

Laboratory of Network Programmability

Assignment P4

Andrea Serafini, Gustavo Mazzanti, Piero Sanchi

Giugno 2023

Indice

1	Requisiti	2
1.1	Requisitazione e Topologia di rete	2
2	Soluzione proposta	4
2.1	Asymmetric Flow	4
2.2	My Tunnel	5
3	Deployment	8
3.1	Step per l'esecuzione	8
4	Testing	9
4.1	Procedura per il testing	9

Capitolo 1

Requisiti

1.1 Requisitazione e Topologia di rete

L'assignment consiste nell'implementazione di una soluzione che combini i due comportamenti relativi ad "Asymmetric Flow" e "My Tunnel" sfruttando il linguaggio p4.

In particolare lo scopo dell'esame 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 customizzato (like My Tunnel) che, tra i vari header necessari, contenga:
 - Un campo **IP_Mal** che possa contenere un indirizzo ip (di default inizializzato a 0.0.0.0);
 - Un campo **TIME** che possa contenere un valore Unix Time (di default inizializzato a 0);
 - 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 threshold della differenza viene raggiunta il programma deve:

- Prima di tutto registrare in un opportuno registro (chiamatelo TRESHOLD) i dati relativi all'ultimo pacchetto che ha causato la threshold ovvero:
 - Ip sorgente
 - Ip destinazione
 - Timestamp ultimo pacchetto
- Secondo NON deve dropare i pacchetti ma deve 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 essere "accettato" (ovvero fare l'apply della tabella) viene controllato se nel registro TRESHOLD definito precedentemente è stato scritto un valore che segnala il raggiungimento della threshold per un determinato flusso. In caso positivo facciamo sì che prima di accettare il pacchetto, venga trascritto il valore nel campo dell'header corrispondente al pacchetto.

In particolare:

- Viene scritto nel campo **TIME** il timestamp del pacchetto che ha raggiunto la threshold
- Viene scritto nel campo **flag** il valore 1

Di seguito in figura 1.1 è possibile vedere la topologia utilizzata per la soluzione, composta da tre switch e tre host connessi tra di loro. La riga blu rappresenta il comportamento dei pacchetti My Tunnel, i quali non utilizzano l'intradamento classico dei pacchetti ipv4, ma vengono direttamente indirizzati verso una porta specifica definita nell'header.

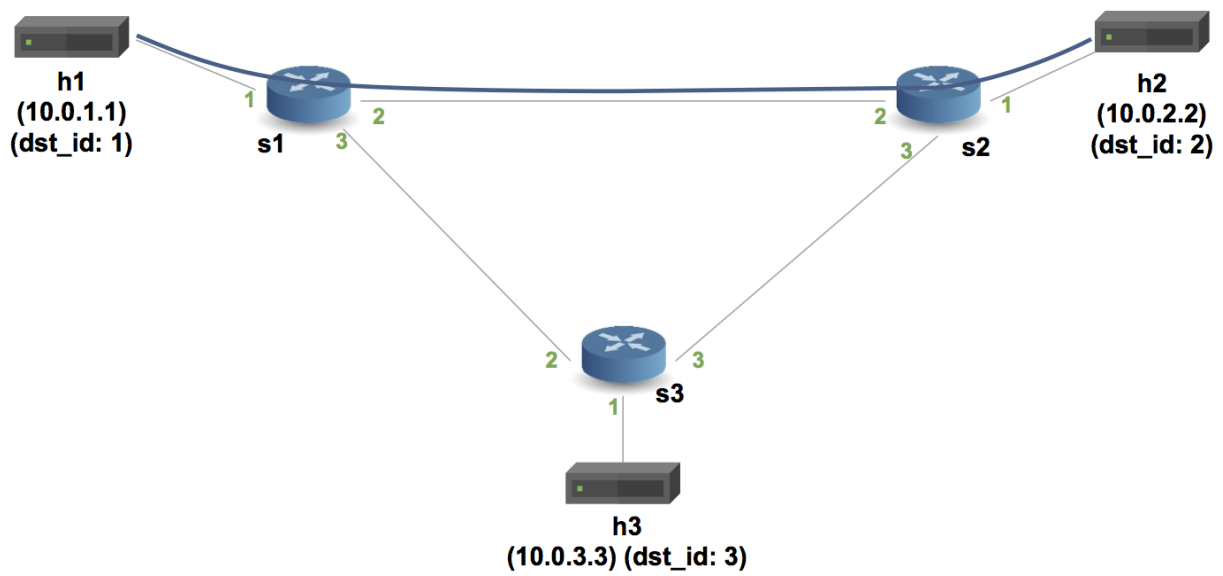


Figura 1.1: Topologia di rete utilizzata.

Capitolo 2

Soluzione proposta

2.1 Asymmetric Flow

Come punto di partenza della soluzione proposta per l'assignment abbiamo sfruttato i file relativi all'esercitazione relativa all' Asymmetric flow. Partendo dall'estratto del file seguente (Listing 2.1), abbiamo modificato il codice in modo tale da non scartare i pacchetti dopo che la TRESHOLD è stata superata, come richiesto dalle specifiche.

Listing 2.1: Old Asymmetric Flow

```
1
2 if (hdr.ipv4.isValid()) {
3     [...]
4     if(tmp < TRESHOLD) {
5         get_inter_packet_gap(last_pkt_cnt,flow);
6     }else{
7         drop();
8     }
9 }
```

Di seguito possiamo notare (Listing 2.2) che nel codice prodotto, anzichè richiamare il metodo **drop()**, dopo aver eseguito altre operazioni, facciamo sì che il pacchetto venga comunque inviato e consegnato.

Listing 2.2: New Asymmetric Flow

```
1 if (hdr.ipv4.isValid()) {
2     [...]
3     if(tmp < TRESHOLD) {
4         get_inter_packet_gap(last_pkt_cnt,flow);
5     }else{
6         [...]
7         get_inter_packet_gap(last_pkt_cnt,flow);
8     }
9 }
```

Per creare poi un'asimmetria nella comunicazione è stata rimossa la riga relativa all'host h1 dalla tabella dello switch s2, per il quale possiamo vedere il file dei comandi in 2.3, allo scopo di evitare che vengano ricevuti i pacchetti ack che impediscono di raggiungere la threshold impostata.

Listing 2.3: s2-commands.txt

```
1 table_set_default ipv4_lpm drop
2 table_add ipv4_lpm ipv4_forward 10.0.2.2/32 => 00:00:00:00:02:02 1
3 table_add ipv4_lpm ipv4_forward 10.0.3.3/32 => 00:00:00:03:03:00 3
```

E' stata poi abbassata la threshold a 4 per poter testare più facilmente il comportamento della rete.

2.2 My Tunnel

Per questa parte della soluzione abbiamo preso a modello l'esercitazione di laboratorio relativa a My Tunnel aggiungendo e modificando il codice come illustrato. Nel file presentato in 2.4 si possono vedere i nuovi campi aggiunti all'header per poter trasportare le informazioni relative all'ultimo pacchetto che ha superato la treshold.

Listing 2.4: myTunnel_header.py

```
1 [...]
2 class MyTunnel(Packet):
3     name = "MyTunnel"
4     fields_desc = [
5         SourceIPField("IP_Mal", 0),
6         IntField("TIME", 0),
7         ShortField("flag", 0),
8         ShortField("pid", 0),
9         ShortField("dst_id", 0)
10    ]
11 [...]
```

Qui di seguito 2.5 è riportato il file dei comandi dello switch s1 per mostrare la nuova tabella inserita contenente le informazioni necessarie per inoltrare i messaggi di tipo myTunnel indipendentemente dall'indirizzo ip mostrato.

Listing 2.5: s1-commands.txt

```
1 table_set_default ipv4_lpm drop
2 table_add ipv4_lpm ipv4_forward 10.0.1.1/32 => 00:00:00:00:01:01 1
3 table_add ipv4_lpm ipv4_forward 10.0.2.2/32 => 00:00:00:02:02:00 2
4 table_add ipv4_lpm ipv4_forward 10.0.2.2/32 => 00:00:00:02:02:00 3
5 table_add ipv4_lpm ipv4_forward 10.0.3.3/32 => 00:00:00:03:03:00 3
6
7 table_set_default myTunnel_exact drop
8 table_add myTunnel_exact myTunnel_forward 1 => 1
9 table_add myTunnel_exact myTunnel_forward 2 => 2
10 table_add myTunnel_exact myTunnel_forward 3 => 3
```

Nelle pagine seguenti è riportato invece in 2.6 e 2.7 il codice aggiunto al file p4 per la gestione dei pacchetti con header customizzato.

Listing 2.6: Implementazione utilities P4 per My Tunnel

```

1  /*HEADERS*/
2
3  header myTunnel_t {
4      bit<32> IP_Mal;
5      bit<32> TIME;
6      bit<16> flag;
7
8      bit<16> proto_id;
9      bit<16> dst_id;
10 }
11
12 struct headers {
13     ethernet_t ethernet;
14     myTunnel_t myTunnel;
15     ipv4_t ipv4;
16 }
17
18 /*PARSER*/
19
20 state parse_myTunnel {
21     packet.extract(hdr.myTunnel);
22     transition select(hdr.myTunnel.proto_id) {
23         TYPE_IPV4: parse_ipv4;
24         default: accept;
25     }
26 }
27
28 /*INGRESS PROCESSING*/
29
30 register<bit<32>>(3) treshold_reg;
31
32 action write_reg(bit<32> id, bit<32> val) {
33     /* Update the register with the new value */
34     bit<32> tmp;
35     tmp = val;
36     treshold_reg.write((bit<32>)id, tmp);
37 }
38
39 action myTunnel_forward(egressSpec_t port) {
40     standard_metadata.egress_spec = port;
41 }
42
43 table myTunnel_exact {
44     key = {
45         hdr.myTunnel.dst_id: exact;
46     }
47     actions = {
48         myTunnel_forward;
49         drop;
50     }
51     size = 1024;
52     default_action = drop();
53 }

```

Listing 2.7: Implementazione comportamento P4 per My Tunnel

```

1  /*INGRESS PROCESSING*/
2
3  apply {
4      if (hdr.ipv4.isValid() && !hdr.myTunnel.isValid()) {
5          [...]
6          if(tmp < TRESHOLD) {
7              get_inter_packet_gap(last_pkt_cnt,flow);
8          }else{
9              write_reg(0,hdr.ipv4.srcAddr);
10             write_reg(1,hdr.ipv4.dstAddr);
11             write_reg(2,(bit<32>)standard_metadata.ingress_global_timestamp);
12
13             get_inter_packet_gap(last_pkt_cnt,flow);
14         }
15     }
16
17     if (hdr.myTunnel.isValid()) {
18         // process tunneled packets
19         bit<32> srcip;
20         bit<32> dstip;
21         bit<32> time;
22
23         threshold_reg.read(srcip,0);
24         threshold_reg.read(dstip,1);
25         threshold_reg.read(time,2);
26
27         if(hdr.myTunnel.flag == 0 && time != 0) {
28             hdr.myTunnel.TIME = time;
29             hdr.myTunnel.flag = 1;
30         }
31         myTunnel_exact.apply();
32     }
33 }

```

La condizione `hdr.myTunnel.flag == 0` alla riga 27 dell'estratto di codice subito sopra è stata inserita per risolvere un problema non chiaro che avevamo introdotto nella logica della programma. Il problema si evidenziava attraverso un apparente errore di scrittura o lettura del registro `threshold_reg` che quindi non andava a riportare i valori attesi nell'header del pacchetto `myTunnel`. I pacchetti arrivavano all'host destinatario con il flag impostato ad 1, ma gli altri campi a 0, evidenziando quindi che il problema fosse nei valori del registro e non nell'header o nel pacchetto in sè. Utilizzando le CLI degli switch `s1` ed `s2` abbiamo monitorato lo stato dei registri e seguendo con `wireshark` il pacchetto nei suoi spostamenti siamo riusciti a risalire all'origine dell'errore.

Il pacchetto veniva inviato da `h1` con i valori (0.0.0.0, 0, 0), entrava in `s1`, il quale settava i valori a (10.0.1.1, 12345, 1) leggendo i primi due dal proprio registro in quanto, come possibile vedere in seguito al capitolo 4, dal suo punto di vista la comunicazione `h1-h2` ed `h2-h1` risulta asimmetrica e quindi la soglia raggiunta. Il pacchetto veniva poi inviato da `s1` in direzione di `h2`.

Il pacchetto però prima di arrivare a destinazione viene preso in ingresso dallo switch `s2`, il quale a differenza di `s1` percepisce i flussi `h1-h2` ed `h2-h1` come simmetrici, dato che `h2` invia sempre una risposta anche se poi però non viene inoltrata dallo switch. `S2` scrive quindi la propria versione dei valori del registro ovvero (0.0.0.0, 0, 1), creando di fatto un'incongruenza nella comunicazione, per poi terminare l'invio inoltrando ora il pacchetto ad `h2`.

La soluzione è stata quindi quella di effettuare delle verifiche prima di sovrascrivere un header, ovvero che nessun altro switch abbia in precedenza aggiunto i valori del proprio registro modificando quindi anche il flag, e che il registro contenga effettivamente delle informazioni prima di utilizzarle. Così facendo garantiamo che le informazioni su quale sia l'ultimo pacchetto ad aver superato il valore di soglia impostato vengano consegnate correttamente.

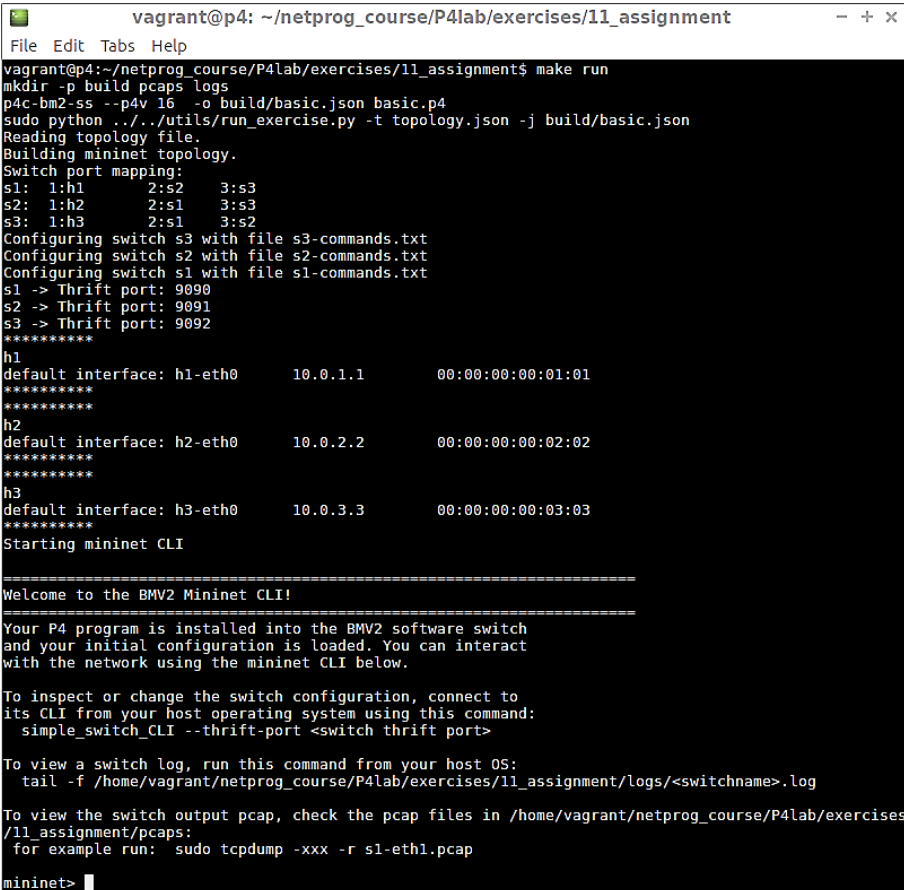
Capitolo 3

Deployment

3.1 Step per l'esecuzione

Comandi:

- Lanciare la topologia mininet con il comando `make run`



```
vagrant@p4: ~/netprog_course/P4lab/exercises/11_assignment
File Edit Tabs Help
vagrant@p4:~/netprog_course/P4lab/exercises/11_assignment$ make run
mkdir -p build pcaps logs
p4c-bm2-ss --p4v 16 -o build/basic.json basic.p4
sudo python ../../utils/run_exercise.py -t topology.json -j build/basic.json
Reading topology file.
Building mininet topology.
Switch port mapping:
s1: 1:h1 2:s2 3:s3
s2: 1:h2 2:s1 3:s3
s3: 1:h3 2:s1 3:s2
Configuring switch s3 with file s3-commands.txt
Configuring switch s2 with file s2-commands.txt
Configuring switch s1 with file s1-commands.txt
s1 -> Thrift port: 9090
s2 -> Thrift port: 9091
s3 -> Thrift port: 9092
*****
h1
default interface: h1-eth0 10.0.1.1 00:00:00:00:01:01
*****
h2
default interface: h2-eth0 10.0.2.2 00:00:00:00:02:02
*****
h3
default interface: h3-eth0 10.0.3.3 00:00:00:00:03:03
*****
Starting mininet CLI

=====
Welcome to the BMV2 Mininet CLI!
=====

Your P4 program is installed into the BMV2 software switch
and your initial configuration is loaded. You can interact
with the network using the mininet CLI below.

To inspect or change the switch configuration, connect to
its CLI from your host operating system using this command:
    simple_switch_CLI --thrift-port <switch thrift port>

To view a switch log, run this command from your host OS:
    tail -f /home/vagrant/netprog_course/P4lab/exercises/11_assignment/logs/<switchname>.log

To view the switch output pcap, check the pcap files in /home/vagrant/netprog_course/P4lab/exercises
/11_assignment/pcaps:
for example run: sudo tcpdump -xxx -r s1-eth1.pcap

mininet> █
```

Figura 3.1: Topologia avviata.

- Avviare i terminali degli host con i seguenti comando `xterm [h1, h2, h3]`
- Aprire il client dello switch con il comando `simple_switch_CLI --thrift-port 9090`
- Eseguire il comando `./receive.py` sul terminale di h2 e h3 per mettere in ascolto e mostrare i messaggi in arrivo;

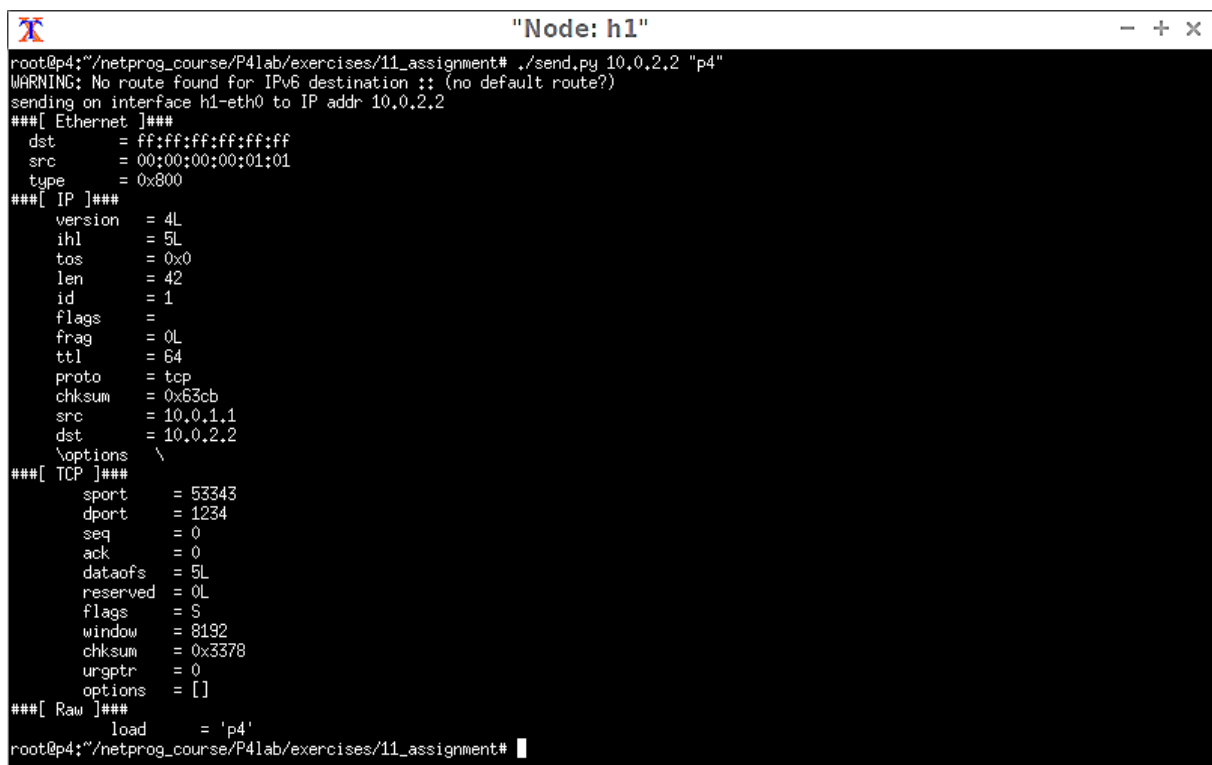
Capitolo 4

Testing

4.1 Procedura per il testing

Per testare il comportamento della rete si possono utilizzare i seguenti comandi:

- Sul terminale di h1 eseguire `./send.py 10.0.2.2 "p4"` per inviare un messaggio normale da h1 ad h2;



```
root@p4:~/netprog_course/P4lab/exercises/11_assignment# ./send.py 10.0.2.2 "p4"
WARNING: No route found for IPv6 destination :: (no default route?)
sending on interface h1-eth0 to IP addr 10.0.2.2
#### Ethernet ####
  dst      = ff:ff:ff:ff:ff:ff
  src      = 00:00:00:00:01:01
  type     = 0x800
#### IP ####
  version  = 4L
  ihl      = 5L
  tos      = 0x0
  len      = 42
  id       = 1
  flags    =
  frag     = 0L
  ttl      = 64
  proto    = tcp
  checksum = 0x63cb
  src      = 10.0.1.1
  dst      = 10.0.2.2
  options  = \
#### TCP ####
  sport    = 53343
  dport    = 1234
  seq      = 0
  ack      = 0
  dataofs  = 5L
  reserved = 0L
  flags    = S
  window   = 8192
  checksum = 0x3378
  urgptr   = 0
  options  = []
#### Raw ####
  load     = 'p4'
root@p4:~/netprog_course/P4lab/exercises/11_assignment#
```

Figura 4.1: Messaggio inviato da h1 a 10.0.2.2

```

Node: h2
root@p4:~/netprog_course/P4lab/exercises/11_assignment# ./receive.py
WARNING: No route found for IPv6 destination :: (no default route?)
sniffing on h2-eth0
got a packet
###[ Ethernet ]###
  dst      = 00:00:00:00:02:02
  src      = 00:00:00:02:02:00
  type     = 0x800
###[ IP ]###
  version  = 4L
  ihl      = 5L
  tos      = 0x0
  len      = 42
  id       = 1
  flags    =
  frag     = 0L
  ttl      = 62
  proto    = tcp
  chksum   = 0x65cb
  src      = 10.0.1.1
  dst      = 10.0.2.2
  \options \
###[ TCP ]###
  sport    = 53343
  dport    = 1234
  seq      = 0
  ack      = 0
  dataofs  = 5L
  reserved = 0L
  flags    = S
  window   = 8192
  chksum   = 0x3378
  urgptr   = 0
  options  = []
###[ Raw ]###
  load     = 'p4'

```

Figura 4.2: Messaggio ricevuto da h2

- Sul terminale di h1 eseguire `./send.py 10.0.2.2 "p4" --dst_id 2` per inviare un messaggio di tipo **My Tunnel** da h1 ad h2, verificare che venga trasmesso e che l'header relativo abbia tutti i campi settati a 0;

```

Node: h1
root@p4:~/netprog_course/P4lab/exercises/11_assignment# ./send.py 10.0.2.2 "p4" --dst_id 2
WARNING: No route found for IPv6 destination :: (no default route?)
sending on interface h1-eth0 to dst_id 2
###[ Ethernet ]###
  dst      = ff:ff:ff:ff:ff:ff
  src      = 00:00:00:00:01:01
  type     = 0x1212
###[ MyTunnel ]###
  IP_Mal   = 0.0.0.0
  TIME     = 0
  flag     = 0
  pid      = 2048
  dst_id   = 2
###[ IP ]###
  version  = 4L
  ihl      = 5L
  tos      = 0x0
  len      = 22
  id       = 1
  flags    =
  frag     = 0L
  ttl      = 64
  proto    = hopopt
  chksum   = 0x63e5
  src      = 10.0.1.1
  dst      = 10.0.2.2
  \options \
###[ Raw ]###
  load     = 'p4'
root@p4:~/netprog_course/P4lab/exercises/11_assignment#

```

Figura 4.3: Messaggio di tipo myTunnel da h1 a —dst_id 2

```

Node: h2
root@p4:~/netprog_course/P4lab/exercises/11_assignment# ./receive.py
WARNING: No route found for IPv6 destination :: (no default route?)
sniffing on h2-eth0
got a packet
#### Ethernet ####
  dst      = ff:ff:ff:ff:ff:ff
  src      = 00:00:00:00:01:01
  type     = 0x1212
#### MyTunnel ####
  IP_Mal   = 0.0.0.0
  TIME     = 0
  flag     = 0
  pid      = 2048
  dst_id   = 2
#### IP ####
  version  = 4L
  ihl      = 5L
  tos      = 0x0
  len      = 22
  id       = 1
  flags    =
  frag     = 0L
  ttl      = 64
  proto    = hopopt
  chksum   = 0x63e5
  src      = 10.0.1.1
  dst      = 10.0.2.2
  \options \
#### Raw ####
  load     = 'p4'

```

Figura 4.4: Messaggio ricevuto da h2 con i campi dell'header ai valori di inizializzazione

- Sul client dello switch s1 eseguire `register_read last_seen` per visualizzare il registro dello switch s1 (utilizzare lo stesso comando anche per accedere al registro degli altri switch);

Figura 4.5: Registro dello switch s1, evidenziato il pacchetto inviato in precedenza

- Sul client dello switch s1 eseguire `register_read threshold_reg` per visualizzare il registro dell'ultimo pacchetto trasmesso;

Figura 4.6: Registri dello switch s1, evidenziati due pacchetti sui flussi h1-h3 e h3-h1, inviati per mostrare il comportamento normale di una comunicazione simmetrica. In basso visibile il secondo registro inizializzato a 0

- [illegible]

14

- Inviare un nuovo pacchetto di tipo **My Tunnel** e verificare che ad h2 arrivi il pacchetto con i dati scritti nel registro;

```

root@p4:~/netprog_course/P4lab/exercises/11_assignment# ./receive.py
WARNING: No route found for IPv6 destination :: (no default route?)
sniffing on h2-eth0
got a packet
###[ Ethernet ]###
  dst      = ff:ff:ff:ff:ff:ff
  src      = 00:00:00:00:01:01
  type     = 0x1212
###[ MyTunnel ]###
  IP_Mal   = 10.0.1.1
  TIME     = 1156371675
  flag     = 1
  pid      = 2048
  dst_id   = 2
###[ IP ]###
  version  = 4L
  ihl      = 5L
  tos      = 0x0
  len      = 22
  id       = 1
  flags    =
  frag     = 0L
  ttl      = 64
  proto    = hopopt
  chksum   = 0xb3e5
  src      = 10.0.1.1
  dst      = 10.0.2.2
  \options \
###[ Raw ]###
  load     = 'p4'

```

Figura 4.8: Messaggio di tipo myTunnel ricevuto da h2 con header contenente i valori letti dal registro

- Eseguire il comando `exit` sul terminale di mininet per uscire dalla topologia, comando `make stop` per terminare l'esecuzione degli switch e degli host infine `make clean` per ripulire i file compilati.