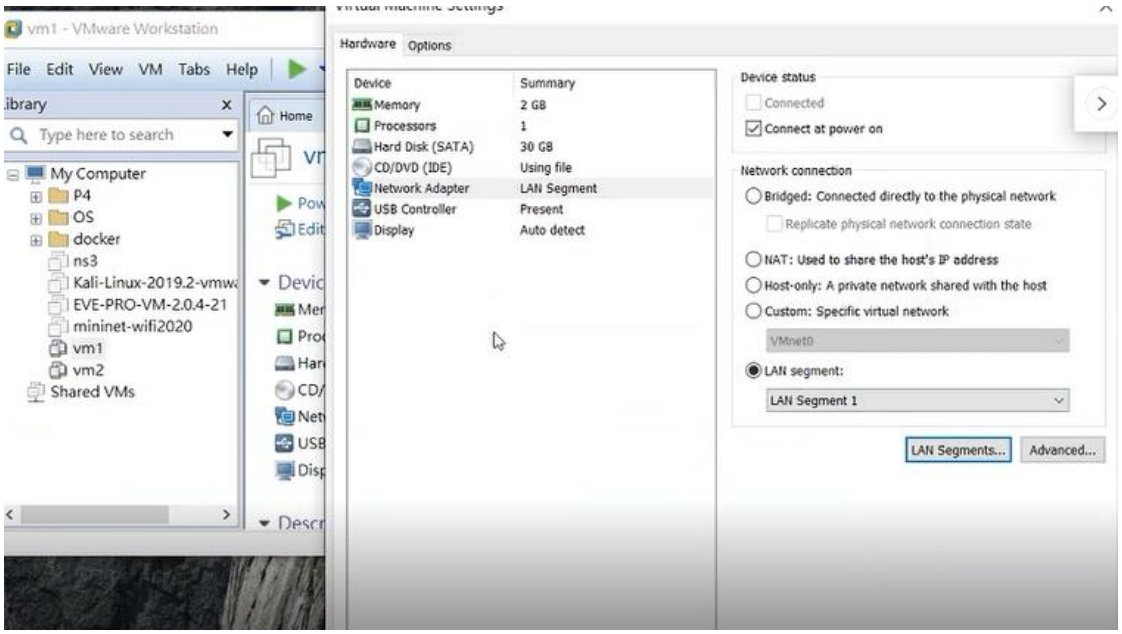
0630 如何做出 p4 軟體交換機

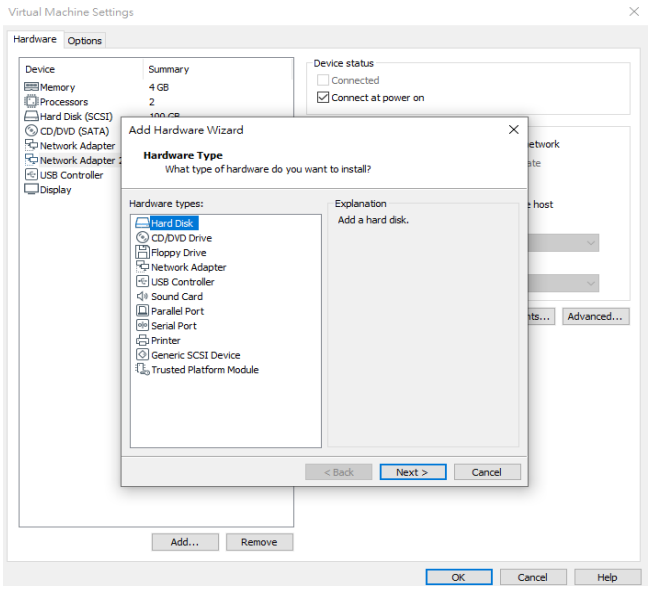
http://csie.nqu.edu.tw/smallko/sdn/p4switch.htm

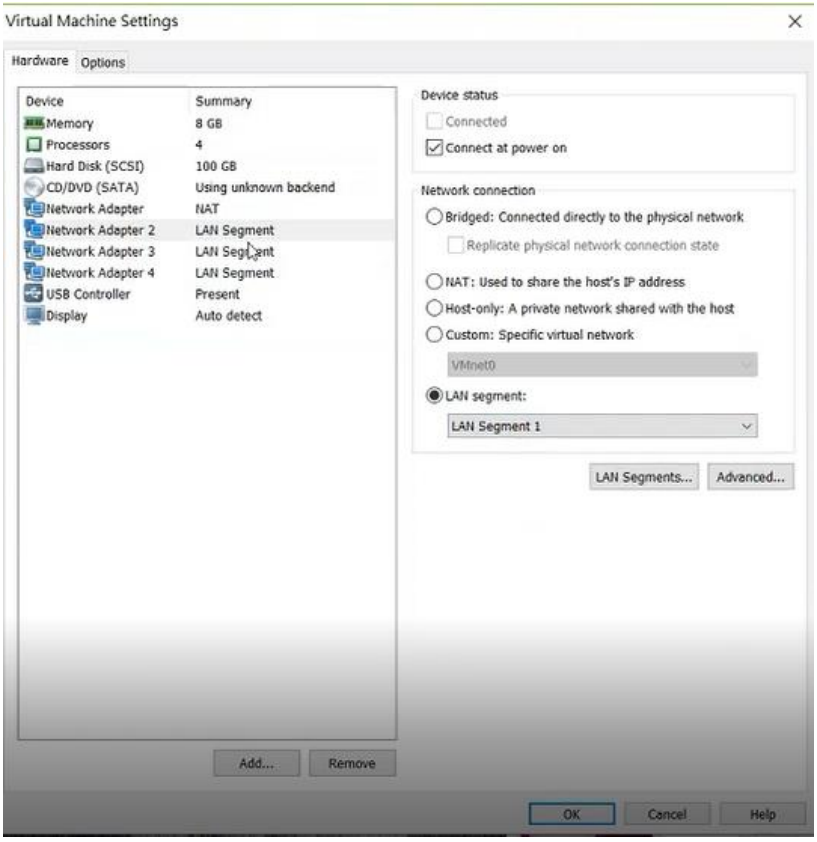
前置操作步驟：

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

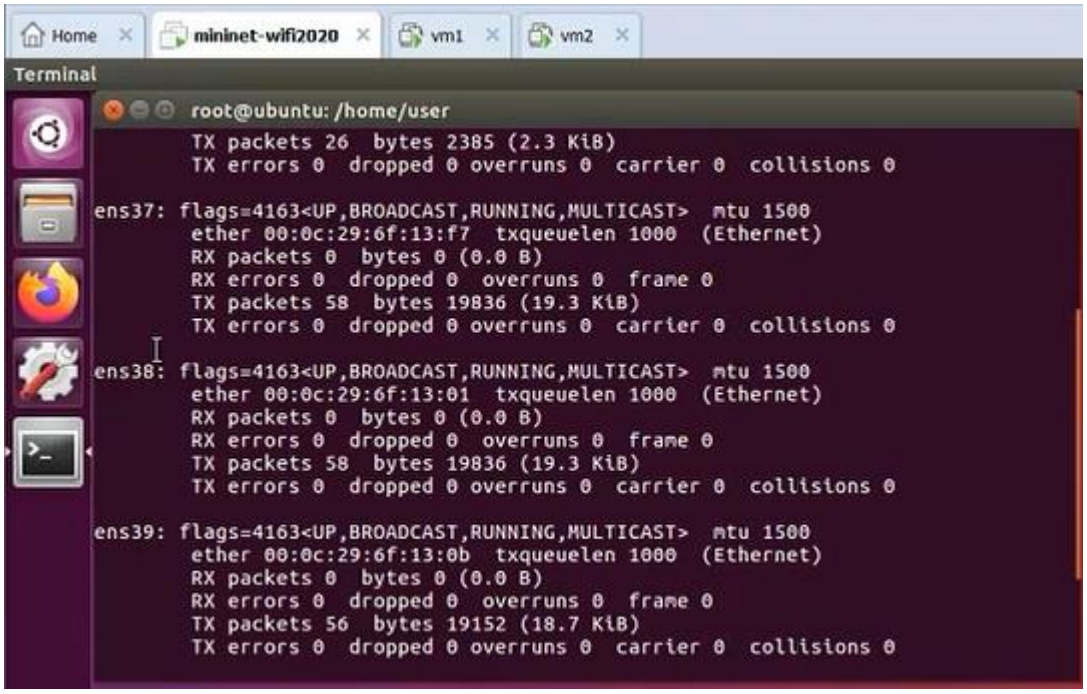




6. 弄好之後把三台機器打開(power on)

執行：

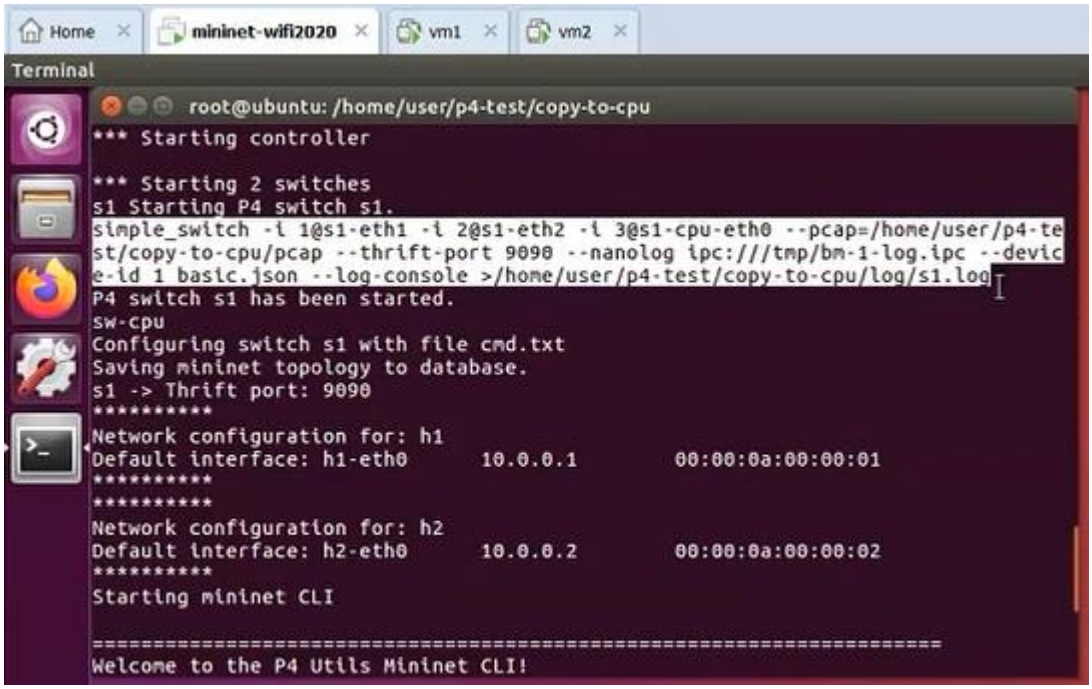
1.切到 mininet，打開終端機，切到超級使用者輸入 ifconfig，會看到



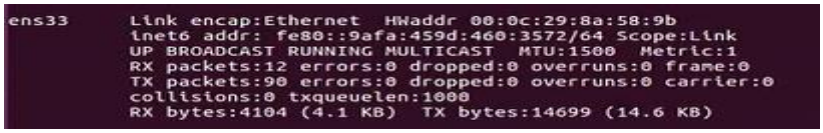
ens37,38,39(不一定是 37,38,39)就是剛剛增加的網路卡 LAN…1, LAN…2, LAN…3

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:

```
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
 --nanolog ipc:///tmp/bm-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 --nanol
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.p4i  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

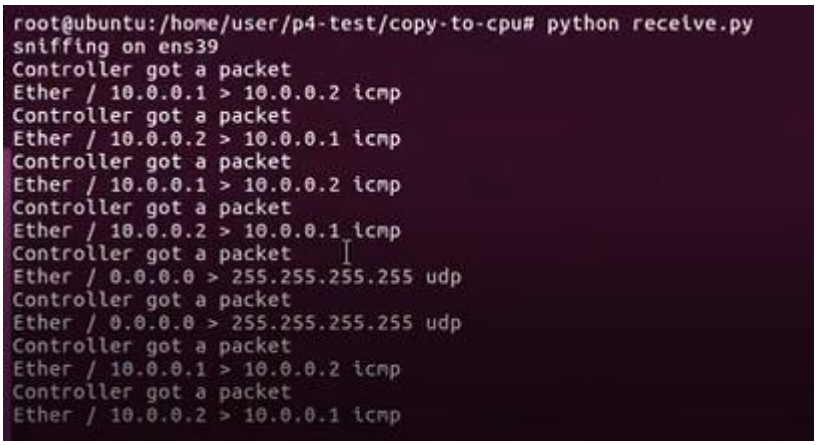
命令是 1 號埠進去 2 號埠出來，2 號埠進去 1 號埠出來

```
root@ubuntu: /home/user/p4-test/copy-to-cpu
[14:33:49.225] [bmv2] [D] [thread 3035] [6.0] [cxt 0] Pipeline 'ingress': end
[14:33:49.225] [bmv2] [D] [thread 3035] [6.0] [cxt 0] Egress port is 511
[14:33:49.225] [bmv2] [D] [thread 3035] [6.0] [cxt 0] Dropping packet at the end
 of ingress
[14:34:08.035] [bmv2] [T] [thread 3047] bm_get_config
[14:34:08.037] [bmv2] [T] [thread 3047] bm_table_add_entry
[14:34:08.037] [bmv2] [D] [thread 3047] Entry 0 added to table 'MyIngress.phy_fo
rward'
[14:34:08.037] [bmv2] [D] [thread 3047] Dumping entry 0
Match key:
* standard_metadata.ingress_port: EXACT     0001
Action entry: MyIngress.forward - 2,

[14:34:08.038] [bmv2] [T] [thread 3047] bm_table_add_entry
[14:34:08.038] [bmv2] [D] [thread 3047] Entry 1 added to table 'MyIngress.phy_fo
rward'
[14:34:08.038] [bmv2] [D] [thread 3047] Dumping entry 1
Match key:
* standard_metadata.ingress_port: EXACT     0002
Action entry: MyIngress.forward - 1,

[14:34:08.038] [bmv2] [T] [thread 3047] mirroring_sesssion_add
[14:34:08.039] [bmv2] [T] [thread 3047] mirroring_sesssion_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
#!/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



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

```
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

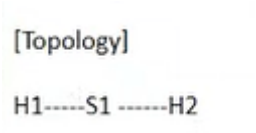
```
show database
```

```
> 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



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。

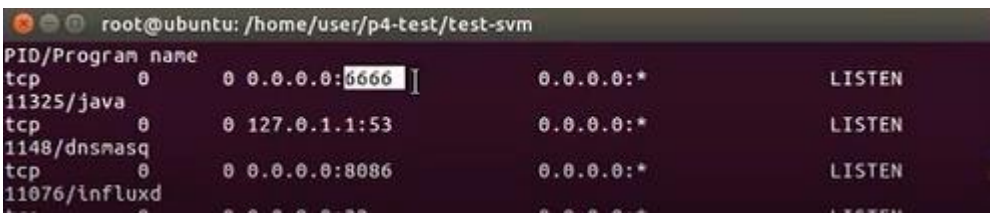
簡單來說，就是 receive.py 這支程式收到資料以後，會丟到 logstash(myicmp.conf)，logstash 收到以後，就會把對應的資料寫到資料庫裏面去。

操作步驟：

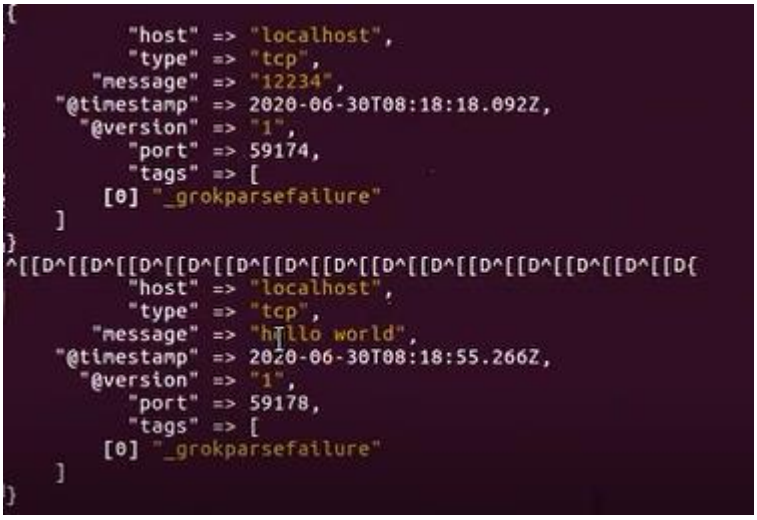
1. 打開 mininet 虛擬機，打開終端機，切到 p4-test 資料夾，建立新資料夾 (mkdir test-svm)
2. 切到 test-svm 資料夾，然後編輯(gedit)basic.p4,cmd.txt,p4app.json &把程式碼分別複製貼上並儲存
3. 接著 gedit receive.py &，myicmp.conf，一樣程式碼複製貼上
4. 開好幾個終端機，通通切到超級使用者
5. 終端機 1 執行 influxdb。(influx –username admin –password admin)，然後 show databases 可以看到有 mydb 資料庫
6.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

8. 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。

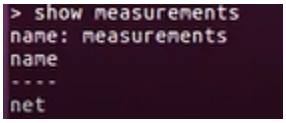


9. 試著輸入 echo “12334” | nc localhost 6666，終端機 2 就會出現訊息



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。

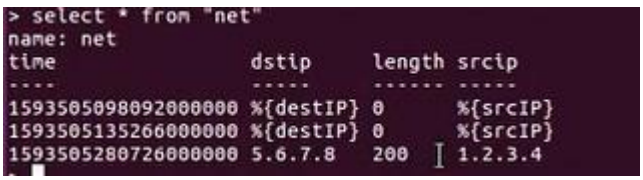
10. 再回到終端機 1，再一次輸入 show measurement 就會多一個 net。



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。

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}"]
  }
}
```



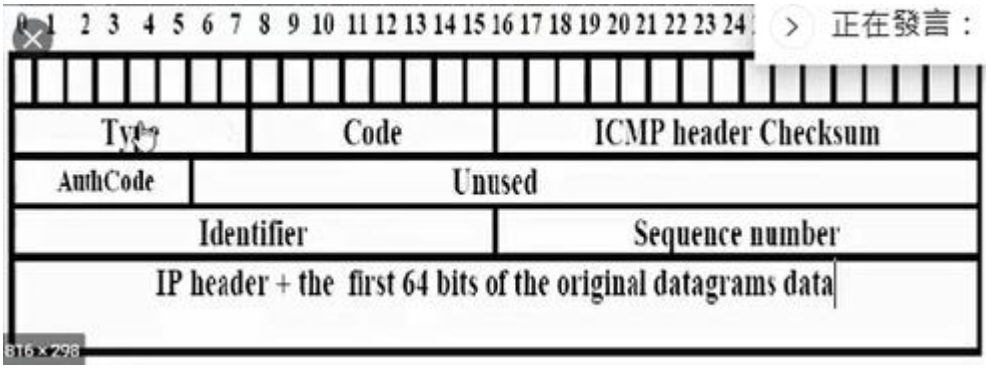
12.

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"
      port => "8086"
      user => "admin"
```

輸出到 influxdb 上

```
        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>
/*************************************************************************
********
********************** 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
        }
    }
}
```

```
}
```