

P4-progetto

Demetrio Andriani - Mat 10033010

Alessandro Tullio - Mat 1052295

Giugno 2023

Indice

1	Requisiti e Topologia di rete	4
2	Implementazione	5
2.1	Parser	6
2.2	Ingress	6
2.3	Deparser	10
3	Deployment	10
4	Testing	11

Elenco delle figure

1	Rappresentazione della topologia	5
2	Rappresentazione del Parser	6
3	Rappresentazione della gestione del traffico IPv4	7
4	Rappresentazione della gestione del traffico con header custom	8
5	Rappresentazione della gestione del traffico con header custom	9
6	Rappresentazione del deparser	10
7	Rappresentazione dei comandi per il deployment	10
8	Rappresentazione del traffico standard	11
9	Rappresentazione della generazione del traffico di rete tra due host	12
10	Rappresentazione del traffico con header custom	13

1 Requisiti e Topologia di rete

L'assignment consiste nell'implementazione di una soluzione che combini due componenenti in linguaggio P4, "Asymmetric Flow" e "My Tunnel". In particolare lo scopo del progetto è dare la possibilità di monitorare l'asimmetria tra flussi nel **REGOLARE** traffico ipv4 senza header modificato. Ma non solo, anche 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 (soglia) della differenza viene raggiunta il programma deve:

1. Prima di tutto registrare in un opportuno registro (chiamatelo TRESHOLDI) i dati relativi all'ultimo pacchetto che ha causato la threshold ovvero:
 - (a) Ip sorgente;
 - (b) Ip destinazione;
 - (c) Timestamp ultimo pacchetto;
2. Secondo NON deve dropare i pacchetti ma devo 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" (ovvero fare l'apply della tabella) si deve controllare se nel registro TRESHOLD definito precedentemente è stato scritto un valore che segnala il raggiungimento della threshold per un determinato flusso. IN CASO POSITIVO si deve, prima di accettare il pacchetto, scrivere dei valori nel campo dell'header corrispondente del pacchetto.

In particolare:

- Scrivere nel campo IP Mal la DESTINAZIONE dell'ip che ha raggiunto la threshold;
- Scrivere nel campo flag il valore 1;

A seguire l'immagine della topologia utilizzata per la soluzione, composta da 3 switch e 3 host.

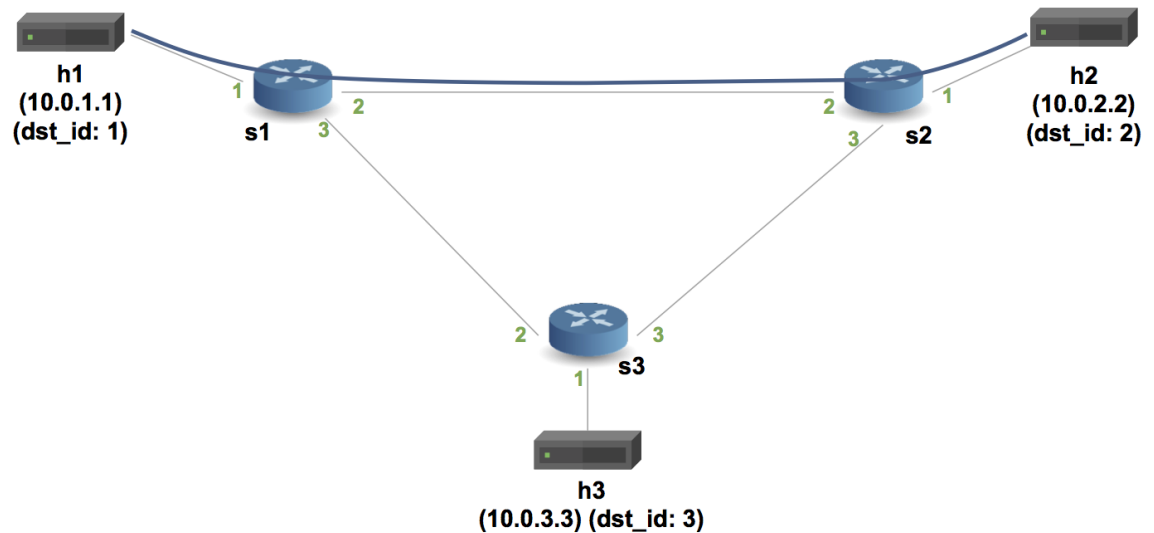


Figura 1: Rappresentazione della topologia

2 Implementazione

Abbiamo iniziato l'implementazione basandoci sull'esercitazione Asymmetric Flow, quindi con la possibilità di monitorare il flusso asimmetrico di dati inviati tra due host. Una volta che la THRESHOLD viene superata, invece di eseguire una drop di tutti i pacchetti ricevuti a seguire, semplicemente salveremo in un registro l'ultimo pacchetto che ha causato il superamento della threshold continuando a inoltrare, come da normale procedura, tutti i pacchetti che verranno ricevuti a seguire. In particolare, per il pacchetto che andrà a superare la soglia, andremo a salvare in un registro i seguenti dati, quali:

1. Ip sorgente;
2. Ip destinazione;
3. Timestamp;

Il registro, chiamato TRESHOLDI, sarà quindi composto da 3 campi, due di 32 bit e uno da 48, per un totale di 112 bit per scelta di ottimizzazione dello spazio.

2.1 Parser

Il parser è stato modificato per poter processare i pacchetti con header custom, in particolare, è stato aggiunto un nuovo stato per il parsing dell'header myTunnel, che contiene i campi aggiuntivi necessari per il tunneling.

```
103
104 parser MyParser(packet_in packet,
105                 out headers hdr,
106                 inout metadata meta,
107                 inout standard_metadata_t standard_metadata) {
108
109     state start {
110         transition parse_ethernet;
111     }
112
113     state parse_ethernet {
114         packet.extract(hdr.ethernet);
115         transition select(hdr.ethernet.etherType) {
116             TYPE_MYTUNNEL: parse_myTunnel;
117             TYPE_IPV4: parse_ipv4;
118             default: accept;
119         }
120     }
121
122     state parse_myTunnel {
123         packet.extract(hdr.myTunnel);
124         transition select(hdr.myTunnel.proto_id) {
125             TYPE_IPV4: parse_ipv4;
126             default: accept;
127         }
128     }
129
130     state parse_ipv4 {
131         packet.extract(hdr.ipv4);
132         transition accept;
133     }
134
135 }
136
```

Figura 2: Rappresentazione del Parser

2.2 Ingress

Prima abbiamo lavorato rispetto al traffico IPv4, successivamente rispetto ai pacchetti con header custom. Per il traffico IPv4, il programma deve essere in grado di processare i pacchetti in arrivo, contare il numero di pacchetti nei flussi in modo tale da poter calcolare la soglia di traffico e inoltrare i pacchetti in arrivo.

```

187     action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
188         standard_metadata.egress_spec = port;
189         hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
190         hdr.ethernet.dstAddr = dstAddr;
191         hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
192     }
193
194     table ipv4_lpm {
195         key = {
196             hdr.ipv4.dstAddr: lpm;
197         }
198         counters = c;
199         actions = {
200             ipv4_forward;
201             drop;
202             NoAction;
203         }
204         size = 1024;
205         default_action = NoAction();
206     }
207

```

Figura 3: Rappresentazione della gestione del traffico IPv4

Di default, il programma inoltra i pacchetti ricevuti, modificando opportunamente i campi dell'header ethernet e decrementando il TTL dell'header IPv4.

Il programma, una volta ricevuto un pacchetto, svolge un controllo sull'header, verificando se l'header IPv4 è valido e se non è presente l'header custom, in modo tale da poter processare il pacchetto. Dopo aver processato il pacchetto, viene computato il numero di pacchetti ricevuti da entrambi i versi del flusso, e se la differenza è maggiore della threshold, l'ultimo pacchetto ricevuto che ha causato il superamento della threshold viene salvato nel registro denominato **TRESHOLDI**. Come detto in precedenza il registro TRESHOLDI è l'unione di 3 campi, due da 32 bit per gli indirizzi e uno da 48 per il timestamp, per un totale di 112 bit, questo per questioni di ottimizzazione.

Dopo aver gestito il traffico IPv4, si deve andare a gestire il traffico avente header custom, dove il programma deve essere in grado di contare il numero di pacchetti nei flussi in modo tale da poter calcolare la soglia di traffico e inoltrare i pacchetti in arrivo.

```

209  ✓    action myTunnel_forward(egressSpec_t port) {
210      standard_metadata.egress_spec = port;
211      bit<112> pkt_data = 0x0;
212      TRESHOLDI.read(pkt_data, 0);
213  ✓    if(pkt_data != 0){
214        hdr.myTunnel.IP_MAL = pkt_data[63:32];
215        hdr.myTunnel.FLAG = 1;
216      }
217    }
218
219  ✓    table myTunnel_exact {
220  ✓      key = {
221        hdr.myTunnel.dst_id: exact;
222      }
223  ✓      actions = {
224        myTunnel_forward;
225        drop;
226      }
227      size = 1024;
228      default_action = drop();
229    }
230

```

Figura 4: Rappresentazione della gestione del traffico con header custom


```

231  ✓   apply {
232  ✓       if (hdr.ipv4.isValid() && !hdr.myTunnel.isValid()) {
233           ipv4_lpm.apply();
234           bit<48> tmp;
235           bit<32> flow;
236           bit<32> flow_opp;
237           compute_reg_index();
238           bit<48> last_pkt_cnt;
239           bit<48> last_pkt_cnt_opp;
240           /* Get the time the previous packet was seen */
241           flow = meta.ingress_metadata.hashed_flow;
242           flow_opp = meta.ingress_metadata.hashed_flow_opposite;
243           last_seen.read(last_pkt_cnt, flow);
244           last_seen.read(last_pkt_cnt_opp, flow_opp);
245           tmp = last_pkt_cnt - last_pkt_cnt_opp + 1;
246           get_inter_packet_gap(last_pkt_cnt, flow);
247  ✓       if (tmp == TRESHOLD) {
248           bit<112> pkt_data = 0x0;
249           pkt_data[31:0] = hdr.ipv4.srcAddr;
250           pkt_data[63:32] = hdr.ipv4.dstAddr;
251           pkt_data[111:64] = standard_metadata.ingress_global_timestamp;
252           save_last_seen(pkt_data);
253       }
254   }
255  ✓   if (hdr.myTunnel.isValid()) {
256       // process tunneled packets
257       myTunnel_exact.apply();
258   }
259 }
260 }

```

Figura 5: Rappresentazione della gestione del traffico con header custom

Il programma, una volta ricevuto un pacchetto, controlla la presenza dell'header custom ed eventualmente della sua validità, se lo è, viene processato il pacchetto. In particolare, viene letto il registro TRESHOLDI, e se il valore è diverso da 0 (quindi la soglia è stata raggiunta), viene inizializzato l'header custom con l'indirizzo IP del mittente del pacchetto che ha causato il superamento della soglia, e viene settato il flag a 1. Il pacchetto viene poi inoltrato, settando il campo "egress spec" con il valore della porta di uscita.

2.3 Deparser

Il deparser ha lo scopo di dover solo emettere i campi degli header che sono stati processati dal programma, nell'ordine corretto.

```
300  ✓ control MyDeparser(packet_out packet, in headers hdr) {
301  ✓      apply {
302          packet.emit(hdr.ethernet);
303          packet.emit(hdr.myTunnel);
304          packet.emit(hdr.ipv4);
305      }
306  }
```

Figura 6: Rappresentazione del deparser

3 Deployment

Per avviare il progetto si deve lanciare la topologia con il comando **make run**. Successivamente, avviare i terminali degli hosts con il comando **xterm h1 h2**.

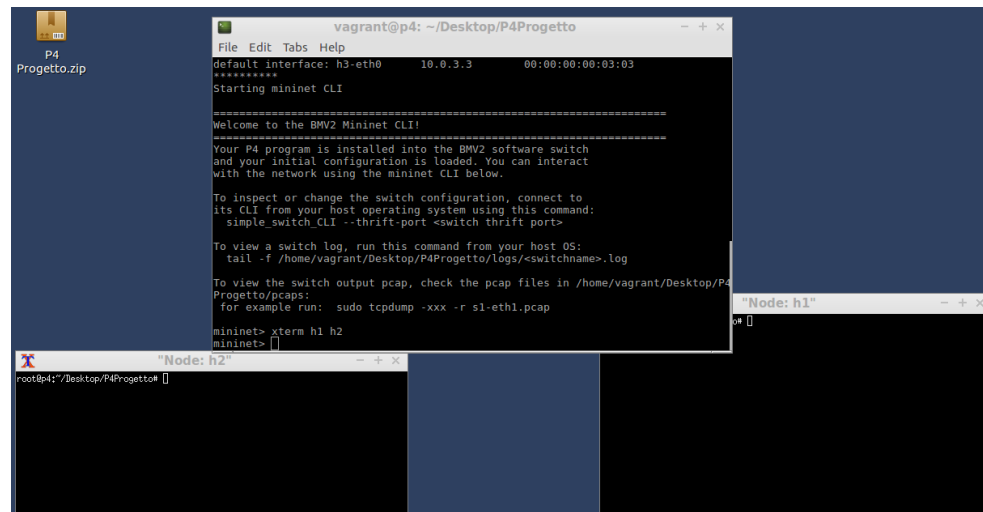
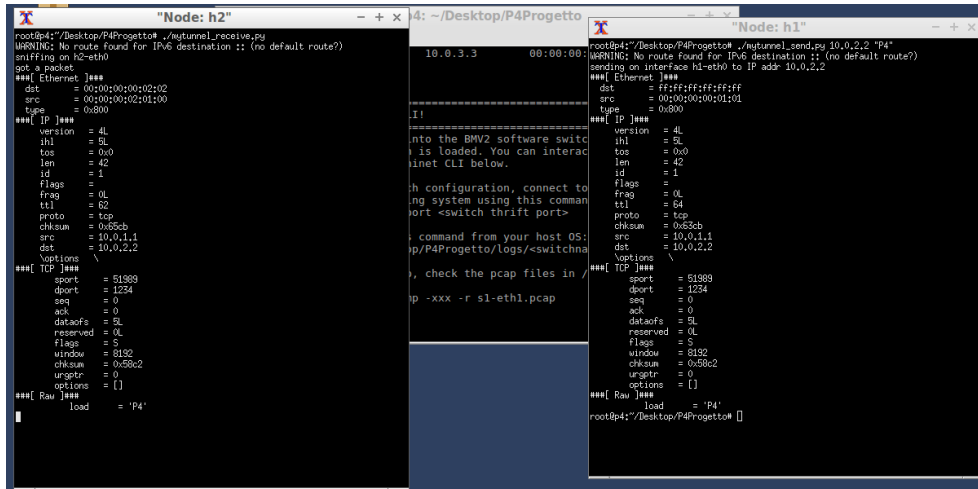


Figura 7: Rappresentazione dei comandi per il deployment

4 Testing

Per il testing, abbiamo nell'host 2 usato il comando `./mytunnel receive.py` per mettere l'host 2 in ascolto e mostrare i messaggi in arrivo. Sul terminale host 1 abbiamo usato il comando `./mytunnel send.py 10.0.2.2 "P4"` per inviare un messaggio standard tra host 1 e host 2.



The image shows two terminal windows side-by-side. The left window is titled "Node: h2" and the right window is titled "Node: h1". Both windows show the output of a Python script that captures and displays network packets. The left window shows a packet received from host 1, and the right window shows a packet sent to host 2. The packets are Ethernet II, IP, and TCP. The IP address in the packet from host 1 is 10.0.2.1, and the IP address in the packet to host 2 is 10.0.2.2. The TCP port is 51989. The data in the packet is "P4".

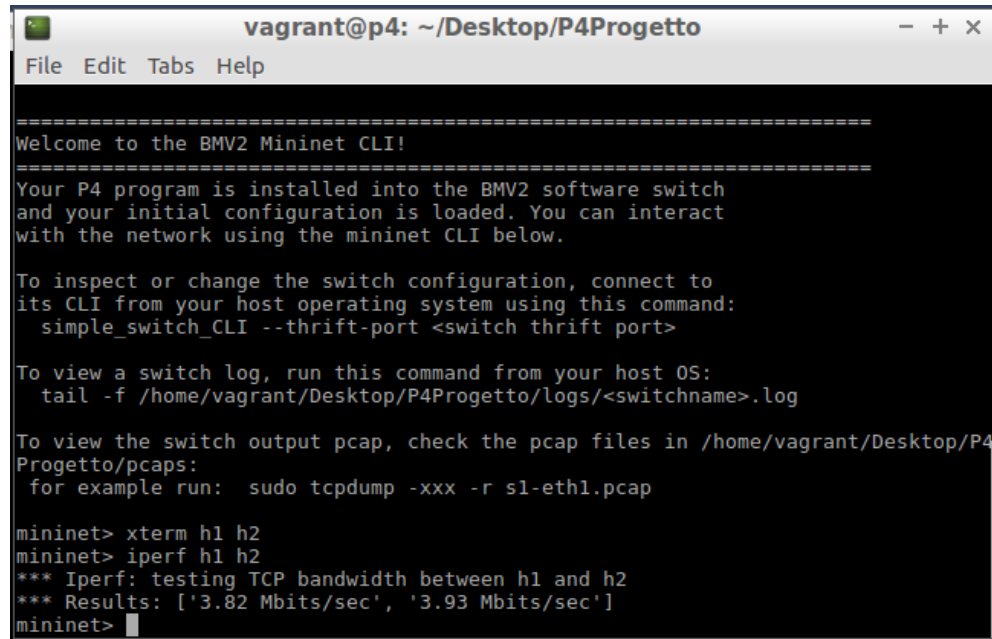
```
root@h2:~/Desktop/P4Progetto# ./mytunnel.receive.py
WARNING: No route found for IPv6 destination :: (no default route?)
sniffing on h2-eth0
got a packet
##### Ethernet II #####
  dst       = 00:00:00:00:00:02
  src       = 00:00:00:00:00:00
  type      = 0x800
##### IP #####
  version   = 4L
  ihl       = 5L
  tos       = 0x0
  len       = 42
  id        = 1
  flags     =
  frag      = 0L
  ttl       = 62
  proto     = tcp
  chksum    = 0x85cb
  src       = 10.0.1.1
  dst       = 10.0.2.2
  Options   \
##### TCP #####
  sport     = 51989
  dport     = 1234
  seq       = 0
  ack       = 0
  reserved  = 0L
  flags     = S
  window    = 65532
  chksum    = 0x55c2
  urgetr    = 0
  options   = []
##### Raw #####
  load      = 'P4'

root@h1:~/Desktop/P4Progetto# ./mytunnel.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 II #####
  dst       = ff:ff:ff:ff:ff:ff
  src       = 00:00:00:00:00:01
  type      = 0x800
##### IP #####
  version   = 4L
  ihl       = 5L
  tos       = 0x0
  len       = 42
  id        = 1
  flags     =
  frag      = 0L
  ttl       = 64
  proto     = tcp
  chksum    = 0x85cb
  src       = 10.0.1.1
  dst       = 10.0.2.2
  Options   \
##### TCP #####
  sport     = 51989
  dport     = 1234
  seq       = 0
  ack       = 0
  reserved  = 0L
  flags     = S
  window    = 65532
  chksum    = 0x55c2
  urgetr    = 0
  options   = []
##### Raw #####
  load      = 'P4'

root@h1:~/Desktop/P4Progetto#
```

Figura 8: Rappresentazione del traffico standard

Successivamente, con lo scopo di far scattare la THRESHOLD, abbiamo utilizzato il comando **iperf**, che ci permette di generare traffico di rete tra due host, come si vede in figura.



```
vagrant@p4: ~/Desktop/P4Progetto
File Edit Tabs Help

=====
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/Desktop/P4Progetto/logs/<switchname>.log

To view the switch output pcap, check the pcap files in /home/vagrant/Desktop/P4
Progetto/pcaps:
  for example run:  sudo tcpdump -xxx -r s1-eth1.pcap

mininet> xterm h1 h2
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['3.82 Mbits/sec', '3.93 Mbits/sec']
mininet>
```

Figura 9: Rappresentazione della generazione del traffico di rete tra due host

Una volta eseguiti i due comandi, la THRESHOLD verrà superata, quindi il pacchetto che ha causato il superamento verrà salvato nel registro THRESHOLDI, tutti i pacchetti successivi verranno inoltrati normalmente, mentre i pacchetti che usano il protocollo MyTunnel verranno modificati, in particolare il campo FLAG verrà settato a 1 e il campo IP MAL verrà settato con l'indirizzo IP di destinazione del pacchetto che ha causato il superamento della THRESHOLD. Per verificare le variazioni, è necessario inviare il pacchetto con header custom con il comando **./mytunnel send.py 10.0.2.2 "P4" -dst id 2**.

```

Node: h2
version = 4L
ihl = 5L
tos = 0x0
len = 22
id = 1
flags =
frag = 0L
ttl = 64
proto = hopopt
chksum = 0x63a5
src = 10.0.1.1
dst = 10.0.2.2
options \
load = 'P4'
got a packet
*** Ethernet ***
dst = ffffffff:ffff:ff
src = 00:00:00:00:01:01
type = 0x1212
*** MtuTunnel ***
pid = 2048L
dst_id = 2L
IP_MAL = 10.0.2.2
TIME = 0L
FLAG = 1L
*** IP ***
version = 4L
ihl = 5L
tos = 0x0
len = 22
id = 1
flags =
frag = 0L
ttl = 64
proto = hopopt
chksum = 0x63a5
src = 10.0.1.1
dst = 10.0.2.2
options \
load = 'P4'

Node: h1
id = 1
flags =
frag = 0L
ttl = 64
proto = hopopt
chksum = 0x63a5
src = 10.0.1.1
dst = 10.0.2.2
options \
load = 'P4'
root@h1: /Desktop/P4Progetto# ./mtunnel_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 = ffffffff:ffff:ff
src = 00:00:00:00:01:01
type = 0x1212
*** MtuTunnel ***
pid = 2048L
dst_id = 2L
IP_MAL = 0.0.0.0
TIME = 0L
FLAG = 0L
*** IP ***
version = 4L
ihl = 5L
tos = 0x0
len = 22
id = 1
flags =
frag = 0L
ttl = 64
proto = hopopt
chksum = 0x63a5
src = 10.0.1.1
dst = 10.0.2.2
options \
load = 'P4'
root@h1: /Desktop/P4Progetto#

```

Figura 10: Rappresentazione del traffico con header custom

Confrontando il pacchetto ricevuto con quello inviato, possiamo notare che il campo FLAG è stato settato a 1, mentre il campo IPMAL è stato settato con l'indirizzo IP di destinazione del pacchetto che ha causato il superamento della threshold, quindi l'header custom è stato modificato correttamente, in relazione alle specifiche date.