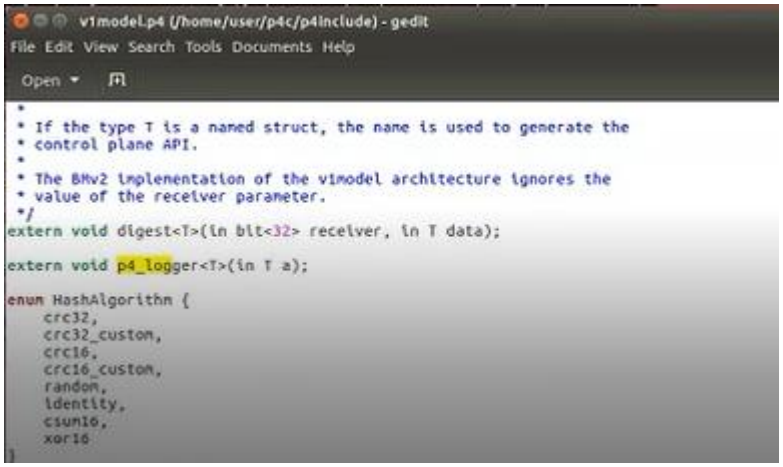
0615 在 p4 程式裡多一個函式：p4_logger

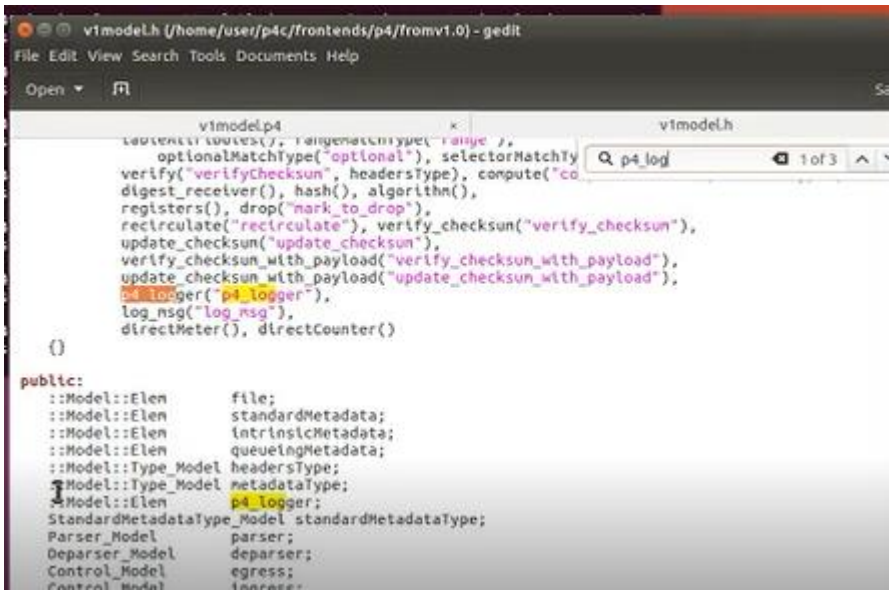
在開始程式的時候，想把那些東西印出來就可以印出來

安裝步驟：

1. 先到 https://github.com/cslev/p4extern 網站
2. 打開終端機，切到 p4-test，執行 gedit &
3. Open -> other documents -> user -> p4c -> p4include -> v1model.p4
4. 把 extern void p4_logger<T>(in T a);加上

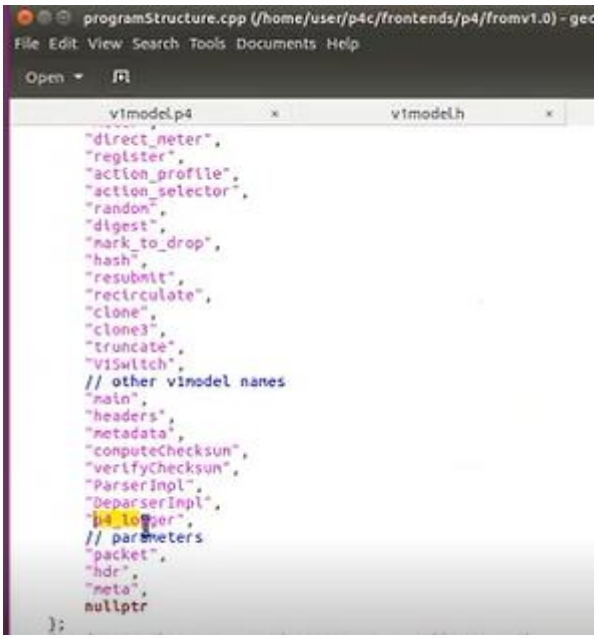


5. 加上以後，繼續加下一個。一樣 Open -> other documents
6. user -> p4c -> frontends -> p4 -> fromv1.0 -> v1model.h
7. 加入 `p4_logger("p4_logger"), /` ::Model::Elem p4_logger;

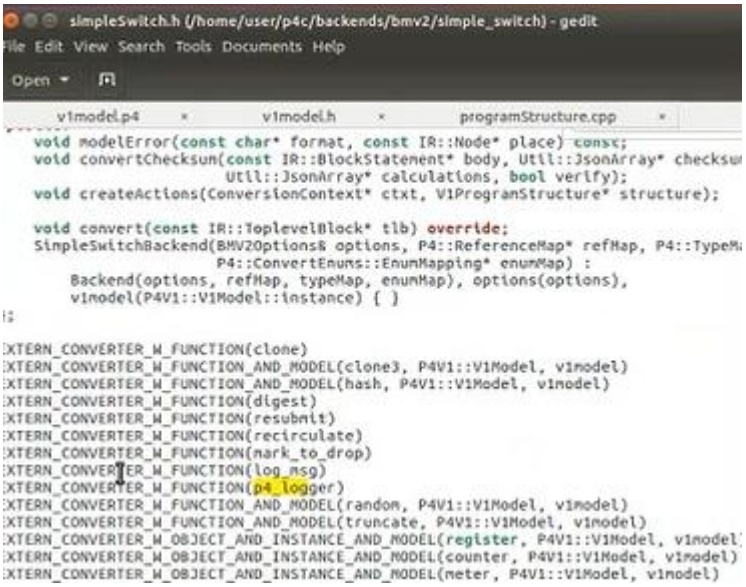


8. 繼續加下一個。Open -> other documents
9. user -> p4c -> frontends -> p4 -> fromv1.0 -> programStructure.cpp

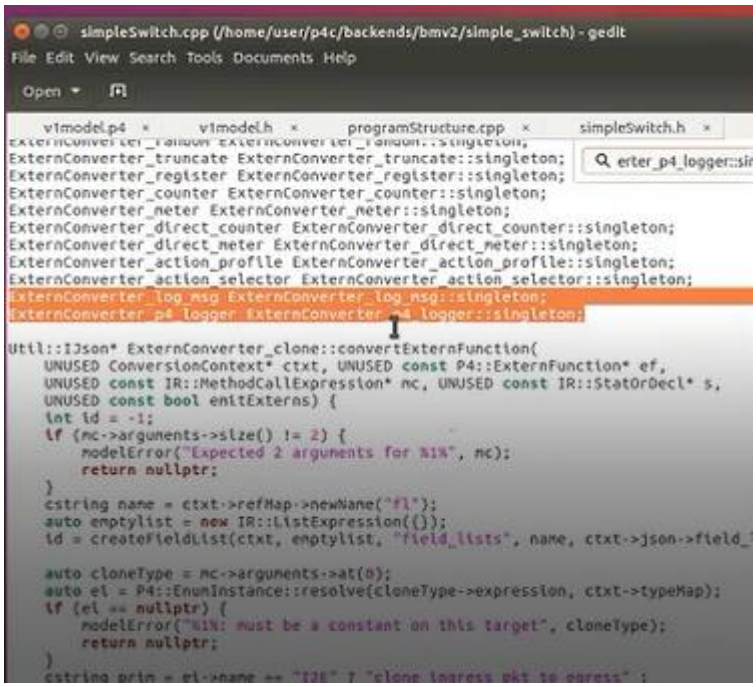
10. 加入　"p4_logger",



11. 繼續加下一個。Open -> other documents
12. user -> p4c -> backends -> bmv2 -> simple_switch -> simpleSwitch.h
13. 加入 EXTERN_CONVERTER_W_FUNCTION(p4_logger)



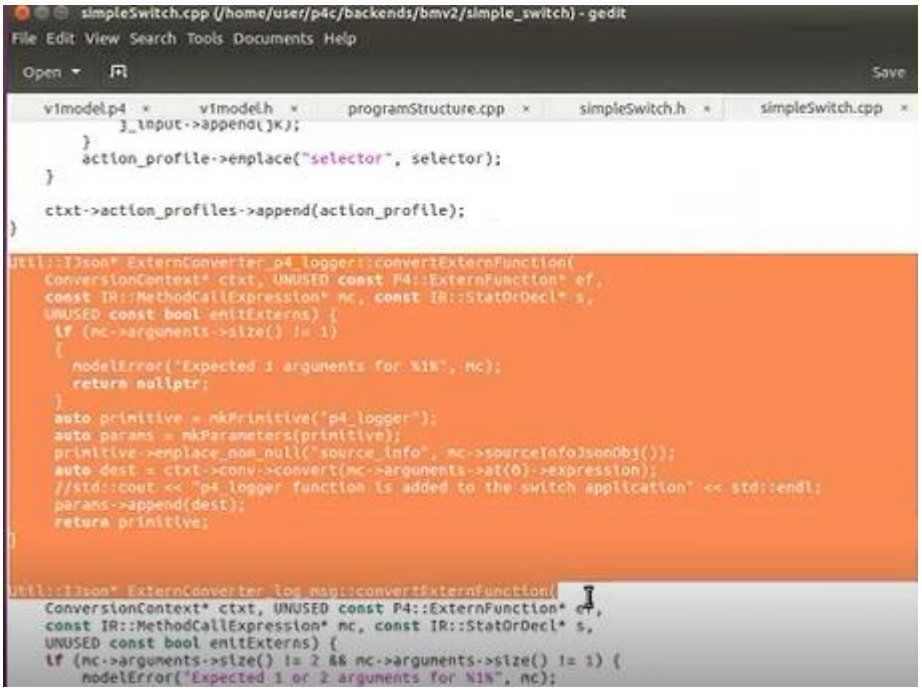
14. 繼續加下一個。Open -> other documents
15. user -> p4c -> backends -> bmv2 -> simple_switch -> simpleSwitch.cpp
    這個地方照抄網站上會錯！要照老師的改！
16. 加入 ExternConverter_p4_logger ExternConverter_p4_logger::singleton;



17. 同檔案還有另一個地方要加，加入整段：

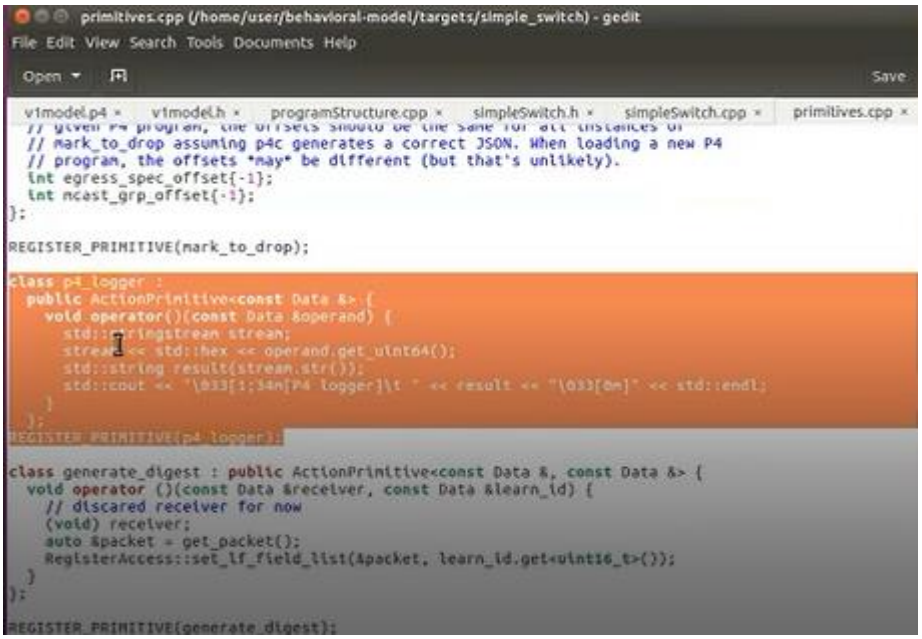
```
Util::IJson* ExternConverter_p4_logger::convertExternFunction(
        ConversionContext* ctxt, UNUSED const P4::ExternFunction* ef,
        const IR::MethodCallExpression* mc, const IR::StatOrDecl* s,
        UNUSED const bool emitExterns) {
          if (mc->arguments->size() != 1)
          {
             modelError("Expected 1 arguments for %1%", mc);
             return nullptr;
          }
          auto primitive = mkPrimitive("p4_logger");
          auto params = mkParameters(primitive);
          primitive->emplace_non_null("source_info", mc->sourceInfoJsonObj());
          auto dest = ctxt->conv->convert(mc->arguments->at(0)->expression);
          //std::cout << "p4_logger function is added to the switch application" <<
  std::endl;
          params->append(dest);
          return primitive;
}
```

整段插在 Util::IJson* ExternConverter_log_msg::convertExternFunction(上面



18. 最後一個加入。Open -> other documents
19. user -> behavior-model -> targets -> simple_switch -> primitives.cpp
20. 加入整段

```
class p4_logger :
    public ActionPrimitive<const Data &> {
        void operator()(const Data &operand) {
            std::stringstream stream;
            stream << std::hex << operand.get_uint64();
            std::string result(stream.str());
            std::cout << "\033[1;34m[P4 logger]\t " << result << "\033[0m]" <<
std::endl;
        }
    };
REGISTER_PRIMITIVE(p4_logger);
```

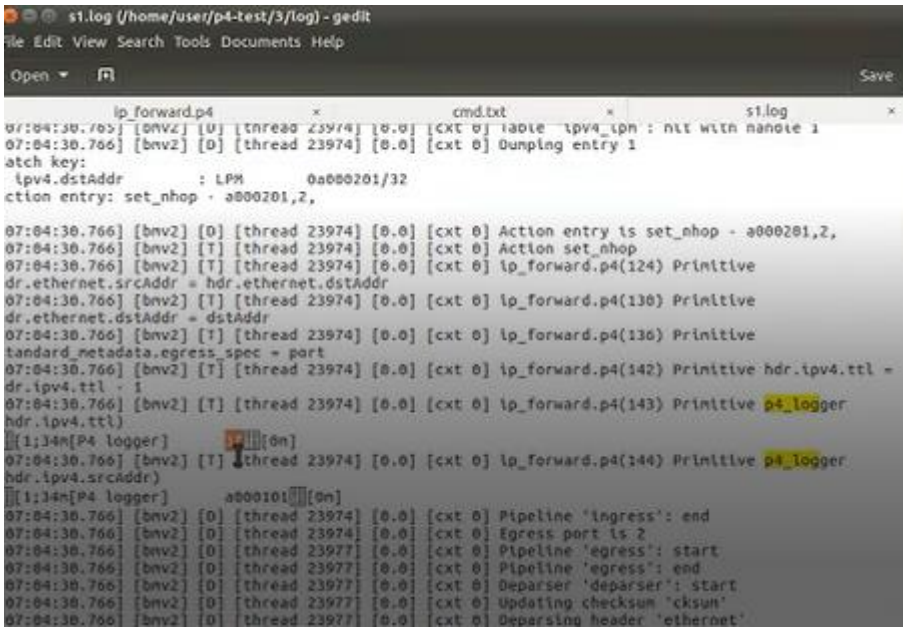


21. 做完以後，程式碼要重新編譯。把改好的程式碼 save 並關掉
22. 切到 user/p4c/build 資料夾
23. 執行 make –j4
24. 執行 make install
25. 跑完後切到 user/ behavior-model /targets/simple_switch 資料夾
26. 執行 make –j4
27. 執行 make install

沒有出錯就可以開始用了！

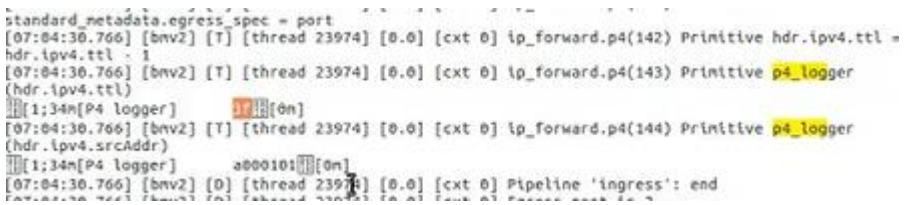
執行步驟：

1. 打開終端機，切到 p4-test/3 資料夾
2. gedit ip_forward.p4 &
3. p4run
4. mininet 執行 h1 ping –c 3 h2，ping 完結束 exit
5. gedit 那裏打開 Open -> other documents -> log -> s1.log
6. 就會看到值是多少，例如說出現 3f，也就是說現在的 ttl 是 3f



7. 打開小算盤，左上角三條線選擇程式設計人員，選擇 16 進位(HEX)
8. 輸入 3f，十進位就是 77

把來源 ip 印出來



a0000101 這個封包就是 10.0.1.1

在程式執行過程當中，在 p4 處理過程當中，想要把哪個欄位或什麼值印出來，只需要在前面加上 p4_logger，然後把想要印出來的東西放在後面，就可以察覺他們之間的變化

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;


        //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;
```

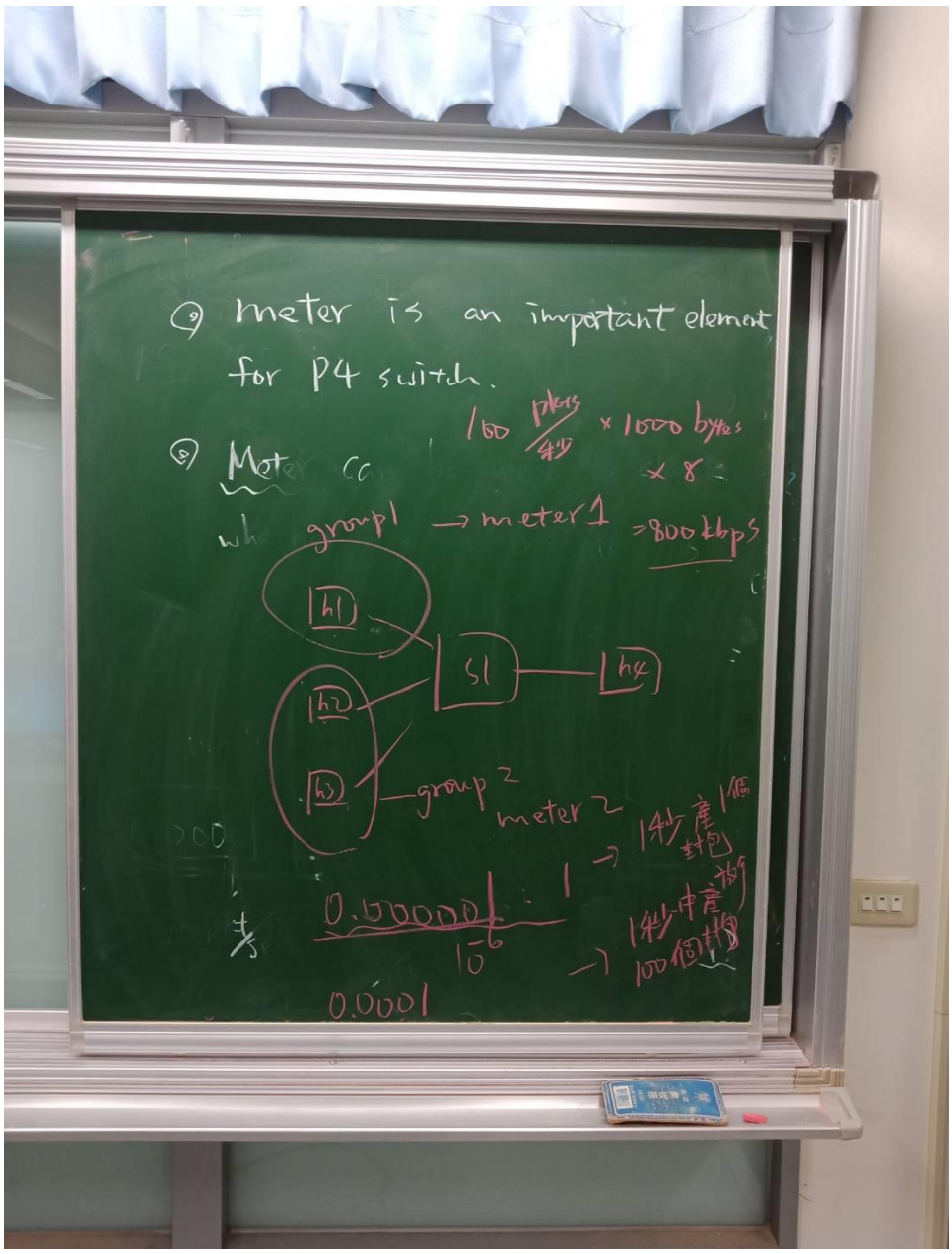
```
        //decrease ttl by 1
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    p4_logger(hdr.ipv4.ttl);
    p4_logger(hdr.ipv4.srcAddr);
    }

    @name("._drop") action _drop() {
        mark_to_drop(standard_metadata);
    }

    @name(".ipv4_lpm") table ipv4_lpm {
        actions = {
            set_nhop;
            _drop;
        }

        key = {
            hdr.ipv4.dstAddr: lpm;
        }

        size = 512;
        const default_action = _drop();
    }

    apply {
        ipv4_lpm.apply();
    }
}


control DeparserImpl(packet_out packet, in headers hdr) {

    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.arp);
        packet.emit(hdr.ipv4);
    }
```

在執行過程中，如果想知道某個欄位的值是多少，例如：執行過程中 ttl 會-1，這個 ttl 值是多少，就可以打上 p4_logger(值，這裡是 hdr.ipv4.ttl)，加上分號並儲存

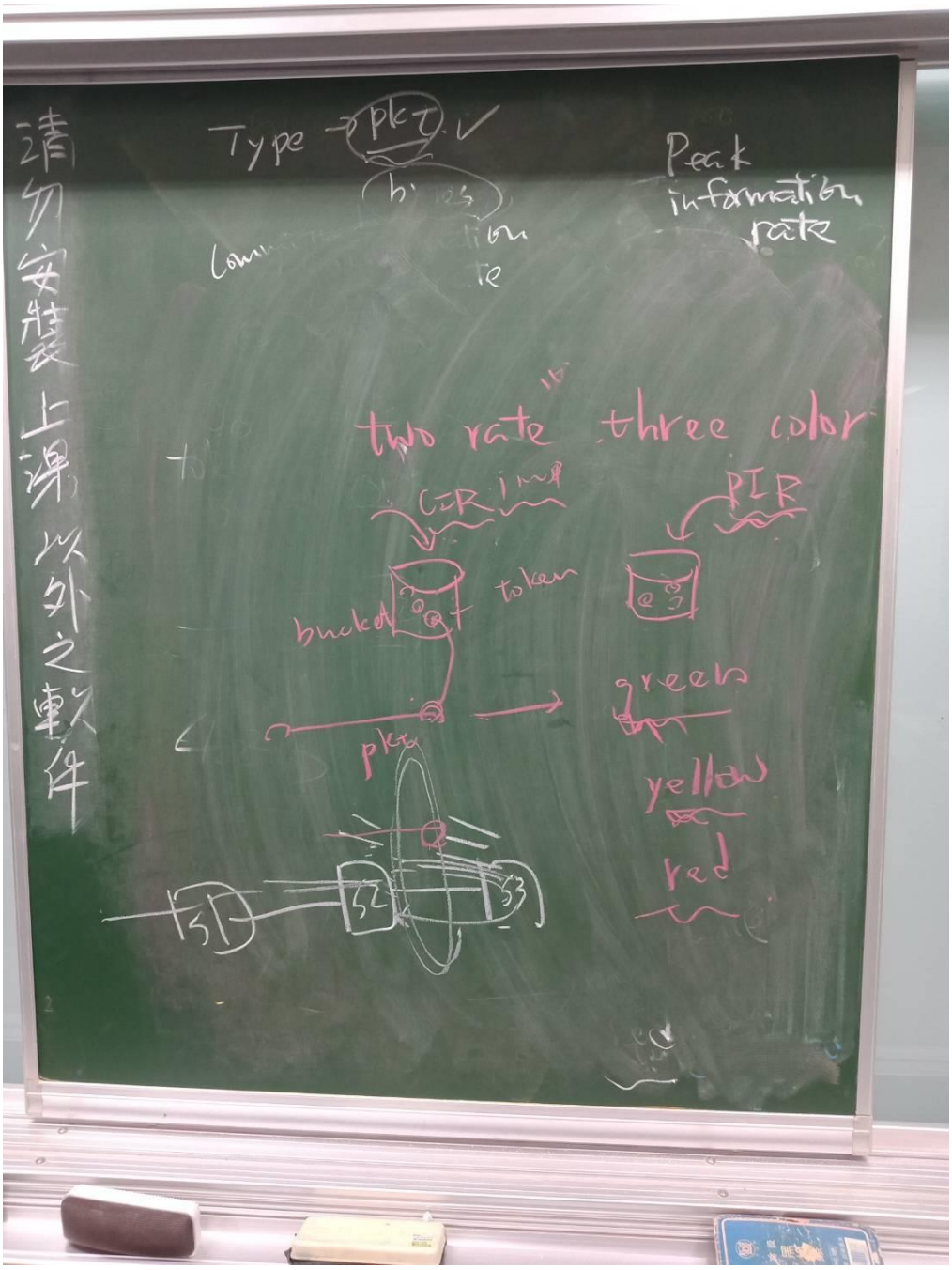
若是想知道這個封包現在來源 ip 是多少，就可以把欄位放()中。
p4_logger(hdr.ipv4.srcAddr);

```
}


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 {
        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);
    }
}


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

test-meter 測量器



網路講到品質服務的時候會用到。Meter 就是客戶跟 IST 業者或網路管理者之間可能會有一些協議，例如說：我希望客戶在傳輸的時候，傳輸的速度不要超過多少，假設不超過 1M，如果傳輸的速度是在 1M 以下，我就保障這些資料都能正確完整且快速送達，超過就不保證。所以它就是在做一件事：量測客戶送進來的資料，會不會送太多，如果送太多，網路管理者就要把多送出來的資料刪除掉。就是檢查客戶傳送進來的資料有沒有符合規範，有就放行，沒有就丟棄。



p4 的 meter 基本上是採用 two rate three color 演算法，這個演算法裡面會有兩個 bucked(桶子)，這個桶子會有相關的兩個速率，CIR & PIR。

CIR：它會填充 token(代幣)，假設封包是 1M 傳輸速率送進來的，就可以得到一個 token，如果送太快，token 就消耗完了就沒有 token 了。一個封包進來，如果它可以得到 token，就把這個封包標記成綠色；如果封包進來的速度很快，第一個桶子已經沒有 token 了，但是第二個有，就標記成黃色；如果封包再更快，兩個桶子都沒有 token，就標記成紅色。基本上，綠色就是放行，黃色&紅色就看有什麼策略，例如：嚴格一點的，黃色跟紅色就都丟棄；緩和一點的，如果還有資源，可以讓黃色通行，沒有資源就把黃色丟棄；紅色一定是丟棄。

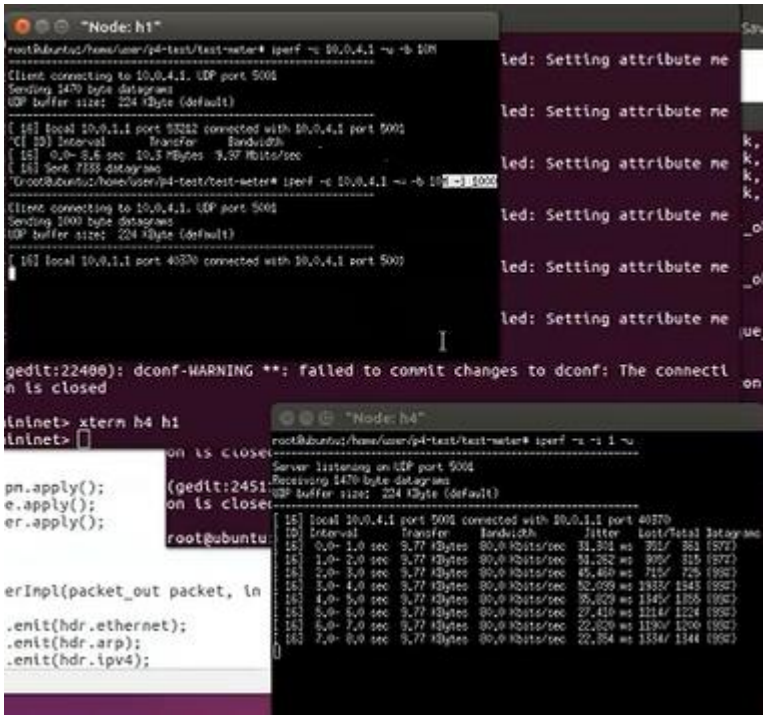
PIR：也會填充代幣。資料傳的時候有可能會高低高低傳，速度不固定的，PIR

就是允許最高可以多快。

執行步驟：

第一個實驗

1. 打開終端機，切到 p4-test/test-meter 資料夾
2. gedit ip_forward.p4 cmd.txt &
3. p4run，然後 xterm h4 h1
4. 在 h4 執行 iperf –s –i 1 –u
5. 在 h1 執行 iperf –c 10.0.4.1 –u –b 10m –l 1000



一秒鐘放行 10 個封包，每個封包 1000byte，10*1000*8(bits)=80kbps(去 3 個 0)

6. 可以試著把 0.00001 改成 0.0001(每秒放行 100 個)
7. 再重新跑一次 p4run，重複 3~5 的動作

h1 是自己一個人用，所以速率是 80k

第二個實驗

1. 把剛剛的 h4 crtl+c 執行 iperf –s –i 1 –u –p 5555
2. 再開一個 h4(mininet 執行 xterm h4)，執行 iperf –s –i 1 –u –p 6666
3. mininet 執行 xterm h2 h3
4. h2 執行 iperf –c 10.0.4.1 –u –b 10m –t 1000 –l 1000 –p 5555(連到 h4)
5. h3 執行 iperf –c 10.0.4.1 –u –b 10m –l 1000 –p 6666(連到另一個 h4)

如果只有 h2 一個人用，5555 那台 h4 會顯示速率 80k，但若 h3 也開始用，5555 那台 h4 速率就會變成 60 幾 k，5555+6666 就會大約等於 80k(因為 h2 & h3 是共用頻寬)

ip_forware.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 {
    bit<32> meter_tag;
}

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) {
    //如果要用 meter，就要先宣告 meter 這樣的物件，10 代表最多有十個
meter 可以用，處理的時候以封包為單位，meter 名稱 my_meter;
    meter(10, MeterType.packets) my_meter;

    @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;
        //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;
          //decrease ttl by 1
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    }
    @name("._drop") action _drop() {
        mark_to_drop(standard_metadata);
    }
    @name(".ipv4_lpm") table ipv4_lpm {
        actions = {
            set_nhop;
            _drop;
        }
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        size = 512;
        const default_action = _drop();
    }
```

```
  action m_action(bit<32> meter_idx) {
        my_meter.execute_meter((bit<32>)meter_idx, meta.meter_tag);
    }

    //選擇它是第幾個 meter
    table m_table {
        key = {
        hdr.ipv4.srcAddr: lpm;
        }
        actions = {
            m_action;
            NoAction;
        }
        size = 1024;
        default_action = NoAction();
    }
    //標記完顏色以後，怎麼處理這些封包
    table m_filter {
        key = {
            meta.meter_tag: exact;
        }

        actions = {
            _drop;
            NoAction;       //0 代表不處理
        }
        size = 1024;
        default_action = _drop();       //預設值是 drop
    }

    apply {
        ipv4_lpm.apply();
        m_table.apply();
        m_filter.apply();
    }
}
```

```
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 {
        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);
    }
}

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

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
table_add ipv4_lpm set_nhop 10.0.3.1/32 => 00:00:0a:00:03:01 3
table_add ipv4_lpm set_nhop 10.0.4.1/32 => 00:00:0a:00:04:01 4
```

4 台主機

table_add m_table m_action 10.0.1.0/24 => 1
table_add m_table m_action 10.0.2.0/24 => 2
table_add m_table m_action 10.0.3.0/24 => 2
table_add m_filter NoAction 0 =>
meter_set_rates my_meter 1 0.00001:1 0.005:1
meter_set_rates my_meter 2 0.0001:1 0.0005:1

從 10.0.1.0 進來的網路用 meter1
從 10.0.2.0/10.0.3.0 進來的用 meter2(共用)

0 代表 green，NoAction 不處理=放行
1 & 2 採用內定動作：drop

meter 編號 CIR 設定方式 PIR 設定方式

0.00001:1： 一秒鐘放行 10 個封包(0.00001*10 的 6 次方)