

Relazione Progetto Mininet

Di Gaudio Antonio 0715598



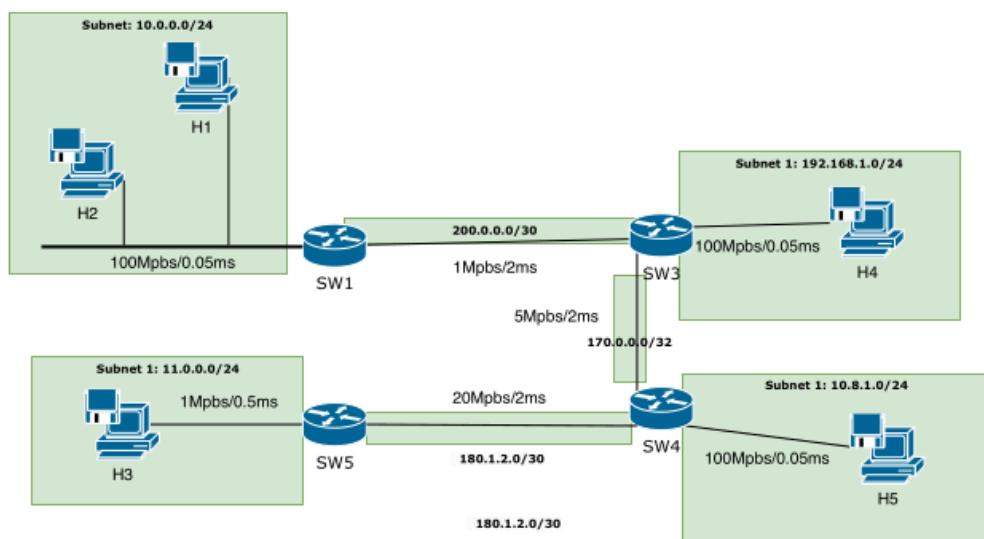
**Università
degli Studi
di Palermo**

Relazione Progetto Mininet	1
Introduzione	3
Obiettivo del progetto:	3
Tecnologie utilizzate:	3
Analisi e Progettazione	4
Obiettivi di rete:	4
Descrizione della topologia di rete:	4
Comunicazione tra i nodi:	5
Comunicazione tramite le API REST (su H1):	5
Dettagli di comunicazione:	6
Link e parametri di comunicazione:	6
Implementazione del Sistema	7
Creazione della rete con Mininet:	7
Integrazione di flask e API REST:	7
Avvio e gestione di iperf sui nodi:	7
Analisi dei test	8
Specifiche e versioni degli strumenti utilizzati	13
Versione Virtual-Box:	13
Macchina host:	13
Macchina guest:	13
Versioni strumenti principali:	14
Pip freeze:	14
Eseguire la rete	15
Flusso attivazione rete:	15
Problemi riscontrati	16
Problema con UDP:	16
Soluzione del problema con UDP:	16
Possibili troubleshooting	17
Possibili sottoprocessi in esecuzione?	17
Possibili problemi con PID occupato?	17

Introduzione

Obiettivo del progetto:

Progettare ed implementare un sistema di comunicazione che emuli una rete composta da switch, router e host. In particolare, si faccia uso di Mininet per la realizzazione della rete. La topologia di rete richiesta è composta da n. 4 nodi che operano come router L3 e n. 5 host. I nodi di rete sono collegati tra loro come mostrato in figura:



Tecnologie utilizzate:

Mininet: Strumento di emulazione di reti.

Flask: Gestione del server HTTP e delle API REST.

Iperf: Testare le prestazioni di rete tra gli host.

Analisi e Progettazione

Obiettivi di rete:

L'obiettivo del progetto è creare una rete nella quale tutti gli host sono interconnessi con indirizzamenti IP configurati secondo le sottoreti fornite, e con link emulati che rispettano specifici parametri di banda e latenza. Inoltre, su uno degli host (H1) è necessario implementare un server HTTP con Flask, che espone delle API REST per gestire esperimenti di rete tramite il tool "iperf", permettendo di avviare e fermare il traffico tra i nodi, personalizzando i parametri come l'indirizzo di destinazione, il protocollo e la velocità di trasmissione.

Descrizione della topologia di rete:

Il sistema è composto da una topologia di rete che include n. 4 router e n. 5 host, interconnessi tramite switch e collegamenti di rete. Ogni nodo della rete è configurato con un indirizzo IP statico e appartiene a una specifica sottorete.

Host (H1, H2, H3, H4, H5)

Ogni host (o al massimo una coppia di host) è connesso a uno switch e ha un indirizzo IP specifico, come indicato nello schema della rete.

Switch (SW1, SW3, SW4, SW5)

Gli switch sono utilizzati per instradare il traffico tra gli host e i router.

Router (R1, R2, R3, R4)

I router gestiscono il traffico tra le diverse sottoreti e sono responsabili del forwarding dei pacchetti tra le sottoreti.

Comunicazione tra i nodi:

La comunicazione tra i vari componenti della rete avviene seguendo i seguenti passaggi:

Host (H1) e Router

Ogni host è configurato con una rotta predefinita che punta al router associato.

Router e Router

I router sono interconnessi tramite link point-to-point. Essi scambiano pacchetti per il forwarding tra le diverse sottoreti. Ogni router è configurato con rotte statiche per raggiungere le altre sottoreti.

Switch e Host

Gli switch instradano il traffico tra gli host collegati. Ogni host può comunicare con altri host tramite il relativo switch.

Comunicazione tramite le API REST (su H1):

Il server Flask è configurato su H1 e fornisce delle API REST per l'interazione tra i nodi.

I comandi principali gestiti dalle API sono:

- **/start_iperf:** Questo endpoint avvia un esperimento di rete utilizzando iperf. Riceve come input l'indirizzo IP di destinazione (*IP_DEST*), il protocollo di trasporto (*L4_proto*, che può essere TCP o UDP) e il tasso di trasmissione (*src_rate*).

Funzionamento: Quando un nodo invia una richiesta a */start_iperf*, il server Flask su H1 avvia il comando iperf sul nodo sorgente (host H1), avviando un flusso di traffico verso il nodo di destinazione con i parametri specificati.

- **/stop_iperf:** Questo endpoint permette di fermare qualsiasi processo di iperf in corso sui nodi client.

Funzionamento: Quando un nodo invia una richiesta a */stop_iperf*, il server Flask su H1 invia un comando a tutti i nodi della rete per fermare eventuali processi iperf in esecuzione.

Dettagli di comunicazione:

La comunicazione tra i vari componenti della rete si articola nei seguenti passaggi:

Comunicazione tra Host e Server (Flask su H1)

Ogni nodo della rete può inviare richieste HTTP al server Flask su H1 (anche H1 stesso), utilizzando le API REST per avviare o fermare il traffico di rete tramite iperf.

L'host mittente invia una richiesta POST al server Flask su H1, includendo i parametri necessari (IP di destinazione, protocollo L4, e rate di trasmissione).

Il server Flask interpreta la richiesta e avvia iperf sul nodo di origine, configurandolo secondo i parametri inviati.

Comunicazione tra Router e Switch

I router comunicano tra loro per instradare correttamente i pacchetti tra le sottoreti.

Ogni router è configurato con una tabella di routing che definisce come inoltrare i pacchetti verso le destinazioni correttamente. Le tabelle di routing sono statiche e preconfigurate, quindi ogni router conosce come raggiungere le altre sottoreti.

Link e parametri di comunicazione:

I link tra i nodi sono emulati da Mininet con i seguenti parametri:

Latenza e Banda

Ogni link tra i nodi (tra host, switch e router) ha una specifica banda e uno specifico ritardo di propagazione come definito nel progetto.

Configurazione della Rete

I nodi sono configurati per usare le sottoreti specifiche e ogni link ha i parametri di propagazione corretti per emulare la rete fisica.

Implementazione del Sistema

Creazione della rete con Mininet:

La creazione della rete ha parametri statici e segue le specifiche richieste nel progetto assegnato

Integrazione di flask e API REST:

Creazione del server HTTP su H1 e configurazione delle API REST per avviare e fermare i test di iperf.

I comandi `/start_iperf` e `/stop_iperf` sono il core del client per gestire i processi Iperf.

I parametri vengono validati dal protocollo (TCP/UDP) e della velocità di invio (`src_rate`).

Avvio e gestione di iperf sui nodi:

Quando si preme “start” e si seleziona TCP con protocollo di livello 4, iperf parte su tutti i nodi in modalità TCP, altrimenti se viene cliccato UDP farà un restart dei server per accettare il protocollo UDP.

I log nel CSV si salveranno con il seguente formato, esempio:

“20241128113057,10.1.1.1,5001,192.168.1.2,56178,4,0.0-12.0
,1441792,957762”

Ma in output (nella dashboard), l’output viene reso “*Human readable*” con un dizionario facile da interpretare.

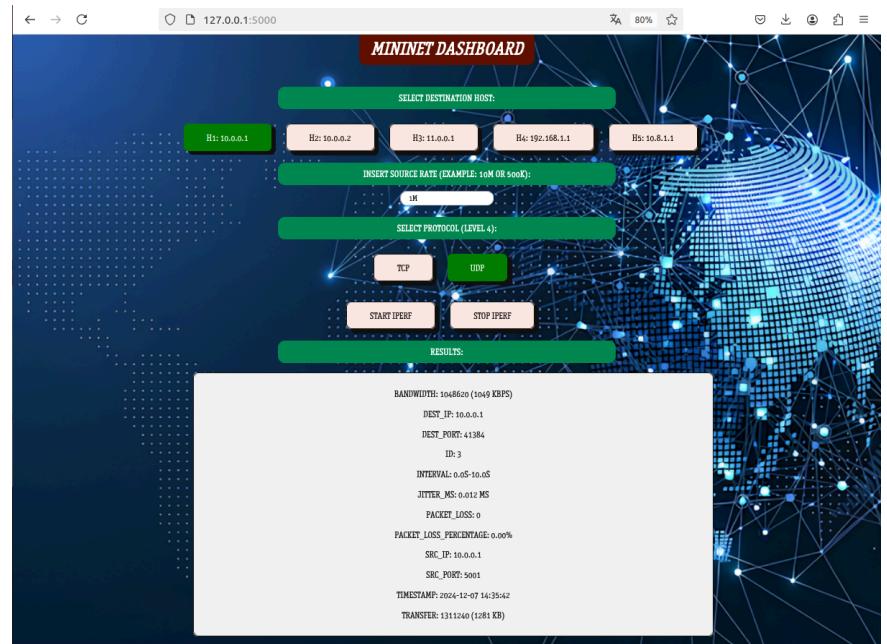
Analisi dei test

Sono tutti eseguiti con Source Rate di 1M, da H1 verso tutti gli host, sia in TCP che UDP e il conseguente salvataggio nei file di log. Mostro per ogni singolo host, il servizio iperf in TCP, UDP e il salvataggio nel file di log in CSV.

H1-TCP



H1-UDP



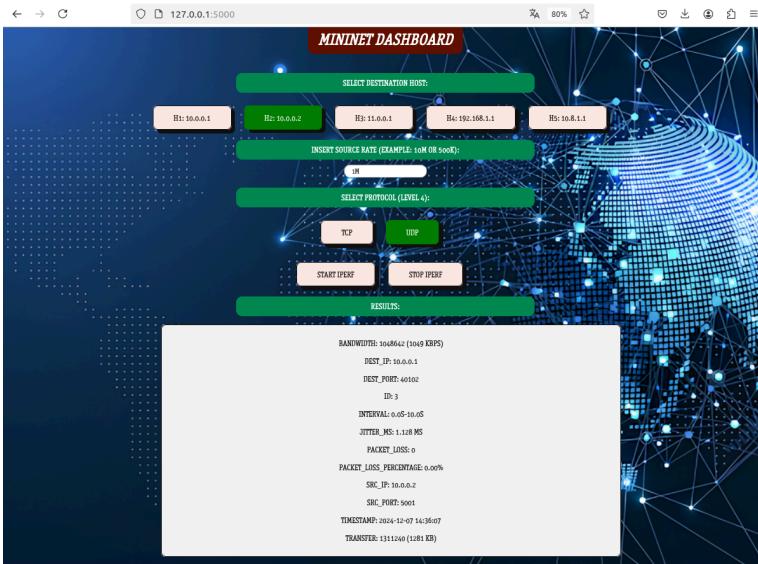
H1-LOG

h1_log.csv	
1	20241207143417,10.0.0.1,5001,10.0.0.1,41266,6,0.0-10.0,1441792,1153208
2	20241207143542,10.0.0.1,5001,10.0.0.1,41384,5,0.0-10.0,1311240,1048620,0.012,0,892,0.000,0
3	

H2-TCP



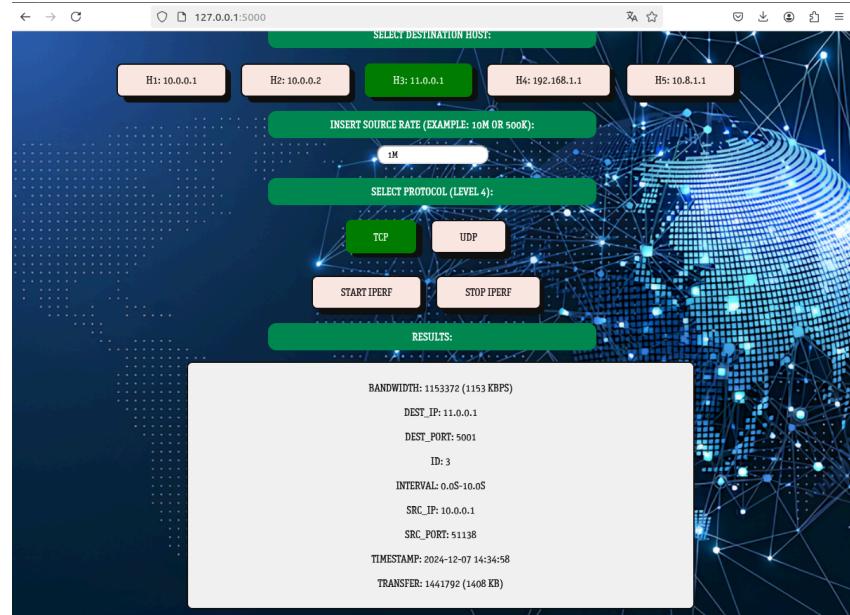
H2-UDP



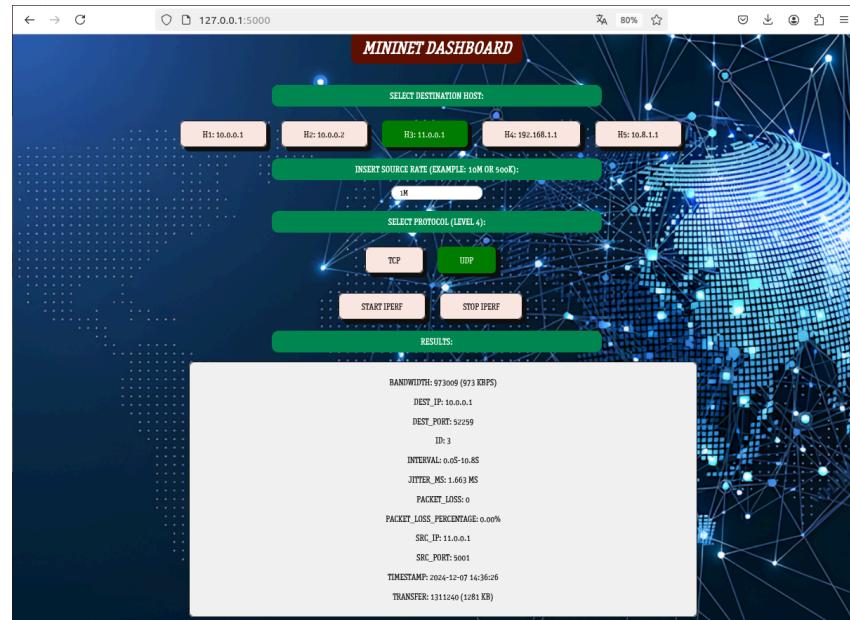
H2-LOG

```
h2_log.csv
1 20241207143444,10.0.0.2,5001,10.0.0.1,48540,6,0.0-10.0,1441792,1151936
2 20241207143607,10.0.0.2,5001,10.0.0.1,40102,5,0.0-10.0,1311240,1048642,1.129,0.892,0.000,0
3
```

H3-TCP



H3-UDP



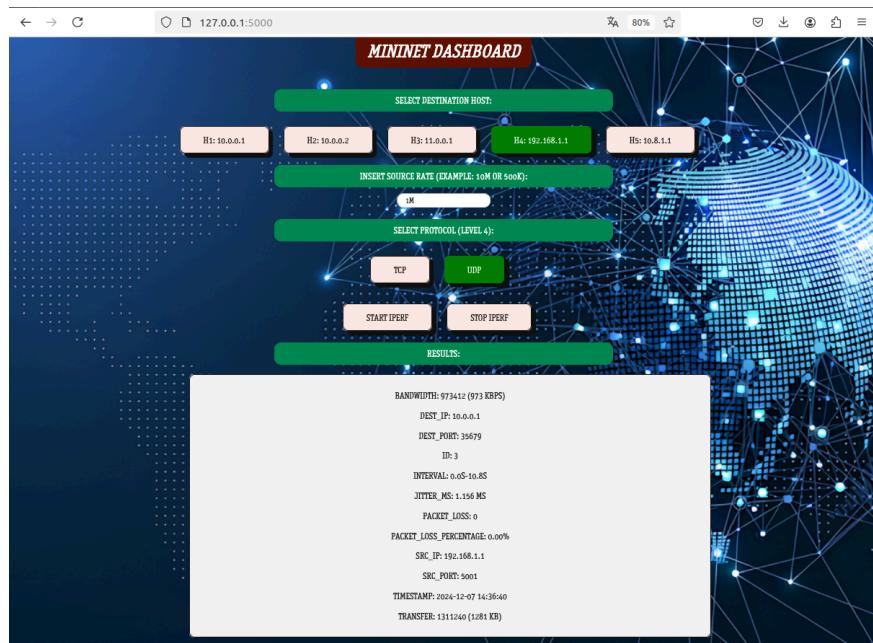
H3-LOG

```
h3_log.csv
1 20241207143500,11.0.0.1,5001,10.0.0.1,51138,6,0.0-12.0,1441792,957675
2 20241207143626,11.0.0.1,5001,10.0.0.1,52259,5,0.0-10.8,1311240,973009,1.664,0.892,0.000,0
3
```

H4-TCP



H4-UDP



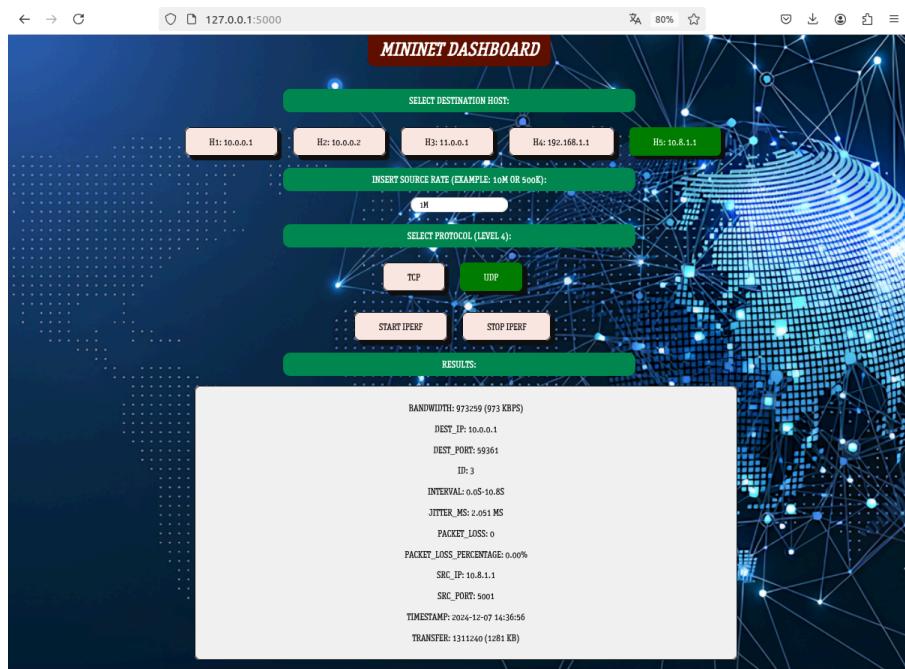
H4-LOG

```
h4_log.csv
1 20241207134039,192.168.1.1,5001,10.0.0.1,42754,5,0.0-10.0,640920,513694,1.496,0,436,0.000,0
2 20241207141902,192.168.1.1,5001,10.0.0.1,46604,6,0.0-10.0,655360,524013
3 20241207141917,192.168.1.1,5001,10.0.0.1,46979,5,0.0-10.0,640920,511997,2.653,0,436,0.000,0
4 20241207143513,192.168.1.1,5001,10.0.0.1,36588,6,0.0-12.0,1441792,957804
5 20241207143640,192.168.1.1,5001,10.0.0.1,35679,5,0.0-10.8,1311240,973412,1.156,0,892,0.000,0
6
```

H5-TCP



H5-UDP



H5-LOG

```
h5_log.csv
1 20241207143529,10.8.1.1,5001,10.0.0.1,46036,6,0.0-12.0,1441792,957590
2 20241207143656,10.8.1.1,5001,10.0.0.1,59361,5,0.0-10.8,1311240,973259,2.052,0.892,0.000,0
3
```

Specifiche e versioni degli strumenti utilizzati

Versione Virtual-Box:

```
(base) antonio@antonio:~$ VBoxManage --version  
7.0.16_Ubuntur162802
```

Macchina host:

```
(base) antonio@antonio:~$ lsb_release -a  
Distributor ID: Ubuntu  
Description:    Ubuntu 24.04.1 LTS  
Release:        24.04  
Codename:       noble
```

Macchina guest:

```
(base) virtualbox@virtualbox:~$ lsb_release -a  
Distributor ID: Ubuntu  
Description:    Ubuntu 20.04.6 LTS  
Release:        20.04  
Codename:       focal
```

Versioni strumenti principali:

Python3

```
● ● ●  
(base) virtualbox@virtualbox:~$ python3 --version  
Python 3.8.10
```

Iperf

```
● ● ●  
(base) virtualbox@virtualbox:~$ iperf -version  
iperf version 2.0.13 (21 Jan 2019) pthreads
```

Mininet

```
● ● ●  
(base) virtualbox@virtualbox:~$ mn --version  
2.3.1b4
```

Pip freeze:

Le dipendenze e gli gli import si trovano nel file requirements.txt.

Eseguire la rete

Flusso attivazione rete:

Si devono aprire due terminali, uno per il controller RYU e uno per avviare la rete Mininet.

1° Terminale:

Vai in `ryu/ryu/app` e dai:

```
$ryu-manager simple_switch_13.py
```



2° Terminale:



Adesso il controller è configurato, la rete è configurata, e ci possiamo collegare a:

127.0.0.1:5000 per avere accesso alla dashboard della rete Mininet.

Problemi riscontrati

Problema con UDP :

In Mininet, con l'uso delle API REST prima di avviare gli esperimenti con iperf si deve scegliere il protocollo (TCP o UDP). Durante la fase di progettazione ho riscontrato problemi riguardo l'avvio di iperf sia in TCP che UDP, e ho notato che non è possibile cambiare protocollo al volo senza riavviare il processo di iperf, perché il protocollo è definito al momento dell'avvio del client(in questo caso).

Soluzione del problema con UDP :

Ho pensato a due soluzioni:

Far partire iperf con due processi separati

Questa opzione l'ho scartata per evitare un elevato uso di sottoprocessi

Riavviare iperf al cambio di protocollo

Questa è l'opzione che ho scelto per risolvere il problema con UDP. Di conseguenza ho definito un funzione /restart_iperf, che ogni volta che il sistema vede il cambio di protocollo, quando viene avviato l'esperimento viene fatto un riavvio di iperf con il protocollo selezionato.

Possibili troubleshooting

Possibili sottoprocessi in esecuzione?

È consigliato, prima di attivare il flusso della rete Mininet, dare il seguente comando per eliminare sottoprocessi che possono creare conflitti con la rete stessa.



A screenshot of a terminal window with a dark background and three colored dots (red, yellow, green) at the top. The text inside the terminal reads:

```
(base) virtualbox@virtualbox:~/progetto/mininet$ sudo mn -c
```

Possibili problemi con PID occupato?

Dare il seguente comando per avere il PID del processo in esecuzione che occupa la porta.

```
$ sudo lsof -t -i:<#Porta>
```

Dopo aver ottenuto il PID possiamo dare il comando:

```
$ sudo kill -9 PID
```