ],
        "name": "napt44-1",
        "type": "service-function-type:napt44",
        "nsh-aware": true
      },
      {
        "rest-uri": "http://localhost:10002",
        "ip-mgmt-address": "10.3.1.103",
        "sf-data-plane-locator": [
          {
            "name": "master",
            "port": 10002,
            "ip": "10.3.1.103",
            "service-function-forwarder": "SFF-br2"
          }
        ]
      }
    ]
  }
}
```

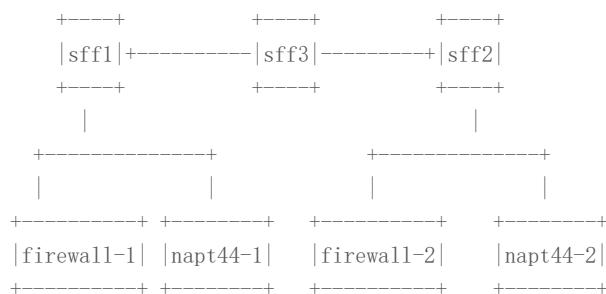
```
],
  "name": "napt44-2",
  "type": "service-function-type:napt44",
  "nsh-aware": true
},
{
  "rest-uri": "http://localhost:10003",
  "ip-mgmt-address": "10.3.1.103",
  "sf-data-plane-locator": [
    {
      "name": "1",
      "port": 10003,
      "ip": "10.3.1.102",
      "service-function-forwarder": "SFF-br1"
    }
  ],
  "name": "firewall-1",
  "type": "service-function-type:firewall",
  "nsh-aware": true
},
{
  "rest-uri": "http://localhost:10004",
  "ip-mgmt-address": "10.3.1.103",
  "sf-data-plane-locator": [
    {
      "name": "2",
      "port": 10004,
      "ip": "10.3.1.101",
      "service-function-forwarder": "SFF-br2"
    }
  ],
  "name": "firewall-2",
  "type": "service-function-type:firewall",
  "nsh-aware": true
},
{
  "rest-uri": "http://localhost:10005",
  "ip-mgmt-address": "10.3.1.103",
  "sf-data-plane-locator": [
    {
      "name": "3",
      "port": 10005,
      "ip": "10.3.1.104",
      "service-function-forwarder": "SFF-br3"
    }
  ],
  "name": "test-server",
  "type": "service-function-type:dpi",
  "nsh-aware": true
},
{
  "rest-uri": "http://localhost:10006",
  "ip-mgmt-address": "10.3.1.103",
  "sf-data-plane-locator": [
    {
      "name": "4",
      "port": 10006,
      "ip": "10.3.1.102",
      "service-function-forwarder": "SFF-br3"
    }
  ],
  "name": "test-client",
```

```

    "type": "service-function-type:dpi",
    "nsh-aware": true
}
]
}
}
}

```

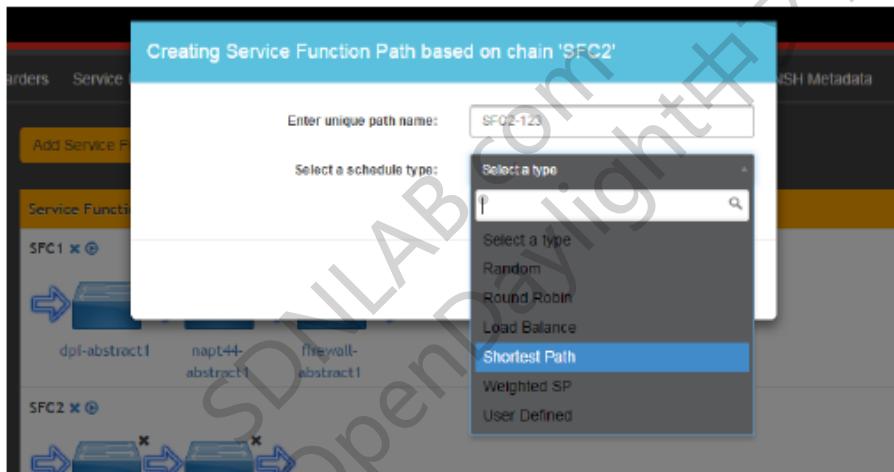
部署的拓扑如下：



3、验证最短路径算法

部署SFC2(firewall-abstract2→napt44-abstract2)，选择“最短路径”作为调度类型，并且点击按钮创建Rendered服务链路(<http://localhost:8181/sfc/index.html>)。

Figure 20.10. select schedule type

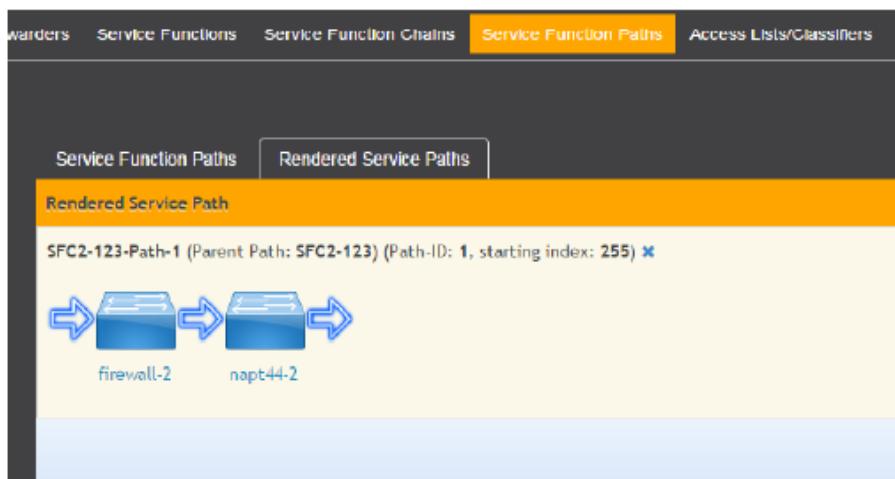


验证Rendered服务链路确保选中的跳连接一个SFF。正确的RSP是firewall-1→napt44-1 或 firewall-2→napt44-2。第一种SF类型是服务功能链中的防火墙，所以算法将会从所有类型是防火墙的SF中随机选出第一跳。假设第一跳是firewall-2，所有从firewall-1到SF且路径类型为Napt44的如下所示：

Path1: firewall-2 → sff2 → napt44-2

Path2: firewall-2 → sff2 → sff3 → sff1 → napt44-1 The shortest path is Path1, so the selected next hop is napt44-2

Figure 20.11. rendered service path



服务功能负载均衡用户指南

概述

SFC负载均衡功能实现服务功能的负载均衡，而不是Service-Function-Forwarder和Service-Function之间的一一映射。

负载均衡架构

在Rendered链路模型中服务功能组（SFG）可以替代服务功能（SF）。可以用SFG或SF定义一个服务链路，但不能同时用这两个。

YANG模型中相关对象如下：

1、Service-Function-Group-Algorithm:

```
Service-Function-Group-Algorithms {
  Service-Function-Group-Algorithm {
    String name
    String type
  }
}
```

Available types: ALL, SELECT, INDIRECT, FAST_FAILURE

2、Service-Function-Group:

```
Service-Function-Groups {
  Service-Function-Group {
    String name
    String serviceFunctionGroupAlgorithmName
    String type
    String groupId
    Service-Function-Group-Element {
      String service-function-name
      int index
    }
  }
}
```

3、ServiceFunctionHop: 引用 SFG (or SF)名称

教程

本教程介绍怎样创建简单的SFC配置，用SFG替代SF。在本例中，SFG包含2个SF。

安装SFC

一般SFC的安装和场景参见SFC wiki: https://wiki.opendaylight.org/view/Service_Function_Chaining:Main→SFC_101

创建一个算法

POST - <http://127.0.0.1:8181/restconf/config/service-function-group-algorithm:servicefunction-group-algorithms>

```
{  
    "service-function-group-algorithm": [  
        {  
            "name": "alg1"  
            "type": "ALL"  
        }  
    ]  
}
```

(Header "content-type": application/json)

验证：获取所有算法

GET - <http://127.0.0.1:8181/restconf/config/service-function-group-algorithm:servicefunction-group-algorithms>

删除所有算法: DELETE - <http://127.0.0.1:8181/restconf/config/servicefunction-group-algorithm:service-function-group-algorithms>

创建一个组

POST - <http://127.0.0.1:8181/restconf/config/service-function-group:service-functiongroups>

```
{  
    "service-function-group": [  
        {  
            "rest-uri": "http://localhost:10002",  
            "ip-mgmt-address": "10.3.1.103",  
            "algorithm": "alg1",  
            "name": "SFG1",  
            "type": "service-function-type:napt44",  
            "sfc-service-function": [  
                {  
                    "name": "napt44-104"  
                },  
                {  
                    "name": "napt44-103-1"  
                }  
            ]  
        }  
    ]  
}
```

}

验证：获取所有SFG的

GET - <http://127.0.0.1:8181/restconf/config/service-function-group:service-function-groups>

21 SNMP插件用户指南

安装功能

SNMP插件能够通过使用一个单独的karaf功能安装： odl-snmp-plugin。

启动Karaf之后：

安装功能： feature:install odl-snmp-plugin

扩展北向API： feature:install odl-restconf

北向API

主要有两个暴露的北向APIs： snmp-get和snmp-set。

SNMP获取

默认URL： <http://localhost:8181/restconf/operations/snmp:snmp-get>。

POST输入

域名 (Field Name)	类型	描述	实例	是否满足需求
ip-address	IPv4地址	预期网络节点的IPv4地址	10.86.3.13	是
oid(对象标识)	String	预期MIB表/对象的对象标识符	1.3.6.1.2.1.1.1	是
get-type	枚举(GET、GET-NEXT、GET-BULK、GET-WALK)	发送得到请求的类型	GET-BULK	是
community	String	为SNMP请求来使用的community字符串	private	否 (默认: public)

实例：

```
{
  "input": {
    "ip-address": "10.86.3.13",
    "oid" : "1.3.6.1.2.1.1.1",
    "get-type" : "GET-BULK",
    "community" : "private"
  }
}
```

POST输出

域名 (Field Name)	类型	描述
结果	List of { "value" : String } pairs	SNMP查询的结果

实例：

```
{
  "snmp:results": [
    {
      "value": "Ethernet0/0/0",
      "oid": "1.3.6.1.2.1.2.2.1.2.1"
    },
    {
      "value": "FastEthernet0/0/0",
      "oid": "1.3.6.1.2.1.2.2.1.2.2"
    },
    {
      "value": "GigabitEthernet0/0/0",
      "oid": "1.3.6.1.2.1.2.2.1.2.3"
    }
  ]
}
```

SNMP设置

默认URL： <http://localhost:8181/restconf/operations/snmp:snmp-set>。

POST输入

SDNLAB.com
OpenDaylight中文社区

域名 (Field Name)	类型	描述	实例	是否满足需求
ip-address	IPv4地址	预期网络节点的IPv4地址	10.86.3.13	是
oid(对象标识)	String	预期MIB表/对象的对象标识符	1.3.6.1.2.1.1.1	是
value	String	在网络设备上设置的值	"Hello World"	是
community	String	为SNMP请求来使用的community字符串	private	否 (默认: public)

实例：

```
{  
  "input": {  
    "ip-address": "10.86.3.13",  
    "oid" : "1.3.6.1.2.1.1.0",  
    "value" : "Sample description",  
    "community" : "private"  
  }  
}
```

POST输出

在一个成功的SNMP-SET中，是没有输出呈现的，只返回一个200的HTTP状态。

错误

如果在设置请求里发生任何错误，将在输出中呈现一个错误信息。

如，在一个失败的设置请求中你可以看到像这样的一个错误：

```
{  
  "errors": {  
    "error": [  
      {  
        "error-type": "application",  
        "error-tag": "operation-failed",  
        "error-message": "SnmpSET failed with error status: 17, error  
index: 1. StatusText: Not writable"  
      }  
    ]  
  }  
}
```

这个错误与在SNMPv2 RFC(<https://tools.ietf.org/html/rfc1905>)中的错误状态17是相一致的。

22 SXP用户指南

概述

SXP项目利用IP-SGT绑定强化OpenDaylight平台。据Smith, Kandula所说，目前支持SXP协议第四版。同时也支持版本1-3。此外，版本4将单向连接拓展为双向连接。

SXP架构

SXP服务器管理所有以单线程连接的客户端并且对等端之间采用通用SXP协议建立连接。每个SXP网络对等端都是模拟与其相关的class，例如，SXP服务器代表SXP Speaker，客户端代表SXP Listener。服务器程序在特定的端口创建一个ServerSocket对象，然后等待客户端启动并向服务器的IP地址和端口发起连接。客户端程序开启一个连接服务器所在主机IP地址和端口的套接字。

SXP Listener维护与对等speaker之间的连接。一个开放的通道，所有收到的SXP消息被多个处理程序处理。消息必须经历的处理过程包括解码、解析和验证。

SXP Speaker对应于SXP Listener，负责维护与监听端的连接以及发送消息。

SXP Binding Handler从一个信息中提取IP-SGT绑定并放进SXP数据库中。如果在提取过程中检测到错误，那么选择合适的错误代码和子代码，并且向连接的对等端发送错误消息。所有传递的消息都直接引导到SXP Binding Dispatcher的输出队列中。

IP-SGT Manager处理多个连接的绑定，如果向SXP数据库添加新数据或删除数据，或者检测到绑定的贡献者改变了，那么管理者就在SXP数据库上进行处理解决绑定的冲突问题，防止可能的信息环路。最后，更新IP-SGT-Master数据库，该数据库由有效的、唯一的绑定组成，每个IP地址只有一个绑定。

IP-SGT Manager还包含RPCs，其他OpenDaylight插件通过RPC进行REST调用。向SXP数据库添加、更新或删除绑定。

SXP Binding Dispatcher代表一个选择器，决定何时从SXP数据发送多少数据。并且负责基于最大消息长度的消息的内容组成。

配置SXP

OpenDaylight karaf有预配置基本的SXP配置。

22-sxp-controller-one-node.xml (定义基本参数)

管理SXP

通过RPC（响应是包含请求数据或操作状态的XML文档）

获取连接 POST <http://127.0.0.1:8181/restconf/operations/sxp-controller:getconnections>

```
<input xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<requested-node>0.0.0.100</requested-node>
</input>
```

添加连接 POST <http://127.0.0.1:8181/restconf/operations/sxp-controller:addconnection>

```
<input xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<connections>
<connection>
<!--vpn>vpn1</vpn-->
<peer-address>172.20.161.50</peer-address>
<!--source-ip></source-ip-->
<tcp-port>64999</tcp-port>
<!-- Password setup: default | none -->
<password>default</password>
<!-- Mode: speaker/listener/both -->
<mode>speaker</mode>
<version>version4</version>
<description>Connection to ASRIK</description>
<!-- Timers setup: 0 to disable specific timer usability, the default value
will be used -->
<connection-timers>
<!-- Speaker -->
<hold-time-min-acceptable>45</hold-time-min-acceptable>
<keep-alive-time>30</keep-alive-time>
</connection-timers>
</connection>
<connection>
<!--vpn>vpn1</vpn-->
<peer-address>172.20.161.178</peer-address>
<!--source-ip></source-ip-->
<tcp-port>64999</tcp-port>
<!-- Password setup: default | none -->
<password>default</password>
<!-- Mode: speaker/listener/both -->
<mode>listener</mode>
<version>version4</version>
<description>Connection to ISR</description>
<!-- Timers setup: 0 to disable specific timer usability, the default value
will be used -->
```

```

<connection-timers>
  <!-- Listener -->
  <!-- NON-CONFIGURABLE delete-hold-down-time -->
  <reconciliation-time>120</reconciliation-time>
  <hold-time>90</hold-time>
  <hold-time-min>90</hold-time-min>
  <hold-time-max>180</hold-time-max>
</connection-timers>
</connection>
</connections>
</input>

```

删除连接 POST <http://127.0.0.1:8181/restconf/operations/sxpcontroller:delete-connection>

```

<input xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <peer-address>172.20.161.50</peer-address>
</input>

```

添加绑定项 POST <http://127.0.0.1:8181/restconf/operations/sxp-controller:addentry>

```

<input xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ip-prefix>192.168.2.1/32</ip-prefix>
  <sgt>20</sgt>
</input>

```

更新绑定项 POST <http://127.0.0.1:8181/restconf/operations/sxpcontroller:update-entry>

```

<input xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <original-binding>
    <ip-prefix>192.168.2.1/32</ip-prefix>
    <sgt>20</sgt>
  </original-binding>
  <new-binding>
    <ip-prefix>192.168.3.1/32</ip-prefix>
    <sgt>30</sgt>
  </new-binding>
</input>

```

删除绑定项 POST <http://127.0.0.1:8181/restconf/operations/sxpcontroller: delete-entry>

```

<input xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ip-prefix>192.168.3.1/32</ip-prefix>
  <sgt>30</sgt>
</input>

```

获取节点绑定

RPC获取某个设备绑定。通过唯一的node-ID识别SXP-aware节点，如果用户请求绑定Speaker **20.0.0.2**，那么RPC就会在SXP数据库本地学习到的SXP数据中搜索包含Node-ID **20.0.0.2**的合适的路径，并且回复相关的绑定。

POST <http://127.0.0.1:8181/restconf/operations/sxp-controller:get-node-bindings>

```

<input xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <requested-node>20.0.0.2</requested-node>
</input>

```

获取绑定SGTs <http://127.0.0.1:8181/restconf/operations/sxp-controller:getbinding-sgts>

```

<input xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

```

```
<ip-prefix>192.168.12.2/32</ip-prefix>
</input>
```

SXP用例

思科的网络设备有广泛的安装基础支持SXP。OpenDaylight中包含SXP，使得策略组和IP地址的绑定可以处理更多的设备和更多运行在OpenDaylight上的应用。并且处理应用的范围是可扩展的，下面就是其中一部分：

基于OpenDaylight的应用可以利用IP-SGT绑定信息。例如，用户可以利用策略组定义访问控制，而OpenDaylight可以使用IP地址在网络元件中配置访问控制列表。

不同设备商之间的互操作性。不同设备商拥有不同的策略系统，了解IP-SGT绑定让思科有能力维护思科和其他设备商之间的策略组。

OpenDaylight可以从多个设备处收集绑定信息并发送给一个网络元件。例如，防火墙可以使用IP-SGT信息根据基于组的ACLs来处理IPs。但是如果只使用SXP，那么防火墙需要维护大量的网络连接才能获取绑定信息。维护所有的SXP对等连接和协议信息需要耗费大量的开销。OpenDaylight可以搜集IP-group信息，那么防火墙只需要连接OpenDaylight即可。通过将信息流从网络元件内部移动到一个集中的位置，可以减少执行元件CPU的压力。只需要一条连接而不是成千上万条，减轻了执行元件的负担让其可以专注于原始的工作，转发和执行。

OpenDaylight可将绑定信息从一个网络元件转发给其他元件。以组的形式转发是一种更高效的模式。例如，安全应用可能会发现某个主机行为异常或违反安全策略，定义的动作是将该主机放到一个特别的源组中以进行下一步，接着降低该主机的服务质量、限制对关键服务器的访问或者设定特殊路由条件以确保安全执行。该主机更新组关系需要迅速与多个网络元件通信。一个高效的转发方式就是使用OpenDaylight作为集中点去转发信息。

尽管IP-SGT绑定只是一个特定的信息，而且SXP只是广泛应用于一个设备商的设备中，利用OpenDaylight处理、分发绑定是应用策略组的一种有效方式，这将大大提高OpenDaylight和策略组的可用性。

23 TCPMD5用户指南

概述

TCPMD5库在TCP堆栈支持的操作系统中提供访问RFC-2385 MD5 Signature Option。这个选项已经用来保护BGP会话，同样的，对保护PCEP会话也是有用的。

重要：在继续按照用户指南操作之前，确保BGP和/或PCEP是被正确配置的。

TCPMD5认证默认是关闭的，为了启用（在BGP和PCEP协议中使用），取消20-tcpmd5.xml文件中的批注内容。在OpenDaylight目录etc/opendaylight/karaf下能够找到这个配置文件。

注意：如果连接没有建立，且没有任何警告或者错误，再次检查配置和密码。

手动配置TCPMD5

BGP

上文已经提及，确定20-tcpmd5.xml中的内容已经取消注释。为BGP协议启动TCPMD5，进行以下步骤：

1.在31-bgp.xml文件中取消TCPMD5会话的注释批注：

```
<!--
Uncomment this block to enable TCPMD5 Signature support
-->
```

```

<md5-channel-factory>
<type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">prefix:md5-
channel-factory</type>
<name>md5-client-channel-factory</name>
</md5-channel-factory>
<md5-server-channel-factory>
<type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">prefix:md5-
server-channel-factory</type>
<name>md5-server-channel-factory</name>
</md5-server-channel-factory>

```

2.在41-bgp-example.xml文件中添加标签到 example-bgp-peer模块中：

```

<!--
For TCPMD5 support, make sure the dispatcher associated with the rib has
"md5-channel-factory" attribute set and then add a "password" attribute
here.

Note that the peer has to have the same password configured, otherwise the
connection will not be established.
-->
<module>
<type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">prefix:bgp-
peer</type>
<name>example-bgp-peer</name>
<host>10.25.2.27</host>
<holdtimer>180</holdtimer>
<rib>
<type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:cfg">prefix:rib</
type>
<name>example-bgp-rib</name>
</rib>
<advertized-table>
<type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">prefix:bgp-
table-type</type>
<name>ipv4-unicast</name>
</advertized-table>
<advertized-table>
<type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">prefix:bgp-
table-type</type>
<name>ipv6-unicast</name>
</advertized-table>
<advertized-table>
<type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">prefix:bgp-
table-type</type>
<name>linkstate</name>
</advertized-table>
<password>changeme</password>
</module>

```

注意：在其他BGP设备上设置密码是不属于本文档范围。

PCEP

上文已经提及，确定20-tcpmd5.xml中的内容已经取消注释。为PCE协议启动TCPMD5，进行以下步骤：

1.在32-pcep.xml文件中取消TCPMD5会话的注释批注：

```
<!--
Uncomment this block to enable TCPMD5 Signature support
-->
<md5-channel-factory>
<type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">prefix:md5-
channel-factory</type>
<name>md5-client-channel-factory</name>
</md5-channel-factory>
<md5-server-channel-factory>
<type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">prefix:md5-
server-channel-factory</type>
<name>md5-server-channel-factory</name>
</md5-server-channel-factory>
```

2.在39-pcep-provide.xml文件中取消一下代码的注释批注：

```
<!--
For TCPMD5 support make sure the dispatcher has the "md5-server-channel-
factory"
attribute set and then set the appropriate client entries here. Note that
if this
option is configured, the PCCs connecting here must have the same password,
otherwise they will not be able to connect.
-->
<client>
<address>192.0.2.2</address>
<password>changeme</password>
</client>
```

重要：改变PCC地址的值，一个是用来通知PCE，另一个是用来提供密码，匹配PCC上的设置。另外，PCC设置密码是在这篇文档范围之外的。

通过RESTCONF配置TCPMD5

重要：开始之前，确定已经安装了BGP或者PCEP的功能。安装其他功能，将提供访问restconf/config/URLs。

```
feature:install odl-netconf-connector-all
```

日志信息中显示成功的启动了netconf-connector：Netconf connector initialized successfully。

检查当前已经配置的模块：

```
http://localhost:8181/restconf/config/network-topology:network-topology/topology=topology-netconf/node/controller-config/yang-ext:mount/config:modules/
```

检查当前已经配置的服务：

```
http://localhost:8181/restconf/config/network-topology:network-topology/topology=topology-netconf/node/controller-config/yang-ext:mount/config:services/
```

这些URL也被用来POST新配置。如果想改变列表中的其他配置，确定包含了正确的namespaces。正确的 namespaces格式总是： urn:opendaylight:params:xml:ns:yang:controller:config。通过在配置yang文件中找到提

供的模块能够发现任何其他域的namespace。

注意：RESTCONF将告诉一些namespace是否是错误的。为了众多协议中的任一个启用TCPMD5，启用TCPMD5模块和服务，必须为每个模块或服务分离POST请求。

模块URL为：<http://localhost:8181/restconf/config/network-topology:network-topology/topology-topology-netconf/node/controller-config/yang-ext:mount/config:modules/>。

模块POST：

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:jni:cfg">x:native-key-
access-factory</type>
<name>global-key-access-factory</name>
</module>
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">x:md5-
client-channel-factory</type>
<name>md5-client-channel-factory</name>
<key-access-factory xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:cfg">x:key-access-
factory</type>
<name>global-key-access-factory</name>
</key-access-factory>
</module>
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
<type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">prefix:md5-
server-channel-factory-impl</type>
<name>md5-server-channel-factory</name>
<server-key-access-factory xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:cfg">x:key-access-
factory</type>
<name>global-key-access-factory</name>
</server-key-access-factory>
</module>
```

服务URL为：<http://localhost:8181/restconf/config/network-topology:network-topology/topology-topology-netconf/node/controller-config/yang-ext:mount/config:services/>。

服务POST：

```
<service xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:cfg">x:key-access-
factory</type>
<instance>
<name>global-key-access-factory</name>
<provider>/modules/module[type='native-key-access-factory'][name='global-
key-access-factory']</provider>
</instance>
</service>
<service xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">x:md5-
```

```

channel-factory</type>
<instance>
<name>md5-client-channel-factory</name>
<provider>/modules/module[type='md5-client-channel-factory'][name='md5-
client-channel-factory']</provider>
</instance>
</service>
<service xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
<type xmlns:prefix=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">prefix:md5-
server-channel-factory</type>
<instance>
<name>md5-server-channel-factory</name>
<provider>/modules/module[type='md5-server-channel-factory-impl'][name='md5-
server-channel-factory']</provider>
</instance>
</service>

```

BGP

注意：必须介绍在先前的会话中提及到的模块和服务。BGP客户端需要已经被配置，为BGP检查用户指南。复制和粘贴完整的模块以取代它，指南中主要显示的是需要改变的部分。

1.在BGP配置中启用TCPMD5:

URL: <http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules/odl-bgp-rib-impl-cfg:bgp-dispatcher-impl/global-bgp-dispatcher>。

PUT:

```

<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:bgp-
dispatcher-impl</type>
<name>global-bgp-dispatcher</name>
<md5-channel-factory xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">x:md5-
channel-factory</type>
<name>md5-client-channel-factory</name>
</md5-channel-factory>
<md5-server-channel-factory xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">x:md5-
server-channel-factory</type>
<name>md5-server-channel-factory</name>
</md5-server-channel-factory>
...
</module>

```

注意复制和粘贴完整的模块以取代它，指南中主要显示的是需要改变的部分。

2.设置密码:

URL: <http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules/odl-bgp-rib-impl-cfg:bgp-peer/example-bgp-peer>

peer**PUT:**

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:bgp-peer</
type>
<name>example-bgp-peer</name>
<password xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">changeme</
password>
...
</module>
```

PCEP

注意：必须介绍在先前的会话中提及到的模块和服务。BGP客户端需要已经被配置，为BGP检查用户指南。复制和粘贴完整的模块以取代它，指南中主要显示的是需要改变的部分。

1.在PCEP配置中启用TCPMD5：

URL: <http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules/odl-pcep-impl-cfg:pcep-dispatcher-impl/global-pcep-dispatcher>

PUT:

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:pcep:impl">x:pcep-dispatcher-
impl</type>
<name>global-pcep-dispatcher</name>
<md5-channel-factory xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:pcep:impl">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">x:md5-
channel-factory</type>
<name>md5-client-channel-factory</name>
</md5-channel-factory>
<md5-server-channel-factory xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:pcep:impl">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:tcpmd5:netty:cfg">x:md5-
server-channel-factory</type>
<name>md5-server-channel-factory</name>
</md5-server-channel-factory>
...
</module>
```

注意复制和粘贴完整的模块以取代它，指南中主要显示的是需要改变的部分。

2.设置密码

URL: <http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules/odl-pcep-impl-cfg:pcep-topology-provider/pcep-topology>

PUT:

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
<type xmlns:x=
"urn:opendaylight:params:xml:ns:yang:controller:pcep:topology:provider">x:pcep-
topology-provider</type>
<name>pcep-topology</name>
<client xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:pcep:topology:provider">
<address xmlns=
"urn:opendaylight:params:xml:ns:yang:controller:pcep:topology:provider">192.0.2.2</address> <!--CHANGE THE
<password>changeme</password> <!--CHANGE THE VALUE -->
</client>
...
</module>
```

24 TSDR H2数据存储使用指南

概述

在锂版本的时间序列数据存储库(TSDR)中，捕获了与OpenFlow统计数据对应的时间序列指标。为了让用户在他们的laptop或者non-production环境尝试一些实验，TSDR功能可以通过安装`odl-tsdr-all`功能，启用默认数据存储H2来实现。

TSDR架构

下面的wiki页面展示了TSDR模型/架构

a.https://wiki.opendaylight.org/view/TSDR_Data_Storage_Service_and_Persistence_with_HBase_Plugin

b.https://wiki.opendaylight.org/view/TSDR_Data_Collection_Service

注意在锂版本中的DataCollection服务的实现仅仅是为了OpenFlow统计。

使用默认数据存储H2配置TSDR

基于H2的存储文件在`/tsdr`下自动存储。如果你想改变默认的存储位置，在`/etc`下找到配置文件。文件名是`org.ops4j.datasourcemetric.cfg`。修改`url=jdbc:h2:/tsdr/metric`的最后一部分，指向别的目录。

经营或管理使用默认datastore H2的TSDR

一旦TSDR默认数据存储功能（`odl-tsdr-all`）启动，可以在Karaf控制台输入以下命令访问TSDR捕获的OpenFlow统计值。

```
tsdr:list <metric-category> <starttimestamp> <endtimestamp>
```

其中，

- = 以下类型之一： FlowGroupStats,
FlowMeterStats, FlowStats, FlowTableStats, PortStats, QueueStats
- = 过滤数据列表启动时间戳

=过滤数据列表关闭时间戳

如果或没有指定，这个命令会抓取至少1000条最新的值。

使用默认H2数据存储的TSDR功能，你还可以得到额外的命令：

```
tsdr:purgeAll
```

这将清除收集的所有tsdr数据记录。

25 TSDR HBase数据存储用户指南

概述

OpenDaylight中的时间序列存储（TSDR）项目创建了一个框架，用于收集、存储、维护OpenDaylight控制器中的时间序列数据。TSDR为数据收集器提供一个框架收集各种各样的时间序列数据并且存储到TSDR数据存储器中。借助一个通用数据模型和通用TSDR数据API，用户可以自主选择将数据存储器插到TSDR框架中。锂版本中支持两种数据存储：HBase NoSQL database和H2 relational database。

管理员借助TSDR提供的数据收集、存储、查询、聚合和清理功能可以在TSDR上使用多种数据驱动的应用，如安全风险探测、性能分析、操作配置优化、流量工程、智能化网络分析等。

TSDR与HBase数据存储器架构

TSDR包含以下服务和组件：数据收集服务、数据存储服务、TSDR Persistence Layer with data stores as plugins、TSDR数据存储、数据查询服务、数据聚集服务、数据清除服务。

数据收集服务负责收集时间序列数据，然后转交给数据存储服务，数据存储服务通过TSDR Persistence层将数据存到TSDR中。TSDR Persistence层提供通用的服务API允许不同的数据存储器插入。数据聚合服务将细粒度数据转化成粗粒度汇总数据。数据清除服务按照用户设定的时间表定期清理细粒度数据和粗粒度数据。

锂版本中，实现了数据收集服务、数据存储服务、TSDR Persistence层、TSDR HBase数据存储和TSDR H2数据存储。在这些服务和组件中，时间序列数据使用通用TSDR数据模型进行通信，该模型是时间序列数据通用性的抽象。有了这些服务功能，TSDR可以从数据源收集数据并存储在TSDR数据存储器中：HBase数据存储或H2数据存储。此外，为用户提供简单的命令查询，可从karaf控制台检索数据存储器中的TSDR数据。

以后的版本会包含数据聚合服务、数据清除服务，以及一个完备的数据查询服务。

用HBase数据存储配置TSDR

在安装OpenDaylight的虚拟机中安装HBase服务器，如果用户接受HBase数据存储器的默认配置，那么用户可以直接从karaf控制台继续安装HBase数据存储器。

当然，用户可以按照HBase数据存储器的配置过程配置TSDR HBase数据存储器。

HBase数据存储器配置步骤：

- 1、在karaf目录下打开etc/tsdr-persistence-hbase.properties文件。
- 2、编辑下列参数：HBase服务器名称、HBase服务器端口、HBase客户端连接池大小、HBase客户端写入缓冲区大小。

配置完HBase数据存储器后，继续从karaf控制台安装HBase数据存储器。

HBase数据存储器安装步骤：

- 1、启动karaf控制台；
- 2、运行命令： feature:install odl-tsdr-hbase。

实施或管理TSDR HBase数据存储器

用户从karaf控制台输入以下命令可以检索HBase数据存储器中的数据：

```
tsdr:list  
tsdr:list <CategoryName> <StartTime> <EndTime>
```

输入命令后按Tab键可以获取上下文信息。

日志文件故障解决问题。

karaf日志与OpenDaylight其他功能组件相似，TSDR HBase数据存储器将日志信息写到karaf日志中。包含所有的信息消息，警告、错误消息和调试消息。

HBase日志是HBase系统级日志，用户可以查看/logs下标准的HBase服务器日志。

教程

怎样使用TSDR收集、存储、查看OpenFlow接口统计数据

概述

教程描述如何在OpenDaylight环境下使用TSDR收集、存储、查看时间序列数据。

先决条件

需要具备以下条件：

有一个或多个OpenFlow交换机、或者使用mininet仿真一台。

成功安装OpenDaylight控制器。

成功按照TSDR HBase数据存储器安装指南安装好HBase数据存储器。

连接OpenFlow交换机和OpenDaylight控制器。（HBase数据存储器只支持Linux操作系统）

使用说明

启动OpenDaylight控制器。

连接OpenFlow交换机和控制器。

如果使用mininet，则运行以下命令：

```
mn --topo single,3 --controller remote, ip=172.17.252.210, port=6653 --switch ovsk, protocols=OpenFlow13
```

如果使用的是物理交换机，那么控制器应该可以发现网络拓扑。

从karaf控制台安装tsdr hbase功能组件：

```
feature:install odl-tsdr-hbase
```

在karaf控制台运行以下命令：

```
tsdr:list InterfaceStats
```

可以在HBase数据存储器中看见交换机的接口统计数据。如果有很多行，则使用"tsdr:list InterfaceStats|more"命令一页一页的看。

在"tsdr:list"后面添加数据种类，例如，"tsdr:list FlowStats"则输出交换机收集到的所有流统计数据。

26 TTP CLI Tools用户指南

概述

Table Type Patterns (TTP) 是Open Networking Foundation开发的一个用于描述和协商OpenFlow协议的一个子集。它用来描述支持OpenFlow协议的硬件应该支持的特性（当然也包括他们不能支持的特性）。可以在OpenFlow说明页看到TTP的更多详细信息。

TTP CLI Tools架构

TTP CLI Tools使用TTP模块和YANG Tools/RESTCONF解码器实现数据传输单元（DTOs）和JSON/XML之间的转换。

27 统一安全通道

概述

在企业网络中，越来越多的控制器和网络管理系统正在被远程部署，如在云上部署。另外，企业网络正变得越来越多样化——分支、物联网、无线（包括云访问控制）等。企业消费者想要一个聚合网络控制器和管理系统解决方案。此功能适用于希望为系统使用统一安全通道的设备和网络管理员。

USC Channel架构

USC Agent: 在设备支持的所有标准协议上提供proxy和agent功能，它发起控制器call-home服务、维护与控制器连接，为USC头包作为一个分路器或复用器，并验证控制器。

USC Plugin: 负责控制器和USC Agent之间的通信，响应控制器call-home服务、与设备保持连接，为USC头包作为一个分路器或复用器，提供支持TLS或DTLS。

USC Manager: 为USC处理支持USC的配置、高可用性、安全、监控、集群。

USC UI: 在OpenDaylight DLUX UI界面负责显示一个GUI（图形用户接口）呈现USC的状态。

安装USC Channel

为安装USC，下载OpenDaylight和使用Karaf控制台安装odl-usc-channel-ui功能：

```
feature:install odl-usc-channel-ui
```

配置USC Channel

主要提供USC中各种组件的详细配置设置。Karaf的USC配置文件位于/karaf/target/assembly/etc/usc目录下：

认证：为安全性考虑，认证目录包含客户端key、pem、rootca文件是需要的。

akka.conf：包含集群的相关配置。在akka网址（<http://doc.akka.io>）可能会找到潜在的配置属性。

usc.properties：包含USC相关的配置。使用文件设置认证的位置、定义额外的akka配置源、为USC行为分配默认设置。

实施或管理USC Channel

从Karaf控制台安装odl-usc-channel-ui功能后，用户能够从UI或APIDOCs explorer上实施和管理USC通道。

登录[http://\\${ipaddress}:8181/index.html](http://${ipaddress}:8181/index.html)，点击USC旁的菜单键，从这儿，用户可以查看USC通道的状态。

登录[http://\\${ipaddress}:8181/apidoc/explorer/index.html](http://${ipaddress}:8181/apidoc/explorer/index.html)，扩展usc-channel面板，从这儿，用户能够执行各种API调用来测试USC部署，如：add-channel、delete-channel、view-channel等。

教程

以下为USC通道的教程。

查看USC Channel

此教程的目的是查看USC Channel。

概述

本教程通过查看USC Channel环境拓扑的进程引导用户，包含USC拓扑中控制器和设备之间已建立的通道连接。

前提条件

此教程中，假设已经安装了一个运行USC代理的设备。

说明

1.运行OpenDaylight，在Karaf控制台安装odl-usc-channel-ui；

2.登录[http://\\${ipaddress}:8181/apidoc/explorer/index.html](http://${ipaddress}:8181/apidoc/explorer/index.html)；

3.根据下面的json数据，执行add-channel：{"input":{"channel": {"hostname":"127.0.0.1","port":1068,"remote":false}}};

4.登录[http://\\${ipaddress}:8181/index.html](http://${ipaddress}:8181/index.html)；

- 5.点击USC旁的菜单键；
- 6.UI上应该显示一个表格，表格中包含步骤3添加的channel。

28 虚拟租户网络 (VTN)

VTN概述

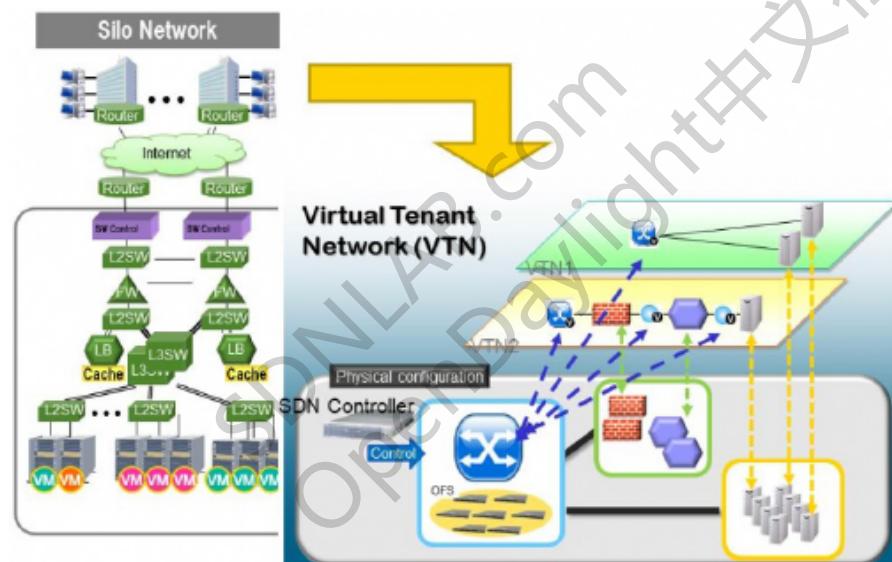
OpenDaylight虚拟租户网络 (VTN) 是一个在SDN控制器上提供多租户虚拟网络的应用。

按照惯例地，巨大的网络系统投资和操作费用是必要的，因为网络是作为每个部门和系统的筒仓被配置。所以，必须为每个租户安装各种网络设备且这些内部boxes将与其他租户隔离，不能与其他租户分享。设计、实施、操作整个复杂的网络是艰巨的。

VTN的独特性在于一个逻辑抽象层面。它使逻辑层面从物理层面中完全的分离，用户在不知道物理网络拓扑或者带宽限制的情况下，能够设计和部署任何想要的网络。

VTN允许用户定义在外观和感觉上与传统L2/3层网络相同的网络。一旦网络在VTN上被设计，它将自动映射到物理网络，然后利用SDN控制协议配置单个交换机。逻辑层面的定义不仅使隐藏底层网络的复杂性变为可能，而且能够更好的管理网络资源。它能够减少重构网络服务的配置时间和最小化网络配置错误。

图28.1 VTN 概述



它的实现主要分为两个部分：

VTN Manager

VTN Coordinator

VTN Manager

OpenDaylight控制器插件与其他模块交互实现VTN模型的组件，提供一个REST接口来配置OpenDaylight控制器中的VTN组件。VTN Manager作为一个到OpenDaylight控制器中的插件被实现，它提供一个REST接口来创建、更新、删除VTN组件。在VTN Coordinator中的用户命令被ODC Driver组件转换成REST API给VTN Manager。除此以外，也提供OpenStack L2层网络功能API的实现。

功能概述

odl-vtn-manage: 提供VTN Manager的JAVA API

odl-vtn-manager-rest: 提供VTN Manager的REST API

odl-vtn-manager-neutron: 提供与Neutron接口的集成

REST API

VTN Manager为虚拟网络功能提供REST API。关于怎样创建一个虚拟租户网络的例子如下：

```
curl --user "admin":"admin" -H "Accept: application/json" -H \
"Content-type: application/json" -X POST \
http://localhost:8282/controller/nb/v2/vtn/default/vtns/Tenant1 \
-d '{"description": "My First Virtual Tenant Network"}'
```

检查所有的租户列表命令如下：

```
curl --user "admin":"admin" -H "Accept: application/json" -H \
"Content-type: application/json" -X GET \
http://localhost:8282/controller/nb/v2/vtn/default/vtns
```

VTN Manager相关REST API文档: <https://jenkins.opendaylight.org/releng/view/vtn/job/vtn-merge-master/lastSuccessfulBuild/artifact/manager/northbound/target/site/wsdocs/rest.html>。

VTN Coordinator

VTN Coordinator是为用户使用VTN虚拟化提供REST接口的外部应用，它与VTN Manager插件交互来实现用户配置，也能够支持多控制业务编排。它在OpenDaylight控制器中（ODC）预先实现虚拟租户网络（VTN）。在OpenDaylight架构中，VTN Coordinator是网络应用、业务编排、服务层的一部分。VTN Coordinator已经作为一个到OpenDaylight控制器的外部应用被实现，这个组件主要负责VTN虚拟化。VTN Coordinator将用VTN Manager扩展的REST接口实现使用OpenDaylight控制器的虚拟网络，使用OpenDaylight APIs（REST）在ODCs中构建虚拟网络，为北向VTN应用提供TEST APIs并且通过协调跨越多个ODCs支持虚拟网络。

VTN Coordinator REST API相关链接：

https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_%28VTN%29:VTN_Coordinator:RestApi。

网络虚拟化功能

用户首先定义一个VTN，然后映射VTN到物理网络，物理网络能够根据VTN的定义来进行通信。随着VTN的定义，L2层和L3层传递功能并基于流的流量控制功能（过滤盒重定向）是可能的。

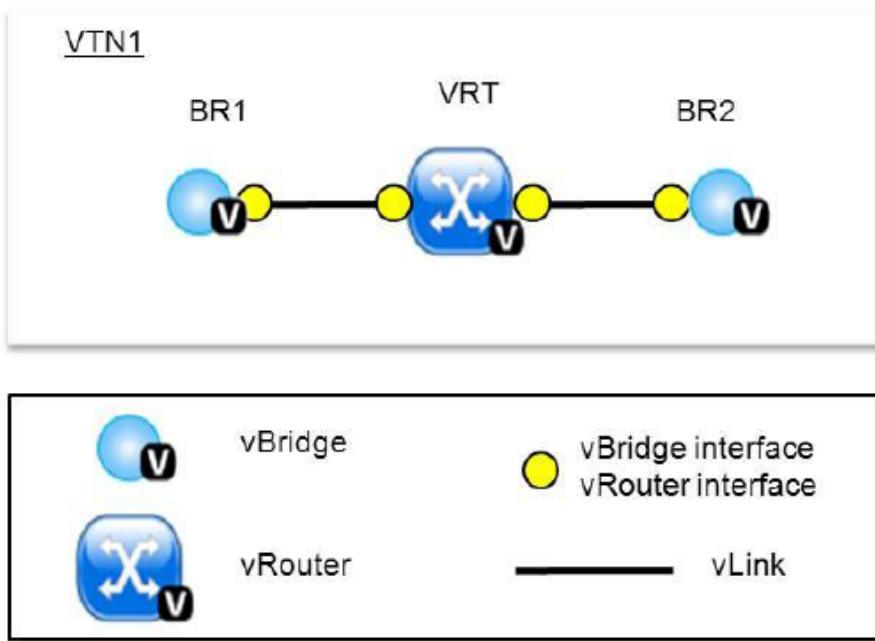
虚拟网络创建

下面的表显示VTN的组成。在VTN中，一个虚拟网络使用虚拟节点（vBridge、vRouter）和虚拟接口以及链路创建组成。通过连接虚拟链路的虚拟节点中创建的虚拟接口，配置L2、L3层传输功能的网络是可能的。

vBridge	L2交换机功能的逻辑表示
vRouter	路由功能的逻辑表示
vTep	隧道终端点 (TEP) 功能的逻辑表示
vTunnel	隧道功能的逻辑表示
vBypass	控制网络间连通性的逻辑表示
虚拟端口	虚拟节点上的endpoint
vLink	虚拟接口间L1层的连通性

下图显示一个已创建的虚拟网络的例子。VRT表示vRouter，BR1、BR2表示vBridges，vRouter和vBridges间的接口通过使用vLinks连接。

图28.2 VTN创建



物理网络资源映射

映射物理网络资源到已创建的虚拟网络，映射被OpenFlow交换机传输或者接收的每个包定义的具体虚拟网络，或者在OpenFlow交换机里传输或接收包的具体端口。主要有两种映射方法，当一个包从OpenFlow交换机被接收到时，端口映射 搜索相应的映射的定义，然后VLAN映射被查找，根据第一个匹配映射，这个包被相关的vBridge映射。

端口映射 映射物理网络资源到vBridge的一个端口用传入L2层网络帧的Switch ID、Port ID和VLAN ID

VLAN映射 映射物理网络资源到vBridge用传入L2层网络帧的VLAN ID。
映射特定交换机的物理资源到vBridge用传入L2层网络帧的Switch ID和VLAN ID。

MAC映射 映射物理资源到vBridge的端口用传入L2层网络帧的MAC地址（最初的贡献不包括这种方法）。

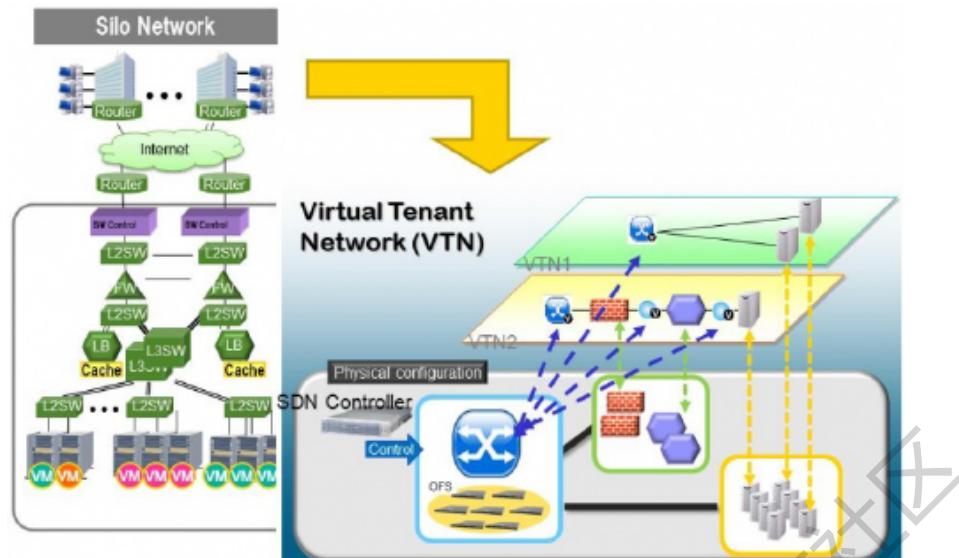
VTN可以学习从连接到一个被映射到VTN交换机的终端信息。进一步地，在VTN上参照终端信息是可能的。

学习终端信息。VTN学习所属VTN的终端信息，存储相关交换机端口所连接终端的MAC地址和VLAN ID。

终端信息衰老化。在VTN中，被VTN学习的终端信息，将一直维持到从终端一直发送数据流。如果终端断开与VTN的连接，衰退时间将开始计时，且终端信息将维持到时间超时。

有关映射配置的例子如下图所示。BR1的端口映射到OFS1的端口GBE0/1，是通过用端口映射，接受自OFS1的端口GBE0/1的包被视为BR1端口相应的包。BR2映射到VLAN200通过用VLAN映射，接收自任何交换机任何端口的带有VLAN标签200的包被视为BR2端口相应的包。

图28.3 VTN映射



vBridge功能

根据目的MAC地址，vBridge提供桥bridge功能，传输数据包到预期端口。当目的MAC地址已经学习到时，vBridge查找MAC地址表并传输数据包到相应的虚拟端口；当目的MAC地址没有学习到时，它传输数据包到所有的虚拟端口除了正在接收的端口（就是指泛洪）。MAC地址被学习如下：

MAC学习。vBridge学习到已连接主机的MAC地址，每个接收帧的源MAC地址被映射到正在接收的虚拟端口，且这个MAC地址被存储在MAC地址表中，这个MAC地址表被创建在per-vBridge基础上。

MAC地址老化。只要主机返回ARP应该，存储在MAC地址表中的MAC地址就会被保留；否则当主机断开连接后，MAC仍存在知道老化时间超时。为了可以有vBridge学习MAC地址静态，可以手动注册MAC地址。

流过滤功能

流过滤功能与ACL（访问控制列表Access Control List）相似。它可以允许或禁止只有某一种满足特定条件的数据包的通信。此外，它可以执行重定向处理程序——站点路由，不同于存在的ACL。流过滤也能应用在VTN中vNode的任何一个端口，控制传递接口的数据包。在流过滤器中被指定的匹配条件如下（注：可以指定多个条件的组合）：

源MAC地址（Source MAC address）；

目的MAC地址（Destination MAC address）；

MAC ether类型（MAC ether type）；

VLAN优先级（VLAN Priority）；

源IP地址（Source IP address）；

目的IP地址（Destination IP address）；

DSCP;

IP 协议 (IP Protocol) ;

TCP/UDP 源端口 (TCP/UDP source port) ;

TCP/UDP 目的端口 (TCP/UDP destination port) ;

ICMP类型 (ICMP type) ;

ICMP代码 (ICMP code) 。

动作的类型，可以在匹配的流过滤条件的数据包中应用到下表中。能够仅使这些匹配特定条件的数据包，通过在动作中指定重定向穿过一个特定的服务器。例如，流的路径可以为一个特定终端发送的每个数据包被改变，这取决于目的IP地址。VLAN优先级控制和DSCP标记也被支持。

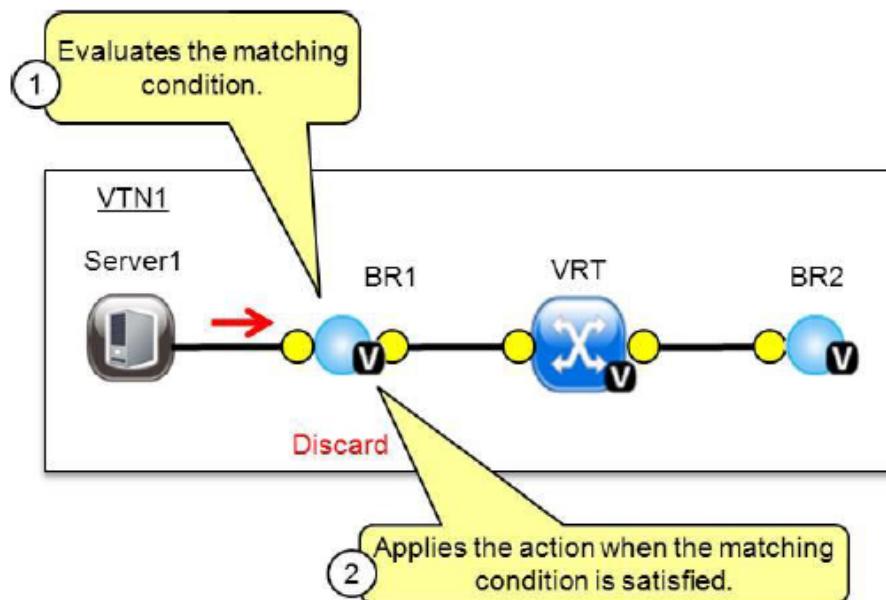
动作	功能
PASS	允许通过匹配指定条件的特定数据包
DROP	丢弃匹配条件的特定数据包
Redirection	重定向数据包到一个需要的虚拟端口。透明重定向（不改变MAC地址）和路由重定向（改变MAC地址）都被支持。

下面的例子显示流过滤功能怎样工作：

当一个在虚拟网络被传送的数据包通过一个虚拟接口时，如果有任何被流过滤器匹配的条件，功能将估算这个匹配条件看传输的数据包是否匹配它。

如果数据包匹配这个条件，功能将应用被流过滤器指定的匹配动作。下面的例子显示配置，功能估算匹配条件在BR1，且如果它匹配这个条件，将丢弃这个数据包。

图28.4 VTN 流过滤



多SDN控制器结合

随着网络的抽象化，VTN能够通过多个SDN控制器配置虚拟网络，提供很高的弹性网络系统。VTN能在每个SDN控制器中被创建，如果用户想在一个策略里管理多个VTNs，这些VTNs可以被集成到一个单一的VTN中。作为一个案例，此功能将被部署到多数据中心环境，即使这些数据中心在地理上是分开的，并被不同的控制器控制，单一策略虚拟网络仍可以通过VTN实现。

很容易将一个新的SDN控制器添加到已存在的VTN中，或者从VTN中删除一个特定的SDN控制器。除此之外，可以定义一个同时涵盖OpenFlow网络和Overlay网络的VTN。流过滤器，被设置在VTN中，在新添加的SDN控制器中将自动应用。

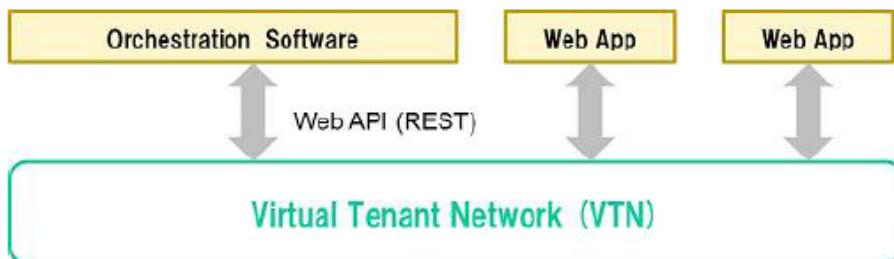
OpenFlow网络和L2/3层网络结合

在一个混合L2/3层交换机的环境里配置VTN也是可行的，L2/3层交换机在VTN中用vBypass显示。流过滤器或者策略不能配置vBypass。但是不管怎么样，vBypass可以作为VTN内部的一个虚拟节点。

虚拟租户网络（VTN）API

VTN提供WEB API，被应用在REST架构中，且提供入口到被URI定义VTN的源。用户可在XML或JSON格式中通过HTTPS通信发送消息到VTN对虚拟网络资源（例如vBridge或虚拟路由器）进行操作，如GET/PUT/POST/DELETE。

图28.5 VTN API



VTN为各种网络资源提供如下操作：

SDNLAB.com
OpenDaylight中文社区

SDNLAB.com
OpenDaylight中文社区

资源	GET	POST	PUT	DELETE
VTN	是	是	是	是
vbridge	是	是	是	是
vRouter	是	是	是	是
vTep	是	是	是	是
vTunnel	是	是	是	是
vBypass	是	是	是	是
vLink	是	是	是	是
接口	是	是	是	是
端口映射	是	否	是	是
Vlan映射	是	是	是	是
流过滤 (ACL/重定向)	是	是	是	是
控制器信息	是	是	是	是
物理拓扑信息	是	否	否	否
警告信息	是	否	否	否

以下是创建一个虚拟网络的示例：

创建VTN：

```
curl --user admin:adminpass -X POST -H 'content-type: application/json' \
-d '{"vtn":{"vtn_name":"VTN1"}}' http://172.1.0.1:8083/vtn-webapi/vtns.json
```

创建控制器信息：

```
curl --user admin:adminpass -X POST -H 'content-type: application/json' \
-d '{"controller": {"controller_id":"CONTROLLER1","ipaddr":"172.1.0.1",
"type":"odc","username":"admin", \
"password":"admin","version":"1.0"}}' http://172.1.0.1:8083/vtn-webapi/
controllers.json
```

在VTN中创建vBridge：

```
curl --user admin:adminpass -X POST -H 'content-type: application/json' \
-d '{"vbridge": {"vbr_name": "VBR1", "controller_id": "CONTROLLER1",
"domain_id": "(DEFAULT)"} }' \
http://172.1.0.1:8083/vtn-webapi/vtns/VTN1/vbridges.json
```

在vBridge下创建端口：

```
curl --user admin:adminpass -X POST -H 'content-type: application/json' \
-d '{"interface": {"if_name": "IF1"} }' http://172.1.0.1:8083/vtn-webapi/vtns/
VTN1/vbridges/VBR1/interfaces.json
```

VTN OpenStack配置

指南描述怎样建议OpenDaylight控制器和OpenStack的集成。

当OpenDaylight控制器提供多种方式来集成OpenStack时，该指南主要集中于使用OpenDaylight控制器中可用的VTN功能来与OpenStack集成。在此集成中，VTN Manager为OpenStack作为网络服务提供者工作。

VTN Manager功能在一个纯粹的OpenFlow环境中（所有数据平面的交换机是OpenFlow交换机）启用OpenStack。

要求

用OpenDaylight控制器（ODL）作为OpenStack的网络服务提供者。

组件

OpenDaylight控制器

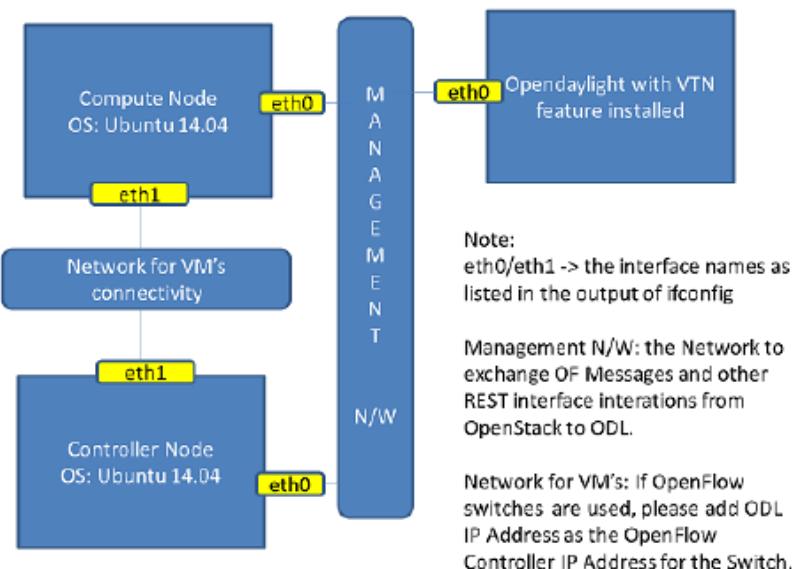
OpenStack控制节点

OpenStack计算节点

OpenFlow交换机，如Mininet（不是强制的）

VTN功能支持多个OpenStack节点，能够部署多个OpenStack计算节点。在管理层面，OpenDaylight控制器、OpenStack节点和OpenFlow交换机应该能够相互通信。在数据层面，运行在OpenStack节点上的Open vSwitch通过物理或者逻辑的OpenFlow交换机互相之间能够互相通信。核心的OpenFlow交换机不是强制的，因此，能够直接连接Open vSwitch。

图28.6 LAB 建立



注意：Ubuntu14.04系统可以被用在节点中，且Vsphere来集中管理。

配置

服务器准备

在两台服务器上安装Ubuntu 14.04 LTS系统（OpenStack控制节点和计算节点）

安装Ubuntu时，要求安装一个用户，我们创建用户“stack”（我们使用相同的用户运行devstack）。注：也可以有多个计算节点。提示：请做最小的安装来避免devstack中的故障问题。

用户设置——登录到两台服务器-禁用Ubuntu防火墙

```
sudo ufw disable
```

选择性地安装这些包

```
sudo apt-get install net-tools
```

编辑： sudo vim /etc/sudoers，添加如下条目

```
stack ALL=(ALL) NOPASSWD: ALL
```

网络设置——检查ifconfig -a的输出，eth0和eth1两个接口在将被列出。——eth0接口已经连接到ODL控制器可达的网络中。——两台服务器的eth1接口连接到不同的网络，充当数据平面（使用OpenStack创建虚拟机）。——手动编辑文件： sudo vim /etc/network/interfaces，做如下修改设置：

```
stack@ubuntu-devstack:~/devstack$ cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).
# The loop-back network interface
auto lo
iface lo inet loopback
# The primary network interface
auto eth0
iface eth0 inet static
address <IP ADDRESS TO REACH ODL>
```

```
netmask <NET_MASK>
broadcast <BROADCAST_IP_ADDRESS>
gateway <GATEWAY_IP_ADDRESS>
auto eth1
iface eth1 inet static
address <IP_ADDRESS_UNIQ>
netmask <NETMASK>
```

注：确保eth0接口为默认路由，可以到达ODL_IP_ADDRESS NOTE: eth1的条目是非强制性的，如果不设置的话，我们需要在stacking全面激活以后手动设置"ifup eth1"。

完成——在用户和网络将设置的网络参数应用于网络之后，重启两个节点——再次登录并检查ifconfig的输出，确保两个接口都列出。

ODL设置和执行

vtn.ini

VTN使用vtn.ini文件中的配置参数用于OpenStack的集成。

在所有的OpenStack节点，都要为Open vSwitch设置这些值。

配置文件vtn.ini需要在configuration目录下手动创建。

vtn.ini的内容如下：

```
bridgename=br-int portname=eth1 protocols=OpenFlow13 failmode=secure
```

配置参数的值一定要基于用户的环境做相应的修改。

尤其是"portname"应该仔细配置，因为如果这个值配错了，OpenDaylight控制器就不能转发数据包。

其他的参数作为通常用例正常工作。

•bridgename

- Open vSwitch中bridge的名字，由OpenDaylight控制器创建。

- 这个名字必须是"br-int"。

•portname

- 端口名，在Open vSwitch的vbridge中创建。

- 这个名字和OpenStack节点接口的名字相同，用于在数据平面与OpenStack节点连接。（在使用案例中，名字是eth1）

- 默认情况下，如果vtn.ini没有被创建，VTN的作用portname为ens33。

•protocols

- OpenFlow协议，OpenFlow交换机和控制器之间通信。

- 协议的值可以是OpenFlow13或者OpenFlow10。

•failmode

- failmode的值可以是"standalone"或者"secure"。

- 一般的案例请使用"secure"。

启动ODL控制器

请安装功能odl-vtn-manager-neutron，提供与Neutron接口集成的功能。

```
install odl-vtn-manager-neutron
```

提示：在ODL启动之后，确保ODL控制器监听端口：6633、6653、6640和8080。

确保devstack防火墙中的接口能够与ODL控制器交互。

注：6633/6653 - OpenFlow端口

6640 - OVS管理端口

8282 - REST API端口

Devstack建立

VTN Devstack脚本

local.conf是用户维护配置文件。所有DevStack自定义设置都可以包含在一个单独的文件中。这个文件是严格按序处理的，需要在local.conf文件中设置以下数据：

当检测到不可达时，设置Host_IP。

设置FLOATING_RANGE到一个范围，这个范围不用在本地网络中，如192.168.1.224/27。IP地址的最后一个字节为225-254的用于浮动IPs。

设置FLAT_INTERFACE为以太网接口，将主机连接到本地网络。这个接口可以配置上面提到的IP地址。

如果*_PASSWORD变量没有设置，在执行stack.sh的时候会提示输入你设置的值。

设置ADMIN_PASSWORD。这个密码用于OpenStack用户的admin和demo账号。

设置MYSQL_PASSWORD。默认是16进制的字符串，如果要直接看数据库中的数据，是很不方便的。

设置RABBIT_PASSWORD。两个节点中消息服务用到的密码。

设置服务密码。OpenStack服务使用这个密码（Nova、Glance等）进行身份认证。

DevStack Control

local.conf(control)

```
#IP Details
HOST_IP=<CONTROL_NODE_MANAGEMENT_IF_IP_ADDRESS>#Please Add The Control Node IP
Address in this line
FLAT_INTERFACE=<FLAT_INTERFACE_NAME>
SERVICE_HOST=$HOST_IP
#Instance Details
MULTI_HOST=1
#config Details
RECLONE=yes #Make it "no" after stacking successfully the first time
VERBOSE=True
LOG_COLOR=True
LOGFILE=/opt/stack/logs/stack.sh.log
SCREEN_LOGDIR=/opt/stack/logs
```

```
#OFFLINE=True #Uncomment this after stacking successfully the first time
#Passwords
ADMIN_PASSWORD=labstack
MYSQL_PASSWORD=supersecret
RABBIT_PASSWORD=supersecret
SERVICE_PASSWORD=supersecret
SERVICE_TOKEN=supersecrettoken
ENABLE_TENANT_TUNNELS=false
#Services
disable_service rabbit
enable_service qpid
enable_service quantum
enable_service n-cpu
enable_service n-cond
disable_service n-net
enable_service q-svc
enable_service q-dhcp
enable_service q-meta
enable_service horizon
enable_service quantum
enable_service tempest
ENABLED_SERVICES+=, n-api, n-crt, n-obj, n-cpu, n-cond, n-sch, n-novnc, n-cauth, ncauth,
nova
ENABLED_SERVICES+=, cinder, c-api, c-vol, c-sch, c-bak
#ML2 Details
Q_PLUGIN=ml2
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openaylight
Q_ML2_TENANT_NETWORK_TYPE=local
Q_ML2_PLUGIN_TYPE_DRIVERS=local
disable_service n-net
enable_service q-svc
enable_service q-dhcp
enable_service q-meta
enable_service neutron
enable_service odl-compute
ODL_MGR_IP=<ODL_IP_ADDRESS> #Please Add the ODL IP Address in this line
OVS_PHYSICAL_BRIDGE=br-int
Q_OVS_USE_VETH=True
url=http://<ODL_IP_ADDRESS>:8080/controller/nb/v2/neutron #Please Add the ODL
IP Address in this line
username=admin
password=admin
```

DevStack Compute

local.conf(compute)

```
#IP Details
HOST_IP=<COMPUTE_NODE_MANAGEMENT_IP_ADDRESS> #Add the Compute node Management
IP Address
SERVICE_HOST=<CONTROLLER_NODE_MANAGEMENT_IP_ADDRESS> #Add the control Node
Management IP Address here
#Instance Details
MULTI_HOST=1
#config Details
RECLONE=yes #Make this "no" after stacking successfully once
#OFFLINE=True #Uncomment this line after stacking successfully first time.
VERBOSE=True
LOG_COLOR=True
LOGFILE=/opt/stack/logs/stack.sh.log
```

```

SCREEN_LOGDIR=/opt/stack/logs
#Passwords
ADMIN_PASSWORD=labstack
MYSQL_PASSWORD=supersecret
RABBIT_PASSWORD=supersecret
SERVICE_PASSWORD=supersecret
SERVICE_TOKEN=supersecrettoken
#Services
ENABLED_SERVICES=n-cpu, rabbit, neutron
#ML2 Details
Q_PLUGIN=ml2
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openaylight
Q_ML2_TENANT_NETWORK_TYPE=local
Q_ML2_PLUGIN_TYPE_DRIVERS=local
enable_service odl-compute
ODL_MGR_IP=<ODL_IP_ADDRESS> #ADD ODL IP address here
OVS_PHYSICAL_BRIDGE=br-int
ENABLE_TENANT_TUNNELS=false
Q_OVS_USE_VETH=True
#Details of the Control node for various services
[[post-config|/etc/neutron/plugins/ml2/ml2_conf.ini]]
Q_HOST=$SERVICE_HOST
MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
KEYSTONE_AUTH_HOST=$SERVICE_HOST
KEYSTONE_SERVICE_HOST=$SERVICE_HOST
NOVA_VNC_ENABLED=True
NOVNCPROXY_URL="http://<CONTROLLER_NODE_IP_ADDRESS>:6080/vnc_auto.html" #Add
Controller Node IP address
VNCSERVER_LISTEN=$HOST_IP
VNCSERVER_PROXYCLIENT_ADDRESS=$VNCSERVER_LISTEN

```

注：我们不得不在local.conf文件中注释OFFLINE=TRUE，这样的话，所有的安装会自动生成。只有当我们从scratch中设置DevStack环境的时候OFFLINE=TRUE。

获取Devstack（所有节点）

安装git应用程序使用

```
sudo apt-get install git get devstack git clone https://git.openstack.org/openstack-dev/
devstack;
```

切换到stable/Juno版本分支

打开devstack git，检查stable/juno

Stack Control节点

local.conf: DevStack Control

```
cd devstack in the controller node
```

拷贝local.conf（devstack control节点）的内容，以文件名"local.conf"保存在devstack中。

根据需求修改IP地址的值。

运行stack

```
./stack.sh
```

验证控制节点堆栈

stack.sh打印出的Horizon可以在如下的URL中访问到: <http://:8080/>。

在控制节点终端运行命令`sudo ovs-vsctl show`, 验证你创建的bridge br-int。

Stack计算节点

local.conf: DevStack Compute。

```
cd devstack in the controller node
```

拷贝local.conf (devstack compute节点) 的内容, 以名字"local.conf"保存在devstack中。

根据需求修改IP地址的值。

运行stack

```
./stack.sh
```

验证计算节点堆栈

stack.sh在你的主机ip: 上

在控制节点终端运行命令`sudo ovs-vsctl show`, 验证你创建的bridge br-int。

`ovs-vsctl show`命令输出和控制节点的相似。

stacking所有的节点以后, 看ODL DLUX GUI: <http://:8181/dlux/index.html>, 交换机、拓扑和端口当前是确定的。

提示:如果OVS中的相互联系没有显示, 请使用以下命令为数据平面提供接口。

```
ifup <interface_name>
```

提示: OVS的一些版本在存在table-miss的时候会丢包, 所以请在所有的节点上添加以下流, 所有的OVS版本(≥ 2.1) :

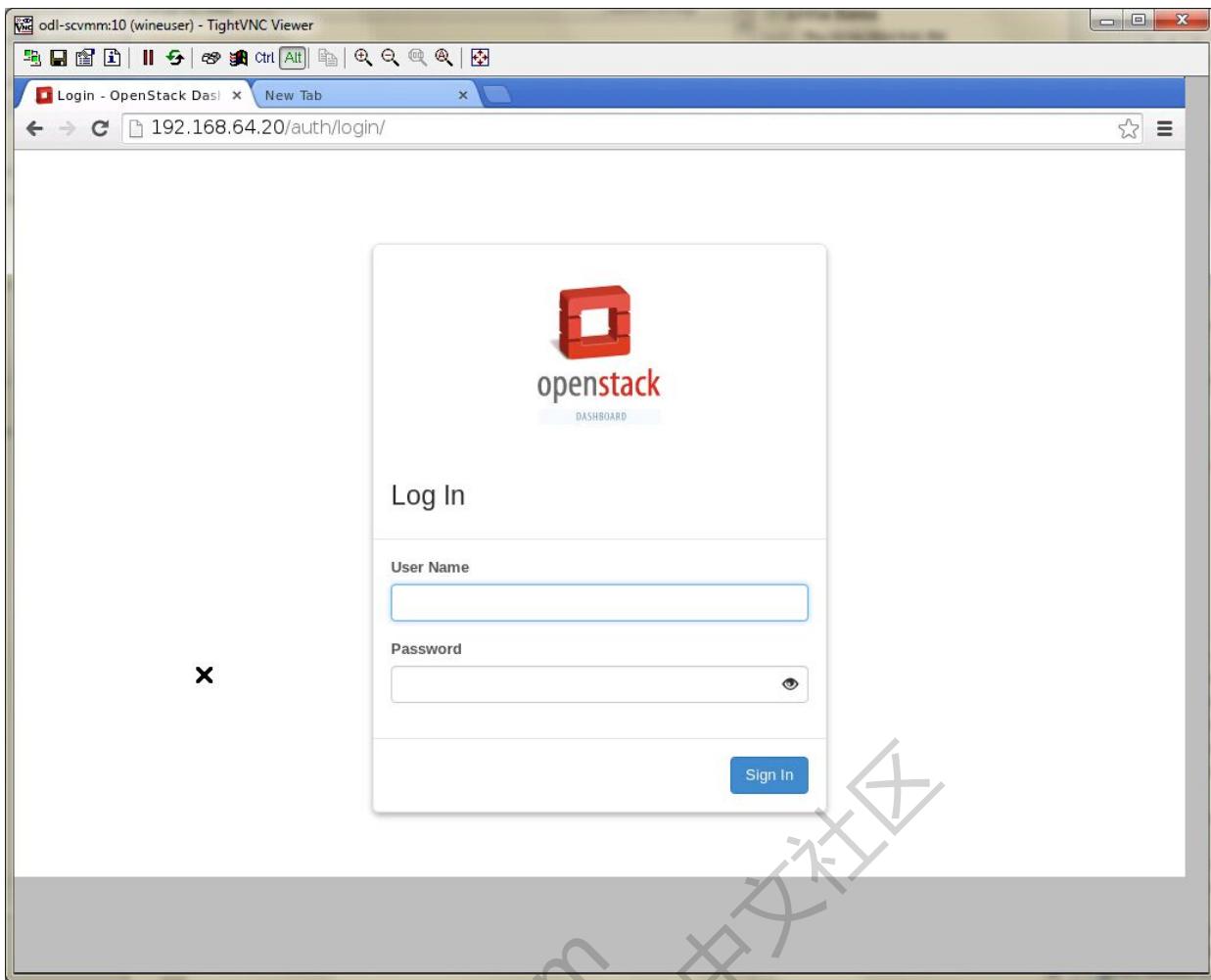
```
ovs-ofctl --protocols=OpenFlow13 add-flow br-int  
priority=0,actions=output:CONTROLLER
```

请在网络互联中接受混杂模式。

为Devstack Horizon GUI创建VM

登录<http://:8080/>, 检查horizon GUI。

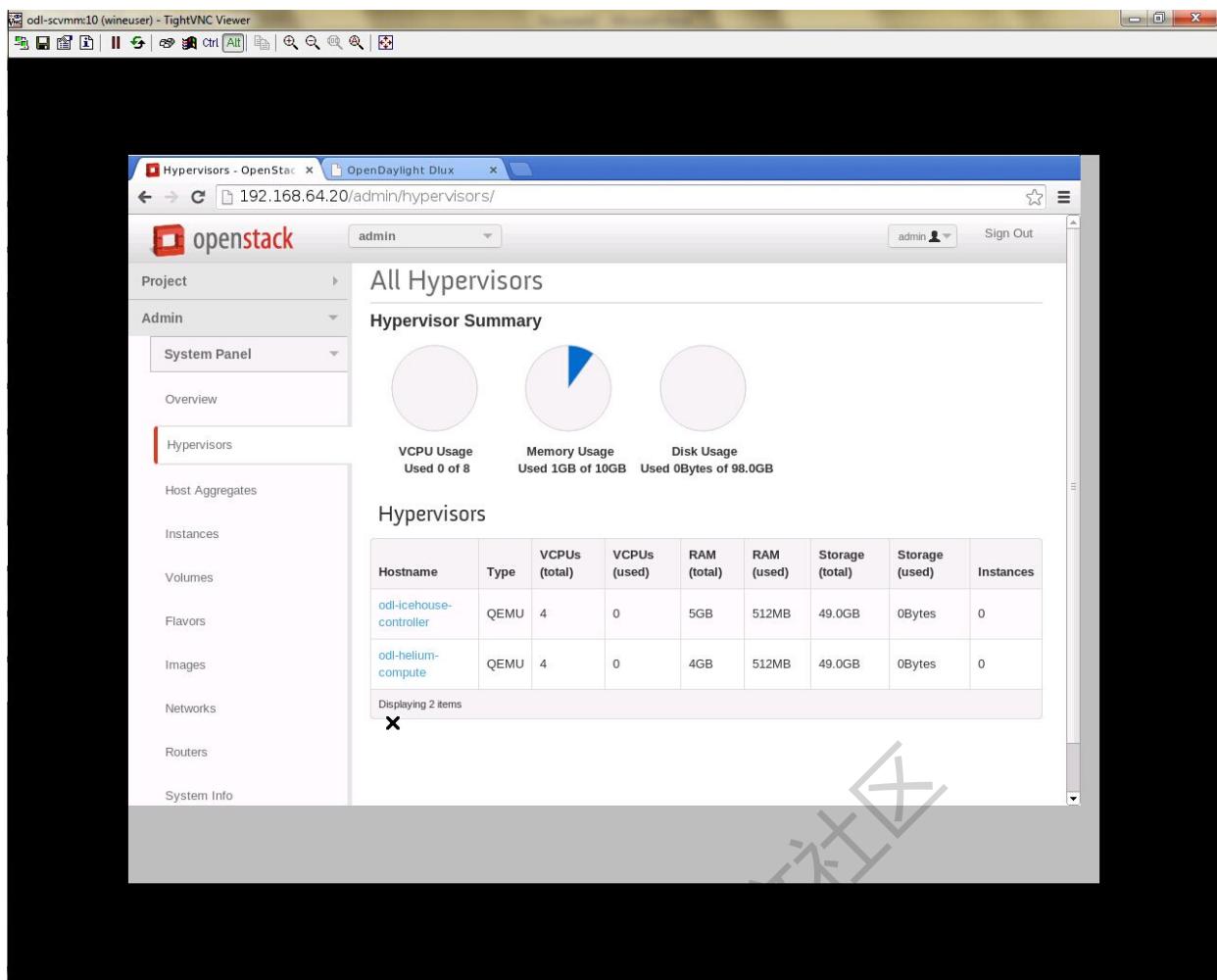
图28.7 Horizon GUI



用户名/密码为admin/labstack。

我们首先要确保hypervisors通过点击Hypervisors选项能够映射（控制节点和计算节点）。

图28.8 Hypervisors

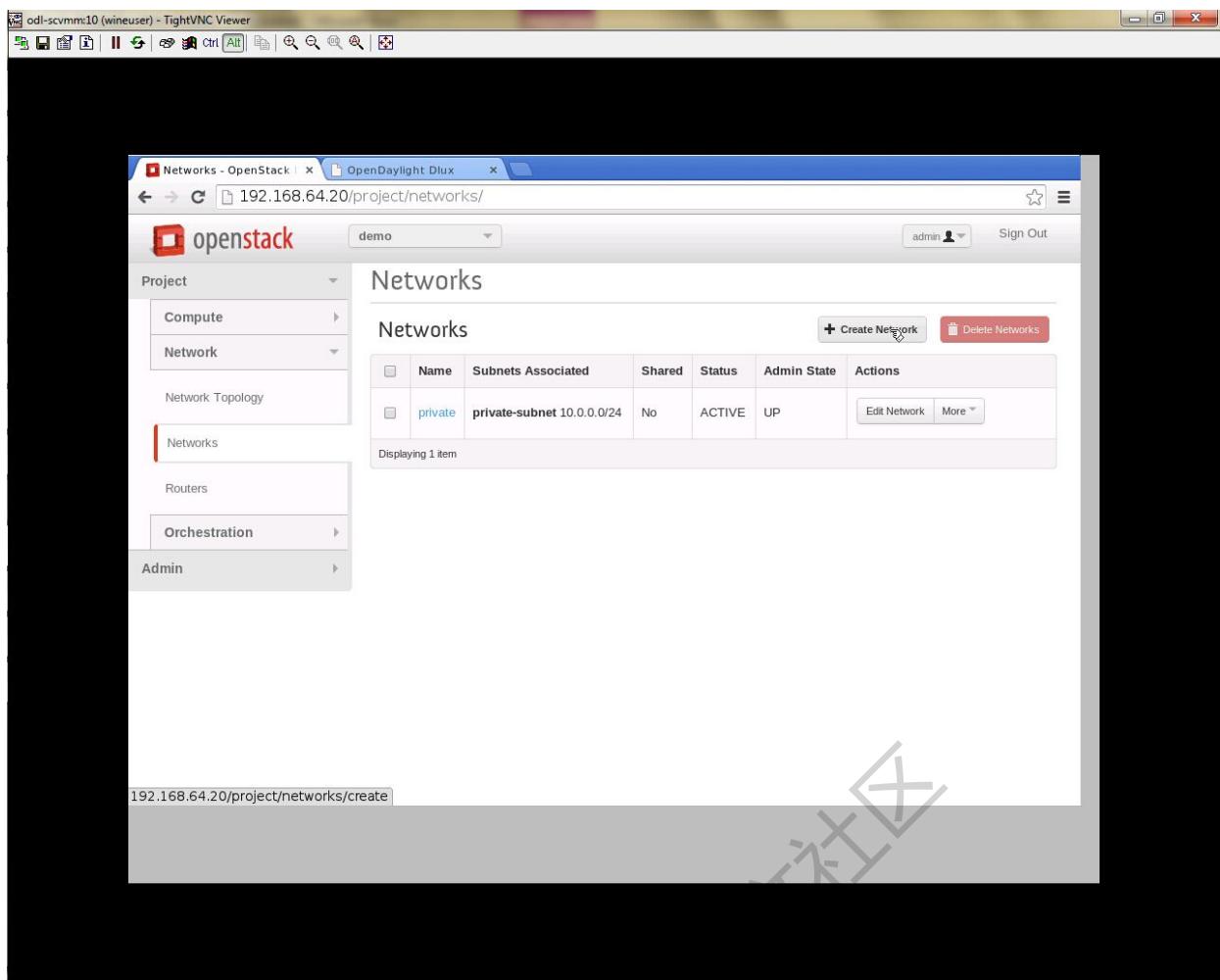


在Horizon GUI上创建新的网络

点击Networks

点击Create Network按钮

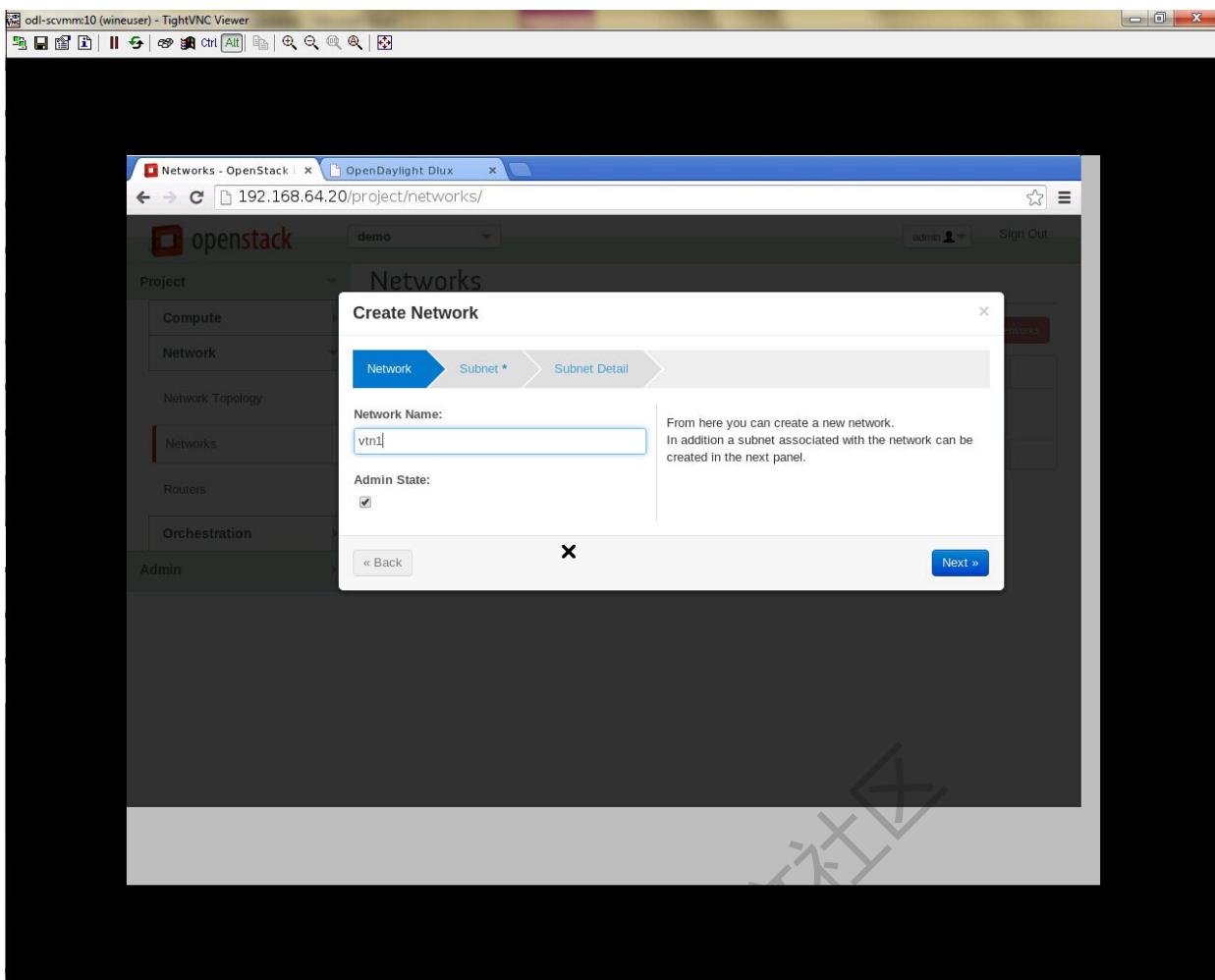
图28.9 创建网络



将出现一个弹窗

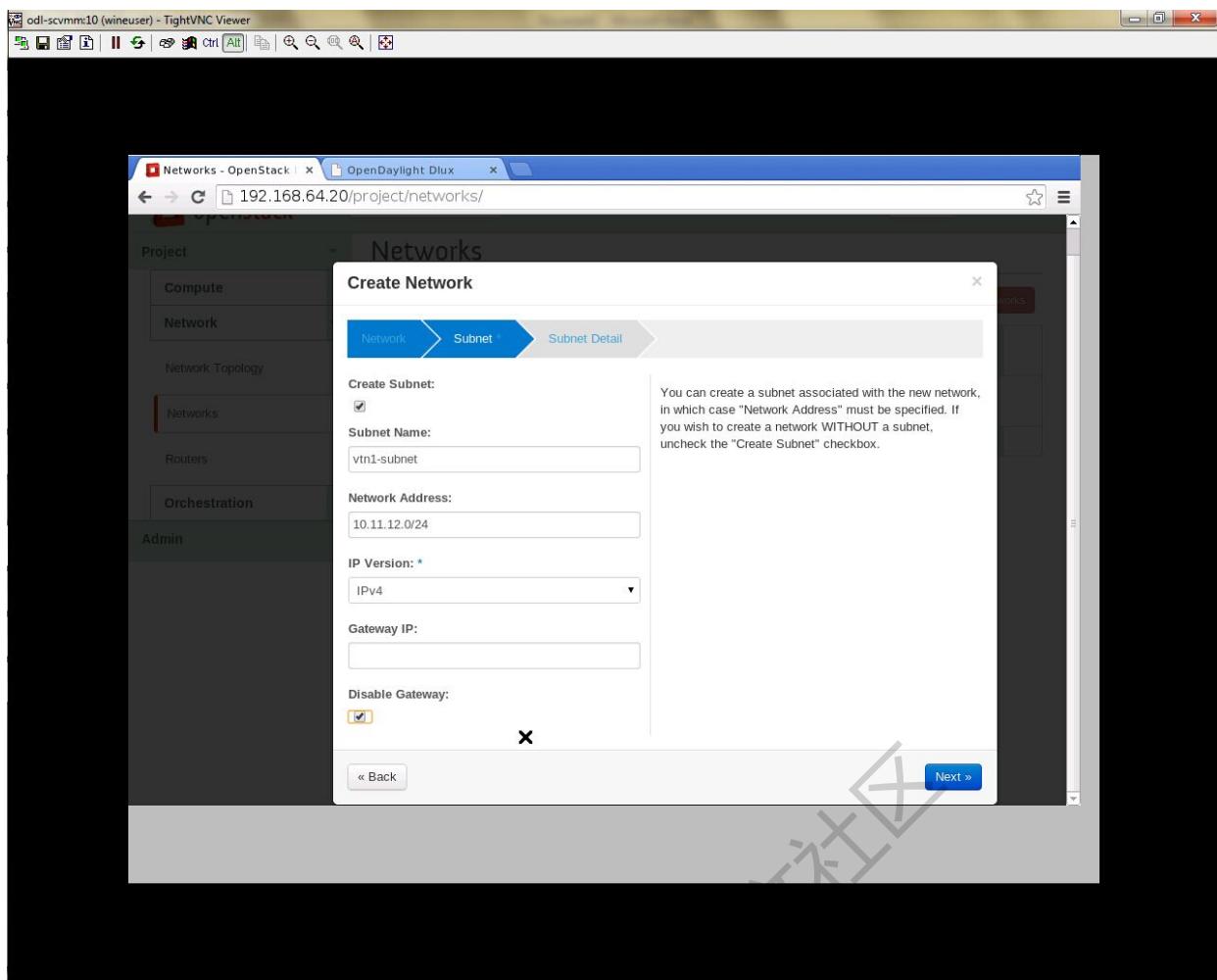
进入网络，点击Next按钮

图28.10 步骤1



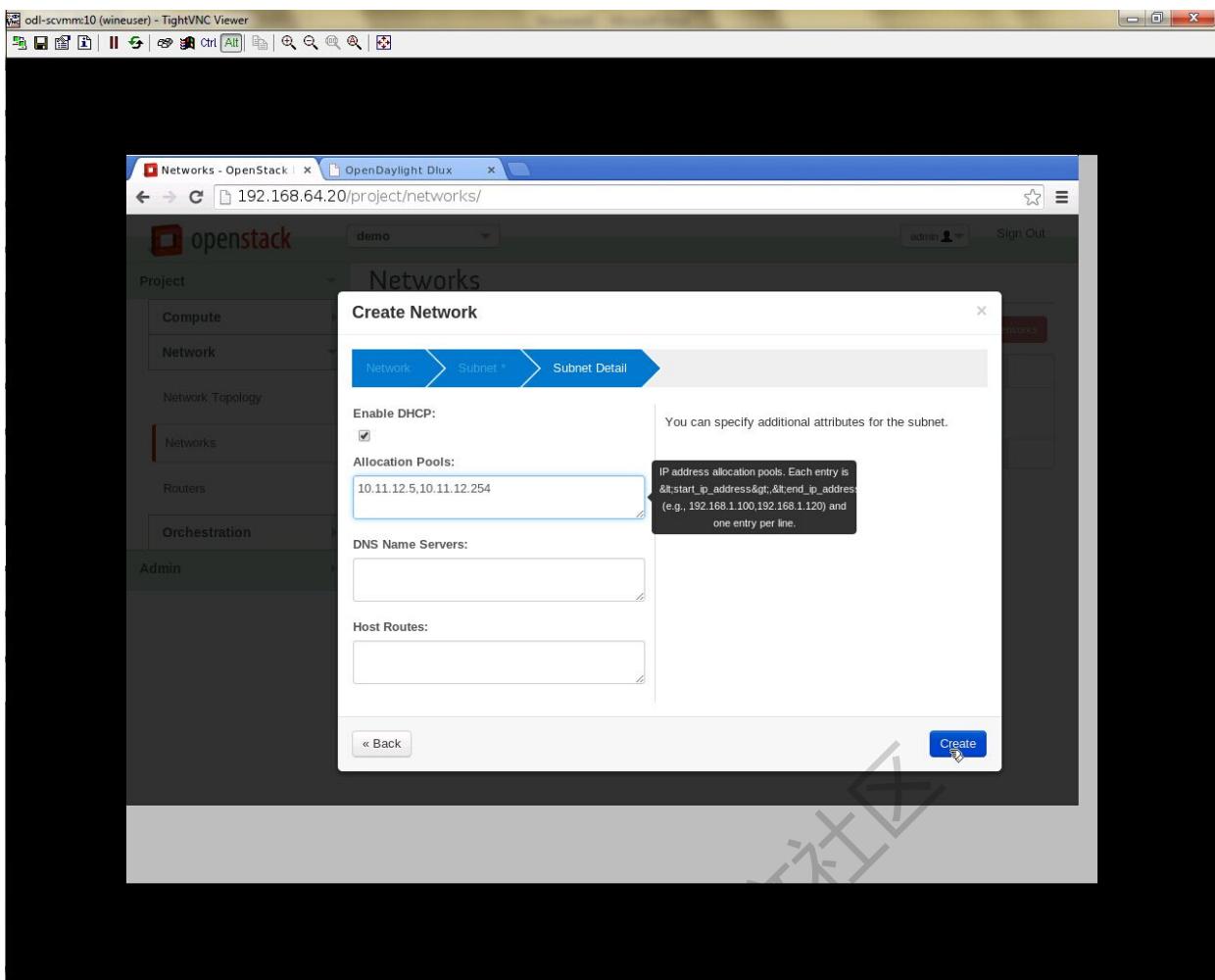
通过给出网络地址来创建一个子网络，点击Next按钮

图28.11 步骤2



指定子网额外的详细信息

图28.12 步骤3

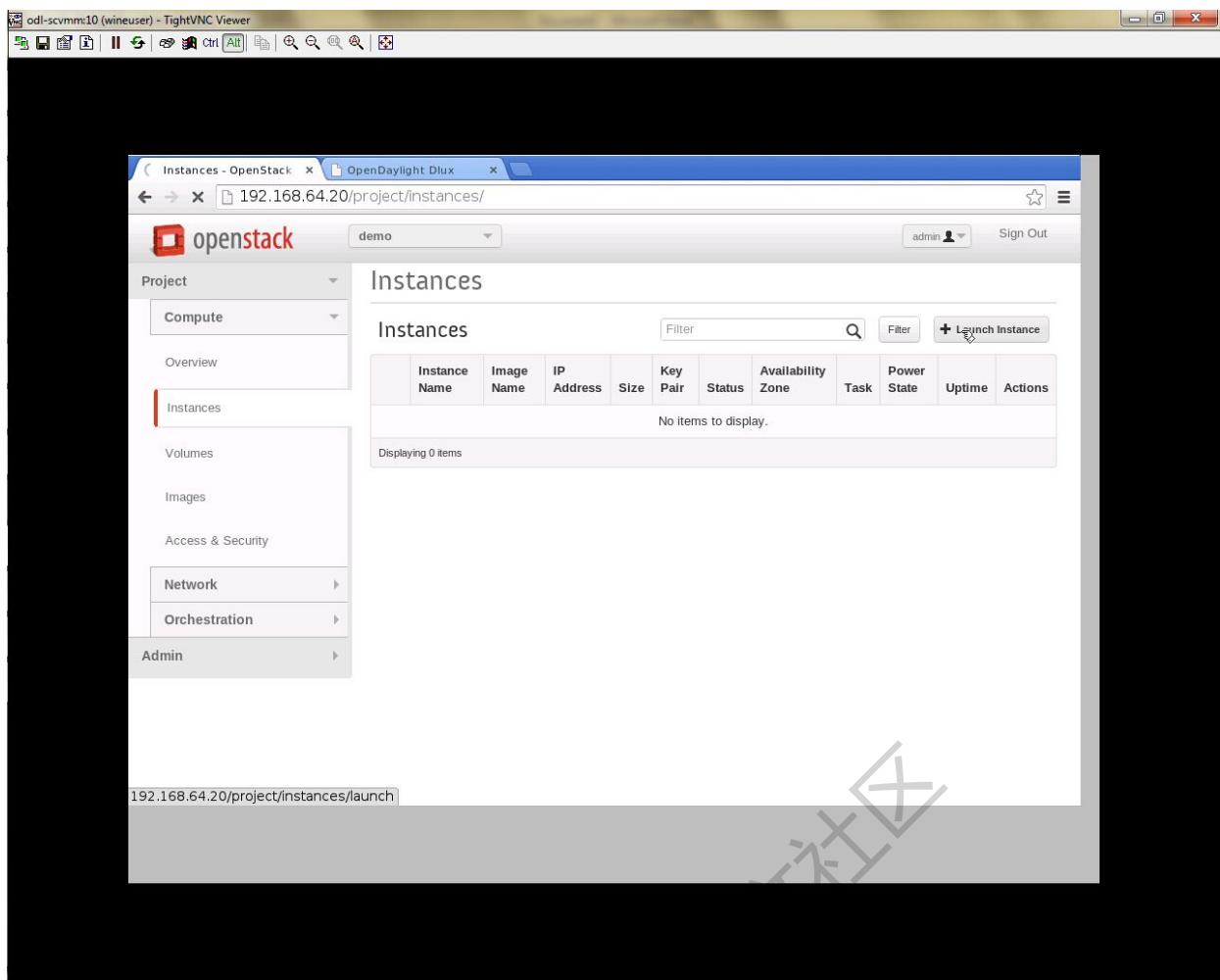


点击Create按钮

创建VM实例

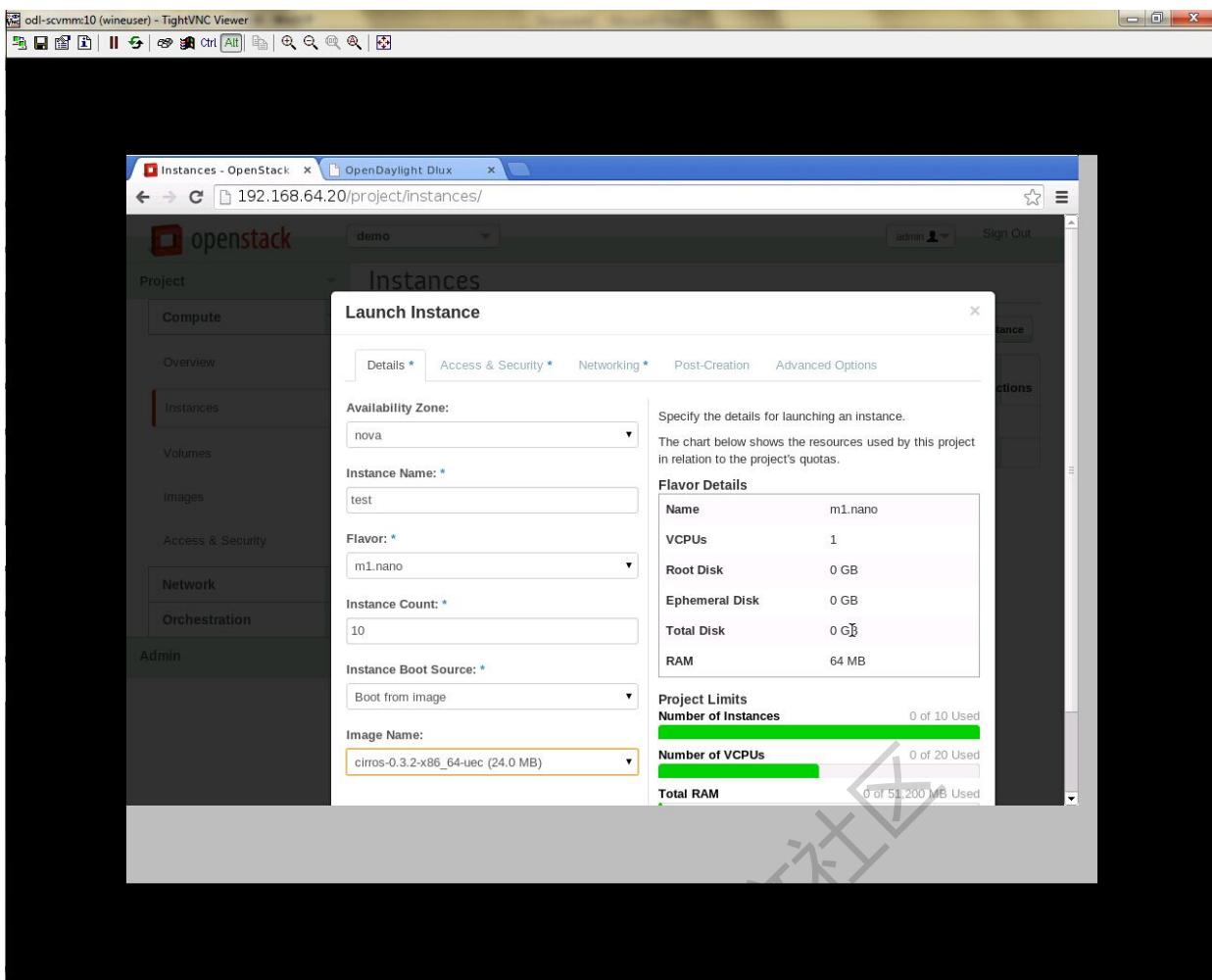
在GUI中切换到instances选项

图28.13 实例创建



点击Launch Instances按钮

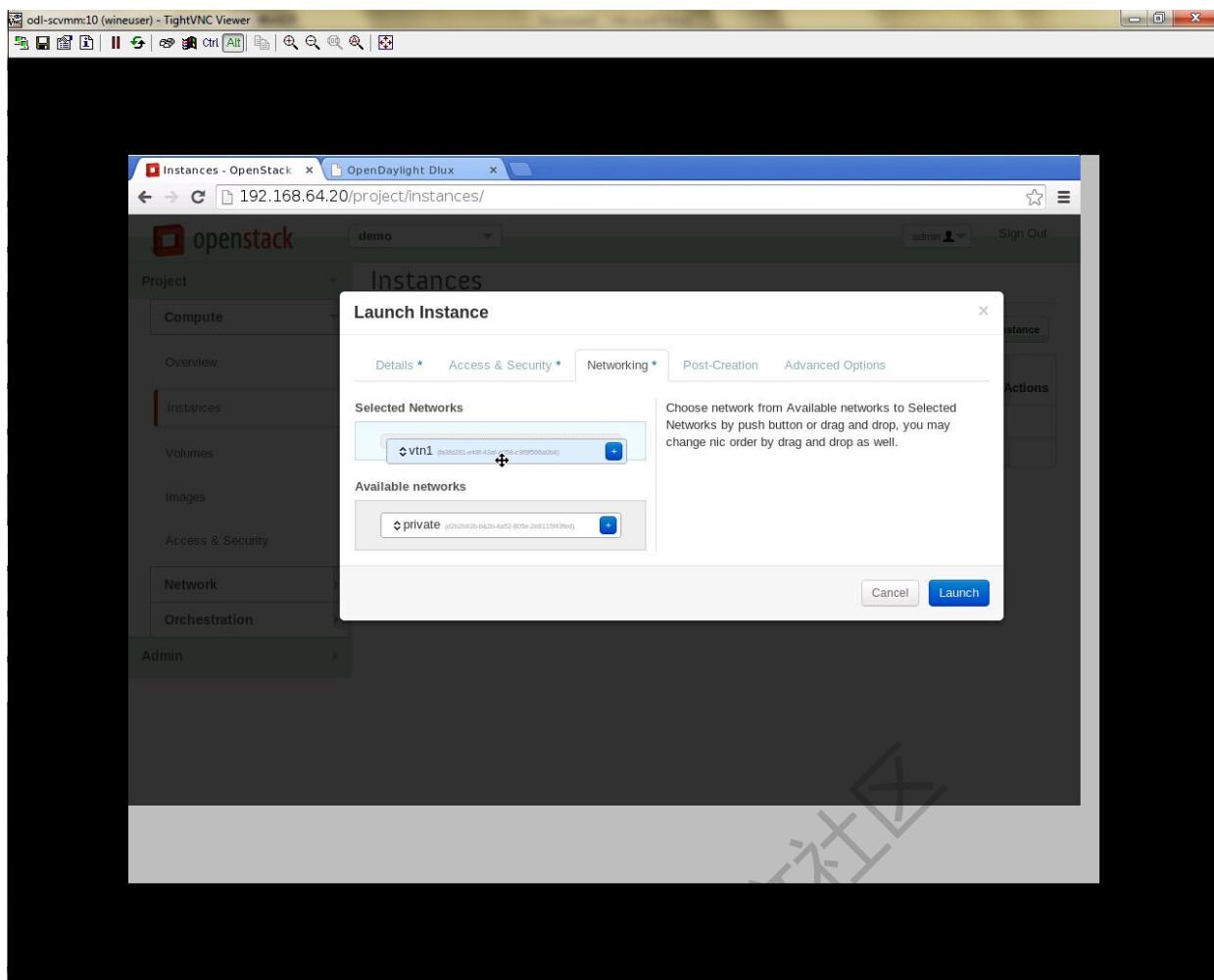
图28.14 启动实例



点击Details，进入VM的详情面板。在这个demo中，我们创建了10台VM（实例）

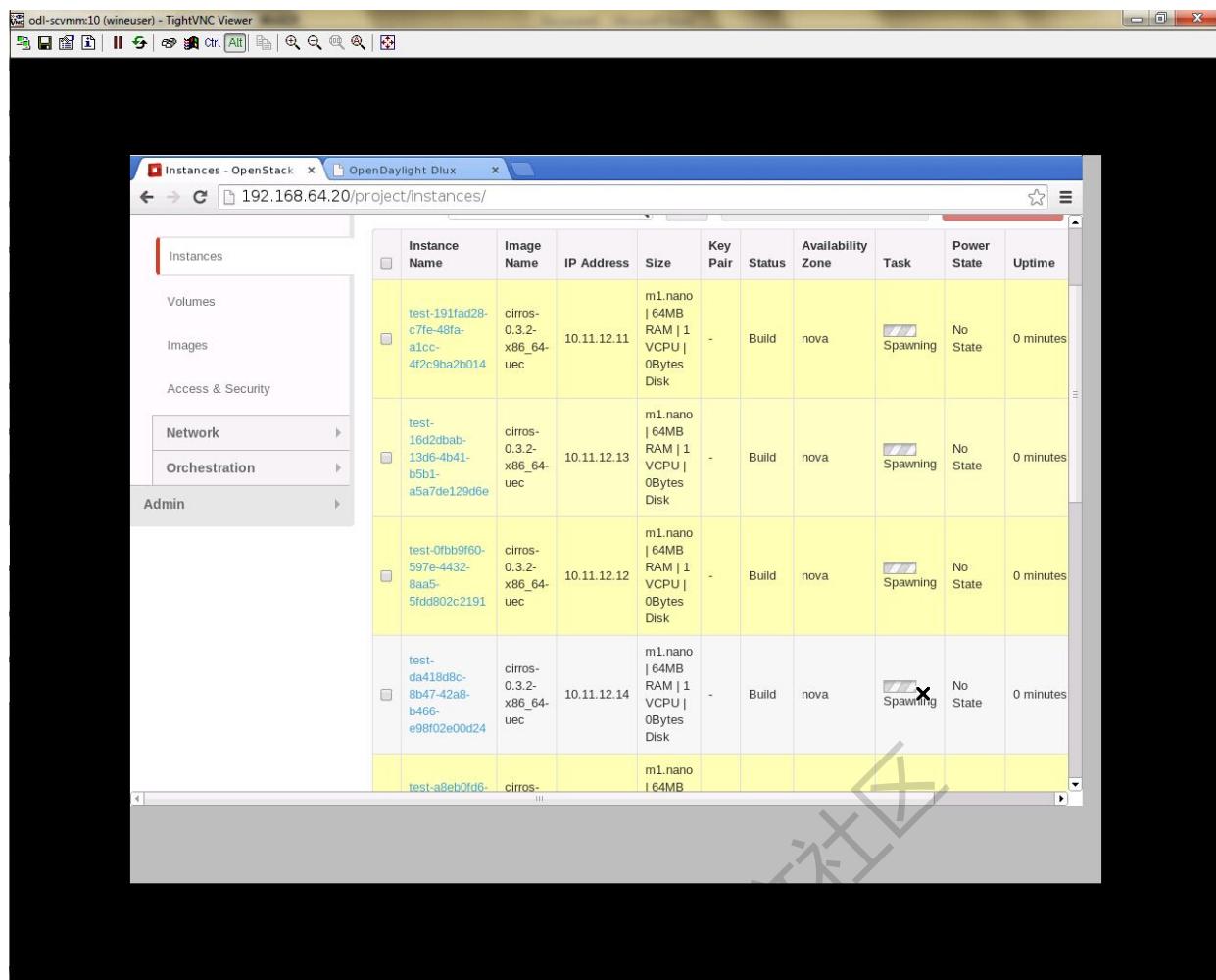
在Networking选项中，必须选择network，为此需要拖拽可用的网络到Selected Networks中，拖拽vtn1，点击Launch创建实例。

图28.15 启动网络



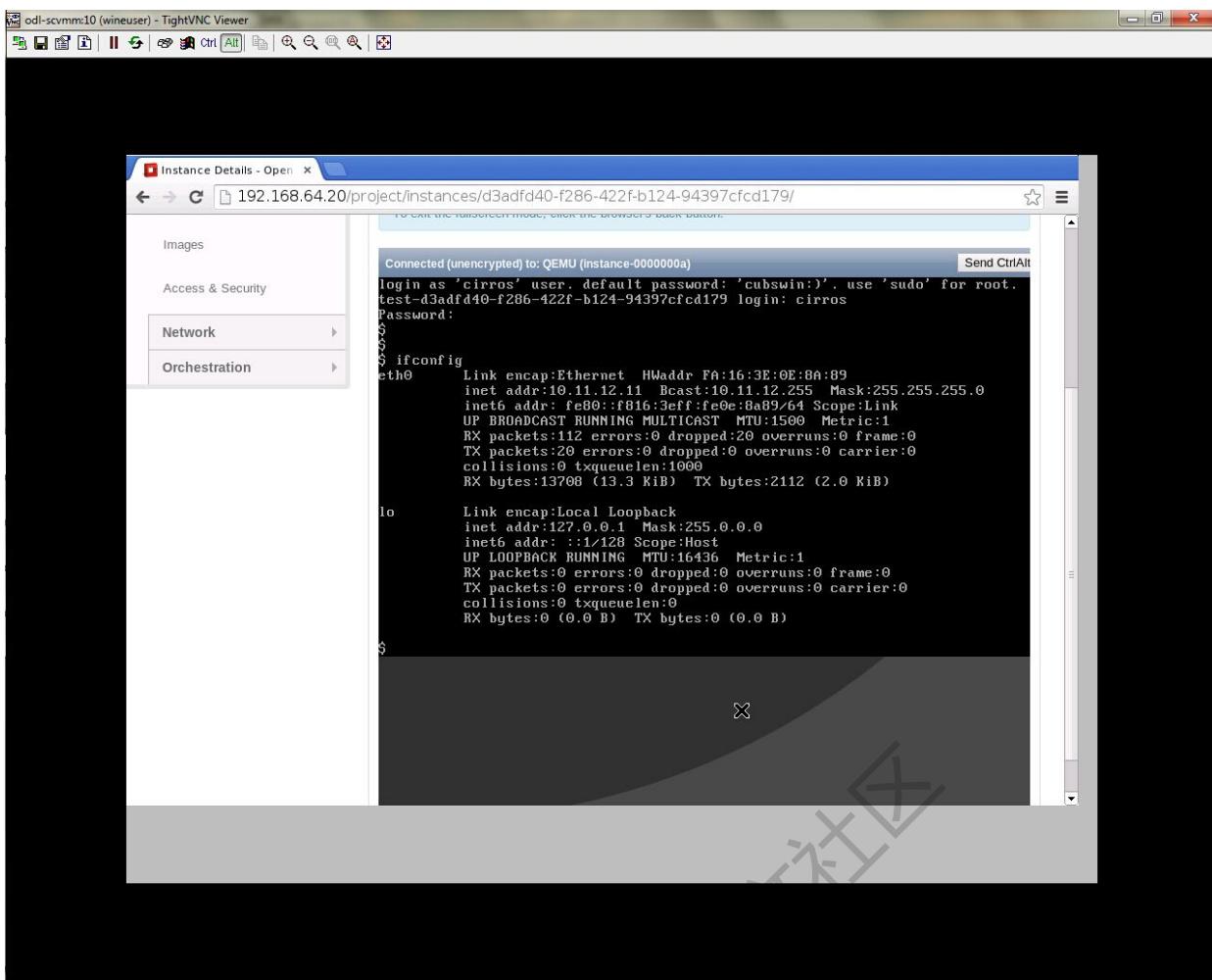
10台VM将被创建

图28.16 加载所有的实例



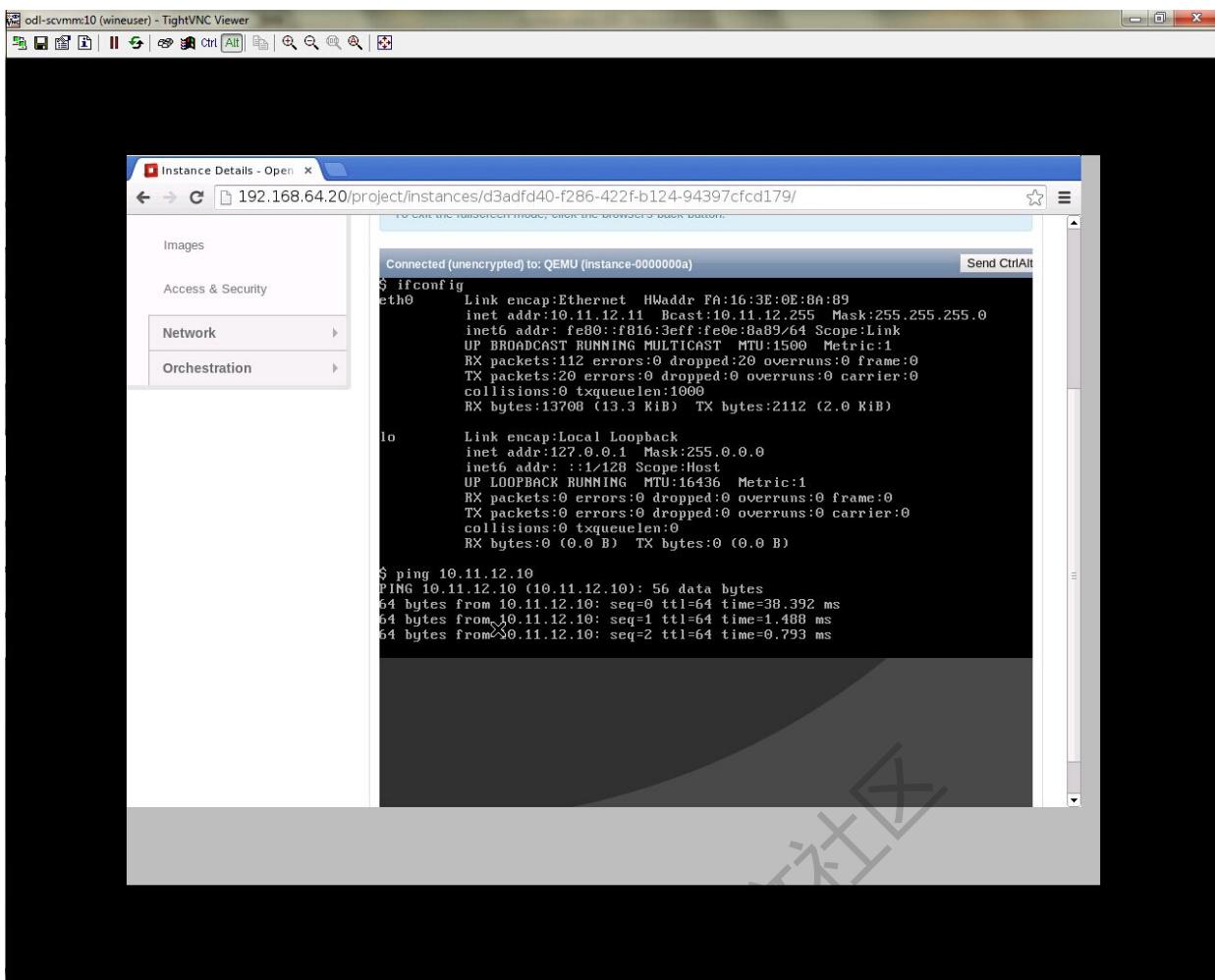
点击实例中任意的VM，点击Console选项

图28.17 实例控制台



登录到VM控制台，用ping命令验证连通性

图28.18 Ping



VM创建后验证控制节点和计算节点

VM创建后，`sudo ovs-vsctl`命令输出：

```

[stack@icehouse-compute-odl devstack]$ sudo ovs-vsctl show Manager
"tcp:192.168.64.73:6640"
is_connected: true
Bridge br-int
Controller "tcp:192.168.64.73:6633"
is_connected: true
fail_mode: secure
Port "tapa2e1ef67-79"
Interface "tapa2e1ef67-79"
Port "tap5f34d39d-5e"
Interface "tap5f34d39d-5e"
Port "tапc2858395-f9"
Interface "tапc2858395-f9"
Port "tapa9ea900a-4b"
Interface "tapa9ea900a-4b"
Port "tапc63ef3de-53"
Interface "tапc63ef3de-53"
Port "tap01d51478-8b"
Interface "tap01d51478-8b"
Port "tapa0b085ab-ce"
Interface "tapa0b085ab-ce"
Port "tapeab380de-8f"
Interface "tapeab380de-8f"
Port "tape404538c-0a"
Interface "tape404538c-0a"
Port "tap2940658d-15"

```

```

Interface "tap2940658d-15"
Port "ens224"
Interface "ens224"
ovs_version: "2.3.0"
<code>[stack@icehouse-controller-odl devstack]$ sudo ovs-vsctl show
Manager "tcp:192.168.64.73:6640"
is_connected: true
Bridge br-int
Controller "tcp:192.168.64.73:6633"
is_connected: true
fail_mode: secure
Port "tap71790d18-65"
Interface "tap71790d18-65"
Port "ens224"
Interface "ens224"
ovs_version: "2.3.0"

```

注：在上面的情境中，计算节点创建了多个节点。

参考文献

<http://devstack.org/guides/multinode-lab.html>

https://wiki.opendaylight.org/view/File:Vtn_demo_hackfest_2014_march.pdf

VTN使用案例

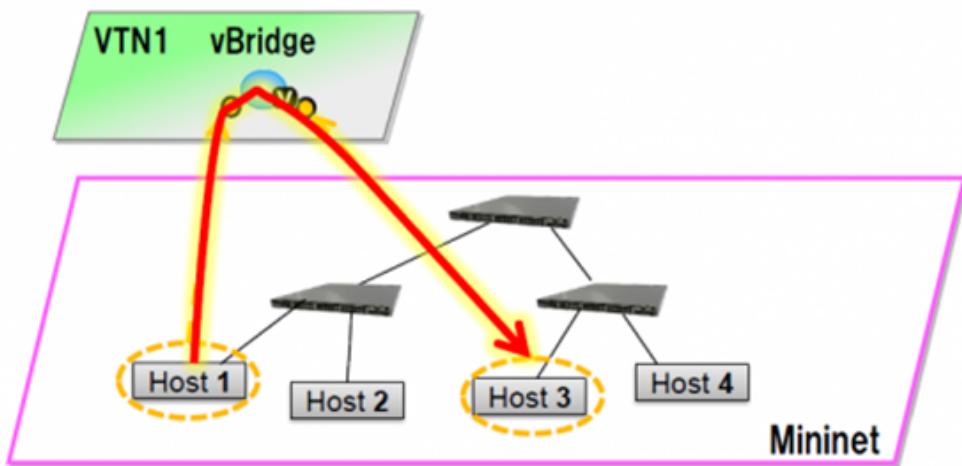
如何在单个控制器上配置2层网络

概述

本例演示使用VTN虚拟化技术（单个控制器）的2层网络中VTN Coordinator的配置。主要示例是使用mininet做单个控制器vBridge接口映射。mininet相关信息和配置参见URL：

https://wiki.opendaylight.org/view/OpenDaylight_Controller:Installation→Using_Mininet

图28.19 单个控制器示例演示



需求

配置mininet，创建拓扑，

```
mininet@mininet-vm:~$ sudo mn --controller=remote,ip= --topo
tree,2

mininet> net

s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth1
s2 lo: s2-eth1:s1-eth2 s2-eth2:h2-eth0
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth2
```

配置

创建一个控制器

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -
d '{"controller": {"controller_id": "controllerone", "ipaddr": "10.0.0.2",
"type": "odc", "version": "1.0", "auditstatus": "enable"}}' http://127.0.0.
1:8083/vtn-webapi/controllers.json
```

创建一个VTN

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"vtn": {"vtn_name": "vtn1", "description": "test VTN" }}' http://127.0.0.
1:8083/vtn-webapi/vtns.json
```

在VTN中创建一个vBridge

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST
-d '{"vbridge": {"vbr_name": "vBridge1", "controller_id": "controllerone",
"domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/
vbridges.json
```

给vBridge创建两个接口

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"interface": {"if_name": "if1", "description": "if_desc1" }}' http://127.0.0.
1:8083/vtn-webapi/vtn1/vbridges/vBridge1/interfaces.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"interface": {"if_name": "if2", "description": "if_desc2" }}' http://127.0.0.
1:8083/vtn-webapi/vtn1/vbridges/vBridge1/interfaces.json
```

获取配置的逻辑端口列表

```
Curl --user admin:adminpass -H 'content-type: application/json' -X GET http://
127.0.0.1:8083/vtn-webapi/controllers/controllerone/domains/\(DEFAULT\)/
logical_ports.json
```

在接口上配置两个映射

```
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d
'{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:03-s3-eth1" }}'
http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if1/
portmap.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d
```

```
'{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:00:02-s2-eth1"}},  
http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if2/  
portmap.json
```

验证

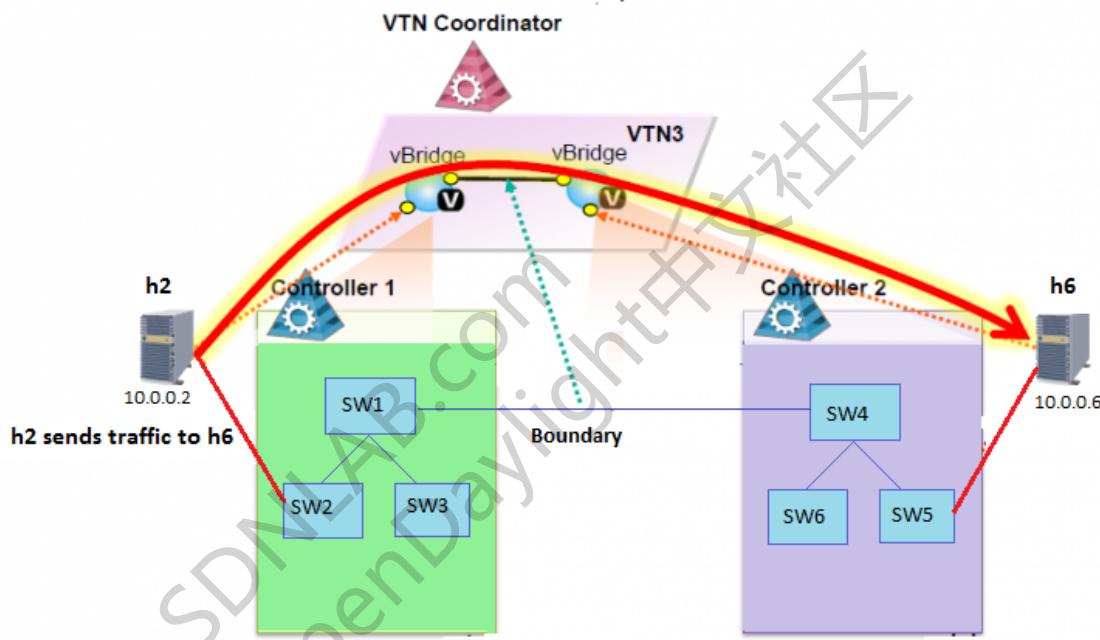
验证Host1和Host3间是否ping通。

```
|mininet> h1 ping h3
```

如何配置多个控制器的2层网络

本例演示使用VTN虚拟化技术（单个控制器）的2层网络中VTN Coordinator的配置。主要示例是使用mininet做多个控制器vBridge接口映射。

图28.20 多个控制器案例展示



需求

使用如下给出的mininet脚本配置多个控制器：https://wiki.opendaylight.org/view/OpenDaylightVirtual_Tenant_Network%28VTN%29:Scripts:Mininet→Network_with_Multiple_Paths_for_delivering_packets

配置

创建一个VTN

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d  
'{"vtn": {"vtn_name": "vtn3"}}' http://127.0.0.1:8083/vtn-webapi/vtns.json
```

创建两个控制器

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d  
'{"controller": {"controller_id": "node1", "ipaddr": "10.100.9.59", "type": "
```

```
"odc", "version": "1.0", "auditstatus":"enable"}' http://127.0.0.1:8083/vtnwebapi/controllers.json

curl --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"controller": {"controller_id": "odc2", "ipaddr": "10.100.9.61", "type":
"odc", "version": "1.0", "auditstatus": "enable"}' http://127.0.0.1:8083/vtnwebapi/controllers.json
```

在Controller1中创建vBridge1，在Controller2中创建vBridge2

```
curl --user admin:adminpass -H 'content-type: application/json' -
X POST -d '{"vbridge": {"vbr_name": "vbr1", "controller_id": "odc1",
"domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/
vbridges.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -
X POST -d '{"vbridge": {"vbr_name": "vbr2", "controller_id": "odc2",
"domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/
vbridges.json
```

创建vBridge接口

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"interface": {"if_name": "if1"}}' http://127.0.0.1:8083/vtn-webapi/vtns/
vtn3/vbridges/vbr1/interfaces.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"interface": {"if_name": "if2"}}' http://127.0.0.1:8083/vtn-webapi/vtns/
vtn3/vbridges/vbr1/interfaces.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"interface": {"if_name": "if1"}}' http://127.0.0.1:8083/vtn-webapi/vtns/
vtn3/vbridges/vbr2/interfaces.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"interface": {"if_name": "if2"}}' http://127.0.0.1:8083/vtn-webapi/vtns/
vtn3/vbridges/vbr2/interfaces.json
```

获取配置的物理端口列表

```
curl --user admin:adminpass -H 'content-type: application/json' -X GET http://
127.0.0.1:8083/vtn-webapi/controllers/odc1/domains/\(DEFAULT\)/logical_ports/
detail.json
```

创建boundary和vLink

```
curl --user admin:adminpass -H 'content-type: application/json' -X
POST -d '{"boundary": {"boundary_id": "b1", "link": {"controller1_id": "odc1",
"domain1_id": "(DEFAULT)", "logical_port1_id": "PPOF:
00:00:00:00:00:01-s1-eth3", "controller2_id": "odc2", "domain2_id": "(DEFAULT)",
"logical_port2_id": "PP-OF:00:00:00:00:00:04-s4-eth3"}}, "boundary_map": {
"boundary_id": "b1", "vlan_id": "50"} }' http://127.0.0.1:8083/vtn-webapi/boundaries.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X
POST -d '{"vlink": {"vlk_name": "vlink1", "vnodel1_name": "vbr1",
"if1_name": "if2", "vnodel2_name": "vbr2", "if2_name": "if2", "boundary_map": {
"boundary_id": "b1", "vlan_id": "50"} } }' http://127.0.0.1:8083/vtn-webapi/
vtns/vtn3/vlinks.json
```

在接口配置端口映射

```
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d
'{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:00:02-s2-eth2"}},'
http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/vbridges/vbr1/interfaces/if1/
portmap.json
```

```
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d
'{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:05-s5-eth2"}},'
http://127.0.0.1:8083/vtn-webapi/vtns/vtn3/vbridges/vbr2/interfaces/if1/
portmap.json
```

验证

验证Host2和Host6之间是否能ping通。

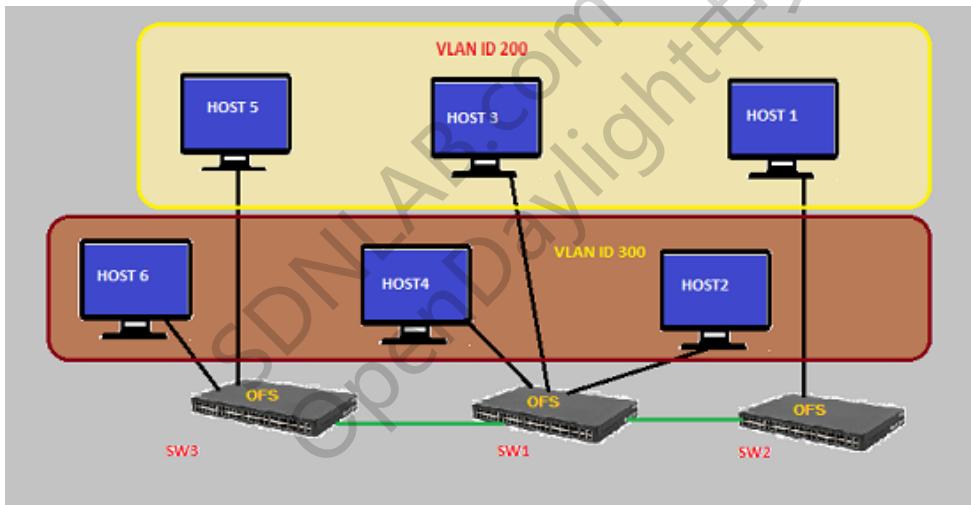
```
mininet> h2 ping h6
```

如何测试mininet环境的Vlan映射

概述

本例介绍了如何在多主机环境测试vlan映射。

图28.21 在mininet环境中演示测试vlan映射



需求

保存mininet脚本vtn_test.py，在mininet环境中运行mininet脚本。

mininet脚本

<https://wiki.opendaylight.org/view/>

OpenDaylightVirtual_Tenant_Network(VTN):Scripts:Mininet→Network_with_hosts_in_different_vlan

运行mininet脚本

```
sudo mn --controller=remote, ip=192.168.64.13 --custom vtn_test.py --topo
mytopo
```

配置

按照以下步骤使用mininet测试vtn映射

```
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"controller": {"controller_id": "controllerone", "ipaddr": "10.0.0.2", "type": "odc", "version": "1.0", "auditstatus": "enable"}}' http://127.0.0.1:8083/vtn-webapi/controllers
```

创建一个VTN

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H 'password: adminpass' -d '{"vtm": {"vtm_name": "vtm1", "description": "test VTN" }}' http://127.0.0.1:8083/vtn-webapi/vtns.json
```

创建一个vBridge (vBridge1)

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H 'password: adminpass' -d '{"vbridge": {"vbr_name": "vBridge1", "controller_id": "controllerone", "domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtm1/vbridges.json
```

为vBridge vBridge1创建vtn映射vlanid 200

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H 'password: adminpass' -d '{"vlanmap": {"vlan_id": 200 }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtm1/vbridges/vBridge1/vlanmaps.json
```

创建一个vBridge (vBridge2)

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H 'password: adminpass' -d '{"vbridge": {"vbr_name": "vBridge2", "controller_id": "controllerone", "domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtm1/vbridges.json
```

为vBridge vBridge2创建vtn映射vlanid 300

```
curl -X POST -H 'content-type: application/json' -H 'username: admin' -H 'password: adminpass' -d '{"vlanmap": {"vlan_id": 300 }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtm1/vbridges/vBridge2/vlanmaps.json
```

验证

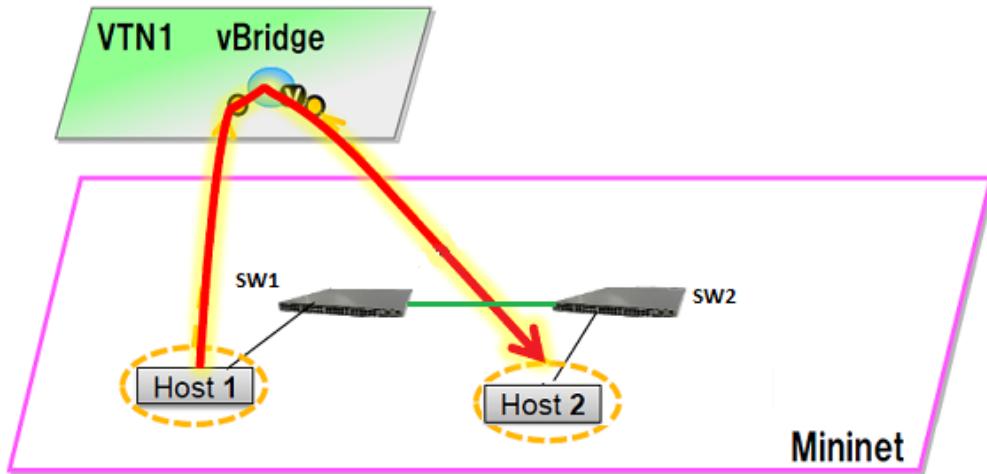
在mininet环境中ping all, 观察主机连接情况

```
mininet> pingall
Ping: testing ping reachability
h1 -> X h3 X h5 X
h2 -> X X h4 X h6
h3 -> h1 X X h5 X
h4 -> X h2 X X h6
h5 -> h1 X h3 X X
h6 -> X h2 X h4 X
```

如何查看特定的VTN配置信息

本例示范了如何查看特定的VTN配置信息。

图28.22 VTN站示例



需求

配置mininet，创建拓扑

```
$ sudo mn --custom /home/mininet/mininet/custom/topo-2sw-2host.py --controller=remote, ip=10.100.9.61 --topo mytopo
mininet> net
s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth1
s2 lo: s2-eth1:s1-eth2 s2-eth2:h2-eth0
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth2
```

分别配置好端口映射以后，ping主机h1和h2

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=16.7 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=13.2 ms
```

配置

创建控制器：

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"controller": {"controller_id": "controllerone", "ipaddr": "10.100.9.61", "type": "odc", "version": "1.0", "auditstatus": "enable"}}' http://127.0.0.1:8083/vtn-webapi/controllers.json
```

创建一个VTN：

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"vtn": {"vtn_name": "vtn1", "description": "test VTN" }}' http://127.0.0.1:8083/vtn-webapi/vtns.json
```

在VTN中创建一个vBridge：

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST
```

```
-d '{"vbridge": {"vbr_name": "vBridge1", "controller_id": "controllerone", "domain_id": "(DEFAULT)"} }' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges.json
```

在vBridge中创建两个接口:

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"interface": {"if_name": "if1", "description": "if_desc1"} }' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d '{"interface": {"if_name": "if2", "description": "if_desc2"} }' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
```

在接口上配置两个映射:

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:01-s1-eth1"} }' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if1/portmap.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X PUT -d '{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:02-s2-eth2"} }' http://17.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if2/portmap.json
```

获取VTN配置信息:

```
curl -v -X GET -H 'content-type: application/json' -H 'username: admin' -H 'password: adminpass' "http://127.0.0.1:8083/vtn-webapi/vtnstations?controller_id=controllerone&vtn_name=vtn1"
```

验证

```
curl -v -X GET -H 'content-type: application/json' -H 'username: admin' -H 'password: adminpass' "http://127.0.0.1:8083/vtn-webapi/vtnstations?controller_id=controllerone&vtn_name=vtn1"
{
  "vtnstations": [
    {
      "domain_id": "(DEFAULT)",
      "interface": {},
      "ipaddrs": [
        "10.0.0.2"
      ],
      "macaddr": "b2c3.06b8.2dac",
      "no_vlan_id": "true",
      "port_name": "s2-eth2",
      "station_id": "178195618445172",
      "switch_id": "00:00:00:00:00:00:02",
      "vnode_name": "vBridge1",
      "vnode_type": "vbridge",
      "vtn_name": "vtn1"
    },
    {
      "domain_id": "(DEFAULT)",
      "interface": {},
      "ipaddrs": [
        "10.0.0.1"
      ],
      "macaddr": "b2c3.06b8.2dac",
      "no_vlan_id": "true",
      "port_name": "s1-eth1",
      "station_id": "178195618445173",
      "switch_id": "00:00:00:00:00:01",
      "vnode_name": "vBridge1",
      "vnode_type": "vbridge",
      "vtn_name": "vtn1"
    }
  ]
}
```

```
"macaddr": "ce82.1b08.90cf",
"no_vlan_id": "true",
"port_name": "sl-eth1",
"station_id": "206130278144207",
"switch_id": "00:00:00:00:00:00:01",
"vnode_name": "vBridge1",
"vnode_type": "vbridge",
"vtm_name": "vtm1"
}
]
}
```

如何在VTN中查看数据流

本例演示如何查看特定的VTN数据流信息。

配置

vlan映射的例子参见<https://wiki.opendaylight.org/view/>

OpenDaylightVirtual_Tenant_Network(VTN):VTNCordinator:RestApi:How_to_test_vlanmap
in_Mininet_environment

验证

获取VTN数据流信息

```
curl -v -X GET -H 'content-type: application/json' --user 'admin:adminpass'  
"http://127.0.0.1:8083/vtn-webapi/dataflows?controller_id=controllerone&  
srcmacaddr=924c.e4a3.a743&vlan_id=300&switch_id=00:00:00:00:00:00:02&  
port_name=s2-eth1"
```

```
{
"dataflows": [
{
"controller_dataflows": [
{
"controller_id": "controllerone",
"controller_type": "odc",
"egress_domain_id": "(DEFAULT)",
"egress_port_name": "s3-eth3",
"egress_station_id": "3",
"egress_switch_id": "00:00:00:00:00:00:03",
"flow_id": "29",
"ingress_domain_id": "(DEFAULT)",
"ingress_port_name": "s2-eth2",
"ingress_station_id": "2",
"ingress_switch_id": "00:00:00:00:00:02",
"match": {
"macdstaddr": [
"4298.0959.0e0b"
],
"macsrcaddr": [
"924c.e4a3.a743"
],
"vlan_id": [
"300"
]
}
}
```

```
"300"
]
},
"pathinfos": [
{
  "in_port_name": "s2-eth2",
  "out_port_name": "s2-eth1",
  "switch_id": "00:00:00:00:00:00:02"
},
{
  "in_port_name": "s1-eth2",
  "out_port_name": "s1-eth3",
  "switch_id": "00:00:00:00:00:01"
},
{
  "in_port_name": "s3-eth1",
  "out_port_name": "s3-eth3",
  "switch_id": "00:00:00:00:00:03"
}
]
}
],
"reason": "success"
}
]
}
```

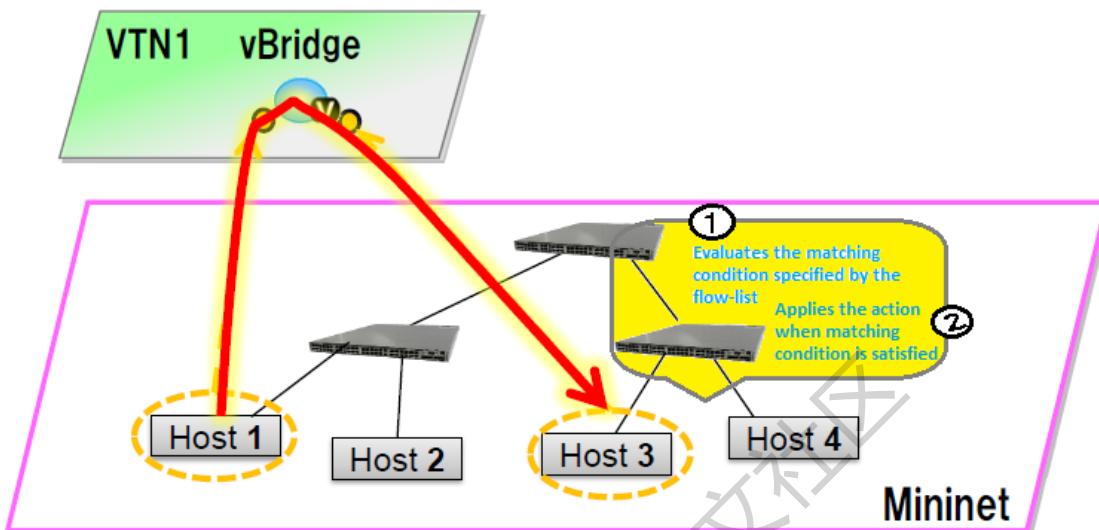
如何使用VTN配置流过滤器

概述

流过滤的匹配规则如下：

Action	功能
Pass	允许包通过。包转发的优先级和DSCP的变化是特定的
Drop	丢弃数据包
Redirect	重定向数据包。当数据包转发的时候可能会改变MAC地址

图28.23 流过滤器



以下步骤分析了流过滤功能：

在虚拟网络中，当一个数据包转发到一个接口，流过滤功能评估转发的数据包是否和流表中特定的条件匹配。

如果数据包匹配相应的条件，流过滤就应用相匹配的action。

需求

要想使用包过滤，配置以下内容：

创建一个flow-list和flow-listentry

指定flow-filter应用的地方，例如VTN、vBridge或者vBridge的接口。

配置mininet，创建拓扑：

```
$ mininet@mininet-vm:~$ sudo mn --controller=remote, ip=<controller-ip> --topo tree
```

生成以下拓扑

```
$ mininet@mininet-vm:~$ sudo mn --controller=remote, ip=<controller-ip> --topo tree,2
mininet> net
c0
s1 lo: s1-eth1:s2-eth3 s1-eth2:s3-eth3
s2 lo: s2-eth1:h1-eth0 s2-eth2:h2-eth0 s2-eth3:s1-eth1
s3 lo: s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s1-eth2
h1 h1-eth0:s2-eth1
```

```
h2 h2-eth0:s2-eth2
h3 h3-eth0:s3-eth1
h4 h4-eth0:s3-eth2
```

配置

创建一个控制器

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST
-d '{"controller": {"controller_id": "controller1", "ipaddr": "10.100.9.61",
"type": "odc", "version": "1.0", "auditstatus": "enable"}}' http://127.0.0.
1:8083/vtn-webapi/controllers
```

创建一个VTN

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"vtm": {"vtm_name": "vtm_one", "description": "test VTN" }}' http://127.0.0.
1:8083/vtn-webapi/vtns.json
```

创建两个vBridge

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST
-d '{"vbridge": {"vbr_name": "vbr_one", "controller_id": "controller1",
"domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtm_one/
vbridges.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"vbridge": {"vbr_name": "vbr_two", "controller_id": "controller1",
"domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtm_one/
vbridges.json
```

创建vBridge接口

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"interface": {"if_name": "if1", "description": "if_desc1" }}' http://127.0.0.
1:8083/vtn-webapi/vtns/vtm_one/vbridges/vbr_two/interfaces.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"interface": {"if_name": "if1", "description": "if_desc1" }}' http://127.0.0.
1:8083/vtn-webapi/vtns/vtm_one/vbridges/vbr_two/interfaces.json
```

在接口上配置两个映射

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X PUT -d
'{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:03-s3-eth1" }}'
http://127.0.0.1:8083/vtn-webapi/vtns/vtm_one/vbridges/vbr_two/interfaces/
if1/portmap.json
curl -v --user admin:adminpass -H 'content-type: application/json' -X PUT -d
'{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:02-s2-eth1" }}'
http://127.0.0.1:8083/vtn-webapi/vtns/vtm_one/vbridges/vbr_two/interfaces/
if2/portmap.json
```

创建Flowlist

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"flowlist": {"fl_name": "flowlist1", "ip_version": "IP" }}' http://127.0.0.
1:8083/vtn-webapi/flowlists.json
```

创建Flowlistentry

```
curl -v --user admin:adminpass -H 'content-type: application/json' -X
POST -d '{"flowlistentry": {"seqnum": "233", "macethertype": "0x8000",
"ipdstaddr": "10.0.0.3", "ipdstaddrprefix": "2", "ipsrcaddr": "10.0.0.2", "ipsrcaddrprefix": "2", "ipproto": "6", "icmpcodenum": "232"} }' http://127.0.0.1:8083/vtn-webapi/flowlists/flowlist1/
flowlistentries.json
```

创建vBridge接口Flowfilter

```
curl -v --user admin:adminpass -X POST -H 'content-type: application/json' -d
'{"flowfilter": {"ff_type": "in"}}' http://127.0.0.1:8083/vtn-webapi/vtns/
vtn_one/vbridges/vbr_two/interfaces/if1/flowfilters.json
```

流过滤演示--DROP

```
curl -v --user admin:adminpass -X POST -H 'content-type: application/
json' -d '{"flowfilterentry": {"seqnum": "233", "f1_name": "flowlist1",
"action_type": "drop", "priority": "3", "dscp": "55"} }' http://127.0.0.1:8083/
vtn-webapi/vtns/vtn_one/vbridges/vbr_two/interfaces/if1/flowfilters/in/
flowfilterentries.json
```

验证

如果我们应用了“drop”，主机间将ping不通。

```
mininet> h1 ping h3
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
```

在控制器中，你可以看到DROP掉的信息，命令： osgi> readflows 0000000000000000

```
[FlowOnNode[flow =Flow[match = Match [fields={DL_VLAN=DL_VLAN(0), IN_PORT=
IN_PORT(OF|1@OF|00:00:00:00:00:03), DL_DST=DL_DST(4e:08:1d:a6:05:08),
DL_SRC=DL_SRC(be:15:00:a4:96:13)}, matches=15], actions = [DROP],
priority = 10, id = 0, idleTimeout = 0, hardTimeout = 300], tableId =
0, sec = 18, nsec = 475000000, pkt = 20, byte = 1232], FlowOnNode[flow
=Flow[match = Match [fields={DL_VLAN=DL_VLAN(0), IN_PORT=IN_PORT(OF|
3@OF|00:00:00:00:00:03), DL_DST=DL_DST(be:15:00:a4:96:13), DL_SRC=
DL_SRC(4e:08:1d:a6:05:08)}, matches=15], actions = [OUTPUT[OF|1@OF|
00:00:00:00:00:03]], priority = 10, id = 0, idleTimeout = 0, hardTimeout
= 0], tableId = 0, sec = 18, nsec = 489000000, pkt = 10, byte = 812]]]
```

流过滤演示--PASS

```
curl -v --user admin:adminpass -X PUT -H 'content-type: application/
json' -d '{"flowfilterentry": {"seqnum": "233", "f1_name": "flowlist1",
"action_type": "pass", "priority": "3", "dscp": "55"} }' http://127.0.0.1:8083/
vtn-webapi/vtns/vtn_one/vbridges/vbr_two/interfaces/if1/flowfilters/in/
flowfilterentries/233.json
```

验证

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=0.984 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.110 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=0.098 ms
```

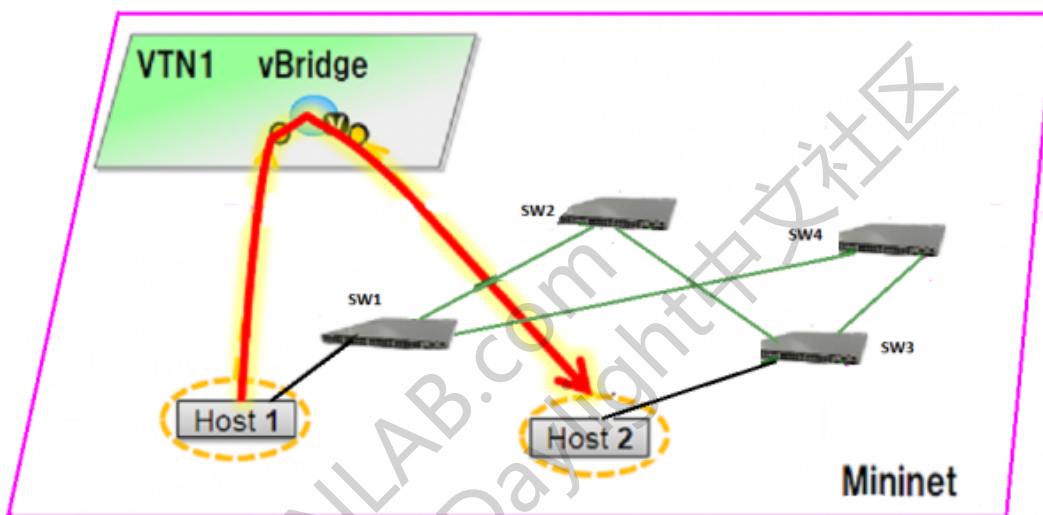
在控制器中，通过以下命令可以看到PASS信息：

```
osgi> readflows 0000000000000003
```

如何使用VTN采用不同的路径转发数据包

本例演示如何创建特定的VTN路径映射信息。

图28.24 PathMap



需求

保存mininet脚本pathmap_test.py，在mininet环境中运行该脚本。

使用以下脚本创建拓扑：

```
from mininet.topo import Topo
class MyTopo( Topo ):
    "Simple topology example."
    def __init__( self ):
        "Create custom topo."
        # Initialize topology
        Topo.__init__( self )
        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's1' )
        middleSwitch = self.addSwitch( 's2' )
        middleSwitch2 = self.addSwitch( 's4' )
        rightSwitch = self.addSwitch( 's3' )
        # Add links
        self.addLink( leftHost, leftSwitch )
```

```

self.addLink( leftSwitch, middleSwitch )
self.addLink( leftSwitch, middleSwitch2 )
self.addLink( middleSwitch, rightSwitch )
self.addLink( middleSwitch2, rightSwitch )
self.addLink( rightSwitch, rightHost )
topos = { 'mytopo': ( lambda: MyTopo() ) }

```

```

mininet> net
c0
s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth1 s1-eth3:s4-eth1
s2 lo: s2-eth1:s1-eth2 s2-eth2:s3-eth1
s3 lo: s3-eth1:s2-eth2 s3-eth2:s4-eth2 s3-eth3:h2-eth0
s4 lo: s4-eth1:s1-eth3 s4-eth2:s3-eth2
h1 h1-eth0:s1-eth1
h2 h2-eth0:s3-eth3

```

在创建端口映射前ping主机h1和h2

```

mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable

```

配置

创建控制器

```

curl --user admin:adminpass -H 'content-type: application/json' -X POST -
d '{"controller": {"controller_id": "odc", "ipaddr": "10.100.9.42", "type": "odc", "version": "1.0", "auditstatus": "enable"}}' http://127.0.0.1:8083/vtnwebapi/controllers.json

```

创建一个VTN

```

curl --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"vtn": {"vtn_name": "vtn1", "description": "test VTN" }}' http://127.0.0.1:8083/vtn-webapi/vtns.json

```

在VTN中创建一个vBridge

```

curl --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"vbridge": {"vbr_name": "vBridge1", "controller_id": "odc", "domain_id": "(DEFAULT)" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges.json

```

给vBridge创建两个接口

```

curl --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"interface": {"if_name": "if1", "description": "if_desc1" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json
curl --user admin:adminpass -H 'content-type: application/json' -X POST -d
'{"interface": {"if_name": "if2", "description": "if_desc2" }}' http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces.json

```

在接口配置两个映射

```
curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d
'{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:00:01-s1-eth1"}}'
http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if1/
portmap.json

curl --user admin:adminpass -H 'content-type: application/json' -X PUT -d
'{"portmap": {"logical_port_id": "PP-OF:00:00:00:00:00:00:03-s3-eth3"}}'
http://127.0.0.1:8083/vtn-webapi/vtns/vtn1/vbridges/vBridge1/interfaces/if2/
portmap.json
```

在创建好端口映射以后，ping主机h1和h2

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=36.4 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.880 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=0.081 ms
```

获取VTN数据流信息

```
curl -X GET -H 'content-type: application/json' --user 'admin:adminpass'
"http://127.0.0.1:8083/vtn-webapi/dataflows?&switch_id=
00:00:00:00:00:01&port_name=s1-eth1&controller_id=odc&srcmacaddr=de3d.
7dec.e4d2&no_vlan_id=true"
```

在VTN中创建Flowcondition

```
curl --user admin:admin -H 'content-type: application/json' -X PUT
-d '{"name": "flowcond_1", "match": [{"index": 1, "ethernet": {"src": "ca:9e:58:0c:1e:f0", "dst": "ba:bd:0f:e8:a8:c8", "type": 2048}, "inetMatch": {"inet4": {"src": "10.0.0.1", "dst": "10.0.0.2", "protocol": 1}}}], "controller_id": "odc", "srcmacaddr": "de3d.7dec.e4d2", "no_vlan_id": true}' http://10.100.9.42:8282/controller/nb/v2/vtn/default/flowconditions/flowcond_1
```

在VTN中创建Pathmap

```
curl --user admin:admin -H 'content-type: application/json' -X PUT -d
'{"index": 10, "condition": "flowcond_1", "policy": 1, "idleTimeout": 300,
"hardTimeout": 0}' http://10.100.9.42:8282/controller/nb/v2/vtn/default/
pathmaps/1
```

获取路径策略信息

```
curl --user admin:admin -H 'content-type: application/json' -X GET -d
'{"id": 1, "default": 100000, "cost": [{"location": {"node": {"type": "OF", "id": "00:00:00:00:00:00:01"}, "port": {"type": "OF", "id": "3", "name": "s1-eth3"}}, {"cost": 1000}, {"location": {"node": {"type": "OF", "id": "00:00:00:00:00:04"}, "port": {"type": "OF", "id": "2", "name": "s4-eth2"}}, {"cost": 1000}, {"location": {"node": {"type": "OF", "id": "00:00:00:00:00:03"}, "port": {"type": "OF", "id": "3", "name": "s3-eth3"}}, {"cost": 100000}]]}' http://10.100.9.42:8282/controller/nb/v2/vtn/
default/pathpolicies/1
```

验证

在部署路径策略信息之前

```
{  
  "pathinfos": [  
    {  
      "in_port_name": "s1-eth1",  
      "out_port_name": "s1-eth2",  
      "switch_id": "00:00:00:00:00:00:00:01"  
    },  
    {  
      "in_port_name": "s2-eth1",  
      "out_port_name": "s2-eth2",  
      "switch_id": "00:00:00:00:00:00:00:02"  
    },  
    {  
      "in_port_name": "s3-eth1",  
      "out_port_name": "s3-eth3",  
      "switch_id": "00:00:00:00:00:00:00:03"  
    }  
  ]  
}
```

在部署路径策略之后

```
{  
  "pathinfos": [  
    {  
      "in_port_name": "s1-eth1",  
      "out_port_name": "s1-eth3",  
      "switch_id": "00:00:00:00:00:00:00:01"  
    },  
    {  
      "in_port_name": "s4-eth1",  
      "out_port_name": "s4-eth2",  
      "switch_id": "00:00:00:00:00:00:00:04"  
    },  
    {  
      "in_port_name": "s3-eth2",  
      "out_port_name": "s3-eth3",  
      "switch_id": "00:00:00:00:00:00:00:03"  
    }  
  ]  
}
```

VTN Coordinator（故障排除指南）

概述

本节演示安装VTN Coordinator故障排除的步骤。

OpenDaylight VTN提供多租户虚拟网络功能。OpenDaylight VTN包括两个部分：

VTN Coordinator

VTN Manager

VTN Coordinator编排多个运行在OpenDaylight控制器上的VTN Managers，提供了VTN API。VTN Manager是运行在OpenDaylight控制器上的OSGI bundles。目前的VTN Manager只支持OpenFlow交换机。它处理PACKET_IN消息，发送PACKET_OUT消息，管理主机信息，给OpenFlow交换机安装流表项，提供给VTN Coordinator虚拟网络功能。安装这两个部分的需求是不一样的。所以，建议在不同的机器上分别安装VTN Manager 和VTN

Coordinator。

安装故障排除的列表

如何安装VTN Coordinator

<https://wiki.opendaylight.org/view/>

OpenDaylightVirtual_Tenant_Network(VTN):Installation:VTN_Coordinator

执行完db_setup后，你会遇到错误"Failed to setup database"吗？

出现这个错误的原因如下：

拥有/usr/local/vtn/目录并且安装VTN Coordinator的用户只能启动db_setup。例：应该出现以下目录（假设用户名是vtm）：

```
# ls -l /usr/local/  
drwxr-xr-x. 12 vtm vtm 4096 Mar 14 21:53 vtn
```

如果用户没有 /usr/local/vtn/目录，运行以下命令（假设用户名是vtm）：

```
chown -R vtm:vtm /usr/local/vtn
```

Postgres不存在

1 如果是Fedora/CentOS/RHEL系统，请检查是否存在/usr/pgsql/目录，确保命令initdb、createdb、pg_ctl和psql可用，如果不能使用的话，重新安装postgres包。

2 如果是Ubuntu系统，检查是否存在/usr/lib/postgres/目录，并且按照1中的步骤进行检查。

没有足够的空间创建表

请检查磁盘df -k，确保有足够的空间。

如果以上步骤没有解决问题，查询log文件定位问题

```
/usr/local/vtn/var/dbm/unc_setup_db.log
```

vtm启用后，需要检查哪些东西？

VTN Coordinator进程列表

运行如下命令，确保Coordinator进程启动

```
/usr/local/vtn/bin/unc_dmctl status  
Name Type IPC Channel PID  
-----  
drvodcd DRIVER drvodcd 15972  
lgcnwd LOGICAL lgcnwd 16010  
phynwd PHYSICAL phynwd 15996
```

发布curl命令，获取版本号，确保进程可以响应。

如何debug一个启动错误？下面的内容发生在启动过程中

数据库服务器在设置好虚拟内存以后启动，任何数据库的启动错误都会在下面的log中呈现。

```
/usr/local/vtn/var/dbm/unc_db_script.log.  
/usr/local/vtn/var/db/pg_log/postgresql-*.log (the pattern will have  
the date)
```

uncd进程开启，该进程依次启动其他的进程。

任何的uncd启动错误都会在/usr/local/vtn/var/uncd/uncd_start.err中显示。

安装好apache tomcat服务器以后，还需要检查的内容

检查catalina是否启用

命令行ps -ef | grep catalina | grep -v grep会显示catalina进程

如果遇到：REST API总是报错

请确保防火墙设置的端口号是:8282(Lithium)或者端口：8083（Lithium）

如何调试一个REST API返回一个错误消息？请检查/usr/share/
java/apache-tomcat-7.0.39/logs/core/core.log查看详细错误信息。

VTN配置中REST API错误，如何debug？所有进程默认的log等级是“INFO”，可能需要debug TRACE或者DEBUG
logs。想要提高log等级，使用如下命令

```
/usr/local/vtn/bin/lgcnw_control loglevel trace -- up11 daemon log  
/usr/local/vtn/bin/phynw_control loglevel trace -- uppl daemon log  
/usr/local/vtn/bin/unc_control loglevel trace -- unc daemon log  
/usr/local/vtn/bin/drvodc_control loglevel trace -- Driver daemon log
```

log等级设置好以后，可以重复操作，log文件可以作为debugging的参考文件。

安装PostgreSQL出现的问题。在试图安装PostgreSQL rpms的时候，可能会出现一些问题。最近PostgreSQL使
用最新的openssl版本升级了所有的二进制文件，参见<http://en.wikipedia.org/wiki/Heartbleed>。请将openssl包更
新到最新版本并且重新安装。[RHEL 6.1/6.4：更新rpms](#)。详情链

接：<https://access.redhat.com/site/solutions/781793>

```
rpm -Uvh http://mirrors.kernel.org/centos/6/os/x86_64/Packages/openssl-1.0.  
1e-15.e16.x86_64.rpm  
rpm -ivh http://mirrors.kernel.org/centos/6/os/x86_64/Packages/openssl-devel-  
1.0.1e-15.e16.x86_64.rpm
```

对于其他的linux平台，请更新yum，并且安装最新的版本。