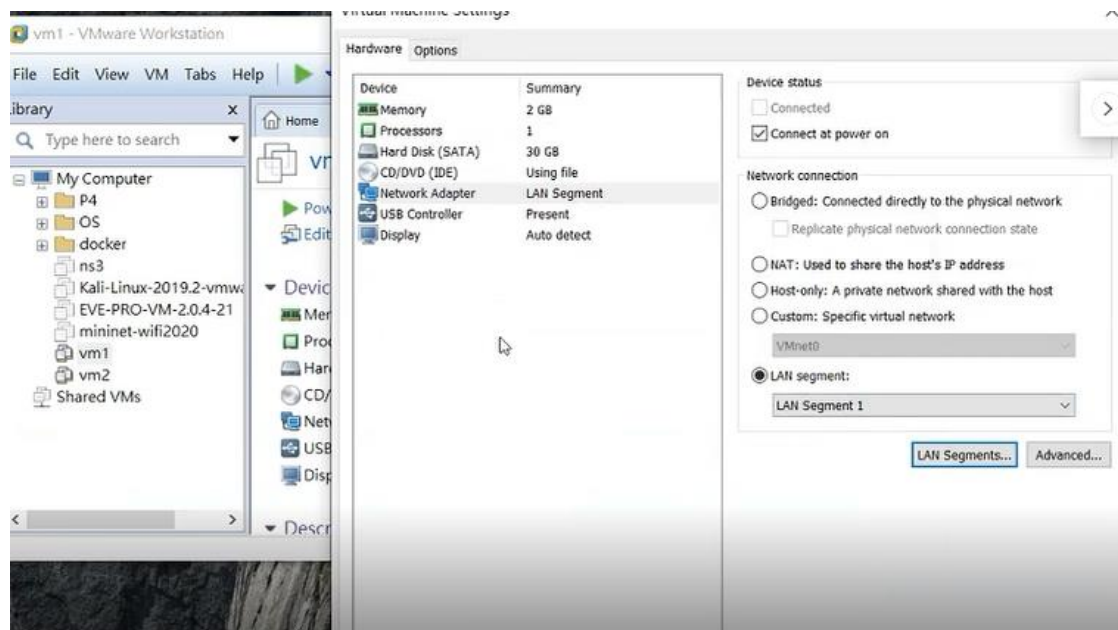


0630 如何做出 p4 軟體交換機及正常封包判斷

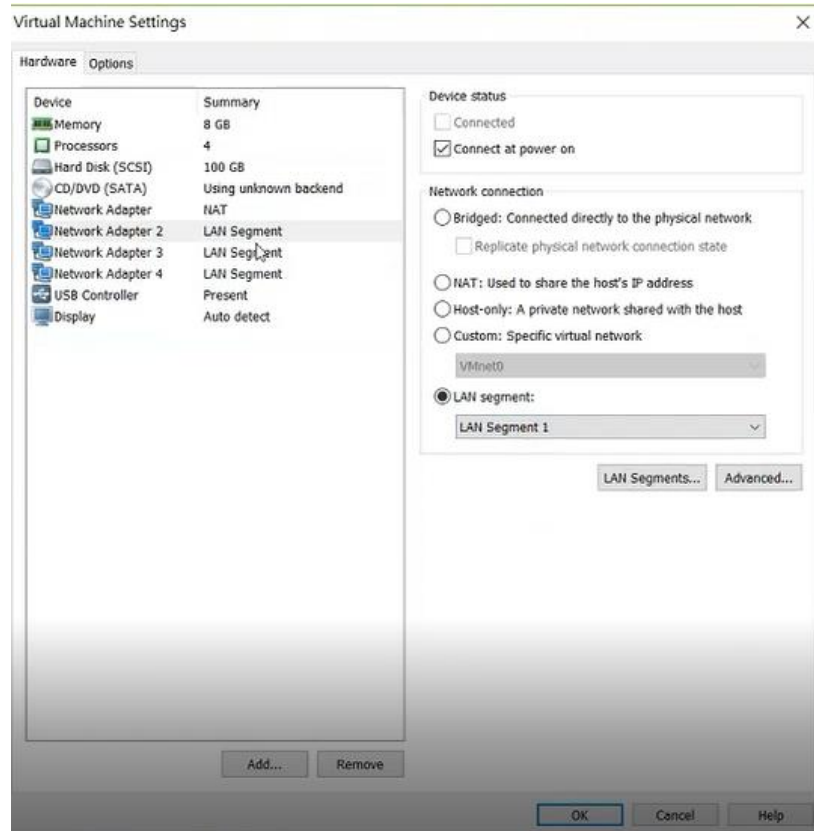
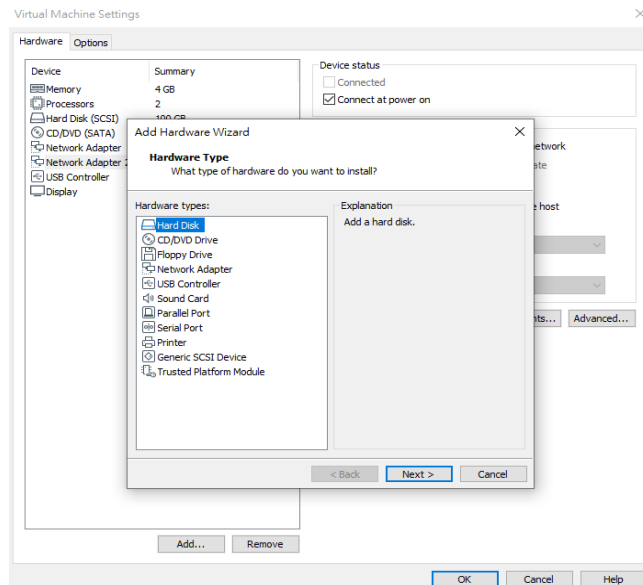
<http://csie.nqu.edu.tw/smallko/sdn/p4switch.htm>

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
2. 在 vm1 點 edit setting，network adapter 選 LAN Segment，如果剛開始沒有，可以點按鈕 LAN Segment-> Add 3 個(LAN Segment1,LAN Segment2,LAN Segment3)
3. 第一台(vm1)選 LAN Segment1



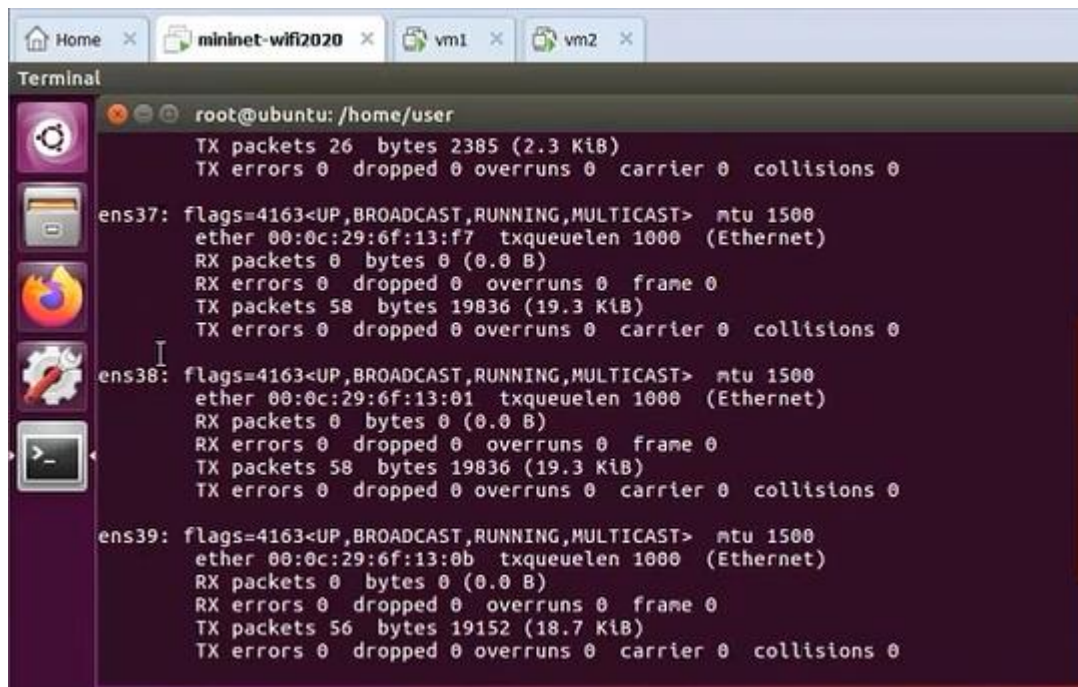
4. vm2 重複步驟 2，LAN Segment 改成 LAN Segment2
5. mininet 那台虛擬機，增加總共四張網路卡，第一張用 NAT，第二張用 LAN Segment1，第三張用 LAN Segment2，第四張用 LAN Segment3



6. 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



```
root@ubuntu: /home/user
TX packets 26  bytes 2385 (2.3 KiB)
TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
ether 00:0c:29:6f:13:f7  txqueuelen 1000  (Ethernet)
RX packets 0  bytes 0 (0.0 B)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 58  bytes 19836 (19.3 KiB)
TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

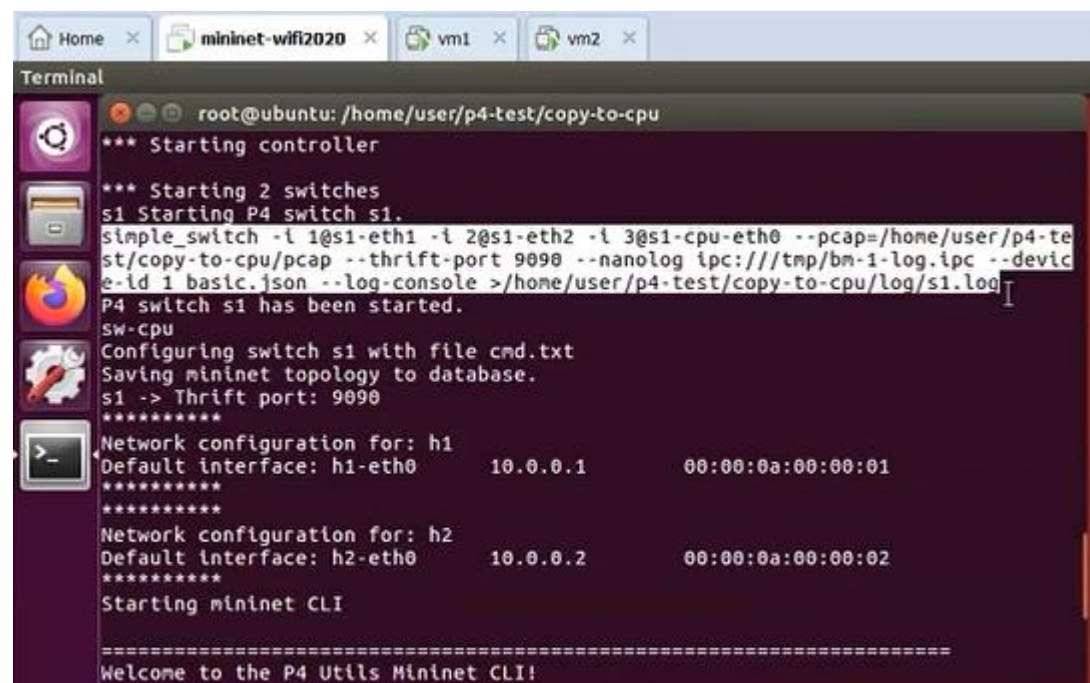
ens38: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
ether 00:0c:29:6f:13:01  txqueuelen 1000  (Ethernet)
RX packets 0  bytes 0 (0.0 B)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 58  bytes 19836 (19.3 KiB)
TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

ens39: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
ether 00:0c:29:6f:13:0b  txqueuelen 1000  (Ethernet)
RX packets 0  bytes 0 (0.0 B)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 56  bytes 19152 (18.7 KiB)
TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
```

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

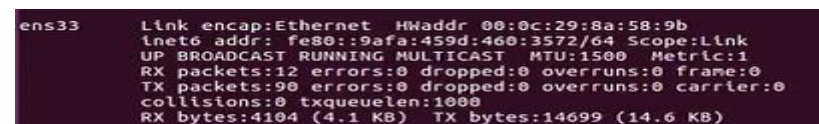
2.切到 p4-test 資料夾(cd p4-test)，切到 copy-to-cpu，然後 p4run

3.把反白部分指令複製，然後結束剛剛 run 的 mininet(exit)



```
root@ubuntu: /home/user/p4-test/copy-to-cpu
*** Starting controller
*** Starting 2 switches
s1 Starting P4 switch s1.
Simple_switch -i 1@s1-eth1 -i 2@s1-eth2 -i 3@s1-cpu-eth0 --pcap=/home/user/p4-test/copy-to-cpu/pcap --thrift-port 9090 --nanolog ipc:///tmp/bm-1-log.ipc --device-id 1 basic.json --log-console >/home/user/p4-test/copy-to-cpu/log/s1.log
P4 switch s1 has been started.
sw-cpu
Configuring switch s1 with file cmd.txt
Saving mininet topology to database.
s1 -> Thrift port: 9090
*****
Network configuration for: h1
Default interface: h1-eth0      10.0.0.1      00:00:0a:00:00:01
*****
Network configuration for: h2
Default interface: h2-eth0      10.0.0.2      00:00:0a:00:00:02
*****
Starting mininet CLI
=====
Welcome to the P4 Utils Mininet CLI!
```

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



```
ens33: Link encap:Ethernet  HWaddr 00:0c:29:8a:58:9b
inet6 addr: fe80::9afa:459d:460:3572/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:12 errors:0 dropped:0 overruns:0 frame:0
TX packets:90 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:4104 (4.1 KB)  TX bytes:14699 (14.6 KB)
```

5. vm1 和 vm2，手動加上 ip 位址

Vm1:

```
root@user-VirtualBox:/home/user# ip addr add 10.0.0.1/24 brd + dev ens33
root@user-VirtualBox:/home/user#
```

10.0.0.1 加上了：

```
ens33    Link encap:Ethernet  HWaddr 00:0c:29:8a:58:9b
          inet addr:10.0.0.1  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::9afa:459d:460:3572/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:127 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4104 (4.1 KB)  TX bytes:20637 (20.6 KB)
```

Vm2:

```
user@user-VirtualBox:~$ su
Password:
root@user-VirtualBox:/home/user# ip addr add 10.0.0.2/24 brd + dev ens33
```

10.0.0.2 加上了：

```
ens33    Link encap:Ethernet  HWaddr 00:0c:29:3a:e0:04
          inet addr:10.0.0.2  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe3a:e004/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:11 errors:0 dropped:0 overruns:0 frame:0
          TX packets:144 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3762 (3.7 KB)  TX bytes:22565 (22.5 KB)
```

切到 vm1

現在用 vm1 去 ping vm2 是不通的

```
root@user-VirtualBox:/home/user# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
```

7. 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

```
root@ubuntu:/home/user/p4-test/copy-to-cpu# simple_switch -i 1@ens37 -i 2@s1-eth
2 -i 3@s1-cpu-eth0 --pcap=/home/user/p4-test/copy-to-cpu/pcap --thrift-port 9090
--nlog ipc:///tmp/bn-1-log.ipc --device-id 1 basic.json --log-console

ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
ether 00:0c:29:6f:13:f7 txqueuelen 1000 (Ethernet)
RX packets 40 bytes 8886 (8.6 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 71 bytes 24282 (23.7 KiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

8. 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


```

root@ubuntu:/home/user/p4-test/copy-to-cpu# simple_switch -i 1@ens37 -i 2@ens38
-i 3@ens39 --pcap=/home/user/p4-test/copy-to-cpu/pcap --thrift-port 9090 --nanolo
og ipc:///tmp/bm-1-log.ipc --device-id 1 basic.json --log-console
Calling target program-options parser
[14:33:20.266] [bmv2] [D] [thread 3029] Set default default entry for table 'MyI
ngress.phy_forward': MyIngress.drop -
[14:33:20.266] [bmv2] [D] [thread 3029] Set default default entry for table 'tbl
_basic90': basic90 -
[14:33:20.266] [bmv2] [D] [thread 3029] Set default default entry for table 'tbl
_basic94': basic94 -
Adding interface ens37 as port 1
[14:33:20.267] [bmv2] [D] [thread 3029] Adding interface ens37 as port 1
Adding interface ens38 as port 2
[14:33:20.320] [bmv2] [D] [thread 3029] Adding interface ens38 as port 2
Adding interface ens39 as port 3
[14:33:20.373] [bmv2] [D] [thread 3029] Adding interface ens39 as port 3
[14:33:20.435] [bmv2] [I] [thread 3029] Starting Thrift server on port 9090
[14:33:20.437] [bmv2] [I] [thread 3029] Thrift server was started

```

這時候再切到 vm1 去 ping vm2 還是不能通，因為規則還沒下

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

```

root@ubuntu:/home/user#
root@ubuntu:/home/user# cd p4-test
root@ubuntu:/home/user/p4-test# cd copy-to-cpu/
root@ubuntu:/home/user/p4-test/copy-to-cpu# ls
basic.json  basic.p4l  log        pcap        topology.db
basic.p4    cmd.txt    p4app.json receive.py
root@ubuntu:/home/user/p4-test/copy-to-cpu# cat cmd.txt
table_add phy_forward forward 1 => 2
table_add phy_forward forward 2 => 1
mirroring_add 100 3
root@ubuntu:/home/user/p4-test/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

```

root@ubuntu:/home/user/p4-test/copy-to-cpu# simple_switch_CLI --thrift-port 9090
< cmd.txt
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: Adding entry to exact match table phy_forward
match key:          EXACT-00:01
action:             forward
runtime data:       00:02
Entry has been added with handle 0
RuntimeCmd: Adding entry to exact match table phy_forward
match key:          EXACT-00:02
action:             forward
runtime data:       00:01
Entry has been added with handle 1
RuntimeCmd: RuntimeCmd: RuntimeCmd:
root@ubuntu:/home/user/p4-test/copy-to-cpu#

```

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

```
root@ubuntu: /home/user/p4-test/copy-to-cpu
[14:33:49.225] [bm_v2] [D] [thread 3035] [6.0] [cxt 0] Pipeline 'ingress': end
[14:33:49.225] [bm_v2] [D] [thread 3035] [6.0] [cxt 0] Egress port is 511
[14:33:49.225] [bm_v2] [D] [thread 3035] [6.0] [cxt 0] Dropping packet at the end
of ingress
[14:34:08.035] [bm_v2] [T] [thread 3047] bm_get_config
[14:34:08.037] [bm_v2] [T] [thread 3047] bm_table_add_entry
[14:34:08.037] [bm_v2] [D] [thread 3047] Entry 0 added to table 'MyIngress.phy_for
ward'
[14:34:08.037] [bm_v2] [D] [thread 3047] Dumping entry 0
Match key:
* standard_metadata.ingress_port: EXACT    0001
Action entry: MyIngress.forward - 2,
[14:34:08.038] [bm_v2] [T] [thread 3047] bm_table_add_entry
[14:34:08.038] [bm_v2] [D] [thread 3047] Entry 1 added to table 'MyIngress.phy_for
ward'
[14:34:08.038] [bm_v2] [D] [thread 3047] Dumping entry 1
Match key:
* standard_metadata.ingress_port: EXACT    0002
Action entry: MyIngress.forward - 1,
[14:34:08.038] [bm_v2] [T] [thread 3047] mirroring_session_add
[14:34:08.039] [bm_v2] [T] [thread 3047] mirroring_session_add
```

10. 再切到 vm1 去 ping vm2 就可以通了

11. 然後回到 mininet 虛擬機，在 copy-to-cpu 資料夾輸入 gedit receive.py &

12. 把 ifac 加 ens39 並 save，如圖

```
*receive.py (/home/user/p4-test/copy-to-cpu) - gedit
File Edit View Search Tools Documents Help
Open [icon] Save

#!/usr/bin/env python
import sys
import struct
import os

from scapy.all import sniff, sendp, hexdump, get_if_list, get_if_hwaddr, bind_layers
from scapy.all import Packet, IPOption, Ether
from scapy.all import ShortField, IntField, LongField, BitField, FieldListField, FieldLenField
from scapy.all import IP, UDP, Raw, ls
from scapy.layers.inet import _IPOption_HDR

def handle_pkt(pkt):
    print "Controller got a packet"
    print pkt.summary()

def main():
    if len(sys.argv) < 2:
        #iface = 's1-cpu-eth1'
        iface = 'ens39'
    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()
```

13. 執行 python receive.py

```
root@ubuntu:/home/user/p4-test/copy-to-cpu# python receive.py
sniffing on ens39
Controller got a packet
Ether / 10.0.0.1 > 10.0.0.2 icmp
Controller got a packet
Ether / 10.0.0.2 > 10.0.0.1 icmp
Controller got a packet
Ether / 10.0.0.1 > 10.0.0.2 icmp
Controller got a packet
Ether / 10.0.0.2 > 10.0.0.1 icmp
Controller got a packet
Ether / 0.0.0.0 > 255.255.255.255 udp
Controller got a packet
Ether / 0.0.0.0 > 255.255.255.255 udp
Controller got a packet
Ether / 10.0.0.1 > 10.0.0.2 icmp
Controller got a packet
Ether / 10.0.0.2 > 10.0.0.1 icmp
```

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，就可以寫進你想寫進的地方去。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


```
root@ubuntu:/home/user/Downloads# influx
Connected to http://localhost:8086 version 1.5.4
InfluxDB shell version: 1.5.4
>
```

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
```

```
> show databases
name: databases
name
----
telegraf
internal
mydb
```

3. 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>

[Topology]

H1-----S1 -----H2

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。

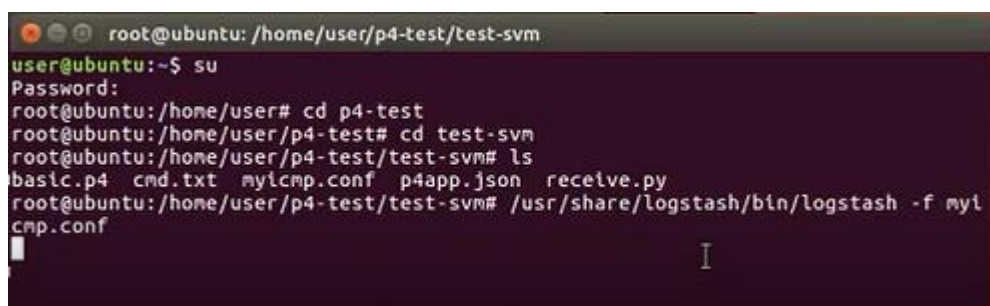
基本上，ping 的封包如果沒有特別指定一些參數，正常一秒鐘會送一個 ping，封包大小大概不到 100byte，就屬於正常型封包。攻擊型封包，每秒鐘產生的量

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：

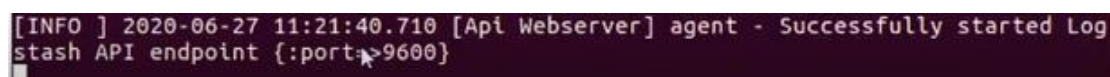
1. 打開 mininet 虛擬機，打開終端機，切到 p4-test 資料夾，建立新資料夾 (mkdir test-svm)
2. 切到 test-svm 資料夾，然後編輯(gedit)basic.p4,cmd.txt,p4app.json &把程式碼分別複製貼上並儲存
3. 接著 gedit receive.py &， myicmp.conf，一樣程式碼複製貼上
4. 開好幾個終端機，通通切到超級使用者
5. 終端機 1 執行 influxdb。(influx -username admin -password admin)，然後 show databases 可以看到有 mydb 資料庫
6. 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。



```
root@ubuntu: /home/user/p4-test/test-svm
user@ubuntu:~$ su
Password:
root@ubuntu:/home/user# cd p4-test
root@ubuntu:/home/user/p4-test# cd test-svm
root@ubuntu:/home/user/p4-test/test-svm# ls
basic.p4  cmd.txt  myicmp.conf  p4app.json  receive.py
root@ubuntu:/home/user/p4-test/test-svm# /usr/share/logstash/bin/logstash -f myicmp.conf
```

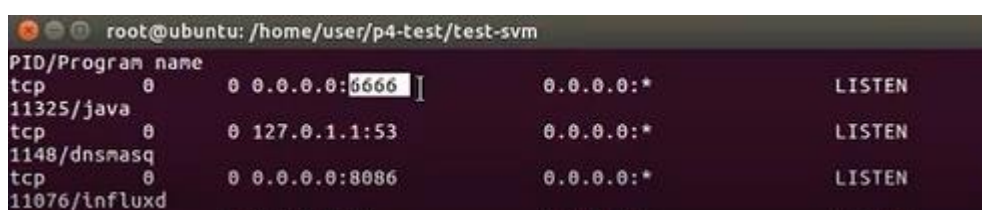
看到這個就表示成功啟動(若有問題，可以試著安裝

/usr/share/logstash/bin/logstash-plugin install logstash-output-influxdb)



```
[INFO ] 2020-06-27 11:21:40.710 [Api Webserver] agent - Successfully started Logstash API endpoint {:port=>9600}
```

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



```
root@ubuntu: /home/user/p4-test/test-svm
PID/Program name
tcp        0      0 0.0.0.0:6666          0.0.0.0:*            LISTEN
11325/java
tcp        0      0 0.0.0.0:53           0.0.0.0:*            LISTEN
1148/dnsmasq
tcp        0      0 0.0.0.0:8086          0.0.0.0:*            LISTEN
11076/influxd
```

9. 終端機 3 試著輸入 `echo "12334" | nc localhost 6666`，終端機 2 就會出現訊息

[illegible]

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。

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

```
> show measurements
name: measurements
name
----
```

11. 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`。

```
> select * from "net"
name: net
time                dstip                length  srcip
-----
1593505098092000000  %{destIP} 0          %{srcIP}
1593505135266000000  %{destIP} 0          %{srcIP}
```

```
{
  "host" => "localhost",
  "srcIP" => "1.2.3.4",
  "type" => "tcp",
  "destIP" => "5.6.7.8",
  "message" => "1.2.3.4 5.6.7.8 200",
  "@timestamp" => "2020-06-30T08:21:20.726Z",
  "@version" => "1",
  "port" => 59182,
  "length" => "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。

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

```
> select * from "net"
name: net
time          dstip          length  srcip
-----
1593505098092000000  %{destIP} 0      %{srcIP}
1593505135266000000  %{destIP} 0      %{srcIP}
1593505280726000000  5.6.7.8   200    1.2.3.4
```

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

```
root@ubuntu: /home/user/p4-test/test-svm
{
  "host" => "localhost",
  "srcIP" => "10.0.0.1",
  "type" => "tcp",
  "destIP" => "10.0.0.2",
  "message" => "10.0.0.1 10.0.0.2 84",
  "@timestamp" => 2020-06-30T08:27:15.219Z,
  "@version" => "1",
  "port" => 59474,
  "length" => "84"
}
{
  "host" => "localhost",
  "srcIP" => "10.0.0.1",
  "type" => "tcp",
  "destIP" => "10.0.0.2",
  "message" => "10.0.0.1 10.0.0.2 84",
  "@timestamp" => 2020-06-30T08:27:16.220Z,
  "@version" => "1",
  "port" => 59478,
  "length" => "84"
}
```

4. 終端機 1 輸入 select * from "net"，就會出現資料

```
> select * from "net"
name: net
time                dstip      length srcip
----                -
1593505098092000000 10.0.0.2  84      10.0.0.1
1593505135260000000 10.0.0.2  84      10.0.0.1
1593505280726000000 10.0.0.2  84      10.0.0.1
1593505627247000000 10.0.0.2  84      10.0.0.1
1593505628201000000 10.0.0.2  84      10.0.0.1
1593505629191000000 10.0.0.2  84      10.0.0.1
1593505630211000000 10.0.0.2  84      10.0.0.1
1593505631241000000 10.0.0.2  84      10.0.0.1
1593505632206000000 10.0.0.2  84      10.0.0.1
1593505633230000000 10.0.0.2  84      10.0.0.1
1593505634251000000 10.0.0.2  84      10.0.0.1
1593505635219000000 10.0.0.2  84      10.0.0.1
1593505636220000000 10.0.0.2  84      10.0.0.1
1593505637236000000 10.0.0.2  84      10.0.0.1
1593505638219000000 10.0.0.2  84      10.0.0.1
1593505639235000000 10.0.0.2  84      10.0.0.1
1593505640243000000 10.0.0.2  84      10.0.0.1
```

5. 也可以做下面這樣的輸入

```
> select count(srcip) from "net" where time <= 1593505656284000000 group by time
(3s) order by time desc limit 5
name: net
time                count
----                -
1593505656000000000 1
1593505653000000000 3
1593505650000000000 3
1593505647000000000 3
1593505644000000000 3
```


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，可以再加一個特徵值(封包大小)如下：

```
> select count(srcip), mean(length) from "net" where time <= 1593505656284000000
group by time(3s) order by time desc limit 5
name: net
time                count mean
----
1593505656000000000 1      84
1593505653000000000 3      84
1593505650000000000 3      84
1593505647000000000 3      84
1593505644000000000 3      84
```

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。

```
> select count(srcip) as a, mean(length) as b from "net" where time <= 1593505656284000000
group by time(3s) order by time desc limit 5
name: net
time                a b
----
1593505656000000000 1 84
1593505653000000000 3 84
1593505650000000000 3 84
1593505647000000000 3 84
1593505644000000000 3 84
>
```

這樣的型態舊是屬於一種正常的封包。

6. 接著測試攻擊型封包，先安裝 hping3(apt install hping3)
7. 終端機 3 再啟動一次 python reveive.py
8. 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

```
mininet> h1 hping3 -V -1 -d 1400 --fast h2
using h1-eth0, addr: 10.0.0.1, MTU: 9500
HPING 10.0.0.2 (h1-eth0 10.0.0.2): icmp mode set, 28 headers + 1400 data bytes
len=1428 ip=10.0.0.2 ttl=64 id=48210 tos=0 lplen=1428
icmp_seq=0 rtt=9.3 ms
```

```
root@ubuntu: /home/user/p4-test/test-svm
{
  "host" => "localhost",
  "srcIP" => "10.0.0.1",
  "type" => "tcp",
  "destIP" => "10.0.0.2",
  "message" => "10.0.0.1 10.0.0.2 1428",
  "@timestamp" => 2020-06-30T08:35:01.859Z,
  "@version" => "1",
  "port" => 59748,
  "length" => "1428"
}

{
  "host" => "localhost",
  "srcIP" => "10.0.0.1",
  "type" => "tcp",
  "destIP" => "10.0.0.2",
  "message" => "10.0.0.1 10.0.0.2 1428",
  "@timestamp" => 2020-06-30T08:35:01.982Z,
  "@version" => "1",
  "port" => 59752,
  "length" => "1428"
}
```

終端機 2 的 logstash 一樣在蒐集資料，封包跑到 300 就可以中斷

9. 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

```
1593506130059000000 10.0.0.2 1428 10.0.0.1
> select count(srcip) as a, mean(length) as b from "net" where time <= 1593506130059000000 group by time(3s) order by time desc limit 5
```

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。

```
name: net
time          a  b
----          -  -
1593506130000000000 1 1428
1593506127000000000 29 1428
1593506124000000000 29 1428
1593506121000000000 29 1428
1593506118000000000 28 1428
```

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。

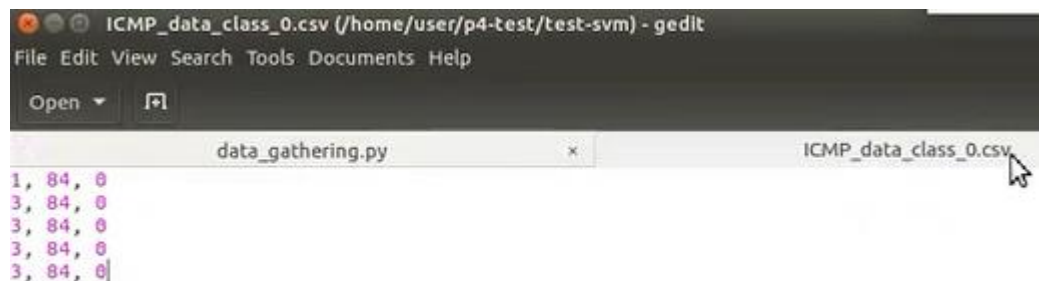
把資料(正常和不正成封包的資料)建立起來

因為剛剛做的都是直接在資料庫操作，沒有儲存成檔案建模，我們需要把資料建立起來。

步驟：

1. 開終端機 3 切到資料夾 test-svm，安裝軟體 `apt install python-influxdb`
2. 終端機 3 `gedit data_gathering.py` &
3. 在做這個實驗之前，要先把資料庫裡面清空，打開終端機 1，使用指令 `drop measurement net`(net 是名稱)就可以把表格刪掉
4. 終端機 3 啟動 `python receive.py`
5. 原始終端機輸入 `h1 ping -c 30 h2` 讓它 ping 30 秒
6. 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
7. 打開 `data_gathering.py` 文檔，第三行程式碼 `xxxx` 部分貼上複製的時間並儲存
8. 把在終端機 3 啟動的 `receive.py` 結束掉，輸入 `python data_gathering.py 0` (這邊 0 是標註 0 的意思，如下圖)

做完以後，電腦裡會出現這個檔案，裡面有三個欄位



第一個欄位(1,3,3,3,3)：統計每秒鐘產生 ICMP 封包的個數

第二個欄位(84,84,...)：平均封包大小

第三個欄位(0,0,0,0,0)：標記 0 是正常封包，如果出現 1 表示不正常

9. 終端機 3 再次啟動 `python receive.py`
10. 產生攻擊型封包，原始終端機輸入 `h1 hping3 -V -1 -d 1400 -fsat h2`，讓他跑到 300 以後終止，重複步驟 6&7
11. 終端機 3 的動作結束掉，輸入 `python data_gathering.py 1`

這時候就會產生 ICMP_data_class_1 檔案

ICMP_data_class_0.csv	45 bytes	16:58
ICMP_data_class_1.csv	60 bytes	16:59

把檔案打開，1 會長這樣

```
ICMP_data_class_1.csv (/home/user/p4-test/test-svm) - gedit
File Edit View Search Tools Documents Help
Open ▾ [icon]
data_gathering.py x ICMP_data_class_0.csv x ICMP_data_class_1.csv
30, 1428, 1
29, 1428, 1
28, 1428, 1
28, 1428, 1
29, 1428, 1
```

測試步驟：

1. 再開一個終端機 4，su，切資料夾到 test-svm
2. 然後 gedit controller.py &，程式碼複製貼上，再輸入 python controller.py
會出現像下圖的畫面，沒有灌資料會長這樣

```
root@ubuntu:/home/user/p4-test/test-svm# python controller.py
features= [[1.0, 84.0], [3.0, 84.0], [3.0, 84.0], [3.0, 84.0], [3.0, 84.0],
0, 1428.0], [29.0, 1428.0], [28.0, 1428.0], [28.0, 1428.0], [29.0, 1428.0]]
labels= [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]

** New entry **
ICMP info: 0 None

** New entry **
ICMP info: 0 None
```

接著在原始終端機(mininet)輸入 h1 ping h2，傳送正常封包

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.045 ms
^C
mininet: h1: icmp_seq=1 ttl=64 time=0.045 ms
mininet: h1: icmp_seq=2 ttl=64 time=0.045 ms
mininet: h1: icmp_seq=3 ttl=64 time=0.045 ms
mininet: h1: icmp_seq=4 ttl=64 time=0.045 ms
mininet: h1: icmp_seq=5 ttl=64 time=0.045 ms
mininet: h1: icmp_seq=6 ttl=64 time=0.045 ms
mininet: h1: icmp_seq=7 ttl=64 time=0.045 ms
mininet: h1: icmp_seq=8 ttl=64 time=0.045 ms
mininet: h1: PING: 0% packet loss, 8 rtt min=0.045 ms, max=0.045 ms, avg=0.045 ms
```

開始會出現 current prediction: 0，0 代表正常

若改用攻擊型封包，輸入 h1 hping3 -V -1 -d 1400 -fsat h2

```
** New entry **
ICMP info: 24 1428
Current prediction: 1
ring_the_alarm

** New entry **
ICMP info: 25 1428
Current prediction: 1
ring_the_alarm
```

出現 current prediction: 1，表示不正常

機器學習：

在座機器學習之前，會先經歷一個狀態叫 **tranning**，資料的訓練，這個訓練的過程主要目的是在建立一個模型，之後，當有新的資料進來，就會用建立好的模型進行預測，判斷它是正常還是攻擊的。

Controller.py

```
from p4utils.utils.topology import Topology
from p4utils.utils.sswitch_API import SimpleSwitchAPI
import influxdb, datetime, time, os, signal
from sklearn import svm
blockip=[]

class myController(object):

    def __init__(self):
        self.topo = Topology(db="topology.db")
        self.controllers = {}
        self.connect_to_switches()

    def connect_to_switches(self):
        for p4switch in self.topo.get_p4switches():
            thrift_port = self.topo.get_thrift_port(p4switch)
            self.controllers[p4switch] = SimpleSwitchAPI(thrift_port)

class gar_py:
    def __init__(self, db_host = 'localhost', port = 8086, db =
'mydb', kern_type = 'linear', dbg = False):
        self.debug = dbg
        self.host = db_host
        self.port = port
        self.dbname = db
        self.client = influxdb.InfluxDBClient(self.host, self.port, 'admin',
'telegraf', self.dbname)
        self.svm_inst = svm.SVC(kernel = kern_type)
        self.training_files = ["/ICMP_data_class_0.csv",
"/ICMP_data_class_1.csv"]
        self.query = """select count(length) as a,mean(length) as b from
net group by time(3s) order by time desc limit 3"""
        //我們會持續在資料庫執行這行指令，拿這幾筆資料去資料庫進行測
        self.train_svm()
        self.controller=myController()
```

這部分(**)是在做機器學習訓練用的

```
* def train_svm(self):
    features, labels = [], []

    for fname in self.training_files:
        meal = open(fname, "rt")
        for line in meal:
            data_list = line.rsplit(" ")
            for i in range(len(data_list)):
                if i < 2:
                    data_list[i] = float(data_list[i])
                else:
                    data_list[i] = int(data_list[i])
            features.append(data_list[:2])
            labels.append(data_list[2])
        meal.close()
    print "features=", features
    print "labels=", labels
    self.svm_inst.fit(features, labels)*
```

[]是特徵值，labels[]是結果。前面兩個是個數，後面是大小，對應到 labels[]的第一個 0 代表正常；第二格[3.0,84.0]對應到 labels[]的第二個也是 0，前五筆都對應到 0 代表這五筆是正常型的封包。後面的[30.0,1428.0]之後五筆對應的是 1

```
root@ubuntu:/home/user/p4-test/test-svm# python controller.py
features= [[1.0, 84.0], [3.0, 84.0], [3.0, 84.0], [3.0, 84.0], [3.0, 84.0], [30.0, 1428.0], [29.0, 1428.0], [28.0, 1428.0], [28.0, 1428.0], [29.0, 1428.0]]
labels= [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
```

這段(**)是，當模型建立好以後，有新的資料進來，要讓它去判斷什麼是正常，要讓它去判斷什麼是正常什麼是不正常的資料。它會去把資料庫的資料，它會去把資料庫拿新的資料去預測是不是正常的資料。

```
* def work_time(self):
    last_entry_time = "0"
    while True:
        for new_entry in
list(self.get_data(self.query).get_points(measurement = 'net')):
            if new_entry['time'] > last_entry_time:
```

```

        last_entry_time = new_entry['time']

        if self.debug:
            print("\n** New entry **\n\tICMP info: " +
                str(new_entry['a']) + " " + str(new_entry['b']))
            self.ring_the_alarm(self.under_attack(new_entry['a']
                ,new_entry['b']))
            time.sleep(3)*

    def under_attack(self, a, b):
        if b is None:
            return False
        if self.debug:
            print("\tCurrent prediction: " +
                str(self.svm_inst.predict([[a,b]])[0]))
            if self.svm_inst.predict([[a,b]])[0] == 1:
                return True
            else:
                return False

    def get_data(self, petition):
        return self.client.query(petition)

    def ring_the_alarm(self, should_i_ring):
        if should_i_ring:
            print("ring_the_alarm")
            #if "10.0.0.1" not in blockip:
                # self.controller.controllers["s1"].table_add("block_pkt",
"drop", [str("10.0.0.1")], [])
                # blockip.append("10.0.0.1")

    def ctrl_c_handler(s, f):
        print("\b\bShutting down MR. SVM... Bye!")
        exit(0)

if __name__ == "__main__":
    signal.signal(signal.SIGINT, ctrl_c_handler)
    ai_bot = gar_py(db_host = '127.0.0.1', dbg = True)

```



```
ai_bot.work_time()
```

data_gathering.py

```
import influxdb, sys
```

```
# Note xxxx will be replaced. I will explain later.
```

```
QUERY = """select count(length) as a,mean(length) as b from net where time <= xxxx  
group by time(3s) order by time desc limit 5"""
```

連到資料庫裏面，然後開啟一個檔案 `ICMP_data_class_{}` ，因為我們用 0，0 取代{}，所以檔名是 `ICMP_data_class_0`，產生.csv 檔

```
if __name__ == "__main__":
```

```
    db = influxdb.InfluxDBClient('127.0.0.1', 8086, 'admin', 'admin', 'mydb')
```

```
    measurement_class = sys.argv[1]
```

```
    out_file = open("ICMP_data_class_{}.csv".format(measurement_class), "w+")
```

```
    *for measurement in db.query(QUERY).get_points(measurement = 'net'):
```

```
        cnt = measurement["a"]
```

```
        meanlen = measurement["b"]
```

```
        out_file.write("{} {}, {} \n".format(cnt, meanlen, measurement_class))*
```

透過這樣(`**`)的方式，python 就會連到 `influxdb` 裡面把東西取出來，取出來的時候，有取出來 `cnt`，還有 `meanlan`(平均大小)，還有種類 (`measurement_class`)，然後會一筆一筆寫到檔案裡，就會產生上圖的 `ICMP_data_class_0` 檔

```
    out_file.close()
```

```
    print("Finished generating a class {} training
```

```
dataset!".format(measurement_class))
```

```
    exit(0)
```

Receive.py

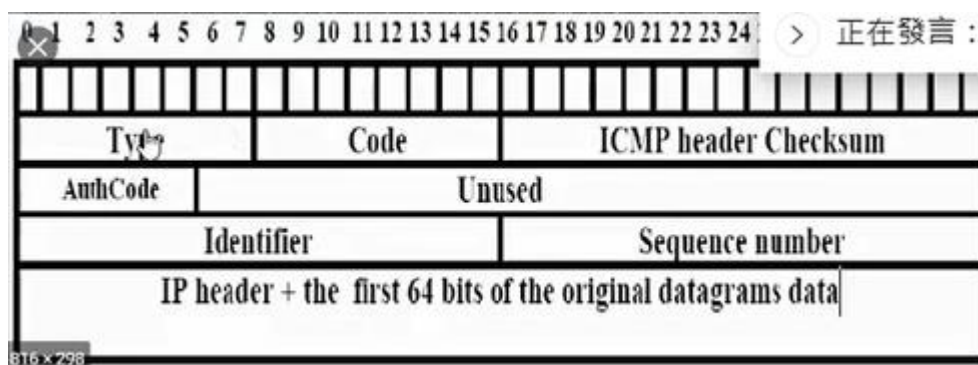
```
#!/usr/bin/env python
import sys
import struct
import os

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()
    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))
```

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



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():
    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()
```

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

myicmp.conf(logstash 的配置檔)

(這段底線到底線是輸入)

```
input {
    tcp {
        type => "tcp"
        port => 6666 //本地端開起 6666 埠
        mode => "server"
    }
}
filter{
    grok{
        match => ["message", "%{IP:srcIP} %{IP:destIP} %{INT:length}"]
    }
}
```

要從網路的 6666 埠進來

(這段底線到底線是輸出)

```
output {
    influxdb {
        db => "mydb"
        host => "localhost"
```

輸出到 influxdb 上


```

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}"
  "dstip"=>"%{destIP}"
  "length"=>"%{length}"
}
coerce_values => {
  "length" => "integer"
}
}
stdout { codec => rubydebug }
}

```

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

basic.p4

```

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

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;
    }
}
/*****
*****

***** CHECKSUM VERIFICATION *****
*****

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

***** INGRESS PROCESSING *****
*****

*****/
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();
    }
}

/*****
*****
***** EGRESS PROCESSING *****
*****
*****/
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);
        }
    }
}

/*****
*****
***** CHECKSUM COMPUTATION *****
*****
*****/
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
    }
  }
}
```


}
}