

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

---

SCUOLA DI INGEGNERIA  
DIPARTIMENTO DELL'ENERGIA ELETTRICA E DELL'INFORMAZIONE  
"GUGLIELMO MARCONI" DEI

Corso di Laurea in  
Ingegneria Elettronica e Telecomunicazioni

Tesi di Laurea in  
Calcolatori Elettronici

## **Strumento per annotazione semantica 3D con realtà virtuale**

Candidato:  
**Luigi Lella**

Relatore:  
Ch.mo prof. **Luigi Di Stefano**  
Correlatore:  
**Pierluigi Zama Ramirez**

---

Anno Accademico 2019/2020

Sessione IV



# Indice

<b>Introduzione</b>	<b>1</b>
<b>1 Il Network Slicing</b>	<b>3</b>
1.1 Concetti principali . . . . .	3
1.1.1 Software Defined Networking . . . . .	4
1.1.2 Virtualized Network Functions . . . . .	5
1.2 Architetture . . . . .	6
1.3 Utilizzi e vantaggi . . . . .	10
<b>2 Introduzione al lavoro</b>	<b>15</b>
2.1 Blender . . . . .	15
2.1.1 Installazione . . . . .	16
2.1.2 Avvio di Mininet . . . . .	16
2.1.3 Gestione della rete . . . . .	17
2.1.4 Topologie complesse . . . . .	19
2.2 OpenFlow . . . . .	21
2.3 Strumenti di analisi prestazionale . . . . .	23
<b>3 Esempio pratico</b>	<b>29</b>
3.1 Idea di base . . . . .	29
3.2 Realizzazione . . . . .	29
3.2.1 Topologia . . . . .	30
3.2.2 Forwarding . . . . .	33
3.3 Prestazioni . . . . .	37
3.3.1 Latenza . . . . .	38
3.3.2 Banda . . . . .	40
<b>Conclusione</b>	<b>43</b>

**Bibliografia**

**45**

# Elenco delle figure

1.1	Software Defined Network . . . . .	4
1.2	Network Function Virtualization, NFV . . . . .	5
1.3	Architettura Network Slicing . . . . .	7
1.4	Network Slice life cycle . . . . .	7
1.5	Architettura NS a singolo controller . . . . .	9
1.6	Architettura NS a doppio controller . . . . .	9
1.7	Vehicle-to-everything . . . . .	12
1.8	Smart Grids . . . . .	12
2.1	Interfaccia di Mininet . . . . .	16
2.2	Comandi di base su Mininet . . . . .	18
2.3	Protocollo OpenFlow . . . . .	21
2.4	Comando "ping" . . . . .	23
2.5	Comando "nmap" . . . . .	24
2.6	Comando "iperf" . . . . .	25
2.7	Interfaccia di Wireshark . . . . .	26
2.8	Utilizzo di Wireshark . . . . .	26
3.1	Topologia d'esempio . . . . .	31
3.2	Pingall con STP disattivato . . . . .	36
3.3	Pingall con STP attivato . . . . .	37
3.4	Latenza STP . . . . .	38
3.5	Latenza Porta 20 . . . . .	38
3.6	Latenza Porta 5000 . . . . .	39
3.7	Latenza media . . . . .	39
3.8	Banda STP . . . . .	40
3.9	Banda Porta 20 . . . . .	40

3.10 Banda Porta 5000 . . . . .	41
3.11 Banda media . . . . .	41

# Introduzione

Negli ultimi anni machine learning e deep learning hanno rivoluzionato il mondo dell'intelligenza artificiale e della computer vision ottenendo risultati impressionanti in molti compiti come classificazione di immagini, segmentazione semantica ecc. superando, in alcuni casi, anche i risultati ottenuti da un utente umano [1].

Per usare tutti questi strumenti abbiamo bisogno di una grande quantità di dati al fine di supervisionare il training di reti neurali. Per questa ragione sono stati creati una grande quantità di dataset contenenti immagini e le rispettive annotazioni come ad esempio KITTI [2], Matterport3D [3] e Replica [4].

La fase di annotazione, come si può facilmente immaginare, è un processo lento, noioso e molto propenso ad errori. Labellare una singola immagine in 2D può richiedere alcune ore ed è un compito che richiede anche una certa esperienza con l'utilizzo di alcuni software. La difficoltà poi cresce se decido di lavorare con dati 3D come mesh o pointcloud.

Per semplificare e velocizzare questa parte si è deciso di realizzare *Shooting Labels* un tool per la segmentazione semantica 3D basato su realtà virtuale. L'utente viene trasportato all'interno di una scena rappresentata come mesh 3D, dove tutte le superfici possono essere "colorate" semanticamente. L'esperienza immersiva fornita dalle tecnologie VR consente all'utente di muoversi fisicamente all'interno dello scenario etichettare e interagire con gli oggetti in modo naturale e coinvolgente rendendo questo compito, oltre che più veloce, anche divertente come giocare a un video-game [5].

In questo elaborato si parlerà dapprima della segmentazione semantica e di tutti i problemi ad essa connessi, verranno poi presentati gli strumenti utilizzati nel corso della stesura della tesi e tirocinio per poi passare ad una

trattazione dettagliata del software. Infine verranno analizzati e commentati alcuni dati ottenuti dall'utilizzo di questo strumento.



# Capitolo 1

## Il Network Slicing

### 1.1 Concetti principali

La sempre più rapida evoluzione e innovazione delle tecnologie ha portato, negli anni, a modificare radicalmente le prestazioni ed i servizi richiesti alle infrastrutture di telecomunicazioni.

Internet, che ad oggi risulta la rete più diffusa al mondo, non rispecchia più le necessità degli utilizzatori. Alla sua nascita, infatti, la Qualità del Servizio (QoS) richiesta era decisamente inferiore. Per questo motivo, la filosofia su cui è basato Internet è detta *best-effort*: il sistema farà tutto il possibile per compiere un'operazione, ma non è garantito se e come questa sarà portata a termine.

Tuttavia, l'evoluzione di applicazioni e servizi ha reso necessaria una maggiore flessibilità delle prestazioni fornite dalla rete. A seconda dei casi, infatti, un servizio potrebbe richiedere bassa latenza, altri potrebbero avere bisogno di alto throughput. L'architettura e i protocolli di Internet non sono pensati per questo scopo e fornire prestazioni di questo tipo richiederebbe modifiche sostanziali. [6]

Da queste necessità nasce l'idea del Network Slicing.

Una *Network slice* è un insieme di risorse che compongono una rete in grado di soddisfare le necessità di una specifica applicazione. Questo comportamento si può ottenere utilizzando diverse architetture e tecnologie che, insieme, ci permettono di manipolare il comportamento della rete.

### 1.1.1 Software Defined Networking

Uno dei concetti alla base dell'idea del NS è il *Software Defined Networking (SDN)*, ovvero la costruzione di reti in cui il *routing* ed il *forwarding* sono gestiti separatamente.

L'SDN prevede infatti la presenza di uno o più controller, che gestiscono il routing ed il comportamento complessivo della rete. I vantaggi che un approccio di questo tipo comporta sono molteplici.

- La rete non è vincolata all'infrastruttura fisica. Il suo comportamento, definito via software, può quindi essere facilmente modificato e riprogrammato.
- La centralizzazione dell'intelligenza della rete stessa permette al gestore di operare solo sul controller, senza la necessità di intervenire su ogni singolo nodo. Inoltre, il controller permette di monitorare e supervisionare agevolmente l'intera infrastruttura [?]

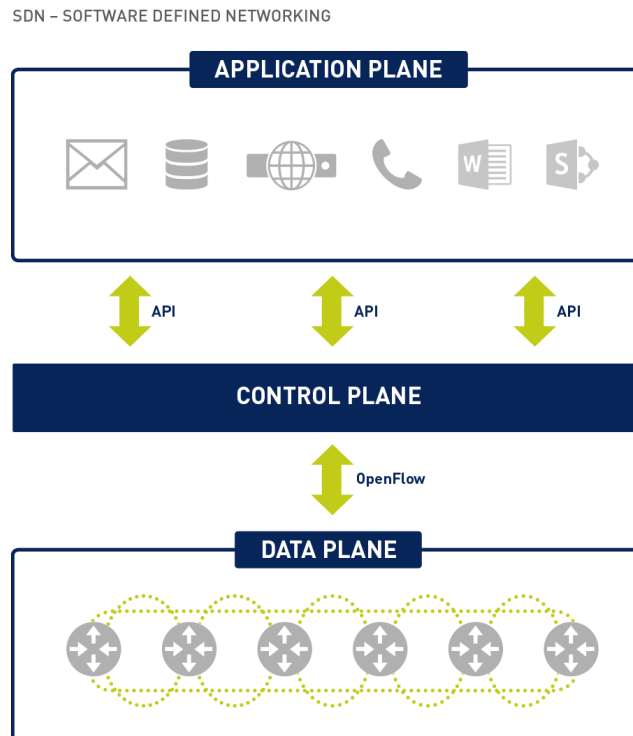


Figura 1.1: Gestione software dell'infrastruttura fisica in una rete di tipo SDN

### 1.1.2 Virtualized Network Functions

Un altro elemento cardine per il Network Slicing è la *virtualizzazione delle funzioni di rete (NFV)*. Le funzioni di rete, infatti, vengono solitamente implementate tramite hardware specifico in ogni nodo. Gli svantaggi di questa architettura includono l'alto prezzo dell'energia necessaria, un maggiore costo dovuto alla progettazione ed alla produzione dell'hardware stesso e l'inevitabile sempre più corto ciclo di vita di quest'ultimo.

La NFV si pone come obiettivo la soluzione di questi problemi, implementando le stesse funzioni di rete tramite software. Le *funzioni di rete virtualizzate* non necessitano quindi di una specifica infrastruttura fisica, ma sono in grado di essere memorizzate ed eseguite da generici switch, server e data center, tramite la creazione su di essi di una o più macchine virtuali. Questo, insieme alla possibilità di poter spostare questi elementi hardware in qualunque nodo della rete, rende estremamente più pratica, economica ed efficiente la gestione della rete stessa. [?]

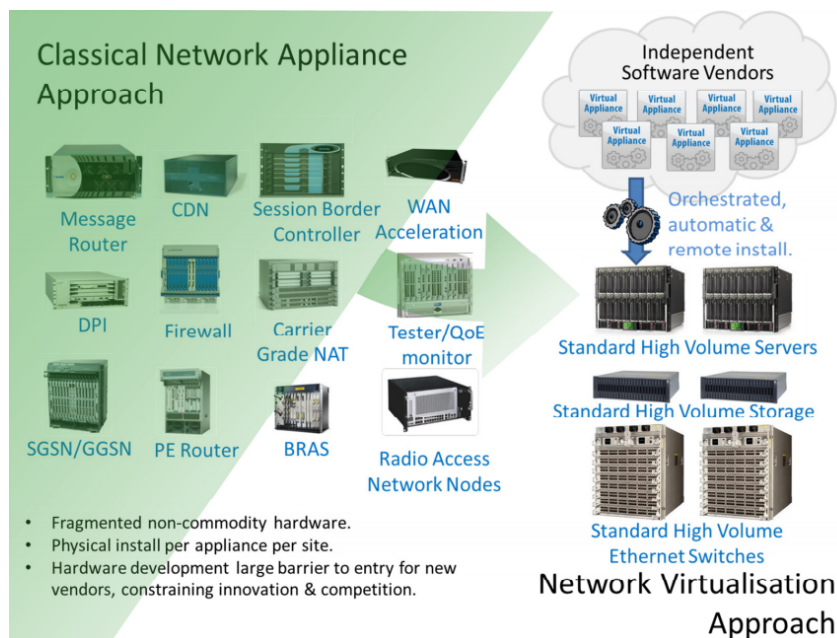


Figura 1.2: Differenze tra l'hardware specifico richiesto normalmente dalle funzioni di rete e da quello generico utilizzato dalle VNF

## 1.2 Architetture

Lo scopo del Network Slicing è fare in modo che un'infrastruttura possa suddividere e organizzare le proprie risorse affinché un'applicazione abbia a disposizione una rete ottimizzata per le sue richieste.

Per fare ciò, queste architetture sono strutturate su 3 livelli:

- *Service Instance Layer*

Rappresenta il servizio che deve essere supportato dalla rete. Ogni servizio è rappresentato da una Service Instance.

- *Network Slice Instance Layer*

Racchiude i vari Network Slices, ovvero le risorse allocate per una determinata Service Instance. Una Network Slice Instance può essere condivisa tra più SI.

- *Resource Layer*

Rappresenta le risorse e le funzioni di rete disponibili sull'infrastruttura, che verranno suddivise e assegnate in base al servizio richiesto.

L'intera architettura viene "orchestrata" da uno o più *controller*, che hanno il compito di creare e organizzare le varie Istanze, stanziando le risorse necessarie per ciascun servizio. [?]

L'utilizzo di un solo controller può costituire un collo di bottiglia per le prestazioni della rete in caso di compiti particolarmente complessi. Per questo motivo può risultare efficace dividere i compiti tra più controller, assegnando a ciascuno la gestione di determinate funzionalità.

Il processo di creazione di un'Istanza inizia con un'operazione di progettazione e preparazione della stessa. Viene poi effettuata una richiesta di creazione, e attivazione della Slice. Una volta attivata, questa viene monitorata e modificata. Quando non è più necessaria, infine, viene disattivata e le risorse vengono rese nuovamente disponibili.

È importante sottolineare come le Slices siano tra loro isolate. Questo è un

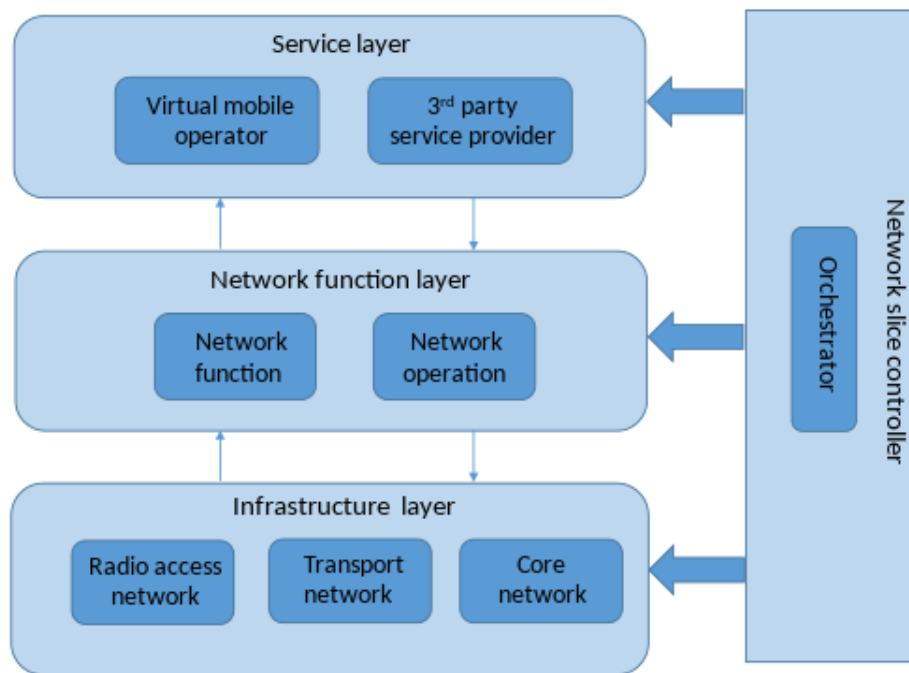


Figura 1.3: Struttura delle architetture di rete basate sul Network Slicing

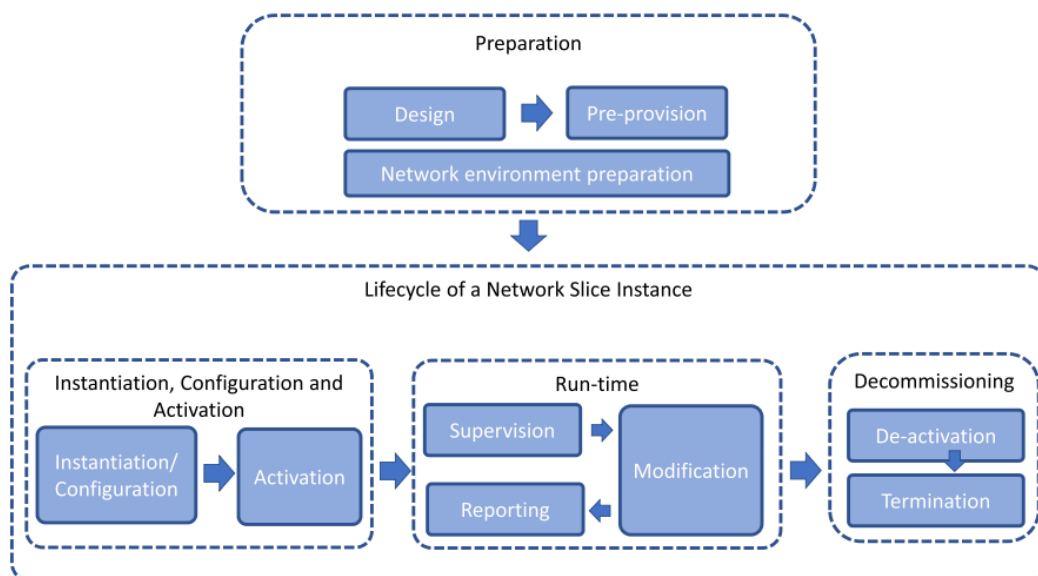


Figura 1.4: Life Cycle di una Network Slice Instance

criterio fondamentale, che permette una maggiore privacy, impedendo di accedere ai dati delle altre Slices, e garantisce che le prestazioni di una Slice non influiscano sulle altre.

Il Network Slicing può essere implementato in diversi modi. Un'infrastruttura con un unico proprietario avrà una gestione diversa rispetto ad una condivisa. Allo stesso modo, a seconda delle necessità, potrebbero essere necessari più o meno controller per gestire la rete.

- **Singolo Proprietario, singolo controller**

Questo è il caso più semplice, applicabile a porzioni ristrette della rete. In un'architettura di questo genere il Controller ha il compito di orchestrare direttamente l'infrastruttura e controllare direttamente tutte le Slices. Questo approccio può rappresentare un collo di bottiglia per l'affidabilità e per le prestazioni.

- **Singolo proprietario, più clienti**

Quando risulta necessario aumentare il numero di quindi di controller, questi non possono più agire direttamente sulla rete. È infatti necessario inserire un intermediario tra di essi: questo ruolo viene svolto da un *SDN proxy*, solitamente gestito dal proprietario della rete. Tra i vantaggi di questa architettura possiamo evidenziare la possibilità di permettere a vari utilizzatori di utilizzare i propri controller su un'infrastruttura fisica condivisa, mantenendo comunque l'isolamento tra le varie Istanze

- **Più proprietari**

I precedenti casi sono riferiti a situazioni in cui l'infrastruttura fisica sottostante risulti sotto il controllo di un unico proprietario. Le Slices possono quindi essere composte semplicemente isolando e dividendo i rami della rete, assegnandoli alle varie Istanze.

Tuttavia, nel caso in cui si vogliano utilizzare infrastrutture fisiche di proprietari diversi è necessario un ulteriore livello di astrazione, che permetta di rappresentarle come un'unica infrastruttura virtuale. Si crea così il concetto di *rete virtuale programmabile*. [6]

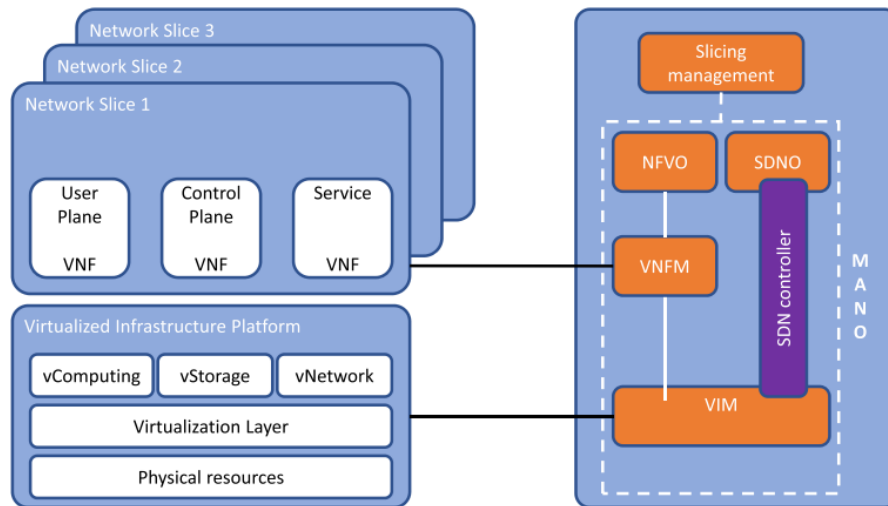


Figura 1.5: Struttura di un'architettura basata su Network Slicing a singolo proprietario e singolo controller

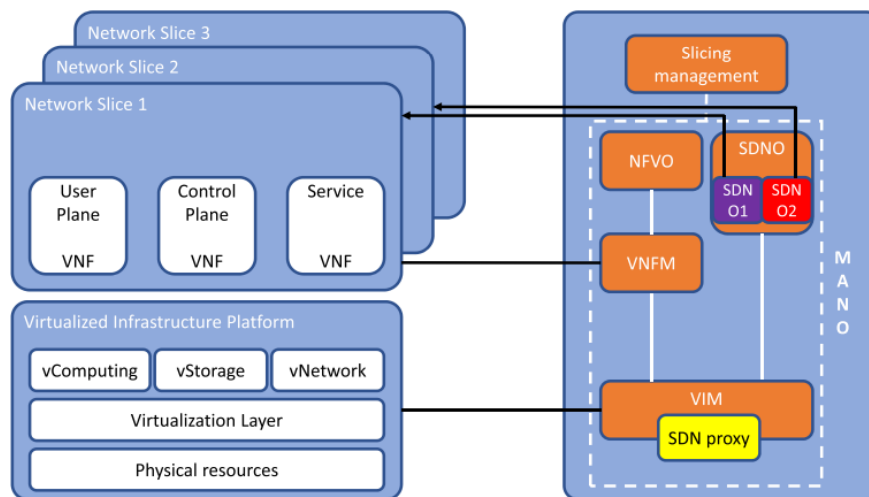


Figura 1.6: Struttura di un'architettura basata su Network Slicing a singolo proprietario e doppio controller

### 1.3 Utilizzi e vantaggi

L'idea del Network Slicing nasce con una strettissima correlazione con il 5G. Si prevede, infatti, che le reti di quinta generazione saranno in grado di creare grandi opportunità per nuove aziende e servizi. Questo porterà ad una grandissima varietà di richieste nei confronti della rete, che dovrà essere in grado di soddisfare necessità molto diverse tra loro.

Il Network Slicing si pone come una soluzione ideale a questo problema. La sua grande versatilità, infatti, risulta lo strumento perfetto per accontentare un'ampia gamma di richieste di varia natura. [?]

Queste tecnologie sono destinate ad innovare molte tipologie di comunicazione. Si prevede che un grosso impatto sarà percepito dall'industria e dai mercati verticali. Una rete flessibile e prestante ha infatti un'influenza significativa sulla produzione e permette di ottimizzare l'efficienza su tutti i livelli della filiera.

Un operatore può fornire al singolo cliente la gestione della propria Network Slice, permettendo di *orchestrarla* a seconda delle proprie necessità. Un cliente può essere rappresentato da un'azienda che voglia basare la sua rete interna su un'infrastruttura di questo genere.

Tra i molteplici mercati che trarrebbero vantaggio da queste tecnologie possiamo evidenziare i seguenti.

- **Automotive**

La potenzialità più interessante delle telecomunicazioni nel campo dell'automotive è senza dubbio la comunicazione *vehicle-to-everything*.

Con questo termine si intende la possibilità di un veicolo di comunicare con diversi interlocutori in grado di fornire informazioni utili per comfort e sicurezza:

- L'*Infotainment* è al giorno d'oggi una caratteristica imprescindibile di ogni veicolo. Una efficace comunicazione tra il veicolo e i passeggeri è fondamentale per una guida più attenta e confortevole.



- È opportuno anche che il veicolo sia in grado di comunicare continuamente con la casa madre, per mantenere il software e i dati sempre aggiornati.
- Per una maggiore sicurezza è importante garantire una comunicazione *vehicle-to-vehicle* a bassissima latenza, per permettere di conoscere posizione e velocità reciproche e avere a disposizione un maggior numero possibile di dati riguardanti le condizioni della strada e del traffico.
- Sarebbe infine possibile offrire un'ulteriore sicurezza permettendo al veicolo di comunicare con vari sensori a terra in grado di monitorare traffico, visibilità o condizioni del manto stradale.

L'utilizzo per questo scopo di una rete pubblica 5g ottimizzata tramite Network Slicing permetterebbe di avere una maggiore copertura, maggiori prestazioni e un costo inferiore.

- **Energia**

Una delle più importanti nuove frontiere nel campo dell'energia è rappresentata dalle *Smart Grids*.

Questo termine sta ad indicare una rete di distribuzione di energia supportata da un costante flusso di informazioni, in grado di gestire in maniera intelligente la distribuzione dell'elettricità, evitando sovraccarichi e minimizzando le perdite.

Inoltre, con la diffusione degli impianti fotovoltaici privati, la rete deve essere in grado di convogliare adeguatamente al proprio interno l'energia prodotta.

Un'infrastruttura che permetta queste operazioni al momento non esiste. Una rete 5g supportata dal Network Slicing sarebbe in grado di svolgere efficacemente queste funzioni, garantendo le prestazioni richieste.

Le necessità di una rete in questo ambito sono una zona di copertura molto estesa, una bassa latenza e una grande affidabilità, anche in situazioni di emergenza.

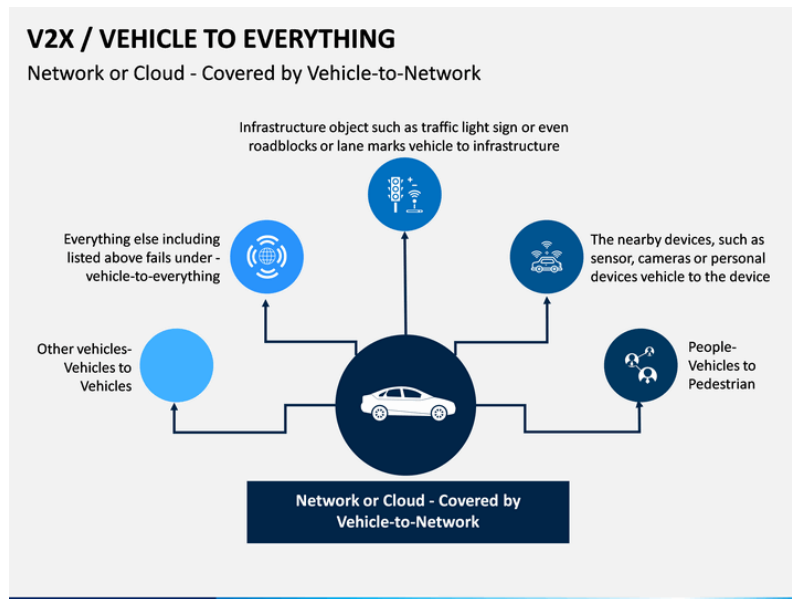


Figura 1.7: Comunicazioni di tipo Vehicle-to-Everything (V2X)

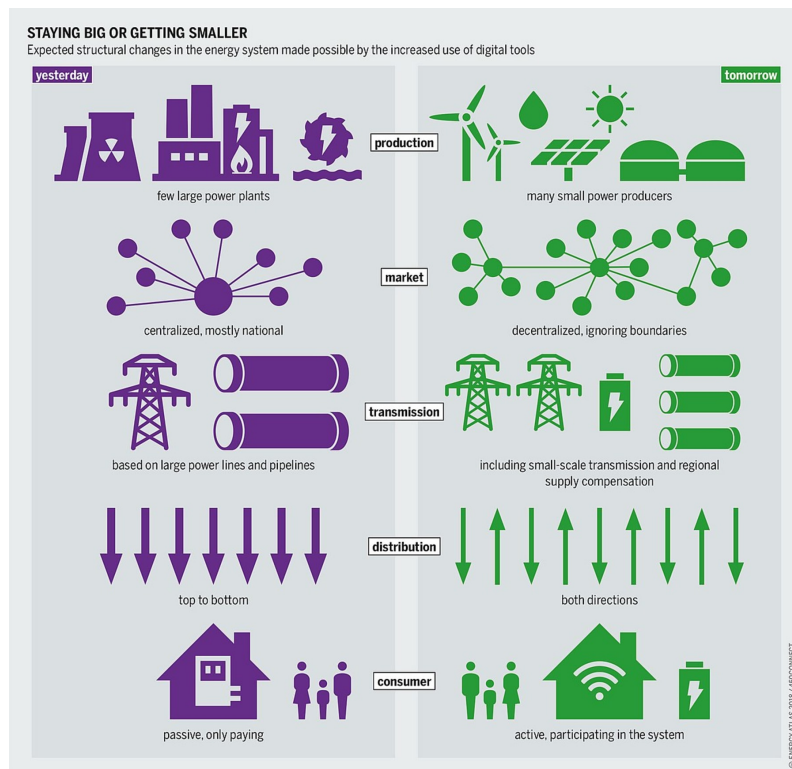


Figura 1.8: Produzione e gestione dell'energia tramite Smart Grid, supportata da una rete di informazione

- **Sanità**

In ambito sanitario, l'implementazione di reti flessibili e prestanti aprirebbe tantissime opportunità.

Una tra le novità sarebbe sicuramente la possibilità di monitorare in tempo reale le strumentazioni e i pazienti dentro e fuori gli ospedali.

Tuttavia, risulta ancora più interessante l'idea della *chirurgia da remoto*. Avere a disposizione una rete in grado di fornire una bassissima latenza, infatti, permetterebbe ai chirurghi di operare pazienti senza essere fisicamente in ospedale. Questo risulterebbe fondamentale in caso di operazioni specialistiche eseguite da esperti, riducendo drasticamente i costi per i pazienti ed eventualmente salvandone la vita.

- **IoT e Smart Cities**

IoT e Smart Cities sono campi in continua espansione, per i quali una connessione su aree di grandi dimensioni risulta fondamentale. L'idea di base è di avere una vasta rete di sensori che permettano di raccogliere dati e gestire di conseguenza svariati parametri, come illuminazione o riscaldamento, in modo da avere un funzionamento ottimale e minimizzare i consumi.

A differenza delle applicazioni precedentemente citate, la latenza non è un requisito stringente. È tuttavia fondamentale che i vari elementi della rete abbiano un basso dispendio energetico e che la copertura sia sufficiente a garantire che tutti i sensori possano comunicare. [?]

I vantaggi apportati dal Network Slicing sono quindi numerosi e di diversa natura, a seconda del servizio richiesto e della necessità del consumatore.

Nel capitolo 3 verranno confrontate le prestazioni di una rete classica con quelle di una rete costruita seguendo i principi del Network Slicing.



# Capitolo 2

## Introduzione al lavoro

### 2.1 Blender

Mininet è uno strumento di simulazione di reti che permette di creare topologie grandi e complesse e che supporta reti di tipo SDN basate sul protocollo OpenFlow.

Le idee chiave su cui è basato Mininet sono:

- *Flessibilità*: nuove topologie e funzioni sono implementate via software, usando linguaggi di programmazione diffusi e sistemi operativi comuni
- *Applicabilità*: le implementazioni su reti simulate devono poter funzionare anche su reti reali
- *Interattività*: la gestione ed il funzionamento delle reti simulate devono essere in tempo reale, come avverrebbe in una rete vera
- *Scalabilità*: l'ambiente di prototipazione deve essere in grado di simulare grandi quantità di host e switch su un solo computer
- *Realismo*: il comportamento della rete simulata deve rispecchiare con un buon livello di confidenza quello della rete reale, in modo da poter implementare applicazioni e protocolli senza modifiche al codice
- *Condivisibilità*: i prototipi creati devono essere facilmente condivisibili con altri collaboratori che a loro volta possano eseguire e modificare le simulazioni [?]

### 2.1.1 Installazione

Il modo più semplice per utilizzare Mininet è scaricare dal sito ufficiale una versione modificata di Ubuntu 14.04 nella quale sono preinstallati, oltre a Mininet, OpenFlow e vari programmi utili per la gestione e l'analisi delle reti. Questo Sistema Operativo può poi essere installato su una macchina virtuale. All'accensione si presenta privo di un'interfaccia grafica, che è tuttavia possibile installare. Per utilizzarlo è quindi opportuno conoscere i comandi fondamentali per utilizzare Linux da linea di comando.

Dopo l'installazione, l'accensione ed il login, l'interfaccia che si presenta all'utilizzatore è questa:



```
Ubuntu 14.04.6 LTS mininet-vm tty1
mininet-vm login: mininet
Password:
Last login: Thu Sep 10 04:07:20 PDT 2020 from 10.0.3.2 on pts/2
Welcome to Ubuntu 14.04.6 LTS (GNU/Linux 4.2.0-27-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
mininet@mininet-vm:~$
```

Figura 2.1: Interfaccia a linea di comando di Mininet all'accensione

### 2.1.2 Avvio di Mininet

Per avviare Mininet è necessario usare il comando "mn" con i permessi di root, quindi "sudo mn".

In questo modo si crea la topologia di default, composta da due host, uno switch ed un controller. È anche possibile creare topologie personalizzate tramite uno script Python da eseguire all'avvio.

Ci si trova quindi davanti la *CLI (Command Line Interface)* di Mininet. Ciascun host possiede a sua volta una CLI in grado di eseguire tutti i comandi di una normale macchina Linux. Per farlo è sufficiente scrivere il nome dell'host sul quale si vuole eseguire il comando, seguito dal comando stesso.

### 2.1.3 Gestione della rete

Mininet mette a disposizione dei comandi specifici per poter monitorare lo stato della rete. Il seguente elenco spiega alcuni tra i comandi principali e la loro applicazione alla topologia di default.

- *mininet> nodes*

Questo comando genera un elenco di tutti i nodi. Nel caso in esame i nodi sono h1, h2, s1 e c0.

- *mininet> net*

Si usa questo comando per conoscere le interfacce tramite cui sono collegati i nodi. In questo caso l'interfaccia eth0 dell'host h1 è collegata all'interfaccia eth1 dello switch s1, mentre l'interfaccia eth0 dell'host h2 è collegata all'interfaccia eth2 dello switch s1.

- *mininet> dump*

Questo comando mostra gli indirizzi IP e gli ID dei processi dei singoli nodi. Possiamo notare che l'indirizzo IP di h1 è 10.0.0.1 con ID del processo uguale a 6806, mentre l'indirizzo di h2 è 10.0.0.2 con ID del processo uguale a 6808.

- *mininet> link s1 h1 down/up*

Questi comandi vengono usati per disattivare o attivare una connessione. È possibile vedere l'effetto di questa azione provando ad eseguire tra i due nodi interessati un ping che, se la connessione non è attiva, fallirà.

- *mininet> h1 ping s1*

Si usa questa sintassi per effettuare un ping. In particolare, si utilizza il comando "ping", utilizzabile normalmente su linux, specificando il nodo che deve eseguire l'azione, h1, e il nodo destinazione, s1. Per *pingare* tutti i nodi si usa il comando "*pingall*".

- *mininet> h1 ifconfig -a*

Anche questo comando è presente di base in tutte le macchine linux. Utilizzato in questo modo serve per visualizzare tutte le informazioni, tra cui indirizzo IP, MAC e netmask, sulle connessioni di h1.

```

mininet@mininet-vm:~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=6806>
<Host h2: h2-eth0:10.0.0.2 pid=6808>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=6813>
<Controller c0: 127.0.0.1:6653 pid=6799>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> h1 ifconfig -a
h1-eth0  Link encap:Ethernet  HWaddr 02:bd:ad:6e:65:67
         inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:4 errors:0 dropped:0 overruns:0 frame:0
         TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:280 (280.0 B)  TX bytes:280 (280.0 B)

lo       Link encap:Local Loopback
         inet addr:127.0.0.1  Mask:255.0.0.0
         UP LOOPBACK RUNNING  MTU:65536  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

```

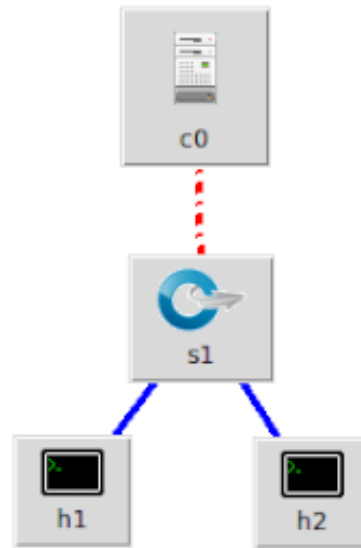


Figura 2.2: Utilizzo dei comandi di base di Mininet applicati alla topologia di default, rappresentata in alto a destra.



### 2.1.4 Topologie complesse

È ovviamente possibile creare topologie più elaborate rispetto a quella di default.

Per farlo bisogna specificare il parametro *"-topo"* quando si esegue *"sudo mn"*. Ad esempio, con il comando

```
sudo mn -topo linear,3
```

si crea una topologia composta da 3 host, ciascuno connesso ad uno switch. Se invece eseguiamo il comando

```
sudo mn -topo single,3
```

creiamo una topologia composta da 3 host, tutti connessi allo stesso switch.

È tuttavia possibile personalizzare ulteriormente le topologie da simulare. Per farlo si possono utilizzare le API Python di Mininet

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        h4 = self.addHost( 'h4' )
        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )
```

```
# Add links
self.addLink( h1, s1 )
self.addLink( h2, s1 )
self.addLink( s1, s2 )
self.addLink( s2, h3 )
self.addLink( s2, h4 )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

La prima riga serve per importare le API di Mininet. L'elenco completo si trova sul sito <http://mininet.org/api>.

Possiamo identificare i comandi principali del codice:

- *addHost* aggiunge un host alla topologia e fornisce in uscita il nome
- *addSwitch* aggiunge uno switch alla topologia e fornisce in uscita il nome
- *addLink* aggiunge una connessione bidirezionale tra i due nodi specificati

È quindi evidente che la topologia così creata sia composta da 4 host (h1, h2, h3 e h4) divisi in due coppie, ciascuna delle quali è connessa ad uno switch (s1 e s2), a loro volta connessi.

Questo codice può essere eseguito tramite il comando

```
sudo mn -custom esempio.py -topo mytopo
```

dove il parametro *-custom* serve per specificare il percorso e il nome del file python, mentre *-topo* permette di scegliere il nome della topologia da usare.

Durante la fase di creazione della topologia è possibile specificare anche altri parametri. Ad esempio, con il comando

```
self.addLink(host, switch, bw=10, delay='5ms', loss=10)
```

si crea un link con banda pari a 10Mbps, ritardo di 5ms, 10% di probabilità d'errore.

Altri esempi verranno forniti nel prossimo capitolo, dove verrà costruita una topologia per la quale sarà necessario utilizzare queste funzioni. [?]

## 2.2 OpenFlow

OpenFlow è il primo protocollo standardizzato per le reti SDN. Lo scopo del protocollo è di permettere l'accesso al piano di controllo di un router o di uno switch tramite la rete.

L'idea di base è quella di dividere il traffico in *flussi*. Un flusso è un insieme di pacchetti contraddistinti da caratteristiche comuni, come ad esempio la porta TCP o l'indirizzo IP o MAC di partenza.

Questi flussi sono organizzati in *flow tables*, ovvero tabelle nella quale viene specificato come il dispositivo possa identificare i pacchetti appartenenti ad un certo flusso e l'azione da eseguire su di essi.

Il controller è in grado di accedere a queste tabelle di flusso ed è in grado di modificarne dinamicamente le regole.

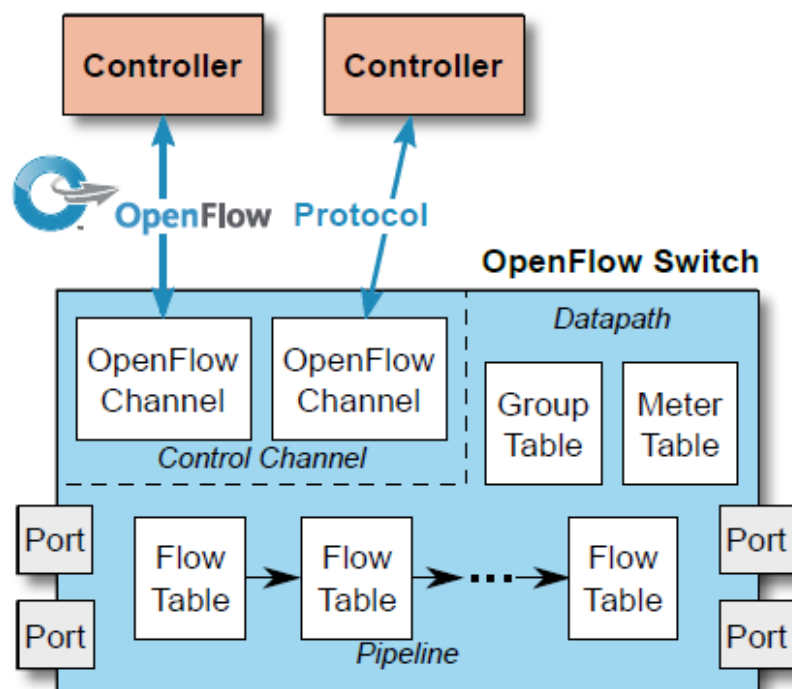


Figura 2.3: Struttura del protocollo openFlow, che permette di accedere al piano di controllo tramite la rete

Il protocollo OpenFlow può essere applicato a router e switch già esistenti, ciascuno dei quali già presenta delle flow tables, senza la necessità di doverli riprogrammare. Si crea quindi uno Switch OpenFlow, caratterizzato dalle tabelle di flusso e dal canale sicuro per poter comunicare con il controller.

Switch e controller devono poter comunicare non solo per permettere a quest'ultimo di modificare le tabelle di flusso, ma anche per permettere allo switch di inviare pacchetti al controller stesso nel caso in cui non sappia come comportarsi con essi. [?]

Mininet è spesso utilizzato per simulare reti di tipo SDN anche grazie al suo supporto di OpenFlow stesso. In fase di creazione di una topologia, infatti, è possibile specificare la tipologia degli switch.

Gli switch che supportano OpenFlow si chiamano OpenvSwitch (OVS) e posseggono una serie di comandi per la loro gestione e per la creazione ed organizzazione delle flow tables.

I principali sono:

- *mininet> sh ovs-ofctl show s1*

Fornisce informazioni sullo switch s1, tra cui dati su porte e flow tables

- *mininet> sh ovs-ofctl dump-tables s1*

Questo comando mostra un elenco delle tabelle di s1 e le relative informazioni

- *mininet> sh ovs-ofctl dump-ports s1*

Questo comando mostra i dispositivi della rete collegati allo switch e i relativi dati.

- *mininet> sh ovs-ofctl add-flow s1 flow*

Si usa questo comando per aggiungere un flusso nella flow table di s1. È necessario specificare le caratteristiche che accomunano i pacchetti appartenenti

Alcuni di questi comandi verranno messi in pratica nel prossimo capitolo. [?]

## 2.3 Strumenti di analisi prestazionale

Una caratteristica fondamentale di un programma come Mininet è la possibilità di poter misurare le prestazioni delle reti che si stanno analizzando. Linux ci fornisce diversi strumenti che ci permettono di fare ciò.

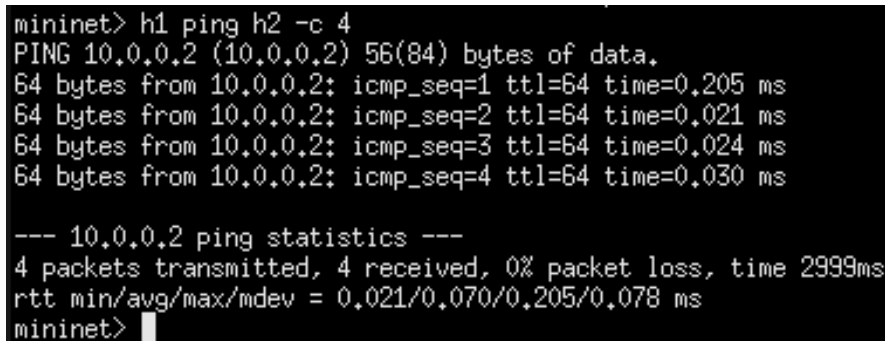
- **Ping**

Questo è senza dubbio lo strumento più semplice che abbiamo a disposizione per testare la comunicazione tra due dispositivi. Viene solitamente usato per verificare il funzionamento e la latenza nella connessione tra due dispositivi.

La sintassi del comando è

**ping {destinazione} [opzioni]**

Nell'esempio possiamo vedere un esempio di utilizzo del comando *ping* applicato agli host *h1* e *h2* della topologia di default. Il parametro *-c* serve per specificare quanti pacchetti di test inviare.



```
mininet> h1 ping h2 -c 4
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.205 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.021 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.024 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.030 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.021/0.070/0.205/0.078 ms
mininet> █
```

Figura 2.4: Utilizzo del comando "ping"

- **Nmap**

Uno dei limiti del comando *ping* è l'impossibilità di specificare la porta TCP della quale effettuare la verifica. *Nmap* ha un funzionamento molto simile al comando precedente, ma ci mette a disposizione questa funzione. Questa possibilità ci sarà molto utile quando, nel prossimo capitolo,

avremo bisogno di verificare la latenza della rete per i diversi servizi richiesti.

La sintassi del comando è

**nmap -p [n° porta] {destinazione}**

Nell'esempio vediamo l'utilizzo di *nmap* tra i due host della topologia di default sulla porta TCP 80.

```
mininet> h1 nmap -p 80 h2

Starting Nmap 6.40 ( http://nmap.org ) at 2020-09-30 01:29 PDT
Nmap scan report for 10.0.0.2
Host is up (0.0013s latency).
PORT      STATE SERVICE
80/tcp    closed http
MAC Address: E6:A0:2E:6D:A8:31 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 13.46 seconds
mininet> █
```

Figura 2.5: Utilizzo del comando "nmap"

- **Iperf**

La latenza non è l'unico parametro che è possibile misurare.

È infatti possibile monitorare anche la banda dei collegamenti, tramite il comando *iperf*.

Per utilizzarlo è necessario impostare uno dei due host come server e uno come client. Non è quindi possibile utilizzare direttamente il comando, è prima necessario aprire un terminale separato per ciascun host scrivendo nella CLI di Mininet

**mininet> xterm h1 h2**

Bisogna poi utilizzare *iperf* per impostare uno dei due host come server e l'altro come client.

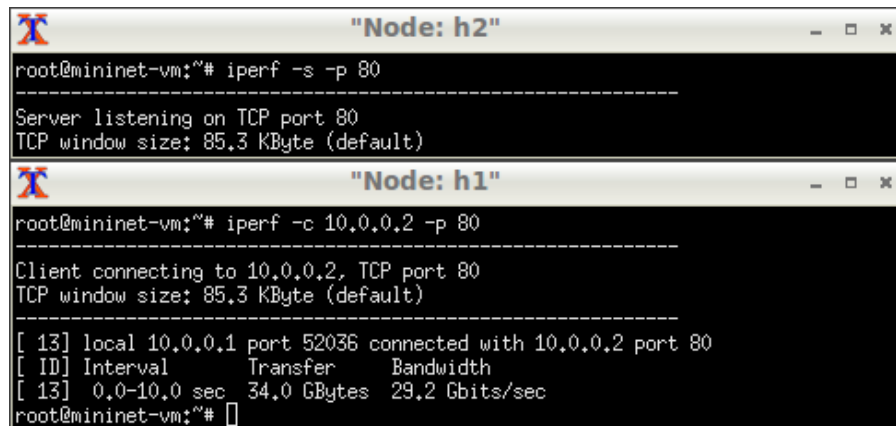
La sintassi del comando per il server è

**iperf -s -p [n° porta]**

La sintassi del comando per il client è

**iperf -c {destinazione} -p [n° porta]**

Nell'esempio h2 è il server e h1 è il client e i due comunicano sulla porta TCP 80.



The image shows two terminal windows. The top window, titled '"Node: h2"', shows the server configuration: 'root@mininet-vm:~# iperf -s -p 80', followed by 'Server listening on TCP port 80' and 'TCP window size: 85.3 KByte (default)'. The bottom window, titled '"Node: h1"', shows the client configuration: 'root@mininet-vm:~# iperf -c 10.0.0.2 -p 80', followed by 'Client connecting to 10.0.0.2, TCP port 80' and 'TCP window size: 85.3 KByte (default)'. Below this, it shows a connection summary: '[ 13] local 10.0.0.1 port 52036 connected with 10.0.0.2 port 80', a table header '[ ID] Interval Transfer Bandwidth', and a result line '[ 13] 0.0-10.0 sec 34.0 GBytes 29.2 Gbits/sec'.

```

"Node: h2"
root@mininet-vm:~# iperf -s -p 80
-----
Server listening on TCP port 80
TCP window size: 85.3 KByte (default)

"Node: h1"
root@mininet-vm:~# iperf -c 10.0.0.2 -p 80
-----
Client connecting to 10.0.0.2, TCP port 80
TCP window size: 85.3 KByte (default)
-----
[ 13] local 10.0.0.1 port 52036 connected with 10.0.0.2 port 80
[ ID] Interval Transfer Bandwidth
[ 13] 0.0-10.0 sec 34.0 GBytes 29.2 Gbits/sec
root@mininet-vm:~#

```

Figura 2.6: Utilizzo del comando "iperf"

- **Wireshark**

A differenza dei precedenti comandi, Wireshark è un vero e proprio programma dotato di interfaccia grafica. Il suo scopo è quello di "catturare" il traffico lungo una connessione, per poter analizzare i pacchetti che vengono inviati durante una trasmissione di dati.

Per avviarlo è sufficiente utilizzare da terminale il comando

**sudo wireshark**

dopo essersi assicurati di avere a disposizione un'interfaccia grafica.

Nella figura 2.7 possiamo vedere come appare il programma all'apertura. Se, dopo aver selezionato l'interfaccia *s1-eth1*, che nella topologia di default di Mininet corrisponde all'interfaccia di s1 collegata ad h1, avviamo un ping sulla porta TCP 80 tramite il comando *nmap* mostrato nell'esempio in figura 2.5, osserviamo il traffico catturato nell'immagine 2.8. Possiamo notare che i pacchetti sono di tipo *http*, che è appunto il servizio associato alla porta TCP 80. [?]

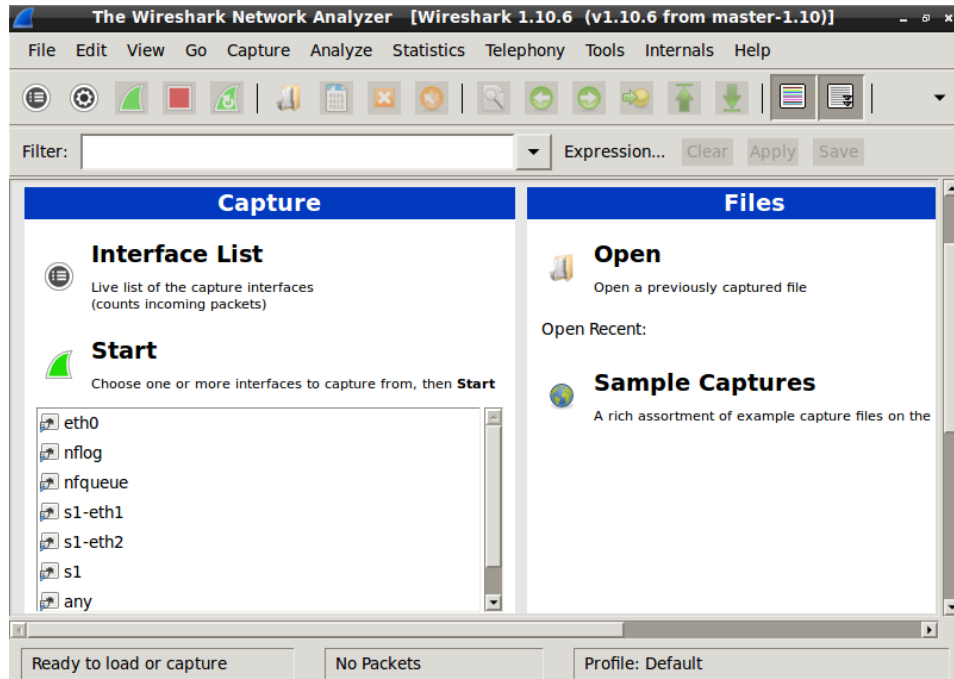


Figura 2.7: Interfaccia grafica di Wireshark all'accensione

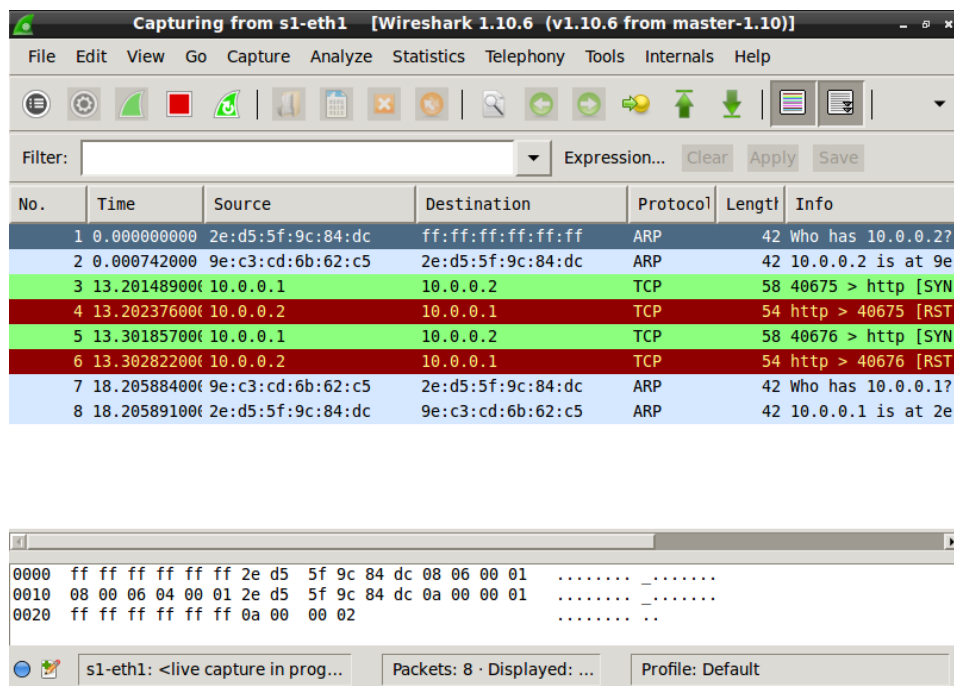


Figura 2.8: Esempio di utilizzo di Wireshark per la cattura di pacchetti



Ovviamente questi sono solo alcuni degli innumerevoli strumenti che Mininet e l'ambiente Linux in generale mettono a disposizione.

I comandi ed i programmi spiegati in questo capitolo verranno messi in pratica nel prossimo capitolo per la simulazione di una rete basata sul concetto di Network Slicing.

La topologia della rete verrà costruita con i comandi mostrati. Verranno poi analizzate le prestazioni per vari servizi prima e dopo averla programmata grazie al protocollo OpenFlow e verranno infine confrontate.



# Capitolo 3

## Esempio pratico

### 3.1 Idea di base

Nei precedenti capitoli abbiamo evidenziato come al giorno d'oggi una rete debba essere sempre più flessibile ed in grado di fornire le prestazioni più adeguate ad un servizio richiesto.

Abbiamo poi visto alcuni strumenti di simulazione che ci permettono di analizzare il funzionamento di reti di varia natura, in particolare le reti di tipo SDN per le quali i programmi citati sono ideali. Abbiamo infine esaminato diversi metodi per poter misurare le prestazioni della nostra rete, con un particolare interesse nei confronti degli strumenti che ci permettono di valutare il comportamento della rete rispetto ad uno specifico servizio.

Avendo a disposizione conoscenze e competenze nell'ambito delle reti SDN e del Network Slicing, è ora interessante simulare e confrontare due diverse reti, di uguale topologia, delle quali solo ad una verranno applicati i principi del Network Slicing.

### 3.2 Realizzazione

La topologia in esame ha una struttura piuttosto semplice: sono presenti due host, i quali dovranno comunicare tramite diverse porte TCP, ciascuna associata ad un servizio.

Queste porte sono state scelte come esempi di applicazioni che richiedono prestazioni diverse:

- **Porta 20: FTP**

FTP (File Transfer Protocol), è un protocollo di trasferimento dati. Possiamo considerare FTP come un esempio di servizio che richiede una banda importante, per permettere un veloce trasferimento dei dati e dei file.

- **Porta 5000: Chat, streaming e gaming**

La porta 5000 è utilizzata da varie applicazioni di diverse tipologie, come chat, videochat, streaming video e gaming. Tutti questi servizi sono accomunati dalla necessità di avere una comunicazione a bassa latenza.

Ovviamente esiste un enorme numero di possibili servizi. Per semplicità e chiarezza verranno usati solo questi tre, ma il processo è facilmente scalabile.

### 3.2.1 Topologia

Il Network Slicing prevede che una rete sia in grado di allocare le risorse necessarie affinché una specifica applicazione abbia a disposizione una rete ottimizzata per le sue necessità.

Ai fini di una dimostrazione efficace è quindi opportuno creare una topologia che sia in grado di fornire sia una bassa latenza sia una larga banda.

La topologia che andremo ad esaminare è composta da due host, collegati da quattro switch. Tra host e switch sono presenti collegamenti con diverse caratteristiche. Questa semplice rete verrà poi programmata affinché le prestazioni che la rete è in grado di fornire siano ripartite al meglio tra i vari servizi.

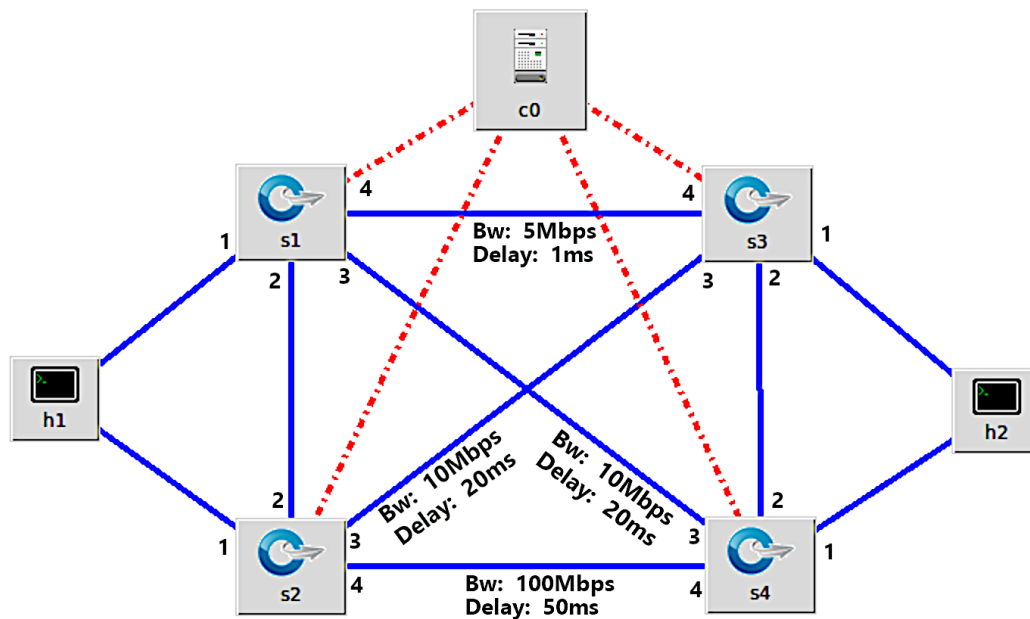


Figura 3.1: Topologia della rete in esame

Come illustrato nel capitolo precedente, per costruire la topologia è necessario avvalersi di uno Script Python.

Questo script verrà salvato nel file *test.py*

```

1 from mininet.topo import Topo
2 from mininet.link import TCLink
3
4 class MyTopo( Topo ):
5     "Simple topology example."
6
7     def __init__( self ):
8         "Create custom topo."
9
10        # Initialize topology
11        Topo.__init__( self )
12
13        # Add hosts and switches
14        h1 = self.addHost( 'h1' )

```

```

15     h2 = self.addHost( 'h2' )
16     s1 = self.addSwitch( 's1' )
17     s2 = self.addSwitch( 's2' )
18     s3 = self.addSwitch( 's3' )
19     s4 = self.addSwitch( 's4' )
20
21     # Add links
22     self.addLink( h1, s1, 1, 1, bw=100, delay='1ms' )
23     self.addLink( h1, s2, 2, 1, bw=100, delay='1ms' )
24     self.addLink( h2, s3, 1, 1, bw=100, delay='1ms' )
25     self.addLink( h2, s4, 2, 1, bw=100, delay='1ms' )
26     self.addLink( s1, s2, 2, 2, bw=100, delay='1ms' )
27     self.addLink( s3, s4, 2, 2, bw=100, delay='1ms' )
28
29     self.addLink( s1, s3, 4, 4, bw=5, delay='1ms' )
30     self.addLink( s1, s4, 3, 3, bw=10, delay='20ms' )
31     self.addLink( s2, s3, 3, 3, bw=10, delay='20ms' )
32     self.addLink( s2, s4, 4, 4, bw=100, delay='50ms' )
33
34     topos = { 'mytopo': ( lambda: MyTopo() ) }

```

Possiamo notare alcune differenze con il codice di esempio fornito nel capitolo precedente.

Nella creazione dei link, infatti, viene specificate le porte di ciascun nodo a cui ciascun ramo è collegato.

Per le connessioni create nelle righe 29, 30, 31 e 32, inoltre, sono state specificate banda e latenza, coerentemente con i dati della figura 3.1, mentre per le altre sono stati scelti come parametri standard 100Mbps di banda e 1ms di latenza.

A causa di questa differenza, è necessario importare ulteriori API, operazione che viene svolta alla riga 2.

Ai fini della dimostrazione è necessario creare due reti distinte, una classica e una di tipo SDN. Lo script Python deve quindi essere eseguito in due

modi diversi.

- Per creare una topologia classica si usa il comando

```
sudo mn -custom test.py -topo mytopo -link tc
```

che avvia Mininet con la topologia "*mytopo*" del file "*test.py*"

Da notare il parametro *-link* che deve essere posto "*tc*" per permettere la personalizzazione di banda e latenza.

- Per creare una topologia SDN bisogna invece scrivere

```
sudo mn -custom test.py -topo mytopo -link tc -switch ovsk
```

specificando quindi che gli switch debbano supportare OpenFlow.

### 3.2.2 Forwarding

È ora necessario configurare gli switch affinché la rete possa essere ottimizzata per i servizi richiesti.

Perché avvenga è necessario che i pacchetti che richiedono una bassa latenza, rappresentati dai servizi sulla porta TCP 5000, siano inviati sul ramo che collega s1 a s3, mentre per i servizi che necessitano di molta banda, rappresentati dai pacchetti *FTP*, è necessario che le informazioni viaggino sul ramo che collega s2 e s4.

I rami che collegano s1 con s2 e s3 con s4 sono dedicati a spostare i pacchetti verso il migliore tra i due percorsi.

Per configurare gli switch utilizziamo i comandi presentati nel capitolo precedente, con sintassi di questo tipo:

```
ovs-ofctl add-flow [switch] priority=[priorità],in_port=[porta  
switch],tcp,tcp_src=[porta TCP],actions=output:[porta switch]
```

Analizziamo il parametri del comando:

- *priority* definisce la priorità dell'azione. Un comando con priorità alta prevale su uno con priorità bassa.
- *in\_port* specifica in quale porta dello switch debbano entrare i pacchetti appartenenti al flusso
- *tcp\_src* specifica quale porta TCP stiano usando i pacchetti appartenenti al flusso
- *output* definisce su quale porta dello switch dovranno essere reindirizzati i pacchetti in uscita

Per comodità, tutti i comandi necessari vengono racchiusi in uno script chiamato *test.sh*, che presenta il seguente contenuto

```
1 #Elimino i flussi preesistenti
2 sudo ovs-ofctl del-flows s1
3 sudo ovs-ofctl del-flows s2
4 sudo ovs-ofctl del-flows s3
5 sudo ovs-ofctl del-flows s4
6
7 #Creo un collegamento tra gli switch e i propri host
8 sudo ovs-ofctl add-flow s1 priority=1,actions=output:1
9 sudo ovs-ofctl add-flow s2 priority=1,actions=output:1
10 sudo ovs-ofctl add-flow s3 priority=1,actions=output:1
11 sudo ovs-ofctl add-flow s4 priority=1,actions=output:1
12
13 #Creo un canale di default
14 sudo ovs-ofctl add-flow s1 priority=2,in_port=1,actions=output:3
15 sudo ovs-ofctl add-flow s2 priority=2,in_port=1,actions=output:3
```



```
16 sudo ovs-ofctl add-flow s3 priority=2,in_port=1,actions=output:3
17 sudo ovs-ofctl add-flow s4 priority=2,in_port=1,actions=output:3
18
19 #Sposto i pacchetti verso le linee ottimizzate
20 sudo ovs-ofctl add-flow s1
21     priority=4,dl_type=0x0800,nw_proto=6,in_port=1,tcp,tcp_dst=20,
22     actions=output:2
23 sudo ovs-ofctl add-flow s3
24     priority=4,dl_type=0x0800,nw_proto=6,in_port=1,tcp,tcp_dst=20,
25     actions=output:2
26 sudo ovs-ofctl add-flow s2
27     priority=4,dl_type=0x0800,nw_proto=6,in_port=1,tcp,tcp_dst=5000,
28     actions=output:2
29 sudo ovs-ofctl add-flow s4
30     priority=4,dl_type=0x0800,nw_proto=6,in_port=1,tcp,tcp_dst=5000,
31     actions=output:2
32
33 #Trasferisco i pacchetti sulle linee ottimizzate
34 sudo ovs-ofctl add-flow s1
35     priority=3,dl_type=0x0800,nw_proto=6,in_port=1,tcp,tcp_dst=5000,
36     actions=output:4
37 sudo ovs-ofctl add-flow s3
38     priority=3,dl_type=0x0800,nw_proto=6,in_port=2,tcp,tcp_dst=5000,
39     actions=output:4
40 sudo ovs-ofctl add-flow s2
41     priority=3,dl_type=0x0800,nw_proto=6,in_port=2,tcp,tcp_dst=20,
42     actions=output:4
43 sudo ovs-ofctl add-flow s4
44     priority=3,dl_type=0x0800,nw_proto=6,in_port=1,tcp,tcp_dst=20,
45     actions=output:4
46
47 sudo ovs-ofctl dump-flows s1
48 sudo ovs-ofctl dump-flows s2
49 sudo ovs-ofctl dump-flows s3
50 sudo ovs-ofctl dump-flows s4
```

Per avviare lo script è necessario renderlo eseguibile. Per farlo bisogna scrivere nella CLI

```
sudo chmod +X test.sh
```

Il comando per avviare lo script invece è

```
sudo bash ./test.sh
```

Le righe tra 2 e 5 servono per eliminare eventuali flussi preesistenti.

Le righe tra 8 e 17 creano un collegamento di default, per permettere ai servizi sconosciuti di poter giungere a destinazione.

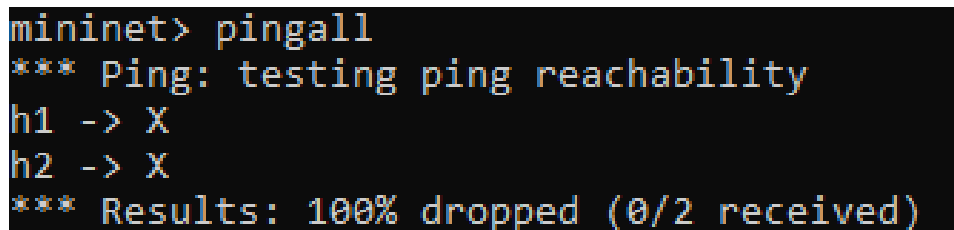
I comandi nelle righe tra 20 e 31 servono per permettere ai pacchetti di giungere sulla linea più adatta al servizio richiesto.

Le righe tra 34 e 45 compiono il vero e proprio trasferimento delle informazioni sulla linea. Una volta giunte allo switch successivo, queste vengono portate all'host grazie ai flussi precedentemente creati.

Per analizzare come le prestazioni vengano modificate dalla programmazione della rete misuriamo latenza e banda della rete per i diversi servizi prima e dopo l'esecuzione dello script.

Per un corretto inoltro delle informazione attraverso gli switch anche senza una programmazione tramite controller è necessario attivare l'*STP (Spanning Tree Protocol)*, che permette alla rete di stabilire un percorso per collegare i due host, interrompendo gli altri collegamenti.

Se si prova ad usare il comando *pingall* subito dopo aver costruito la topologia, infatti, possiamo notare che gli host non sono in grado di comunicare.



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)
```

Figura 3.2: Il comando "pingall" ci permette di vedere che gli host non riescono ad inviarsi informazioni

È quindi necessario attivare l'STP. Per farlo, bisogna usare uno specifico comando, anche in questo caso presentato in uno script per semplicità.

```
sudo ovs-vsctl set Bridge 's1' stp_enable=true
sudo ovs-vsctl set Bridge 's2' stp_enable=true
sudo ovs-vsctl set Bridge 's3' stp_enable=true
sudo ovs-vsctl set Bridge 's4' stp_enable=true
```

Una volta attivato l'STP, la rete stabilisce un percorso tra i due host, che riescono finalmente a comunicare. Tuttavia, è importante notare che, così facendo, la maggior parte delle risorse rimangono inutilizzate.

Come sarà evidente in fase di analisi prestazionale, il collegamento tra i due host può coincidere con uno dei percorsi della rete SDN.

Tuttavia, gli aspetti che si vogliono evidenziare con questa dimostrazione non sono le differenze delle prestazioni in senso assoluto, ma l'incapacità di una rete classica di adattarsi allo specifico servizio richiesto.

Possiamo riutilizzare il comando *pingall* per testare nuovamente il funzionamento della rete.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

Figura 3.3: Dopo aver attivato l'STP esiste un percorso tra i due host

### 3.3 Prestazioni

Avendo a disposizione la rete programmata ed un'alternativa non ottimizzata, è possibile raccogliere dei dati riguardanti le prestazioni tramite gli strumenti presentati nel capitolo precedente.

### 3.3.1 Latenza

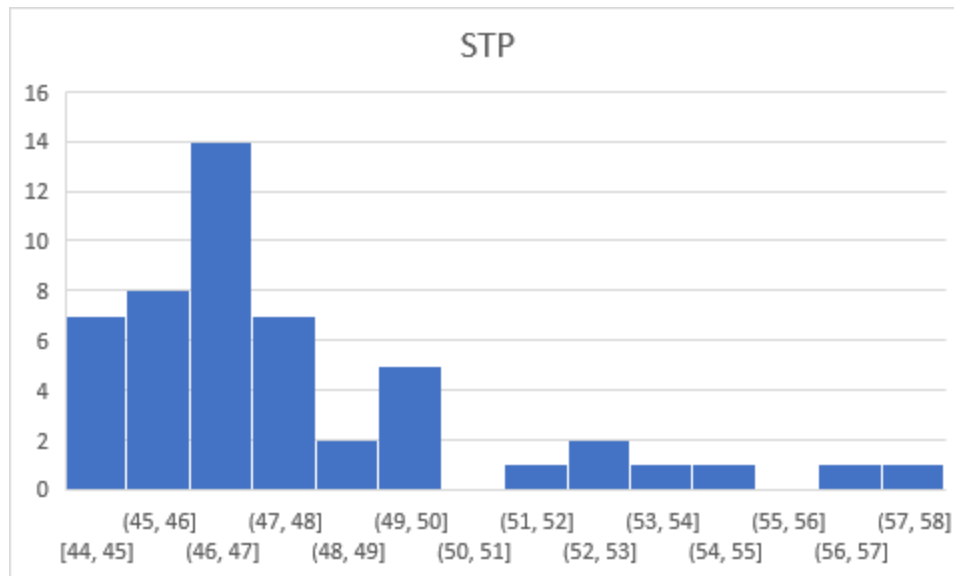


Figura 3.4: Latenza misurata su 50 misurazioni con la topologia basata su STP

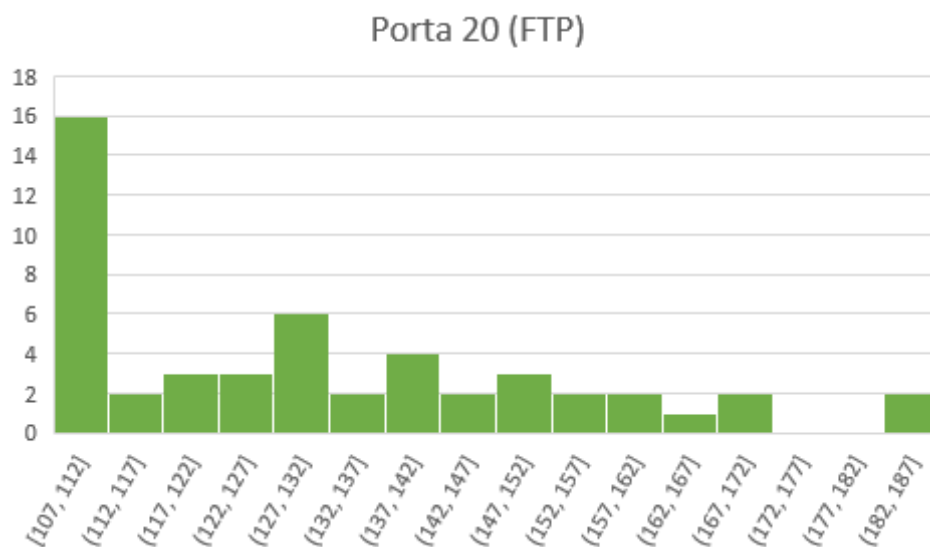


Figura 3.5: Latenza misurata su 50 misurazioni con il percorso ottimizzato per la porta 20

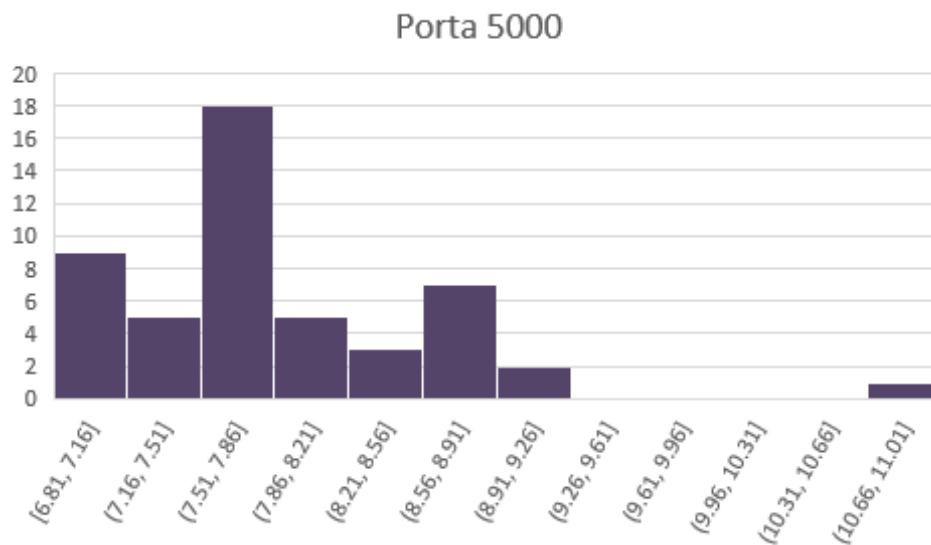


Figura 3.6: Latenza misurata su 50 misurazioni con il percorso ottimizzato per la porta 5000

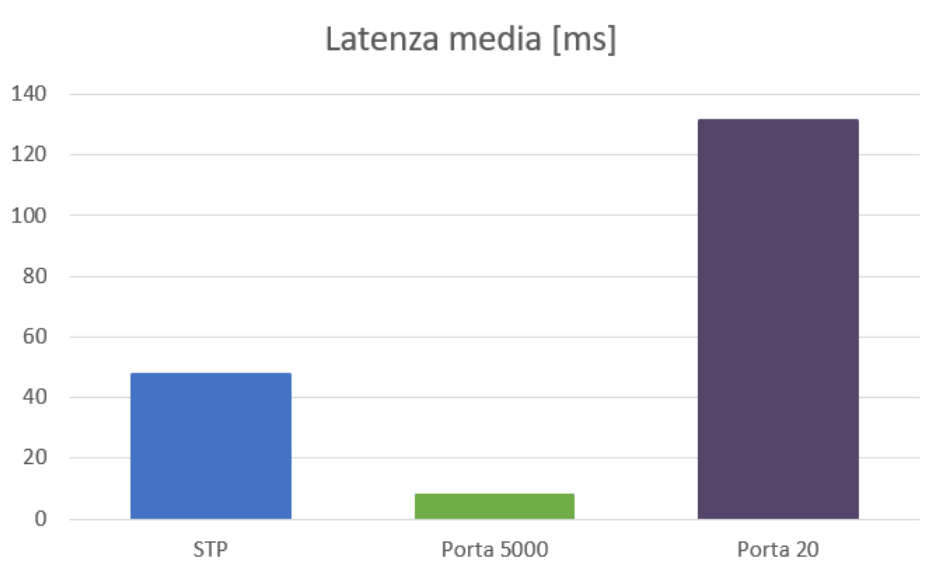


Figura 3.7: Latenza media misurata su 50 misurazioni per ogni servizio

Possiamo osservare che il percorso stabilito tramite Spanning Tree Protocol coincide con il percorso che, nella rete programmata, viene svolto dai pacchetti di servizi non specificati.

### 3.3.2 Banda

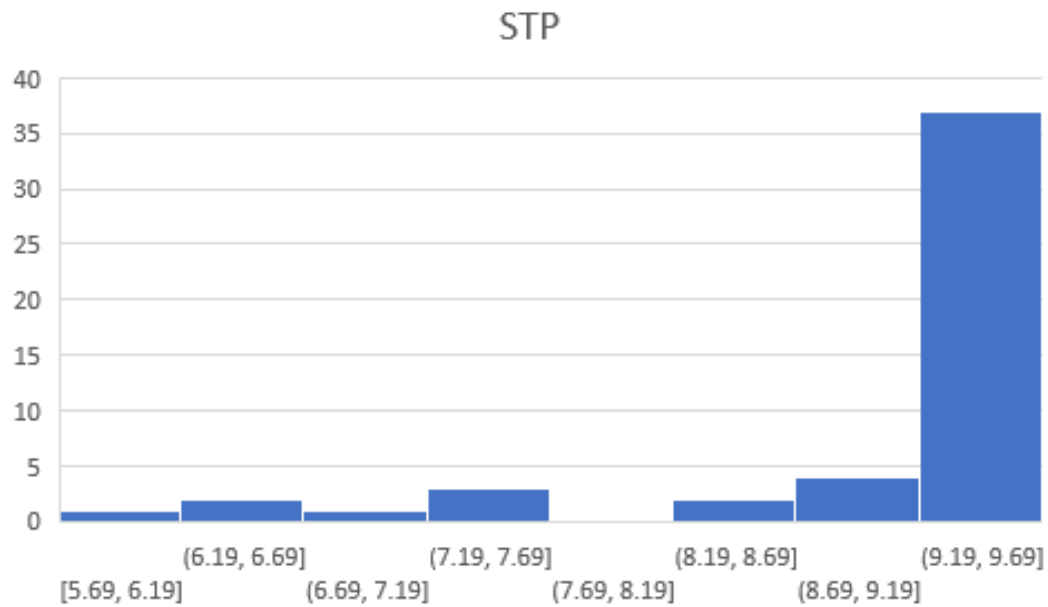


Figura 3.8: Banda misurata su 50 misurazioni con la topologia basata su STP

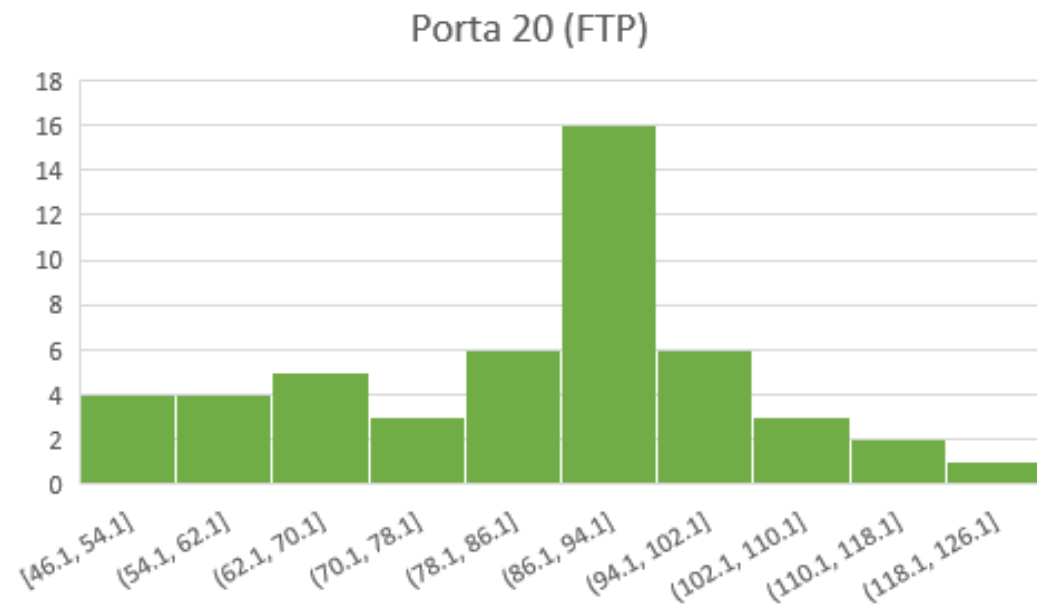


Figura 3.9: Banda misurata su 50 misurazioni con il percorso ottimizzato per la porta 20

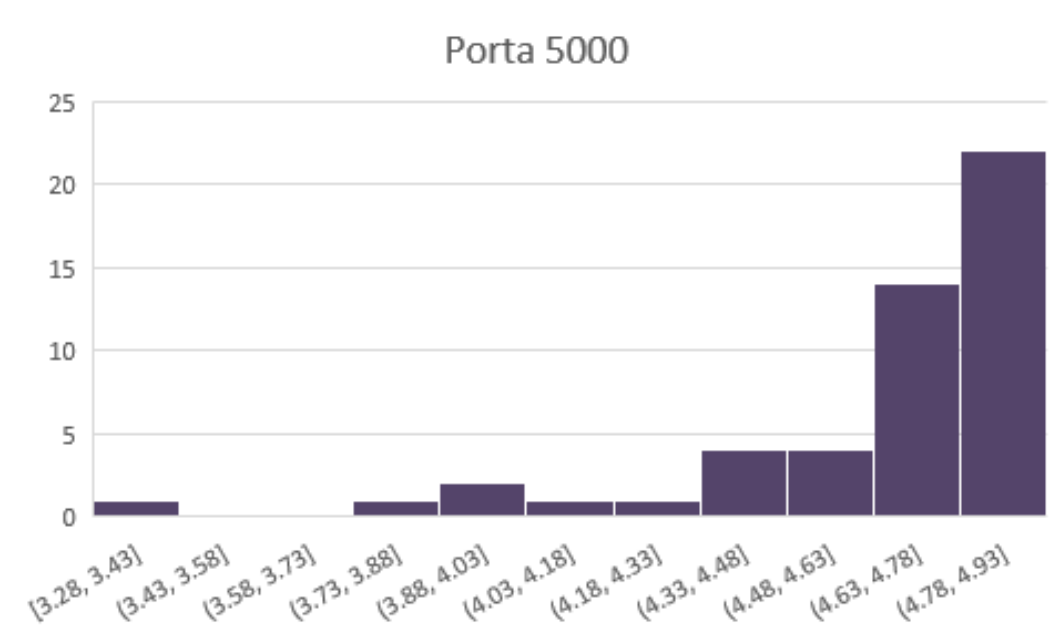


Figura 3.10: Banda misurata su 50 misurazioni con il percorso ottimizzato per la porta 5000

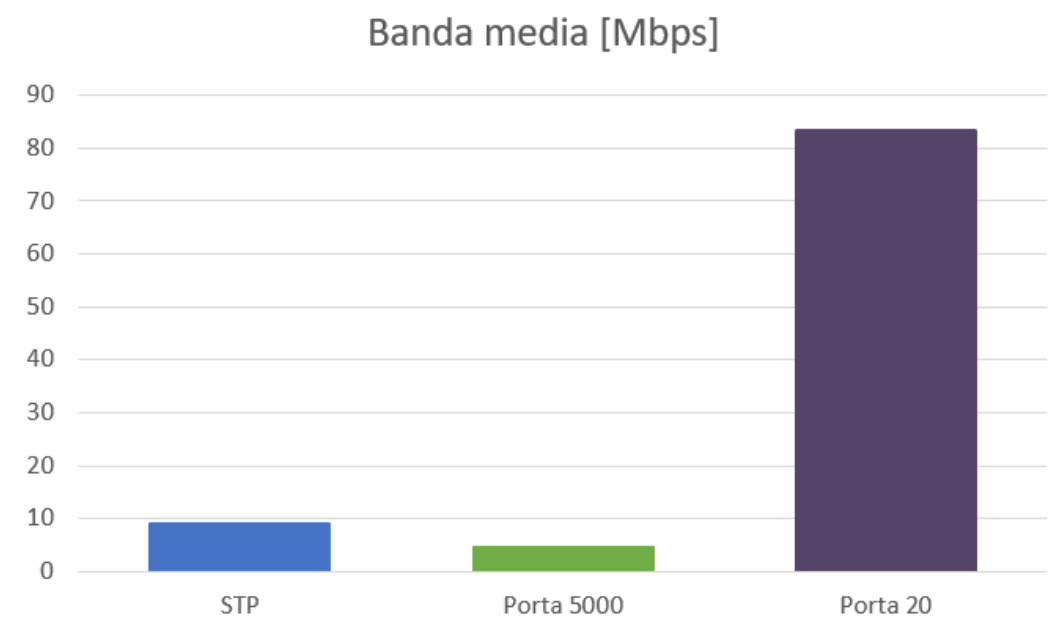


Figura 3.11: Banda media misurata su 50 misurazioni per ogni servizio

È possibile notare come le prestazioni della rete standard non siano necessariamente peggiori rispetto alla rete programmata. È importante però osservare come queste non dipendano dallo specifico servizio richiesto e non siano quindi in grado di accontentare al meglio le diverse esigenze.



# Conclusione

In questo elaborato è stato presentato il concetto di Network Slicing e come il suo impatto potrebbe influenzare diverse tipologie di servizi.

È stato poi analizzato un potente strumento di simulazione, Mininet, mostrandone alcuni utilizzi di base e spiegandone le affinità con le reti di tipo SDN. Sono stati mostrati anche diversi metodi per analizzare alcune tra le prestazioni che una rete può essere in grado di offrire.

Questi concetti e questi strumenti sono stati poi uniti per la creazione di una rete basata sull'idea di Network Slicing, dando la possibilità di osservare come questa tipologia di architettura possa impattare la costruzione, la gestione e soprattutto l'utilizzo di un'infrastruttura.

L'esempio svolto riguarda una rete piuttosto semplice. Tuttavia, queste tecnologie possono essere usate per progettare, simulare ed infine costruire reti di dimensioni e complessità molto maggiori, andandone ad influenzare molto sensibilmente l'efficienza e le prestazioni.



# Bibliografia

- [1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *NIPS*, 2012.
- [2] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset,” *The international Journal of Robotic Research*, 2013.
- [3] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Nießner, M. Savva, S. Song, A. Zeng, and Y. Zhang, “Matterport3d: Learning from rgb-d data in indoor environments,” *International Conference on 3D Vision (3DV)*, 2017.
- [4] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, A. Clarkson, M. Yan, B. Budge, Y. Yan, X. Pan, J. Yon, Y. Zou, K. Leon, N. Carter, J. Briales, T. Gillingham, E. Mueggler, L. Pesqueira, M. Savva, D. Batra, H. M. Strasdat, R. D. Nardi, M. Goesele, S. Lovegrove, and R. Newcombe, “The replica dataset: A digital replica of indoor spaces,” *arXiv preprint arXiv:1906.05797*, 2019.
- [5] P. Z. Ramirez, C. Paternesi, L. D. Luigi, L. Lella, D. D. Gregorio, and L. D. Stefano, “Shooting labels: 3d semantic labeling by virtual reality,” *IEEE AIVR*, 2020.
- [6] F. Granelli, *Computing in Communication Networks, From Theory to Practice*. Frank Fitzek and Fabrizio Granelli and Patrick Seeling, 2020.