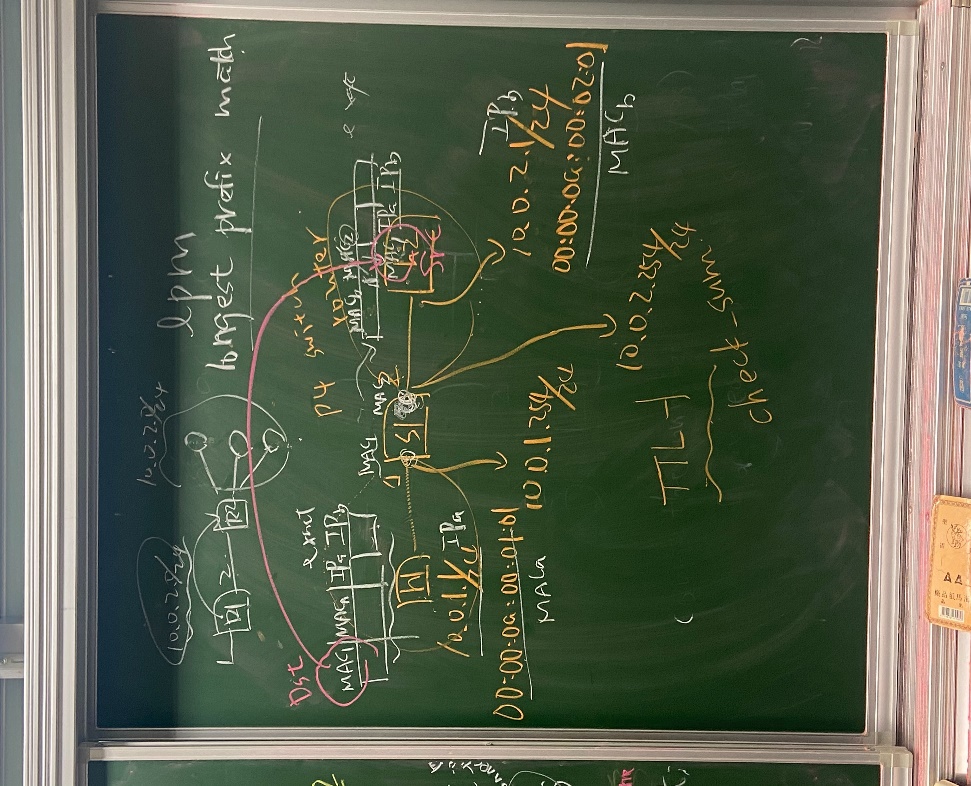
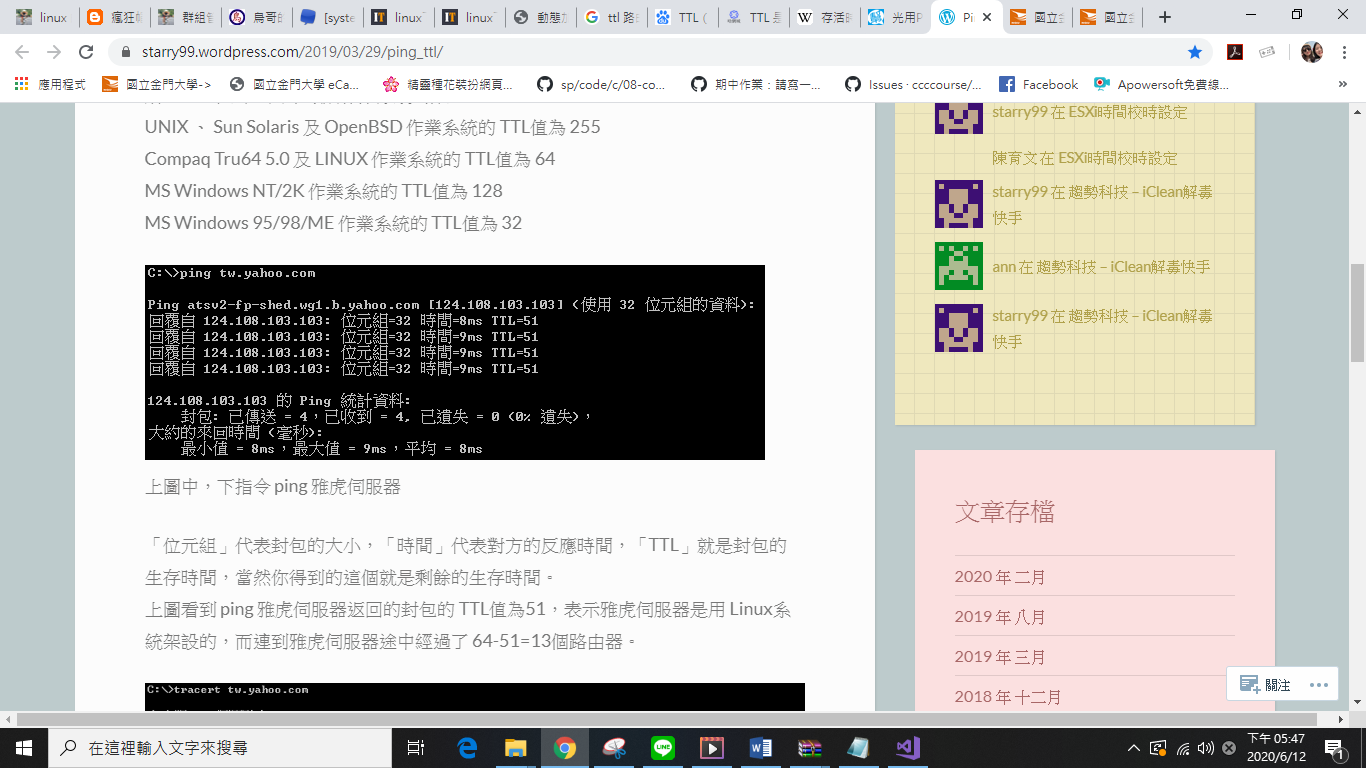
0601 根據ip進行轉發，把p4交換機模擬成路由器

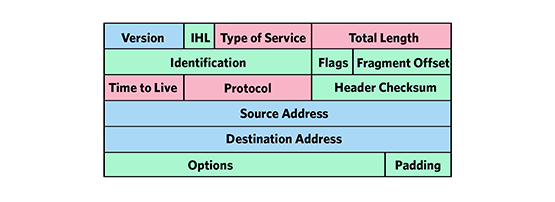


封包進來(區域網路)以後，進行路由表查詢，查詢完以後就往目的端進行轉送，路由器每經過一跳，TTL就會-1。



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。

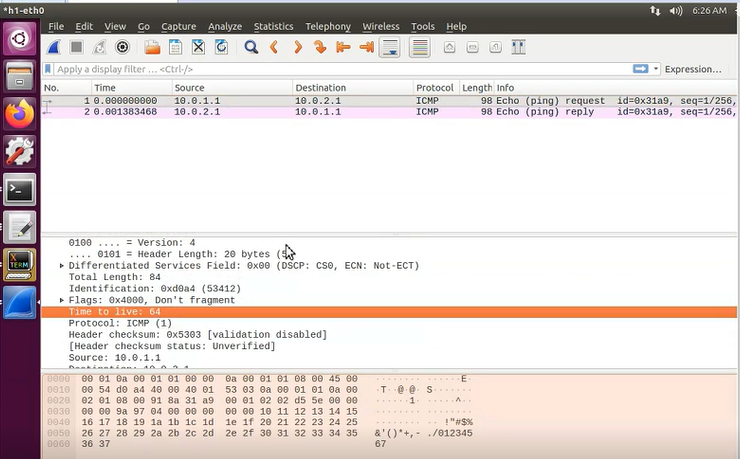
因為TTL有變動，所以在ip層(第三層)，有一個叫check-sum(錯誤檢查碼)要重新計算。第一層實體層轉發或第二層mac層轉發ip標頭是不會動的，因為第二層在進行轉發的時候，基本上節點還是在同一個區域網路裡面，所以ip裡面的東西不會變，不需要重新計算check-sum。

****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

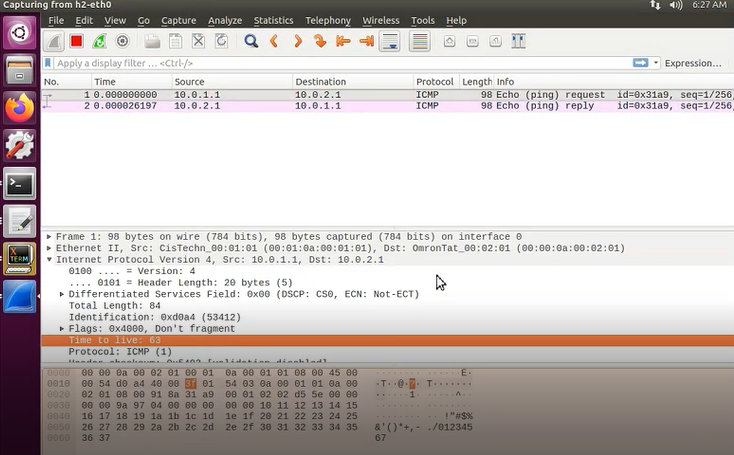
\*若header checksum是validation disabled可以讓它強制判斷是對還是錯

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

**Ip\_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 = {

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

set\_nhop;

\_drop;

}

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

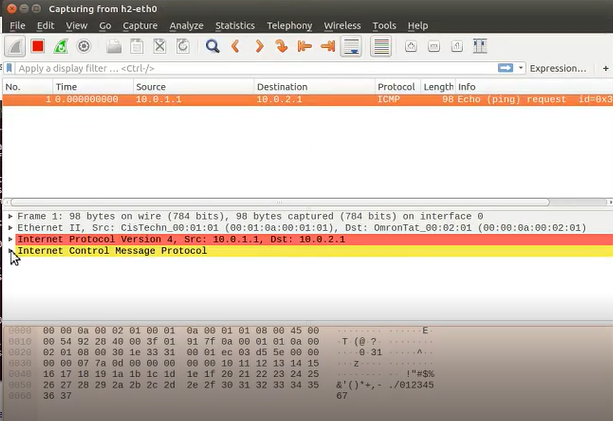
更新ipv4的check-sum

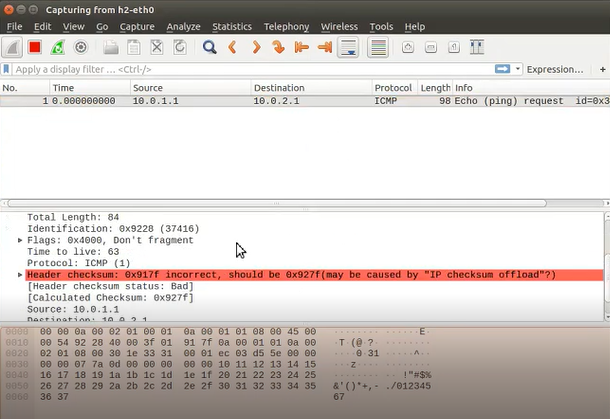
\*若是把checksum拿掉，就會ping失敗，因為封包從h1送到s1，s1沒有重新計算checksum就到達h2，封包的mac位址是可以的，所以它會一直收到mac層，然後再往ip層送，ip層送的時候，會去檢查它的checksum，發現checksum是錯的，就直接在網路層把東西丟掉，就不會往上送，所以沒有回應

}

}

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)



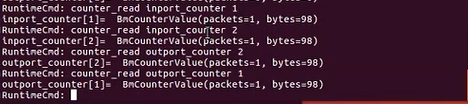
剛開始沒有任何封包在傳，所以1號埠進來的封包數量是0，byte數是0

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

一個ping出去，一個icmp回來，兩個封包

10.切回終端機b，輸入5-8指令



*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

inport\_counter是變數名稱，可以自己取

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

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？

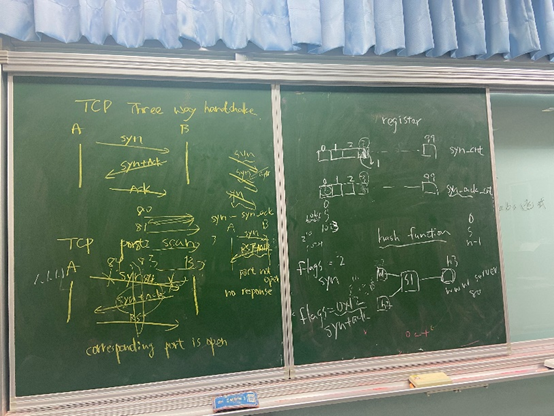
Tcp要進行通訊之前，必須要先完成三項交握，必須要先由syn封包、syn+ack、ack這樣的封包，才能完成三項交握，連線才能建立，建立完以後，他們之間才可以進行通訊。

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：

使用python內定的模組啟用http伺服器，伺服器開在80埠

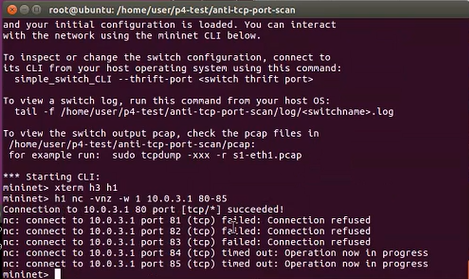
1. 打開終端機，cd anti-tcp-port-scan資料夾
2. gedit basic.p4 p4app.json cmd.txt 把程式碼貼上
3. anti-tcp-port-scan執行p4run
4. mininet執行xterm h1 h3

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

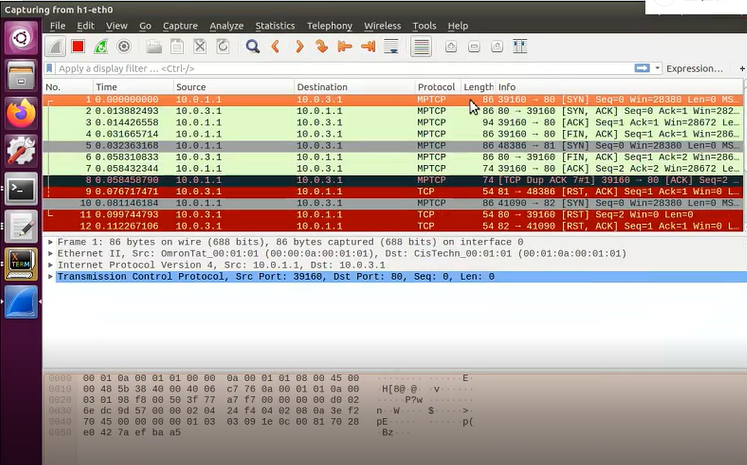
1. 在h3執行python -m SimpleHTTPServer 80
2. h1執行wireshark，把它打開，選擇h1-eth0
3. mininet執行h1 nc –vnz –w 1 10.0.3.1 80-85



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

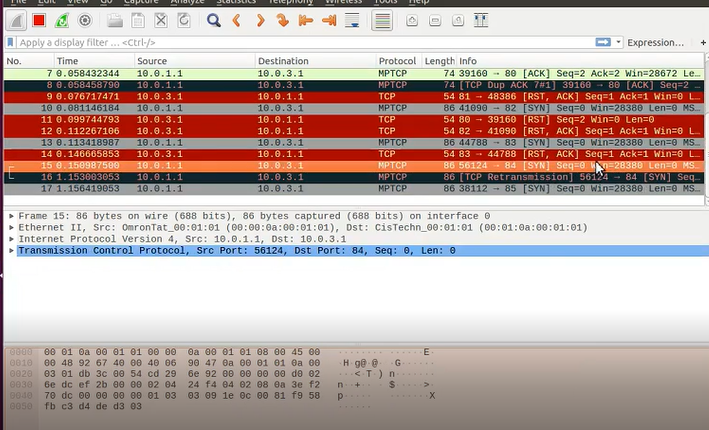


**h1:10.0.0.1 h3:10.0.3.1**

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

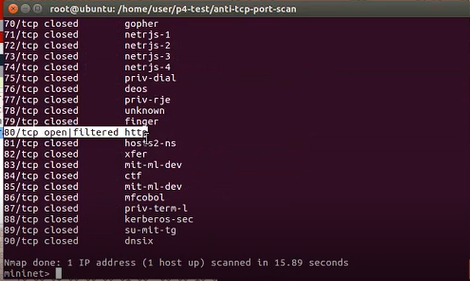


然後84就只有syn，沒有rst，81,82,83還是會放行，可是84就不放，所以後面的h3就不行，就被擋掉，再ping一次會完全不能ping(包刮80 port)

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;

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

register<bit<10>>(100) syn\_cnt;

register<bit<10>>(100) syn\_ack\_cnt;

乙太網路表頭

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;

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

bit<32> ackNo;

bit<4> dataOffset;

bit<4> res;

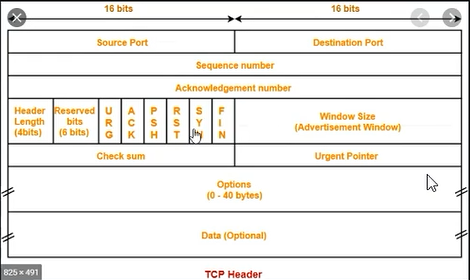
bit<8> flags;

bit<16> window;

bit<16> checksum;

bit<16> urgentPtr;

}



header udp\_t {

bit<16> srcPort;

bit<16> dstPort;

bit<16> udplength;

bit<16> checksum;

}

struct metadata {

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

bit<10> flowlet\_map\_index;

bit<10> syn\_count;

bit<10> syn\_ack\_count;

}

struct headers {

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

ethernet\_t ethernet;

ipv4\_t ipv4;

tcp\_t tcp;

udp\_t udp;

}

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 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 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);

因為這封包是要回去，是回去的方向，所以是destination

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 {

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

bit<1> set\_drop=0;

if (hdr.tcp.isValid()){

//看看是不是syn封包

if(hdr.tcp.flags==2) {

//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

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

add\_syn\_cnt();

//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;

}

如果是0x12就做add\_syn\_ack\_cnt

} 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 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

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