

SDN 软件定义网络基础操作手册

一、OpenDayLight-Carbon：

ODL 拥有一套模块化、可插拔灵活地控制平台作为核心，这个控制平台基于 Java 开发，理论上可以运行在任何支持 Java 的平台上，其官方文档推荐的最佳运行环境是最新的 Linux (Ubuntu 12.04+) 及 JVM1.7+。

ODL 控制器采用 OSGI 框架，SGI 框架是面向 Java 的动态模型系统，它实现了一个优雅、完整和动态的组件模型，应用程序(Bundle)无需重新引导可以被远程安装、启动、升级和卸载，通过 OSGI 捆绑可以灵活地加载代码与功能，实现功能隔离，解决了功能模块可扩展问题，同时方便功能模块的加载与协同工作。

ODL 控制平台引入了 SAL，SAL 北向连接功能模块，以插件的形式为之提供底层设备服务，南向连接多种协议，屏蔽不同协议的差异性，为上层功能模块提供一致性服务，使得上层模块与下层模块之间的调用相互隔离。SAL 可自动适配底层不同设备，使开发者专注于业务应用的开发。

此外，ODL 控制平台采用了 Infinispan 技术，Infinispan 是一个高扩展性、高可靠性、键值存储的分布式数据网格平台，选用 Infinispan 来实现数据的存储、查找及监听，用开源网格平台实现 controller 的集群。

OpenDaylight 发布了第六个版本——碳 (Carbon)，OpenDaylight 碳版本的发布增加新的功能，以更好地支持城域以太网、有线运营商以及物联网 (IoT) 部署。碳版本进一步提升了平台的可扩展性和稳定性，支持多地多点部署，并增加了应用程序性能和容

错能力的新功能。南向协议的 OpenFlow 和 Netconf 在可扩展性和性能方面以及各种管理应用程序获得成功。

二、Mininet

Mininet 是一个强大的网络仿真平台，通过这个平台，我们可以很方便的模拟真实环境中的网络操作与架构。当前 SDN/OpenFlow 发展的如火如荼，但是在真实网络中又不可以进行相关的网络实验，自然需要一个仿真平台可以对这种新型的网络架构，而 Mininet 就应运而生，承担了这个光荣而艰巨的使命。

Mininet 自带这个交换机(switch)、主机(host)、控制器(controller)，同时，在 mininet 上可以安装 OpenvSwitch、多种控制器 (NOX\POX\RYU\Floodlight\OpenDaylight 等)，同时，Mininet 可以运行在多种操作系统上 (windows\linux\Mac OS)，具有很强的系统兼容性。最最令人兴奋的一点是：在 Mininet 上进行的实验，可以无缝的移到真实的环境中去 (这一点还没试过，只是看到 Mininet 官网是这么说的，希望移植操作成功的大神可以不吝赐教)。

三、Open vSwitch

Open vSwitch 即开放虚拟交换标准 具体点说，Open vSwitch 是在开源的 Apache2.0 许可下的产品级质量的多层虚拟交换标准！它旨在通过编程扩展，使庞大的网络自动化 (配置、管理、维护)，同时还支持标准的管理接口和协议 (如 NetFlow，sFlow，SPAN，RSPAN，CLI，LACP，802.1ag)。总的来说，它被设计为支持分布在多个物理服务器，例如 VMware 的 vNetwork 分布式 vSwitch 或思科的 Nexus1000V。

四、软件版本

序号	软件名称	版本号	备注
1	OpenDayLight	distribution-karaf-0.6.0-Carbon	在/home 目录下
2	Mininet	2.2.2	已安装
3	OpenSwitch	2.0.2	已安装

下载通道：链接：<https://pan.baidu.com/s/1hrMF2RU> 密码：4qur


[返回上一级](#) | [全部文件](#) > [大赛配套手册](#) > [JCOS镜像](#)

☐ 文件名

☐  Win2008EnterpriseR2_64Bit.qcow2

☐  ODL集成工具.ova

基于vmware三合一

☐  odl.qcow2

基于JCOS云平台

☐  centos7_iso_50.qcow2


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

```
feature:install odl-mdsal-apidocs
```

```
feature:install odl-dluxapps-applications
```

然后通过谷歌浏览器访问页面：

<http://192.168.76.50:8181/index.html#/login> 其中 IP 地址

192.168.76.50 地址由自己来配置，实际情况应根据题意来设置。

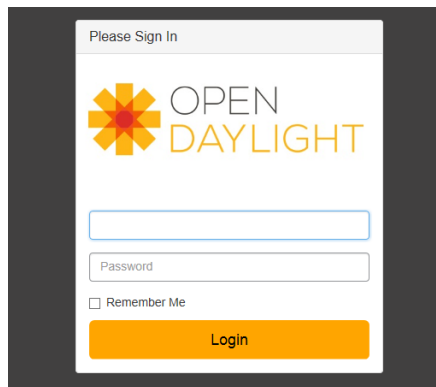
配置文件在/etc/network/interface 目录下。

```
iface eth0 inet static
```

```
address 172.16.9.100 ----- 修订成自己的 IP 地址
```

```
netmask 255.255.255.0 ----- 修订成自己的掩码地址
```

```
gateway 172.16.9.254 ----- 修订成自己的网关地址
```



登录用户名和密码都是 admin。

2. Mininet 常见的命令汇总。

miminet 常见命令：

Help: sudo mn [-h] 通过-h 可以查看到帮助信息。

- Connect to remote controller : sudo mn --controller=remote,ip=127.0.0.1,port=6633 端口可以省略，默认值是 6633，可指定。

- Topology: `sudo mn --topo=tree,n,m` 第一个参数为深度, 第二个位扇出系数。可以写成 `--tree,depth=2,fanout=8`
`single, n` : 单个交换机,n 个交换机
 - `liner, n`: 线性拓扑, n 个交换机
 - Test : `--test [pingall/pingpair..]`
 - Link : `--link=tc, bw=10M, delay=10ms, loss=5%`
 - Custom Topo : `--custom ~/mininet/custom/topo-2sw-2host.py --topo mytopo`
 - ID=MAC: `sudo mn --mac`
 - Xterm: `sudo mn -x` 启动 xterm 到每一个 host 和 switch
 - Switch type: `sudo mn --switch ovsk/user,`
 - Help: `help` 查看帮助信息
 - Exit: `exit` 退出 mininet
 - Python: `py "hello" / py dir(s1) py hello.py` 运行 python 文件 :
 - Link: `link s1 h1 down/up` 参数分别为 link 两端网元。
 - Xterm: `xterm s1/h1 xterm` 到某一个主机或交换机
 - Node: `nodes` 查看节点
 - Dump: `dump` 查看所有节点信息
- 节点命令 :
- ```
s1 ifconfig
s1 ps -a
h1 arp -s
h1 ping -c 10 h2
py h1.setIP('10.0.0.3/24')
```
- Iperf: `iperf h1 h2` 启用 iperf 功能
  - Ping: `h1 ping h2 pingall/pingpair` 发送 ping 消息
  - HTTPSERVER : `h1 python -m SimpleHTTPServer 80 &`
  - HTTPCLIENT: `h2 wget -O - h1`

通过桥接来解决访问互联网的问题 ;

```
sudo mn --custom /net.py --topo=mytopo mytopo
py h1.setIP('10.0.0.3/24')
```

### 3. OVS 常见的命令汇总。

#### 控制管理类

##### 1.查看网桥和端口

```
ovs-vsctl show
```

##### 2.创建一个网桥

```
ovs-vsctl add-br br0
ovs-vsctl set bridge br0 datapath_type=netdev
```

### 3.添加/删除一个端口

```
for system interfaces
ovs-vsctl add-port br0 eth1
ovs-vsctl del-port br0 eth1
for DPDK
ovs-vsctl add-port br0 dpdk1 -- set interface dpdk1 type=dpdk
options:dpdk-devargs=0000:01:00.0
for DPDK bonds
ovs-vsctl add-bond br0 dpdkbond0 dpdk1 dpdk2 \
-- set interface dpdk1 type=dpdk options:dpdk-devargs=0000:01:00.0 \
-- set interface dpdk2 type=dpdk options:dpdk-devargs=0000:02:00.0
```

### 4.设置/清除网桥的 openflow 协议版本

```
ovs-vsctl set bridge br0 protocols=OpenFlow13
ovs-vsctl clear bridge br0 protocols
```

### 5.查看某网桥当前流表

```
ovs-ofctl dump-flows br0
ovs-ofctl -O OpenFlow13 dump-flows br0
ovs-appctl bridge/dump-flows br0
```

### 6.设置/删除控制器

```
ovs-vsctl set-controller br0 tcp:1.2.3.4:6633
ovs-vsctl del-controller br0
```

### 7.查看控制器列表

```
ovs-vsctl list controller
```

### 8.设置/删除被动连接控制器

```
ovs-vsctl set-manager tcp:1.2.3.4:6640
ovs-vsctl get-manager
ovs-vsctl del-manager
```

### 9.设置/移除可选选项

```
ovs-vsctl set Interface eth0 options:link_speed=1G
ovs-vsctl remove Interface eth0 options link_speed
```

---

10.设置 fail 模式，支持 standalone 或者 secure

standalone(default): 清除所有控制器下发的流表，ovs 自己接管

secure: 按照原来流表继续转发

```
ovs-vsctl del-fail-mode br0
ovs-vsctl set-fail-mode br0 secure
ovs-vsctl get-fail-mode br0
```

11.查看接口 id 等

```
ovs-appctl dpif/show
```

12.查看接口统计

```
ovs-ofctl dump-ports br0
```

---

## 流表类

### 流表操作

1.添加普通流表

```
ovs-ofctl add-flow br0 in_port=1,actions=output:2
```

2.删除所有流表

```
ovs-ofctl del-flows br0
```

3.按匹配项来删除流表

```
ovs-ofctl del-flows br0 "in_port=1"
```

---

## 匹配项

1.匹配 vlan tag，范围为 0-4095



```
ovs-ofctl add-flow br0
priority=401,in_port=1,dl_vlan=777,actions=output:2
```

2.匹配 vlan pcp, 范围为 0-7

```
ovs-ofctl add-flow br0
priority=401,in_port=1,dl_vlan_pcp=7,actions=output:2
```

3.匹配源/目的 MAC

```
ovs-ofctl add-flow br0
in_port=1,dl_src=00:00:00:00:00:01/00:00:00:00:00:01,actions=output:2
ovs-ofctl add-flow br0
in_port=1,dl_dst=00:00:00:00:00:01/00:00:00:00:00:01,actions=output:2
```

4.匹配以太网类型, 范围为 0-65535

```
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,actions=output:2
```

5.匹配源/目的 IP

条件: 指定 dl\_type=0x0800, 或者 ip/tcp

```
ovs-ofctl add-flow br0
ip,in_port=1,nw_src=10.10.0.0/16,actions=output:2
ovs-ofctl add-flow br0
ip,in_port=1,nw_dst=10.20.0.0/16,actions=output:2
```

6.匹配协议号, 范围为 0-255

条件: 指定 dl\_type=0x0800 或者 ip

# ICMP

```
ovs-ofctl add-flow br0 ip,in_port=1,nw_proto=1,actions=output:2
```

7.匹配 IP ToS/DSCP, tos 范围为 0-255, DSCP 范围为 0-63

条件: 指定 dl\_type=0x0800/0x86dd, 并且 ToS 低 2 位会被忽略(DSCP 值为 ToS 的高 6 位, 并且低 2 位为预留位)

```
ovs-ofctl add-flow br0 ip,in_port=1,nw_tos=68,actions=output:2
ovs-ofctl add-flow br0 ip,in_port=1,ip_dscp=62,actions=output:2
```

8.匹配 IP ecn 位, 范围为 0-3

条件: 指定 dl\_type=0x0800/0x86dd

```
ovs-ofctl add-flow br0 ip,in_port=1,ip_ecn=2,actions=output:2
```

9.匹配 IP TTL, 范围为 0-255

```
ovs-ofctl add-flow br0 ip,in_port=1,nw_ttl=128,actions=output:2
```

10.匹配 tcp/udp, 源/目的端口, 范围为 0-65535

# 匹配源 tcp 端口 179

```
ovs-ofctl add-flow br0 tcp,tcp_src=179/0xffff0,actions=output:2
```

# 匹配目的 tcp 端口 179

```
ovs-ofctl add-flow br0 tcp,tcp_dst=179/0xffff0,actions=output:2
```

# 匹配源 udp 端口 1234

```
ovs-ofctl add-flow br0 udp,udp_src=1234/0xffff0,actions=output:2
```

# 匹配目的 udp 端口 1234

```
ovs-ofctl add-flow br0 udp,udp_dst=1234/0xffff0,actions=output:2
```

11.匹配 tcp flags

tcp flags=fin, syn, rst, psh, ack, urg, ece, cwr, ns

```
ovs-ofctl add-flow br0 tcp,tcp_flags=ack,actions=output:2
```

12.匹配 icmp code, 范围为 0-255

条件: 指定 icmp

```
ovs-ofctl add-flow br0 icmp,icmp_code=2,actions=output:2
```

13.匹配 vlan TCI

TCI 低 12 位为 vlan id, 高 3 位为 priority, 例如 tci=0xf123 则 vlan\_id 为 0x123 和 vlan\_pcp=7

```
ovs-ofctl add-flow br0 in_port=1,vlan_tci=0xf123,actions=output:2
```

14.匹配 mpls label

条件: 指定 dl\_type=0x8847/0x8848

```
ovs-ofctl add-flow br0 mpls,in_port=1,mpls_label=7,actions=output:2
```

15.匹配 mpls tc, 范围为 0-7

条件: 指定 dl\_type=0x8847/0x8848

```
ovs-ofctl add-flow br0 mpls,in_port=1,mpls_tc=7,actions=output:2
```

16.匹配 tunnel id, 源/目的 IP

# 匹配 tunnel id

```
ovs-ofctl add-flow br0 in_port=1,tun_id=0x7/0xf,actions=output:2
```

# 匹配 tunnel 源 IP

```
ovs-ofctl add-flow br0
```

```
in_port=1,tun_src=192.168.1.0/255.255.255.0,actions=output:2
```

#### # 匹配 tunnel 目的 IP

```
ovs-ofctl add-flow br0
```

```
in_port=1,tun_dst=192.168.1.0/255.255.255.0,actions=output:2
```

一些匹配项的速记符

| 速记符   | 匹配项                         |
|-------|-----------------------------|
| ip    | dl_type=0x800               |
| ipv6  | dl_type=0x86dd              |
| icmp  | dl_type=0x0800,nw_proto=1   |
| icmp6 | dl_type=0x86dd,nw_proto=58  |
| tcp   | dl_type=0x0800,nw_proto=6   |
| tcp6  | dl_type=0x86dd,nw_proto=6   |
| udp   | dl_type=0x0800,nw_proto=17  |
| udp6  | dl_type=0x86dd,nw_proto=17  |
| sctp  | dl_type=0x0800,nw_proto=132 |
| sctp6 | dl_type=0x86dd,nw_proto=132 |

| 速记符   | 匹配项            |
|-------|----------------|
| arp   | dl_type=0x0806 |
| rarp  | dl_type=0x8035 |
| mpls  | dl_type=0x8847 |
| mplsm | dl_type=0x8848 |

## 指令动作

### 1.动作为出接口

从指定接口转发出去

```
ovs-ofctl add-flow br0 in_port=1,actions=output:2
```

### 2.动作为指定 group

group id 为已创建的 group table

```
ovs-ofctl add-flow br0 in_port=1,actions=group:666
```

### 3.动作为 normal

转为 L2/L3 处理流程

```
ovs-ofctl add-flow br0 in_port=1,actions=normal
```

### 4.动作为 flood

从所有物理接口转发出去，除了入接口和已关闭 flooding 的接口

```
ovs-ofctl add-flow br0 in_port=1,actions=flood
```

### 5.动作为 all

从所有物理接口转发出去，除了入接口

```
ovs-ofctl add-flow br0 in_port=1,actions=all
```

#### 6.动作为 local

一般是转发给本地网桥

```
ovs-ofctl add-flow br0 in_port=1,actions=local
```

#### 7.动作为 in\_port

从入接口转发回去

```
ovs-ofctl add-flow br0 in_port=1,actions=in_port
```

#### 8.动作为 controller

以 packet-in 消息上送给控制器

```
ovs-ofctl add-flow br0 in_port=1,actions=controller
```

#### 9.动作为 drop

丢弃数据包操作

```
ovs-ofctl add-flow br0 in_port=1,actions=drop
```

#### 10.动作为 mod\_vlan\_vid

修改报文的 vlan id，该选项会使 vlan\_pcp 置为 0

```
ovs-ofctl add-flow br0 in_port=1,actions=mod_vlan_vid:8,output:2
```

#### 11.动作为 mod\_vlan\_pcp

修改报文的 vlan 优先级，该选项会使 vlan\_id 置为 0

```
ovs-ofctl add-flow br0 in_port=1,actions=mod_vlan_pcp:7,output:2
```

#### 12.动作为 strip\_vlan

剥掉报文内外层 vlan tag

```
ovs-ofctl add-flow br0 in_port=1,actions=strip_vlan,output:2
```

#### 13.动作为 push\_vlan

在报文外层压入一层 vlan tag，需要使用 openflow1.1 以上版本兼容

```
ovs-ofctl add-flow -O OpenFlow13 br0
```

```
in_port=1,actions=push_vlan:0x8100,set_field:4097->vlan_vid,output:2
```

ps: set field 值为 4096+vlan\_id，并且 vlan 优先级为 0，即 4096-8191，对应的 vlan\_id 为 0-4095

#### 14.动作为 push\_mpls

修改报文的 `ethertype`，并且压入一个 MPLS LSE

```
ovs-ofctl add-flow br0 in_port=1,actions=push_mpls:0x8847,set_field:10-
\>mpls_label,output:2
```

#### 15.动作为 pop\_mpls

剥掉最外层 mpls 标签，并且修改 `ethertype` 为非 mpls 类型

```
ovs-ofctl add-flow br0
mpls,in_port=1,mpls_label=20,actions=pop_mpls:0x0800,output:2
```

#### 16.动作为修改源/目的 MAC，修改源/目的 IP

##### # 修改源 MAC

```
ovs-ofctl add-flow br0
in_port=1,actions=mod_dl_src:00:00:00:00:00:01,output:2
```

##### # 修改目的 MAC

```
ovs-ofctl add-flow br0
in_port=1,actions=mod_dl_dst:00:00:00:00:00:01,output:2
```

##### # 修改源 IP

```
ovs-ofctl add-flow br0
in_port=1,actions=mod_nw_src:192.168.1.1,output:2
```

##### # 修改目的 IP

```
ovs-ofctl add-flow br0
in_port=1,actions=mod_nw_dst:192.168.1.1,output:2
```

#### 17.动作为修改 TCP/UDP/SCTP 源目的端口

##### # 修改 TCP 源端口

```
ovs-ofctl add-flow br0 tcp,in_port=1,actions=mod_tp_src:67,output:2
```

##### # 修改 TCP 目的端口

```
ovs-ofctl add-flow br0 tcp,in_port=1,actions=mod_tp_dst:68,output:2
```

##### # 修改 UDP 源端口

```
ovs-ofctl add-flow br0 udp,in_port=1,actions=mod_tp_src:67,output:2
```

##### # 修改 UDP 目的端口

```
ovs-ofctl add-flow br0 udp,in_port=1,actions=mod_tp_dst:68,output:2
```

#### 18.动作为 mod\_nw\_tos

条件：指定 `dl_type=0x0800`

修改 ToS 字段的高 6 位，范围为 0-255，值必须为 4 的倍数，并且不会去修改 ToS 低 2 位 ecn 值

```
ovs-ofctl add-flow br0 ip,in_port=1,actions=mod_nw_tos:68,output:2
```

19.动作为 mod\_nw\_ecn

条件：指定 dl\_type=0x0800，需要使用 openflow1.1 以上版本兼容

修改 ToS 字段的低 2 位，范围为 0-3，并且不会去修改 ToS 高 6 位的 DSCP 值

```
ovs-ofctl add-flow br0 ip,in_port=1,actions=mod_nw_ecn:2,output:2
```

20.动作为 mod\_nw\_ttl

修改 IP 报文 ttl 值，需要使用 openflow1.1 以上版本兼容

```
ovs-ofctl add-flow -O OpenFlow13 br0
in_port=1,actions=mod_nw_ttl:6,output:2
```

21.动作为 dec\_ttl

对 IP 报文进行 ttl 自减操作

```
ovs-ofctl add-flow br0 in_port=1,actions=dec_ttl,output:2
```

22.动作为 set\_mpls\_label

对报文最外层 mpls 标签进行修改，范围为 20bit 值

```
ovs-ofctl add-flow br0 in_port=1,actions=set_mpls_label:666,output:2
```

23.动作为 set\_mpls\_tc

对报文最外层 mpls tc 进行修改，范围为 0-7

```
ovs-ofctl add-flow br0 in_port=1,actions=set_mpls_tc:7,output:2
```

24.动作为 set\_mpls\_ttl

对报文最外层 mpls ttl 进行修改，范围为 0-255

```
ovs-ofctl add-flow br0 in_port=1,actions=set_mpls_ttl:255,output:2
```

25.动作为 dec\_mpls\_ttl

对报文最外层 mpls ttl 进行自减操作

```
ovs-ofctl add-flow br0 in_port=1,actions=dec_mpls_ttl,output:2
```

26.动作为 move NXM 字段

使用 move 参数对 NXM 字段进行操作

# 将报文源 MAC 复制到目的 MAC 字段，并且将源 MAC 改为 00:00:00:00:00:01

```
ovs-ofctl add-flow br0 in_port=1,actions=move:NXM_OF_ETH_SRC[]-
>NXM_OF_ETH_DST[],mod_dl_src:00:00:00:00:00:01,output:2
```

ps: 常用 NXM 字段参照表

| NXM 字段          | 报文字段       |
|-----------------|------------|
| NXM_OF_ETH_SRC  | 源 MAC      |
| NXM_OF_ETH_DST  | 目的 MAC     |
| NXM_OF_ETH_TYPE | 以太网类型      |
| NXM_OF_VLAN_TCI | vid        |
| NXM_OF_IP_PROTO | IP 协议号     |
| NXM_OF_IP_TOS   | IP ToS 值   |
| NXM_NX_IP_ECN   | IP ToS ECN |
| NXM_OF_IP_SRC   | 源 IP       |
| NXM_OF_IP_DST   | 目的 IP      |
| NXM_OF_TCP_SRC  | TCP 源端口    |
| NXM_OF_TCP_DST  | TCP 目的端口   |
| NXM_OF_UDP_SRC  | UDP 源端口    |



| NXM 字段          | 报文字段      |
|-----------------|-----------|
| NXM_OF_UDP_DST  | UDP 目的端口  |
| NXM_OF_SCTP_SRC | SCTP 源端口  |
| NXM_OF_SCTP_DST | SCTP 目的端口 |

## 27.动作为 load NXM 字段

使用 load 参数对 NXM 字段进行赋值操作

```
push mpls label, 并且把 10(0xa)赋值给 mpls label
ovs-ofctl add-flow br0 in_port=1,actions=push_mpls:0x8847,load:0xa-
\>NXM_OF_MPLS_LABEL[],output:2
对目的 MAC 进行赋值
ovs-ofctl add-flow br0 in_port=1,actions=load:0x001122334455-
\>NXM_OF_ETH_DST[],output:2
```

- 1
- 2
- 3
- 4

## 28.动作为 pop\_vlan

弹出报文最外层 vlan tag

```
ovs-ofctl add-flow br0
in_port=1,d1_type=0x8100,d1_vlan=777,actions=pop_vlan,output:2
```

- 1