Floodlight 指南

最新版: yeasy@github

更新历史:

V0.3: 2013-05-07

根据最新官方文档，进行大量更新。包括更新 REST API、添加新的模块、对 OpenStack 的支持等。

V0.2: 2012-11-01

完成全部章节。

第1章 安装运行

官方网站为 http://www.openflowhub.org。

1.1 安装

1.1.1 获取源码和编译

执行代码为

```
sudo apt-get install build-essential default-jdk ant python-dev eclipse



git clone git://github.com/floodlight/floodlight.git

cd floodlight

git checkout stable

ant;
```

1.1.2 配置 eclipse

打开 eclipse，创建新的 workspace。

从菜单选择 File -> Import -> General -> Existing Projects into Workspac。点击下一步。

从 Select root directory 里面，找到 floodlight 所在目录。导入后勾上 Floodlight，点击 Finish。

下面配置运行时环境：

点击 Run->Run Configurations。

右键点击 Java Application，选择 New。

名字选择 FloodlightLaunch，Project 选择 Floodlight，Main 选择 net.floodlightcontroller.core.Main，最后点击 Apply。

1.1.3 运行

```
java -jar target/floodlight.jar
```

1.2 自定义配置

配置需要编译的模块，修改

```
src/main/resources/META-INF/services/net.floodlightcontroller.core.module
```

配置需要加载或运行的模块，配置启动参数等，修改

```
src/main/resources/floodlightdefault.properties
```

floodlight 作为服务启动时，配置文件在

```
/opt/floodlight/floodlight/configuration/floodlight.properties
```

第2章REST API

尚未稳定。重新设计过的 API 功能更为完善。

REST API 是 Floodlight 与应用程序交互的推荐核心渠道之一，目前，大部分应用都支持通过 API 与 Floodlight 进行交互。

一个 REST 调用的例子为：

```
curl http://192.168.110.2:8080/wm/core/controller/switches/json
```

其他的 通过 REST 的函数包括：

URI	Method	Description	Arguments
/wm/core/switch/all/<statType>/json	GET	Retrieve aggregate **stats ac**ross all switches	statType: port, queue, flow, aggregate, desc, table, features
/wm/core/switch/<switchId>/<statType>/json	GET	**Retrieve** per switch stats	switchId: Valid Switch DPID (X**X:XX:XX:**XX:XX:XX:XX:XX) statType: port, queue, flow, aggregate, desc, table, features
/wm/core/controller/switches/json	GET	List of all switch DPIDs connected to the controller	none
/wm/core/counter/<counterTitle>/json	GET	List of global traffic counters in the control**ler (across** all switches)	counterTitle: "all" or s*omething of the* form DPID_Port#OFEvent L3/4_Type. See CounterStore.java for details.
/wm/core/counter/<switchId>/<counterName>/json	GET	List of tr**affic co**unters per switch	switchId: Valid Switch DPID CounterTitle: see above

/wm/core/memory/json	GET	Current controller memory usage	none
/wm/topology/links/json	GET	List all the inter-switch links. Note that these are only for switches connected to the same controller. This is not available in the 0.8 release.	none
/wm/topology/switchclusters/json	GET	List of all switch clusters connected to the controller. This is not available in the 0.8 release.	none
/wm/topology/external-links/json	GET	Show "external" links, i.e., multi-hop links discovered by BDDP instead of LLDP packets	none
/wm/topology/links/json	GET	Show DIRECT and TUNNEL links discovered based on LLDP	none

		packets	
/wm/device/	GET	List of all devices tracked by the controller. This includes MACs, IPs, and attachment points.	**Pas**sed as GET parameters: mac (co lo**n-se**parated hex-encoded**), i**pv**4 (d**ott ed decimal), vlan, dpid attachment point DPID (colon-s**epar**ated hex-encoded) and port the attachment point port.
/wm/staticflowentrypusher/json	POST/DELETE	Add/Delete static flow	HTTP POST data (add flow), HTTP DELETE (for deletion)
/wm/staticflowentrypusher/list/<switch>/json	GET	List static flows for **a swit**ch or all switches	switch: Valid Switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) or "all"
/wm/staticflowentrypusher/clear/<switch>/json	GET	Clear static flows for **a swit**ch or all switches	switch: Valid Switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) or "all"
More information available on How to Use Static Flow Pusher API			
/networkService/v1.1/tenants/<tenant>/networks/<network>	PUT/POST/DELETE	Creates a new virtual network. Name and ID are required, gateway **is opti**onal.	URI argument: t**enant:** Currently ignored. network: ID (not name) of the network HTTP data: {"network": { "gateway": "<I**P>**", "name": "<Name>" }} IP: Gateway IP in

			"1.1.1.**1" f**ormat, can be null Name: Network name as string
/networkService/v1.1/tenants/<tenant>/networks/<network>/ports/<port>/attachment	PUT/DELETE	Attaches a host to a virt**ual net**work.	URI argument: t**enant:** Currently ignored. network: ID (**not** name) of the network. port: Logical port name HTTP data: {"attachment": {"id": "<Networ**k ID>", "m**ac": "<MAC>"}} Network ID: Network ID as string, th**e o**ne assigned at create MAC: MAC address in "XX:XX:XX:XX:XX:XX" format
/networkService/v1.1/tenants/<tenant>/networks	GET	Shows all networks and their gateway, ID, and hosts mac in j**son for**mat.	URI argument: tenant: Currently ignored.
More information available on Virtual Network Filter REST API			
/wm/firewall/module/<op>/json	GET		query the status of, enable, and disable the firewall
/wm/firewall/rules/json	GET/POST/DELETE	GET: None POST: {"<field 1>":"<value 1>", "<field 2>":"<value 2>", ...} DELETE: {"<	List all existing rules in json format Create new firewall rule **Delet**e a **rule** by ruleid "field":"value" pairs

		ruleid>":"<int>"}	below in any **order and** combination: "switchid":"<xx**:xx:xx:xx:**xx:xx:xx:xx>", "s**rc-inpo**rt":"<short>", "src-mac"**:** **"<xx:**xx:xx:xx:xx:xx>", "dst-mac": "**<xx:xx:xx:**xx:xx:xx:*xx*>", "dl-**type"**: "<ARP or IPv4>", **"src-i**p": "<A.B.C.D/M>", "ds**t-ip":** **"**<A.B.C.D/*M*>", ″nw-proto": **"<TCP** or UDP or ICMP**>", "t**p-src": "<short>", **"tp-dst**": "<short>", **"p**riority": "<*i*nt>", "acti**on":** **"**<ALLOW or DENY>" "ruleid": "<int>"

More information available on Firewall REST API

2.1 Virtual Network Filter

URI	Method	URI Arguments	Data	Data Fields	Description
/networkService/v1.1/tenants/{tenant}/networks/{network}	PUT/POST/DELETE	Tenant: Currently ignored Network: The ID (not	{"network": { "gateway": "<IP>", "name": "<Name>" }}\	IP: Gateway IP in "1.1.1.1" format, can be null Name: Network name a string	Creates a new virtual network. Name and ID are required,

		name) of the network			gateway is optional.
/networkService/v1.1/tenants/{tenant}/networks/{network}/ports/{port}/attachment	PUT/DELETE	Tenant: Currently ignored Network: The ID (not name) of the network Port: Logical port name	{"attachment": {"id": "<Network ID>", "mac": "<MAC>"}}	Network ID: Network ID as a string, the one you just created MAC: MAC address in "00:00:00:00:00:09" format	Attaches a host to a virtual network.
/networkService/v1.1/tenants/{tenant}/networks	GET	Tenant: Currently ignored	None	None	Shows all networks and their gateway, ID, and hosts mac in json format

例子：

创建一个虚拟网络，名称为“VirtualNetwork1”，ID 为“NetworkId1”，网关是“10.0.0.7”，租户是“default”。

```
curl -X PUT -d '{ "network": { "gateway": "10.0.0.7", "name": "virtualNetwork1" } }' http://localhost:8080/networkService/v1.1/tenants/default/networks/NetworkId1
```

添加一个主机到 VirtualNetwork1，MAC 地址为“00:00:00:00:00:08”，端口为”port1”。

```
curl -X PUT -d '{"attachment": {"id": "NetworkId1", "mac": "00:00:00:00:00:08"}}'
http://localhost:8080/networkService/v1.1/tenants/default/networks/NetworkId
1/ports/port1/attachment
```

2.2 Static Flow Pusher

通过该模块，能让用户手动修改每个交换机上的流表项，从而创建任意转发路径。

2.2.1 主动和被动添加流

OpenFlow 支持两种方式的添加流：被动和主动。被动方式情况下，网包到达交换机，发现不存在匹配的流，则转发给控制器，控制器进行计算并添加正确的流到交换机，让交换机继续进行转发。另外一种主动方式，则是在网包到达前，控制器主动让交换机添加流。

Floodlight 支持两种方式。Static Flow Pusher 模块一般用于主动添加流。

注意在默认情况下，Forwarding 模块被加载，控制器会进行被动（reactive）的流加载，如果不想进行被动加载，则需要在 floodlight.properties 文件中禁用 forwarding 模块。。

2.2.2 API

支持的 api 有

URI	Description	Arguments
/wm/staticflowentrypusher/json	Add/Delete static flow	HTTP POST data (add flow), HTTP DELETE (for deletion)
/wm/staticflowentrypusher/list/<switch>/json	List static flows for a switch or all switches	**switch**: Valid Switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) or "all"
/wm/staticflowentrypusher/clear/<switch>/json	Clear static flows for a switch or all switches	**switch**: Valid Switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) or "all"

2.2.2.1 添加流

向 switch 1 添加一条静态流表项（从 port 1 接收包，从 port 2 发出包）的例子如下，其中，第二条指令 dump 出所有的流来。

```
curl -d '{"switch": "00:00:00:00:00:00:00:01", "name":"flow-mod-1", "priority":"32
768", "ingress-port":"1","active":"true", "actions":"output=2"}' http://<controller
_ip>:8080/wm/staticflowentrypusher/json
```

```
curl http://<controller_ip>:8080/wm/core/switch/1/flow/json;
```

2.2.2.2 删除流

```
curl -X DELETE -d '{"name":"flow-mod-1"}' http://<controller_ip>:8080/wm/staticflowentrypusher/json
```

2.2.2.3 流属性列表

Key	Value	Notes
switch	<switch ID>	ID of the switch (data path) that this rule should be added to xx:xx:xx:xx:xx:xx:xx:xx
name	<string>	Name of the flow entry, this is the primary key, it MUST be unique
actions	<key>=<value>	See table of actions below Specify multiple actions using a comma-separated list Specifying no actions will cause the packets to be dropped
priority	<number>	default is 32767 maximum value is 32767
active	<boolean>	
wildcards		
ingress-port	<number>	switch port on which the packet is received Can be hexadecimal (with leading 0x) or decimal
src-mac	<mac address>	xx:xx:xx:xx:xx:xx
dst-mac	<mac address>	xx:xx:xx:xx:xx:xx
vlan-id	<number>	Can be hexadecimal (with leading 0x) or decimal
vlan-priority	<number>	Can be hexadecimal (with leading 0x) or decimal
ether-type	<number>	Can be hexadecimal (with leading 0x) or decimal
tos-bits	<number>	Can be hexadecimal (with leading 0x) or decimal
protocol	<number>	Can be hexadecimal (with leading 0x) or decimal

src-ip	<ip address>	xx.xx.xx.xx
dst-ip	<ip address>	xx.xx.xx.xx
src-port	<number>	Can be hexadecimal (with leading 0x) or decimal
dst-port	<number>	Can be hexadecimal (with leading 0x) or decimal

2.2.2.4 流行动列表

Key	Value	Notes
output	<number> all controller local ingress-port normal flood	no "drop" option (instead, specify no action to drop packets)
enqueue	<number>:<number>	First number is port number, second is queue ID Can be hexadecimal (with leading 0x) or decimal
strip-vlan		
set-vlan-id	<number>	Can be hexadecimal (with leading 0x) or decimal
set-vlan-priority	<number>	Can be hexadecimal (with leading 0x) or decimal
set-src-mac	<mac address>	xx:xx:xx:xx:xx:xx
set-dst-mac	<mac address>	xx:xx:xx:xx:xx:xx
set-tos-bits	<number>	
set-src-ip	<ip address>	xx.xx.xx.xx
set-dst-ip	<ip address>	xx.xx.xx.xx
set-src-port	<number>	Can be hexadecimal (with leading 0x) or decimal
set-dst-port	<number>	Can be hexadecimal (with leading 0x) or decimal

2.2.3 使用实践

利用 python 来调用 Static Flow Pusher，与 mininet 交互进行操作。
首先，启动 mininet 生成一张简单网络。

```
sudo mn --controller=remote --ip=<controller ip> --port=6633
```

创建 python 脚本，将 h2 和 h3 之间添加流表项，让两者互通。

```
import httplib

import json



class StaticFlowPusher(object):



    def __init__(self, server):

        self.server = server



    def get(self, data):

        ret = self.rest_call({}, 'GET')

        return json.loads(ret[2])



    def set(self, data):

        ret = self.rest_call(data, 'POST')

        return ret[0] == 200
```

```
    def remove(self, objtype, data):
        ret = self.rest_call(data, 'DELETE')
        return ret[0] == 200

    def rest_call(self, data, action):
        path = '/wm/staticflowentrypusher/json'
        headers = {
            'Content-type': 'application/json',
            'Accept': 'application/json',
            }
        body = json.dumps(data)
        conn = httplib.HTTPConnection(self.server, 8080)
        conn.request(action, path, body, headers)
        response = conn.getresponse()
        ret = (response.status, response.reason, response.read())
        print ret
        conn.close()
        return ret

pusher = StaticFlowPusher('<insert_controller_ip')
```

```
flow1 = {
    'switch':"00:00:00:00:00:00:00:01",
    "name":"flow-mod-1",
    "cookie":"0",
    "priority":"32768",
    "ingress-port":"1",
    "active":"true",
    "actions":"output=flood"
    }

flow2 =      {
    'switch':"00:00:00:00:00:00:00:01",
    "name":"flow-mod-2",
    "cookie":"0",
    "priority":"32768",
    "ingress-port":"2",
    "active":"true",
    "actions":"output=flood"
    }
```

```
pusher.set(flow1)

pusher.set(flow2)
```

2.3 Firewall

支持的 REST 方法如下：

URI	**Method**	**URI Arguments**	**Data**	Data Fields	Description
/wm/fire**wa**ll/module/<op>/json	GET	op: status, enable, disable, storageRules, subnet-mask	None	None	query the status of, enable, and disable the firewall
/wm/firewall/rules/json	GET	None	None	None	List all existing rules in json format
	POST	None	{"<field 1>":"<value 1>", "<fie**ld 2>**":"**<valu**e 2>", ...}	"field":"value" pairs below in any **order a**nd combination: "switchid":"<xx**:xx:xx:xx:**xx:xx:xx:xx>", "**src-inp**ort":"<short>", "src-mac"**:** **"<xx:**xx:xx:xx:xx:xx>", "dst-mac": **"<xx:xx**:xx:xx:xx:xx>", "dl-**type"**: "<ARP or IPv4>", **"src-i**p": "<A.B.C.D/M>", "d**st-ip":** "<A.B.C.D/*M*>", *"*nw-proto": **"<TCP** or UDP or ICMP**>", "t**p-src": "<short>**", "tp-ds**t": "<short>**",** **"p**riority": "<*i*nt>", "action":	Create new firewall rule

				"<ALLOW or DENY>" Note: specifying src-ip/dst-ip without specifying dl-type as ARP, or specifying any IP-based nw-proto will automatically set dl-type to match IPv4.	
	DELETE	None	{"<ruleid>":"<int>"}	"ruleid": "<int>" Note: ruleid is a random number generated and returned in the json response upon successful creation	Delete a rule by ruleid

例子：
假设控制器在本地运行，查看防火墙是否开启。

```
curl http://localhost:8080/wm/firewall/module/status/json
```

开启防火墙，默认情况下，防火墙禁止任何流量通过。

```
curl http://localhost:8080/wm/firewall/module/enable/json
```

添加一条 ALLOW 规则到交换机 00:00:00:00:00:00:00:01，允许所有流通过。

```
curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:01"}' http://localhost:8080/wm/firewall/rules/json
```

添加一条 ALLOW 规则，到主机 10.0.0.3 和主机 10.0.1.5 之间的所有流。

```
curl -X POST -d '{"src-ip": "10.0.0.3/32"}' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"dst-ip": "10.0.0.3/32"}' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"src-ip": "10.0.0.7/32"}' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"dst-ip": "10.0.0.7/32"}' http://localhost:8080/wm/firewall/rules/json
```

添加一条 ALLOW 规则，到主机 00:00:00:00:00:0a 和主机 00:00:00:00:00:0b 之间的所有流。

```
curl -X POST -d '{"src-mac": "00:00:00:00:00:0a"}' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"dst-mac": "00:00:00:00:00:0a"}' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"src-mac": "00:00:00:00:00:0b"}' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"dst-mac": "00:00:00:00:00:0b"}' http://localhost:8080/wm/firewall/rules/json
```

添加一条 ALLOW 规则到主机 10.0.0.3 和 10.0.0.7 之间，允许 ping 通过。

```
curl -X POST -d '{"src-ip": "10.0.0.3/32", "dl-type":"ARP" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"dst-ip": "10.0.0.3/32", "dl-type":"ARP" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"src-ip": "10.0.0.7/32", "dl-type":"ARP" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"dst-ip": "10.0.0.7/32", "dl-type":"ARP" }' http://localhost:8080/wm/firewall/rules/json



curl -X POST -d '{"src-ip": "10.0.0.3/32", "nw-proto":"ICMP" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"dst-ip": "10.0.0.3/32", "nw-proto":"ICMP" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"src-ip": "10.0.0.7/32", "nw-proto":"ICMP" }' http://localhost:8080/wm/firewall/rules/json
```

```
curl -X POST -d '{"dst-ip": "10.0.0.7/32", "nw-proto":"ICMP" }' http://localhost:8080/wm/firewall/rules/json
```

添加一条 ALLOW 规则到主机 10.0.0.4 和 10.0.0.10 之间，允许 UDP 通过，但禁用端口 5010。

```
curl -X POST -d '{"src-ip": "10.0.0.4/32", "dl-type":"ARP" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"dst-ip": "10.0.0.4/32", "dl-type":"ARP" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"src-ip": "10.0.0.10/32", "dl-type":"ARP" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"dst-ip": "10.0.0.10/32", "dl-type":"ARP" }' http://localhost:8080/wm/firewall/rules/json



curl -X POST -d '{"src-ip": "10.0.0.4/32", "nw-proto":"UDP" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"dst-ip": "10.0.0.4/32", "nw-proto":"UDP" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"src-ip": "10.0.0.10/32", "nw-proto":"UDP" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"dst-ip": "10.0.0.10/32", "nw-proto":"UDP" }' http://localhost:8080/wm/firewall/rules/json



curl -X POST -d '{"src-ip": "10.0.0.4/32", "nw-proto":"UDP", "tp-src":"5010", "action":"DENY" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"dst-ip": "10.0.0.4/32", "nw-proto":"UDP", "tp-dst":"5010", "action":"DENY" }' http://localhost:8080/wm/firewall/rules/json
```

```
curl -X POST -d '{"src-ip": "10.0.0.10/32", "nw-proto":"UDP", "tp-src":"5010", "ac
tion":"DENY" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"dst-ip": "10.0.0.10/32", "nw-proto":"UDP", "tp-dst":"5010", "ac
tion":"DENY" }' http://localhost:8080/wm/firewall/rules/json
```

第3章Java 模块加载系统

floodlight 使用一个模块系统来决定运行哪些模块，包括通过配置文件来决定加载的模块、指定模块的实现、方便扩展 floodlight 等。

模块系统包括 loader、modules、services、配置文件、在一个 jar 中可用 modules 的列表文件。

3.1 Modules

模块是实现了 IfloodlightModule 接口的一个类。

```
/**
* Defines an interface for loadable Floodlight modules.
*
* At a high level, these functions are called in the following order:
* <ol>
* <li> getServices() : what services does this module provide
* <li> getDependencies() : list the dependencies
* <li> init() : internal initializations (don't touch other modules)
* <li> startUp() : external initializations (<em>do</em> touch other modules)
* </ol>
*
* @author alexreimers
*/
public interface IFloodlightModule {
```

```
    /**
     * Return the list of interfaces that this module implements.
     * All interfaces must inherit IFloodlightService
     * @return
     */

    public Collection<Class<? extends IFloodlightService>> getModuleService
s();


    /**
     * Instantiate (as needed) and return objects that implement each
     * of the services exported by this module.  The map returned maps
     * the implemented service to the object.  The object could be the
     * same object or different objects for different exported services.
     * @return The map from service interface class to service implementation
     */
    public Map<Class<? extends IFloodlightService>,
            IFloodlightService> getServiceImpls();


    /**
     * Get a list of Modules that this module depends on.  The module system
```

```
     * will ensure that each these dependencies is resolved before the
     * subsequent calls to init().
     * @return The Collection of IFloodlightServices that this module depnds
     *         on.
     */

    public Collection<Class<? extends IFloodlightService>> getModuleDependencies();

    /**
     * This is a hook for each module to do its <em>internal</em> initialization,
     * e.g., call setService(context.getService("Service"))
     *
     * All module dependencies are resolved when this is called, but not every module
     * is initialized.
     *
     * @param context
     * @throws FloodlightModuleException
     */
```

```
    void init(FloodlightModuleContext context) throws FloodlightModuleExcep
tion;

    /**
     * This is a hook for each module to do its <em>external</em> initializati
ons,
     * e.g., register for callbacks or query for state in other modules
     *
     * It is expected that this function will not block and that modules that wan
t
     * non-event driven CPU will spawn their own threads.
     *
     * @param context
     */

    void startUp(FloodlightModuleContext context);
}
```

3.2 Services

一个 module 可以提供若干 service，service 是扩展了 IfloodlightService 接口的一个接口。

```
/**
 * This is the base interface for any IFloodlightModule package that provides
```

```
* a service.

* @author alexreimers

*

*/

public abstract interface IFloodlightService {

   // This space is intentionally left blank....don't touch it

}
```

3.3 配置文件

配置文件定义加载哪些 modules，格式是标准的 java 属性格式，定义为 floodlight.modules =xx，xx。默认位于 src/main/resources/floodlightdefault.properties

```
floodlight.modules = net.floodlightcontroller.staticflowentry.StaticFlowEntryPus
her,\

net.floodlightcontroller.forwarding.Forwarding,\

net.floodlightcontroller.jython.JythonDebugInterface
```

3.4 模块文件

该文件列出可用的模块。位于 src/main/resources/META-INFO/services/net.floodlightcontroll er.module.IfloodlightModule。如

```
net.floodlightcontroller.core.CoreModule

net.floodlightcontroller.storage.memory.MemoryStorageSource

net.floodlightcontroller.devicemanager.internal.DeviceManagerImpl

net.floodlightcontroller.topology.internal.TopologyImpl

net.floodlightcontroller.routing.dijkstra.RoutingImpl
```

```
net.floodlightcontroller.forwarding.Forwarding

net.floodlightcontroller.core.OFMessageFilterManager

net.floodlightcontroller.staticflowentry.StaticFlowEntryPusher

net.floodlightcontroller.perfmon.PktInProcessingTime

net.floodlightcontroller.restserver.RestApiServer

net.floodlightcontroller.learningswitch.LearningSwitch

net.floodlightcontroller.hub.Hub

net.floodlightcontroller.jython.JythonDebugInterface
```

3.5 启动顺序

3.5.1 发现模块

目前提供三种 map。
服务 map：从服务（service）到提供者。
模块服务 map：从模块到它提供的服务。
模块名字 map：从字符串格式的名字到模块类。

3.5.2 确定需要加载模块的最小集合

利用深度优先搜索算法找到需要加载模块的最小集合。所有配置文件中指定的模块会被添加到队列中。每当一个模块出队时，添加到将被启动的模块列表中，然后该模块的依赖模块被检查是否已经添加到被启动的模块列表中。

3.5.3 初始化需要加载的模块

调用各个模块的 init()。主要做两件事情：一是检查和处理依赖的模块。二是进行内部数据结构的初始化。

3.5.4 启动模块

在调用 init()之后，调用各个模块的 startUp()。本调用中允许调用依赖其他模块的函数。

3.6 运行控制器

3.6.1 默认配置

可以使用

```
java -jar floodlight.jar
```

3.6.2 使用多个 jar 情况

可以使用

```
java -cp floodlight.jar:/path/to/other.jar net.floodlightcontroller.core.Main
```

3.6.3 指定运行的配置文件所在

可以使用-cf 指定配置文件（默认为 src/main/resources/floodlightdefault.properties）。

```
java -cp floodlight.jar:/path/to/other.jar net.floodlightcontroller.core.Main -cf p
ath/to/config/file.properties
```

3.6.4 单独指定某个 module 的配置

有两种模式。

一种是在配置文件中具体写出某模块的配置信息，然后在该模块初始化时候进行解析，例如配置文件中添加

net.floodlightcontroller.restserver.RestApiServer.port = 8080

然后，在模块初始化 init()时执行读取

```
// read our config options

Map<String, String> configOptions = context.getConfigParams(this);

String port = configOptions.get("port");

if (port != null) {

    restPort = Integer.parseInt(port);

}
```

另一种，是在运行控制器时通过命令行参数传递，例如

```
java -Dnet.floodlightcontroller.restserver.RestApiServer.port=8080 -jar floodligh
t.jar。
```

3.7 注意事项

多模块情况下，各个模块的初始化和启动顺序随机，不能保证一定顺序。
自定义的配置文件不能与默认文件（floodlightdefault.properties）重名。
模块的构造器 contructor 不能接受参数。初始化工作尽量在 init()中完成。
多模块之间不能存在 service 上的冲突和覆盖。

第4章 应用

4.1 REST 应用

4.1.1 Circuit Pusher

Circuit Pusher 模块利用 REST API 创建一个在两端点之间的双向链路，例如创建在两个设备之间路由路径上交换机的永久的流表项。

目前仅支持两个 IP 节点，并且，在发出 restAPI 请求之前，控制器必须已经看到了两个节点。

目前支持的语法有

a) circuitpusher.py --controller={IP}:{rest port} --type ip --src {IP} --dst {IP} --add --name {circuit-name}

adds a new circuit between src and dst devices Currently ip circuit is supported. ARP is automatically supported.

Currently a simple circuit record storage is provided in a text file circuits.json in the working directory.

The file is not protected and does not clean itself between controller restarts. The file is needed for correct operation

and the user should make sure deleting the file when floodlight controller is restarted.

b) circuitpusher.py --controller={IP}:{rest port} --delete --name {circuit-name}

deletes a created circuit (as recorded in circuits.json) using the previously given name

4.2 模块应用

4.2.1 Firewall

防火墙作为一个模块，被设计为通过在交换机上的静态流表来允许或阻止流量。每个流的首包会匹配配置好的防火墙规则，防火墙规则按照优先级进行匹配包头（OF1.0）。最高优先级的匹配规则执行行动。

防火墙按照被动模式运行。规则被创建时，按照优先级进行排序。每个 packet-in 会跟所有规则进行比较，行动（ALLOW 或 DENY）被保存在 IRoutingDecision 对象中，并被传递以在后续的 packet-in 处理中使用。最终达到 Forwarding 或其他负责转发的模块。如果行动是 ALLOW，则添加一条静态的流表项，否则删除流表项。

规则之间允许存在空间的覆盖。例如：

protocol	**destination IP**	**destination port**	action	priority*
TCP	192.168.1.0/24	80	ALLOW	1
TCP	192.168.1.0/24	wildcard	DENY	2

4.2.2 Load Balancer

一个简单的负载均衡模块，为 ping、tcp、udp 流提供均衡。可以通过 REST API 来访问，格式类似于 OpenStack Quantum 的 LBaaSv1.0 API（参考 http://wiki.openstack.org/Quantum/LBaaS）。

目前存在的局限包括：客户端记录和静态流在使用后没有被清除；在不同服务器之间基于连接的轮询策略，而不是流量；未实现健康监控。

可以通过如下步骤测试该模块的基本特性：

确认 floodlight.defaultproperties 文件中启用了 net.floodlightcontroller.loadbalancer.LoadBalancer，开启 floodlight 和 mininet（至少包括 8 台主机），例如 sudo mn --controller=remote --ip=<controller_ip> --mac --topo=tree,3。在 mininet 中执行 pingall。

在部署了 load balancer vips、pools 或成员的 linux 命令行中执行：

```
#!/bin/sh



curl -X POST -d '{"id":"1","name":"vip1","protocol":"icmp","address":"10.0.0.100","port":"8"}' http://localhost:8080/quantum/v1.0/vips/

curl -X POST -d '{"id":"1","name":"pool1","protocol":"icmp","vip_id":"1"}' http://localhost:8080/quantum/v1.0/pools/

curl -X POST -d '{"id":"1","address":"10.0.0.3","port":"8","pool_id":"1"}' http://localhost:8080/quantum/v1.0/members/
```

```
curl -X POST -d '{"id":"2","address":"10.0.0.4","port":"8","pool_id":"1"}' http://localhost:8080/quantum/v1.0/members/



curl -X POST -d '{"id":"2","name":"vip2","protocol":"tcp","address":"10.0.0.200","port":"100"}' http://localhost:8080/quantum/v1.0/vips/

curl -X POST -d '{"id":"2","name":"pool2","protocol":"tcp","vip_id":"2"}' http://localhost:8080/quantum/v1.0/pools/

curl -X POST -d '{"id":"3","address":"10.0.0.5","port":"100","pool_id":"2"}' http://localhost:8080/quantum/v1.0/members/

curl -X POST -d '{"id":"4","address":"10.0.0.6","port":"100","pool_id":"2"}' http://localhost:8080/quantum/v1.0/members/



curl -X POST -d '{"id":"3","name":"vip3","protocol":"udp","address":"10.0.0.150","port":"200"}' http://localhost:8080/quantum/v1.0/vips/

curl -X POST -d '{"id":"3","name":"pool3","protocol":"udp","vip_id":"3"}' http://localhost:8080/quantum/v1.0/pools/

curl -X POST -d '{"id":"5","address":"10.0.0.7","port":"200","pool_id":"3"}' http://localhost:8080/quantum/v1.0/members/

curl -X POST -d '{"id":"6","address":"10.0.0.8","port":"200","pool_id":"3"}' http://localhost:8080/quantum/v1.0/members/
```

在 mininet 中，执行 h1 ping -c1 10.0.0.100，然后执行 h2 ping -c1 10.0.0.100。这两个 ping 会被不同的 host 处理。

4.3 OpenStack

Floodlight 可以为 OpenStack 的 Quantum 插件提供网络后端。Quantum 通过 REST API 提供了网络即服务的模型，而 Floodlight 恰好实现了相应的 API。在这个方案中，主要有两个组件：VirtualNetworkFilter（实现了 Quantum 的 API）和 Quantum RestProxy 插件（将 Floodlight 连接到 Quantum）。

VirtualNetworkFilter 实现了基于 MAC 的二层网络隔离，并向外提供 API。这个模块是 Flo

odlight 中的自有模块，无需依赖 OpenStack 环境。

RestProxy 插件，作为 OpenStack Quantum 服务的一部分。如果想要了解 Quantum，可以参考 https://wiki.openstack.org/wiki/Quantum。

4.3.1 安装

可以在 ubuntu12.04+中安装 Floodlight+OpenStack。

4.3.1.1 安装 Floodlight

通过如下命令安装。

```
$ sudo apt-get update

$ sudo apt-get install zip default-jdk ant

$ wget --no-check-certificate https://github.com/floodlight/floodlight/archive/master.zip

$ unzip master.zip

$ cd floodlight-master; ant

$ java -jar target/floodlight.jar -cf src/main/resources/quantum.properties
```

为了验证 VirtualNetworkFilter 已经被成功激活，可以执行如下命令进行检测。

```
$ curl 127.0.0.1:8080/networkService/v1.1

{"status":"ok"}
```

4.3.1.2 安装 OpenStack 和 RestProxy 插件

通过 install-devstack 脚本来安装 OpenStack 的 Folsom 版本。

```
$ wget https://github.com/bigswitch/devstack/archive/floodlight/folsom.zip

$ unzip folsom.zip

$ cd devstack-floodlight-folsom
```

```
Use your favorite editor to edit ./localrc with 'BS_FL_CONTROLLERS_PORT=<flo
odlight IP address>:8080'. If you have run floodlight in the same VM, them use
127.0.0.1 for <floodlight IP address>; otherwise, use the IP address of whicheve
r VM or host where you run floodlight. Then:

$ ./stack.sh
```

如果安装成功，会提示类似如下信息：

```
Horizon is now available at http://10.10.2.15/

Keystone is serving at http://10.10.2.15:5000/v2.0/

Examples on using novaclient command line is in exercise.sh

The default users are: admin and demo

The password: nova

This is your host ip: 10.10.2.15

stack.sh completed in 102 seconds.
```

4.3.2 验证安装

4.3.3 Quantum REST 代理插件

该模块提供了一个 quantum 插件“QuantumRestProxy”，尅也将 quantum 的函数调用转换为认证的 REST 请求，发给相关的外部网络控制器。

该模块可以让网络控制器代码跟插件解耦，允许：

- 为 quantum 和网络控制器提供独立的认证和冗余解决；
- 为 quantum 和网络控制器提供独立的升级和开发。

同时该模块保存了用 API 设置的本地持久的 quantum 状态，以方便：

- 独立的更新和恢复外部的网络控制器；
- 为查询提供快速的本地回复。

4.3.3.1 下行的 REST API

该代理使用的下行 API 跟为 quantum 本身定义的 API 几乎完全相同。此外，还需要一个对额外 PUT 的支持，以对持久性数据进行批处理 dump。

4.3.3.2 使用插件

Floodlight 控制器使用该插件提供基于 openflow/VLAN 的虚拟网络，需要四个组件：

- 一个运行 nova 和 quantum 服务的 openstack 控制器；
- 一个运行 VirtualNetworkFilter 的 Floodlight 控制器；
- 本插件，转换 nova/quantum api 为向 VirtualNetworkFilter 的 REST 请求；
- 在每个 nova-compute 节点上都要运行 OpenvSwitch。

4.3.3.3 配置插件

Ovs 配置

所有的 OpenvSwitch 必须配置上控制器信息，可以在所有的 nova-compute 节点上利用如下的脚本来完成。

```
NETWORK_CONTROLERS=<comma-seperated-list-of-network-ctrls>

sudo ovs-vsctl \--no-wait \-\- \--if-exists del-br br-int

sudo ovs-vsctl \--no-wait add-br br-int

sudo ovs-vsctl \--no-wait br-set-external-id br-int bridge-id br-int

for ctrl in `echo ${NETWORK_CONTROLERS} \| tr ',' ' '`

do

 sudo ovs-vsctl set-controller br-int "tcp:${ctrl}:6633"

done
```

Plugin 配置（quantum server 节点上）

1、首先必须安装了 MySQL。

```
$ mysql -u root -p$PASS -e 'DROP DATABASE IF EXISTS restproxy_quantum;'

$ mysql -u root -p$PASS -e 'CREATE DATABASE IF NOT EXISTS restproxy_quant
um;'
```

2、编辑/etc/quantum/quantum.conf 文件，修改为

```
[DEFAULT]

core_plugin = quantum.plugins.bigswitch.plugin.QuantumRestProxyV2

allow_overlapping_ips = False
```

```
lock_path = <path_to_which_quantum_process_can_write_to>
```

其中，lock_path 仅在利用包安装的时候需要设置。当从 devstack 安装的时候，默认的 lock_path 值是允许的。

3、编辑/etc/nova/nova.conf，设置为

```
libvirt_vif_type=ethernet

libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtHybridOVSBridgeDriver
```

4、编辑/etc/quantum/plugins/restproxy/restproxy.ini，设置为

```
[DATABASE]

sql_connection = mysql://<username>:<password>@<database_ip>:3306/restproxy_quantum

[RESTPROXY]

servers=<controller_ip:port_num>,<controller_ip:port>

serverauth=<username>:<password>

serverssl=False
```

样例配置为

```
[DATABASE]

sql_connection = mysql://root:pass@127.0.0.1:3306/restproxy_quantum

[RESTPROXY]

servers=192.168.1.100:80,192.168.1.101:80

serverauth=user:pass

serverssl=False
```

5、开启 quantum 服务，并用参数指定配置文件。

```
cd <quantum_path> && python <quantum_path>/bin/quantum-server --config-file /etc/quantum/quantum.conf --config-file /etc/quantum/plugins/restproxy/restproxy.ini
```

4.3.3.4 单元测试

在 Quantum 的顶级目录下测试相关的插件。

```
./run_tests.sh quantum.tests.unit.bigswitch.test_restproxy_plugin
```

4.3.3.5 用简单控制器测试

可以使用如下命令来运行一个简单的控制器。

```
cd <quantum_path>/plugins/bigswitch/tests/

python test_server
```

在 restproxy.ini 中配置“servers”属性为 127.0.0.1:8899，并运行 quantum 服务器。之后可以利用 Quantum CLI 来进行测试。

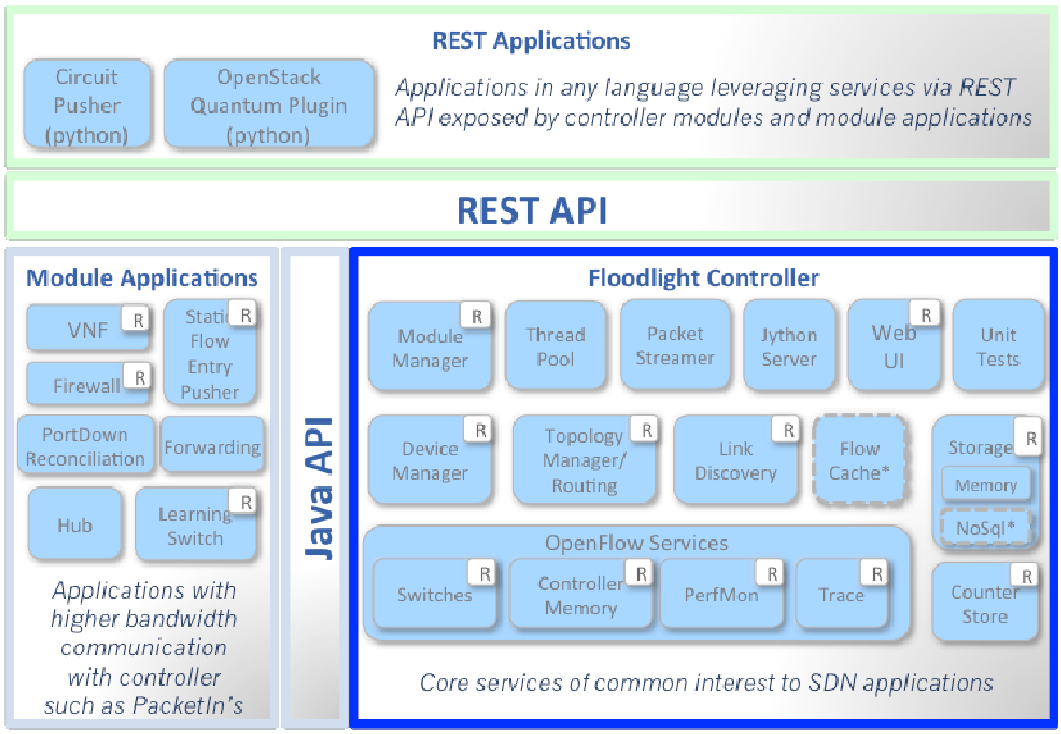
第5章 真实用例

5.1 Coraid-软件定义的存储

介绍的信息在 http://www.projectfloodlight.org/blog/2012/08/31/floodlight-use-case-software-defined-storage/。

第6章 架构

Floodlight 包括 controller 和运行在其上的应用。应用跟 controller 之间可以通过 java 模块方式或者 REST API 方式相互合作。



开发者可以使用 java 模块系统方式（实现 IFloodlightModule）来创建自己的应用。从功能上，应用程序包括控制器模块（controller modules）和应用模块（application modules）两大类。前者实现核心的网络功能，后者可以实现一些自定义的操作。

6.1 控制器模块

目前，控制器模块包括
FloodlightProvider (Dev)
DeviceManagerImpl (Dev)
LinkDiscoveryManager (Dev)
TopologyService (Dev)
RestApiServer (Dev)
ThreadPool (Dev)
MemoryStorageSource (Dev)
Flow Cache (API only)
Packet Streamer

6.1.1 FloodlightProvider (Dev)

提供两个主要功能：处理与交换机之间的连接，将 OF 消息转换为事件，以供其他模块监

听获取。另外负责确定 OF 消息在各个监听模块之间的分发顺序。模块自身可以通过返回值确定消息继续由下一个监听模块处理，还是停止。

6.1.1.1 提供服务

IfloodlightProviderService。

6.1.1.2 服务依赖

IStorageSourceService
IPktinProcessingTimeService
IRestApiService
ICounterStoreService
IThreadPoolService

6.1.1.3 源文件

net.floodlightcontroller.core.FloodlightProvider

6.1.1.4 配置

该模块默认启用，不需要在配置文件中进行指定。相关配置选项有

Name	Type	Default	Description
openflowport	Int	6633	The TCP port to listen on for OpenFlow connections from devices that support OpenFlow.
workerthreads	Int	0 (2 * # of CPUs)	The number of Netty threads to spawn. If this number is 0 it will default to twice the number of CPUs available on the machine.
controllerid	String	localhost	The ID of the controller.
role	String	master	Values can either be: master, slave, equal. Slave controllers will not accept connections from switches and will be on "cold" standby.

6.1.1.5 REST API

URI	Description	Arguments
/wm/core/switch/all/<statType>/json	Retrieve aggregate stats across all switches.	**statType**: port, queue, flow, aggregate, desc, table, features, host
/wm/core/switch/<switchId>/<statType>/json	Retrieve per switch stats.	**switchId**: Valid Switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) **statType**: port, queue, flow,

		aggregate, desc, table, features, host
/wm/core/controller/switches/json	List of all switch DPIDs connected to the controller.	none
/wm/core/role/json	Gets the current controller role.	None.
/wm/core/counter/<counterTitle>/json	List of global traffic counters in the controller (across all switches).	**counterTitle**: "all" or something of the form DPID_*Port#OFEventL3/4*_Type. See CounterStore.java for details.
/wm/core/counter/<switchId>/<counterName>/json	List of traffic counters per switch.	**switchId**: Valid Switch DPID CounterTitle: see above
/wm/core/memory/json	Current controller memory usage.	none
/wm/core/module/{all}/json	Returns information about modules and their dependencies.	**all:** "all" or "loaded".

6.1.2 DeviceManagerImpl (Dev)

该模块负责追踪设备在网络中的移动，为新的流确定目标设备。

6.1.2.1 提供服务

IDeviceService

6.1.2.2 服务依赖

IStorageSourceService
IRestApiService
ICounterStoreService
IThreadPoolService
IFlowReconcileService
IFloodlightProviderService

6.1.2.3 源代码模块

net.floodlightcontroller.devicemanager.internal.DeviceManagerImpl

6.1.2.4 实现

该模块通过 PacketIn 请求来感知设备。根据分类器的创建过程来分类设备，默认利用 mac 地址和 vlan 来区分设备，也可以通过 ip 地址等进行辨识。交换机收到 packetin 请求，则创建一个挂载点。

6.1.2.5 限制

设备是无法修改的，因此，不能拥有一个设备的引用，只能通过 IdeviceService 提供的 API 来进行查询。

6.1.2.6 配置

该模块默认启用，不需要在配置文件中进行指定。

6.1.2.7 REST API

URI	Description	Arguments
/wm/device/	List of all devices tracked by the controller. This includes MACs, IPs, and attachment points.	Passed as GET parameters: **mac** (colon-separated hex-encoded), **ipv4** (dotted decimal),**vlan**, **dpid** attachment point DPID (colon-separated hex-encoded) and **port** the attachment point port.

样例
获取所有设备

```
curl -s http://localhost:8080/wm/device/
```

获取 ip 地址为 1.1.1.1 的设备

```
curl -s http://localhost:8080/wm/device/?ipv4=1.1.1.1
```

6.1.3 LinkDiscoveryManager (Dev)

该模块负责发现和维护网络中的链路状态。

6.1.3.1 提供服务

ILinkDiscoveryService

6.1.3.2 服务依赖

IStorageSourceService
IThreadPoolService
IFloodlightProviderService

6.1.3.3 源代码模块

net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager

6.1.3.4 实现

链路发现服务同时采用 LLDP 和广播包（BDDP）进行链路探测。两者的目标 mac 地址分别为 01:80:c2:00:00:0e 和 ff:ff:ff:ff:ff:ff。以太类型分别为 0x88cc 和 0x8999。探测能成功实现需要交换机满足两个条件：一个是支持 LLDP 包的转发，一个是支持二层的广播。

6.1.3.5 限制

无

6.1.3.6 配置

该模块默认启用，不需要在配置文件中进行指定。

6.1.3.7 REST API

URI	Description	Arguments
/wm/topology/links/json	List of all links detected by the controller.	None.

样例
获取上所有的设备

```
curl -s http://localhost:8080/wm/topology/links/json
```

6.1.4 TopologyService (Dev)

该模块维护网络的拓扑信息。

6.1.4.1 提供服务

ITopologyService
IRoutingService

6.1.4.2 服务依赖

ILinkDiscoveryService
IThreadPoolService
IFloodlightProviderService
IRestApiService

6.1.4.3 源代码模块

net.floodlightcontroller.topology.TopologyManager

6.1.4.4 实现

利用 IlinkDiscoveryService 发现的链路状态信息计算拓扑信息。注意拓扑信息同时维护了 OF 岛的概念（同一个控制器下互相连接的一组 OF 交换机）。拓扑信息不可被人为修改，其他模块如果想监听拓扑信息，则需要实现 ItopologyListener 接口。

6.1.4.5 限制

拓扑中，如果一台非 OF 交换机连接到 OF 岛，则不能存在多条连接。

6.1.4.6 配置

该模块默认启用，不需要在配置文件中进行指定。

6.1.4.7 REST API

URI	Description	Arguments
/wm/topology/switchclusters/json	Lists the switch clusters computed by the controller.	None.

样例
获取所有设备

```
curl -s http://localhost:8080/wm/topology/switchclusters/json
```

6.1.5 RestApiServer (Dev)

让模块可以通过 HTTP 发出 REST API。

6.1.5.1 提供服务

IRestApiService

6.1.5.2 服务依赖

无。

6.1.5.3 源代码模块

net.floodlightcontroller.restserver.RestApiServer

6.1.5.4 实现

该模块使用了 Restlets 库。其他模块可以通过实现 RestletRoutable 来作为一个 REST 服务器。每个 RestletRoutable 包括一个 Router，用来绑定 Restlet 资源（常见的资源为 ServerResource）。用户将挂载自己的类（扩展了一个 Restlet 资源），以便提供对指定 URL 的请求处理。资源内部的@Get 和@Put 等，实现不同响应 http 请求的方法。

序列化由 Jackson 库来实现（已被包含到 Restlet 库），Jackson 库有两种方式来序列化，一种是自动序列化对象内支持 getters 的属性，一种是采用用户自定义的序列化方法。

6.1.5.5 限制

URL 路径必须唯一。
Restlet 只能通过服务接口获取模块的数据。因此，模块必须提供接口来获取相关数据。

6.1.5.6 配置

该模块默认启用，不需要在配置文件中进行指定。相关配置选项有

Name	Type	Default	Description
port	Int	8080	The TCP port to listen on for HTTP connections.

6.1.5.7 REST API

无。

6.1.6 ThreadPool (Dev)

利用 java ScheduledExecutorService 进行的封装，该模块提供线程支持。

6.1.6.1 提供服务

IThreadPoolService

6.1.6.2 服务依赖

无。

6.1.6.3 源代码模块

net.floodlightcontroller.threadpool.ThreadPool

6.1.6.4 实现

参考 http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ScheduledExecutorService.html。

6.1.6.5 限制

无。

6.1.6.6 配置

该模块默认启用，不需要在配置文件中进行指定。

6.1.6.7 REST API

无。

6.1.7 MemoryStorageSource (Dev)

维护 NOSQL 风格的内存资源。当数据库被改动时支持提供通知。

6.1.7.1 提供服务

IStorageSourceService

6.1.7.2 服务依赖

ICounterStoreService
IRestApiService

6.1.7.3 源代码模块

net.floodlightcontroller.storage.memory.MemoryStorageSource

6.1.7.4 实现

其他模块，只要实现了 IstorageSourceService 接口，就可以使用内存资源。所有的数据是共享的。

6.1.7.5 限制

主程序关闭后，所有状态清除。
数据没有隔离保护。

6.1.7.6 配置

该模块默认启用，不需要在配置文件中进行指定。

6.1.7.7 REST API

无。

6.1.8 Flow Cache (API only)

负责处理一系列的网络事件。

6.1.8.1 提供服务

6.1.8.2 服务依赖

6.1.8.3 源代码模块

6.1.8.4 实现

6.1.8.5 限制

6.1.8.6 配置

该模块默认启用，不需要在配置文件中进行指定。相关配置选项有

6.1.8.7 REST API

样例

6.1.9 Packet Streamer

将交换机和控制器之间的 openflow 网包变成流，发送到一个 observer。

6.1.9.1 提供服务

6.1.9.2 服务依赖

6.1.9.3 源代码模块

6.1.9.4 实现

6.1.9.5 限制

6.1.9.6 配置

该模块默认启用，不需要在配置文件中进行指定。相关配置选项有

6.1.9.7 REST API

过滤器利用 **Error! Hyperlink reference not valid.**中的 POST 请求。例如

```
{'mac':<hostMac>, 'direction':<direction>, 'period':<period>, 'sessionId':<sessionid>}
```

Name	Value	Description
mac	<hostMac>	The OFMessage with matching hostMac (in the Ethernet frame in its payload) will be streamed.
direction	in	OFPacketIn
	out	OFPacketOut and FlowMod
	both	in and out
period	<period>	Defines the duration of the streaming session in seconds.
	-1	terminate the given session
sessionid	<sessionid>	The session to be terminated when period = -1. Otherwise, it is ignored.

REST API 返回 session id，用于从流 thrift 服务器收取网包，返回的是 json 格式。

```
{'sessionId':<sessionid>}
```

下面的 python 称许创建了一个流 session，并提供关闭 session 的函数。

```
url = 'http://%s:8080/wm/core/packettrace/json' % controller
```

```
filter = {'mac':host, 'direction':'both', 'period':1000}

post_data = json.dumps(filter)

request = urllib2.Request(url, post_data, {'Content-Type':'application/json'})

response_text = None


try:

    response = urllib2.urlopen(request)

    response_text = response.read()

except Exception, e:

    # Floodlight may not be running, but we don't want that to be a fatal

    # error, so we just ignore the exception in that case.

    print "Exception:", e

    exit


if not response_text:

    print "Failed to start a packet trace session"

    sys.exit()


response_text = json.loads(response_text)
```

```
sessionId = None
if "sessionId" in response_text:
sessionId = response_text["sessionId"]


def terminateTrace(sid):
    global controller


    filter = {SESSIONID:sid, 'period':-1}
    post_data = json.dumps(filter)
    url = 'http://%s:8080/wm/core/packettrace/json' % controller
    request = urllib2.Request(url, post_data, {'Content-Type':'application/json'})
    try:
        response = urllib2.urlopen(request)
        response_text = response.read()
    except Exception, e:
        # Floodlight may not be running, but we don't want that to be a fatal
        # error, so we just ignore the exception in that case.
        print "Exception:", e
```

基于 thrift 的流服务。接口如下：

```
service PacketStreamer {
```

```
/**
 * Synchronous method to get packets for a given sessionid
 */
list<binary> getPackets(1:string sessionid),

/**
 * Synchronous method to publish a packet.
 * It ensure the order that the packets are pushed
 */
i32 pushMessageSync(1:Message packet),

/**
 * Asynchronous method to publish a packet.
 * Order is not guaranteed.
 */
oneway void pushMessageAsync(1:Message packet)

/**
 * Terminate a session
```

```
   */

   void terminateSession(1:string sessionid)

}
```

floodlight 为 thrift 服务产生 java 和 python 库。

REST API 提供了创建流 session 的功能。一旦 session id 被创建，thrift 接口、getPackets(sessionId)可以被用于接受给定 session 的 OF 网包。terminateSession(sessionId)可以用于关闭 session。

一个 python 程序例子如下

```
try:

   # Make socket

   transport = TSocket.TSocket('localhost', 9090)

   # Buffering is critical. Raw sockets are very slow

   transport = TTransport.TFramedTransport(transport)

   # Wrap in a protocol

   protocol = TBinaryProtocol.TBinaryProtocol(transport)

   # Create a client to use the protocol encoder

   client = PacketStreamer.Client(protocol)

   # Connect!

   transport.open()



   while 1:

      packets = client.getPackets(sessionId)
```

```
        for packet in packets:
            print "Packet: %s"% packet
            if "FilterTimeout" in packet:
                sys.exit()


except Thrift.TException, e:
    print '%s' % (e.message)
    terminateTrace(sessionId)
```

现有的 floodlight 归功了一个基于 mac 的包-流服务。包的格式参考 net/floodlightcontroller/core/OFMessageFilterManager.java。

6.2 应用模块

6.2.1 VirtualNetworkFilter

一个简单的二层逻辑网络虚拟化模块，可以利用一个物理二层网络创建多个逻辑二层网络。可以提供给 OpenStack 使用，也可以独立使用。

6.2.1.1 提供服务

IVirtualNetworkService

6.2.1.2 服务依赖

IDeviceService
IFloodlightProviderService
IRestApiService

6.2.1.3 源代码模块

net.floodlightcontroller.virtualnetwork.VirtualNetworkFilter

6.2.1.4 实现

用户创建虚拟网络后，在 forwarding 模块前，本模块首先接收 PacketIn 消息。一旦收到 PacketIn 消息，模块先查找源 mac 地址和目的 mac 地址。如果源和目的 mac 地址都在同一个虚拟网络，则返回 Command.CONTINUE。如果源和目的 mac 地址在不同的网络，则返回 Command.STOP。

6.2.1.5 限制

物理网络必须都在同一个二层域中。
每个虚拟网络最多有一个虚拟网关。
多播和广播包没被隔离。
DHCP 流量都被允许。

6.2.1.6 配置

该模块默认未被启用，需要在配置文件 src/main/resources/quantum.properties 中进行指定。修改为

```
# The default configuration for openstack

floodlight.modules = net.floodlightcontroller.storage.memory.MemoryStorageSource,\

net.floodlightcontroller.staticflowentry.StaticFlowEntryPusher,\

net.floodlightcontroller.forwarding.Forwarding,\

net.floodlightcontroller.jython.JythonDebugInterface,\

net.floodlightcontroller.counter.CounterStore,\

net.floodlightcontroller.perfmon.PktInProcessingTime,\

net.floodlightcontroller.ui.web.StaticWebRoutable,\

net.floodlightcontroller.virtualnetwork.VirtualNetworkFilter

net.floodlightcontroller.restserver.RestApiServer.port = 8080

net.floodlightcontroller.core.FloodlightProvider.openflowport = 6633

net.floodlightcontroller.jython.JythonDebugInterface.port = 6655
```

如果使用了 Floodlight VM，则配置文件默认存在，只需要启用即可。

```
floodlight@localhost:~$ touch /opt/floodlight/floodlight/feature/quantum

floodlight@localhost:~$ sudo service floodlight stop
```

```
floodlight@localhost:~$ sudo service floodlight start
```

6.2.1.7 REST API

URI	Method	URI Arguments	Data	Data Fields	Description
/quantum/v1.0/tenants/{tenant}/networks/{network}	PUT/POST/DELETE	Tenant: Currently ignored Network: The ID (not name) of the network	{"network": { "gateway": "<IP>", "name": "<Name>" }}\	IP: Gateway IP in "1.1.1.1" format, can be null Name: Network name a string	Creates a new virtual network. Name and ID are required, gateway is optional.
/quantum/v1.0/tenants/{tenant}/networks/{network}/ports/{port}/attachment	PUT/DELETE	Tenant: Currently ignored Network: The ID (not name) of the network Port: Logical port name	{"attachment": {"id": "<Network ID>", "mac": "<MAC>"}}	Network ID: Network ID as a string, the one you just created MAC: MAC address in "00:00:00:00:00:09" format	Attaches a host to a virtual network.
/quantum/v1.0/tenants/{tenant}/networks	GET	Tenant: Currently	None	None	Shows all networks and

		ignored			their gateway, ID, and hosts mac in json format

样例

创建一个名为“VirtualNetwork1”，id 为“NetworkId1”的虚拟网络，网关为 10.0.0.7，租户是“default”（忽略）。

```
curl -X PUT -d '{ "network": { "gateway": "10.0.0.7", "name": "virtualNetwork1" } }
' http://localhost:8080/quantum/v1.0/tenants/default/networks/NetworkId1
```

添加一个地址为 00:00:00:00:00:08、端口为 port1 的主机到虚拟网络“VirtualNetwork1”中。

```
curl -X PUT -d '{"attachment": {"id": "NetworkId1", "mac": "00:00:00:00:00:08"}}'
http://localhost:8080/quantum/v1.0/tenants/default/networks/NetworkId1/port
s/port1/attachment
```

6.2.2 Forwarding

该模块在两个设备之间转发网包。源设备和目标设备将被 IdeviceService 区分。

6.2.2.1 提供服务

无。

6.2.2.2 服务依赖

IDeviceService
IFloodlightProviderService
IRestApiService
IRoutingService
ITopologyService
ICounterStoreService

6.2.2.3 源代码模块

net.floodlightcontroller.forwarding.Forwarding

6.2.2.4 实现

模块将发现所有的 OF 岛。FlowMod 将被安装到最短路径上。如果一条 PacketIn 被接收到一个 OF 岛，而该岛没有挂载点，则这个网包将被洪泛。

6.2.2.5 限制

路由功能目前未被提供。
不包括 vlan 的加减包头。

6.2.2.6 配置

该模块默认启用，不需要在配置文件中进行指定。相关配置选项有

6.2.2.7 REST API

无。

6.2.3 Firewall

根据 ACL 规则进行流量过滤。
每个被首包触发的 PacketIn 消息将跟规则集进行匹配。按照最高权值匹配规则行为进行处理。

6.2.3.1 提供服务

6.2.3.2 服务依赖

6.2.3.3 源代码模块

6.2.3.4 实现

6.2.3.5 限制

6.2.3.6 配置

该模块默认启用，不需要在配置文件中进行指定。相关配置选项有

6.2.3.7 REST API

URI	Method	URI Arguments	Data	Data Fields	Description
/wm/firewall/module/<op>/json	GET	**op**: status, enable, disable, storageRules, subnet-mask	None	None	query the status of, enable, and disable the firewall
/wm/firewall/rules/json	GET	None	None	None	List all existing rules in

					json format
	POST	None	{"<field 1>":"<value 1>", "<field 2>":"<value 2>", ...}	"**field**":"**value**" pairs below in any order and combination: "**switchid**":"<xx:xx:xx:xx:xx:xx:xx:xx>", "**src-inport**":"<short>", "**src-mac**": "<xx:xx:xx:xx:xx:xx>", "**dst-mac**": "<xx:xx:xx:xx:xx:xx>", "**dl-type**": "<ARP *or* IPv4>", "**src-ip**": "<A.B.C.D/M>", "**dst-ip**": "<A.B.C.D/M>", "**nw-proto**": "<TCP *or* UDP *or*ICMP>", "**tp-src**": "<short>", "**tp-dst**": "<short>", "**priority**": "<int>", "**action**": "<ALLOW *or* DENY>" Note: specifying src-ip/dst-ip without specifying dl-type as ARP, or specifying any IP-based nw-proto will automatically set dl-type to match IPv4.	Create new firewall rule
	DELETE	None	{"<ruleid>":"<int>"}	"**ruleid**": "<int>" Note: ruleid is a random number generated and returned in the json response upon successful creation	Delete a rule by ruleid

样例

6.2.4 PortDown Reconciliation

当某个端口或者链路 down 的时候，该模块调整流量。该模块将删除所有往故障端口发出的 flow，删除后重新计算所有流路径并更新拓扑。

6.2.4.1 提供服务

6.2.4.2 服务依赖

6.2.4.3 源代码模块

6.2.4.4 实现

6.2.4.5 限制

6.2.4.6 配置

该模块默认启用，不需要在配置文件中进行指定。相关配置选项有

6.2.4.7 REST API

样例

第7章开发指南

7.1 编写模块

配置好 eclipse 之后，在 eclipse 的 package 浏览器中找到目录 src/main/java。
右键，选择新建类。
在 Package 框中填写 net.floodlightcontroller.mactracker。
在 Name 框填写 MACTracker。
在 Interface 框，选择 Add，然后进行搜索，添加 IOFMessageListener 和 IFloodlightModule。
点击 Finish。
eclipse 会自动生成相关的代码。之后需要手动添加代码。

7.1.1 添加依赖和初始化

首先，添加如下依赖

```
import net.floodlightcontroller.core.IFloodlightProviderService;

import java.util.ArrayList;

import java.util.concurrent.ConcurrentSkipListSet;

import java.util.Set;

import net.floodlightcontroller.packet.Ethernet;

import org.openflow.util.HexString;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;
```

在类中定义使用的变量。需要监听 OF 消息，所以注册 IfloodlightProviderService，用 set 结构来存储 mac 地址，最后，用 logger 来记录日志。

```
protected IFloodlightProviderService floodlightProvider;

protected Set macAddresses;

protected static Logger logger;
```

修改 getModuleDependecies()，告诉 module loader，需要依赖它。

```
@Override
public Collection<Class<? extends IFloodlightService>> getModuleDependencies() {
    Collection<Class<? extends IFloodlightService>> l =
        new ArrayList<Class<? extends IFloodlightService>>();
    l.add(IFloodlightProviderService.class);
    return l;
}
```

创建初始化函数，加载依赖并初始化数据结构。

```
@Override
public void init(FloodlightModuleContext context) throws FloodlightModuleException {
    floodlightProvider = context.getServiceImpl(IFloodlightProviderService.class);
    macAddresses = new ConcurrentSkipListSet<Long>();
    logger = LoggerFactory.getLogger(MACTracker.class);
}
```

7.1.2 处理 Packet-In 消息

首先，在 startUp 方法中，注册 PACKET_IN 消息的处理。

```
@Override
public void startUp(FloodlightModuleContext context) {
```

```
    floodlightProvider.addOFMessageListener(OFType.PACKET_IN, this);

}
```

同时，需要为 OFMessage listener 提供 id，这可以利用 getName()函数实现。

```
@Override

public String getName() {

    return MACTracker.class.getSimpleName();

}
```

现在，定义 PACKET_IN 消息才处理机制，在最后返回 Command.CONTINUE 将允许这套消息继续被其他的 PACKET_IN handler 处理。

```
@Override

  public net.floodlightcontroller.core.IListerner.Command receive(IOFSwitch sw,
OFMessage msg, FloodlightContext cntx) {

      Ethernet eth =

            IFloodlightProviderService.bcStore.get(cntx,

                                    IFloodlightProviderService.CONTEXT_PI_PAYLOAD);



      Long sourceMACHash = Ethernet.toLong(eth.getSourceMACAddress());

      if (!macAddresses.contains(sourceMACHash)) {

          macAddresses.add(sourceMACHash);

          logger.info("MAC Address: {} seen on switch: {}",

                HexString.toHexString(sourceMACHash),
```

```
                sw.getId());

        }

        return Command.CONTINUE;

    }
```

7.1.3 注册模块

通过注册模块，告诉 Floodlight 在启动时加载模块。
首先要告诉 Floodlight 该模块存在，需要在
src/main/resources/META-INF/services/net.floodlight.core.module.IfloodlightModule
中添加一行

```
net.floodlightcontroller.mactracker.MACTracker
```

之后要告诉加载器进行加载，需要修改模块配置文件，默认为
src/main/resources/floodlightdefault.properties 中修改 floodlight.modules 变量。
修改为

```
floodlight.modules = <leave the default list of modules in place>, net.floodlight
controller.mactracker.MACTracker
```

运行 Main.java 即可启动 controller。

7.2 添加服务到模块

控制器中最为核心的是 core 模块，负责监听 of 套接字并且分发事件。其他的次要模块可以注册到 core 模块上，获取并处理事件。在程序启动后可以从日志或 debug 输出中看到注册的信息，例如

```
17:29:23.231 [main] DEBUG n.f.core.internal.Controller - OFListeners for PACKET
_IN: devicemanager,

17:29:23.231 [main] DEBUG n.f.core.internal.Controller - OFListeners for PORT_S
TATUS: devicemanager,

17:29:23.237 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.
floodlightcontroller.restserver.RestApiServer
```

17:29:23.237 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.forwarding.Forwarding

17:29:23.237 [main] DEBUG n.f.forwarding.Forwarding - Starting net.floodlightcontroller.forwarding.Forwarding

17:29:23.237 [main] DEBUG n.f.core.internal.Controller - OFListeners for PACKET_IN: devicemanager,forwarding,

17:29:23.237 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.storage.memory.MemoryStorageSource

17:29:23.240 [main] DEBUG n.f.restserver.RestApiServer - Adding REST API routable net.floodlightcontroller.storage.web.StorageWebRoutable

17:29:23.242 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.core.OFMessageFilterManager

17:29:23.242 [main] DEBUG n.f.core.internal.Controller - OFListeners for PACKET_IN: devicemanager,forwarding,messageFilterManager,

17:29:23.242 [main] DEBUG n.f.core.internal.Controller - OFListeners for PACKET_OUT: messageFilterManager,

17:29:23.242 [main] DEBUG n.f.core.internal.Controller - OFListeners for FLOW_MOD: messageFilterManager,

17:29:23.242 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.routing.dijkstra.RoutingImpl

17:29:23.247 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.core.CoreModule

17:29:23.248 [main] DEBUG n.f.core.internal.Controller - Doing controller internal setup

17:29:23.251 [main] INFO n.f.core.internal.Controller - Connected to storage source

```
17:29:23.252 [main] DEBUG n.f.restserver.RestApiServer - Adding REST API routable net.floodlightcontroller.core.web.CoreWebRoutable

17:29:23.252 [main] DEBUG n.f.c.module.FloodlightModuleLoader - Starting net.floodlightcontroller.topology.internal.TopologyImpl

17:29:23.254 [main] DEBUG n.f.core.internal.Controller - OFListeners for PACKET_IN: topology,devicemanager,forwarding,messageFilterManager,

17:29:23.254 [main] DEBUG n.f.core.internal.Controller - OFListeners for PORT_STATUS: devicemanager,topology,
```

产生事件的消息类型有很多，但最基本的一类都是 PACKET_IN 消息。下面以处理 PACKET_IN 消息为例，解释如何创建服务和提供 REST API。

7.2.1 添加类

在 eclipse 的 package 浏览器中找到目录 src/main/java。
右键，选择新建类。
在 Package 框中填写 net.floodlightcontroller.pktinhistory。
在 Name 框填写 PktInHistory。
在 Interface 框，选择 Add，然后进行搜索，添加 IOFMessageListener 和 IFloodlightModule。
点击 Finish。
eclipse 会自动生成相关的代码。之后需要手动添加代码。
产生如下代码

```
package net.floodlightcontroller.pktinhistory;



import java.util.Collection;

import java.util.Map;



import org.openflow.protocol.OFMessage;

import org.openflow.protocol.OFType;
```

```
import net.floodlightcontroller.core.FloodlightContext;

import net.floodlightcontroller.core.IOFMessageListener;

import net.floodlightcontroller.core.IOFSwitch;

import net.floodlightcontroller.core.module.FloodlightModuleContext;

import net.floodlightcontroller.core.module.FloodlightModuleException;

import net.floodlightcontroller.core.module.IFloodlightModule;

import net.floodlightcontroller.core.module.IFloodlightService;


public class PktInHistory implements IFloodlightModule, IOFMessageListener {


    @Override

    public String getName() {

        // TODO Auto-generated method stub

        return null;

    }


    @Override

    public int getId() {

        // TODO Auto-generated method stub

        return 0;
```

```
    }

    @Override
    public boolean isCallbackOrderingPrereq(OFType type, String name) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean isCallbackOrderingPostreq(OFType type, String name) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public Command receive(IOFSwitch sw, OFMessage msg, FloodlightContext c
ntx) {
        // TODO Auto-generated method stub
        return null;
    }
```

```
    @Override
    public Collection<Class<? extends IFloodlightService>> getModuleServices()
{
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Map<Class<? extends IFloodlightService>, IFloodlightService>
            getServiceImpls() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Collection<Class<? extends IFloodlightService>>
            getModuleDependencies() {
        // TODO Auto-generated method stub
        return null;
    }
```

```
    @Override
    public
        void
        init(FloodlightModuleContext context)
                                    throws FloodlightModuleException {
        // TODO Auto-generated method stub

    }

    @Override
    public void startUp(FloodlightModuleContext context) {
        // TODO Auto-generated method stub

    }

}
```

7.2.2 设置模块依赖

生成类之后，添加变量

```
protected IFloodlightProviderService floodlightProvider;
```

设置模块的依赖信息

```
@Override

public Collection<Class<? extends IFloodlightService>> getModuleDependencies() {

    Collection<Class<? extends IFloodlightService>> l = new ArrayList<Class<? extends IFloodlightService>>();

    l.add(IFloodlightProviderService.class);

    return l;

}
```

初始化内部变量。

```
@Override

public void init(FloodlightModuleContext context) throws FloodlightModuleException {

    floodlightProvider = context.getServiceImpl(IFloodlightProviderService.class);

}
```

7.2.3 处理 Openflow 消息

本部分实现对 of packet_in 消息的处理，利用一个 buffer 来存储近期收到的 of 消息，以备查询。

在 startup()中注册监听器，告诉 provider 我们希望处理 OF 的 PacketIn 消息。

```
@Override

public void startUp(FloodlightModuleContext context) {

    floodlightProvider.addOFMessageListener(OFType.PACKET_IN, this);

}
```

为 OFMessage 监听器提供 id 信息。

```
@Override

public String getName() {

   return "PktInHistory";

}
```

作为类内部变量，创建 circular buffer（import 相关包），存储 packet in 消息。

```
protected ConcurrentCircularBuffer<SwitchMessagePair> buffer;
```

在初始化过程中初始化该变量。

```
@Override

public void init(FloodlightModuleContext context) throws FloodlightModuleExc
eption {

   floodlightProvider = context.getServiceImpl(IFloodlightProviderService.class);

   buffer = new ConcurrentCircularBuffer<SwitchMessagePair>(SwitchMessage
Pair.class, 100);

}
```

编写收到 packet in 消息后的处理过程。

```
@Override

public Command receive(IOFSwitch sw, OFMessage msg, FloodlightContext cnt
x) {

   switch(msg.getType()) {

       case PACKET_IN:

          buffer.add(new SwitchMessagePair(sw, msg));

          break;
```

```
        default:
            break;
    }
    return Command.CONTINUE;
}
```

7.2.4 添加 rest API

在实现了一个完整的模块之后，我们可以实现一个 rest api，来获取该模块的相关信息。需要完成两件事情：利用创建的模块导出一个服务，并把该服务绑到 REST API 模块。

具体说来，注册一个新的 Restlet，包括

在 net.floodlightcontroller.controller.internal.Controller 中注册一个 restlet。

实现一个*WebRoutable 类。该类实现了 RestletRoutable，并提供了 getRestlet()和 basePath()函数。

实现一个*Resource 类，该类扩展了 ServerResource()，并实现了@Get 或@Put 函数。

下面具体来看该如何实现。

7.2.4.1 创建并绑定接口 IPktInHistoryService

首先在 pktinhistory 包中创建一个从 IFloodlightService 扩展出来的接口 IPktInHistoryService（IPktInHistoryService.java），该服务拥有一个方法 getBuffer()，来读取 circular buffer 中的信息。

```
package net.floodlightcontroller.pktinhistory;

import net.floodlightcontroller.core.module.IFloodlightService;
import net.floodlightcontroller.core.types.SwitchMessagePair;

public interface IPktinHistoryService extends IFloodlightService {
    public ConcurrentCircularBuffer<SwitchMessagePair> getBuffer();
}
```

现在回到原先创建的 PktInHistory.java。让它具体实现 IpktInHistoryService 接口，

```
public class PktInHistory implements IFloodlightModule, IPktinHistoryService, I
OFMessageListener {
```

并实现服务的 getBuffer()方法。

```
@Override

public ConcurrentCircularBuffer<SwitchMessagePair> getBuffer() {

   return buffer;
```

}

通过修改 PktInHistory 模块中 getModuleServices()和 getServiceImpls()方法通知模块系统，我们提供了 IPktInHistoryService。

```
@Override

public Collection<Class<? extends IFloodlightService>> getModuleServices() {

   Collection<Class<? extends IFloodlightService>> l = new ArrayList<Class<?
extends IFloodlightService>>();

   l.add(IPktinHistoryService.class);

   return l;

}



@Override

public Map<Class<? extends IFloodlightService>, IFloodlightService> getServic
eImpls() {

   Map<Class<? extends IFloodlightService>, IFloodlightService> m = new Has
hMap<Class<? extends IFloodlightService>, IFloodlightService>();
```

```
    m.put(IPktinHistoryService.class, this);

    return m;

}
```

getServiceImpls()会告诉模块系统，本类（PktInHistory）是提供服务的类。

7.2.4.2 添加变量引用 REST API 服务

之后，需要添加 REST API 服务的引用（需要 import 相关包）。

```
protected IRestApiService restApi;
```

并添加 IRestApiService 作为依赖，这需要修改 init()和 getModuleDependencies()。

```
@Override

public Collection<Class<? extends IFloodlightService>> getModuleDependencies() {

    Collection<Class<? extends IFloodlightService>> l = new ArrayList<Class<? extends IFloodlightService>>();

    l.add(IFloodlightProviderService.class);

    l.add(IRestApiService.class);

    return l;

}



@Override

public void init(FloodlightModuleContext context) throws FloodlightModuleException {

    floodlightProvider = context.getServiceImpl(IFloodlightProviderService.class);
```

```
    restApi = context.getServiceImpl(IRestApiService.class);

    buffer = new ConcurrentCircularBuffer<SwitchMessagePair>(SwitchMessage
Pair.class, 100);

}
```

7.2.4.3 创建 REST API 相关的类 PktInHistoryResource 和 PktInHistoryWebRoutable

现在创建用在 REST API 中的类，包括两部分，创建处理 url call 的类和注册到 REST API 的类。

首先创建处理 REST API 请求的类 PktInHistoryResource（PktInHistoryResource.java）。当请求到达时，该类将返回 circular buffer 中的内容。

```
package net.floodlightcontroller.pktinhistory;



import java.util.ArrayList;

import java.util.List;



import net.floodlightcontroller.core.types.SwitchMessagePair;



import org.restlet.resource.Get;

import org.restlet.resource.ServerResource;



public class PktInHistoryResource extends ServerResource {

    @Get("json")

    public List<SwitchMessagePair> retrieve() {
```

```
        IPktinHistoryService pihr = (IPktinHistoryService)getContext().getAttributes
().get(IPktinHistoryService.class.getCanonicalName());

        List<SwitchMessagePair> l = new ArrayList<SwitchMessagePair>();

        l.addAll(java.util.Arrays.asList(pihr.getBuffer().snapshot()));

        return l;

    }

}
```

现在创建 PktInHistoryWebRoutable 类（PktInHistoryWebRoutable.java），负责告诉 REST API 我们注册了 API 并将它的 URL 绑定到指定的资源上。

```
package net.floodlightcontroller.pktinhistory;



import org.restlet.Context;

import org.restlet.Restlet;

import org.restlet.routing.Router;



import net.floodlightcontroller.restserver.RestletRoutable;



public class PktInHistoryWebRoutable implements RestletRoutable {

    @Override

    public Restlet getRestlet(Context context) {

        Router router = new Router(context);
```

```
        router.attach("/history/json", PktInHistoryResource.class);
        return router;
    }

    @Override
    public String basePath() {
        return "/wm/pktinhistory";
    }
}
```

并将 Restlet PktInHistoryWebRoutable 注册到 REST API 服务，这通过修改 PktInHistory 类中的 startUp()方法来完成。

```
@Override
public void startUp(FloodlightModuleContext context) {
    floodlightProvider.addOFMessageListener(OFType.PACKET_IN, this);
    restApi.addRestletRoutable(new PktInHistoryWebRoutable());
}
```

7.2.4.4 自定义序列化类

数据会被 Jackson 序列化为 REST 格式。如果需要指定部分序列化，需要自己实现序列化类 OFSwitchImplJSONSerializer（OFSwitchImplJSONSerializer.java，位于 net.floodlightcontroller.web.serialzers 包中），并添加到 net.floodlightcontroller.web.serialzers 包。

```
package net.floodlightcontroller.core.web.serializers;
```

```
import java.io.IOException;


import net.floodlightcontroller.core.internal.OFSwitchImpl;


import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.SerializerProvider;
import org.openflow.util.HexString;


public class OFSwitchImplJSONSerializer extends JsonSerializer<OFSwitchImpl>
 {


    /**
     * Handles serialization for OFSwitchImpl
     */
    @Override
    public void serialize(OFSwitchImpl switchImpl, JsonGenerator jGen,
                          SerializerProvider arg2) throws IOException,
                                                  JsonProcessingException {
        jGen.writeStartObject();
```

```
        jGen.writeStringField("dpid", HexString.toHexString(switchImpl.getId()));
        jGen.writeEndObject();
    }

    /**
     * Tells SimpleModule that we are the serializer for OFSwitchImpl
     */
    @Override
    public Class<OFSwitchImpl> handledType() {
        return OFSwitchImpl.class;
    }

}
```

现在需要告诉 Jackson 使用我们的序列器。打开 OFSwitchImpl.java（位于 net.floodlightcontroller.core.internal 包），修改如下（需要 import 我们创建的 OFSwitchImplJSONSerializer 包）

```
@JsonSerialize(using=OFSwitchImplJSONSerializer.class)
public class OFSwitchImpl implements IOFSwitch {
```

7.2.5 载入模块

首先告诉 loader 我们的模块存在，添加模块名字到 src/main/resources/META-INF/services/net.floodlight.core.module.IfloodlightModule。

```
net.floodlightcontroller.pktinhistory.PktInHistory
```

然后告知模块需要被加载。修改模块配置文件 src/main/resources/floodlightdefault.properties

中的 floodlight.modules 变量。

```
floodlight.modules = net.floodlightcontroller.staticflowentry.StaticFlowEntryPusher,\

net.floodlightcontroller.forwarding.Forwarding,\

net.floodlightcontroller.pktinhistory.PktInHistory
```

7.2.6 测试

启动 mininet。

```
mn --controller=remote --ip=[Your IP Address] --mac --topo=tree,2

*** Adding controller

*** Creating network

*** Adding hosts:

h1 h2 h3 h4

*** Adding switches:

s5 s6 s7

*** Adding links:

(h1, s6) (h2, s6) (h3, s7) (h4, s7) (s5, s6) (s5, s7)

*** Configuring hosts

h1 h2 h3 h4

*** Starting controller

*** Starting 3 switches

s5 s6 s7
```

```
*** Starting CLI:
```

启动后，运行

```
mininet> pingall

*** Ping: testing ping reachability

h1 -> h2 h3 h4

h2 -> h1 h3 h4

h3 -> h1 h2 h4

h4 -> h1 h2 h3

*** Results: 0% dropped (0/12 lost)
```

利用 REST URL 拿到结果

```
curl -s http://localhost:8080/wm/pktinhistory/history/json | python -mjson.tool

[

  {

    "message": {

      "bufferId": 256,

      "inPort": 2,

      "length": 96,

      "lengthU": 96,

      "packetData":  "MzP/Uk+PLoqIUk+Pht1gAAAAABg6/wAAAAAAAAAAAAAAAAAAAAD/AgAAAAAAAAAAAH/Uk+PhwAo2gAAAAD+gAAAAAAACyKiP/+Uk+P",
```

```
        "reason": "NO_MATCH",
        "totalLength": 78,
        "type": "PACKET_IN",
        "version": 1,
        "xid": 0
      },
      "switch": {
        "dpid": "00:00:00:00:00:00:00:06"
      }
    },
    {
      "message": {
        "bufferId": 260,
        "inPort": 1,
        "length": 96,
        "lengthU": 96,
        "packetData":  "MzP/Uk+PLoqIUk+Pht1gAAAAABg6/wAAAAAAAAAAAAAAAAAAAAAD/AgAAAAAAAAAAAH/Uk+PhwAo2gAAAD+gAAAAAACyKiP/+Uk+P",
        "reason": "NO_MATCH",
        "totalLength": 78,
```

```
        "type": "PACKET_IN",
        "version": 1,
        "xid": 0
    },
    "switch": {
        "dpid": "00:00:00:00:00:00:00:05"
    }
},
```

7.3 REST 应用

利用支持的 REST API 构建程序可能是最快速的开发方式。

可以使用任意开发语言来撰写 REST 应用，主要步骤为：

- 确定应用需要的网络服务和信息。
- 检查提供的 Floodlight REST API 列表，找到支持这些服务的 API。
- 有了所需的 API 调用后，设计和编写应用。
- 实现测试，并添加应用的服务模块和 API。

一个例子可以在 floodlight/apps 目录下的 Circuit Pusher 应用中找到。

该例子展示了如何在一个 OpenFlow cluster 中两个 IP 节点之间创建一条静态链路，按照上面的步骤如下：

1、需要的网络服务和信息：

- 两个 IP 节点的网络挂载点。
- 挂载点之间的路由。
- 在路由沿线所有交换机上配置转发的服务。

2、查找 Floodlight 的 REST API。

- /wm/device/的 GET 操作可以获取设备的挂载点信息（例如 IP 地址）
- /wm/topology/route/<switchIdA>/<portA>/<switchIdB>/<portB>/json 的 GET 操作可以在挂载点上（交换机，端口）找到路由。
- /wm/staticflowentrypusher/json 的 POST 操作可以配置流表项到指定的交换机。

3、应用设计（源代码）

- 决定用 python 来开发。
- 决定用 os.popen 来发出 curl 命令，来进行 REST API 调用。
- 明确/vm/device 语法，为两台主机解析挂载点的交换机。
- 明确/wm/topology/route 返回两个挂载点的交换机端口对。
- 对每个交换机对口对，通过/wm/staticflowentrypusher/json 安装流表项：
  - ether-type '0x0800', port M → N
  - ether-type '0x0806', port M → N

    - ether-type '0x0800', port N → M
    - ether-type '0x0806', port N → M
- 通过 ping 来测试是否已经连通。
- 添加删除静态流表项的操作。
- 利用一个简单的 text 文件来记录曾经添加的静态流表项。
- 完成测试撰写。

apps/ciruitpusher 是利用 python 解析和发出 REST API 指令。支持的 API 有

URI	Method	Description	Arguments
/wm/core/switch/all/<statType>/json		Retrieve aggregate stats across all switches	**statType**: port, queue, flow, aggregate, desc, table, features
/wm/core/switch/<switchId>/<statType>/json		Retrieve per switch stats	**switchId**: Valid Switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) **statType**: port, queue, flow, aggregate, desc, table, features
/wm/core/controller/switches/json		List of all switch DPIDs connected to the controller	none
/wm/core/counter/<counterTitle>/json		List of global traffic counters in the controller (across all switches)	**counterTitle**: "all" or something of the form DPID_*Port#OFEventL3/4*_Type. See CounterStore.java for details.
/wm/core/counter/<switchId>/<counterName>/json		List of traffic counters per switch	**switchId**: Valid Switch DPID CounterTitle: see above
/wm/core/memory/json		Current controller memory usage	none
/wm/topology/links/json		List all the inter-switch links. Note that these are only for switches connected to the same controller. This is not available in the 0.8 release.	none
/wm/topology/switc		List of all switch clusters connected to	none

hclusters/json		the controller. This is not available in the 0.8 release.	
/wm/device/		List of all devices tracked by the controller. This includes MACs, IPs, and attachment points.	Passed as GET parameters: **mac** (colon-separated hex-encoded), **ipv4** (dotted decimal), **vlan**, **dpid** attachment point DPID (colon-separated hex-encoded) and **port** the attachment point port.
/wm/staticflowentrypusher/json		Add/Delete static flow	HTTP POST data (add flow), HTTP DELETE (for deletion)
/wm/staticflowentrypusher/list/<switch>/json		List static flows for a switch or all switches	**switch**: Valid Switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) or "all"
/wm/staticflowentrypusher/clear/<switch>/json		Clear static flows for a switch or all switches	**switch**: Valid Switch DPID (XX:XX:XX:XX:XX:XX:XX:XX) or "all"
More information available on How to Use Static Flow Pusher API			
/quantum/v1.0/tenants/<tenant>/networks/<network>	PUT/POST/DELETE	Creates a new virtual network. Name and ID are required, gateway is optional.	URI argument: **tenant**: Currently ignored. **network**: ID (not name) of the network HTTP data: {"network": { "gateway": "<IP>", "name": "<Name>" }} **IP**: Gateway IP in "1.1.1.1" format, can be null **Name**: Network name as string

/quantum/v1.0/tenants/<tenant>/networks/<network>/ports/<port>/attachment	PUT/DELETE	Attaches a host to a virtual network.	URI argument: **tenant**: Currently ignored. **network**: ID (not name) of the network. **port**: Logical port name HTTP data: {"attachment": {"id": "<Network ID>", "mac": "<MAC>"}} **Network ID**: Network ID as string, the one assigned at create **MAC**: MAC address in "XX:XX:XX:XX:XX:XX" format
/quantum/v1.0/tenants/<tenant>/networks	GET	Shows all networks and their gateway, ID, and hosts mac in json format.	URI argument: **tenant**: Currently ignored.
More information available on Virtual Network Filter REST API			
/wm/firewall/module/<op>/json	GET		query the status of, enable, and disable the firewall
/wm/firewall/rules/json	GET/POST/DELETE	GET: None POST: {"<field 1>":"<value 1>", "<field 2>":"<value 2>", ...} DELETE: {"<ruleid>":"<int>"}	List all existing rules in json format Create new firewall rule Delete a rule by ruleid "**field**":"**value**" pairs below in any order and combination: "**switchid**":"<xx:xx:xx:xx:xx:xx:xx:xx>", "**src-inport**":"<short>", "**src-mac**": "<xx:xx:xx:xx:xx:xx>", "**dst-mac**": "<xx:xx:xx:xx:xx:xx>", "**dl-type**":

			"<ARP *or* IPv4>", "**src-ip**": "<A.B.C.D/M>", "**dst-ip**": "<A.B.C.D/M>", "**nw-proto**": "<TCP *or* UDP *or* ICMP>", "**tp-src**": "<short>", "**tp-dst**": "<short>", "**priority**": "<int>", "**action**": "<ALLOW *or* DENY>" "**ruleid**": "<int>"

推荐的实现步骤为
确定需要的网络/设备信息
GET 是发现信息，POST 是安装流到指定的交换机上。

第8章 编写测试

8.1 Floodlight 测试

8.2 单元测试

第9章 测试控制器性能

9.1 配置

修改 src/main/resources/floodlightdefault.properties 如下，尽量减少 load 的模块个数。

```
floodlight.modules = net.floodlightcontroller.learningswitch.LearningSwitch,net.
floodlightcontroller.counter.NullCounterStore,net.floodlightcontroller.perfmon.
NullPktInProcessingTime
```

执行 ant 编译 floodlight，执行 java -jar target/floodlight.jar 运行 floodlight。

9.2 Cbench

cbench 可以为新流产生 packet in 事件来测试 of 控制器。

9.2.1 安装

执行

```
sudo apt-get install autoconf automake libtool libsnmp-dev libpcap-dev

git clone git://gitosis.stanford.edu/openflow.git

cd openflow; git checkout -b mybranch origin/release/1.0.0

git clone git://gitosis.stanford.edu/oflops.git

cd oflops ; sh ./boot.sh ; ./configure --with-openflow-src-dir=<absolute path to
 openflow branch>; make

cd oflops ; sudo make install
```

9.2.2 运行

各个参数含义为
M，每个交换机上的 MAC（主机）个数
s，模拟的交换机个数
t，吞吐量测试模式
例如

```
cbench -c localhost -p 6633 -m 10000 -l 10 -s 16 -M 1000 -t
```