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，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

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

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

[Topology]

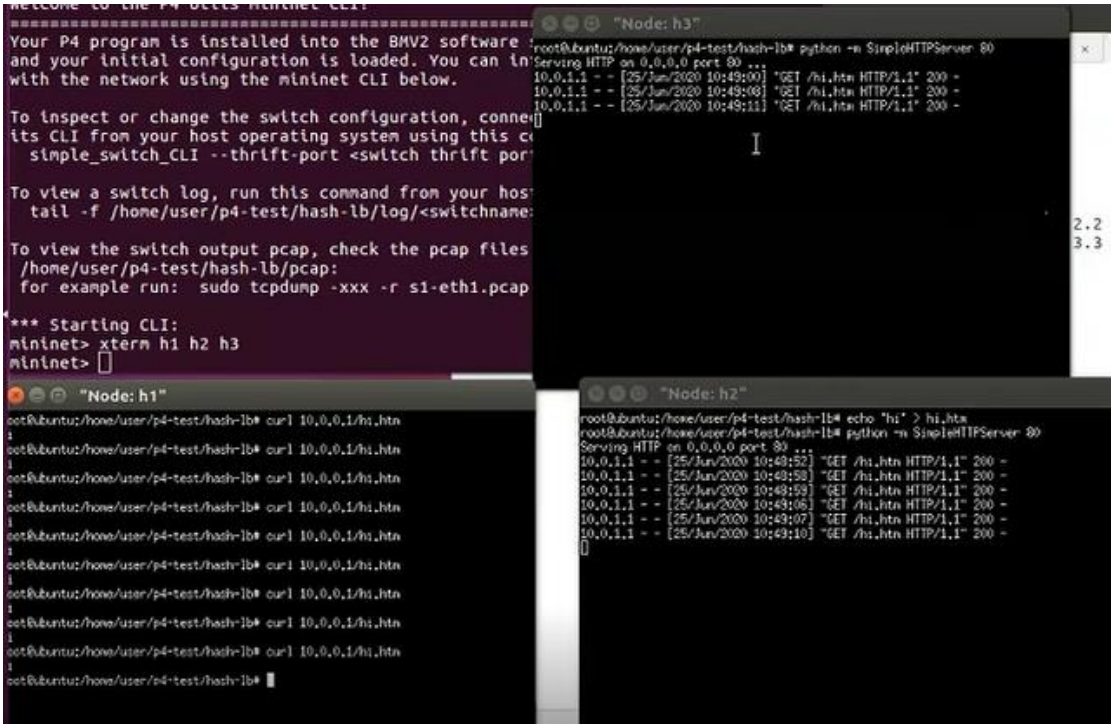
H1 (10.0.1.1) -----(P4 switch: load balancer)------ H2 (Simple HTTP Server, 10.0.2.2)

------ H3 (Simple HTTP Server, 10.0.3.3)

Virtual IP: 10.0.0.1

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



s1-commands.txt

table_set_default forward nop
table_set_default ecmp_group nop
table_set_default ecmp_nhop nop
table_set_default send_frame nop

原來是 0 要改成 a

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
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 send_frame rewrite_sip 1 => 10.0.0.1

如果目的地是 10.0.0.1(vip 位址)，2 代表有兩個選擇，一個是 1 一個是 2。如果 hash 的結果是 1(bit)，代表要把請求丟給伺服器 2，如果 hash 值是 3 就丟到伺服器 3(第二台)

當封包回去的時候，要把原本的來源 ip 轉換成 vip

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"]],
  "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"
    }
  },
  "switches": {
    "s1": {
      "cli_input":"command.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;
}

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;
        }
        @name(".nop") action nop() {
        }
        @name(".send_frame") table send_frame {
            actions = {
                _drop;
                rewrite_sip;
                nop;
```

```
            }
            key = {
                standard_metadata.egress_port: exact;
            }
            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
            hash(meta.mymetadata.ecmp_select, HashAlgorithm.crc16,
(bit<14>)ecmp_base, { hdr.ipv4.srcAddr, hdr.ipv4.dstAddr, hdr.ipv4.protocol,
hdr.tcp.srcPort, hdr.tcp.dstPort }, (bit<28>)ecmp_count);
//hash 五個欄位：來源 ip，目的 ip`,通訊協定，來源埠號，目的埠號
            meta.mymetadata.ecmp_select = meta.mymetadata.ecmp_select + 14w1;
        }
        @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 {
            forward.apply();
            ecmp_group.apply();
            ecmp_nhop.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;

```
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;
```