動態加入規則

1.開終端機切到 p4-test 複製檔案：cp 1 –r 1-2 拷貝到 1-2 資料夾，切到 1-1(或 1-2)資料夾

2.輸入 gedit basic.json p4app.json & 編輯這兩個檔案 (p4app.json 用來描述環境拓樸)並儲存，然後 p4run

3.開兩個新的終端機，切到 1-1(或 1-2)資料夾，分別輸入：

simple_switch_CLI --thrift-port 9090 (connect to s1)

simple_switch_CLI --thrift-port 9091 (connect to s2)

4.在 s1，可以打 help 查看支持的命令是什麼 ex:help table_dump 檢視 match-table 的 entries(每筆紀錄)

5.table_dump phy_forward(phy_forward 是表格名稱)：顯示表格資訊

6.一開始的預設是 drop，因為還沒寫規則，h1 ping 不到 h2，所以要加入規則

7.使用 table_add，若不會用就使用 help table_add

指令：table_add phy_forward forward 1 => 2 (1:match field 2:action parameter)

table_add phy_forward forward 2 => 1 (是對稱的，1 進 2 出，2 進 1 出)

8.這樣就把兩筆紀錄加上去了，可以用 table_dump phy_forward 查看有沒有規則

9.在 s2 也執行

指令：table_add phy_forward forward 1 => 2

table_add phy_forward forward 2 => 1

10.在最開始的終端機(mininet 那個)輸入：h1 ping –c 3 h2)就可以 ping 了

Basic.json:

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

標頭檔(寫所有 p4 程式記得加

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

}

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

做資料處理的時候，需要把一些資料先暫存到暫時的變數，就可以在這裡面設定暫時性的變數(這裡沒有用到但留著)

這個範例不需要分析任何封包表頭，因為 1 號進來就從 2 號出去，2 號進來就從 1 號出去，封包裡有什麼東西不需要關心，完全沒有考慮到封包的結構，所以不需要 parse(沒有用到但留著)

封包進來，parser 不需要做任何動作，所以只需要做一件事：transition accept，代表分析已經結束

```
************  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() {                          //定義動作 1:drop，把他丟棄
            mark_to_drop(standard_metadata);
      }

      action forward(bit<9> port) {            //定義動作 2:forward
            standard_metadata.egress_spec = port;  //把想要轉發的埠號寫到
egress_spec，就會從 egress_spec 轉發出去
//當我把值(2)設定過去給它的時候，到時候那個封包就會從第幾號 port(2)送進
去
// metadata.egress 是 action parameters 要把他丟到?(2)好埠
      }

      table phy_forward {              //phy_forward：表格名稱(實體層_轉發)
            key = {                    //key：根據什麼去做匹配
                  standard_metadata.ingress_port: exact;
// (standard_metadata.ingress_port)是系統內建參數，這個參數是用來存放，記
錄這個封包是從哪個 port 號進來
```

如果有些情況之下，需要去檢查標頭裡面的 checksum，封包在傳輸過程中有沒有對跟錯(這裡沒有用到但要留著)

去匹配規則，做對應的事，這塊要從 apply 開始看

```
//exact:匹配方式有很多種，這種是完全匹配，要一模一樣，不能有不一樣的
// metadata.ingress 是 match fields，封包從哪裡進來，要符合這個規則
        }

        actions = {                 //匹配完要做的事
            forward;                //轉發
            drop;                   //丟棄
        }
        size = 1024;                //每條就是一個規則，數字就是這條最多可存放幾個(1024)規則
        default_action = drop(); //如果沒有指派，規則表裡沒寫的預設
    }

    apply {                         //真正的程式執行從這開始看
        phy_forward.apply();    //套用這個表格執行程式
    }
}

/*************************************************************************
*****
****************  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 {  }
}

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

在 p4 裡，規則是有一種結構存放的，叫 table，所以在設定規則的時候，要先準備一個 table，然後把規則還有要做的事情都放在這裡面，所以它是一個資料結構

Egress 剛開始沒有寫
把它留空

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

如果封包在處理過程當中，有可能需要重新計算checksum，錯誤檢查碼可以在這做(這個範例沒做，留空)

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

control MyDeparser(packet_out packet, in headers hdr) {
      apply {
      }
}
```

重組回來，因為沒有做解析，所以必須要重組

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

V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()

) main;
```

P4app.json

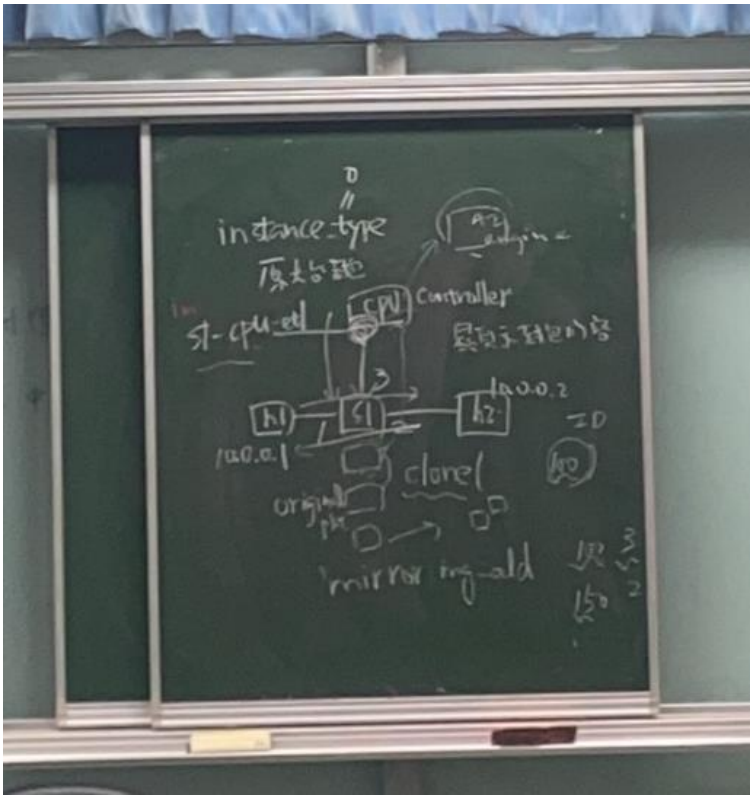
```
{
```

```
"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", //l2是layer2第二層，把h1跟h2放在同一個區域網路，分配ip的時候，他們是同個網域的ip位址
    "links": [["h1", "s1"], ["h2", "s2"], ["s1", "s2"]],
    "hosts": {
      "h1": {
      },
      "h2": {
      }
    },
    "switches": {
      "s1": {
        "program": "basic.p4"
      },
      "s2": {
```

```
            "program": "basic.p4"
          }
        }
      }
    }
}
```

Copy to cpu



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       //代表到時候會有多一個CPU的埠號，要送出去給CPU的地方
```

```
        }

    }

  }

}
```

Cmd.txt

```
table_add phy_forward forward 1 => 2
table_add phy_forward forward 2 => 1
mirroring_add 100 3            //拷貝完的送到 3 號埠  100 是 ID
```

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 {  }
}

//拷貝的時候可以在兩個地方做，一個是 ingress，或者是出去的時候做
/*************************************************************************
*****
```

```
************* 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,                    //在出口的地方做複製
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    apply {
        if (standard_metadata.instance_type == 0 ){
//.instance_type == 0 代表這是原始的封包
          clone(CloneType.E2E,100); //原始的封包做複製，100 就是給他一個
ID，E2E:E 就是在出口的部分，複製完他還是直接在出口的部分
        }
        if (standard_metadata.instance_type != 0 ){
          truncate(34);
        }
    }
}
```

clone(CloneType.E2E,100);拷貝的指令

原本的封包要去做一次複製，對 clone 的來講，要去進行截斷(truncate)的動作，只截 34byte，再傳出去，其他的都不要，也就是說，可以把部分資料丟上去就好

```
/*************************************************************************
*****
*************    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;
```

Receive.py

```
#!/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

class CpuHeader(Packet):
    name = 'CpuPacket'
    fields_desc = [BitField("device_id",0,16), BitField('reason',0,16),
BitField('counter', 0, 80)]
```

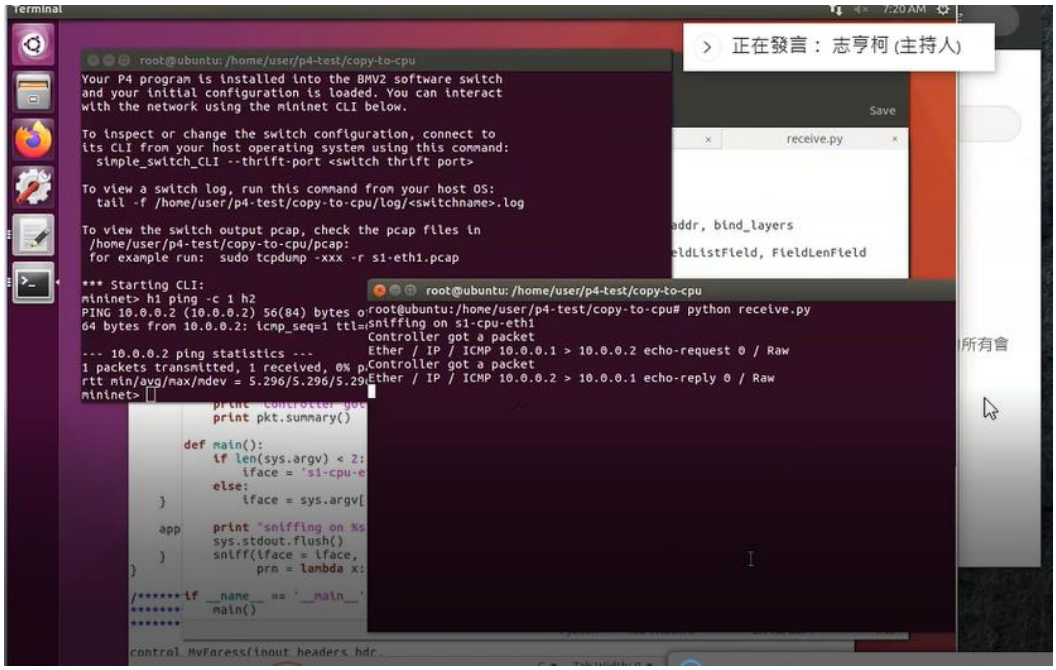
```
bind_layers(CpuHeader, Ether)

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

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__':      //程式的入口點
        main()                          //看到 main，呼叫 main 函式
```

print pkt.summary()
封包的一些部份資訊顯示出來，可以改成 print pkt.show()會出現比較完整的資訊

最重要的是這個：sniff(iface = iface,
prn = lambda x: handle_pkt(x))
他要做一件事情，他要在監聽端口 s1-cpu-eth1，做監聽的動作，聽到東西以後，就會把封包丟給 handle_pkt 這個函式去處理

執行：

1. 在一個終端機(a 左)輸入 p4run
2. 另一個終端機(b 右)輸入 python reveive.py
3. 在 a 輸入 h1 ping –c 1 h2
4. B 就會出現兩個封包



add_cmd.sh (在 1-1(或 1-2)資料夾)

規則很多的時候，這方法適合單一，例如只需要新增一筆規則

```
#!/bin/bash

//把前面結果輸出當作後面的輸入
echo ”table_add phy_forward forward 1 => 2” | simple_switch_CLI - -thrift-port 9090
echo ”table_add phy_forward forward 2 => 1” | simple_switch_CLI - -thrift-port 9090
simple_switch_CLI - -thrift-port 9091 < cmds2.txt
```

如果是比較大量，可以先把所有要做的規則寫到一個檔案(cmds2.txt)裡，再用導向(<)的方式丟到 s2(9091 port)

用./add_cmd.sh 就把規則匯進去了