P4 là gì? Data plane, control plane là gì?

Control plane của 1 thiết bị mạng (router hay switch) là tập hợp bất kỳ thứ gì cần thiết để thực hiện việc định tuyến trên thiết bị đó.

Những gói tin thuộc control plane là những gói tin được thiết bị đó gửi đi, hoặc được gửi tới thiết bị đó.

Data plane (forwarding plane) là bất kỳ thứ gì chạy qua thiết bị đó. Tức là trong quá trình định tuyến, các gói tin dùng thiết bị này (router or switch) để tới điểm đích mà chúng được gửi tới. Hay thiết bị này chỉ kaf 1 trung gian để chuyển gói tin đó tới một đích nào đó.

Eg: Topo: R1 --- R2 --- R3

R1 & R3 được config BGP. Như vậy, với R1 và R3, routing updates từ 2 router này thuộc về control plane. Với R2, những gói tin này thuộc về data plane.

P4 là ngôn ngữ giúp lập trình quản lý & điều khiển phần data plane này. Ví dụ, R2 được nạp 1 đoạn code p4 để mỗi gói tin đi qua sẽ tách phần header và phần payload, để tính toán lại checksum, giám TTL của header và gộp lại để chuyển tiếp đi.

## Cấu trúc của p4:

### \* Templates

header\_type ethernet\_t {

fields {

..

}

}

parser start {

...

}

action rewrite\_mac(smac) {

...

}

table send\_frame {

reads {

standard\_metadata.egress\_port: exact;

}

actions {

rewrite\_mac;

\_drop;

}

size: 256;

}

control egress {

apply(send\_frame);

}

### 1. P4 types

typedef bit<48> macAddr\_t;

typedef bit<32> ip4Addr\_t;

header ethernet\_t {

macAddr\_t dstAddr;

macAddr\_t 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;

ip4Addr\_t srcAddr;

ip4Addr\_t dstAddr;

}

### 2. Parsers

Đây là các hàm tách mỗi gói tin thành 2 phần: headers và metadata.

Mỗi parser có 3 states được định nghĩa trước:

- start

- accept

- reject

Có thể định nghĩa thêm các state khác

Tại mỗi state, có thể có hoặc không thực thi câu lệnh nào, sau đó chuyển tiếp (transition) sang state khác (loops are ok).

Eg:

parser parse\_ipv4 {

extract(ipv4);

return ingress;

}

/\* or \*/

parser parse\_ethernet {

extract(ethernet);

return select(latest.etherType) {

ETHERTYPE\_IPV4 : parse\_ipv4;

default: ingress;

}

}

/\* or \*/

parser MyParser ( packet\_in packet,

out headers hdr,

inout metadata meta,

inout standard\_metadata\_t std\_meta

) {

state start {

parser.extract(hdr.ethernet);

transition accept;

}

}

• Parser sẽ tạo ra 1 “Parsed Representation” (PR) cho mỗi gói tin

• Match+action tables sử dụng PR để match và thực thi action

• Bất kỳ giá trị nào được sử dụng bởi chuwong trình đều phải có trong PR (or metadata from the target)

**\* Multi-field select statement**

parser parse\_ipv4 {

extract(ipv4);

set\_metadata(ipv4\_metadata.lkp\_ipv4\_sa, ipv4.srcAddr);

set\_metadata(ipv4\_metadata.lkp\_ipv4\_da, ipv4.dstAddr);

set\_metadata(l3\_metadata.lkp\_ip\_proto, ipv4.protocol);

set\_metadata(l3\_metadata.lkp\_ip\_ttl, ipv4.ttl);

/\* Fields are joined for a match \*/

return select(latest.fragOffset, latest.ihl, latest.protocol) {

0x0000501 : parse\_icmp;

0x0000506 : parse\_tcp;

0x0000511 : parse\_udp;

default : ingress;

}

}

**\* Calculated fields**

header ipv4\_t ipv4;

field\_list ipv4\_checksum\_list {

ipv4.version;

ipv4.ihl;

ipv4.diffserv;

ipv4.totalLen;

ipv4.identification;

ipv4.flags;

ipv4.fragOffset;

ipv4.ttl;

ipv4.protocol;

ipv4.srcAddr;

ipv4.dstAddr;

}

field\_list\_calculation ipv4\_checksum {

input {

ipv4\_checksum\_list;

}

algorithm : csum16;

output\_width : 16;

}

calculated\_field ipv4.hdrChecksum {

verify ipv4\_checksum;

update ipv4\_checksum;

}

### 3. Controls

Trong một control có thể khai báo biến, tạo bảng, khởi tạo externs.

Có thể sử dụng apply {} để thiết lập các câu lệnh thực thi.

Hoặc thực thi các actions bằng cách apply 1 table, không thể apply trực tiếp các actions này. Muốn gọi 1 actions mà không phải match (ko cần điều kiện gì), có thể sử dụng 1 bảng trống (“empty” table)

Eg:

control ingress {

If (valid(ipv4) and ipv4.ttl > 0) {

apply(ipv4\_lpm);

apply(forward);

}

}

/\* or \*/

/\* Đổi source và dest MAC address, và chuyển tiếp traffic \*/

control MyIngress ( inout headers hdr,

inout metadata meta,

inout standard\_metadata\_t std\_meta

) {

bit<48> tmp;

apply {

tmp = hdr.ethernet.dstAddr;

hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;

hdr.ethernet.srcAddr = tmp;

std\_meta.egress\_spec = std\_meta.ingress\_port;

// • ingress\_port - the port on which the packet arrived

// • egress\_spec - the port to which the packet should be sent to

// • egress\_port - the port on which the packet is departing from (read only in egress pipeline)

}

}

Ingress/Egress port ~ switch port related.

Ingress port (network port) là cổng nhận traffic.

Egress port (tool port) là cổng chuyển tiếp (forward network traffic).

Phải thiết lập kết nối giữa Ingress port và egress port để có thể chuyển tiếp network traffic.

Eg: Với chế độ “switchport mode access”, sheader info phải bị xóa trước khi chuyển tiếp ra cổng egress port. Với chế độ switchport mode trunk, header info được giữ nguyên.

### 4. Tables

1 table là 1 đơn vị cơ bản của Match-Action pipeline.

- Định nghĩa what data to match on và match kind.

- Định nghĩa list các actions có thể.

- (Optional) Định nghĩa 1 số properties.

+ Size

+ Default action

+ Static entries

- Mỗi bảng gồm 1 hoặc nhiều entries (rules)

- Mỗi entry gồm:

+ 1 key để match

+ 1 (và chỉ 1) action sẽ được thực thi nếu gói tin match với entry này (match on the key)

+ Action data (có thể rỗng)

Tóm lại, 1 bảng có nhiệm vụ map những gói tin đến (incoming packets) (sử dụng PR) thành logical incoming interface.

Eg1:

table choose\_input\_intf {

/\* what to examine from each packet \*/

reads {

/\* exact is match kinds \*/

standard\_metadata.ingress\_port : exact;

vlan\_tag : valid;

vlan\_tag.vid : exact;

}

/\* permitted actions \*/

actions {

drop;

Ingress\_intf\_extract;

}

/\* properties (size) \*/

size 16384;

}

Eg2:

|  |  |  |
| --- | --- | --- |
| Key | Action | Action data |
| 10.0.1.1/32 | Ipv4\_forward | dstAddr=00:00:00:00:01:01  port=1 |
| 10.0.1.2/32 | drop |  |
| \*` | NoAction |  |

table ipv4\_lpm {

/\* lookup on keys / or can use reads \*/

key = {

/\* lpm~ match kinds \*/

hdr.ipv4.dstAddr: lpm;

}

/\* perform actions on those matched \*/

actions = {

ipv4\_forward;

drop;

NoAction;

}

/\* properties \*/

size = 1024;

default\_action = NoAction();

}

### 4. Match kinds

|  |  |
| --- | --- |
| /\* core.p4 \*/  match\_kind {  exact,  ternary,  lpm  }  /\* v1model.p4 \*/  match\_kind {  range,  selector  }  /\* Some other architecture \*/  match\_kind {  regexp,  fuzzy  } | • The standard library (core.p4) defines three standard match kinds:  ◦ Exact match  ◦ Ternary match  ◦ LPM match  • The architecture (v1model.p4) defines two additional match kinds:  ◦ range  ◦ selector  • Other architectures may define (and provide implementation for) additional match kinds |

### 5. Actions

Các actions con trong 1 action cha được thực thi tuần tự (trước v1.0.2 thì được thực thi song song)

**Applying tables in controls:**

control MyIngress ( inout headers hdr,

inout metadata meta,

inout standard\_metadata\_t standard\_metadata

) {

table ipv4\_lpm {

...

}

apply {

...

ipv4\_lpm.apply();

...

}

}

## Example simple\_router.p4:

header\_type ethernet\_t {

fields {

dstAddr : 48;

srcAddr : 48;

etherType : 16;

}

}

header\_type ipv4\_t {

fields {

version : 4;

ihl : 4;

diffserv : 8;

totalLen : 16;

identification : 16;

flags : 3;

fragOffset : 13;

ttl : 8;

protocol : 8;

hdrChecksum : 16;

srcAddr : 32;

dstAddr: 32;

}

}

parser start {

return parse\_ethernet;

}

#define ETHERTYPE\_IPV4 0x0800

header ethernet\_t ethernet;

parser parse\_ethernet {

extract(ethernet);

return select(latest.etherType) {

ETHERTYPE\_IPV4 : parse\_ipv4;

default: ingress;

}

}

header ipv4\_t ipv4;

field\_list ipv4\_checksum\_list {

ipv4.version;

ipv4.ihl;

ipv4.diffserv;

ipv4.totalLen;

ipv4.identification;

ipv4.flags;

ipv4.fragOffset;

ipv4.ttl;

ipv4.protocol;

ipv4.srcAddr;

ipv4.dstAddr;

}

field\_list\_calculation ipv4\_checksum {

input {

ipv4\_checksum\_list;

}

algorithm : csum16;

output\_width : 16;

}

calculated\_field ipv4.hdrChecksum {

verify ipv4\_checksum;

update ipv4\_checksum;

}

parser parse\_ipv4 {

extract(ipv4);

return ingress;

}

action \_drop() {

drop();

}

header\_type routing\_metadata\_t {

fields {

nhop\_ipv4 : 32;

}

}

metadata routing\_metadata\_t routing\_metadata;

action set\_nhop(nhop\_ipv4, port) {

modify\_field(routing\_metadata.nhop\_ipv4, nhop\_ipv4);

modify\_field(standard\_metadata.egress\_spec, port);

add\_to\_field(ipv4.ttl, -1);

}

table ipv4\_lpm {

reads {

ipv4.dstAddr : lpm;

}

actions {

set\_nhop;

\_drop;

}

size: 1024;

}

action set\_dmac(dmac) {

modify\_field(ethernet.dstAddr, dmac);

}

table forward {

reads {

routing\_metadata.nhop\_ipv4 : exact;

}

actions {

set\_dmac;

\_drop;

}

size: 512;

}

action rewrite\_mac(smac) {

modify\_field(ethernet.srcAddr, smac);

}

table send\_frame {

reads {

standard\_metadata.egress\_port: exact;

}

actions {

rewrite\_mac;

\_drop;

}

size: 256;

}

control ingress {

if(valid(ipv4) and ipv4.ttl > 0) {

apply(ipv4\_lpm);

apply(forward);

}

}

control egress {

apply(send\_frame);

}

Execute p4 program

\* Dependencies: bmv2, p4c-bm

1. Compile p4 program to .json file

p4c-bmv2 --json <path to JSON file> <path to P4 file>

2. Setup veth

[sudo] ./veth\_setup.sh

3. Using bmv2 backend to execute:

sudo ./simple\_switch -i 0@<iface0> -i 1@<iface1> <path to JSON file>

Using external libraries:

Install p4c

1. Compile

p4c-bm2-ss -o simple\_router.json simple\_router.p4

p4c [-b bmv2-ss-p4org] simple\_router.json simple\_router.p4 (output is a folder)

^ declare backend

Using p4app to compile