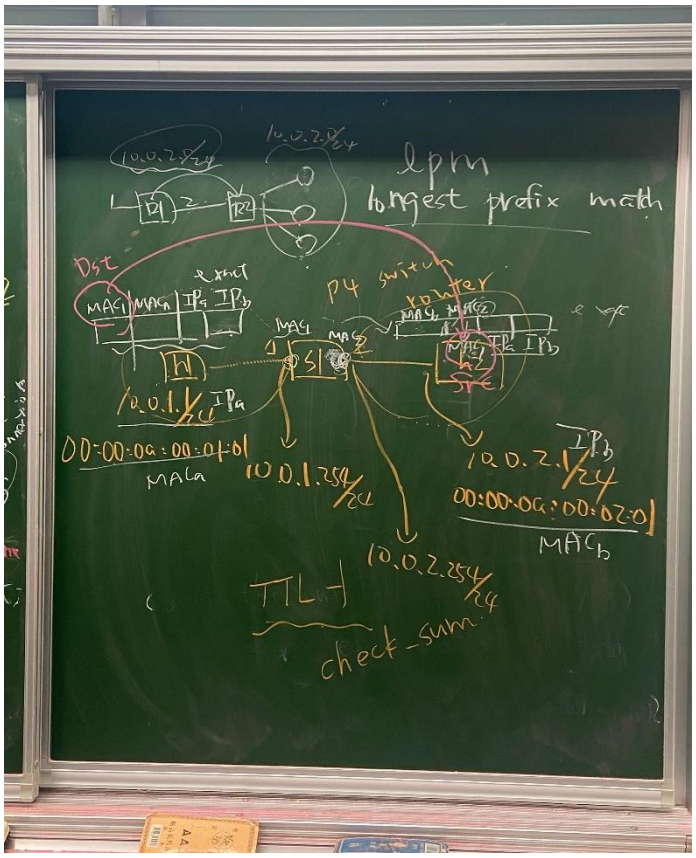
0601 根據 ip 進行轉發，把 p4 交換機模擬成路由器



封包進來(區域網路)以後，進行路由表查詢，查詢完以後就往目的端進行轉送，路由器每經過一跳，TTL 就會-1。


UNIX、Sun Solaris 及 OpenBSD 作業系統的 TTL值為 255
Compaq Tru64 5.0 及 LINUX 作業系統的 TTL值為 64
MS Windows NT/2K 作業系統的 TTL值為 128
MS Windows 95/98/ME 作業系統的 TTL值為 32

```
C:\>ping tw.yahoo.com

Ping atsv2-fp-shed.wg1.b.yahoo.com [124.108.103.103] (使用 32 位元組的資料):
回覆自 124.108.103.103: 位元組=32 時間=9ms TTL=51
回覆自 124.108.103.103: 位元組=32 時間=9ms TTL=51
回覆自 124.108.103.103: 位元組=32 時間=9ms TTL=51
回覆自 124.108.103.103: 位元組=32 時間=9ms TTL=51

124.108.103.103 的 Ping 統計資料:
    封包: 已傳送 = 4，已收到 = 4，已遺失 = 0 (0% 遺失)，
大約的來回時間 (毫秒):
    最小值 = 8ms，最大值 = 9ms，平均 = 8ms
```

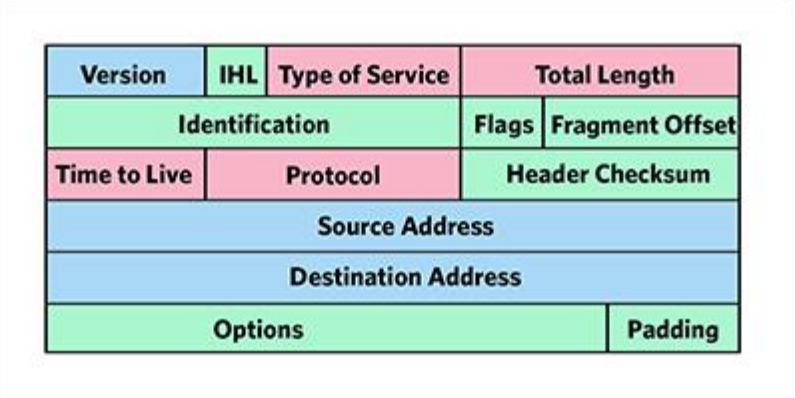
上圖中，下指令 ping 雅虎伺服器

「位元組」代表封包的大小，「時間」代表對方的反應時間，「TTL」就是封包的生存時間，當然你得到的這個就是剩餘的生存時間。
上圖看到 ping 雅虎伺服器返回的封包的 TTL值為51，表示雅虎伺服器是用 Linux系統架設的，而連到雅虎伺服器途中經過了 64-51=13個路由器。

TTL 的全名是 Time To Live，其值代表還有多少「生存時間」，其實就是還可以被轉發處理多少次。
每個路由器在轉發 ICMP 封包時，都會把 IP Header 的 TTL 值會減 1，如果 TTL 值已經到 0，就代表 TTL 已經到期，接著就會傳送錯誤訊息給原本發送的網路設備。

因為 TTL 有變動，所以在 ip 層(第三層)，有一個叫 check-sum(錯誤檢查碼)要重新計算。第一層實體層轉發或第二層 mac 層轉發 ip 標頭是不會動的，因為第二層在進行轉發的時候，基本上節點還是在同一個區域網路裡面，所以 ip 裡面的東西不會變，不需要重新計算 check-sum。

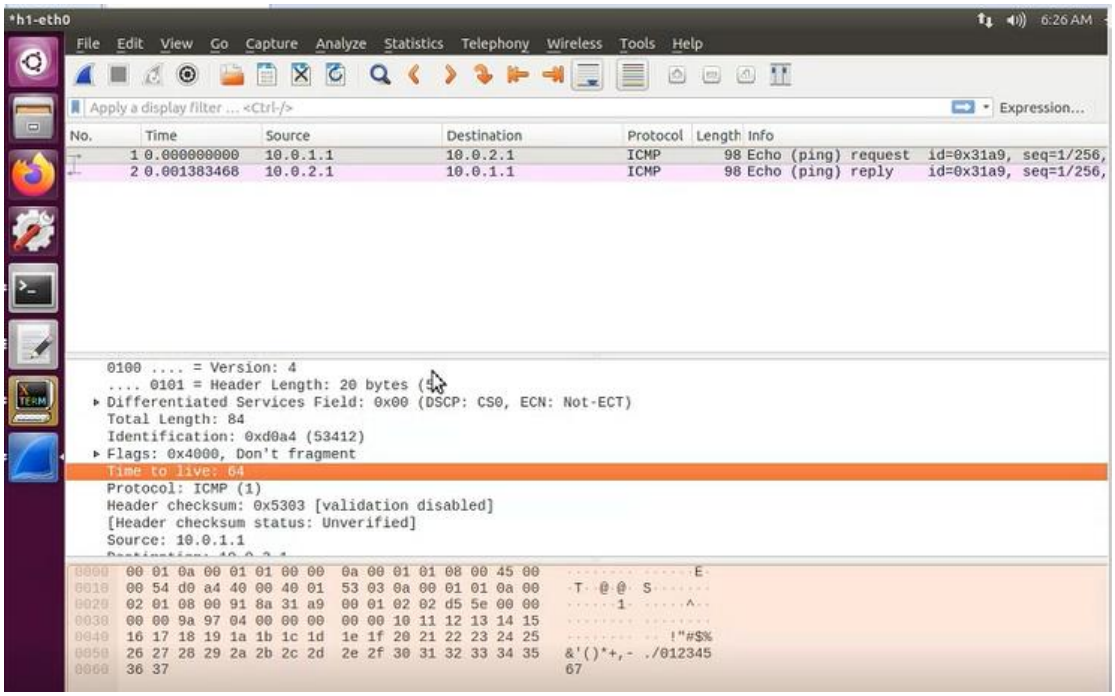
但是在第三層進行轉發的時候，已經跨到不同區域網路，跨路由器，ip 標頭的資訊就會變。



執行指令：

1. 打開終端機，切到資料夾 3，gedit ip_forward.p4 cmd.txt p4app.json &
2. p4run，xterm h1 h2，在 h1 h2 輸入 wireshark 打開它
3. wireshark 選 h2-eth0 然後點左上角藍色的魚鰭，h1 選 h1-eth0 點左上角藍色的魚鰭
4. 在 mininet 輸入 h1 ping –c 1 h2，ping 完以後看一下 wireshark h1-eth0，再看 h2-eth0

h1-eth0



10.0.1.1=h1/10.0.2.1=h2/TTL=64

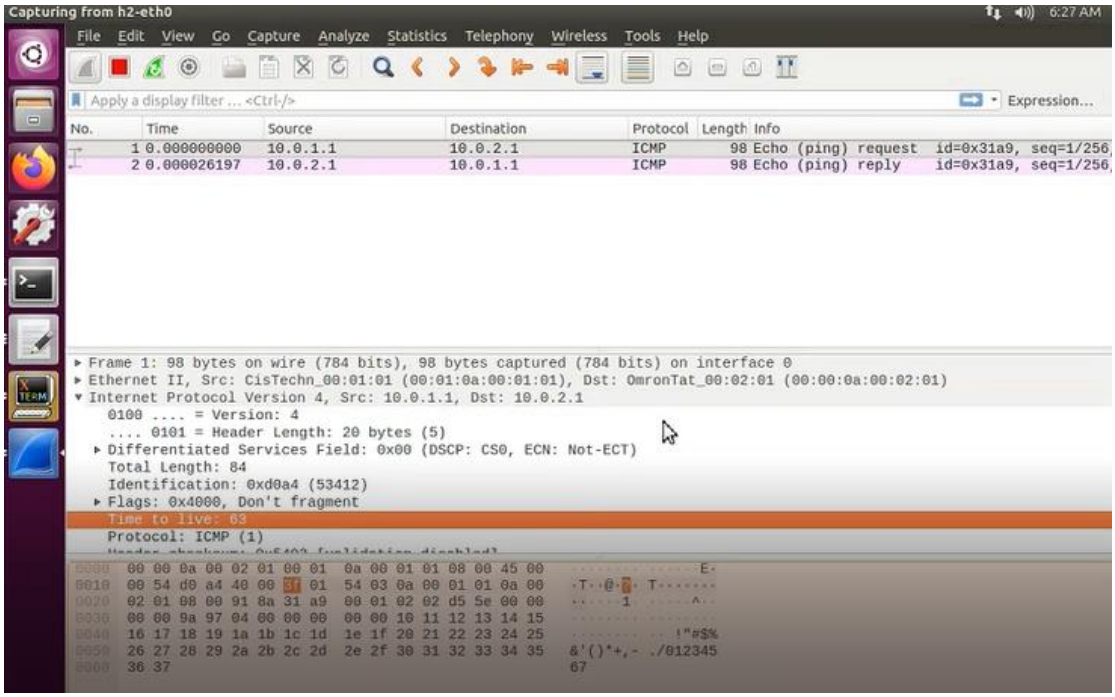
*若 header checksum 是 validation disabled 可以讓它強制判斷是對還是錯

步驟：

上排 edit -> 最下面的 preferences -> 左邊 protocols 展開 -> 選 ipv4 -> 把 validate the ipv4 那行打勾(預設是沒有打勾的)按 ok，checksum 就會變 correct

h2 也做一樣動作

h2-eth0



TTL=63

從 h1 送出來的時候是 64，經過了 s1，TTL-1 變成 63

lp_forward.p4

```
#include <core.p4>

#include <v1model.p4>

typedef bit<48> macAddr_t;

typedef bit<9> egressSpec_t;



header arp_t {
```

```
        bit<16> htype;
        bit<16> ptype;
        bit<8>    hlen;
        bit<8>    plen;
        bit<16> opcode;
        bit<48> hwSrcAddr;
        bit<32> protoSrcAddr;
        bit<48> hwDstAddr;
        bit<32> protoDstAddr;
    }

    header ethernet_t {
        bit<48> dstAddr;
        bit<48> srcAddr;
        bit<16> etherType;
    }

    header ipv4_t {
        bit<4>    version;
```

```
        bit<4>    ihl;
        bit<8>    diffserv;
        bit<16> totalLen;
        bit<16> identification;
        bit<3>    flags;
        bit<13> fragOffset;
        bit<8>    ttl;
        bit<8>    protocol;
        bit<16> hdrChecksum;
        bit<32> srcAddr;
        bit<32> dstAddr;
}

struct metadata {
}

struct headers {
        @name(".arp")
        arp_t           arp;
```

```
    @name(".ethernet")
    ethernet_t ethernet;
    @name(".ipv4")
    ipv4_t          ipv4;
}

parser ParserImpl(packet_in packet, out headers hdr, inout metadata meta, inout
standard_metadata_t standard_metadata) {
    @name(".parse_arp") state parse_arp {
        packet.extract(hdr.arp);
        transition accept;
    }
    @name(".parse_ethernet") state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            16w0x800: parse_ipv4;
            16w0x806: parse_arp;
            default: accept;
        }
    }
```

```
        @name(".parse_ipv4") state parse_ipv4 {
            packet.extract(hdr.ipv4);
            transition accept;
        }
        @name(".start") state start {
            transition parse_ethernet;
        }
}

control egress(inout headers hdr, inout metadata meta, inout standard_metadata_t
standard_metadata) {
        apply {
        }
}

control ingress(inout headers hdr, inout metadata meta, inout standard_metadata_t
standard_metadata) {
        @name(".set_nhop") action set_nhop(macAddr_t dstAddr, egressSpec_t port) {
            //set the src mac address as the previous dst, this is not correct right?
            hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
```

```
        //先把原本封包的目的的網路卡卡號改成它的來源



        //set the destination mac address that we got from the match in the table

        hdr.ethernet.dstAddr = dstAddr;



        //set the output port that we also get from the table

        standard_metadata.egress_spec = port;
        //輸出埠設定


        //decrease ttl by 1

        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
        //TTL-1
}

@name("._drop") action _drop() {

        mark_to_drop(standard_metadata);

}

@name(".ipv4_lpm") table ipv4_lpm {

        actions = {

                set_nhop;

                _drop;

        }
```

進行查詢，符合(lpm)的規則，就去執行 set_nhop
set_nhop:改變網路卡卡號

```
            key = {

                hdr.ipv4.dstAddr: lpm;          //(.dstAddr)判斷目的端的網路卡卡號
//(lpm)搭配網路遮罩處理，只要符合規則的，都往 x 埠號丟(行為一樣)，只要寫
這個網路的代表，就只要寫一條路由，就不用一條一條寫；如果用 exact，有
100 台主機就要寫 100 條路由

            }

            size = 512;

            const default_action = _drop();

        }

        apply {

            ipv4_lpm.apply();    //封包一進來，就會去執行 ipv4_lpm 這個表格
(@name(".ipv4_lpm") table ipv4_lpm )


        }

}

control DeparserImpl(packet_out packet, in headers hdr) {

        apply {

            packet.emit(hdr.ethernet);

            packet.emit(hdr.arp);

            packet.emit(hdr.ipv4);

        }

}
```

```
control verifyChecksum(inout headers hdr, inout metadata meta) {

    apply {

        verify_checksum(true, { hdr.ipv4.version, hdr.ipv4.ihl, hdr.ipv4.diffserv,
hdr.ipv4.totalLen, hdr.ipv4.identification, hdr.ipv4.flags, hdr.ipv4.fragOffset,
hdr.ipv4.ttl, hdr.ipv4.protocol, hdr.ipv4.srcAddr, hdr.ipv4.dstAddr },
hdr.ipv4.hdrChecksum, HashAlgorithm.csum16);

    }

}

control computeChecksum(inout headers hdr, inout metadata meta) {

    apply {

//check-sum 用下面的式子做重新計算
//這邊一定要呼叫，因為 TTL-1，如果沒有加，會發生封包無法傳送
        update_checksum(true, { hdr.ipv4.version, hdr.ipv4.ihl, hdr.ipv4.diffserv,
hdr.ipv4.totalLen, hdr.ipv4.identification, hdr.ipv4.flags, hdr.ipv4.fragOffset,
hdr.ipv4.ttl, hdr.ipv4.protocol, hdr.ipv4.srcAddr, hdr.ipv4.dstAddr },
hdr.ipv4.hdrChecksum, HashAlgorithm.csum16);
```

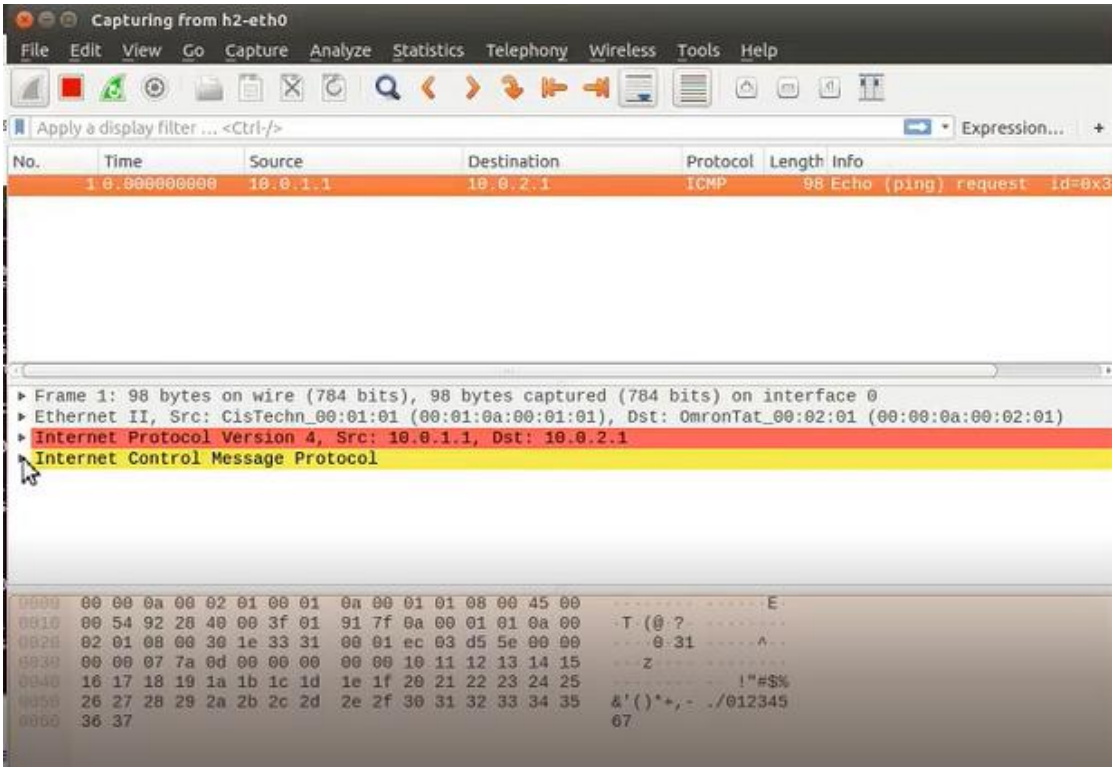
更新 ipv4 的 check-sum

更新的演算法，用 csum16 演算法，根據以上欄位的新的值，重新計算出 ipv4 的 check-sum

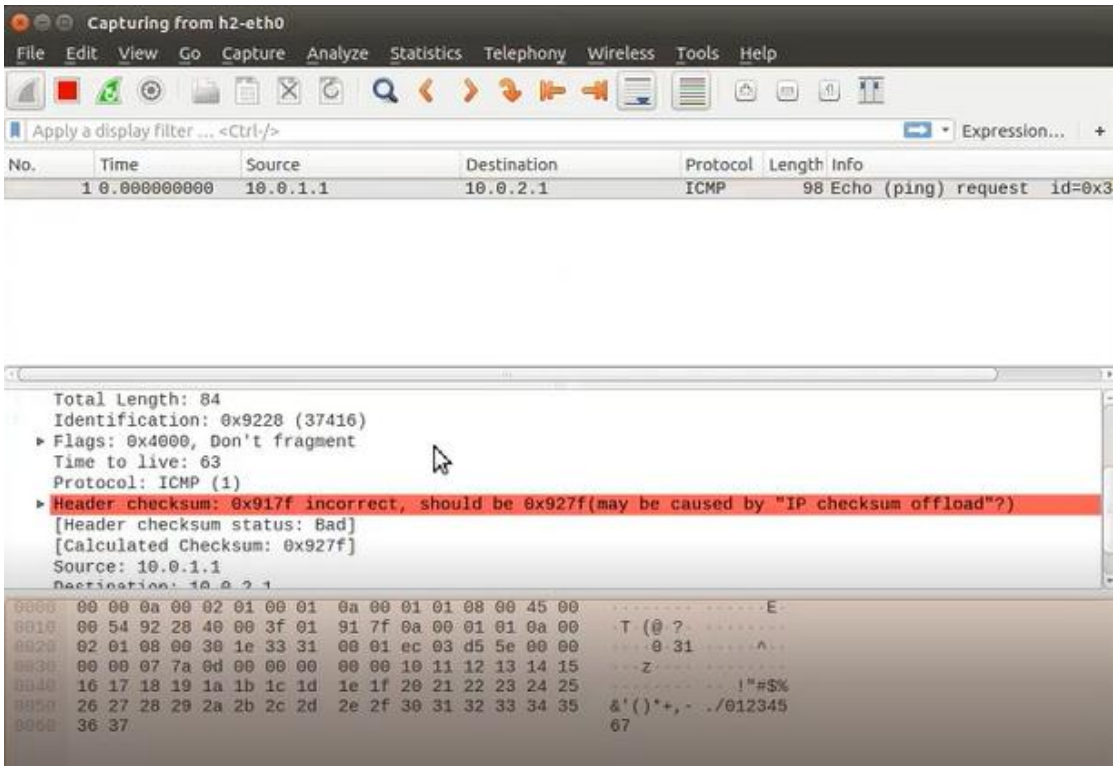
*若是把 checksum 拿掉，就會 ping 失敗，因為封包從 h1 送到 s1，s1 沒有重新計算 checksum 就到達 h2，封包的 mac 位址是可以的，所以它會一直收到 mac 層，然後再往 ip 層送，ip 層送的時候，會去檢查它的 checksum，發現 checksum 是錯的，就直接在網路層把東西丟掉，就不會往上送，所以沒有回應

```
    }

}


V1Switch(ParserImpl(), verifyChecksum(), ingress(), egress(), computeChecksum(),
DeparserImpl()) main;
```

封包進來了，卻發生錯誤？



原因： **checksum** 發生錯誤



P4app.json


```
{
    "program": "ip_forward.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": "manual",
    "auto_arp_tables": "true",
    "auto_gw_arp": "true",
    "links": [["h1", "s1"], ["h2", "s1"]],
    "hosts": {
        "h1": {
            "ip": "10.0.1.1/24",
            "gw": "10.0.1.254"
        },
        "h2": {
            "ip": "10.0.2.1/24",
```

```
            "gw": "10.0.2.254"
        }
    },
    "switches": {
        "s1": {
            "cli_input": "cmd.txt",
            "program": "ip_forward.p4"
        }
    }
  }
}
```

Cmd.txt

table_add ipv4_lpm set_nhop 10.0.1.1/32 => 00:00:0a:00:01:01 1

table_add ipv4_lpm set_nhop 10.0.2.1/32 => 00:00:0a:00:02:01 2

1-2 計數器

在 p4 裡面有一個東西叫 counter，它會幫你做計數的動作。例如說：h1 送封包到 s1 的時候，可以去計算說，封包從 1 號埠進來，進來了什麼封包，多少個 byte，從 2 號埠出去的時候，出去了多少個 byte，多少個 packet。

為什麼要有這些值？
因為有時候要去統計，這個網路的接口，收了多少封包，送出去多少封包，這些東西都可以用來做一個統計量，例如說：它使用的頻寬用了多少。

因為我們要處理封包進來跟出去，所以會有兩個 counter：
In 的 counter：計算封包進來交換機的封包數量
Out 的 counter：計算從這個埠送出去的封包的數量

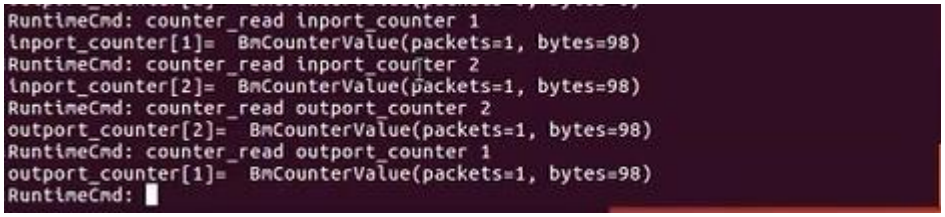
執行：

1. 打開終端機 a，切到 p4-test/1-2 資料夾
2. 輸入 gedit basic.p4 & 並 save
3. p4run，然後再開另一台終端機 b
4. 輸入 simple_switch_CLI - - thrift-port 9090 代表連接到交換機上
5. 輸入 counter_read inport_counter(名稱) 1(port)
6. 輸入 counter_read inport_counter(名稱) 2(port)
7. 輸入 counter_read outport_counter(名稱) 2(port)
8. 輸入 counter_read outport_counter(名稱) 1(port)

剛開始沒有任何封包在傳，所以 1 號埠進來的封包數量是 0，byte 數是 0
2 號埠進來的是 0，byte 也是 0，出去的 1,2 號埠也都是 0

9.切回終端機 a，輸入 h1 ping –c 1 h2
10.切回終端機 b，輸入 5-8 指令

一個 ping 出去，一個 icmp 回來，兩個封包



這個功能主要用來計算封包進來的量，還有出去的量，透過這些指令，就可以去統計，這個接口到底有多少封包的傳送

1 號埠有一個封包進來，大小是 98byte，2 號埠也是一個封包進來，大小是 98byte

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) {
```

```
//統計封包到底進來多少個，所以需要一個 in 的計數器
counter(512, CounterType.packets_and_bytes) inport_counter;
```

使用 counter 這個關鍵字定義 counter

因為不知道這個 p4 交換機有多少個埠，所以可以定義一個 512，或是大於交換機埠的數量

CounterType.packets_and_bytes 這個參數會統計同時有多少個封包有多少個 byte。
總共三種選擇，可以只統計 packet，也可以只統計 byte，也可以統計 packet 跟 byte

inport_counter 是變數名稱，可以自己取

```
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 {
    //怎麼計算：
    inport_counter.count 統計；ingress_port 記錄封包從哪個埠號進來
    inport_counter.count((bit<32>)standard_metadata.ingress_port);
```

當今天使用這樣的方式，它就會在對應的埠號，進行封包 +1，然後看看封包大小是多少就把 byte 數加上去

```
            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) {
    counter(512, CounterType.packets_and_bytes) outport_counter;
    //定義一個 counter，叫 outport_counter
    apply {
        outport_counter.count((bit<32>)standard_metadata.egress_port);
                                        //記錄他從哪個 pory 出去
    }
}

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

anti-tcp-port-scan 做一個防止 port-scan 的裝置

Port-scan：

當一個駭客，要去攻擊對方，首先一定要知道，對方的主機，那些埠號是打開的，如果知道哪些埠號是打開的，才能嘗試從那些埠號登入，才能入侵對方的主機。
所以 port-scan 最基本的用法就是，去查看某一台特定主機，哪些埠號現在是打開的。

如何做 port-scan？

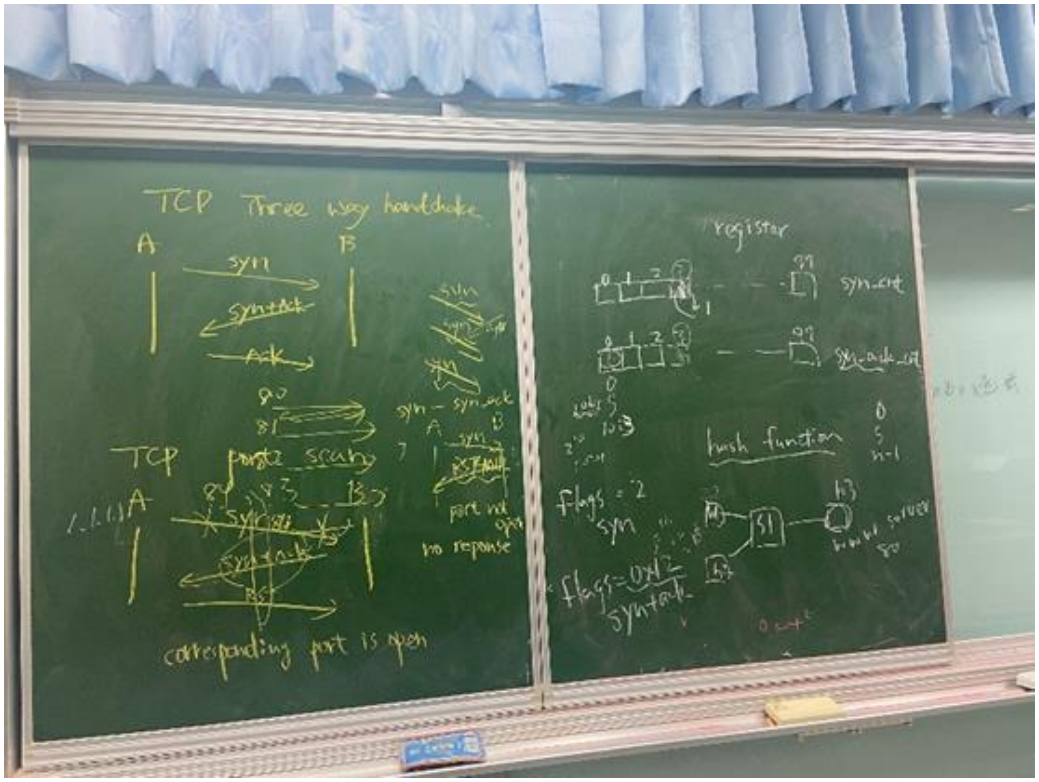
Tcp 要進行通訊之前，必須要先完成三項交握，必須要先由 syn 封包、syn+ack、ack 這樣的封包，才能完成三項交握，連線才能建立，建立完以後，他們之間才可以進行通訊。

如果要進行 port-scan，我就送一個 syn，到固定的 port 號，如果有回 syn 跟 ack，代表這個埠號有開，如果那個埠號沒有開，他會回一個 RST+ACK 封包回來。

所以可以用這樣去判斷，syn 跟 syn+ack 的量，正常來講應該是 1，因為一個封包對應一個 syn+ack，但是難免網路發生錯誤，所以有可能 syn 的量會比較多一點，但是可以去設定一個臨界值。

我去計算 syn 跟 syn+ack 的差值，只要差超過一定的量，就可以假設 A 在做 port-scan，因為一直在送 syn，syn 可能送 80.81.82.83.84 埠，然後等著 B 回我，但是如果送太多，對方沒有回，就代表他在做 port-scan，所以我們就可以去計算 syn 跟 syn+ack 的數量，只要他們兩個之間的數量，差超過一定的數值，就可以認為他在做 port-scan。在做 port-scan 的時候，就可以把 A 的 ip 給 block 住，也就是不要讓他再傳。

暫存器



暫存器的結構跟 counter 類似，它也像是一個 array，然後看需要多少元素

Hash 函數(雜湊函數)：做訊息摘要用。它可以把一個很大很大的數據，縮小成一個很短很短的數值。Ex:當我們在網路上要下載一個大的檔案，我們會擔心下載的對不對，所以通常有一些很大型的檔案在網路上會告訴你它的 md5 值是多少，下載完以後把這個檔案做雜湊函數，看得到的值是不是和網路上提供的相同，如果相同，代表下載過程當中檔案是一樣的，如果不一樣代表下載過程中發生錯誤。

例如說：A 的 ip 叫 1.1.1.1，透過雜湊函數得到它的值是 3，如果送了一個 syn 封包進來，原本裡面的值是 0，syn 對應的位置會+1，所以這個 3 代表是 A 的 ip 位址。基本的概念是，只要 syn 跟 syn+ack 的差值超過 3，就認為在做 port-scan，就把 A 的 ip block 住，不讓他送。

為什麼需要雜湊函數這個概念？

EX:假設我需要監控 A 這個人，實際上 A 就是一個 ip 位址，ip 總共有 32 個 bit，

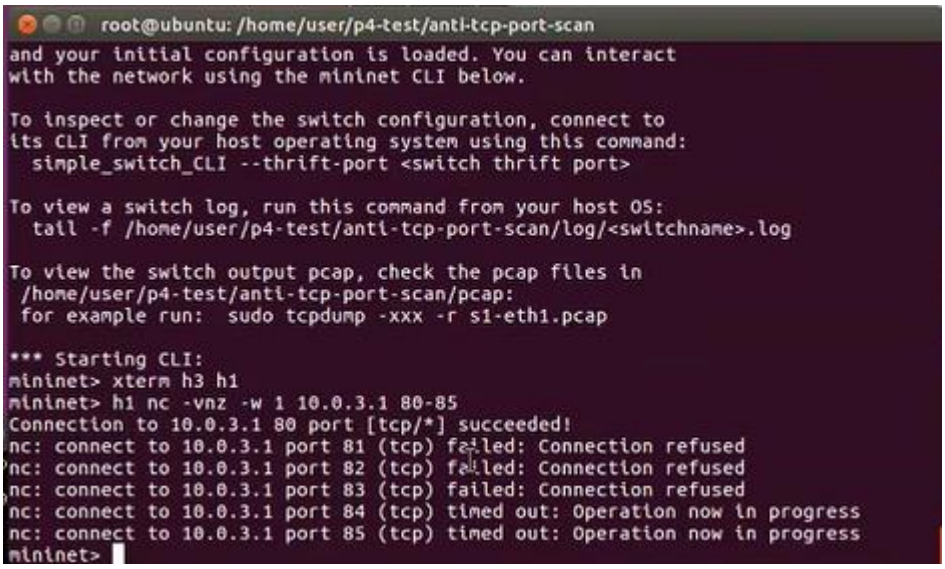
所以對一個來源 ip 來講，如果今天要用暫存器去代表每一個 ip，需要 2 的 32 次方的大小。但是以設備來講，記憶體空間是有限的，不可能放那麼大的資料量，所以這時候就需要雜湊函數，它可以把一個很大的值，透過運算縮小到一個範圍(0~n-1)裏面，這個值就可以用來代表它是從哪個來源來的。

執行步驟：

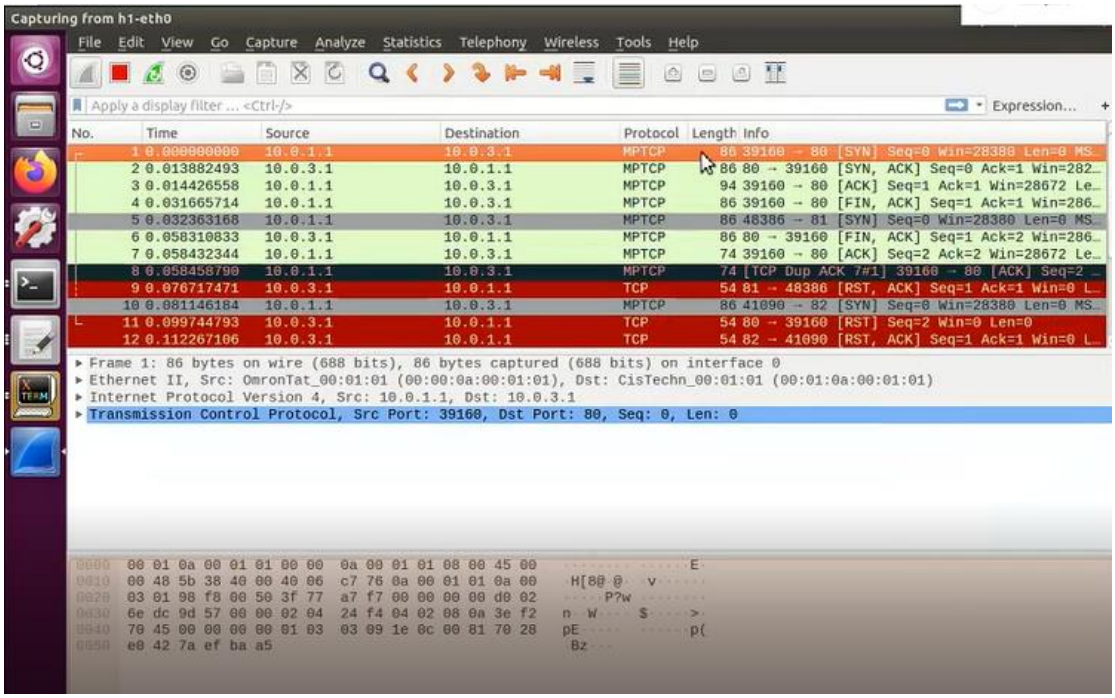
1. 打開終端機，cd anti-tcp-port-scan 資料夾
2. gedit basic.p4 p4app.json cmd.txt 把程式碼貼上
3. anti-tcp-port-scan 執行 p4run
4. mininet 執行 xterm h1 h3
5. 在 h3 執行 python -m SimpleHTTPServer 80
6. h1 執行 wireshark，把它打開，選擇 h1-eth0
7. mininet 執行 h1 nc –vnz –w 1 10.0.3.1 80-85

使用 python 內定的模組啟用 http 伺服器，伺服器開在 80 埠

nc=netcat 網路管理工具
vnz 是作封包偵測
-w 1 代表等待一秒鐘，若沒有回應表示連線失敗，80-85 代表要從第幾號埠觀測到第幾號埠



連到 80 是 succeeded 代表成功，因為剛剛開了一個 http 伺服器在 80 port
81,82,83 沒開所以失敗
84 是 timeout，因為 84 開始的時候差值就大於 3，於是他就把 syn 封包擋住

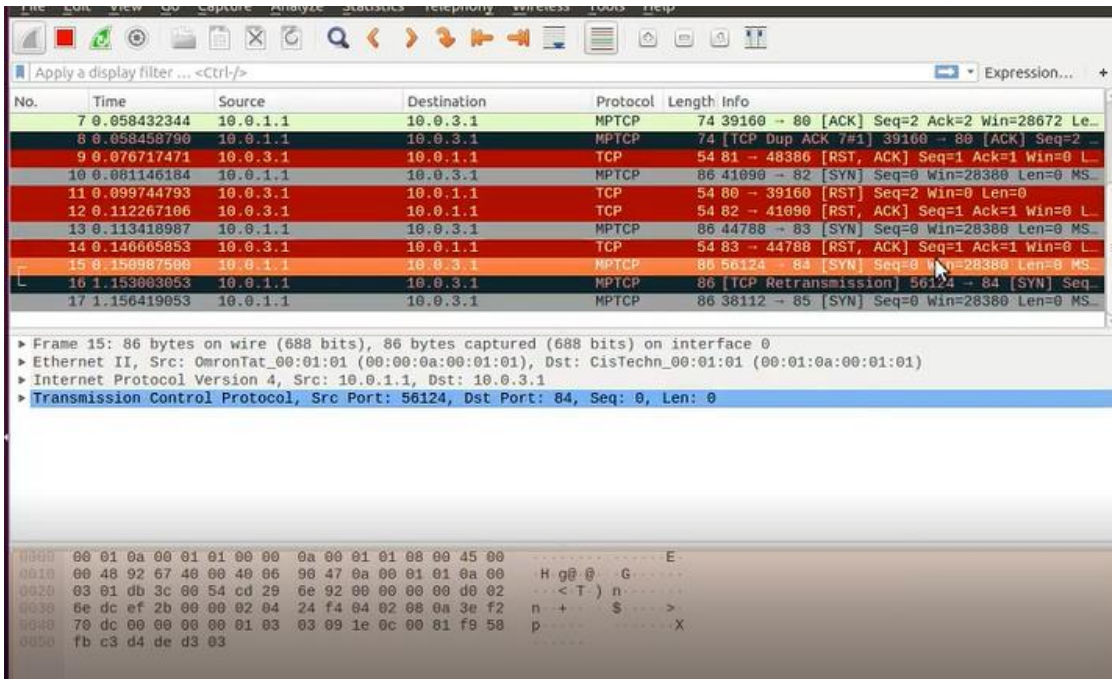


h1:10.0.0.1　h3:10.0.3.1

10.0.1.1 送到 10.0.3.1 是從 86 port 送到 80 port，然後對方會回 syn+ack，ack 完成三項交握，h1 就把連線關閉(fin)

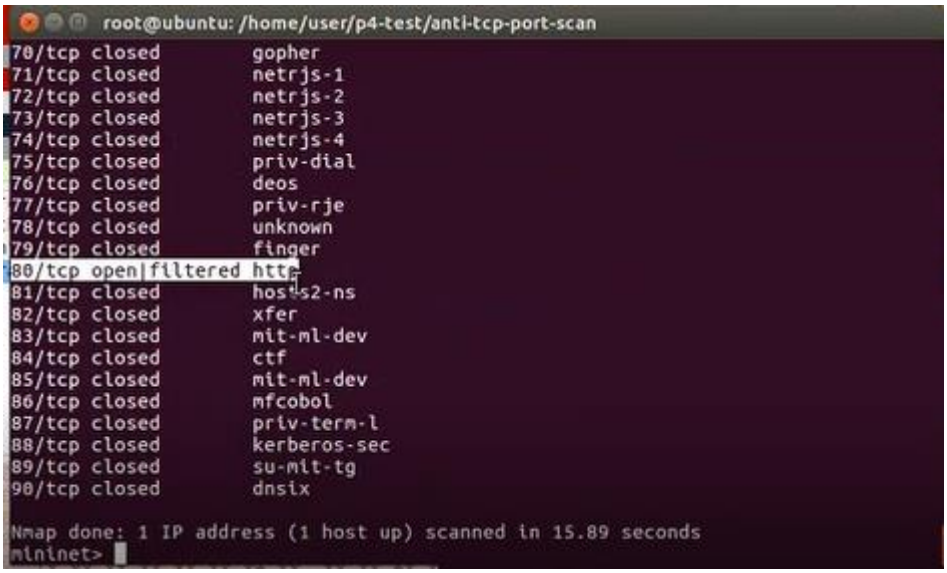
前面四個封包是去偵測 80 port 是不是 ok，如果是那封包行為就像圖上

但是開始從 81 port 送出去，回應的是 rst+ack 封包，82,83 也是都不回應，因為他們的 port 沒有開



然後 84 就只有 syn，沒有 rst，81,82,83 還是會放行，可是 84 就不放，所以後面的 h3 就不行，就被擋掉，再 ping 一次會完全不能 ping(包刮 80 port)

另一種方式是用 nmap：sudo apt-get install nmap
執行 nmap -sF -p 70-90 10.0.3.1
它一樣可以掃描(70-90port)，可是它就不能擋掉(因為 80 有被掃到被打開的)



basic.p4

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>
const bit<16> TYPE_IPV4 = 0x800;

/*************************************************************************
*********
********************** H E A D E R S
*********************************
*************************************************************************
********/
typedef bit<9>    egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;

register<bit<10>>(100) syn_cnt;
register<bit<10>>(100) syn_ack_cnt;
```

乙太網路表頭

定義兩個暫存器 register，大小是 100，<bit<10>>代表這裡面可以存放的值是 0~1023(2 的 10 次方=10 個 bit)，裡面每個值最小是 0，最大是 1023。總共有一百個。

兩個變數(暫存器名稱)：
syn_cnt
syn_ack_cnt

```
header ethernet_t {
        macAddr_t dstAddr;
        macAddr_t srcAddr;
        bit<16>      etherType;
}
```

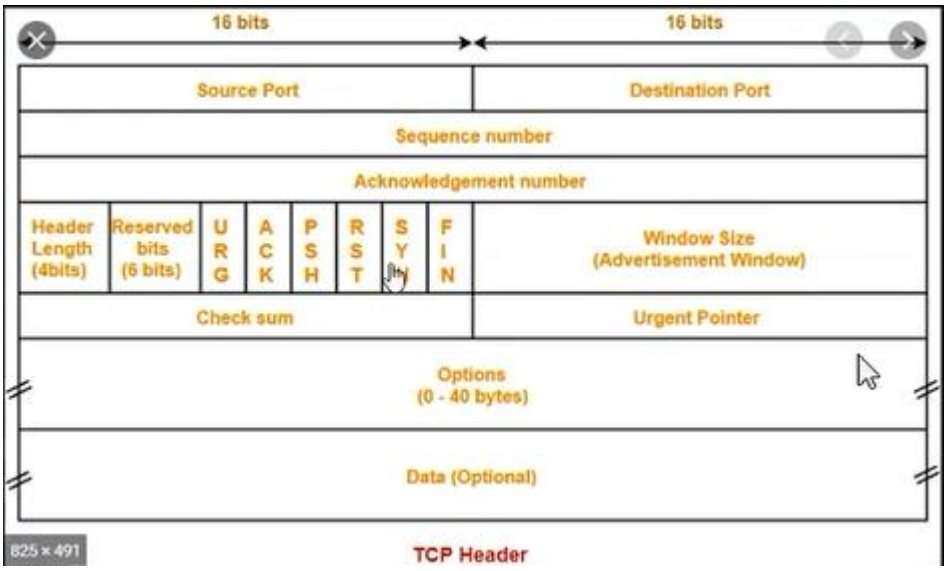
Ipv4 表頭

```
header ipv4_t {
        bit<4>       version;
        bit<4>       ihl;
        bit<8>       diffserv;
        bit<16>      totalLen;
        bit<16>      identification;
        bit<3>       flags;
        bit<13>      fragOffset;
        bit<8>       ttl;
        bit<8>       protocol;
        bit<16>      hdrChecksum;
        ip4Addr_t srcAddr;
        ip4Addr_t dstAddr;
}
```

tcp 表頭

```
header tcp_t {
        bit<16> srcPort;
        bit<16> dstPort;
        bit<32> seqNo;
        bit<32> ackNo;
        bit<4>    dataOffset;
        bit<4>    res;
        bit<8>    flags;
        bit<16> window;
        bit<16> checksum;
        bit<16> urgentPtr;
}
```

如果 flags 值=2，就代表它是一個 syn 的封包，它在第二個 bit，第一個 bit 是 fin
如果它值是 0x12 代表是 syn+ack 封包
如果是 01 代表是 fin 封包
所以可以透過它的值代表哪個欄位是有被設定的




```
header udp_t {
        bit<16> srcPort;
        bit<16> dstPort;
        bit<16> udplength;
        bit<16> checksum;
}

struct metadata {
        bit<10>     flowlet_map_index;
        bit<10>      syn_count;
        bit<10>      syn_ack_count;
}

struct headers {
        ethernet_t      ethernet;
        ipv4_t          ipv4;
        tcp_t           tcp;
        udp_t           udp;
}

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

需要一個變數 flowlet_map_index，做完雜湊函數，紀錄它到底是哪個位置，從來源 ip 位址轉換成這邊的 index，來源 ip 位址是一個比較大的空間，index 是比較小的空間，所以要記錄透過雜湊函數轉換完對應的 index 值

syn_count 要去看這裏面的值取出來是多少; syn_ack_count 看值是多少要放到臨時性變數裡面

```
********/
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.protocol) {
            0x06: parse_tcp;
            0x11: parse_udp;
            default: accept;
        }
    }

    state parse_tcp {
        packet.extract(hdr.tcp);
        transition accept;
    }

    state parse_udp {
        packet.extract(hdr.udp);
        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) {
```

透過這個函式，它會把來源 ip 雜湊到一個 0~99 的範圍存放到 meta.flowlet_map_index

```
    action add_syn_cnt() {
    //呼叫 hash function，hash 完的值存放到 meta.flowlet_map_index
crc16 是 hash 的其中一個演算法
0 指的是說 index 從哪裡開始，一般都寫 0 代表從 0 開始
//hash 來源 ip 位址(hdr.ipv4.srcAddr); 100 register 是 100 這邊就寫 100
        hash(meta.flowlet_map_index, HashAlgorithm.crc16, (bit<16>)0,
{ hdr.ipv4.srcAddr }, (bit<32>)100);
        //呼叫 syn_cnt.read 到對應的位置(lowlet_map_index)，把原本裡
面的值取出來放到 syn_count，剛開始是 0 取出來就是 0
        syn_cnt.read(meta.syn_count, (bit<32>)meta.flowlet_map_index);
        //必須做+1 的動作，做+1 之前要把裡面的值取出來，看是多少進
行+1 再寫回去，剛開始是 0 把它+1 就變成 1
        meta.syn_count=meta.syn_count+1;
        //再寫回去
        syn_cnt.write((bit<32>)meta.flowlet_map_index, meta.syn_count);
    }

    action add_syn_ack_cnt() {
```

```
            hash(meta.flowlet_map_index, HashAlgorithm.crc16, (bit<16>)0,
{ hdr.ipv4.dstAddr }, (bit<32>)100);
            syn_ack_cnt.read(meta.syn_ack_count,
(bit<32>)meta.flowlet_map_index);
            meta.syn_ack_count=meta.syn_ack_count+1;
            syn_ack_cnt.write((bit<32>)meta.flowlet_map_index,
meta.syn_ack_count);
        }

        action drop() {
            mark_to_drop(standard_metadata);
        }

        action forward(macAddr_t dstAddr, egressSpec_t port) {
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
            hdr.ethernet.dstAddr = dstAddr;
            standard_metadata.egress_spec = port;
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
        }

        table ip_forward {
            key = {
                hdr.ipv4.dstAddr: exact;
            }
            actions = {
                forward;
                drop;
            }
            size = 1024;
            default_action = drop();
        }

        apply {
            bit<1> set_drop=0;
            if (hdr.tcp.isValid()){
            //看看是不是 syn 封包
                if(hdr.tcp.flags==2) {
```

因為這封包是要回去，是回去的方向，所以是 destination

宣告一個變數叫 set_drop=0，代表這個封包預設要不要被丟棄，剛開始預設是 0 代表不丟棄

```
                //確認是 syn 的封包後，要到對應的位置+1，統計數量要執行 syn_cnt 這個函式
                    add_syn_cnt();
```

tcp.isValid 看 tcp 表頭是不是有效的，有可能它不是 tcp 的封包，不是就不管它，直接做轉發的動作(if(hdr.ipv4.isValid() && set_drop==0){
ip_forward.apply();
) 預設值也是 0 就開始進行轉發

```
    //如果是一個 syn 封包進來，除了+1 以外，先去判斷差值有沒有大於 3
    hash(meta.flowlet_map_index, HashAlgorithm.crc16, (bit<16>)0,
    { hdr.ipv4.srcAddr }, (bit<32>)100);
            bit<10> tmp;        //臨時性變數 tmp
                //把目前的值取出來
                syn_ack_cnt.read(tmp, (bit<32>)meta.flowlet_map_index);
            if(tmp==0 && meta.syn_count>3){
                set_drop=1;
                }
            //這兩個值相減看看有沒有大於 3，目前的 count 值跟 tmp+3 是不是有大於，如果大於就設定成 drop，封包就不會再放過去，被 p4 擋掉
                if (tmp!=0 && meta.syn_count > (bit<10>)(3+tmp)){
                    set_drop=1;
                    }
                } else if (hdr.tcp.flags==0x12) {
                    add_syn_ack_cnt();
                }
            }

            if( hdr.ipv4.isValid() && set_drop==0){
                    ip_forward.apply();
            }
        }
    }

    /*************************************************************************
    *********
    ***************  E G R E S S    P R O C E S S I N G
    ******************
    *************************************************************************
```

如果是 0x12 就做 add_syn_ack_cnt

```
********/
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
**************
*************************************************************
********/
control MyComputeChecksum(inout headers  hdr, inout metadata meta) {
     apply {
         update_checksum(
             hdr.ipv4.isValid(),
             { hdr.ipv4.version,
               hdr.ipv4.ihl,
               hdr.ipv4.diffserv,
               hdr.ipv4.totalLen,
               hdr.ipv4.identification,
               hdr.ipv4.flags,
               hdr.ipv4.fragOffset,
               hdr.ipv4.ttl,
               hdr.ipv4.protocol,
               hdr.ipv4.srcAddr,
               hdr.ipv4.dstAddr },
             hdr.ipv4.hdrChecksum,
             HashAlgorithm.csum16);
    }
}

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

```
********/
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
        packet.emit(hdr.tcp);
        packet.emit(hdr.udp);
    }
}

/*************************************************************************
*********
***********************  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": "manual",
    "auto_arp_tables": "true",
    "auto_gw_arp": "true",
    "links": [["h1", "s1"], ["h2", "s1"],["h3", "s1"]],
    "hosts": {
        "h1": {
    "ip": "10.0.1.1/24",
    "gw": "10.0.1.254"
        },
        "h2": {
    "ip": "10.0.2.1/24",
    "gw": "10.0.2.254"
        },
        "h3": {
    "ip": "10.0.3.1/24",
    "gw": "10.0.3.254"
        }
    },
    "switches": {
        "s1": {
            "cli_input": "cmd.txt",
            "program": "basic.p4"
        }
    }
```

```
        }
    }
```

cmd.txt

```
table_add ip_forward forward 10.0.1.1 => 00:00:0a:00:01:01 1
table_add ip_forward forward 10.0.2.1 => 00:00:0a:00:02:01 2
table_add ip_forward forward 10.0.3.1 => 00:00:0a:00:03:01 3
```