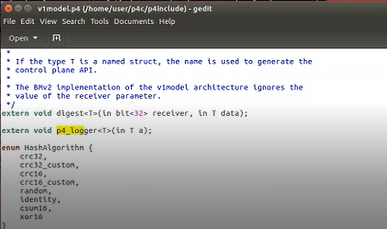
**0615在p4程式裡多一個函式：p4\_logger**

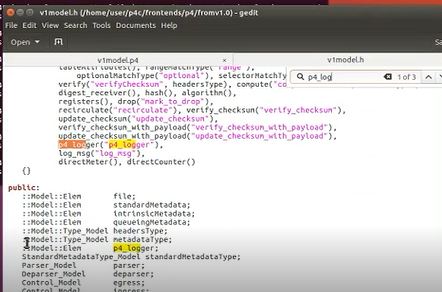
*在開始程式的時候，想把那些東西印出來就可以印出來*

安裝步驟：

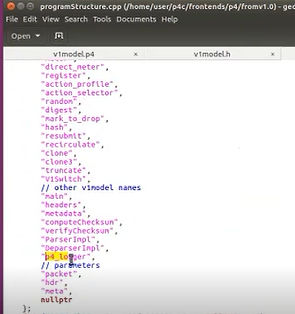
1. 先到<https://github.com/cslev/p4extern>網站
2. 打開終端機，切到p4-test，執行gedit &
3. Open -> other documents -> user -> p4c -> p4include -> v1model.p4
4. 把extern void p4\_logger<T>(in T a);加上



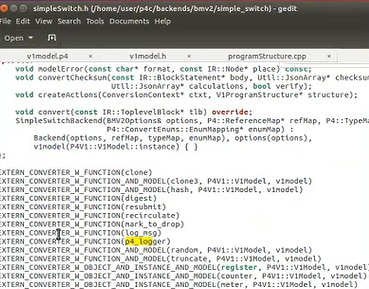
1. 加上以後，繼續加下一個。一樣Open -> other documents
2. user -> p4c -> frontends -> p4 -> fromv1.0 -> v1model.h
3. 加入p4\_logger("p4\_logger"), / ::Model::Elem p4\_logger;



1. 繼續加下一個。Open -> other documents
2. user -> p4c -> frontends -> p4 -> fromv1.0 -> programStructure.cpp
3. 加入 "p4\_logger",



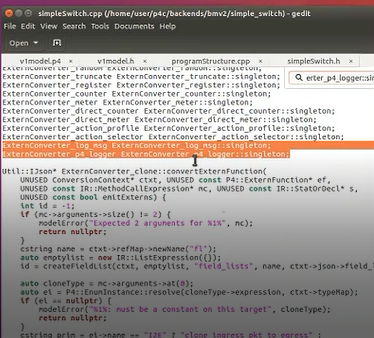
1. 繼續加下一個。Open -> other documents
2. user -> p4c -> backends -> bmv2 -> simple\_switch -> simpleSwitch.h
3. 加入EXTERN\_CONVERTER\_W\_FUNCTION(p4\_logger)



1. 繼續加下一個。Open -> other documents
2. 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！

1. 加入ExternConverter\_p4\_logger ExternConverter\_p4\_logger::singleton;



1. 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);

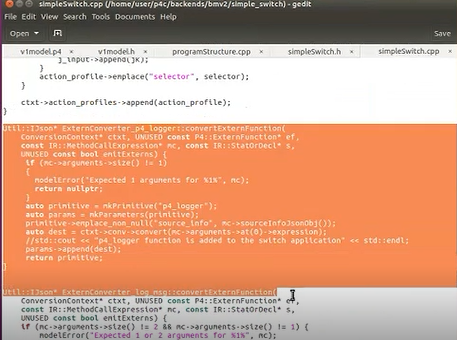
//std::cout << "p4\_logger function is added to the switch application" << std::endl;

params->append(dest);

return primitive;

}

整段插在Util::IJson\* ExternConverter\_log\_msg::convertExternFunction(上面



1. 最後一個加入。Open -> other documents
2. user -> behavior-model -> targets -> simple\_switch -> primitives.cpp
3. 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();

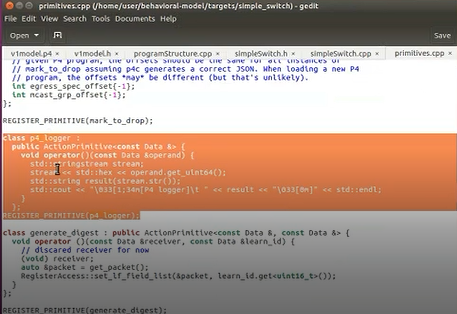
std::string result(stream.str());

std::cout << "\033[1;34m[P4 logger]\t " << result << "\033[0m]" << std::endl;

}

};

REGISTER\_PRIMITIVE(p4\_logger);

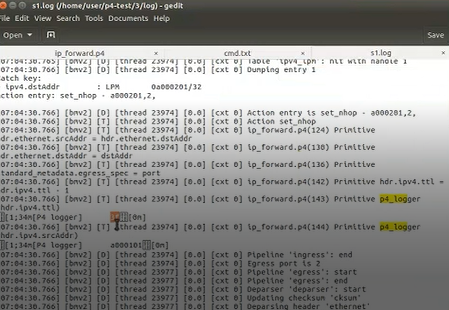


1. 做完以後，程式碼要重新編譯。把改好的程式碼save並關掉
2. 切到user/p4c/build資料夾
3. 執行make –j4
4. 執行make install
5. 跑完後切到user/ behavior-model /targets/simple\_switch資料夾
6. 執行make –j4
7. 執行make install

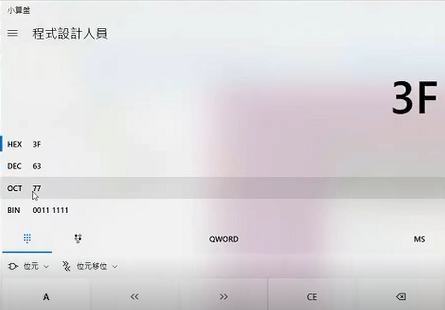
*沒有出錯就可以開始用了！*

執行步驟：

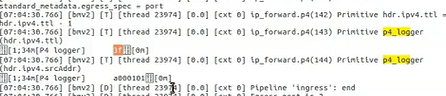
1. 打開終端機，切到p4-test/3資料夾
2. gedit ip\_forward.p4 &
3. p4run
4. mininet執行h1 ping –c 3 h2，ping完結束exit
5. gedit那裏打開Open -> other documents -> log -> s1.log
6. 就會看到值是多少，例如說出現3f，也就是說現在的ttl是3f



1. 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)
2. 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

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

hdr.ipv4.ttl = hdr.ipv4.ttl - 1;

p4\_logger(hdr.ipv4.ttl);

p4\_logger(hdr.ipv4.srcAddr);

}

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

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);

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);

}

}

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;