**开机前注意事项：**

将一根电源线连接交换机即可，电源线接入后机器直接启动，直接断电即可关机。

可以使用串口登录，波特率为115200；或者ssh登录（机器暂未配置ip）

向机器传输文件可用U盘（usb口在控制口和管理口旁）

机器使用系统基于Linux,命令行操作都是使用linux指令

账号:root

密码:onl

修改交换机IP：

#vi /etc/network/interfaces

写入以下内容（IP自定义）:

auto ma1

iface ma1 inet static

address 192.168.10.100

netmask 255.255.255.0

之后输入ifup ma1，使设置IP生效

ssh无法登录时：

#vi /etc/ssh/sshd.config

写入：PermitRootLogin yes

然后直接输入#/etc/init.d/ssh restart

机器降噪：

输入#find / -name pwm2

之后会显示一个路径：

输入#echo 60 > (路径)

降噪成功。

**编译程序**

编译运行程序前先加载驱动：

在sde目录下：

输入#$SDE\_INSTALL/bin/bf\_kdrv\_mod\_load $SDE\_INSTALL

再输入#ls /dev/bf0查看/dev/bf0驱动是否加载完成。

C:\Users\朝仓绫\AppData\Local\Temp\1626402716(1).png

**示例1:port程序**

使用ssh登录交换机。

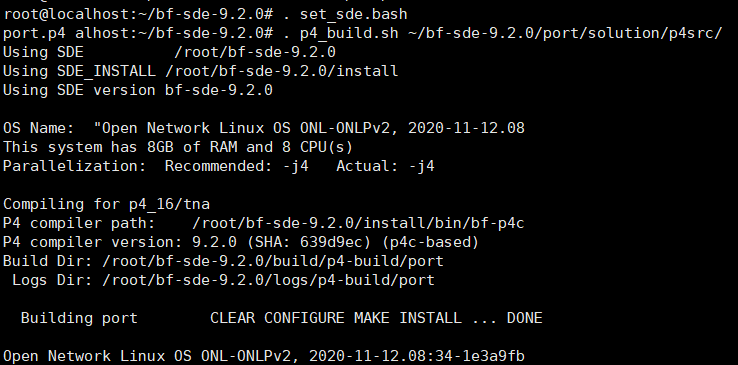
**1.编译运行程序：**

在sde目录下:

#source set\_sde.bash

#./p4\_build.sh ~/bf-sde-9.2.0/port/solution/p4src/port.p4 #编译程序

显示DONE则编译成功。



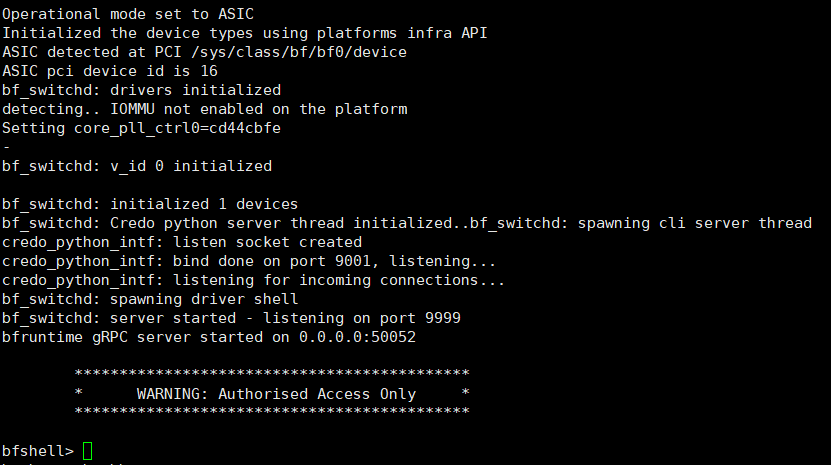
编译后掉线为正常现象。

新建一个窗口，在sde目录下：

#source set\_sde.bash

#./run\_switchd.sh –p port #运行程序（这里port指p4程序名，一般为.p4文件的名字）

进入bfshell则运行成功



**2.流表下发：**

方法一：手动下发

在bfshell下：

bfshell>bfrt\_python

bfrt\_python>bfrt

bfrt>port

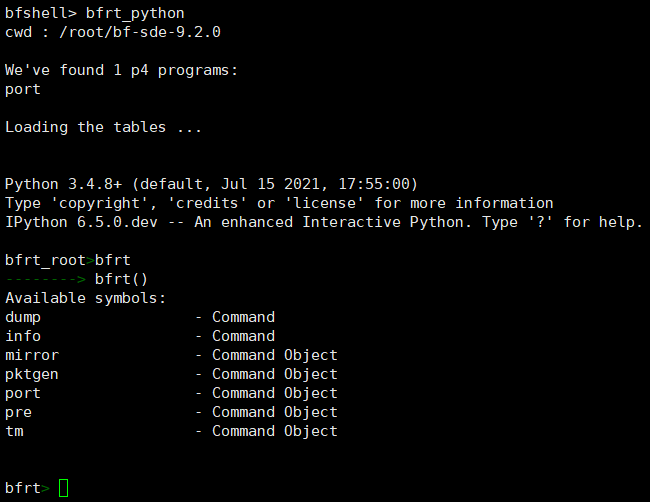
bfrt.port>pipe

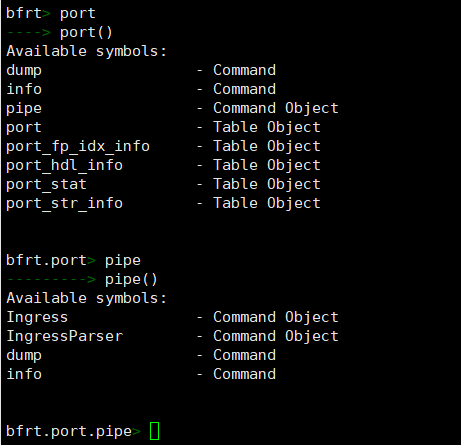
bfrt.port.pipe>Ingress

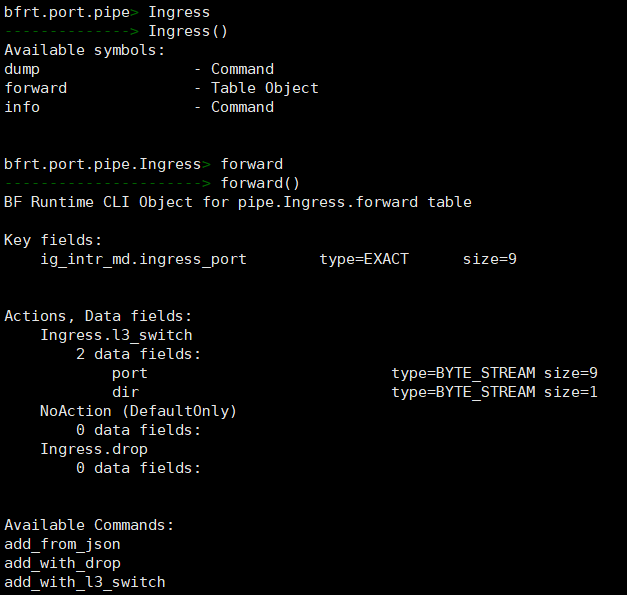
bfrt.port.pipe.Ingress>forward

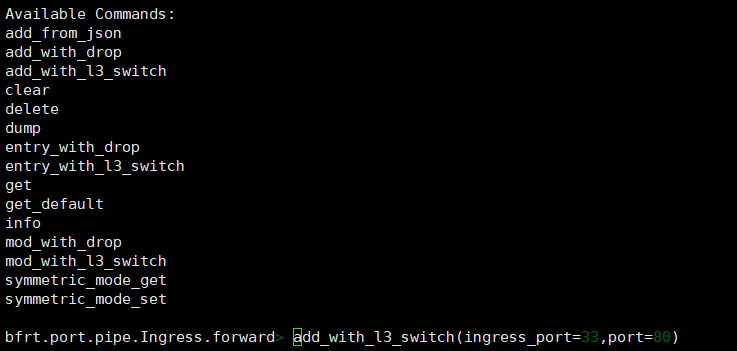
bfrt.port.pipe.Ingress.forward>add\_with\_l3\_switch(ingress\_port=33,port=80) #输入进出的逻辑端口号

其他流表操作也可以在bfrt内完成。









方法二：脚本下发（常用）：

在run\_switchd.sh运行后另外一个窗口：

在sde目录下：

#./run\_bfshell.sh -b ~/bf-sde-9.2.0/port/solution/bfrt\_python/setup.py –i

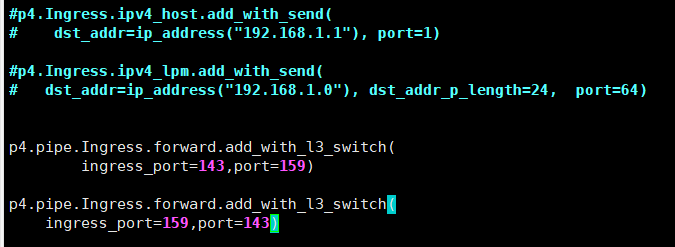
运行后可以看到下发的流表。

下发流表前，先在端口管理界面查看端口对应的逻辑端口号，再在该setup.py下修改：

在下面 **3.配置端口** 中端口7和11的逻辑端口号为143、159

#vi ~/bf-sde-9.2.0/port/solution/bfrt\_python/setup.py

修改对应逻辑端口号（写两行配置双向）：



**3.配置端口：**

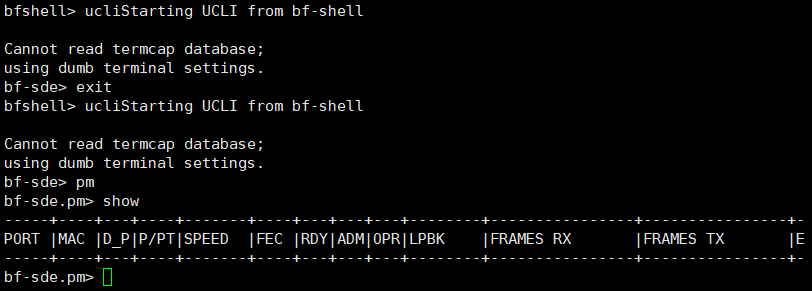
在bfshell输入ucli进入管理界面，再输入pm进入端口管理，可以输入port查看port功能介绍.

添加端口并设置速率：port-add 10/- 10G NONE #10表示端口号，10G表示速率，NONE为不限速

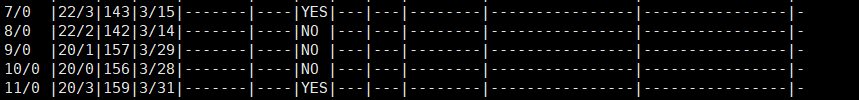
端口使能：port-enb 10/- #使能后该端口才能运作

输入show可以看到已经使能、可以转发包的端口。show –a能查看所有端口状态。

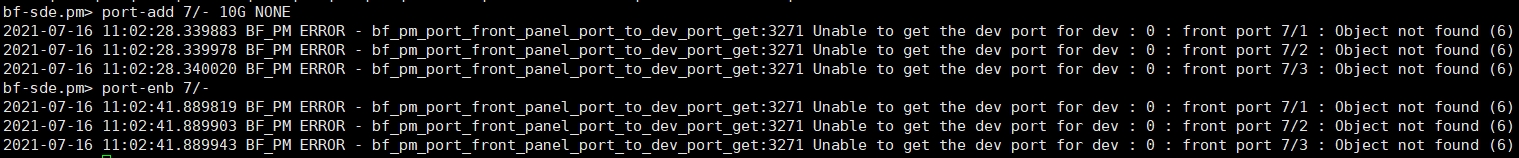
C:\Users\朝仓绫\AppData\Local\Temp\1626404115(1).png

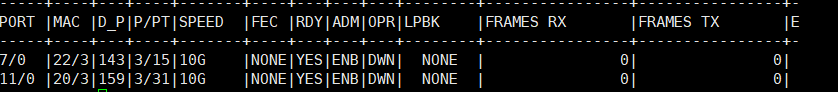


下面以在7和11端口插上模块为例：



可以看到模块已经接上，下面端口使能：





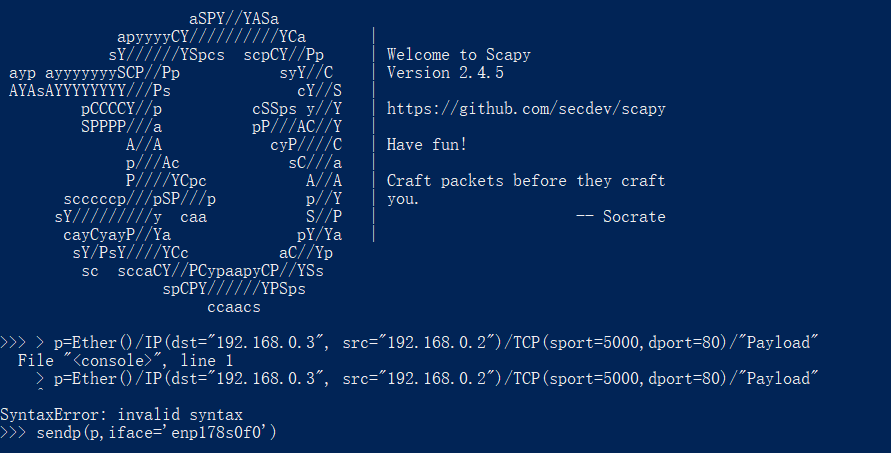
上图模块暂未与外部设备连接，故OPR一栏显示为DWN，连接后将显示UP。

D\_P是P4程序输入输出的端口，即逻辑端口。

FRAMES RX 、FRAMES TX是端口发送和接收数据包数的显示。

**4.利用python的scapy发包，用wireshark抓包**

#ps –aux | grep switch

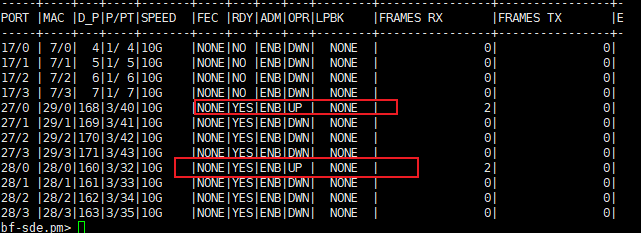


图中enp178s0f0是发包的网卡的名称。发包后，能在抓包软件上看到对应的包。

在linux系统下ifconfig可以看到网卡名称；

Windows下在cmd输入ipconfig /all查看，一般为“以太网”。

交互：



Up之后，在服务器1上进行发包

sudo scapy

p=Ether()/IP(dst="192.168.0.3", src="192.168.0.2")/TCP(sport=5000,dport=80)/"Payload"

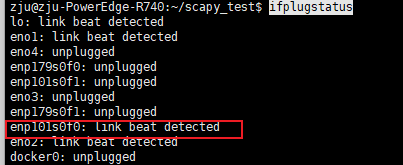
sendp(p,iface=’enp101s0f0’)

p=Ether()/IP(dst="192.168.123.3", src="192.168.123.2")/TCP(sport=5000,dport=80)/"Payload"

sendp(p,iface=’enp101s0f0’)

另一端采用wireshark监听该端口

如果监听不到包，检查ifplugstatus，看是否网卡检测到了，若是unplugged，则重新插拔一边，等等等等等，一般就起来了，起来的图如下图：



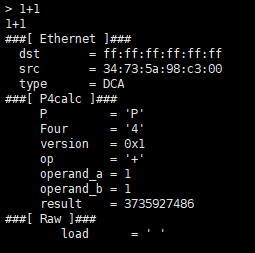
之后就可以抓到包了

二、协议尝试

此处参考example中“cal”例子，实现一个基本计算器。

因为TNA架构和v1model架构中一些元数据和设置方式不同，因此将计算器的协议内容增加至port文件中，经修改后的p4文件为：附录A中文件。使用bfshell启动端口，且发送数据包的文件为cal.py，设置传输端口为enp101s0f0，传送流程为：

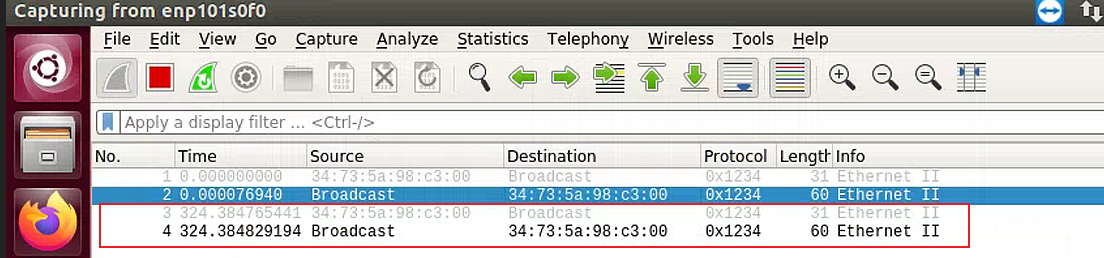
**发送端：**

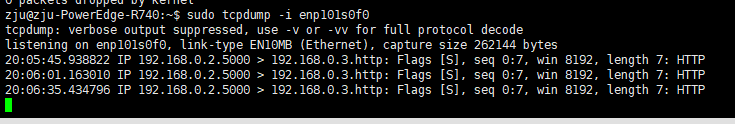


**接收端：**



**抓包：**



附录A：

|  |
| --- |
| /\* -\*- P4\_16 -\*- \*/  #include <core.p4>  #include <tna.p4>  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\* C O N S T A N T S A N D T Y P E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  const bit<16> ETHERTYPE\_TPID = 0x8100;  const bit<16> ETHERTYPE\_IPV4 = 0x0800;  const bit<16> ETHERTYPE\_IPV6 = 0x86DD;  typedef bit<16> tcpPort\_t;  typedef bit<32> ip4Addr\_t;  typedef bit<9> egressSpec\_t;  const bit<16> P4CALC\_ETYPE = 0x1234;  const bit<8> P4CALC\_P = 0x50; // 'P'  const bit<8> P4CALC\_4 = 0x34; // '4'  const bit<8> P4CALC\_VER = 0x01; // v0.1  const bit<8> P4CALC\_PLUS = 0x2b; // '+'  const bit<8> P4CALC\_MINUS = 0x2d; // '-'  const bit<8> P4CALC\_AND = 0x26; // '&'  const bit<8> P4CALC\_OR = 0x7c; // '|'  const bit<8> P4CALC\_CARET = 0x5e; // '^'  const int ipv4\_mask\_length = 24;  const int ipv4\_length = 32;  const int sharing\_ratio = 256;  const int M = 4;  const int m = 2;  /\* Table Size \*/  const int IPV4\_HOST\_SIZE = 65536;  const int IPV4\_LPM\_SIZE = 12288;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* H E A D E R S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* Define all the headers the program will recognize \*/  /\* The actual sets of headers processed by each gress can differ \*/  /\* Standard ethernet header \*/  header ethernet\_t {  bit<48> dst\_addr;  bit<48> src\_addr;  bit<16> ether\_type;  }  header vlan\_tag\_h {  bit<3> pcp;  bit<1> cfi;  bit<12> vid;  bit<16> ether\_type;  }  header ipv4\_h {  bit<4> version;  bit<4> ihl;  bit<8> diffserv;  bit<16> total\_len;  bit<16> identification;  bit<3> flags;  bit<13> frag\_offset;  bit<8> ttl;  bit<8> protocol;  bit<16> hdr\_checksum;  bit<32> src\_addr;  bit<32> dst\_addr;  }  header p4calc\_t {  bit<8> p;  bit<8> four;  bit<8> ver;  bit<8> op;  bit<32> operand\_a;  bit<32> operand\_b;  bit<32> res;  bit<16> ether\_type;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\* I N G R E S S P R O C E S S I N G \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/        /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* H E A D E R S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  struct my\_ingress\_headers\_t {  ethernet\_t ethernet;  p4calc\_t p4calc;  vlan\_tag\_h vlan\_tag;  ipv4\_h ipv4;  }  /\*\*\*\*\*\* G L O B A L I N G R E S S M E T A D A T A \*\*\*\*\*\*\*\*\*/  struct my\_ingress\_metadata\_t {  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* P A R S E R \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  parser IngressParser(packet\_in pkt,  /\* User \*/  out my\_ingress\_headers\_t hdr,  out my\_ingress\_metadata\_t meta,  /\* Intrinsic \*/  out ingress\_intrinsic\_metadata\_t ig\_intr\_md)  {  /\* This is a mandatory state, required by Tofino Architecture \*/  state start {  pkt.extract(ig\_intr\_md);  pkt.advance(PORT\_METADATA\_SIZE);  transition parse\_ethernet;  }    state parse\_ethernet {  pkt.extract(hdr.ethernet);  transition select(hdr.ethernet.ether\_type) {  P4CALC\_ETYPE : check\_p4calc;  ETHERTYPE\_TPID : parse\_vlan\_tag;  ETHERTYPE\_IPV4 : parse\_ipv4;  default : accept;  }  }  state check\_p4calc {  transition select(pkt.lookahead<p4calc\_t>().p,  pkt.lookahead<p4calc\_t>().four,  pkt.lookahead<p4calc\_t>().ver) {  (P4CALC\_P, P4CALC\_4, P4CALC\_VER) : parse\_p4calc;  default : accept;  }  }  state parse\_p4calc {  pkt.extract(hdr.p4calc);  transition select(hdr.p4calc.ether\_type) {  ETHERTYPE\_TPID : parse\_vlan\_tag;  ETHERTYPE\_IPV4 : parse\_ipv4;  default : accept;  }  }  state parse\_vlan\_tag {  pkt.extract(hdr.vlan\_tag);  transition select(hdr.vlan\_tag.ether\_type) {  ETHERTYPE\_IPV4 : parse\_ipv4;  default: accept;  }  }  state parse\_ipv4 {  pkt.extract(hdr.ipv4);  transition accept;  }    }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* M A T C H - A C T I O N \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  control Ingress(  /\* User \*/  inout my\_ingress\_headers\_t hdr,  inout my\_ingress\_metadata\_t meta,  /\* Intrinsic \*/  in ingress\_intrinsic\_metadata\_t ig\_intr\_md,  in ingress\_intrinsic\_metadata\_from\_parser\_t ig\_prsr\_md,  inout ingress\_intrinsic\_metadata\_for\_deparser\_t ig\_dprsr\_md,  inout ingress\_intrinsic\_metadata\_for\_tm\_t ig\_tm\_md)  {  action send\_back(bit<32> result) {  bit<48> tmp;  /\* Put the result back in \*/  hdr.p4calc.res = result;  /\* Swap the MAC addresses \*/  tmp = hdr.ethernet.dst\_addr;  hdr.ethernet.dst\_addr = hdr.ethernet.src\_addr;  hdr.ethernet.src\_addr = tmp;  ig\_tm\_md.ucast\_egress\_port = ig\_intr\_md.ingress\_port;  /\* Send the packet back to the port it came from \*/  //standard\_metadata.egress\_spec = standard\_metadata.ingress\_port;  }  action drop() {  ig\_dprsr\_md.drop\_ctl = 1;  }  action l3\_switch(PortId\_t port,bit<1> dir) {  ig\_tm\_md.ucast\_egress\_port = port;  hdr.ipv4.ttl = hdr.ipv4.ttl - 1;  }    action operation\_add() {  send\_back(hdr.p4calc.operand\_a + hdr.p4calc.operand\_b);  }  action operation\_sub() {  send\_back(hdr.p4calc.operand\_a - hdr.p4calc.operand\_b);  }  action operation\_and() {  send\_back(hdr.p4calc.operand\_a & hdr.p4calc.operand\_b);  }  action operation\_or() {  send\_back(hdr.p4calc.operand\_a | hdr.p4calc.operand\_b);  }  action operation\_xor() {  send\_back(hdr.p4calc.operand\_a ^ hdr.p4calc.operand\_b);  }  table forward{  key = { ig\_intr\_md.ingress\_port: exact; }  actions = {  drop; l3\_switch;  @defaultonly NoAction;  }  const default\_action = NoAction();  size = 250;  }  table calculate {  key = {  hdr.p4calc.op : exact;  }  actions = {  operation\_add;  operation\_sub;  operation\_and;  operation\_or;  operation\_xor;  drop;  }  const default\_action = drop();  const entries = {  P4CALC\_PLUS : operation\_add();  P4CALC\_MINUS: operation\_sub();  P4CALC\_AND : operation\_and();  P4CALC\_OR : operation\_or();  P4CALC\_CARET: operation\_xor();  }  }    /\* The algorithm \*/  apply {  if (hdr.p4calc.isValid()) {  calculate.apply();  } else {  drop();  }  if (hdr.ipv4.isValid()) {  forward.apply();  }  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* D E P A R S E R \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  control IngressDeparser(packet\_out pkt,  /\* User \*/  inout my\_ingress\_headers\_t hdr,  in my\_ingress\_metadata\_t meta,  /\* Intrinsic \*/  in ingress\_intrinsic\_metadata\_for\_deparser\_t ig\_dprsr\_md)  {  apply{  //pkt.emit(hdr);  pkt.emit(hdr.ethernet);  pkt.emit(hdr.p4calc);  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* E G R E S S P R O C E S S I N G \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* H E A D E R S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  struct my\_egress\_headers\_t {  }  /\*\*\*\*\*\*\*\* G L O B A L E G R E S S M E T A D A T A \*\*\*\*\*\*\*\*\*/  struct my\_egress\_metadata\_t {  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* P A R S E R \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  parser EgressParser(packet\_in pkt,  /\* User \*/  out my\_egress\_headers\_t hdr,  out my\_egress\_metadata\_t meta,  /\* Intrinsic \*/  out egress\_intrinsic\_metadata\_t eg\_intr\_md)  {  /\* This is a mandatory state, required by Tofino Architecture \*/  state start {  pkt.extract(eg\_intr\_md);  transition accept;  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* M A T C H - A C T I O N \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  control Egress(  /\* User \*/  inout my\_egress\_headers\_t hdr,  inout my\_egress\_metadata\_t meta,  /\* Intrinsic \*/  in egress\_intrinsic\_metadata\_t eg\_intr\_md,  in egress\_intrinsic\_metadata\_from\_parser\_t eg\_prsr\_md,  inout egress\_intrinsic\_metadata\_for\_deparser\_t eg\_dprsr\_md,  inout egress\_intrinsic\_metadata\_for\_output\_port\_t eg\_oport\_md)  {  apply {  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* D E P A R S E R \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  control EgressDeparser(packet\_out pkt,  /\* User \*/  inout my\_egress\_headers\_t hdr,  in my\_egress\_metadata\_t meta,  /\* Intrinsic \*/  in egress\_intrinsic\_metadata\_for\_deparser\_t eg\_dprsr\_md)  {  apply {  pkt.emit(hdr);  }  }  /\*\*\*\*\*\*\*\*\*\*\*\* F I N A L P A C K A G E \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  Pipeline(  IngressParser(),  Ingress(),  IngressDeparser(),  EgressParser(),  Egress(),  EgressDeparser()  ) pipe;  Switch(pipe) main; |