Relazione Esercitazione P4

Laboratory of Network Programmability and Automation

Riccardo Bacca
Filippo Benvenuti

Prof. Franco Callegati
Prof. Andrea Melis
Prof.ssa Chiara Contoli

12 giugno 2023

Linee guida

Lo scopo è creare una soluzione p4 che preveda:

- La possibilità di monitorare l'asimmetria tra flussi nel regolare traffico IPV4 senza header modificato;
- La possibilità di processare del traffico con un header custom che, tra i vari header necessari, contenga:
    1. Un campo *IP_Mal* che possa contenere un indirizzo ip (di default inizializzato a 0.0.0.0);
    2. Un camp *Time* che possa contenere un valore Unix Time (di default inizializzato a 0);
    3. Un campo *flag* che possa contenere un intero (inizializzato a 0).

Il programma deve quindi comportarsi come asymmetric flow nel caso generale con IPV4, quando però la treshold della differenza viene raggiunta, il programma deve:

- Registrare in un opportuno registro (chiamatelo *Threshold*) i dati relativi all'ultimo pacchetto che ha causato la treshold:
    1. Ip sorgente
    2. Ip destinazione
    3. Timestamp ultimo pacchetto
- Non droppare i pacchetti ma comunque continuare a forwardarli correttamente.

CONTEMPORANEAMENTE

Il programma deve essere in grado di processare i pacchetti del protocollo custom, che per quanto riguarda l'indirizzamento si comporta esattamente

come myTunnel (quindi la porta di destinazione è specificata con *--dst-id*) ma che abbia una funzionalità in più.

Ogni volta che viene processato un pacchetto myTunnel, prima di "accettarlo", controllare se nel registro *TRESHOLDI* definito precedentemente sia stato scritto un valore che segnala il raggiungimento della treshold per un determinato flusso. In caso positivo, prima di accettare il pacchetto, scrivere dei valori nell'header del pacchetto.

In particolare, la consegna a noi fornita prevedeva di:

- Scrivere nel campo IP_Mal la SORGENTE dell'ip che ha raggiunto la treshold;
- Scrivere nel campo flag il valore 1.

Soluzione proposta

La prima richiesta fornita nelle specifica è la seguente: *"La possibilità di monitorare..."*:

```
if (hdr.ipv4.isValid() && !hdr.myTunnel.isValid()) {
    ipv4_lpm.apply();
    bit<48> tmp;
    bit<32> flow;
    bit<32> flow_opp;
    compute_reg_index();
    bit<48> last_pkt_cnt;
    bit<48> last_pkt_cnt_opp;
    flow = meta.ingress_metadata.hashed_flow;
    flow_opp = meta.ingress_metadata.hashed_flow_opposite;
    /* Update and get packet count of flow */
    update_inter_packet_gap(last_pkt_cnt,flow);
    /* Get packet count of flow_opp */
    last_seen.read(last_pkt_cnt_opp,flow_opp);
    tmp = last_pkt_cnt - last_pkt_cnt_opp;
    /* If asymmetry hits the threshold */
    if(tmp == TRESHOLD) {
        bit<112> pkt_data = 0x0;
        pkt_data[31:0] = hdr.ipv4.srcAddr;
        pkt_data[63:32] = hdr.ipv4.dstAddr;
        pkt_data[111:64] = last_pkt_cnt;
        save_last_seen(pkt_data);
    }
```

```
action update_inter_packet_gap(out bit<48> pkt_count,bit<32> flow_id)
{
  bit<48> last_pkt_cnt;
  bit<32> index;
  /* Get the current count of packet in flow_id flow */
  last_seen.read(last_pkt_cnt,flow_id);
  pkt_count = last_pkt_cnt + 1;
  /* Update the register with the incremented count */
  last_seen.write((bit<32>)flow_id, pkt_count);
}
```

Con il primo If, viene verificato il corretto formato del traffico monitorato, escludendo quello con header modificato. Successivamente computiamo l'hash dei flow nelle due direzioni (da sorgente a destinazione e viceversa).

Tramite *update_inter_packet_gap* (...) leggiamo e aggiorniamo il conteggio dei pacchetti inoltrati tra sorgente e destinazione (incrementando *last_pkt_cnt),* tramite *last_seen.read(...),* leggiamo il conteggio dei pacchetti inoltrati tra destinazione e sorgente.

Nel campo *tmp,* inseriamo la differenza tra i due conteggi precedentemente descritti (asimmetria).

Successivamente, si richiedeva: "La possibilità di processare...":

```
header myTunnel_t {
    bit<16> proto_id;
    bit<16> dst_id;
    bit<32> IP_MAL;
    bit<48> TIME;
    bit<8> FLAG;
}
```

```
if (hdr.myTunnel.isValid()) {
    // process tunneled packets
    myTunnel_exact.apply();
}
```

L'header contiene i campi aggiuntivi che la consegna prevede:

- *IP_Mal;*
- *Time;*
- *Flag*.

L'if distingue la gestione dei pacchetti che contengono l'header custom, dal regolare traffico IPV4.

Inoltre, si richiedeva: "Il programma deve quindi comportarsi come asymmetric flow...":

```
/* If asymmetry hits the threshold */
if(tmp == TRESHOLD) {
    bit<112> pkt_data = 0x0;
    pkt_data[31:0] = hdr.ipv4.srcAddr;
    pkt_data[63:32] = hdr.ipv4.dstAddr;
    pkt_data[111:64] = standard_metadata.ingress_global_timestamp;
    save_last_seen(pkt_data);
}
```

```
action save_last_seen(bit<112> pkt_data) {
    TRESHOLDI.write(0, pkt_data);
}
```

Nel campo *tmp,* è inserita la differenza tra i due conteggi precedentemente descritti (asimmetria), quando questo raggiunge il *Threshold,* inseriamo nel campo *pkt_data,* i seguenti valori:

- Ip sorgente;
- Ip destinazione;
- *standard_metadata.ingress_global_timestamp* che rappresenta il timestamp del momento in cui il pacchetto è stato ricevuto;

Come indicato nelle specifiche.

Tramite *save_last_seen* salviamo nel registro *TRESHOLDI* i dati sopra elencati, come richiesto dalle specifiche.

Viene poi richiesto che: "Ogni volta che viene processato un pacchetto MyTunnel...":

```
table myTunnel_exact {
    key = {
        hdr.myTunnel.dst_id: exact;
    }
    actions = {
        myTunnel_forward;
        drop;
    }
    size = 1024;
    default_action = drop();
}
```

```
action myTunnel_forward(egressSpec_t port) {
    standard_metadata.egress_spec = port;
    bit<112> pkt_data = 0x0;
    TRESHOLDI.read(pkt_data, 0);
    if(pkt_data != 0){
        hdr.myTunnel.IP_MAL = pkt_data[31:0];
        hdr.myTunnel.FLAG = 1;
    }
}
```

Modifichiamo il forward del pacchetto custom, droppando quello originale, creandone uno custom nel cu header, se nel registro *TRESHOLDI* è inserito un valore, inseriamo i seguenti campi aggiuntivi:

- Ip sorgente nel campo IP_Mal;
- Il valore 1 nel campo flag;

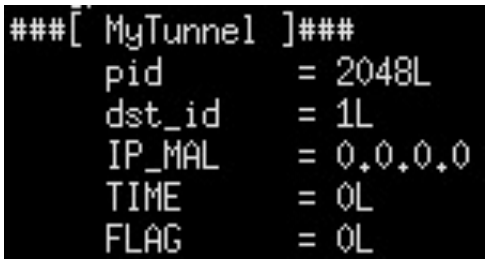
Come indicato da specifiche.

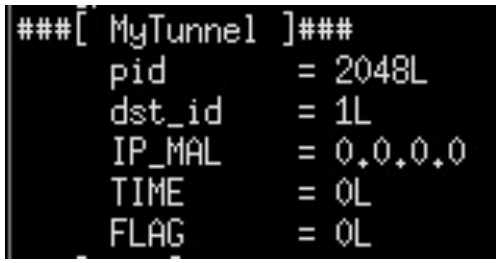
Test sul funzionamento

Per eseguire il codice fornito, utilizzare il comando: *make run*.
Successivamente ci troviamo sulla console di *Mininet,* in cui apriamo due terminali (relativi agli host H1 e H2) con il comando: *xterm h1 h2*.

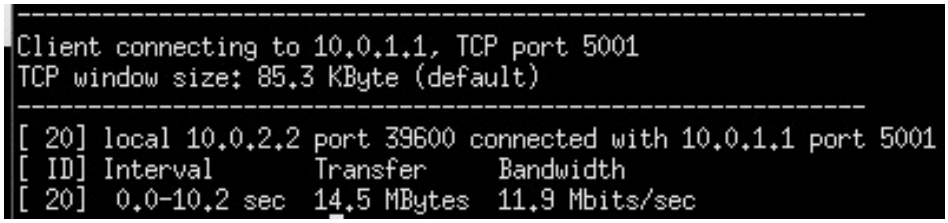
Procediamo come segue:

- Abilitiamo H1 a ricevere pacchetti custom tramite lo script python *mytunnel_receive.py;*
- H2 invia un pacchetto custom ad H1 tramite lo script python *mytunnel_send.py 10.0.1.1 "P4 is cool" —dst_id 1;*
- Verichiamo l'Header *MyTunnel* sia alla sorgente che alla destinazione. Notiamo che come da aspettative, l'Header non è stato modificato;

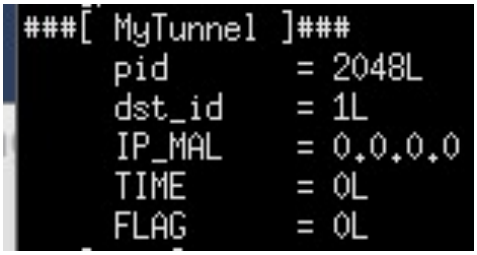


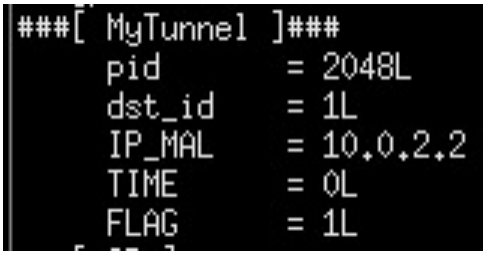


- Abilitiamo H1 a ricevere un flood di pacchetti tramite il comando: *iperf -s;*
- H2 manda un flood di pacchetti ad H1 tramite il comando: *iperf -c 10.0.1.1;*



- In questo modo l'asimmetria tra H1 e H2, ha sicuramente superato la soglia, quindi ri-eseguendo il test con gli script python, ci si aspetta che l'Header custom di *MyTunnel* venga modificato. DI seguito il risultato:





- Notiamo che l'header alla sorgente (lato sinistro), contiene i valori di default. Mentre l'header alla destinazione (lato destro) contiene i campi modificati come da specifiche;
- In questo modo abbiamo verificato che il forward si comporta come da aspettative.