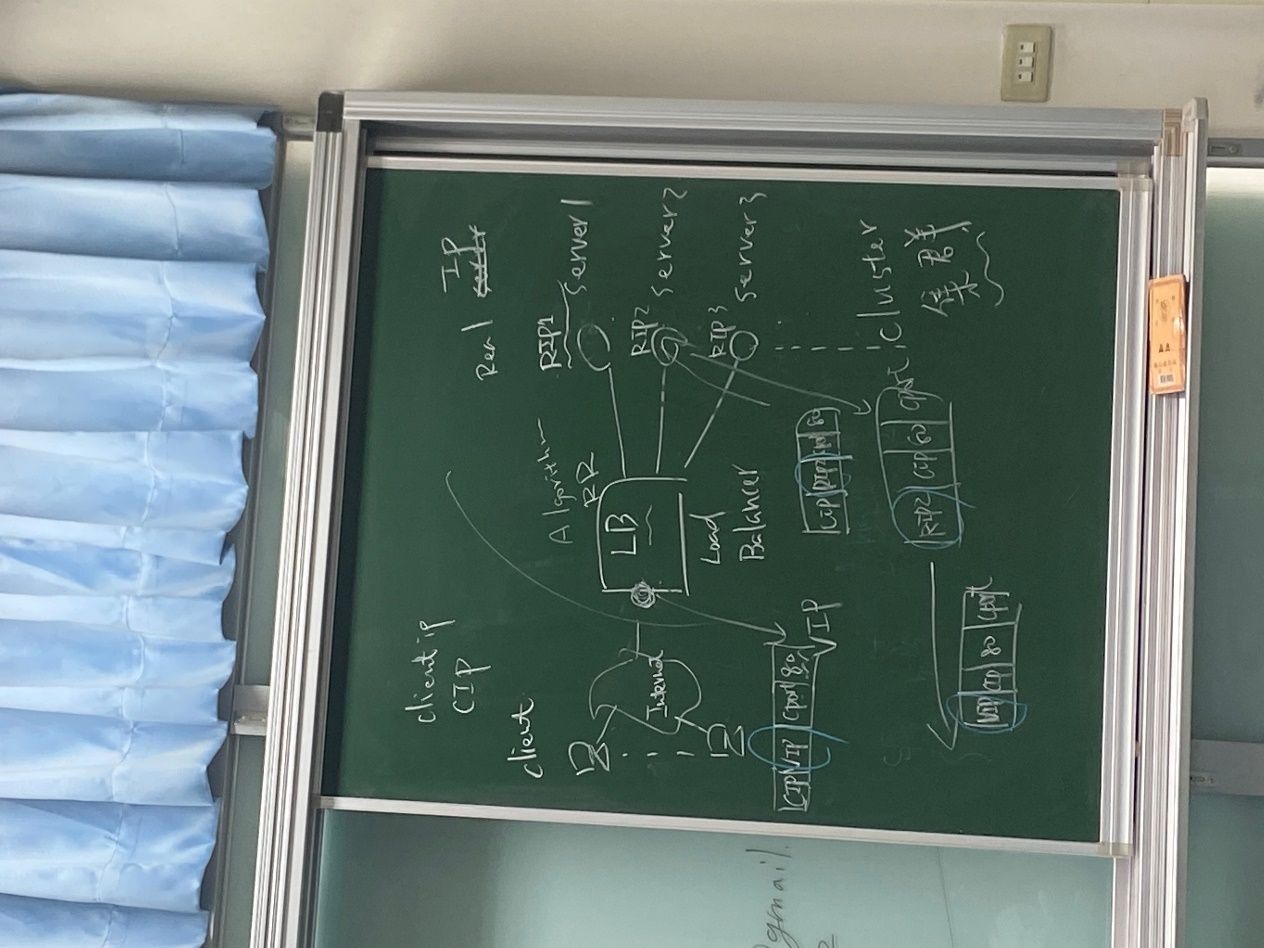
**0625 Load Balancer 使用p4實作負載均衡器**

我們現在所使用的網頁伺服器，一台機器如果性能比較好的，大概能服務一兩萬條連線，如果人數再更多，就沒有辦法負荷。所以，通常這種網頁伺服器都是用集群式的方式在服務，也就是說，它會開很多台，同時很多台在等待把使用者的請求服務分散掉，因為像一些購物節或者是搶票，使用者請求會非常多，幾台機器可能沒辦法應付，所以就會有很多台機器。

**Connection hash**

當很多的客戶，如果集中在一台，容易爆掉，所以負載均衡就是把客戶的請求平均分散到不同伺服器上，這樣的話每一台伺服器的壓力就比較小，反應時間快，如果全都集中在單一台就會來不及服務，就會產生佇列，瀏覽器畫面就會在那邊等待，所以我們希望，當一鍵按下去可以快速得到我想要的東西。



因為這些伺服器每台都會有一個ip，這些伺服器的ip叫做real ip (rip)，這個負載均衡器會有一個對外ip，叫做virtual ip (vip)。

**為什麼要有vip？**

因為當使用者要連線的時候，他們不會去記每一台的ip位址，所以就固定一個固定的ip，他們只要連到這個ip，就由它幫忙做分配。ex:選課系統

客戶的ip叫做client ip (cip)。

負載均衡器的概念就是，它會提供一個固定ip，只要連線連進來這個固定ip，它會自動幫你進行分發的動作。

只要來源相同，雜湊值就相同，如果來源不同，雜湊值相同的機率很小很小，如果真的相同，這種情況就稱為碰撞，那麼這個雜湊函數就不是很好。

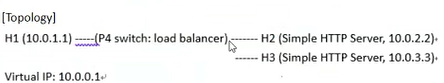
**p4-14寫法轉換成p4-16**

執行步驟：

1. 打開終端機，切到p4-test資料夾
2. 建立資料夾(mkdir hash-lb)，並切到hash-lb資料夾
3. gedit hash-lb.p4 &，去<http://csie.nqu.edu.tw/smallko/sdn/LBP4.htm> 把load\_balance.p4複製並貼上
4. 在終端機輸入指令p4c-bm2-ss –p4v 14 –pp hash-lb16.p4 hash-lb.p4
5. gedit hash-lb16.p4 & 就把p4-14寫法轉換成p4-16了

p4-14轉p4-16網站：<https://p4tw.org/%E5%B0%87-p4-14-%E5%BF%AB%E9%80%9F%E8%BD%89%E6%8F%9B%E8%87%B3-p4-16-%E6%96%B9%E6%B3%95/>

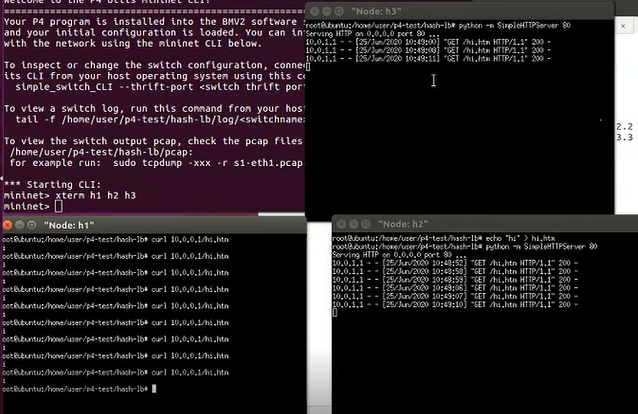
**負載均衡器**

****

**第一個實驗：**

1. 打開終端機，切到p4-test/hash-lb資料夾
2. gedit commands.txt & ，一樣去LBP4.htm網站把s1-command.txt複製並貼上
3. gedit p4app.json &，把p4app.json貼上
4. 在終端機執行p4run，開啟三個終端xterm h1 h2 h3
5. 在h3執行python –m SimpleHTTPServer 80
6. 在h2執行echo “hi” > hi.htm 產生一個簡單的網頁，再執行python –m SimpleHTTPServer 80
7. 在h1執行curl 10.0.0.1/hi.htm

*每次存取的時候，就會選擇不一樣伺服器*



量測效能指標的工具：apachebench

<https://blog.miniasp.com/post/2008/06/30/Using-ApacheBench-ab-to-to-Web-stress-test>

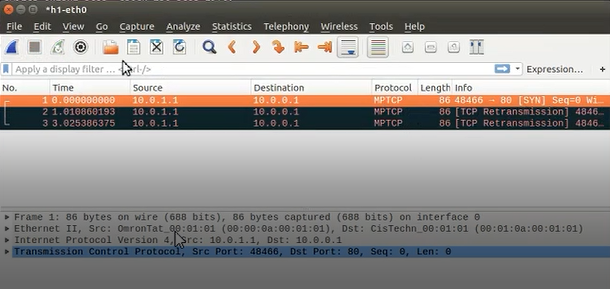
**第二個實驗**

模擬同時最多十條連線，總共要有一千次存取，看看需要花多少時間

*先安裝ab：apt install apache2-utils*

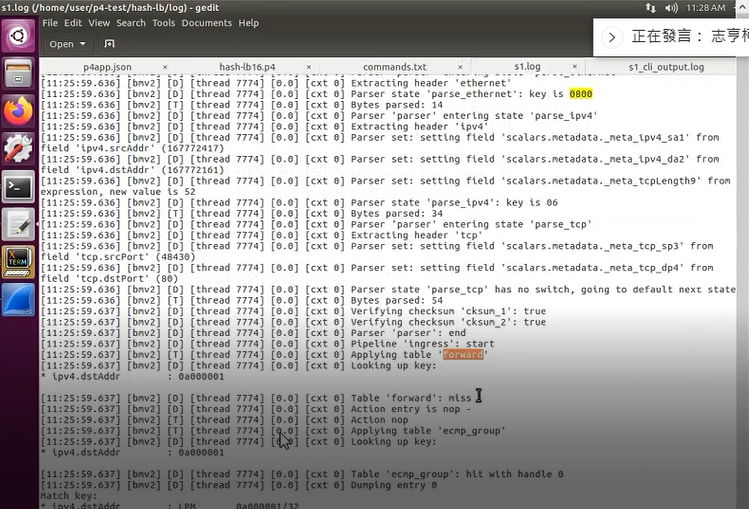
1. 在剛剛的h1輸入 ab –n 1000 –c 10 http://10.0.0.1/hi.htm
2. h2,h3就會跑，h1就會顯示100,200次的時間，最後做完會告訴你總共花了多久
3. 接下來測試4台機器，先去p4app.json,s1-commands.txt改規則
4. 再執行一次p4run，xterm h1 h2 h3 h4 h5
5. h2,3,4,5把伺服器都打開python –m SimpleHTTPServer 80
6. 在h1執行curl 10.0.0.1/hi.htm

*若是失敗，可以再開一個h1，打開wireshark，選擇h1-etho並點左上角鯊魚鰭啟動，然後在h1執行curl 10.0.0.1/hi.htm，下面是問題畫面*

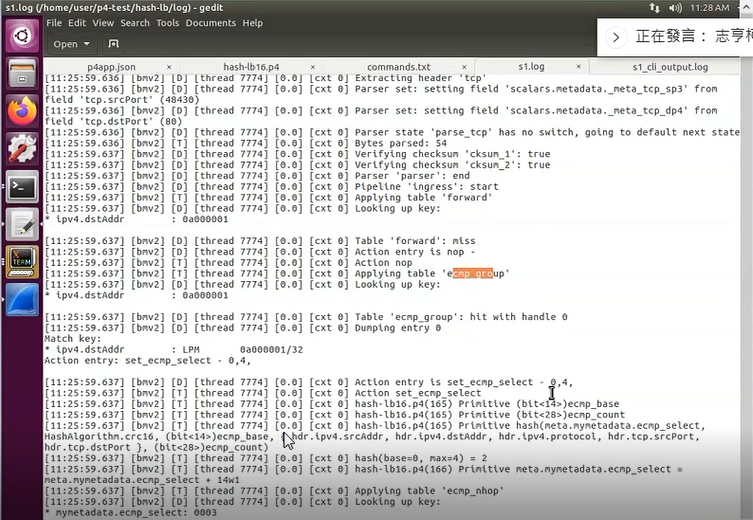


*接著可以到log資料夾(open/otherdocuments)把s1.log,s1\_cli\_output.log打開*

*在s1.log用search/find輸入0800，因為這是第一個ip封包，所以從這邊開始查*

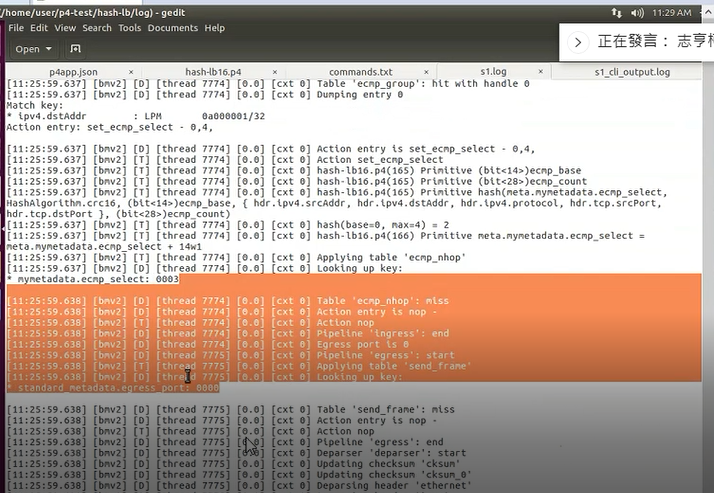


*\*Forward table是反向的時候才會用到，所以第一個基本上不會用到就是miss*

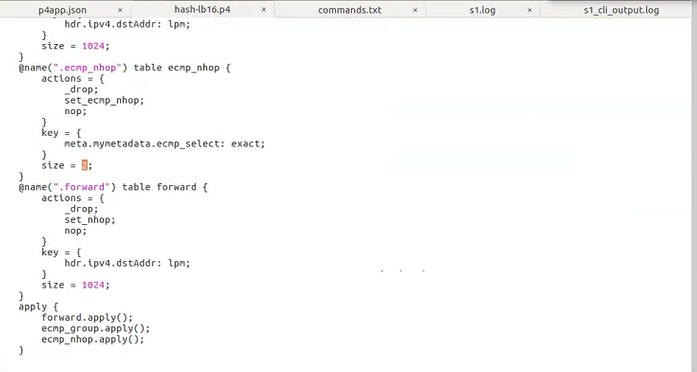


*\*第二個使用hash function去挑1,2,3,4裡面的值出來看看挑到什麼值，所以是hit，所以它就會去挑0~4裡面的值，結果它挑2，規則上要+1就變成3*

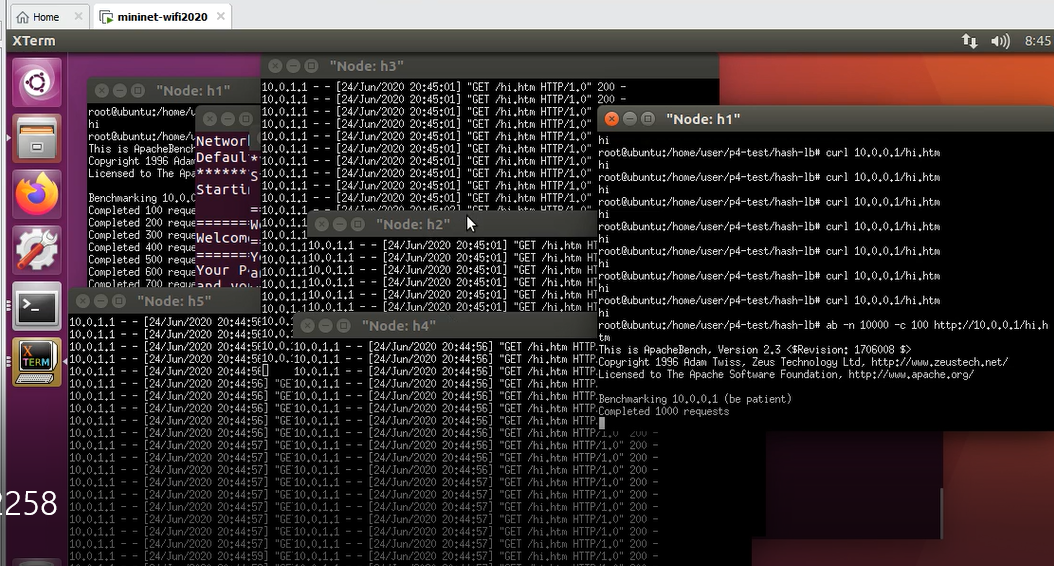
*加完以後，基本上select是3，但下一個的egress\_port是0000，而且是miss，這是問題所在！*



*原因是：剛開始設定的時候，這個表格只能記錄兩筆記錄，但我在commands裡面寫了四筆，也就是說，前面兩筆是有用的，後面兩筆是無用的，會導致後面這兩筆加不進去，所以只要把2改成1024(調大一點)就行*



1. 解決之後，重複上面4~6步驟，不斷在h1執行curl 10.0.0.1/hi.htm
2. 發現幾乎都是在h4 & h5跑，所以改用ab –n 1000 –c 10 http://10.0.0.1/hi.htm跑看看，但還是一樣
3. 只要把1000改10000，10改成100，就可以成功了



**s1-commands.txt**

|  |
| --- |
| table\_set\_default forward nop  table\_set\_default ecmp\_group nop  table\_set\_default ecmp\_nhop nop  原來是0要改成a  table\_set\_default send\_frame nop  table\_add forward set\_nhop 10.0.1.1/32 => 00:00:0a:00:01:01 1  table\_add forward set\_nhop 10.0.2.2/32 => 00:00:0a:00:02:02 2  table\_add forward set\_nhop 10.0.3.3/32 => 00:00:0a:00:03:03 3  table\_add ecmp\_group set\_ecmp\_select 10.0.0.1/32 => 0 2(2改成4)  table\_add ecmp\_nhop set\_ecmp\_nhop 1 => 00:00:0a:00:02:02 10.0.2.2 2  table\_add ecmp\_nhop set\_ecmp\_nhop 2 => 00:00:0a:00:03:03 10.0.3.3 3  table\_add ecmp\_nhop set\_ecmp\_nhop 3 => 00:00:0a:00:03:04 10.0.4.4 4  table\_add ecmp\_nhop set\_ecmp\_nhop 4 => 00:00:0a:00:03:05 10.0.5.5 5  table\_add send\_frame rewrite\_sip 1 => 10.0.0.1  如果目的地是10.0.0.1(vip位址)，就用hash去挑，2代表有兩個選擇，一個是1一個是2。如果定義hash的結果是1(bit)，代表要把請求丟給伺服器2，如果hash值是2就丟到伺服器3(第二台)  0代表base值是0，hash從0開始(參照load\_balance.p4)  當封包回去的時候，要把原本的來源ip轉換成vip(原本是rip2，跑回來變vip 如黑板圖) |

**p4app.json**

{

  "program": "hash-lb16.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": "AppTopo"

  },

  "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",

    "default\_bw":10,

    "default\_delay":"1ms",

    "auto\_gw\_arp": true,

    "links": [["h1", "s1"], ["s1", "h2"], ["s1", "h3"],["s1", "h4"],["s1", "h5"]],

    "hosts": {

      "h1": {

        "ip": "10.0.1.1",

        "gw":"10.0.1.254"

      },

      "h2": {

        "ip" : "10.0.2.2",

        "gw":"10.0.2.254"

      },

      "h3": {

        "ip": "10. 0.3.3",

        "gw":"10.0.3.254"

      }

 "h4": {

        "ip": "10. 0.4.4",

        "gw":"10.0.4.254"

      }

 "h5": {

        "ip": "10. 0.5.5",

        "gw":"10.0.5.254"

      }

    },

    "switches": {

      "s1": {

"cli\_input":"commands.txt",

        "program": "hash-lb16.p4"

      }

    }

  }

}

**Load\_balance.p4**

#include <core.p4>

#include <v1model.p4>

struct meta\_t {

bit<1> do\_forward;

bit<32> ipv4\_sa;

bit<32> ipv4\_da;

bit<16> tcp\_sp;

bit<16> tcp\_dp;

bit<32> nhop\_ipv4;

bit<32> if\_ipv4\_addr;

bit<48> if\_mac\_addr;

bit<1> is\_ext\_if;

bit<16> tcpLength;

bit<8> if\_index;

}

//在資料處理的過程當中，如果有一些比較重要的東西，可以放到metadata裡面

struct mymetadata\_t {

bit<14> ecmp\_select;

}

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;

}

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;

}

header udp\_t {

bit<16> srcPort;

bit<16> dstPort;

bit<16> length\_;

bit<16> checksum;

}

struct metadata {

@name(".meta")

meta\_t meta;

@name(".mymetadata")

mymetadata\_t mymetadata;

}

struct headers {

@name(".arp")

arp\_t arp;

@name(".ethernet")

ethernet\_t ethernet;

@name(".ipv4")

ipv4\_t ipv4;

@name(".tcp")

tcp\_t tcp;

@name(".udp")

udp\_t udp;

}

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);

meta.meta.ipv4\_sa = hdr.ipv4.srcAddr;

meta.meta.ipv4\_da = hdr.ipv4.dstAddr;

meta.meta.tcpLength = hdr.ipv4.totalLen - 16w20;

transition select(hdr.ipv4.protocol) {

8w6: parse\_tcp;

8w17: parse\_udp;

default: accept;

}

}

@name(".parse\_tcp") state parse\_tcp {

packet.extract(hdr.tcp);

meta.meta.tcp\_sp = hdr.tcp.srcPort;

meta.meta.tcp\_dp = hdr.tcp.dstPort;

transition accept;

}

@name(".parse\_udp") state parse\_udp {

packet.extract(hdr.udp);

transition accept;

}

@name(".start") state start {

meta.meta.if\_index = (bit<8>)standard\_metadata.ingress\_port;

transition parse\_ethernet;

}

}

control egress(inout headers hdr, inout metadata meta, inout standard\_metadata\_t standard\_metadata) {

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

mark\_to\_drop(standard\_metadata);

}

@name(".rewrite\_sip") action rewrite\_sip(bit<32> sip) {

hdr.ipv4.srcAddr = sip; //如果sip是1(參照commands.txt)出口是1

}

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

}

@name(".send\_frame") table send\_frame {

actions = {

\_drop;

rewrite\_sip;

nop;

}

key = {

standard\_metadata.egress\_port: exact;

//如果出口是1，也就是要離開load balancer，離開的方向是1，就要把來源ip改成10.0.0.1(vip)

}

size = 256;

}

apply {

send\_frame.apply();

}

}

control ingress(inout headers hdr, inout metadata meta, inout standard\_metadata\_t standard\_metadata) {

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

mark\_to\_drop(standard\_metadata);

}

@name(".set\_ecmp\_select") action set\_ecmp\_select(bit<8> ecmp\_base, bit<8> ecmp\_count) {

//這個地方要來做hash

//count值選完以後，hash完的值放在這(meta……select)

hash(meta.mymetadata.ecmp\_select, HashAlgorithm.crc16,//演算法 (bit<14>)ecmp\_base,//hash的值的基準點從哪裡開始(預設值0,base=0，參照commands.txt)

//hash五個欄位：來源ip，目的ip，通訊協定，來源埠號，目的埠號

{ hdr.ipv4.srcAddr, hdr.ipv4.dstAddr, hdr.ipv4.protocol, hdr.tcp.srcPort, hdr.tcp.dstPort }, (bit<28>)ecmp\_count);//最多挑選挑的上限，從0開始挑到count-1，出來的值最多是多少(2就是選擇0 & 1，3就是0,1,2) meta.mymetadata.ecmp\_select = meta.mymetadata.ecmp\_select + 14w1;

//我們挑出來的值雖然是從0 & 1，但我希望它的值可以再+1，變成1 & 2

}

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

}

@name(".set\_ecmp\_nhop") action set\_ecmp\_nhop(bit<48> nhop\_mac, bit<32> nhop\_ipv4, bit<9> port) {

standard\_metadata.egress\_spec = port;

hdr.ipv4.dstAddr = nhop\_ipv4;

hdr.ethernet.dstAddr = nhop\_mac;

hdr.ipv4.ttl = hdr.ipv4.ttl - 8w1;

}

@name(".set\_nhop") action set\_nhop(bit<48> dmac, bit<9> port) {

standard\_metadata.egress\_spec = port;

hdr.ethernet.dstAddr = dmac;

hdr.ipv4.ttl = hdr.ipv4.ttl - 8w1;

}

@name(".ecmp\_group") table ecmp\_group {

actions = {

\_drop;

set\_ecmp\_select;

nop;

}

key = {

hdr.ipv4.dstAddr: lpm;

}

size = 1024;

}

@name(".ecmp\_nhop") table ecmp\_nhop {

actions = {

\_drop;

set\_ecmp\_nhop;

nop;

}

key = {

meta.mymetadata.ecmp\_select: exact;

}

size = 1024;

}

@name(".forward") table forward {

actions = {

\_drop;

set\_nhop;

nop;

}

key = {

hdr.ipv4.dstAddr: lpm;

}

size = 1024;

}

apply {

Ingress的部分有三個table：

**forward.apply();**：table\_add forward set\_nhop 10.0.1.1/32 => 00:00:0a:00:01:01 1

**ecmp\_group.apply();**：table\_add ecmp\_group set\_ecmp\_select 10.0.0.1/32 => 0 2

如果他選擇vip，就set\_ecmp\_select;

**ecmp\_nhop.apply();**

根據select值，去挑選伺服器

forward.apply();

ecmp\_group.apply();

ecmp\_nhop.apply();

}

}

control DeparserImpl(packet\_out packet, in headers hdr) {

apply {

packet.emit(hdr.ethernet);

packet.emit(hdr.arp);

packet.emit(hdr.ipv4);

packet.emit(hdr.udp);

packet.emit(hdr.tcp);

}

}

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);

verify\_checksum\_with\_payload(true, { hdr.ipv4.srcAddr, hdr.ipv4.dstAddr, 8w0, hdr.ipv4.protocol, meta.meta.tcpLength, hdr.tcp.srcPort, hdr.tcp.dstPort, hdr.tcp.seqNo, hdr.tcp.ackNo, hdr.tcp.dataOffset, hdr.tcp.res, hdr.tcp.flags, hdr.tcp.window, hdr.tcp.urgentPtr }, hdr.tcp.checksum, HashAlgorithm.csum16);

}

}

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

apply {

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);

update\_checksum\_with\_payload(true, { hdr.ipv4.srcAddr, hdr.ipv4.dstAddr, 8w0, hdr.ipv4.protocol, meta.meta.tcpLength, hdr.tcp.srcPort, hdr.tcp.dstPort, hdr.tcp.seqNo, hdr.tcp.ackNo, hdr.tcp.dataOffset, hdr.tcp.res, hdr.tcp.flags, hdr.tcp.window, hdr.tcp.urgentPtr }, hdr.tcp.checksum, HashAlgorithm.csum16);

}

}

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