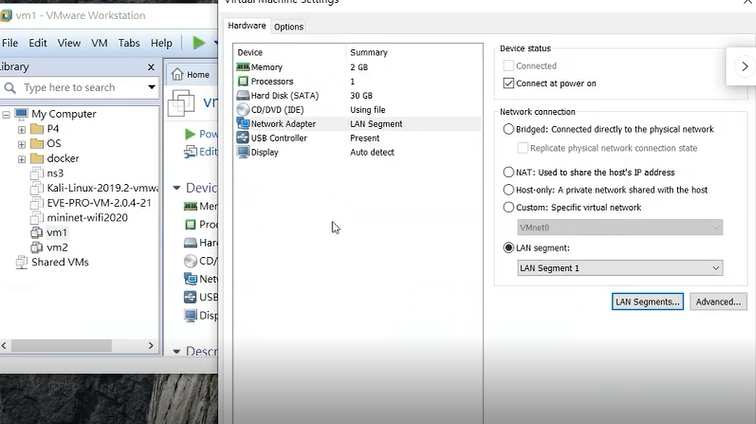
**0630如何做出p4軟體交換機及正常封包判斷**

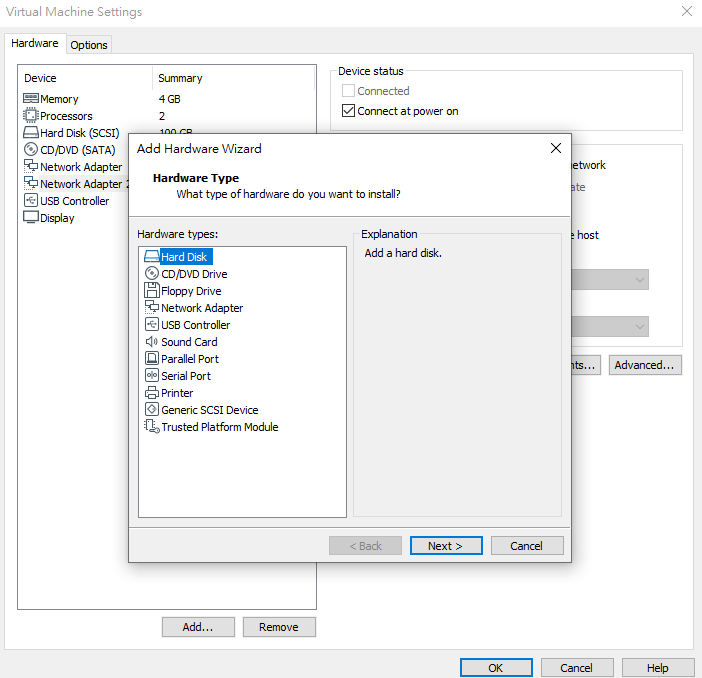
<http://csie.nqu.edu.tw/smallko/sdn/p4switch.htm>

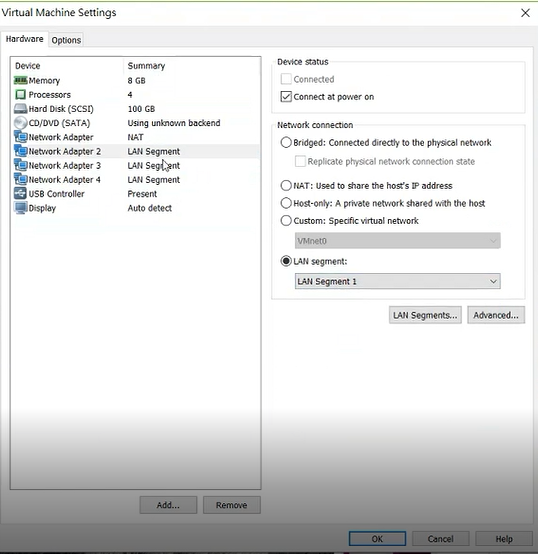
**前置操作步驟：**

1. 複製兩台mininet虛擬機vm1(h1),vm2(h2)

游標移至mininet ->右鍵->manage->clone

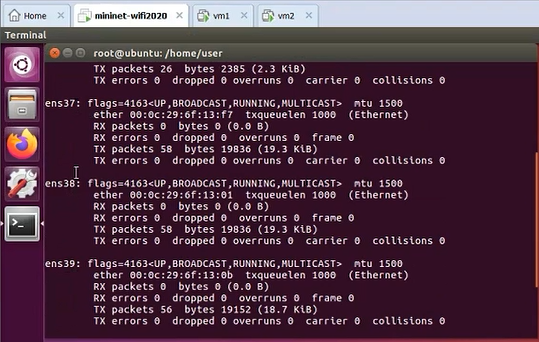
1. 在vm1點edit setting，network adapter選LAN Segment，如果剛開始沒有，可以點按紐LAN Segment-> Add 3個(LAN Segment1,LAN Segment2,LAN Segment3)
2. 第一台(vm1)選LAN Segment1
3. vm2重複步驟2，LAN Segment改成LAN Segment2
4. mininet那台虛擬機，增加總共四張網路卡，第一張用NAT，第二張用LAN Segment1，第三張用LAN Segment2，第四張用LAN Segment3





1. 弄好之後把三台機器打開(power on)

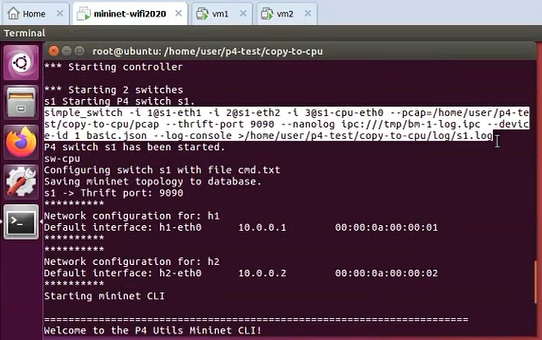
**執行：**

1.切到mininet，打開終端機，切到超級使用者輸入ifconfig，會看到

*ens37,38,39(不一定是37,38,39)就是剛剛增加的網路卡LAN…1, LAN…2, LAN…3*

2.切到p4-test資料夾(cd p4-test)，切到copy-to-cpu，然後p4run

3.把反白部分指令複製，然後結束剛剛run的mininet(exit)

4.然後切到vm1，切到超級使用者(su)，然後用ifconfig查看是不會有ip位址的

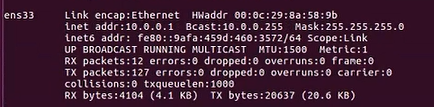


5. vm1和vm2，手動加上ip位址

Vm1:



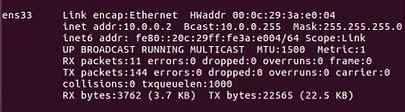
10.0.0.1加上了：



Vm2:



10.0.0.2加上了：



切到vm1

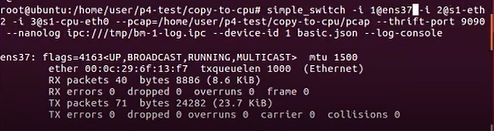
現在用vm1去ping vm2是不通的



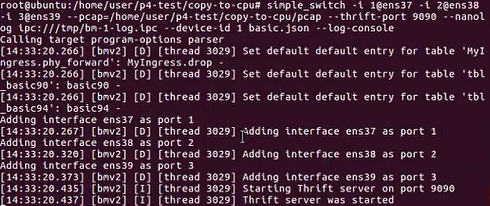
1. 切到mininet虛擬機，把剛剛複製的指令修改一下貼上

(倒數第二行console後面拿掉，第一行1@s1-eth1改1@ens37，看網卡編號，這邊是ens37)

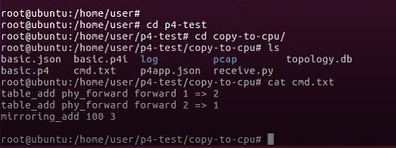
1@:1號埠,2@:2號埠



1. 然後第一行2@s1-eth2改2@ens38，第二行3@s1-cpu-eth0改3@ens39然後enter執行會跑這樣



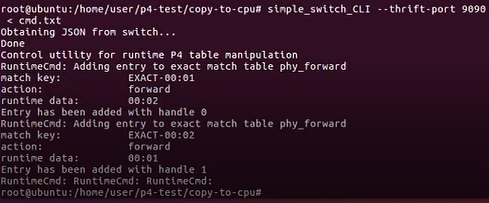
*這時候再切到vm1 去ping vm2還是不能通，因為規則還沒下*

1. 切到mininet下規則，重開一個終端機並切到copy-to-cpu 

下規則指令：simple\_switch\_CLI –thrift-port 9090 < cmd.txt

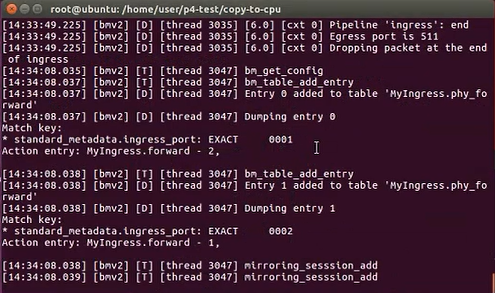
把命令透過指令丟到cmd.txt

命令丟進去後長這樣

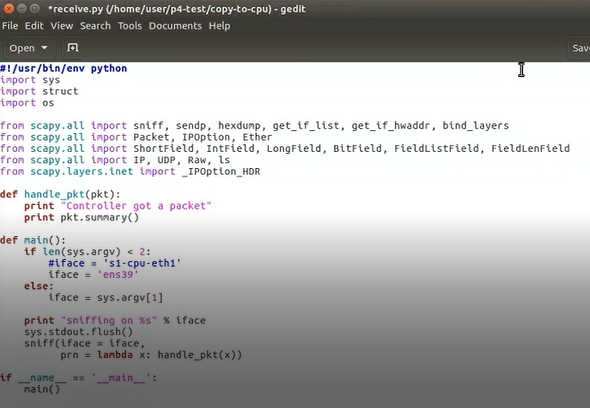


在另一台終端機可以看到規則被寫入了

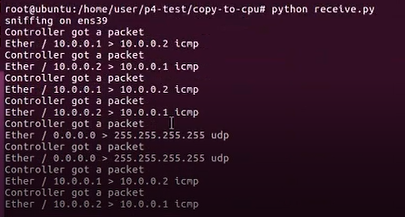
命令是1號埠進去2號埠出來，2號埠進去1號埠出來



1. 再切到vm1去ping vm2就可以通了
2. 然後回到mininet虛擬機，在copy-to-cpu資料夾輸入gedit receive.py &
3. 把ifac加ens39並save，如圖



1. 執行python receive.py



LAN1接的是第一台主機，LAN2接的是第二台，LAN3就可以讓他丟到控制器上

**裝資料庫系統logstash&資料庫influxdb**

**什麼是logstash？**

<https://www.elastic.co/cn/logstash>

蒐集資料、解析資料、並進行資料轉換成想要的格式，就可以寫道你想寫進的地方去。Ex:檔案裡、資料庫

我們會先在網路上傳送正常的ping的封包，然後讓logstash得到ping封包的屬性，這個logstash會把ping的封包寫到資料庫系統裡面。

為什麼要透過它而不直接寫進資料庫，是因為ping的封包裡面得到像封包的長度，或其他像來源IP，這些東西寫進去資料庫的時候，會有一種格式的轉換，或者是有些東西需要做篩選，就需要這樣的工具。

安裝logstash步驟：

<https://www.elastic.co/guide/en/logstash/current/installing-logstash.html>

1. 打開mininet虛擬機的終端機，切超級使用者
2. 貼上指令(按順序)

wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -

sudo apt-get install apt-transport-https

echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee -a /etc/apt/sources.list.d/elastic-7.x.list

sudo apt-get update && sudo apt-get install logstash

**什麼是influxdb？**

跟時間有關的資料庫系統

安裝influxdb步驟：

<https://docs.influxdata.com/influxdb/v1.8/introduction/install/>

1. 打開mininet虛擬機的終端機，切超級使用者
2. 貼上指令(按順序)

wget -qO- https://repos.influxdata.com/influxdb.key | sudo apt-key add -

source /etc/os-release

echo "deb https://repos.influxdata.com/debian **$(**lsb\_release -cs**)** stable" | sudo tee /etc/apt/sources.list.d/influxdb.list

sudo apt-get update && sudo apt-get install influxdb

使用指令啟動資料庫：systemctl start influxdb

查看有沒有成功啟動：systemctl status influxdb

**登錄資料庫做基本設定**

步驟：

<https://dotblogs.com.tw/DizzyDizzy/2018/07/10/influxUbuntu>

1.進入資料庫指令：influx



2.執行指令

開帳號跟名字。Ex:建立一組帳號叫admin，密碼也是admin

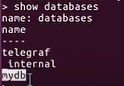
**CREATE** **USER** **admin** **WITH** **PASSWORD** 'admin' **WITH** ALL **PRIVILEGES**

創建一個資料庫mydb

**creat** **database mydb**

查看資料庫

**show database**



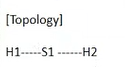
1. 使用exit關閉，接著打指令輸入資料庫帳密就可以進去

influx –username admin –password admin

然後輸入show database 就會出現如上圖的畫面

**判斷是否是正常封包**

<http://csie.nqu.edu.tw/smallko/sdn/p4-svm.htm>



S1讓它有一個clone的功能，也就是H1,H2流經到S1的封包，都會clone到cpu，就是controller這邊。Controller會跑logstash做資料蒐集，蒐集完之後會存到資料庫裏面。

首先，要先送正常的ping封包，讓它去了解什麼是正常的封包，ping的封包有什麼樣的行為。第二，去傳送攻擊型封包，傳送大量ping的封包，S1會收到，logstash也會把資料送到資料庫。

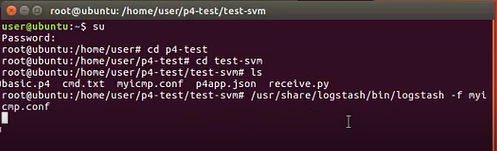
有這兩筆資料(正常&不正常)，就要透過機器學習去建立模型，去分析什麼是正常ping什麼是不正常ping。H1可以丟封包，只要封包經過S1，它就會去判斷是不是正常的封包。

基本上，ping的封包如果沒有特別指定一些參數，正常一秒鐘會送一個ping，封包大小大概不到100byte，就屬於正常型封包。攻擊型封包，每秒鐘產生的量會很大，可能上百上千個，每個ping的封包大概1000byte以上，因為他需要把對方的資源消耗完畢。

簡單來說，就是receive.py這支程式收到資料以後，會丟到logstash(myicmp.conf)，logstash收到以後，就會把對應的資料寫到資料庫裏面去。

**操作步驟：**

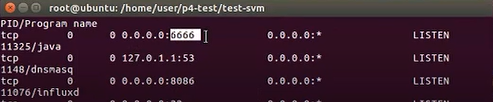
1. 打開mininet虛擬機，打開終端機，切到p4-test資料夾，建立新資料夾(mkdir test-svm)
2. 切到test-svm資料夾，然後編輯(gedit )basic.p4,cmd.txt,p4app.json &把程式碼分別複製貼上並儲存
3. 接著gedit receive.py &，myicmp.conf，一樣程式碼複製貼上
4. 開好幾個終端機，通通切到超級使用者
5. 終端機1執行influxdb。(influx –username admin –password admin)，然後show databases可以看到有mydb資料庫
6. 用指令use mydb使用資料庫，使用指令show measurement可以顯示表格內容，一開始什麼都沒有。
7. 打開終端機2，執行logstash。如下所示操作。



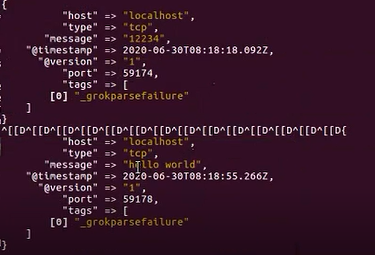
看到這個就表示成功啟動(若有問題，可以試著安裝/usr/share/logstash/bin/logstash-plugin install logstash-output-influxdb)



1. 打開終端機3，切到p4-test底下的test-svm資料夾，輸入指令netstate –tulnp，當logstash跑起來，6666埠就出現了。



1. 終端機3試著輸入echo “12334” | nc localhost 6666，終端機2就會出現訊息

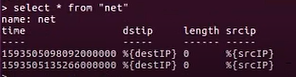
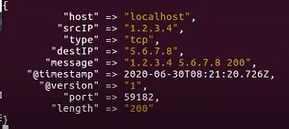


Influxdb是一種時間型資料庫，它跟傳統資料庫有點不同，基本上，丟了一個資訊，就會把這個資訊透過這種型式寫到資料庫。

1. 再回到終端機1，再一次輸入show measurement就會多一個net。

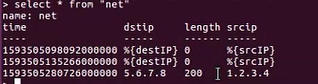


1. 輸入指令select \* from “net”可以查看內容。剛剛如果輸入hello word是沒辦法轉換的，但如果是輸入echo “1.2.3.4 5.6.7.8 200” | nc localhost 6666是會被轉換的，它會自動解析srcip是1.2.3.4，destip是5.6.7.8，length是200。



收到的訊息會用這種格式丟給logstash，logstash透過分析就會把來源ip,目的ip，封包長度寫進資料庫，就會變下圖的格式。

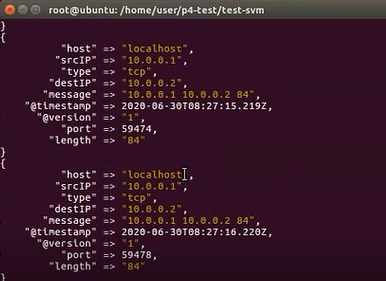




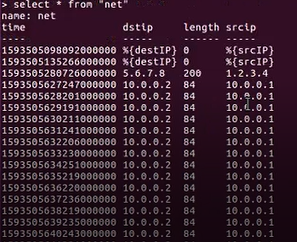
**執行步驟：**

1. 原始終端機執行p4run(資料夾p4-test/test-svm)
2. 終端機3輸入指令python reveive.py
3. 原始終端機輸入h1 ping –c 30 h2

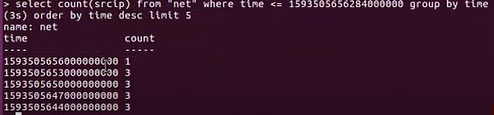
Ping的時候，終端機2的logstash會一直蒐集資料



1. 終端機1輸入select \* from “net”，就會出現資料



1. 也可以做下面這樣的輸入



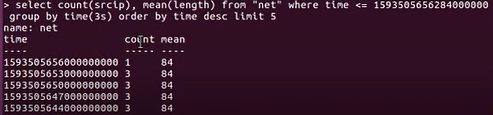
***limit 5***:只顯示五筆

***order by time desc***:用時間進行排序，從距離現在最新的時間往回排序到舊的時間(可以看1593…5600…往回排到1593…4400…)

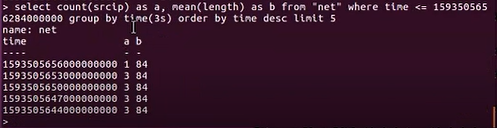
***group by time (3s)***:用三秒鐘(內)把資料統整起來，合在一起算

***count(srcip)***:把三秒鐘之內的來源ip個數做統計

第一筆比較特別，它可能沒算的那麼準，但下面幾筆每次算出來都是3，意思就是說，ping封包一秒送一個，三秒就送三個。這邊只有一個特徵值也就是封包個數，可以再多加一個特徵值(封包大小)如下：



意思是三秒鐘有三個封包，平均的封包大小是84byte

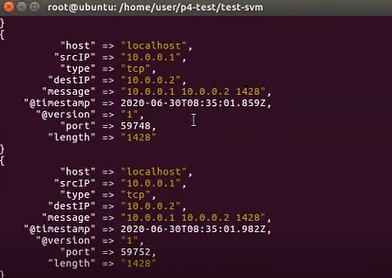
上面的東西有時候名稱會比較複雜，所以可以用一些單字代替，例如。多加一個指令count(srcip)as a，用a代替count，b就是mean。

這樣的型態舊是屬於一種正常的封包。

1. 接著測試攻擊型封包，先安裝hping3(apt install hping3)
2. 終端機3再啟動一次python reveive.py
3. 原始終端機(開好mininet那個)輸入指令h1 hping3 –V -1 –d 1400 –fsat h2

h1跟h2之間會大量發送ping封包，每個封包大約1400byte，會盡最快速度送



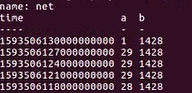


終端機2的logstash一樣在蒐集資料，封包跑到300就可以中斷

1. 終端機1輸入select \* from “net”，找到最新一筆資料記錄的時間並複製，然後輸入這個指令，把原來的時間部分(1593…這串數字)替換掉



就會出現如下圖。a的部分數量變多了，b的部分大小變大了。



這兩個特徵值就可以判斷什麼是正常的封包，什麼是危險的封包。

**Receive.py**

#!/usr/bin/env python

import sys

import struct

import os

from scapy.all import sniff

from scapy.all import Packet, IPOption, Ether

from scapy.all import IP, UDP, ICMP, Raw, ls

def handle\_pkt(pkt):

print "Controller got a packet"

print pkt.summary()

先判斷是不是ICMP封包，如果是且它是一個請求封包(request)，也就是傳送端送出去的封包，不是收到回覆的封包

然後把來源ip,目的ip，封包大小長度記起來，把這些資訊透過nc丟到本地端的6666埠，這個6666埠是要開給logstash用

if ICMP in pkt and pkt[ICMP].type == 8:

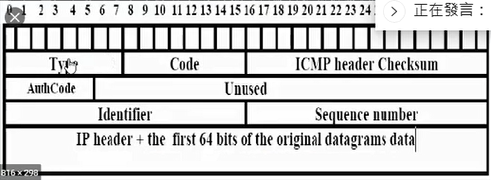
ip\_src=pkt[IP].src

ip\_dst=pkt[IP].dst

ip\_len=pkt[IP].len

print ip\_src,ip\_dst,ip\_len

os.system(" echo %s %s %s | nc localhost 6666" % (ip\_src,ip\_dst,ip\_len))



這是一個ping封包，type如果是8就代表是echo request，回覆(echo reply)就是1，

def main():

s1-cpu-eth1:s1的出口，cpu的監聽端口進行聆聽，聆聽完以後會送到handle\_pkt

if len(sys.argv) < 2:

iface = 's1-cpu-eth1'

else:

iface = sys.argv[1]

print "sniffing on %s" % iface

sys.stdout.flush()

sniff(iface = iface,

prn = lambda x: handle\_pkt(x))

if \_\_name\_\_ == '\_\_main\_\_':

main()

**myicmp.conf(logstash的配置檔)**

(這段底線到底線是輸入)

input {

tcp {

要從網路的6666埠進來

type => "tcp"

port => 6666 //本地端開起6666埠

mode => "server"

}

}

filter{

grok{

match => ["message", "%{IP:srcIP} %{IP:destIP} %{INT:length}"]

}

}

(這段底線到底線是輸出)

output {

influxdb {

輸出到influxdb上

db => "mydb"

host => "localhost"

port => "8086"

user => "admin"

password => "admin"

measurement => "net" //在influxdb裡，table(MySQL)叫做measurement

資料要寫到measurement裡面

allow\_time\_override => true

flush\_size => "1"

data\_points => {

"srcip"=>"%{srcIP}"

寫來源、目的ip，封包長度進去

指名寫進去的封包長度是整數型態

"dstip"=>"%{destIP}"

"length"=>"%{length}"

}

coerce\_values => {

"length" => "integer"

}

}

stdout { codec => rubydebug }

}

**basic.p4**

/\* -\*- P4\_16 -\*- \*/

#include <core.p4>

#include <v1model.p4>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* H E A D E R S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct metadata {

/\* empty \*/

}

struct headers {

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* P A R S E R \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

parser MyParser(packet\_in packet,

out headers hdr,

inout metadata meta,

inout standard\_metadata\_t standard\_metadata) {

state start {

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 forward(bit<9> port) {

standard\_metadata.egress\_spec = port;

}

table phy\_forward {

key = {

standard\_metadata.ingress\_port: exact;

}

actions = {

forward;

drop;

}

size = 1024;

default\_action = drop();

}

apply {

phy\_forward.apply();

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

control MyEgress(inout headers hdr,

inout metadata meta,

inout standard\_metadata\_t standard\_metadata) {

apply {

if (standard\_metadata.instance\_type == 0 ){

clone(CloneType.E2E,100);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\* 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 {

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* D E P A R S E R \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

control MyDeparser(packet\_out packet, in headers hdr) {

apply {

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* S W I T C H \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

V1Switch(

MyParser(),

MyVerifyChecksum(),

MyIngress(),

MyEgress(),

MyComputeChecksum(),

MyDeparser()

) main;

**Cmd.txt**

table\_add phy\_forward forward 1 => 2

table\_add phy\_forward forward 2 => 1

mirroring\_add 100 3

**p4app.json**

{

"program": "basic.p4",

"switch": "simple\_switch",

"compiler": "p4c",

"options": "--target bmv2 --arch v1model --std p4-16",

"switch\_cli": "simple\_switch\_CLI",

"cli": true,

"pcap\_dump": true,

"enable\_log": true,

"topo\_module": {

"file\_path": "",

"module\_name": "p4utils.mininetlib.apptopo",

"object\_name": "AppTopoStrategies"

},

"controller\_module": null,

"topodb\_module": {

"file\_path": "",

"module\_name": "p4utils.utils.topology",

"object\_name": "Topology"

},

"mininet\_module": {

"file\_path": "",

"module\_name": "p4utils.mininetlib.p4net",

"object\_name": "P4Mininet"

},

"topology": {

"assignment\_strategy": "l2",

"links": [["h1", "s1"], ["h2", "s1"]],

"hosts": {

"h1": {

},

"h2": {

}

},

"switches": {

"s1": {

"cli\_input": "cmd.txt",

"program": "basic.p4",

"cpu\_port": true

}

}

}

}