LABORATORIO DI RETI E SISTEMI DISTRIBUITI

HANDS-ON 7

Interfacce virtuali: indirizzamento diretto e indiretto

Antonio Mastrolembo Ventura

543644

March 30, 2025

Indice

1 Introduzione **3**

2 Definizione del problema **3**

3 Esercizio 1: Indirizzamento diretto **4**

3.1 Introduzione 4

3.2 Metodologia e risultati 4

3.2.1 Creazione dell'infrastruttura 4

3.2.2 Verifica degli host 4

3.2.3 Sessioni xterm 5

3.2.4 Connessione TCP 5

3.2.5 Analisi del traffico con Wireshark 6

3.3 Conclusioni 7

4 Esercizio 2: Indirizzamento indiretto **8**

4.1 Introduzione 8

4.2 Metodologia e risultati 8

4.2.1 Librerie 8

4.2.2 Classe del Router 9

4.2.3 Topologia - Router 9

4.2.4 Topologia - Host 10

4.2.5 Topologia - Link e interfacce 10

4.2.6 Creazione e avvio della rete 11

4.2.7 Verifica dell'infrastruttura 12

4.3 Conclusioni 13

5 Esercizio 3: Physical Constraints **14**

5.1 Introduzione 14

5.2 Metodologia e risultati 14

5.2.1 Libreria 14

5.2.2 Vincoli 14
5.2.3 Test - ping 14
5.2.4 Test - iperf 15
5.3 Conclusioni 16

1 Introduzione

In questo hands-on andremo a trattare le **interfacce virtuali** di rete attraverso l'utilizzo di un framework chiamato **mininet**. Le interfacce virtuali di rete sono dispositivi di rete emulati che operano interamente nel software, senza essere direttamente legati a un'interfaccia fisica. In sostanza un'interfaccia virtuale di rete non ha un corrispettivo fisico ma permette comunque di inviare e ricevere pacchetti dati. Sono generalmente utilizzate per il testing, le VPN o il tunneling. Su Linux, per esempio, un tipo di interfacce virtuali sono le **TUN/TAP** che operano a livello 3 e 2 dello stack OSI (utilizzate per tunnel VPN o per collegare macchine virtuali a reti fisiche ad esempio) il cui device file si trova al percorso

```
$ cd /dev/net/tun
```

Un altro tipo di interfacce virtuali sono le **Veth** (Virtual Ethernet) **Pair**, utilizzate da Docker per connettere i container alla rete dell'host. Una veth pair è una coppia di interfacce virtuali collegate tra loro, dove ogni pacchetto inviato su un'estremità appare immediatamente sull'altra. Su Docker, un'estremità di questa coppia viene assegnata al namespace di rete del container, mentre l'altra al bridge di rete dell'host, in questo modo i container possono comunicare fra di loro e con l'esterno.

2 Definizione del problema

Dopo una breve introduzione sulle interfacce virtuali, l'hands-on prevede la realizzazione di due piccole infrastrutture di rete attraverso l'utilizzo di mininet. Mininet è un framework che consente di creare e simulare reti virtuali su una singola macchina, permette di emulare host, switch e collegamenti di rete testando varie configurazioni. In questo hands-on realizzeremo due infrastrutture differenti: una formata da due host interconnessi da uno switch ed una formata da due host ed un router.

3 Esercizio 1: Indirizzamento diretto

3.1 Introduzione

Come anticipato poc'anzi, il primo esercizio si basa sulla creazione di una piccola infrastruttura formata da due host interconnessi da uno switch. Poiché lo switch opera a livello 2 dello stack OSI, entrambi gli host si trovano nella stessa rete locale (LAN) e possono comunicare direttamente senza bisogno di un router. Per creare l'infrastruttura attraverso Mininet abbiamo diverse possibilità: possiamo scrivere uno script Python utilizzando la libreria Mininet per definire esplicitamente i nodi e le connessioni, oppure, per questo esercizio, possiamo utilizzare direttamente il comando mn, che crea automaticamente una topologia predefinita con due host, uno switch e i relativi collegamenti.

3.2 Metodologia e risultati

Per questo primo esercizio utilizziamo il comando `mn`, vedremo più avanti una manipolazione dell'infrastruttura migliore attraverso il linguaggio python.

3.2.1 Creazione dell'infrastruttura

Il primo comando da eseguire è:

```
$ sudo mn
```

Attraverso questo comando, Mininet avvia un ambiente di rete predefinito formato da:

- Due host (h1 e h2), ciascuno con un'interfaccia di rete virtuale;
- Uno switch (s1) che connette gli host tra loro.

3.2.2 Verifica degli host

Una volta avviato, Mininet fornisce un prompt interattivo dal quale è possibile eseguire comandi. Vogliamo per esempio verificare la connettività tra i due

host effettuando un ping da h1 a h2. Per farlo, possiamo utilizzare il seguente comando all'interno del prompt di Mininet:

```
mininet> h1 ping h2
```

Questo comando invia pacchetti ICMP da h1 a h2 e ci permette di verificare se la comunicazione tra i due host avviene correttamente.

3.2.3 Sessioni xterm

A questo punto è possibile aprire una sessione xterm per ogni host. Xterm è un emulatore di terminale che consente di aprire una finestra di terminale separata per ciascun host. In questo modo è possibile eseguire comandi direttamente su ogni nodo della rete simulata. Utilizziamo il comando

```
mininet> xterm h1 h2
```

In questo modo apriremo due finestre di terminale per ciascun host.

3.2.4 Connessione TCP

Possiamo ora instaurare una connessione TCP tra i due host. Per farlo possiamo utilizzare `netcat` mettendoci in ascolto sul primo nodo.

Nodo h1:

```
$ nc -l -p 5432
```

Il comando digitato mette l'host in ascolto (-l) sulla porta (-p) 5432, pronto a ricevere dati.

Nodo h2:

```
$ nc 10.0.0.1 5432
```

Questo comando avvia una connessione TCP verso l'host h1 (10.0.0.1) sulla stessa porta 5432.

La connessione è andata a buon fine, dunque qualsiasi testo digitato in un terminale verrà immediatamente visualizzato nell'altro, simulando una semplice comunicazione tra i due nodi.

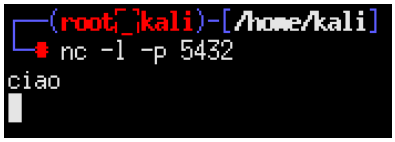

Figure 1: Host 1

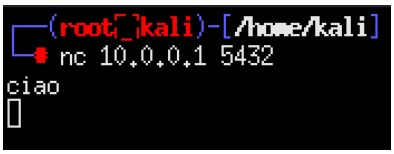

Figure 2: Host 2

3.2.5 Analisi del traffico con Wireshark

Attraverso l'uso di Wireshark possiamo vedere più nel dettaglio cosa avviene durante la comunicazione tra i due host. Una volta avviato il software possiamo metterci in ascolto su una delle due interfacce di rete usate da Mininet:

- s1-eth1 (porta dello switch collegata a h1)
- s1-eth2 (porta dello switch collegata a h2)

Nel mio caso utilizzerò la s1-eth2.

Una volta avviata la comunicazione TCP sul nodo h2, su Wireshark vedremo la fase del 3-Way Handshake, caratteristica delle connessioni TCP:

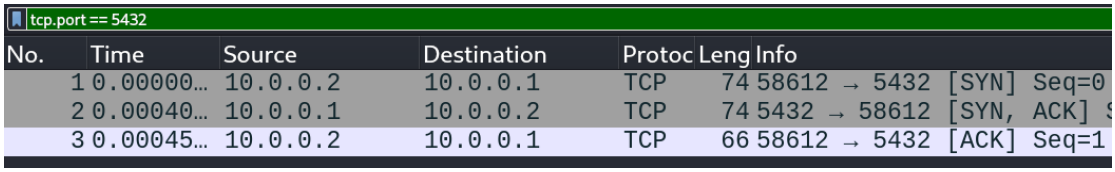

Figure 3: 3-Way Handshake

Da questa cattura notiamo immediatamente i tre passaggi chiave per stabilire una connessione affidabile tra i due host:

- **SYN**: h2 indica l'intenzione di stabilire una connessione;
- **SYN-ACK**: h1 conferma la richiesta;
- **ACK**: h2 completa il processo di handshake.

Dopo aver avviato la comunicazione tra h2 e h1, ho digitato il messaggio "ciao" nella finestra xterm di h2. Wireshark ha catturato due pacchetti principali relativi allo scambio di dati:

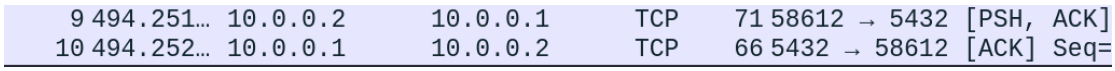

Figure 4: Scambio dati

Il primo pacchetto mostra il trasferimento del messaggio "ciao" da h2 a h1, con il flag PSH che indica che i dati devono essere consegnati immediatamente.
Il secondo pacchetto è una semplice risposta ACK, senza dati, per confermare che h1 ha ricevuto il messaggio correttamente.

Infine, analizzando il pacchetto catturato su Wireshark, possiamo osservare che il contenuto del messaggio "ciao" è visibile in chiaro all'interno del payload del pacchetto TCP. Il motivo è legato al fatto che netcat non utilizza alcuna forma di crittografia, trasmettendo i dati in chiaro.

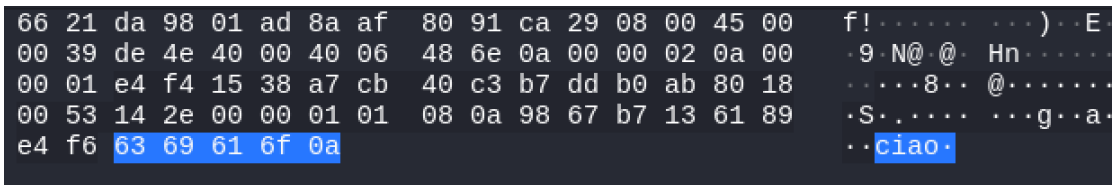

Figure 5: Payload

3.3 Conclusioni

In questo primo esercizio abbiamo avuto modo di osservare il funzionamento del protocollo TCP e l'implementazione delle interfacce virtuali attraverso il framework Mininet. Dopo aver configurato una rete composta da due host e uno switch virtuale, abbiamo stabilito una comunicazione tra i nodi, analizzando le fasi del 3-Way Handshake e la trasmissione dei dati tra i due endpoint. Inoltre, l'esercizio ci ha permesso di familiarizzare con il framework Mininet, che è uno strumento potente per simulare reti virtuali.

4 Esercizio 2: Indirizzamento indiretto

4.1 Introduzione

Il secondo esercizio si basa sulla creazione di un'altra infrastruttura di rete ma, a differenza del precedente esercizio, abbiamo due host che devono comunicare attraverso un router. Per questo motivo gli indirizzi IP dei due host sono:

- Host 1: 10.0.1.1/24;
- Host 2: 10.0.2.1/24.

La maschera di sottorete /24 indica infatti che i due host appartengono a due sottoreti diverse. Poiché si trovano in due subnet differenti, i due host non possono comunicare direttamente tramite un semplice switch, come avveniva nel primo esercizio. Per consentirne la comunicazione è necessario introdurre un router che instradi i pacchetti tra le due reti.

4.2 Metodologia e risultati

Per questo secondo esercizio utilizzeremo le API minent attraverso il linguaggio Python.

4.2.1 Librerie

```
1 from mininet.topo import Topo
2 from mininet.net import Mininet
3 from mininet.node import Node
4 from mininet.log import setLogLevel, info
5 from mininet.cli import CLI
```

Queste sono le librerie principali che importiamo: `Topo` permette di definire la topologia di rete specificando host, eventuali switch e collegamenti, `Mininet` ci permette di creare e gestire la rete virtuale, con `Node` possiamo creare nodi personalizzati (nel nostro caso il router) e `CLI` ci permette di interagire direttamente con Mininet dopo aver avviato la topologia.

4.2.2 Classe del Router

```
1  class GenericRouter(Node):
2      def config(self, **params):
3          super(GenericRouter, self).config(**params)
4
5          # Enable forwarding on the router
6          self.cmd('sysctl net.ipv4.ip_forward=1')
7
8      def terminate(self):
9          self.cmd('sysctl net.ipv4.ip_forward=0')
10         super(GenericRouter, self).terminate()
```

Definiamo ora la classe `GenericRouter` che rappresenta il router all'interno della topologia che andremo a creare. Per abilitarne l'instradamento, a riga 6, andiamo a modificare un parametro del kernel linux `net.ipv4.ip_forward` e lo impostiamo a 1.

4.2.3 Topologia - Router

```
1  class MyTopo(Topo):
2      def build(self):
3          # Creazione del router
4          router = self.addNode('r1', cls=GenericRouter, ip='
       10.0.1.254/24')
```

A questo punto definiamo la classe `MyTopo` per costruire la nostra topologia personalizzata. Per prima cosa creiamo il router `r1` assegnando l'indirizzo IP `10.0.1.254/24`. Questo indirizzo ci fa capire che il router si trova nella stessa sottorete dell'host 1 e dunque farà da gateway per la sottorete `10.0.1.0/24`. Per convenzione, ho scelto come indirizzo .254 poichè solitamente i gateway utilizzano indirizzi facilmente riconoscibili.

4.2.4 Topologia - Host

```
1         # Creazione dei nodi host
2         h1 = self.addHost('h1', ip='10.0.1.1/24', defaultRoute='via
    10.0.1.254')
3         h2 = self.addHost('h2', ip='10.0.2.1/24', defaultRoute='via
    10.0.2.254')
```

Dopo aver definito il router passiamo alla creazione degli host `h1` e `h2`. Ad entrambi assegniamo gli indirizzi IP descritti nell'introduzione e impostiamo anche un parametro `defaultRoute`, il gateway predefinito per ogni host:

- h1 ha come gateway 10.0.1.254, che è l'indirizzo dell'interfaccia del router sulla sottorete 10.0.1.0/24;
- h2 ha come gateway 10.0.2.254, che è l'indirizzo dell'interfaccia del router sulla sottorete 10.0.2.0/24.

In questo modo, quando un host deve inviare pacchetti destinati a un altro dispositivo che non si trova nella stessa sottorete, i pacchetti vengono inviati al gateway. Il router, a sua volta, inoltrerà i pacchetti alla rete corretta.

4.2.5 Topologia - Link e interfacce

```
1         # Creazione delle interfacce di rete
2         self.addLink(h1, router, intfName1='h1-eth0', intfName2='r1-
    eth0', params1={'ip': '10.0.1.1/24'}, params2={'ip': '
    10.0.1.254/24'})
3         self.addLink(h2, router, intfName1='h2-eth0', intfName2='r1-
    eth1', params1={'ip': '10.0.2.1/24'}, params2={'ip': '
    10.0.2.254/24'})
```

Ora, dobbiamo creare i link tra gli host e il router per stabilire la comunicazione tra le due sottoreti. Per ciascun link, associamo un'interfaccia per l'host e un'interfaccia per il router:

- Per il link tra h1 e il router, assegniamo l'interfaccia `h1-eth0` all'host h1 e l'interfaccia `r1-eth0` al router. L'indirizzo IP dell'host h1 è 10.0.1.1/24, mentre l'indirizzo IP dell'interfaccia del router è 10.0.1.254/24.
- Per il link tra h2 e il router, assegniamo l'interfaccia `h2-eth0` all'host h2 e l'interfaccia `r1-eth1` al router. L'indirizzo IP dell'host h2 è 10.0.2.1/24, mentre l'indirizzo IP dell'interfaccia del router è 10.0.2.254/24.

In questo modo ogni host ha la sua interfaccia dedicata per comunicare con il router.

4.2.6 Creazione e avvio della rete

```
def main():
    # Istanza della topologia
    topo = MyTopo()

    # Creazione della rete
    net = Mininet(topo=topo, controller=None)

    # Avvio della rete
    net.start()

    # Avvio la CLI per fare test
    CLI(net)

    # Terminazione della rete
    net.stop()
```

Adesso che gli step per la creazione della topologia sono terminati, possiamo creare la rete con la libreria importata `Mininet` a cui passiamo un'istanza della topologia appena definita. Infine avviamo la rete con `net.start` e apriamo un'interfaccia a riga di comando con `CLI(net)` grazie alla quale possiamo interagire con la rete e verificare che il tutto funzioni correttamente.

4.2.7 Verifica dell'infrastruttura

Una volta avviato lo script con:

```
$ sudo python3 mininet_topology.py
```

l'infrastruttura verrà creata e avremo subito a disposizione il prompt mininet per eseguire i comandi.

Il primo comando che possiamo utilizzare per verificare l'infrastruttura è `net`. Questo comando ci darà una panoramica della topologia di rete, mostrando gli host, i router e i link tra di essi:

```
mininet> net
h1 h1-eth0:r1-eth0
h2 h2-eth0:r1-eth1
r1 r1-eth0:h1-eth0 r1-eth1:h2-eth0
```

Figure 6: net

Successivamente, possiamo utilizzare il comando `dump` per avere dei dettagli in più sui nodi e sulle loro configurazioni di rete, comprese le interfacce e gli indirizzi IP:

```
mininet> dump
<Host h1: h1-eth0:10.0.1.1 pid=1699>
<Host h2: h2-eth0:10.0.2.1 pid=1701>
<GenericRouter r1: r1-eth0:10.0.1.254,r1-eth1:10.0.2.254 pid=1705>
```

Figure 7: dump

Adesso possiamo passare ai comandi per testare la connettività tra gli host. Per esempio, possiamo eseguire il comando `ping` tra `h1` e `h2` per vedere se i pacchetti vengono correttamente instradati attraverso il router:

```
mininet> h1 ping h2
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.2.1: icmp_seq=1 ttl=63 time=0.369 ms
64 bytes from 10.0.2.1: icmp_seq=2 ttl=63 time=0.028 ms
64 bytes from 10.0.2.1: icmp_seq=3 ttl=63 time=0.027 ms
64 bytes from 10.0.2.1: icmp_seq=4 ttl=63 time=0.026 ms
```

Figure 8: ping

Infine, possiamo utilizzare il comando `traceroute` per analizzare il percorso seguito dai pacchetti da `h1` a `h2` e verificare che passino attraverso il router:

```
mininet> h1 traceroute h2
traceroute to 10.0.2.1 (10.0.2.1), 30 hops max, 60 byte packets
 1  10.0.1.254 (10.0.1.254)  0.738 ms  0.182 ms  0.174 ms
 2  10.0.2.1 (10.0.2.1)  0.170 ms  0.142 ms  0.129 ms
```

Figure 9: traceroute

Come mostrato in Figura 9, l'output di `traceroute` conferma che i pacchetti inviati da `h1` raggiungono il router `r1` (10.0.1.254) prima di essere inoltrati alla destinazione finale `h2` (10.0.2.1).

4.3 Conclusioni

In questo secondo esercizio abbiamo trattato un indirizzamento indiretto. A differenza del primo esercizio, in cui i due host si trovavano nella stessa rete e dunque non avevano bisogno di un router per comunicare, in questo tipo di infrastruttura è necessario introdurlo in quanto i due host si trovano in due reti differenti. In questo esercizio abbiamo inoltre utilizzato le API mininet di Python che ci hanno permesso di avere maggiore flessibilità nella creazione dell'infrastruttura di rete. Infine, abbiamo verificato la corretta configurazione dell'infrastruttura con una serie di comandi UNT (Unix Network Toolkit) e, secondo i risultati in output, possiamo concludere che la rete è stata configurata correttamente.

5 Esercizio 3: Physical Constraints

5.1 Introduzione

L'ultima parte di questa esercitazione prevede l'aggiunta di alcuni vincoli fisici all'esercizio 2.1 attraverso la classe `TCLink` del modulo `mininet.link` ed in particolare un delay di 75ms ed una larghezza di banda di 100Mbps ad ogni link.

5.2 Metodologia e risultati

Per questo ultimo esercizio utilizzeremo la classe `TCLink` appartenente al modulo `mininet.link` e andremo a modificare l'esercizio precedente come segue.

5.2.1 Libreria

```
1 from mininet.link import TCLink
```

5.2.2 Vincoli

```
1       self.addLink(h1, router, intfName1='h1-eth0', intfName2='r1-
    eth0', params1={'ip': '10.0.1.1/24'}, params2={'ip': '
    10.0.1.254/24'}, cls=TCLink, bw=100, delay='75ms')
2
3       self.addLink(h2, router, intfName1='h2-eth0', intfName2='r1-
    eth1', params1={'ip': '10.0.2.1/24'}, params2={'ip': '
    10.0.2.254/24'}, cls=TCLink, bw=100, delay='75ms')
```

Aggiungiamo i vincoli ai link, in particolare un delay di 75ms e una bandwidth (bw) di 100Mbps.

5.2.3 Test - ping

A questo punto possiamo testare la rete rispetto all'esercizio precedente. Il primo comando che utilizziamo è un semplice `ping`:

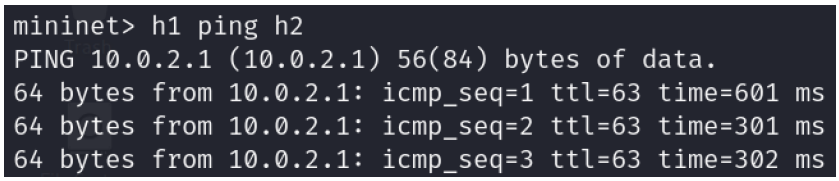

Figure 10: ping

Come possiamo osservare in questo esperimento, rispetto alla Figura 8, il tempo di latenza è aumentato. Vuol dire che i pacchetti devono attendere più tempo per attraversare la rete rispetto alla configurazione precedente.

5.2.4 Test - iperf

Infine possiamo utilizzare il comando `iperf` per misurare la larghezza di banda disponibile tra i due host. Utilizziamo due comandi, il primo è

```
mininet> h1 iperf -s &
```

Questa istruzione fa partire iperf in modalità server sull'host h1 e lo esegue in background (&). In questo modo h1 può ricevere connessioni dai client.

Successivamente utilizziamo il comando

```
mininet> h2 iperf -c h1
```

Questo comando avvia iperf in modalità client su h2, che si connette al server iperf in esecuzione su h1 e misura la larghezza di banda disponibile tra i due host.

Nel caso dell'infrastruttura senza vincoli avremo questo risultato:

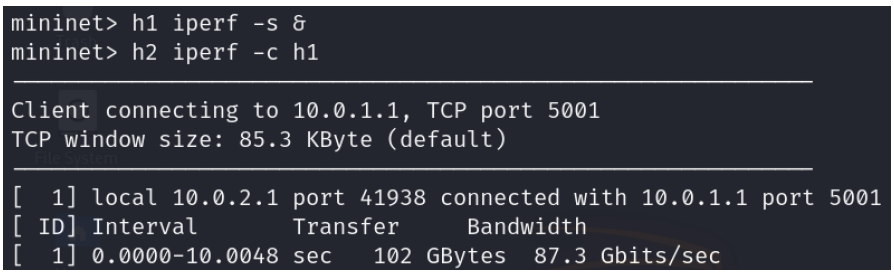

Figure 11: Infrastruttura senza vincoli

Il tasso di trasferimento è di 87.3 Gbits/sec. Senza vincoli la larghezza di banda è illimitata e dunque la comunicazione tra h1 e h2 avviene alla massima velocità possibile all'interno dell'ambiente virtuale.

Se però mettiamo i vincoli già descritti avremo una situazione di questo tipo: In questo caso il tasso di trasferimento è di 62.3 Mbits/sec. Abbiamo una signi-

```
mininet> h1 iperf -s &
mininet> h2 iperf -c h1
------------------------------------------------------------
Client connecting to 10.0.1.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  1] local 10.0.2.1 port 58992 connected with 10.0.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-11.1459 sec  82.8 MBytes  62.3 Mbits/sec
```

Figure 12: Infrastruttura con vincoli

ficativa riduzione della velocità di trasferimento addirittura sotto al limite che abbiamo impostato di 100Mbps, dovuta anche al delay di 75ms che abbiamo introdotto.

5.3 Conclusioni

In questo ultimo esercizio abbiamo visto come l'introduzione di vincoli di banda e latenza influenzi in maniera importante le prestazioni della rete. Rispetto all'esercizio precedente, in cui la rete non aveva limitazioni, abbiamo osservato un aumento della latenza nei pacchetti ICMP e una riduzione del tasso di trasferimento misurato con `iperf`.