p4 第一次实践作业

学号：181700319 姓名：林鑫祥

官方的 P4 实验

basic 实验和 basic_tunnel 实验

basic 实验

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>

const bit<16> TYPE_IPV4 = 0x800;

/*************************************************************************
*********************** H E A D E R S  ***********************************
*************************************************************************/

typedef bit<9>  egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;

header ethernet_t {                 /* 定义帧的头部 需要的类型和数据 */
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}

header ipv4_t {                    /* 定义 ipv4 协议里的各种变量类型   */
    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 metadata {       /* 元数据    */
    /* empty */
}

struct headers {                  /*头部  */
    ethernet_t   ethernet;
    ipv4_t       ipv4;
}

/*************************************************************************
********************** P A R S E R  ***********************************
*************************************************************************/

parser MyParser(packet_in packet,                                          /* # 这
里 parser 是解析器分析报文，parser 有三种状态，分别是 start,accept 和 reject */
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {                                                         /*  # 直
接使用 transition 状态字，提取对应的 hdr.ethernet,有两种写法，这里采用最简单的，bsic_tunnel
用到 select*/
        /* TODO: add parser logic */
        packet.extract(hdr.ethernet);
        transition accept;
    }
}


/*************************************************************************
************   C H E C K S U M    V E R I F I C A T I O N  *************
*************************************************************************/

control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
    apply {  }
}
```

```
/*************************************************************************
*************  I N G R E S S   P R O C E S S I N G   *******************
*************************************************************************/

control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
    action drop() {
        mark_to_drop(standard_metadata);
    }

    action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {            /* # 传入
dstaddr 更新目标 MAC 地址 和 port 端口，同时跳数减一，要记得加上 更新一下原来的 src 地址（易
忽略）*/
        /* TODO: fill out code in action body */
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = dstAddr;                                    /* # 需要在最
前面更新 ethernet 的目的地址 */
        standard_metadata.egressSpec_t = port;
        hdr.ipv4.ttl = hdr.ipv4.ttl
-1;                                                                              .ttl

    }

    table ipv4_lpm {                        /*  定义一个 ipv4 最长前缀匹配规
则           */
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        actions = {
            ipv4_forward;
            drop;
            NoAction;
        }
        size = 1024;
        default_action = NoAction();
    }

    apply {
        if (hdr.ipv4.isValid()) {                                                /*
验证头部的 ipv4 是否有效 */
            ipv4_lpm.apply();
        }
        /* TODO: fix ingress control logic
```

```
         *  - ipv4_lpm should be applied only when IPv4 header is valid
         */

    }
}

/*************************************************************************
***************  E G R E S S   P R O C E S S I N G   *******************
*************************************************************************/

control MyEgress(inout headers hdr,                                    /* 缓冲区，
用于转发 */
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    apply {  }
}

/*************************************************************************
*************   C H E C K S U M   C O M P U T A T I O N   **************
*************************************************************************/

control MyComputeChecksum(inout headers hdr, inout metadata meta) {    /* 检验校验和 */
     apply {
        update_checksum(
            hdr.ipv4.isValid(),
            { hdr.ipv4.version,
              hdr.ipv4.ihl,
              hdr.ipv4.diffserv,
              hdr.ipv4.totalLen,
              hdr.ipv4.identification,
              hdr.ipv4.flags,
              hdr.ipv4.fragOffset,
              hdr.ipv4.ttl,
              hdr.ipv4.protocol,
              hdr.ipv4.srcAddr,
              hdr.ipv4.dstAddr },
            hdr.ipv4.hdrChecksum,
            HashAlgorithm.csum16);
    }
}


/*************************************************************************
***********************  D E P A R S E R  *******************************
```

```
*************************************************************************/

control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        /* TODO: add deparser logic */
        packet.emit(hdr.ethernet);                                          /*  #
取出头部的 ethernet 和 ipv4 协议头部 */
        packet.emit(hdr.ipv4);
    }
}

/*************************************************************************
***********************  S W I T C H  *******************************
*************************************************************************/

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

basic_tunnel 文件

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>

// NOTE: new type added here

const bit<16> TYPE_MYTUNNEL = 0x1212;                /* 定义 my_tunnel    */
const bit<16> TYPE_IPV4 = 0x800;

/*************************************************************************
*********************** H E A D E R S  ***********************************
*************************************************************************/

typedef bit<9>  egressSpec_t;             /* 自定义 egress 端口，mac 地址，ipv4 地址类型
*/
typedef bit<48> macAddr_t;
```

```
typedef bit<32> ip4Addr_t;

header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}

// NOTE: added new header type
header myTunnel_t {                         /* 自定义 my_Tunnel 里的协议和 目标 id  */
    bit<16> proto_id;
    bit<16> dst_id;
}

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 metadata {
    /* empty */
}

// NOTE: Added new header type to headers struct
struct headers {                                /*   头部数据组成    */
    ethernet_t   ethernet;
    myTunnel_t   myTunnel;
    ipv4_t       ipv4;
}

/*************************************************************************
*********************** P A R S E R  ***********************************
*************************************************************************/
```

```
// TODO: Update the parser to parse the myTunnel header as well
parser MyParser(packet_in packet,                    /* 根据状态选择 select,取出头
部      */
                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;
    }


}

/*************************************************************************
************   C H E C K S U M    V E R I F I C A T I O N   *************
*************************************************************************/

control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
    apply {  }
}


/*************************************************************************
**************  I N G R E S S   P R O C E S S I N G   *******************
*************************************************************************/

control MyIngress(inout headers hdr,             /* ingress 解析  */
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
    action drop() {
```

```
        mark_to_drop(standard_metadata);
    }

    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 {                            /*     最长前缀匹配              */
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        actions = {
            ipv4_forward;
            drop;
            NoAction;
        }
        size = 1024;
        default_action = drop();
    }

    // TODO: declare a new action: myTunnel_forward(egressSpec_t port)
    action myTunnel_forward(egressSpec_t port)
{                                           /* 这里已经给出端口，给出端口输出端口的端口号,
egress  */
        standard_metadata.egress_spec = port
    }
    table
myTunnel_exact{                                                                 /*
exact 对应 ternary 匹配，对于每一个 ternary 匹配，匹配表每一个表项都有掩码 */

        key = {
            hdr.dst_id:exact;
        }
        actions = {
            ipv4_forward;
            drop;
            NoAction;
        }
        size = 1024;
        default_action = drop();
    }
```

```
    // TODO: declare a new table: myTunnel_exact
    // TODO: also remember to add table entries!


    apply {
        // TODO: Update control flow
        if (hdr.ipv4.isValid()) {                                          /*
判断是否匹配路由器自身的 MAC 地址， if else 选择 apply */
            ipv4_lpm.apply();
        }else{
            if (hdr.myTunnel.isValid()){
                myTunnel_exact.apply();
            }
        }


    }
}

/*************************************************************************
***************  E G R E S S   P R O C E S S I N G   ******************
*************************************************************************/

control MyEgress(inout headers hdr,             /* 缓冲区             */
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    apply {  }
}

/*************************************************************************
*************   C H E C K S U M   C O M P U T A T I O N   **************
*************************************************************************/

control MyComputeChecksum(inout headers  hdr, inout metadata meta) {
     apply {
        update_checksum(
            hdr.ipv4.isValid(),
            { hdr.ipv4.version,
              hdr.ipv4.ihl,
              hdr.ipv4.diffserv,
              hdr.ipv4.totalLen,
              hdr.ipv4.identification,
              hdr.ipv4.flags,
              hdr.ipv4.fragOffset,
```

```
              hdr.ipv4.ttl,
              hdr.ipv4.protocol,
              hdr.ipv4.srcAddr,
              hdr.ipv4.dstAddr },
            hdr.ipv4.hdrChecksum,
            HashAlgorithm.csum16);
    }
}

/*************************************************************************
***********************  D E P A R S E R  ***************************
*************************************************************************/

control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        // TODO: emit myTunnel header as well
        packet.emit(hdr.myTunnel)                                        /*
简单的 emit 操作 */
        packet.emit(hdr.ipv4);
    }
}

/*************************************************************************
***********************  S W I T C H  *******************************
*************************************************************************/

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

send.py

```
#!/usr/bin/env python3
import random
import socket
import sys
```

```
from scapy.all import IP, TCP, Ether, get_if_hwaddr, get_if_list, sendp


def get_if():
    ifs=get_if_list()             # 提供一系列接口的名字
    iface=None # "h1-eth0"        # 初始化为空
    for i in get_if_list():
        if "eth0" in i:           # 找到接口名字为 'eth0'的接口名
            iface=i
            break;
    if not iface:                 #找不到返回 'eth0'
        print("Cannot find eth0 interface")
        exit(1)
    return iface                  # 返回一个 iface

def main():

    if len(sys.argv)<3:           # 如果用户传入的参数低于两个，输出 目的地址 和消息
        print('pass 2 arguments: <destination> "<message>"')
        exit(1)

            # gethostbyname 返回的是主机名 的 IPv4 的地址格式，如果传入的参数是 IPv4 的地址格式，则返回值跟参数一样，不支持 IPv6 的域名解析
    addr = socket.gethostbyname(sys.argv[1])          # 从一个域名返回一个 IP 地址
    iface = get_if()                                  #获取一个接口名

    print("sending on interface %s to %s" % (iface, str(addr)))    # 显示发送消息的端口和接受端口
    pkt =  Ether(src=get_if_hwaddr(iface), dst='ff:ff:ff:ff:ff:ff') # 将源地址返回为为 mac 地址，Ether 采用分层形式构造数据包
    pkt = pkt /IP(dst=addr) / TCP(dport=1234, sport=random.randint(49152,65535)) / sys.argv[2]  # 自底向上构造成 IP 数据包，再有 IP 数据包转成 Tcp 数据包
    pkt.show2()                       # 展示数据      sport 是允许数据进入的端口，dport 是允许数据发出的端口，并在两个数之间随机生成
    sendp(pkt, iface=iface, verbose=False)                          # 发送 ether 数据包，工作在第二层
                                                                    # send 是发送 IP 数据包 ，verbose 是有无信息显示（keras 中是有无日志显示）


if __name__ == '__main__':
    main()
```

receive.py

```
#!/usr/bin/env python3
import os
import sys

from scapy.all import (
    TCP,
    FieldLenField,
    FieldListField,
    IntField,
    IPOption,
    ShortField,
    get_if_list,
    sniff
)
from scapy.layers.inet import _IPOption_HDR


def get_if():
    ifs=get_if_list()                                   # 从接口名里去寻找 etho 接口名，进行匹配，函数与 send.py 类似相同
    iface=None
    for i in get_if_list():
        if "eth0" in i:
            iface=i
            break;
    if not iface:
        print("Cannot find eth0 interface")             # 探测接口
        exit(1)
    return iface

class IPOption_MRI(IPOption):                           # ip 可选择类，
    name = "MRI"
    option = 31                                         # IP 的选项和填充设置
    fields_desc = [ _IPOption_HDR,                      # 设置目的 field，field 是数据包中的成员
                    FieldLenField("length", None, fmt="B",  # 名字 ，fmt B 强制使用一字节无符号， H 是 struct 里的格式字符
                                  length_of="swids",            # 可选字段字符串
                                  adjust=lambda pkt,l:l+4),     # lambda 表达式生成 pkt 函数，adjust 可调整，类型为可调用和数据包组成
                    ShortField("count", 0),                     # 名字，可选字段
```

```
                    FieldListField("swids",                         # 可选字段
                                   [],
                                   IntField("", 0),                 # 可选字段设置为0
                                   length_from=lambda pkt:pkt.count*4) ]          #一种
呼叫管理封包，由被呼叫的资料终端设备发出，表示它接受发来的呼叫
def handle_pkt(pkt):
    if TCP in pkt and pkt[TCP].dport == 1234:                        # 如果满足条件，且
报文 tcp 的 发送端口是 1234
        print("got a packet")
        pkt.show2()                                                 # 数据包展示
    #    hexdump(pkt)
        sys.stdout.flush()                                          # sys 以 flush 刷
存的方式将报文显示出来


def main():
    ifaces = [i for i in os.listdir('/sys/class/net/') if 'eth' in i]  # 列表表达式，
列出在对于路径下的'eth'字段的接口名
    iface = ifaces[0]                                               # 列表中第一个接
口名
    print("sniffing on %s" % iface)                                  # 探测接口
    sys.stdout.flush()                                              #流的形式刷出
    sniff(iface = iface,                                            # 探测对应的接口，
分析对应的数据，并返回对应的数据包并显示
          prn = lambda x: handle_pkt(x))

if __name__ == '__main__':
    main()
```

问答

问答题

如果将 basic 和 basic_tunnel 项目移除 tutorials/exercises 目录，不能继续运行。若要继续运行，需要修改 makefile 文件，使得项目运行的时候能够找到正确的路径。根据实际把运行文件处于的位置修改 TOPO 和 include 两行即可

send.py 的方式是通过传入一个 IP 地址，接着获取一个接口名，这里是' eth0' 。然后 pkt 构造 mac 帧数据，附加上目标 mac 地址，接着 ether 分层构造数据包，变作 IP 数据报·再变为 TCP 数据报文，并随机指定一个发送端口，发送信息。 receive.py 首先查看 对应目录下接口名为 ‘eth0 ’的接口，取第一个，使用 scapy 包中的 sniff 探测 1234 端口的报文。

传统 IPV4 转发流程。

（1）从数据报的首部提取目的主机的 IP 地址 D，得出目的网络地址 N。

（2） 若网络 N 与此路由器直接相连，则把数据报直接交付给目的主机 D，否则执行 3

（3）. 若路由器中有目的地址 D 的特定主机路由，则把数据报传送给路由表指明的下一条路由器，否则执行 4

（4）. 若路由表有到底 N 的路由，则把数据包传送给路由表指明的下一条路由，否则执行 5

（5） 若路由表有一个默认路由，则把数据报传送给路由表指明的默认路由，否则报告出错

网关把互联网划分成为较小的自治系统。有内部网关协议和外部网关协议。

在内部转发时，RIP 是只和本节点相邻的路由器交换， OSPF 是和网络中所有的路由器交换节点。

当转发时源和目的站处于不同的自治系统中，数据报传到一个自治系统的边界时候，就要用到外部网关协议 EGP,一般是 BGP。 从每一个自治系统选择路由器作为代言人，之间建立 TCP 连接交换信息，找到各个自治系统的较好路由。首次是和本节点相邻的路由器交换节点，后来就和有变化的部分交换节点信息即可。

Basic 实验当中数据转发是用户定义，由交换机实现，同时建立合适的拓扑连接，需要定义好 mac, IP 地址，ARP,host 以及以及端口以及各个连接,。parser 输入数据，取出首部，control 模块控制 ingress 解析器解析，定义各种 table,action,egress 执行转发，deparser 发出头部 header.

提高题

以下为对应的代码

```
/* -*- P4_16 -*- */

#include <core.p4>
#include <v1model.p4>

/*************************************************************************
 ***********************  C O N S T A N T S  *****************************
 *************************************************************************/
```

```
      /*  Define the useful global constants for your program */
const bit<16> ETHERTYPE_IPV4 = 0x0800;
const bit<16> ETHERTYPE_ARP  = 0x0806;
const bit<8>  IPPROTO_ICMP   = 0x01;

/*************************************************************************
 **********************  H E A D E R S  ********************************
 *************************************************************************/
       /*  Define the headers the program will recognize */

/*
 * Standard ethernet header
 */
typedef bit<48>  mac_addr_t;
typedef bit<32>  ipv4_addr_t;
typedef bit<9>   port_id_t;

header ethernet_t {
    mac_addr_t dstAddr;
    mac_addr_t srcAddr;
    bit<16>    etherType;
}

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;
    ipv4_addr_t  srcAddr;
    ipv4_addr_t  dstAddr;
}

const bit<16> ARP_HTYPE_ETHERNET = 0x0001;
const bit<16> ARP_PTYPE_IPV4     = 0x0800;
const bit<8>  ARP_HLEN_ETHERNET  = 6;
const bit<8>  ARP_PLEN_IPV4      = 4;
const bit<16> ARP_OPER_REQUEST   = 1;
const bit<16> ARP_OPER_REPLY     = 2;
```

```
header arp_t {
    bit<16> htype;
    bit<16> ptype;
    bit<8>  hlen;
    bit<8>  plen;
    bit<16> oper;
}

header arp_ipv4_t {
    mac_addr_t  sha;
    ipv4_addr_t spa;
    mac_addr_t  tha;
    ipv4_addr_t tpa;
}

const bit<8> ICMP_ECHO_REQUEST = 8;
const bit<8> ICMP_ECHO_REPLY   = 0;

header icmp_t {
    bit<8>  type;
    bit<8>  code;
    bit<16> checksum;
}

/* Assemble headers in a single struct */
struct my_headers_t {
    ethernet_t   ethernet;
    arp_t        arp;
    arp_ipv4_t   arp_ipv4;
    ipv4_t       ipv4;
    icmp_t       icmp;
}

/*************************************************************************
 ***********************  M E T A D A T A  ***************************
 *************************************************************************/
       /*  Define the global metadata for your program */

struct my_metadata_t {
    ipv4_addr_t dst_ipv4;
    mac_addr_t  mac_da;
    mac_addr_t  mac_sa;
    port_id_t   egress_port;
```

```
    mac_addr_t  my_mac;
}
/*************************************************************************
********************** P A R S E R  ***********************************
*************************************************************************/

parser MyParser(                                                        /*
parser 进行解析                    */
    packet_in             packet,
    out   my_headers_t    hdr,
    inout my_metadata_t   meta,
    inout standard_metadata_t standard_metadata)
{
    state start {                                                       /* 取
出 hdr 的 ethernet 帧  */
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            ETHERTYPE_IPV4 : parse_ipv4;
            ETHERTYPE_ARP  : parse_arp;
            default        : accept;
        }
    }

    state parse_arp {
        packet.extract(hdr.arp);                                        /*
取出对应的 arp */
        transition select(hdr.arp.htype, hdr.arp.ptype,
                          hdr.arp.hlen,  hdr.arp.plen) {
            (ARP_HTYPE_ETHERNET, ARP_PTYPE_IPV4,
             ARP_HLEN_ETHERNET,  ARP_PLEN_IPV4) : parse_arp_ipv4;
            default : accept;
        }
    }

    state parse_arp_ipv4 {                                              /* 取
出 hdr 的 ARP_IPV4  */
        packet.extract(hdr.arp_ipv4);
        meta.dst_ipv4 = hdr.arp_ipv4.tpa;
        transition accept;
    }

    state parse_ipv4 {                                                  /* 取
出 hdr 的 ipv4   */
        packet.extract(hdr.ipv4);
```

```
        meta.dst_ipv4 = hdr.ipv4.dstAddr;
        transition select(hdr.ipv4.protocol) {
            IPPROTO_ICMP : parse_icmp;
            default      : accept;
        }
    }

    state parse_icmp {                                                  /* 取
出·hdr 的 icmp  */
        packet.extract(hdr.icmp);
        transition accept;
    }
}

/*************************************************************************
************  C H E C K S U M   V E R I F I C A T I O N  *************
*************************************************************************/

control MyVerifyChecksum( in my_headers_t  hdr,inout my_metadata_t  meta){
    apply {     }
}


/*************************************************************************
**************  I N G R E S S  P R O C E S S I N G  *******************
*************************************************************************/

control MyIngress(
    inout my_headers_t     hdr,
    inout my_metadata_t    meta,
    inout standard_metadata_t  standard_metadata)
{
    action drop() {
        mark_to_drop();
        exit;
    }

    action set_dst_info(mac_addr_t mac_da,                              /*
设定 mata 元数据里的各个信息 */
                        mac_addr_t mac_sa,
                        port_id_t  egress_port)
    {
        meta.mac_da      = mac_da;
        meta.mac_sa      = mac_sa;
```

```
        meta.egress_port = egress_port;
    }

    table ipv4_lpm {                                                    /*
定义一个 ipv4 最长匹配的 table */
        key     = { meta.dst_ipv4 : lpm; }
        actions = { set_dst_info; drop;  }
        default_action = drop();
    }

    action forward_ipv4() {                                             /*
更新目标和源的 MAC 地址       */
        hdr.ethernet.dstAddr = meta.mac_da;
        hdr.ethernet.srcAddr = meta.mac_sa;
        hdr.ipv4.ttl         = hdr.ipv4.ttl - 1;

        standard_metadata.egress_spec = meta.egress_port;
    }

    action send_arp_reply() {                                           /*
定义一个 action reply 操作，对送入的 arp 报文解析后，arp 设置好进和出的端口地址 */
        hdr.ethernet.dstAddr = hdr.arp_ipv4.sha;
        hdr.ethernet.srcAddr = meta.mac_da;

        hdr.arp.oper         = ARP_OPER_REPLY;

        hdr.arp_ipv4.tha     =
hdr.arp_ipv4.sha;                                   /* 转发目的和源地址进行更
新           */
        hdr.arp_ipv4.tpa     = hdr.arp_ipv4.spa;
        hdr.arp_ipv4.sha     = meta.mac_da;
        hdr.arp_ipv4.spa     = meta.dst_ipv4;

        standard_metadata.egress_spec = standard_metadata.ingress_port;
    }

    action send_icmp_reply() {                                          /*
定义一个 action reply 操作，icmp 差错报文回复，交换目的和源端口地址  */
        mac_addr_t   tmp_mac;
        ipv4_addr_t  tmp_ip;

        tmp_mac              = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
        hdr.ethernet.srcAddr = tmp_mac;
```

```
        tmp_ip               = hdr.ipv4.dstAddr;                                    /*
交换目的和源端口 ipv4 地址            */
        hdr.ipv4.dstAddr     = hdr.ipv4.srcAddr;
        hdr.ipv4.srcAddr     = tmp_ip;

        hdr.icmp.type        =
ICMP_ECHO_REPLY;                                    /* 定义差错类型          */
        hdr.icmp.checksum    = 0; // For now

        standard_metadata.egress_spec =
standard_metadata.ingress_port;             /* 缓冲端口变为解析端口  */
    }

    table forward {                                                             /*
foward 定义一个到下一阶段的工作，根据条件进行各种匹配  */
        key = {
            hdr.arp.isValid()      : exact;
            hdr.arp.oper           : ternary;
            hdr.arp_ipv4.isValid() : exact;
            hdr.ipv4.isValid()     : exact;
            hdr.icmp.isValid()     : exact;
            hdr.icmp.type          : ternary;
        }
        actions = {                                                             /*
atcion 动作                 */
            forward_ipv4;
            send_arp_reply;
            send_icmp_reply;
            drop;
        }
        const default_action = drop();
        const entries = {                                                       /*
定义一个 entry ， 先发送 arp_reply ,载同步执行 forward_ipv4 和 send_icmp_reply 操作    */
            ( true, ARP_OPER_REQUEST, true, false, false, _  ) :
                                                    send_arp_reply();
            ( false, _,               false, true, false, _  ) :
                                                    forward_ipv4();
            ( false, _,               false, true, true, ICMP_ECHO_REQUEST ) :
                                                    send_icmp_reply();
        }
    }

    apply {                                                                     /*
```

```
执行，设定 自己的 mac 地址           */
        meta.my_mac = 0x000102030405;
        ipv4_lpm.apply();
        forward.apply();
    }
}

/*************************************************************************
***************  E G R E S S   P R O C E S S I N G   ******************
*************************************************************************/

control MyEgress(inout my_headers_t hdr,
                 inout my_metadata_t   meta,
                 inout standard_metadata_t standard_metadata) {
    apply {  }
}

/*************************************************************************
*************   C H E C K S U M   C O M P U T A T I O N   **************
*************************************************************************/

control MyComputeChecksum(inout my_headers_t  hdr,inout my_metadata_t meta)
{          /*  校验和   */
     apply {
        update_checksum(
        hdr.ipv4.isValid(),
            { hdr.ipv4.version,
              hdr.ipv4.ihl,
              hdr.ipv4.diffserv,
              hdr.ipv4.totalLen,
              hdr.ipv4.identification,
              hdr.ipv4.flags,
              hdr.ipv4.fragOffset,
              hdr.ipv4.ttl,
              hdr.ipv4.protocol,
              hdr.ipv4.srcAddr,
              hdr.ipv4.dstAddr },
            hdr.ipv4.hdrChecksum,
            HashAlgorithm.csum16);
    }
}

/*************************************************************************
***********************  D E P A R S E R  *******************************
```

```
*************************************************************************/

control MyDeparser(packet_out packet, in my_headers_t hdr) {
    apply {
        packet.emit(hdr.ethernet);                                          /*
取出协议头部   */
        packet.emit(hdr.ipv4);
        packet.emit(hdr.arp);
        packet.emit(hdr.arp_ipv4);
        packet.emit(hdr.icmp);
    }
}

/*************************************************************************
**********************  S W I T C H  ****************************
*************************************************************************/

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

总结

本次实验主要是对应 p4 的第一次实验，运行了 官方的实验代码，代码的各个模块功能划分得很清楚。在本次过程中，首先我先查看了一下老师给出的文档，重新梳理了一下各个模块的流程，并回顾了一下协议格式。了解了 table,action 以及 apply 等关键字。提高题，在定义 arp 的时候，我完全没有考虑到 ICMP 和 arp_ipv4 两个头部。同时我查阅了知乎，知乎上有给出 p4 的学习笔记，对于各个模块也同样做出了说明，也让我能够更加清晰地了解代码以及对于的执行操作。在查看 send.py 和 receive.py 时，利用 Pycharm 我较方便地就可以查看对于的每一个模块的函数，对于一些不了解的有参考一些有关于 scapy 的博客，同时也参考了一些 scapy 教程 。确实，学习一个新的东西时比较花费时间的，特别是百度上关于 p4 的比较少，B 站上也几乎没有关于 P4 的教程，所以很多时候需要花费大量的时间才能找出真正对于你有用的信息。也希望以后能有关于 P4 的视频教程出来。