

SDN Experiment 3

SDN Experiment 3

- 1 前言
- 2 实验环境
 - 2.1 virtualbox & ubuntu
 - 2.1.1 virtuablbox
 - 2.1.2 ubuntu16.04
 - 2.2编写P4程序
 - 2.2.1 header
 - 2.2.2 parser
 - 2.2.3 ingress
 - 2.2.4 main
 - 2.3 Wireshark
 - 2.3 VS Code
 - 2.4 Mininet
 - 2.4.1 常用命令
 - 2.4.2 自定义拓扑
 - 2.5 bmv2
 - 2.6 p4c
 - 2.7 P4Runtime
- 3 实验内容
 - 3.1 实验1
 - 题目
 - 说明
 - 3.2 实验2
 - 题目
 - 说明
- 4 总结
- 5 扩展资料

1 前言

- 《软件定义网络》课程实验总计四次，这是第三次的实验指导书
- 实验完成情况可当场验收，可提交实验报告，鼓励当场验收
- 实验虚拟机请提前在[moodle](#)下载，建议使用个人电脑运行
- 实验内容要求各位提前准备，机房现场主要负责答疑和验收
- 实验需独自完成，鼓励互相学习和交流，严禁抄袭
- 关于实验部分的疑问或反馈或Anything请发送邮件至：sdnexp2019@outlook.com

标题格式：

cs60-小胖-关于xxx

2 实验环境

本次实验主要用到的工具如下所示，提供安装好所需工具的虚拟机，也可自行参考文档手动安装

- 虚拟机及系统
virtualbox (free, GPL)
ubuntu16.04
- 网络模拟
Mininet
-软件交换机 bmv2
-P4程序编译器 p4c
- 控制器
grpc
- 抓包工具
Wireshark (或tcpdump)
- 文本编辑器
虚拟机中安装了VS Code，自选均可

以下内容为上述工具的基本教程，有熟悉的章节自行跳过

2.1 virtualbox & ubuntu

2.1.1 virtualbox

- 安装

根据自己的操作系统选择[virtualbox下载](#)，默认选项安装（以下教程在6.0.4版本测试通过）

- 虚拟机导入

从[moodle](#)下载的虚拟机，按照下图所示导入。可根据电脑配置在设置中分配更多的核心数和内存，其余选项默认

在机房电脑进行实验的同学每次结束后需将虚拟机关闭后导出，保存到自己的U盘中，下次实验再次导入继续实验

2.1.2 ubuntu16.04

本次实验使用虚拟机下载：

链接：<https://pan.baidu.com/s/1NKeJuuZddqY3ThUTbhQirw>
提取码：7gsr

实验虚拟机镜像基于ubuntu16.04 64位制作，各工具均安装在 `~/p4`

用户名：`sdn`，密码：`sdn`

2.2编写P4程序

P4是一种声明式编程语言，它主要用于编写程序以下达指令给数据平面的设备(如交换机、网卡、防火墙、过滤器等)如何处理数据包。在 `/home/sdn/p4/p4_program` 下可以看到一个简单的L2路由实例，`simple_router.p4` 和 `simple_router-16` 分别用P4-14和P4-16实现。以下简单介绍P4-16实现的版本，也更推荐大家使用P4-16。

2.2.1 header

可以自定义包头的域和各个字段以及元数据：

```
//定义ethernet
header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16> etherType;
}

//定义ipv4
header ipv4_t {
    bit<4> version;
    bit<4> ihl;
    bit<8> diffserv;
    bit<16> totalLen;
    bit<16> identification;
    bit<3> flags;
    bit<13> fragOffset;
    bit<8> ttl;
    bit<8> protocol;
    bit<16> hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}

struct headers {
    ethernet_t ethernet;
    ipv4_t ipv4;
}
```

2.2.2 parser

定义包头如何解析：

```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }

}
```

2.2.3 ingress

定义转发表table，定义动作action，以及各个表之间的关系apply。：

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {

    action drop() {
        mark_to_drop(standard_metadata);
    }

    //交换机必须对每个数据包执行以下操作：（i）更新源和目标MAC地址，（ii）递减IP头中的生存时间（TTL），以及（iii）将数据包转发到适当的端口。
    action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
        standard_metadata.egress_spec = port;
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = dstAddr;
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    }

    table ipv4_lpm {
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        actions = {
            ipv4_forward;
        }
    }
}
```

```

        drop;
        NoAction;
    }
    size = 1024;
    default_action = drop();
}

apply {
    if (hdr.ipv4.isValid()) {
        ipv4_lpm.apply();
    }
}
}

```

2.2.4 main

```

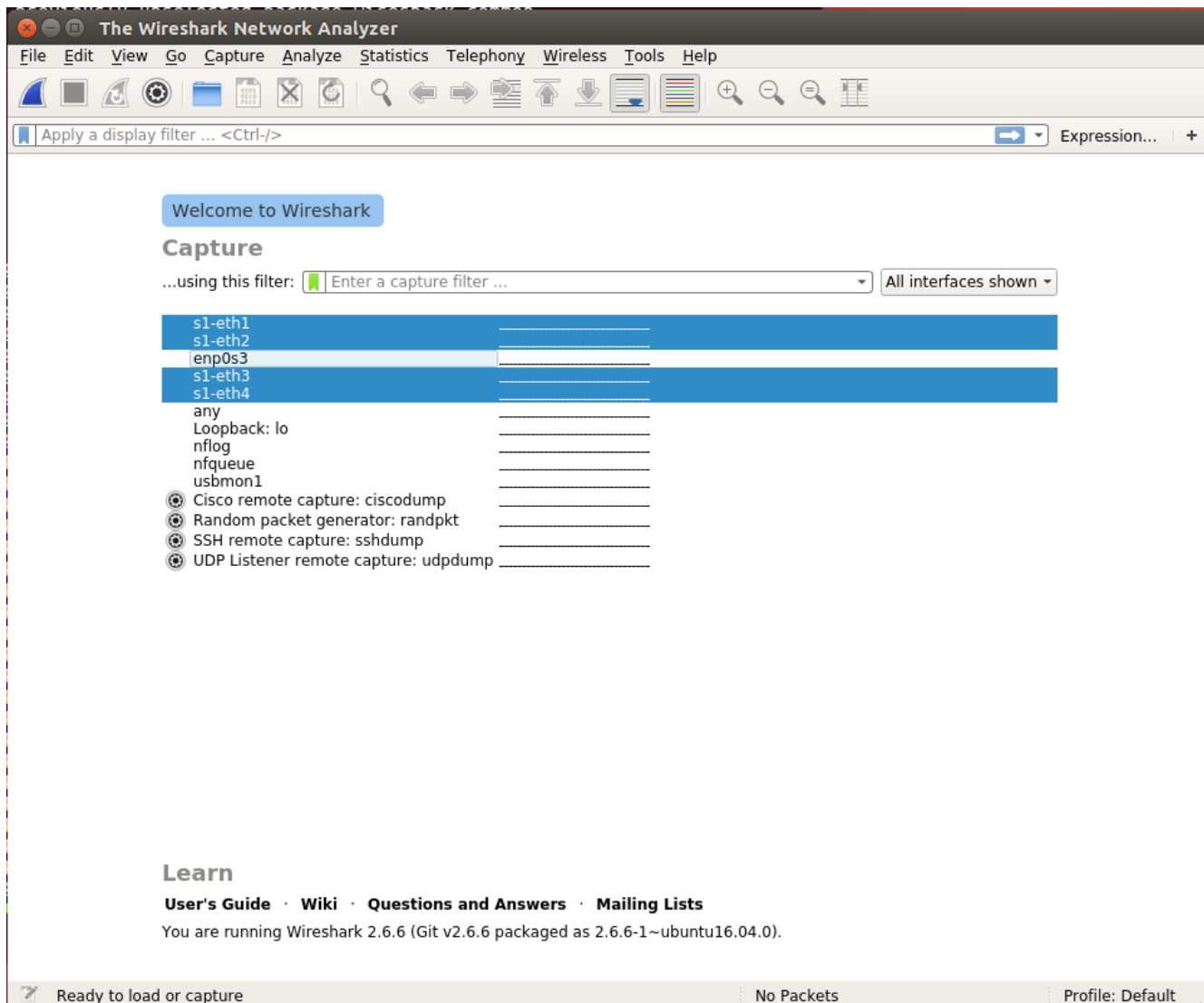
V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;

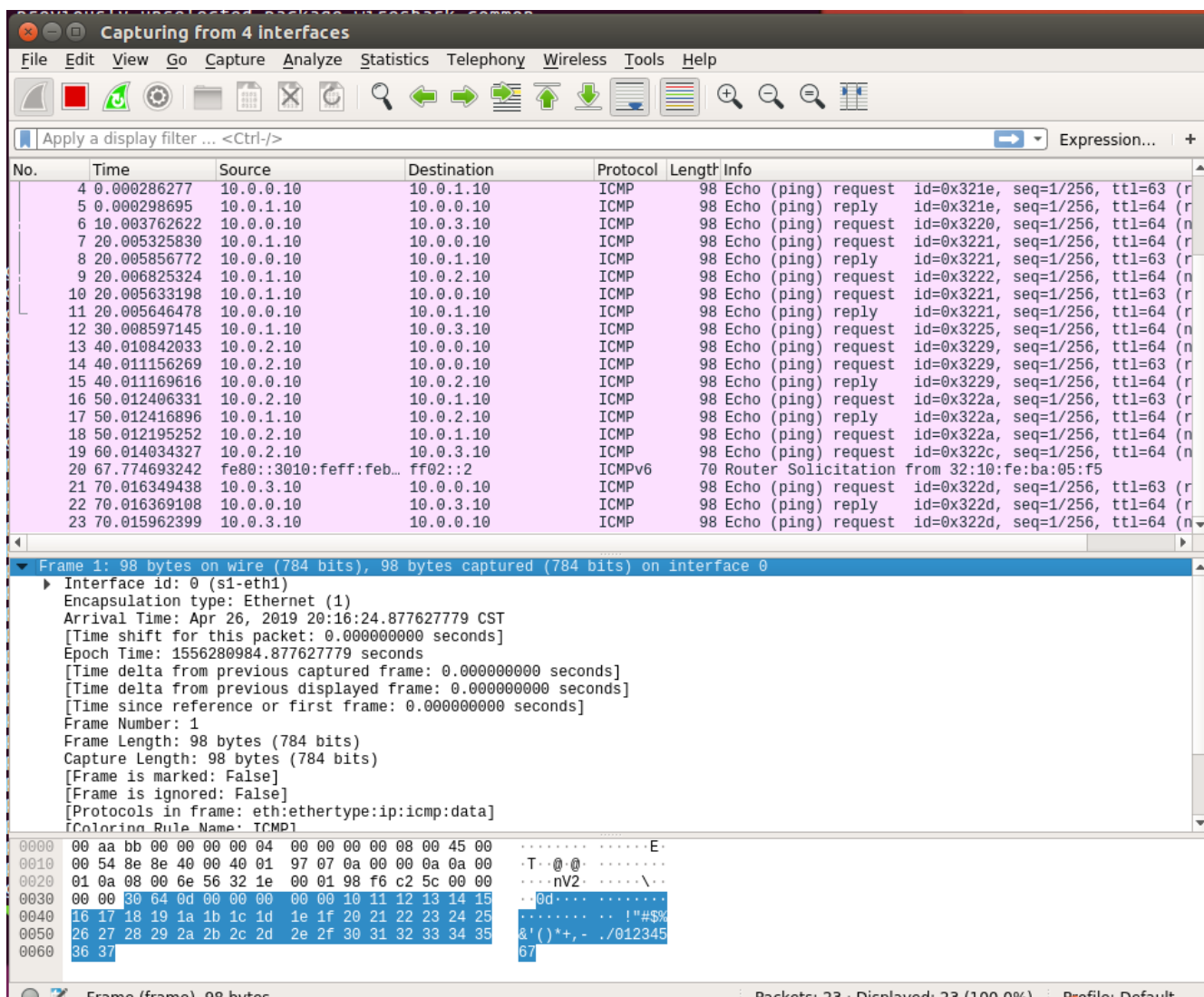
```

2.3 Wireshark

用Mininet搭建网络拓扑后，Wireshark可以抓取控制器和交换机通讯的数据包，也可以抓取指定交换机端口上的数据包

- 选择合适的端口
- 抓取loopback时需root权限，故 `sudo wireshark` 启动
- 可以查看数据包的各个字段，验证数据包经过交换机是否被正确处理





2.3 VS Code

这里选择自己习惯的文本编辑器即可

若选择VS Code

- 建议安装推荐的Python Extension
- Python Interpreter切换至 `/usr/bin/python`

这样练习时可以参考提供的示例，同时VSC可以提供舒服的命令补全

2.4 Mininet

Mininet的基本教程请阅读官网提供的[walkthrough](#)，安装方式推荐源码安装

2.4.1 常用命令

```
# shell prompt
mn -h # 查看mininet命令中的各个选项
sudo mn -c # 不正确退出时清理mininet

# 下面的命令可以在 'sudo mn' 新建的简单拓扑上查看运行结果
```



```
# mininet CLI
net # 显示当前网络拓扑
dump # 显示当前网络拓扑的详细信息
xterm h1 # 给节点h1打开一个终端模拟器
sh [COMMAND] # 在mininet命令中执行COMMAND命令
h1 ping -c3 h2 # 即h1 ping h2 3次
pingall # 即ping all
h1 ifconfig # 查看h1的网络端口及配置
h1 arp # 查看h1的arp表
link s1 h1 down/up # 断开/连接s1和h1的链路
exit # 退出mininet CLI
```

The image shows two terminal windows. The left window shows the mininet CLI setup process, including adding a controller, hosts (h1, h2), and a switch (s1), and then running a ping test from h1 to h2. The right window shows the output of the command 'sudo ovs-ofctl dump-flows s1', displaying detailed flow table information for switch s1, including cookie, duration, table, and various match/action rules.

2.4.2 自定义拓扑

- 简单的写法

示例位于 `/home/sdn/p4/behavioral-model/mininet/1sw_demo.py`, 定义了一个单个交换机, 主机数量由参数指定的拓扑, 默认的交换机model由sw_path指定, 运行的p4交换机代码由json_path指定, thrift_port指定thrift server绑定的端口, 不同的交换机设备需要绑定不同的端口, 部分代码如下,:

```
class SingleSwitchTopo(Topo):
    "Single switch connected to n (< 256) hosts."
    def __init__(self, sw_path, json_path, thrift_port, pcap_dump, n, **opts):
        # Initialize topology and default options
        Topo.__init__(self, **opts)

        switch = self.addSwitch('s1',
                                sw_path = sw_path,
                                json_path = json_path,
                                thrift_port = thrift_port,
                                pcap_dump = pcap_dump)

        for h in xrange(n):
            host = self.addHost('%d' % (h + 1),
                                ip = "10.0.%d.10/24" % h,
                                mac = '00:04:00:00:00:%02x' % h)
            self.addLink(host, switch)

def main():
```

```
num_hosts = args.num_hosts
mode = args.mode

topo = SingleSwitchTopo(args.behavioral_exe,
                        args.json,
                        args.thrift_port,
                        args.pcap_dump,
                        num_hosts)

net = Mininet(topo = topo,
              host = P4Host,
              switch = P4Switch,
              controller = None)

net.start()
```

运行拓扑的命令为：

```
cd ~/p4/behavioral-model/mininet

sudo python 1sw_demo.py --behavioral-exe ../targets/simple_router/simple_router --json
../targets/simple_router/simple_router.json
```

创建4台主机1交换机的拓扑，在mininet终端下输入 pingall 测试连通性：

```
sdn@ubuntu: ~/p4/behavioral-model/mininet
sdn@ubuntu:~/p4/behavioral-model/mininet$ sudo python 1sw_demo.py --behavioral-exe ../targets/simple_router/simple_router --json ../targets/simple_router/simple_router.json --n 4
*** Creating network
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller

*** Starting 1 switches
s1 Starting P4 switch s1.
../targets/simple_router/simple_router -i 1@s1-eth1 -i 2@s1-eth2 -i 3@s1-eth3 -i 4@s1-eth4 --thrift-port 9090 --nanolog ipc:///tmp/bm-0-log.ipc --device-id 0 ../targets/simple_router/simple_router.json
P4 switch s1 has been started.

*****
h1
default interface: eth0 10.0.0.10          00:04:00:00:00:00
*****
*****
h2
default interface: eth0 10.0.1.10          00:04:00:00:00:01
*****
*****
h3
default interface: eth0 10.0.2.10          00:04:00:00:00:02
*****
*****
h4
default interface: eth0 10.0.3.10          00:04:00:00:00:03
*****
Ready !
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X X X
h3 -> X X X
h4 -> X X X
*** Results: 100% dropped (0/12 received)
mininet> 
```

此时主机之间不连通，因为交换机上还没有流表。

2.5 bmv2

bmv2是一款P4软件交换机，可以直接执行p4c编译P4程序生成的JSON文件，并且提供直观的CLI来控制交换机运行时状态和行为。

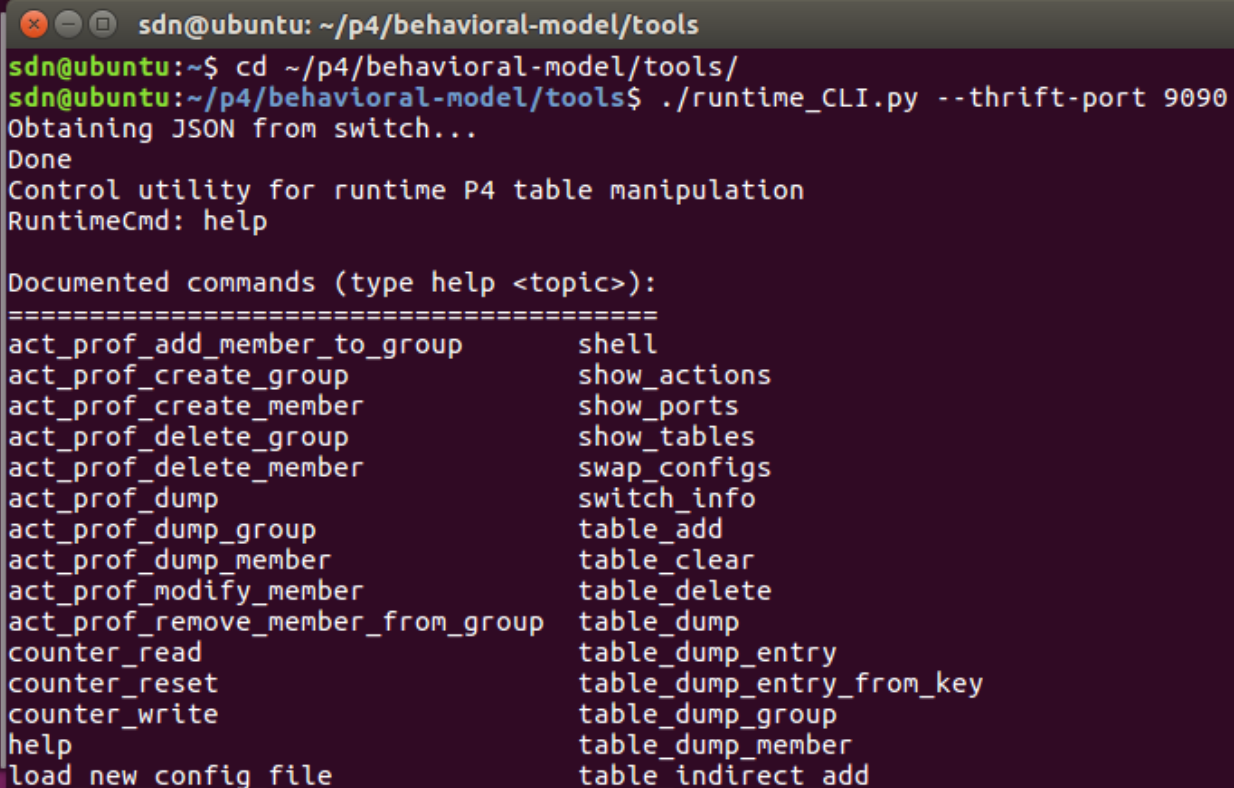
- 用bmv2来执行一个P4程序编译的JSON程序，例如使用simple_switch模型，与表示与交换机绑定的端口（eg:port0 and port1）：

```
cd ~/p4/behavioral-model/targets/simple_switch
sudo ./simple_switch -i 0@<iface0> -i 1@<iface1> <path to JSON file>
```

- 使用CLI下发流表。CLI连接到在每个交换机进程中运行的Thrift RPC服务器。9090是默认值(例如2.4.2中CLI运行在9090)，但当然，如果在计算机上运行多个交换机设备，则需要为每个设备提供不同的端口。一个CLI实例只能连接到一个交换机设备。可以这样使用CLI：

```
cd ~/p4/behavioral-model/tools/
./runtime_CLI.py --thrift-port 9090
```

打开另一个terminal，2.4.2中拓扑交换机默认绑定端口为9090，打开2.4.2拓扑中的p4交换机的CLI：



```
sdn@ubuntu: ~/p4/behavioral-model/tools
sdn@ubuntu:~$ cd ~/p4/behavioral-model/tools/
sdn@ubuntu:~/p4/behavioral-model/tools$ ./runtime_CLI.py --thrift-port 9090
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: help

Documented commands (type help <topic>):
=====
act_prof_add_member_to_group      shell
act_prof_create_group            show_actions
act_prof_create_member           show_ports
act_prof_delete_group            show_tables
act_prof_delete_member           swap_configs
act_prof_dump                   switch_info
act_prof_dump_group             table_add
act_prof_dump_member            table_clear
act_prof_modify_member          table_delete
act_prof_remove_member_from_group table_dump
counter_read                    table_dump_entry
counter_reset                   table_dump_entry_from_key
counter_write                   table_dump_group
help                            table_dump_member
load_new_config_file            table_indirect_add
```

CLI是使用python的cmd模块实现的，支持自动补全。它支持查看table，查看action，增删流表等操作，也可以通过'help'查看支持的命令，部分命令示例如下：

```
- table_set_default <table name> <action name> <action parameters>
- table_add <table name> <action name> <match fields> => <action parameters> [priority]
- table_delete <table name> <entry handle>
```

在CLI中下发并查看流表，更多操作的使用方式可以输入指令直接查询：

```
sdn@ubuntu: ~/p4/behavioral-model/tools
port_add                                table_info
port_remove                             table_modify
register_read                            table_num_entries
register_reset                           table_reset_default
register_write                           table_set_default
reset_state                             table_set_timeout
serialize_state                          table_show_actions
set_crc16_parameters                     write_config_to_file
set_crc32_parameters

Undocumented commands:
=====
EOF greet

RuntimeCmd: table_set_default ipv4_lpm _drop
Setting default action of ipv4_lpm
action: _drop
runtime data:
RuntimeCmd: table_add ipv4_lpm set_nhop 10.0.0.10/32 => 10.0.0.10 1
Adding entry to lpm match table ipv4_lpm
match key: LPM-0a:00:00:0a/32
action: set_nhop
runtime data: 0a:00:00:0a 00:01
Entry has been added with handle 0
RuntimeCmd: table_add ipv4_lpm set_nhop 10.0.1.10/32 => 10.0.1.10 2
Adding entry to lpm match table ipv4_lpm
match key: LPM-0a:00:01:0a/32
action: set_nhop
runtime data: 0a:00:01:0a 00:02
Entry has been added with handle 1
RuntimeCmd: table_dump ipv4_lpm
=====
TABLE ENTRIES
*****
Dumping entry 0x0
Match key:
* ipv4.dstAddr : LPM 0a00000a/32
Action entry: set_nhop - 0a00000a, 01
*****
Dumping entry 0x1
Match key:
* ipv4.dstAddr : LPM 0a00010a/32
Action entry: set_nhop - 0a00010a, 02
=====
Dumping default entry
Action entry: _drop -
=====
RuntimeCmd: █
```

也可以将指令写入txt文件，打开CLI同时下发流表，可以打开文件 `vim /home/sdn/p4/behavioral-model/targets/simple_router/command.txt` 查看具体的流表项：

```
cd targets/simple_router
./runtime_CLI.py --thrift-port 9090 < ../targets/simple_router/commands.txt
```

```

sdn@ubuntu:~/p4/behavioral-model/tools$ ./runtime_CLI.py --thrift-port 9090 < ../
targets/simple_router/commands.txt
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: Setting default action of send_frame
action:          _drop
runtime data:
RuntimeCmd: Setting default action of forward
action:          _drop
runtime data:
RuntimeCmd: Setting default action of ipv4_lpm
action:          _drop
runtime data:
RuntimeCmd: Adding entry to exact match table send_frame
match key:      EXACT-00:01
action:         rewrite_mac
runtime data:   00:aa:bb:00:00:00
Entry has been added with handle 0
RuntimeCmd: Adding entry to exact match table send_frame
match key:      EXACT-00:02
action:         rewrite_mac
runtime data:   00:aa:bb:00:00:01
Entry has been added with handle 1
RuntimeCmd: Adding entry to exact match table forward
match key:      EXACT-0a:00:00:0a
action:         set_dmac
runtime data:   00:04:00:00:00:00
Entry has been added with handle 0
RuntimeCmd: Adding entry to exact match table forward
match key:      EXACT-0a:00:01:0a
action:         set_dmac
runtime data:   00:04:00:00:00:01
Entry has been added with handle 1
RuntimeCmd: Adding entry to lpm match table ipv4_lpm
match key:      LPM-0a:00:00:0a/32
action:         set_nhop
runtime data:   0a:00:00:0a      00:01
Invalid table operation (DUPLICATE_ENTRY)
RuntimeCmd: Adding entry to lpm match table ipv4_lpm
match key:      LPM-0a:00:01:0a/32
action:         set_nhop
runtime data:   0a:00:01:0a      00:02
Invalid table operation (DUPLICATE_ENTRY)
RuntimeCmd:
sdn@ubuntu:~/p4/behavioral-model/tools$

```

此时流表已经填充。可以在Mininet中运行Pingall或在主机h1和h2之间使用iperf发送TCP流来检查网络的连通性:

```
sdn@ubuntu: ~/p4/behavioral-model/mininet
h3
default interface: eth0 10.0.2.10      00:04:00:00:00:02
*****
*****
h4
default interface: eth0 10.0.3.10      00:04:00:00:00:03
*****
Ready !
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X X X
h3 -> X X X
h4 -> X X X
*** Results: 100% dropped (0/12 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X
h2 -> h1 X X
h3 -> X X X
h4 -> X X X
*** Results: 83% dropped (2/12 received)
mininet> 
```

可以看到下发流表后h1，h2之连通，其他主机之间不通，具体原因如果想要其他主机之间也连通，还需要编写流表并下发。

2.6 p4c

p4c是用于编译p4语言的编译器，它同时支持P4-14和P4-16。默认P4语言版本为p4-16，如果需要编译p4-14，需要通过--std p4-14指定。可以通过一下命令编译p4程序：

```
p4c-bm2-ss xx.p4 -o xx.json
```

或

```
p4c xx.p4 -o xxx
```

通过第二种方式编译，如果成功编译，会生成一个文件夹，包含以下几个文件：后缀为.p4i的文件，它是在P4程序上运行预处理器时的输出。后缀为.p4rt的文件，它是一种二进制格式，包含P4程序中具有自动生成的控制平面API的表和其他对象的描述。后缀为.json的文件，是bm2交换机模型simple_switch所期望的json文件格式文件。可以通过p4c --help或p4c --target-help查看p4c支持的其他命令：

```
p4c --help
p4c --target-help
```



```
sdn@ubuntu: ~/p4/p4_program/simple_router-1.json
sdn@ubuntu:~/p4/p4_program$ ls
simple_router.p4
sdn@ubuntu:~/p4/p4_program$ sudo p4c-bm2-ss simple_router.p4 --std p4-14 --toJSON simple_router.json
sdn@ubuntu:~/p4/p4_program$ ls
simple_router.json  simple_router.p4
sdn@ubuntu:~/p4/p4_program$ sudo p4c simple_router.p4 --std p4-14 -o simple_router-1.json
sdn@ubuntu:~/p4/p4_program$ ls
simple_router-1.json  simple_router.json  simple_router.p4
sdn@ubuntu:~/p4/p4_program$ cd simple_router-1.json/
sdn@ubuntu:~/p4/p4_program/simple_router-1.json$ ls
simple_router.json  simple_router.p4i
sdn@ubuntu:~/p4/p4_program/simple_router-1.json$
```

2.7 P4Runtime

3 实验内容

第一次实验主要为验证性实验，要求各位在学习第2部分内容的基础上完成下面两道题目

3.1 实验1

题目

- 参考2.1的示例，实现mytunnel路由，填充部分代码，要求实现以下功能：
 - 添加了一个名为mytunnel_t的新头部字段类型，其中包含两个16位字段：proto_id和dst_id。
 - mytunnel头部需要添加到headers结构中。
 - 更新解析器，根据Ethernet头部中的ethertype字段解析mytunnel头部或ipv4头部。与mytunnel头段对应的ethertype是0x1212。如果mytunnel的proto_id==type_ipv4（即0x0800），解析器还应在mytunnel头部之后解析ipv4头部。
 - 定义一个名为mytunnel_forward的新的action，该动作只需将出口端口（即standard_metadata的egress_spec字段）设置为控制平面提供的端口号。
 - 定义一个名为mytunnel_exact的新的table，该表根据mytunnel头部的dst_id字段执行精确匹配（exact）。如果表中存在匹配项，则此表应调用myTunnel_forward转发动作，否则应调用Drop动作。
 - 更新myIngress控制模块中的apply语句，使得在myTunnel头部有效的情况下应用新定义的mytunnel_exact表。否则，如果ipv4头部有效，则调用ipv4_lpm表。
 - 更新deparser，由于头部的有效性是由隐集解析器提取，因此，这里不需要检查头部有效性。

说明

- 验收（二选一）

- 当场验收：请指导老师查看运行结果和代码即可
- 提交报告：报告中需包括姓名学号、代码、拓扑截图及总结，命名格式 `cs60-小胖-exp11.pdf`

3.2 实验2

题目

编译实验1实现的p4程序，仿照/home/sdn/p4/behavioral-model/minnet/1sw.py，重新实现一个拓扑，交换机数量不小于3，两个主机之间最好有多条路径，在mininet中查看拓扑。并且运行拓扑，交换机执行实验1生成的json代码。并且手动向各个交换机下发流表，使主机之间连通。在两台主机之间发送数据包，并用wireshark抓包。

说明

- 不同交换机设备需要绑定不同的端口，可以在程序中设置默认值或用过参数指定，使得在相应端口运行thrift server可以控制相应的交换机
- 根据下发的流表，可以分别根据mytunnel或ipv4转发，需要构造数据包（包含mytunnel，不含mytunnel。可用Python scapy构造）在主机之间发包。
- 验收（二选一）
 - 当场验收：请指导老师查看运行结果和代码
 - 提交报告：报告中需包括姓名学号、代码、wireshark抓包截图及总结，命名格式 `cs60-小胖-exp12.pdf`

4 总结

希望本次实验大家能掌握以下知识点：

- 能够编写简单的P4程序
- 利用Mininet的API自定义P4网络拓扑
- 学会使用P4相关工具
- 二层自学习交换机的基本原理

5 扩展资料

- sdn论坛：[sdnlab](#)
- 关于Mininet的更多资料：[Mininet Doc](#)，[Mininet API](#)
- 关于P4的更多资料：[p4 language](#)
- SDN网络的部署案例：[Google P4](#)