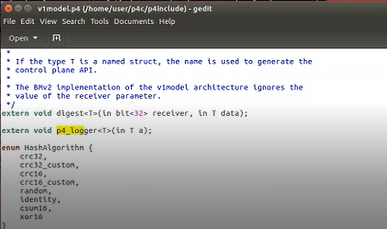
**0615在p4程式裡多一個函式：p4\_logger**

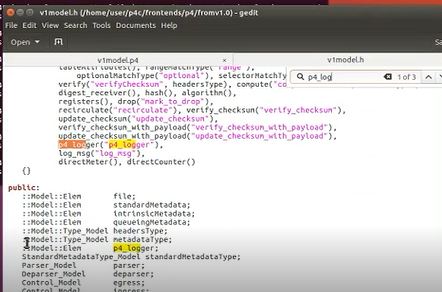
*在開始程式的時候，想把那些東西印出來就可以印出來*

安裝步驟：

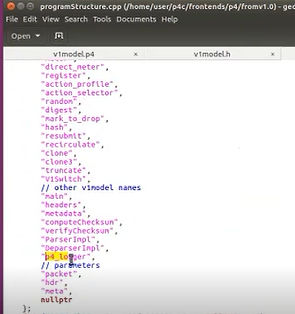
1. 先到<https://github.com/cslev/p4extern>網站
2. 打開終端機，切到p4-test，執行gedit &
3. Open -> other documents -> user -> p4c -> p4include -> v1model.p4
4. 把extern void p4\_logger<T>(in T a);加上



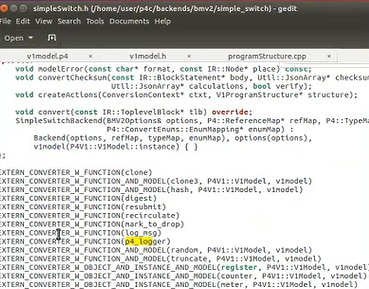
1. 加上以後，繼續加下一個。一樣Open -> other documents
2. user -> p4c -> frontends -> p4 -> fromv1.0 -> v1model.h
3. 加入p4\_logger("p4\_logger"), / ::Model::Elem p4\_logger;



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



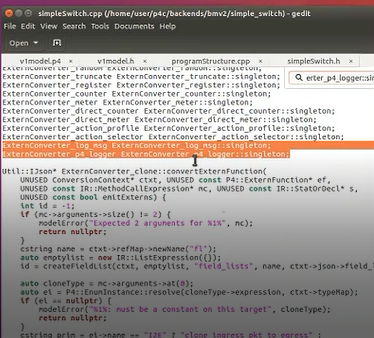
1. 繼續加下一個。Open -> other documents
2. user -> p4c -> backends -> bmv2 -> simple\_switch -> simpleSwitch.h
3. 加入EXTERN\_CONVERTER\_W\_FUNCTION(p4\_logger)



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);

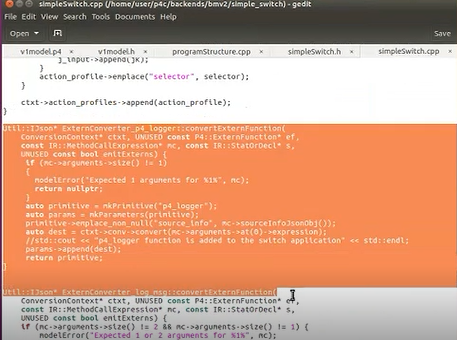
//std::cout << "p4\_logger function is added to the switch application" << std::endl;

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();

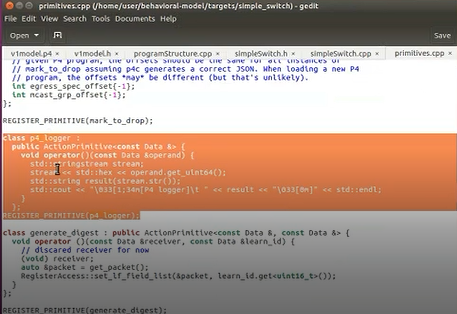
std::string result(stream.str());

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

}

};

REGISTER\_PRIMITIVE(p4\_logger);

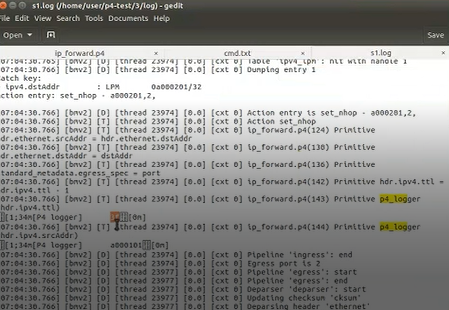


1. 做完以後，程式碼要重新編譯。把改好的程式碼save並關掉
2. 切到user/p4c/build資料夾
3. 執行make –j4
4. 執行make install
5. 跑完後切到user/ behavior-model /targets/simple\_switch資料夾
6. 執行make –j4
7. 執行make install

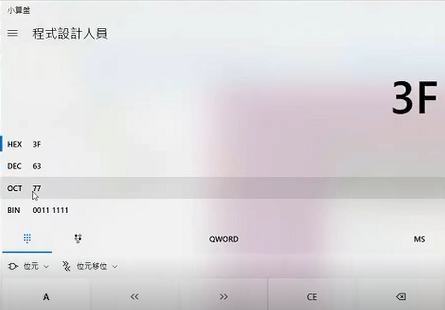
*沒有出錯就可以開始用了！*

執行步驟：

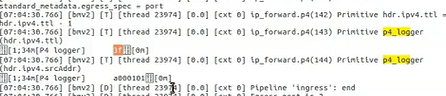
1. 打開終端機，切到p4-test/3資料夾
2. gedit ip\_forward.p4 &
3. p4run
4. mininet執行h1 ping –c 3 h2，ping完結束exit
5. gedit那裏打開Open -> other documents -> log -> s1.log
6. 就會看到值是多少，例如說出現3f，也就是說現在的ttl是3f



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 {

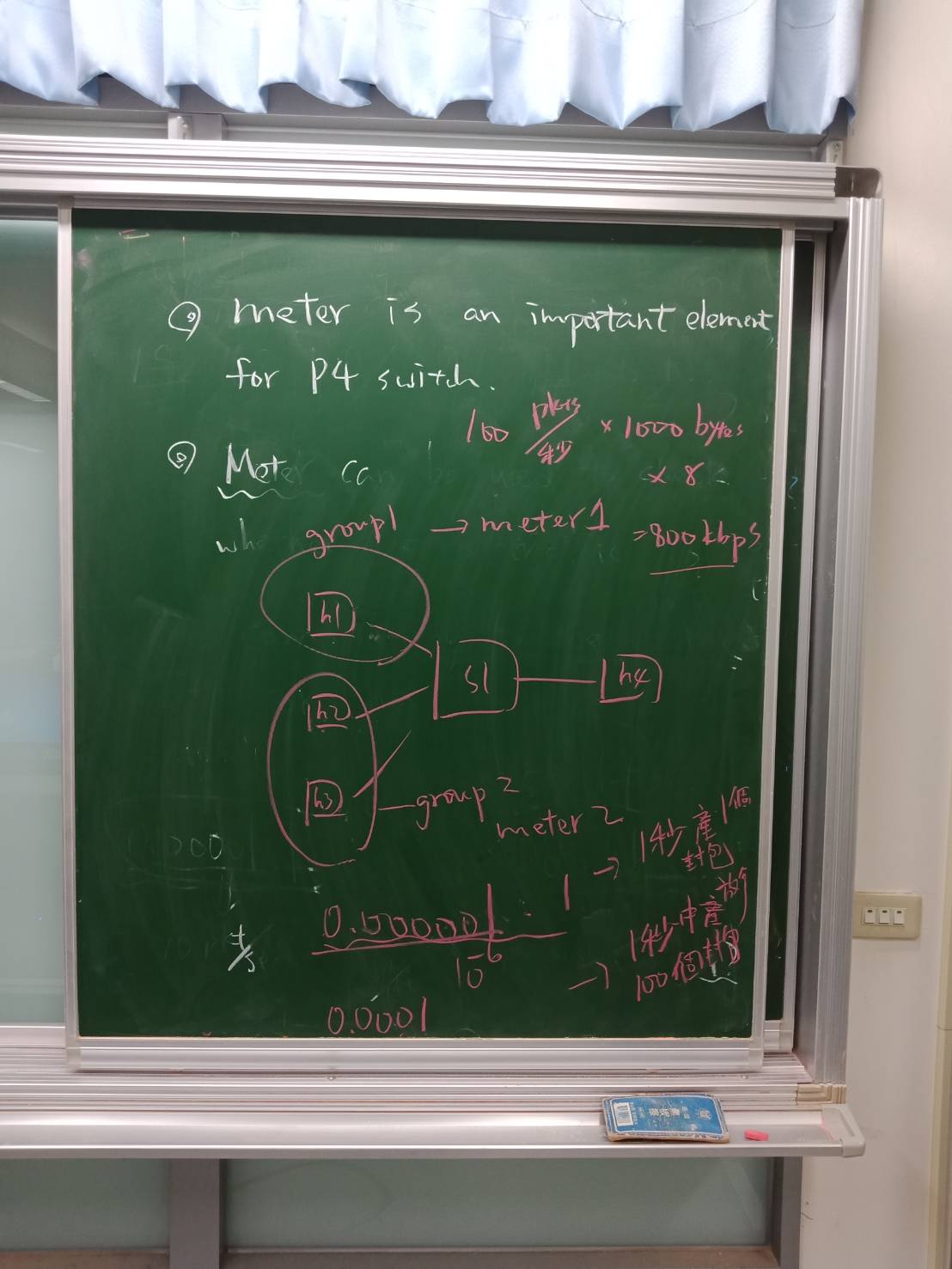
update\_checksum(true, { hdr.ipv4.version, hdr.ipv4.ihl, hdr.ipv4.diffserv, hdr.ipv4.totalLen, hdr.ipv4.identification, hdr.ipv4.flags, hdr.ipv4.fragOffset, hdr.ipv4.ttl, hdr.ipv4.protocol, hdr.ipv4.srcAddr, hdr.ipv4.dstAddr }, hdr.ipv4.hdrChecksum, HashAlgorithm.csum16);

}

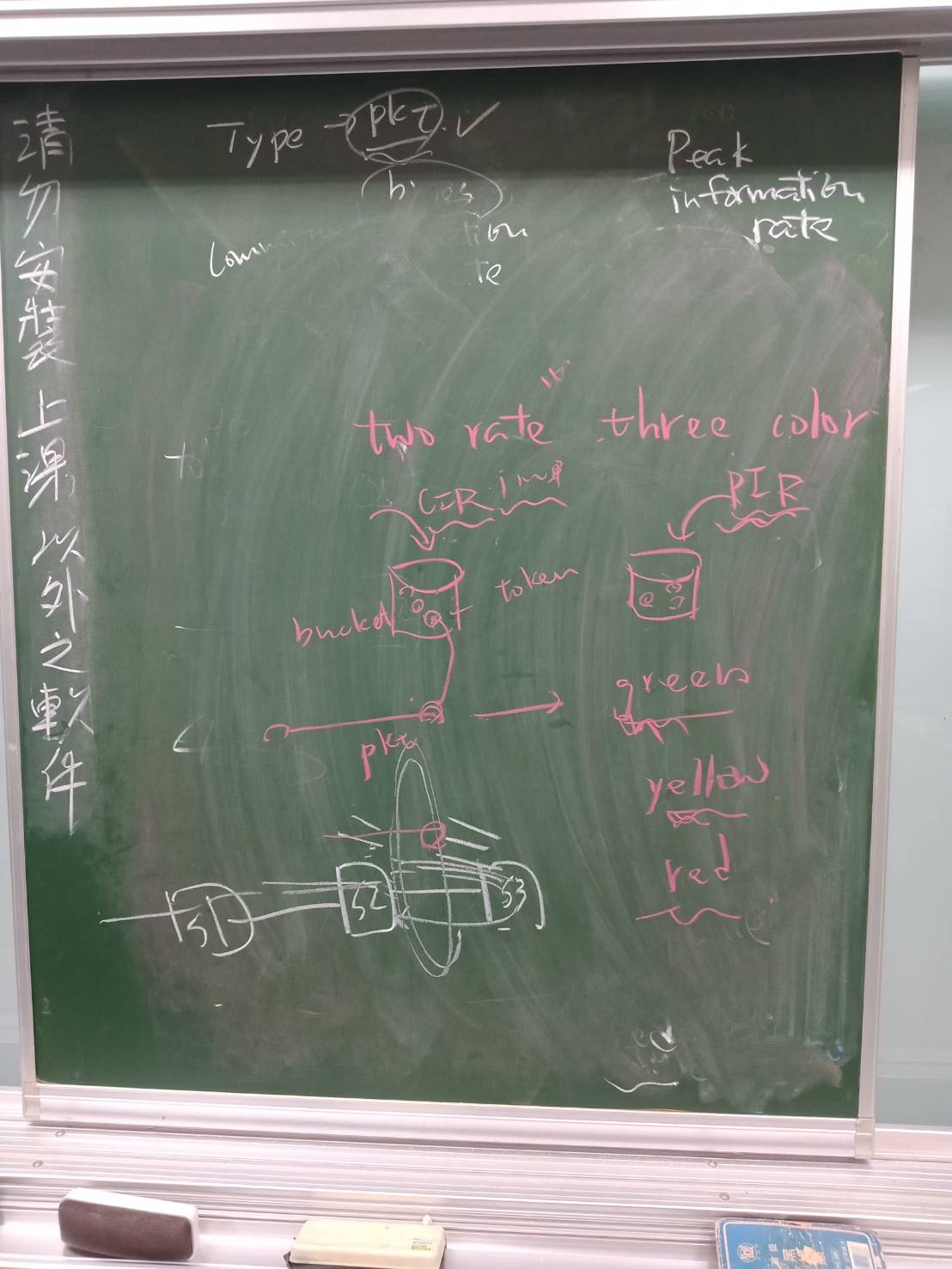
}

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

**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。

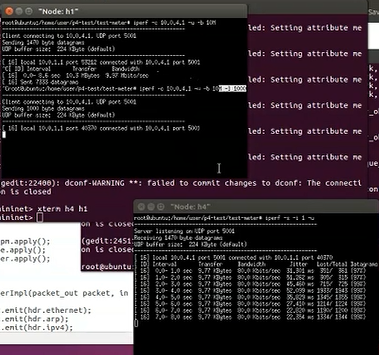
CIR：它會填充token(代幣)，假設封包是1M傳輸速率送進來的，就可以得到一個token，如果送太快，token就消耗完了就沒有token了。一個封包進來，如果它可以得到token，就把這個封包標記成綠色；如果封包進來的速度很快，第一個桶子已經沒有token了，但是第二個有，就標記成黃色；如果封包再更快，兩個桶子都沒有token，就標記成紅色。基本上，綠色就是放行，黃色&紅色就看有什麼策略，例如：嚴格一點的，黃色跟紅色就都丟棄；緩和一點的，如果還有資源，可以讓黃色通行，沒有資源就把黃色丟棄；紅色一定是丟棄。

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)

1. 可以試著把0.00001改成0.0001(每秒放行100個)
2. 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

4台主機

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

table\_add m\_table m\_action 10.0.1.0/24 => 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)

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 =>

0代表green，NoAction不處理=放行

1 & 2採用內定動作：drop

meter\_set\_rates my\_meter 1 0.00001:1 0.005:1

meter\_set\_rates my\_meter 2 0.0001:1 0.0005:1

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)