UNIVERSITÀ DEGLI STUDI ROMA TRE

Dipartimento di Ingegneria

Corso di Laurea Triennale in Ingegneria Informatica

Tesi di Laurea

Progetto e implementazione di componenti del Controller SDN POX in ambiente emulativo Mininet

Laureando

Alexandru Rotariu

Matricola 519621

Relatore

Prof. Giuseppe Di Battista

Correlatore

Francesco Giacinto Lavacca

Anno Accademico 2019/2020

There is only one corner of the universe you can be certain of improving, and that’s your own self.

Grazie a chi mi ha supportato e sopportato.

Introduzione

Nell'approccio tradizionale al networking la maggior parte della funzionalità di rete, come quella di controllo e quella di forwarding, è implementata in apparecchi dedicati (appliance): switch e router. Inoltre, al loro interno la maggior parte della funzionalità viene elaborata in un hardware dedicato, ossia un circuito integrato studiato per risolvere una specifica applicazione di calcolo. Il problema fondamentale di questo tipo di approccio è che gli apparecchi di rete devono essere configurati manualmente, alcune operazioni richiedono molto tempo e soprattutto molti di questi sono proprietari, quindi, soltanto il produttore possiede diritti di modifica e distribuzione.

Le attuali applicazioni informatiche e i nuovi servizi richiedono che le reti di telecomunicazioni siano sempre più dinamiche, ovvero che siano in grado di adattarsi alle singole esigenze degli utenti. Ciò ha portato allo sviluppo dell'approccio Software Defined Networking (SDN). Con questo nuovo tipo di architettura si vuole separare il piano di controllo dal piano dati che attualmente convivono negli apparecchi di rete, lasciando a questi ultimi soltanto il compito di forwarding e centralizzando il controllo dell'intera rete in applicazioni dedicate che ne permettano la modifica software: essi sono detti controller. Questo è reso possibile grazie all'introduzione di nuovi protocolli come Openflow.

In questa tesi si approfondiscono tutti questi caratteri e si mettono in pratica, grazie all' emulatore di reti SDN detto Mininet; inoltre si andrà a sviluppare un nuovo componente per il Controller POX, presente in Mininet, analizzando in seguito i risultati.

Indice

Capitolo 1

Architettura della rete

1.1 SOFTWARE DEFINED NETWORKING

Con l'ampliamento di internet, è mutato notevolmente il tipo di utenza e di conseguenza anche la quantità : se inizialmente la rete era stata sviluppata e usata all'interno di Università con fini accademici, industriali e militari, ora circa il 60% della popolazione mondiale è online.

La continua crescita dell'utenza ha portato ad un altrettanto continuo potenziamento dell'infrastruttura di rete, offrendo collegamenti e dispositivi sempre più performanti per poter gestire la grande mole di traffico, ma una rete più veloce non è detto che sia una rete migliore, difatti questo tipo di sviluppo ha causato una "Ossificazione della rete".

Tale fenomeno ha portato la rete a diventare una struttura sempre più rigida e complessa, nella quale anche un minimo cambiamento richiede ingenti risorse; parte della causa può essere attribuita alla staticità dei nodi intermedi di una rete: difatti i nodi terminali si sono continuati a sviluppare e a crescere liberamente, grazie alle semplici modalità di sviluppo e modifica, ovvero grazie al software.

Questa osservazione ha reso evidente la necessita di un'architettura che permetta di gestire dinamicamente il comportamento della rete, possibile solo per via software, così è nata una nuova architettura di rete: il *Software Defined Networking* (SDN).

SDN è un paradigma centralizzato di routing e processamento del traffico, nel quale il controllo è completamente disaccoppiato dall'instradamento dei pacchetti e questo consente di avere un piano di controllo centralizzato completamente programmabile che mette a disposizione una visione astratta delle risorse di rete per lo sviluppo di applicazioni e nuovi servizi.

Si possono osservare tre specifici livelli nell'architettura SDN:

- Application Layer
- Control Layer
- Infrastructure Layer

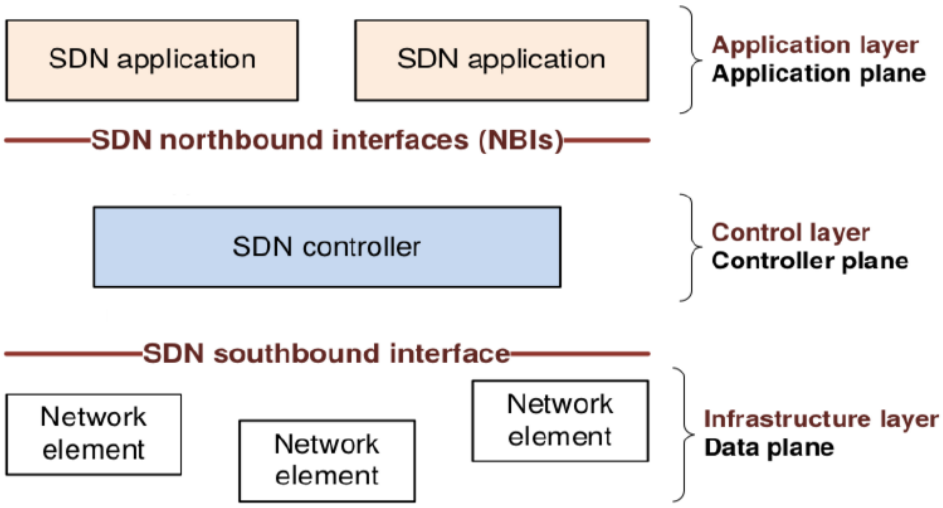


Figura 1.1: Logica base dell'architettura [1]

▪ L'**Application Layer**, ovvero **livello delle applicazioni**, come si evince dal nome, è lo spazio dedicato alle Business Applications. Tramite l'utilizzo di specifiche API, SDN permette alle applicazioni di comunicare le risorse di cui necessitano per il loro corretto funzionamento.

▪ Il **Control Layer** è il **controller SDN** il quale rappresenta il vero fulcro dell'architettura: gran parte dell'intelligenza della rete viene centralizzata in esso e svolge funzioni di middleware (l'interoperabilità) tra i dispositivi che siano essi fisici o virtuali, dei quali nasconde le specificità e l'Application Layer, mantenendo una visione globale della rete.

In particolare, è:

-fisicamente disaccoppiato dal livello di trasporto, al contrario dell'architettura attuale. In quest' ultima infatti, il controllo è confinato nei dispositivi di rete, limitando l'accesso a software esterni e quindi ostacolando sviluppi specifici a supporto degli operatori e degli utilizzatori di rete;

- centralizzato nel SDN Controller;

- aperto agli operatori e agli utilizzatori della rete. La rete può essere definita tramite il software, da cui deriva il termine Software Defined Networking;

- direttamente programmabile.

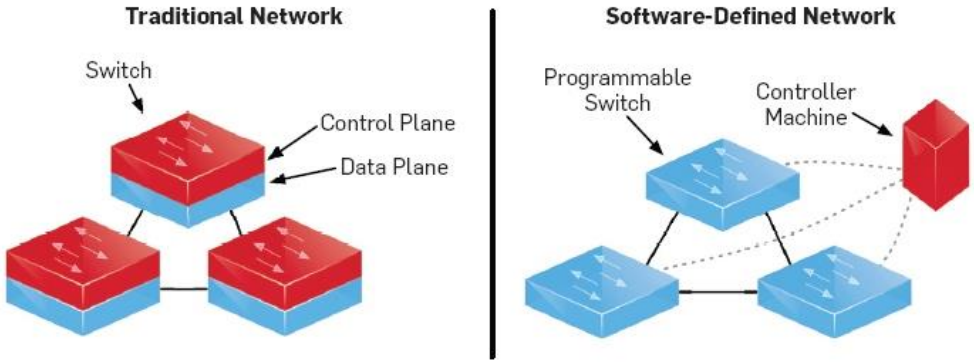


Figura 1.2: Differenza tra una rete tradizionale e SDN [2]

▪ **Infrastructure Layer** è il livello più basso che comprende quindi i dispositivi fisici e rappresenta l'infrastruttura di rete.

1.2 OpenFlow

Nelle reti SDN la selezione della destinazione di un pacchetto non è più affidata al singolo hardware, ma è gestito da software specifico, chiamato "*controller*".

Il controller può trovarsi su qualsiasi host della rete, anche in modo distribuito, e può gestire in modo diverso flussi differenti, in modo trasparente alle applicazioni, rendendo più facilmente riprogrammabili le reti.

Tutti i dispositivi devono presentare la stessa interfaccia al controller e quindi per la gestione del traffico dati è definito il protocollo **OpenFlow**, il quale specifica come deve essere strutturato uno switch per il supporto dell'architettura SDN e le regole di gestione da parte del controller.

La sua implementazione va a sostituire o affiancare il classico metodo di inoltro dei pacchetti da parte dei dispositivi che era realizzato solo tramite tabelle associative hardware andando a rendere programmabili le tabelle di classificazione ed instradamento dei pacchetti presenti negli apparati di networking. In questo modo le entries (le "righe" delle tabelle) possono essere configurate direttamente dalle applicazioni, tramite un piano di controllo esterno ai dispositivi e mediante opportune interfacce.

1.2.1 Switch OpenFlow

Uno switch è un dispositivo di rete simile ad un hub che si occupa dell'indirizzamento ed instradamento dei flussi di dati mediante l'utilizzo degli indirizzi fisici (MAC), ognuno dei quali corrisponde in modo univoco ad una sola porta di un dispositivo.

In uno switch tradizionale, l'inoltro di pacchetti (il piano dati) e l'instradamento di alto livello (il piano di controllo) avvengono sullo stesso dispositivo, mentre per uno switch OpenFlow il piano dati è disaccoppiato dal piano di controllo: il primo è implementato nello switch stesso mentre il secondo è centralizzato in un server che è in grado di determinare per ogni flusso di traffico uno specifico cammino nella rete, ovvero nel Controller (vedi Figura 1.2).

Tali switch sono muniti di uno o più "canali sicuri", detti **Secure Channel**, che mettono in comunicazione il nodo ed il controller, consentendo lo scambio di messaggi e pacchetti tramite il protocollo OpenFlow.

Inoltre, possiedono diverse tabelle tramite cui controllano e inoltrano i pacchetti; il contenuto delle tabelle è detto **entry** mentre l'entità base che rappresenta il traffico dati è detto **flow**, ovvero il flusso di pacchetti classificato in base alla compilazione dei campi della relativa intestazione.

Seguono le 3 tipologie di tabelle:

- **Flow table**, tramite esse si individuano i flussi in ingresso che corrispondono ad una delle voci della tabella, a cui è legata l'azione da svolgere con essi;

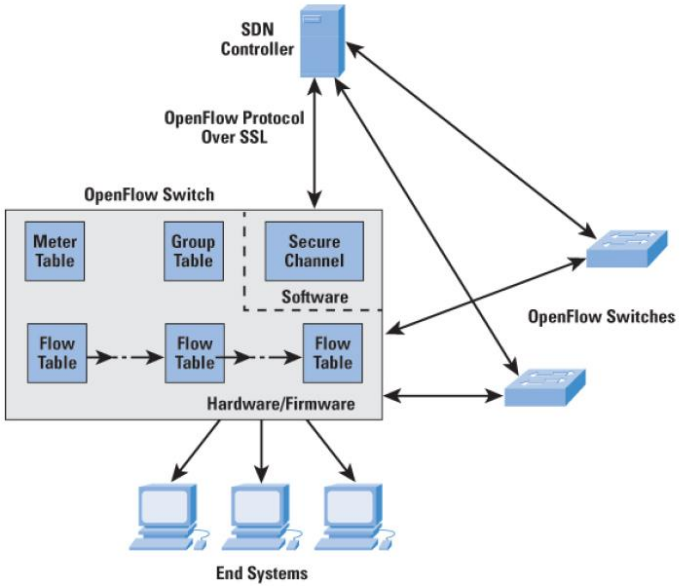


Figura 1.3: Switch OpenFlow in una rete SDN

- **Group table**, consentono di individuare un gruppo di flussi ed applicare azioni in base al gruppo di appartenenza;
- **Meter table**, consente di eseguire azioni su un flusso in base alle prestazioni.

Le azioni fondamentali che si possono associare ad una voce nella tabella dei flussi sono quattro:

- Inoltrare i pacchetti del flusso ad una o più porte, questo consente di instradare i pacchetti attraverso la rete. Gli switch implementano una serie di porte virtuali utili per l'inoltro dei pacchetti:
  - OFPP_ALL per tutte le porte tranne quella di ingresso.
  - OFPP_FLOOD per tutte le porte tranne quelle escluse in fase di configurazione.
  - OFPP_PORTIN per la porta di ingresso.
  - OFPP_LOCAL per la porta locale.
  - OFPP_CONTROLLER per il canale sicuro verso il controller.
- Inoltrare i pacchetti del flusso al secure channel dove vengono incapsulati ed inviati al controller. Tipicamente quest'azione viene utilizzata solo al primo pacchetto di un nuovo flusso affinché il controller possa decidere se aggiungere o meno la regola di flusso alla tabella del relativo switch.
- Eliminare i pacchetti di un flusso, utilizzata per questioni di sicurezza o per ridurre il traffico;

Lo switch OpenFlow porta diversi vantaggi come la possibilità di instradare traffico in blocco su percorsi più lunghi che non sono completamente utilizzati, la semplice implementazione del bilanciamento del carico ed infine facilita lo sviluppo di nuovi servizi e idee, tutte nel software sul controller SDN, nonché di accelerare nuove funzionalità e servizi.

1.2.2 Flow Table

La tipologia di tabella più importante e anche la più usata è la Flow Table (tabella di flusso), introdotta con il protocollo OpenFlow 1.0.

Ogni switch ha una (o più) flow table usate per il forwarding, ed una flow table è composta da flow entries, ciascuna formata da 5 campi (Figura 1.4):

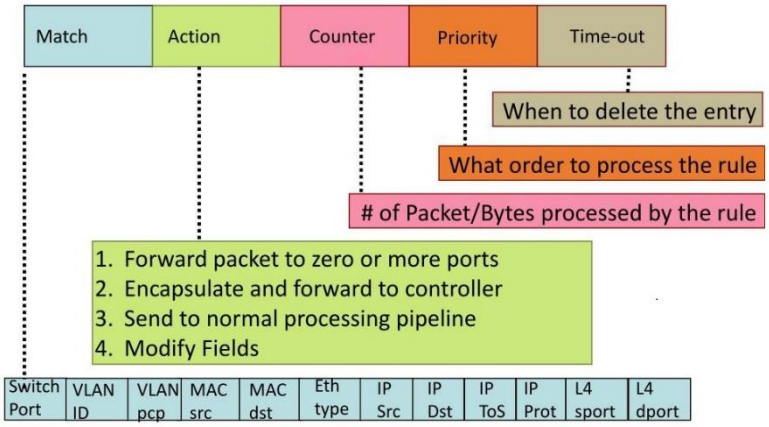


Figura 1.4: Struttura di una Flow Entry

- **Match**, una serie di regole che permettono di trovare la corrispondenza dei pacchetti, e quindi i flussi. Le regole possono essere programmate staticamente o dinamicamente dal controller.

  Tale campo è composto da più sezioni come ad esempio la porta di ingresso, l'intestazione del pacchetto e atri campi opzionali, con l'obiettivo di individuare i flussi;
- **Action**, contiene le istruzioni da eseguire su quel flusso (es. inoltra alla porta X, inoltra al controller, DROP…);
- **Counter**, utilizzato per contare quanti pacchetti corrispondenti a quella entry sono stati elaborati;
- **Priority**, individua la priorità di tale flow entry, usata nel caso di voci con più corrispondenze;
- **Time-out**, definisce il tempo massimo di una voce della tabella, dopo tale periodo esse viene cancellata. Con Idle time-out la regola scade dopo un determinato tempo di inattività, mentre con Hard time-out la regola scade dopo un determinato tempo.

1.2.3 Controller OpenFlow

Il controller OpenFlow è un'applicazione che gestisce il controllo di flusso in una rete SDN, funziona come una sorta di sistema operativo per l'intera rete, tutte le applicazioni e le connessioni passano attraverso di esso. Il protocollo OpenFlow connette il controller, attraverso il secure channel.

A livello funzionale quando un nuovo pacchetto raggiunge uno switch OpenFlow, quest'ultimo verifica se l'intestazione del pacchetto è presente tra i campi delle tabelle presenti al suo interno: se non lo è, il pacchetto viene inoltrato al controllore. A questo punto il controller SDN ha pieno potere decisionale riguardo il percorso su cui veicolare il pacchetto per farlo arrivare a destinazione, esso provvede quindi a comunicare allo switch se scartare il pacchetto oppure se inoltrarlo inserendo una nuova voce in tabella. Questo approccio potrebbe portare ad una congestione della rete, infatti, se il controllore non fosse in grado di soddisfare tutte le richieste che gli vengono inoltrate dai vari switch contemporaneamente, esso potrebbe collassare portando al blocco delle comunicazioni. Un problema di complessità e scalabilità come questo si traduce nella decisione di quanti controller avere all'interno della rete e su dove posizionarli.

Essendo un software, il controller può essere dislocato ovunque all' interno della rete (in-band), ma può essere anche collocato esternamente ad essa (out-of-band), inoltre vi è la possibilità di utilizzare più controller contemporaneamente in modo da sopperire a eventuali problemi.

I controller possono essere implementati utilizzando diversi metodi basati su diversi linguaggi di programmazione e framework , alcuni dei più comuni sono:

-NOX basato su linguaggio C++/python;

-POX anche esso basato su linguaggio python e simile al Nox;

-Floodlight in java;

1.2.4 Messaggi OpenFlow

Ogni coppia switch-controller è connessa tramite un canale sicuro OpenFlow attraverso il quale il controller configura e gestisce lo switch, riceve eventi e spedisce pacchetti attraverso lo switch; per poter effettuare tali operazioni il protocollo OpenFlow prevede l'utilizzo di messaggi la cui struttura è definita dallo stesso. I messaggi sono divisi in tre tipologie:

Controller-To-Switch

Questi messaggi sono inviati dal controller e possono richiedere una risposta dallo switch; consentono al controller di gestire lo stato logico dello switch, inclusi dettagli e configurazioni delle voci delle tabelle. I messaggi di questa tipologia sono i seguenti:

- Features, attraverso un messaggio features-request il controller chiede l'identità e le capacità di uno switch, lo switch risponde a tale messaggio con un features-response;
- Configuration, con questo messaggio il controller richiede o imposta la configurazione dei parametri dello switch, la risposta viene inviata solo nel caso di una richiesta di informazioni;
- Modify-State, questo messaggio permette al controller di aggiungere, eliminare e modificare le flow-entries o le group-entries delle tabelle e di impostare le proprietà delle porte dello switch;
- Read-State, utilizzato dal controller per richiedere informazioni allo switch come la configurazione corrente, le statistiche e le capacità;
- Packet-out, attraverso questo messaggio il controller può inviare pacchetti in uscita da una specifica porta dello switch o può inoltrare i pacchetti ricevuti dallo switch attraverso un messaggio Packet-in;
- Flow-mod, grazie al quale si possono inserire, cancellare e modificare le regole presenti nella flow table di uno switch;
- Barrier, utilizzati dal controllore per garantire che le dipendenze dei messaggi siano soddisfatte o per ricevere notifiche quando un’operazione viene completata;

Asincroni

Questi messaggi sono inviati dallo switch senza alcuna sollecitazione da parte del controller; vengono inviati per avvisare il controller di un cambio di stato o dell'arrivo di un nuovo pacchetto di cui non si hanno voci per elaborarlo. I messaggi asincroni principali sono i seguenti:

- Packet-in, utilizzato dallo switch per trasferire un pacchetto al controller, solitamente utilizzato quando non vi sono voci nella tabella di flusso che corrispondono a quel pacchetto a parte la miss-table;
- Flow-Removed, inviato dallo switch per informare il controller che una voce di flusso è stata rimossa dalla flow table, la causa può essere una richiesta di un controller o l'esaurimento del time-out;
- Port-Status, attraverso questo messaggio lo switch informa il controller del cambiamento della configurazione o dello stato di una porta.

Simmetrici

Questi messaggi possono essere inviati sia dal controller che dallo switch e non hanno bisogno di alcuna sollecitazione; vengono generalmente utilizzati per verificare lo stato della connessione, per misurarne la banda e la latenza. I messaggi sincroni principali sono i seguenti:

- Hello, scambiato tra controller e switch ad inizio connessione;
- Echo, può essere inviato da entrambi, il primo viene denominato echo-request e ne segue dall'altra parte un messaggio del tipo echo-reply; viene utilizzato per verificare che la connessione rimanga attiva, per misurarne la latenza o la banda.

Capitolo 2

Tecnologie usate

2.1 Mininet

Mininet è un emulatore open source di reti che gestisce terminali di rete (hosts), switches, routers e collegamenti (links), su un singolo kernel Linux.

Mininet consente di creare, personalizzare, condividere e testare facilmente reti SDN, presentando dei risultati reali (e non risultati dati da modelli matematici come quelli che si otterrebbero con un simulatore).

Essendo un emulatore, i risultati dei test dipendo in parte dalla macchina su cui vengono eseguiti, ma grazie ad una virtualizzazione leggera e ai network namespaces, che permettono un notevole risparmio di risorse, i risultati non vengono influenzati in modo rilevante.

I network namespaces consentono la creazione di un dominio di rete virtuale con un proprio insieme di interfacce, indirizzi IP, tabella di routing ecc. e si connettono al mondo esterno utilizzando collegamenti Ethernet virtuali tra due endpoint, in genere un endpoint si trova nel namespace locale e l'altro nel namespace globale.

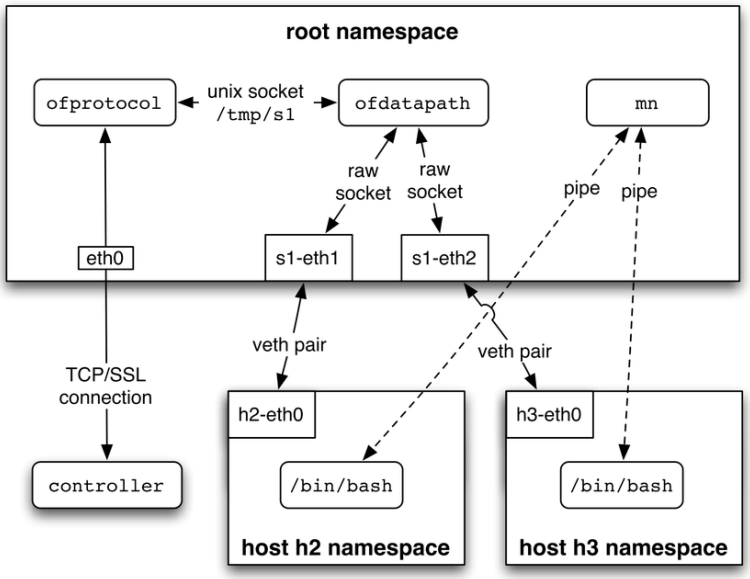


Figura 2.1: Esempio dell'approccio di Mininet con due host [3]

Un host Mininet si comporta esattamente come una macchina reale, i programmi che vengono eseguiti possono inviare pacchetti attraverso quella che sembra una vera Ethernet interfaccia, con una data velocità e ritardo del collegamento. I pacchetti vengono elaborati da quello che sembra un vero switch Ethernet, router o middlebox, con una determinata quantità di accodamento.

2.1.1 Vantaggi

Mininet ha diversi punti di forza: [4]

- Velocità: l'avvio di una semplice rete richiede pochi secondi, dunque il ciclo di run-edit-debug può essere molto breve.
- Possibilità di creare topologie personalizzate: rete con un unico switch, topologie più ampie simili a internet, un centro dati o qualsiasi altra cosa.
- Possibilità di eseguire programmi veri e propri: tutto ciò che funziona su Linux è disponibile per l'esecuzione sui singoli switch e host creati.
- Possibilità di personalizzare l'inoltro dei pacchetti: gli switch di Mininet sono programmabili utilizzando il protocollo OpenFlow e le funzionalità testate su di essi possono essere facilmente trasferite in switch reali.
- Mininet può essere eseguito su un semplice computer portatile, su un server, su una macchina virtuale, su macchina nativa Linux e sul cloud.
- Possibilità di condividere e replicare il codice: chiunque possegga un computer ha la possibilità di eseguire il codice una colta copiato opportunamente.
- La facilità di utilizzo: di possono creare ed eseguire esperimenti scrivendo semplici (o complessi, se necessari) script in linguaggio Python.
- Il codice è Open Source: si può esaminare e modificare il codice sorgente scaricabile dal sito *https://github.com/mininet* .
- Mininet e in fase di sviluppo attivo: è possibile interagire direttamente con la comunità di sviluppatori.

2.1.2 Limitazioni

Mininet ha anche qualche punto debole: [4]

- Eseguire una rete su un unico sistema è comodo ma impone alcune limitazioni: le risorse dovranno essere bilanciate tra gli host della rete.
- Mininet utilizza un unico kernel Linux per tutti gli host virtuali: non è quindi possibile eseguire software che dipenda da BSD, Windows o altri kernel differenti.
- Mininet non crea il controller, se si necessità di un controller personalizzato bisognerà implementarlo per poi poterlo utilizzare.
- Per impostazioni predefinite , la rete Mininet è isolata dalla LAN e da internet (per evitare inconvenienti nella rete). Tuttavia, ci sono diversi modi per poter connettere alla rete esterna, la rete creata.
- Per impostazioni predefinite, tutti gli host Mininet condividono il file host del sistema e lo spazio PID: bisogna porre attenzione a non terminare processi necessari.
- A differenza di un simulatore, Mininet non ha una nozione forte di tempo virtuale: questo significa che le misure temporali saranno basate sul tempo reale.

2.1.3 Installare Mininet

Vi sono diversi metodi per installare Mininet [5] , ma quello che gli sviluppatori consigliano è di avvalersi della versione per la Virtual Machine (VM), la quale include Ubuntu Linux con Mininet, tutti i tools OpenFlow preinstallati e alcune modifiche al kernel per poter supportare anche le reti più estese. Tale pacchetto dovrà poi essere installato su un software che funge da sistema di virtualizzazione (macchina virtuale, VM), come ad esempio VirtualBox (Oracle).

Molto spesso si utilizza VirtualBox solo per avviare la macchina che permette di usare Mininet, in quanto è più pratico utilizzare la VM tramite una connessione SSH che permette la creazione di una sessione remota tramite interfaccia a riga di comando (CLI) con la possibilità di lanciare i comandi e semplificare l'utilizzo di Mininet, permettendo anche l'apertura di più terminali contemporaneamente.

Dunque, l'utilizzo tipico è quello di avviare la Virtual Machine, e di connettersi ad essa dalla riga di comando del proprio terminale .

Per far sì che ciò avvenga si necessita di programmi esterni come PuTTY e Xming.

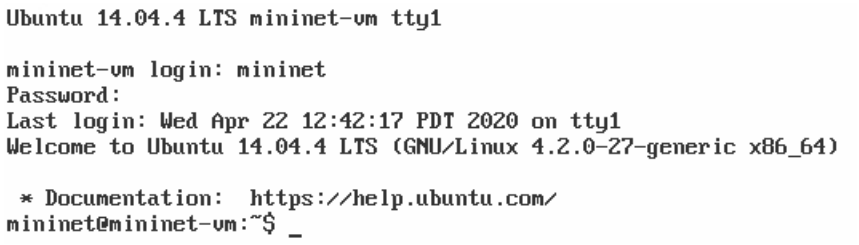


Figura 2.2: Primo avvio di mininet

2.2 Macchina Virtuale

Con il termine Macchina Virtuale (VM), si indica un software che, attraverso un processo di virtualizzazione, crea un ambiente virtuale che emula tipicamente il comportamento di una macchina fisica (PC, client o server) grazie all'assegnazione di risorse ed in cui alcune applicazioni possono essere eseguite come se interagissero con tale macchina. Tra i vantaggi vi è il fatto di poter offrire contemporaneamente ed efficientemente a più utenti diversi ambienti operativi separati, ciascuno attivabile su effettiva richiesta, senza sporcare il sistema fisico reale con il partizionamento del disco rigido. [6]

2.2.1 Configurazione usata

Per il funzionamento della VM, vengono impostate due interfacce di rete: la prima è un'interfaccia NAT usata per l'accesso a internet, la seconda invece permette alla VM di comunicare con l'host (il pc ospitante).

Configurazione: avviare l' Oracle VM e selezionare *File* → *Gestore di rete dell'host* e creare una nuova scheda con i seguenti campi compilati manualmente:

- *Indirizzo IPv4*: 192.168.56.1
- *Maschera di rete IPv4*: 255.255.255.0
- *DHCP server attivo*: 192.168.56.2
- *Indirizzo del server*: 192.168.56.2

- *Maschera del server*: 255.255.255.0
- *Indirizzo limite inferiore*: 192.168.56.101
- *Indirizzo limite superiore*: 192.168.56.254

Tali valori possono anche essere cambiati con altri IP validi.
Una volta impostata la VM si passa alla configurazione della Mininet-VM:
Mininet-VM → *Impostazioni* → *Rete* → *Scheda 1* ed abilitarla con *NAT*;
la *Scheda 2* invece deve essere abilita con *Scheda solo host* e nella sezione *Nome* selezionare la scheda appena creata.

2.3 Secure Shell

Secure Shell (SSH) è un protocollo che permette di stabilire una sessione remota cifrata tramite interfaccia a riga di comando con un altro host di una rete. [7]

2.3.1 Putty e Xming

PuTTY è un client SSH combinato con un emulatore di terminale per la gestione in remoto di sistemi informatici (es. computer, server ecc..). [8]
Per poter utilizzare il protocollo SSH in windows, è necessario installare un programma apposito in quanto, al contrario di Linux, non è già implementato sul sistema operativo. Putty permette in maniera molto semplice, mediante un'interfaccia grafica, o tramite riga di comando, di avviare una sessione SSH.
Per motivi di praticità, la maggior parte delle volte si attiva Putty dalla riga di comando, dunque si suggerisce l'installazione del software nella cartella principale (utente) in modo da avere il programma subito disponibile nel terminale:

```
C:\Users\UTENTE >
```

Per poter interagire con i diversi host creati nella rete emulata o per avviare programmi grafici tramite connessione SSH, si necessita di un programma che consenta il tunneling delle applicazioni grafiche tra Linux e Windows, detto Xming.
Xming abilita l' X11 forwarding e crea un X server per Windows consentendo il passaggio di applicazioni grafiche tra i terminali delle 2 macchine connesse, e permette

quindi di sfruttare la possibilità di eseguire più emulatori di terminale ognuna delle quali fornisce un sistema di input-output per i processi lanciati.

Nel caso di Mininet, per potersi connettere dal proprio pc alla MV si necessita dell' IP di quest'ultima.

Quindi, dopo aver avviato la Mininet-VM per ottenere l'IP viene usato il comando `ifconfig`.

```
mininet@mininet-vm:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:10:5d:b2
          inet addr:192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:78 errors:0 dropped:0 overruns:0 frame:0
          TX packets:48 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:13203 (13.2 KB)  TX bytes:9419 (9.4 KB)

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:236 errors:0 dropped:0 overruns:0 frame:0
          TX packets:236 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:18852 (18.8 KB)  TX bytes:18852 (18.8 KB)
```

Figura 2.3: Indirizzo IP che si userà per la connessione SSH

Sarà in seguito necessario avviare Xming (verrà eseguito in background), e tramite il proprio terminale (cmd in Windows) connettersi alla VM grazie al comando :

```
putty.exe -X mininet@X.X.X.X
```

ovvero si specifica che si avvia Putty abilitando l'X11 Forwarding (il quale permette la visualizzazione grafica delle applicazioni), e ci si connette alla macchina che ha come nome "mininet" e che ha un certo IP (l'ultimo campo rappresenta l'IP della Mininet-VM, trovato precedentemente con il comando `ifconfig`).

Capitolo 3

Tecnologie all'opera

3.1 Comandi base di Mininet

Dopo aver illustrato i passi necessari al funzionamento delle tecnologie, si passi ora all'analisi del loro funzionamento, tutto supportato da degli esempi.

Per avviare Mininet si utilizza di base il comando:

```
$ sudo mn
```

Dove `sudo` è l'istruzione che abilita i permessi di amministratore su Linux e `mn` è l'istruzione che permette l'avvio di Mininet.

Se non si specifica nient'altro, verrà creata una semplice topologia predefinita (chiamata minimal), adatta per testare e comprendere i primi comandi di Mininet; si vedrà in seguito come creare topologie più complesse.

```
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>
```

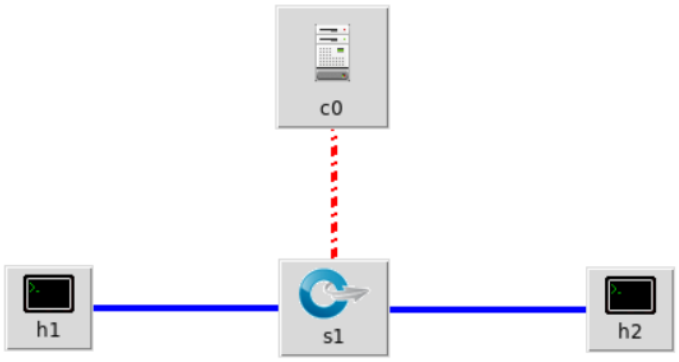


Figura 3.1: Avvio di Mininet con la topologia di base tramite comando sudo mn

Da terminale è possibile recuperare informazioni sulla topologia della rete.

Avviato l'ambiente Mininet è possibile interagire con esso tramite linea di comando CLI (Command Line Interface).

Per avere l'intera lista di comandi che possono essere utilizzati è possibile eseguire:

```
mininet> help
```

```
mininet> help

Documented commands (type help <topic>):
========================================
EOF    gterm  iperfudp  nodes        pingpair      py      switch
dpctl  help   link      noecho       pingpairfull  quit    time
dump   intfs  links     pingall      ports         sh      x
exit   iperf  net       pingallfull  px            source  xterm
```

Figura 3.2 :Esecuzione del comando help

Nonostante vi siano molti comandi disponibili, molti di essi hanno applicazioni particolari per cui generalmente non vengono usati. Tra i comandi più utili si possono individuare quelli per ottenere informazioni specifiche sulla topologia della rete: seguono degli esempi.

Per avere una panoramica generale sulla rete si usano i comandi:

`mininet> nodes` e `mininet> links`

```
mininet> nodes
available nodes are:
c0 h1 h2 s1
```

```
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
```

Figura 3.3: Esecuzione del comando nodes e del comando links

Mentre se si vogliono ottenere le informazioni sulle interfacce dei singoli dispositivi e di come essi sono collegati tra di loro devono essere usati i comandi:

`mininet> dump` e `mininet> net`

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=1584>
<Host h2: h2-eth0:10.0.0.2 pid=1586>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1591>
<Controller c0: 127.0.0.1:6653 pid=1577>
```

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

Figura 3.4: Esecuzione del comando dump e net

Tali comandi sono molto utili quando non si conosce dal principio la topologia della rete o si desidera approfondire sulle interfacce dei dispositivi per una loro eventuale gestione.

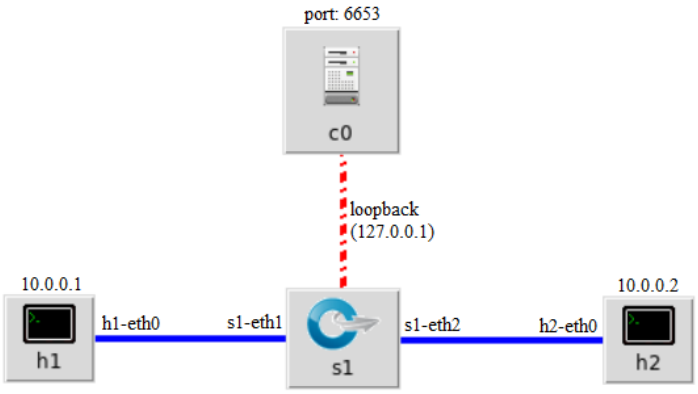


Figura 3.5:Ricostruzione della rete grazie ai comandi dump e net

Per avere più informazioni su un host (interfacce, indirizzo MAC, ecc.) si esegue il comando:

```
mininet> X ifconfig
```

dove X è il nome del dispositivo (h1, h2, s1, c0 ecc.)

```
mininet> h1 ifconfig
h1-eth0   Link encap:Ethernet  HWaddr 9e:47:6b:7d:85:54 (MAC)
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0 (IP)
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 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
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figura 3.6: Esecuzione del comando `ifconfig`

Infine, per testare la comunicazione tra gli host della rete sono possibili due alternative:

`mininet> X ping Y` oppure `mininet> pingall`

dove X è il nome dell'host che invia il messaggio ICMP Echo Request e Y è il nome dell' host che lo riceve e che risponde con un messaggio ICMP Echo Reply. Si noti che il primo comando verifica la comunicazione tra soli due hosts (X e Y), mentre il secondo verifica la comunicazione tra tutte le coppie possibili di hosts nella rete.

Per uscire dalla simulazione di Mininet e tornare alla normale esecuzione Linux è sufficiente eseguire:

```
mininet> exit
```

In caso di errori poco chiari provare sempre ad eseguire:

```
$ sudo mn -c
```

che permette di ripulire Mininet e di rimuovere eventuali errori.

3.2 Creazione di topologie articolate

Vi sono più modi per creare topologie di reti: tramite CLI, script Python o MiniEdit.

Si analizzino in dettaglio le possibilità che offre il CLI.

Per poter vedere tutte le opzioni possibili con cui avviare Mininet viene usato il comando:

```
mininet> sudo mn --help
```

I comandi che seguiranno dovranno essere intesi come opzioni, dunque dovranno essere usati in combinazione al comando `sudo mn`.

Per avviare una topologia con un singolo switch ed n host si usa l' opzione:

```
--topo = single,n
```

dove n è il numero di hosts che si desiderano collegare allo switch.

Per usare una topologia lineare, ovvero con n switch in serie si usa l' opzione:

```
--topo = linear,n
```

dove n è il numero di switch che si desidera usare; si noti che ad ogni switch verrà collegato un host.

Se si desidera una topologia ad albero binario (completo), si usi l'opzione:

```
--topo = tree,n
```

dove n è la profondità a cui verranno posizionati gli host, dunque ogni cammino dalla radice (lo switch s1) ad un host qualsiasi passa esattamente per n switch.

Un esempio completo di come avviare una rete:

```
$ sudo mn --topo single,3 --mac  --controller remote
```

Le quattro istruzioni fornite a Mininet con questo comando sono:

`--topo single,3` in cui si specifica di usare un singolo switch a cui sono collegati tre host;

`--mac` che permette di assegnare ad ogni host un indirizzo MAC legato al suo indirizzo IP (es: 00:00:00:00:00:n e 10.0.0.n), invece di uno casuale;
`--controller remote` per specificare a Mininet di non avviare il suo controller predefinito in quanto ne verrà usato uno "remoto" (verrà avviato aprendo un'altra connessione SSH con la Mininet-VM, in seguito si vedrà come fare) e dato che non è stato specificato alcun IP ed alcuna porta, gli switch si connetteranno in automatico all'indirizzo 127.0.0.1 e alla porta 6633.

Se si desidera creare una topologia particolare, allora è possibile scrivere uno script Python in cui la si definisce. Di seguito verranno descritte le principali direttive per creare switch e host e per aggiungere i collegamenti tra di essi utilizzando classi, metodi, funzioni e variabili messi a disposizione in API per Mininet.
Tutte le API si possono trovare sul sito http://mininet.org/api/annotated.html, e per poterle usare in uso script basta importarle ad inizio file:

```
from mininet.net import Mininet
```

Alcuni esempi:

- *Topo*: Classe base per le topologie Mininet.
- *Mininet*: Classe principale per poi poter interagire con la rete
- *build()*: metodo per modificare nella classe topologia.
- *addSwitch()*: aggiunge uno switch alla topologia e in uscita mostra il nome dello switch.
- *addHost()*: aggiunge un host alla topologia e in uscita mostra il nome dell'host.
- *addLink()*: aggiunge link bidirezionali alla topologia .
- *start()*: inizializza la rete.
- *pingAll()*: verifica la connettività cercando di far eseguire il ping a tutti i nodi.
- *stop()*: ferma la rete.
- *dumpNodeConnections()*: effettua il dump delle connessioni da / verso un insieme di nodi.

Per avviare Mininet con la topologia personalizzata, dopo aver creato lo script, basta aggiungere un'opzione al comando di creazione delle topologie standard:

```
$ sudo mn --custom MyTopology.py --topo=MyTopo
```

dove `MyTopology.py` è il file Python creato e `MyTopo` è il nome che si è dato alla topologia.

3.3 MiniEdit

L'ultimo metodo per creare una topologia è quello di usare MiniEdit, ovvero un semplice editor di rete con un'interfaccia grafica, avviabile tramite il comando:

```
$ sudo~/mininet/examples/miniedit.py
```

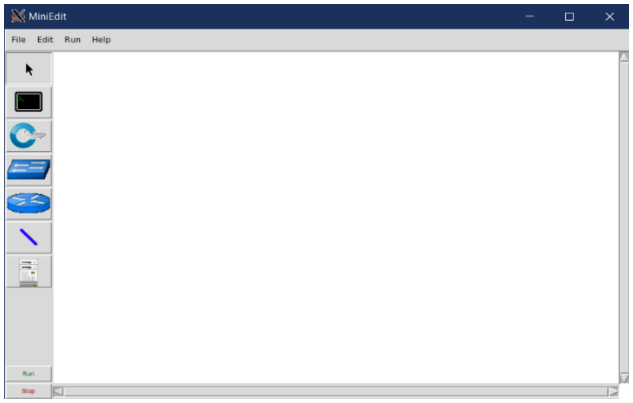


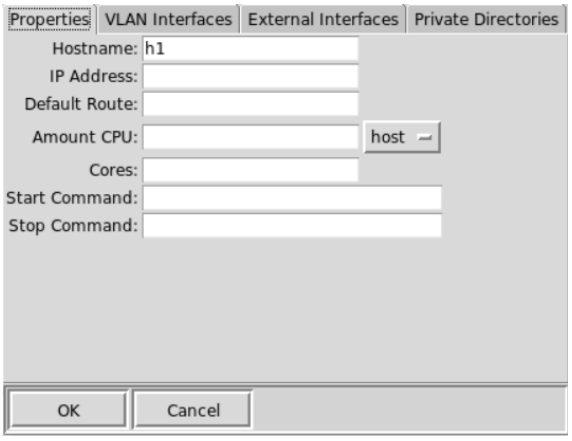
Figura 3.7: Interfaccia grafica di MiniEdit

Sulla sinistra si trovano diversi strumenti [9] :

Lo strumento "*Seleziona*" viene utilizzato per spostare i nodi nell'area di disegno. Per eliminare un nodo, selezionarlo e premere il tasto "*canc*".

Lo strumento "*Host*" crea nodi nell'area di disegno che eseguiranno la funzione dei computer host. Per usarlo nella topologia bisogna selezionare lo strumento, e posizionarlo in un punto qualsiasi dell'area di disegno. Finché lo strumento rimane selezionato, è possibile continuare ad aggiungere host facendo click.

L'utente può configurare ciascun host facendo click con il tasto destro su di esso e scegliendo "*Proprietà*" dal menu. Si possono impostare diverse proprietà come il suo nome, l' IP e il default gateway; inoltre, è possibile aggiungere delle interfacce per la LAN (VLAN) e interfacce esterne.

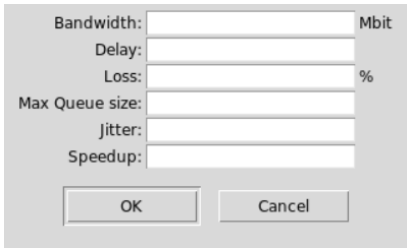


Lo strumento "*Switch*" crea switch abilitati per il protocollo OpenFlow nell'area di disegno. Questi switch dovrebbero essere tutti collegati a un controller. Lo strumento funziona allo stesso modo dello strumento "*Host*". L'utente può configurare ciascuna opzione facendo click con il tasto destro su di essa e scegliendo "*Proprietà*" dal menu.

Lo strumento "*Switch legacy*" crea uno switch Ethernet di apprendimento con impostazioni predefinite (switch classico) dunque funzionerà in modo indipendente, senza controller. Lo switch legacy non può essere configurato ed è impostato con lo Spanning Tree disabilitato, quindi non bisogna usarlo in topologie di rete che presentano cicli.

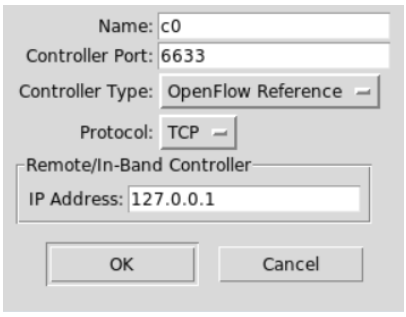
Lo strumento "*Legacy Router*" crea un router di base che funzionerà in modo indipendente, senza controller. Fondamentalmente è solo un host con IP Forwarding abilitato. Il router non può essere configurato dalla GUI di MiniEdit.

Lo strumento "*NetLink*" crea collegamenti tra i nodi nell'area di disegno. Basta selezionarlo e fare click su un nodo, trascinando il collegamento sul nodo di destinazione.



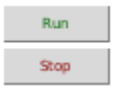
L'utente può configurare le proprietà di ciascun collegamento facendo click con il tasto destro su esso. Si possono modificare diversi campi importanti come la larghezza di banda (Bandwidth), il ritardo (Delay) e la percentuale di pacchetti persi (Loss).

Lo strumento "*Controller*" crea un controller. Si noti che si possono usare anche più controller nella stessa topologia e l'utente può configurare le proprietà di ciascun controller facendo click con il tasto destro su di esso e scegliendo "*Proprietà*" dal menu.



La configurazione comprende i campi come il nome (Name), la porta TCP che verrà usata per la comunicazione con gli switch (Controller Port) e l' IP (IP Address). Si noti che se vengono usati più controller nell' emulazione, ogni controller deve avere una TCP port diversa (6633, 6634, ecc.). Il campo più importante è il "Controller Type", che di default è impostato a "OpenFlow

Reference" e si comporta come uno switch che apprende la posizione dei diversi host e gli permette di comunicare. Se invece si desidera usare un controller personalizzato bisogna modificare il campo e scegliere "Remote Controller" (in seguito si vedrà come avviare un controller).



"*Run*" avvia la topologia creata con MiniEdit mente "*Stop*" lo interrompe. Quando la simulazione MiniEdit è in esecuzione. facendo click con il pulsante destro del mouse sugli elementi di rete, vengono visualizzate funzioni operative come ad esempio l'apertura di una finestra del terminale, la visualizzazione della configurazione dello switch o la configurazione dello stato di un collegamento su "*up*" (attivo) o "*down*" (disattivo).

Quando si desidera interrompere l'uso di Mininet, ed è in esecuzione una topologia tramite MiniEdit, è molto importante uscire prima ma Mininet con il comando `exit` e solo in seguito terminare l'esecuzione di MiniEdit con il bottone "*Stop*", altrimenti si rischia di ricevere errori indesiderati e di bloccare la macchina virtuale a causa di un bug.

Prima di avviare una simulazione è buona norma controllare anche le impostazioni generiche di MiniEdit e per farlo bisogna seguire il percorso *Edit → Preferences*

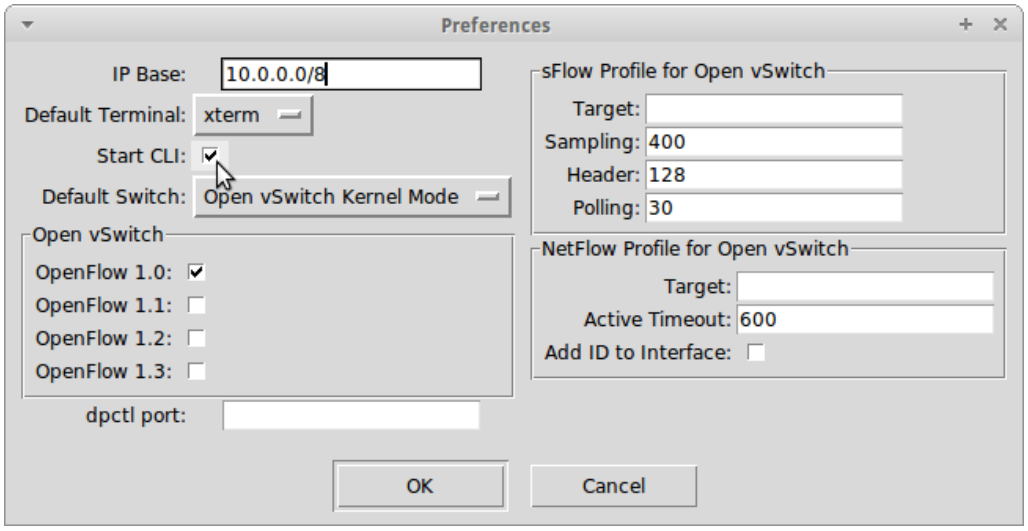


Figura 3.8: Schermata delle impostazioni di MiniEdit

Il primo campo (IP Base) permette di modificare il prefisso e la Netmask con cui verranno assegnati gli IP ai dispositivi nella rete.

Per impostazione predefinita, la finestra della console di MiniEdit non consente all'utente di accedere all'interfaccia della riga di comando di Mininet . Se si desidera poter utilizzare la CLI di Mininet quando è in esecuzione una simulazione, selezionare

la casella "*Start CLI*". Si noti come sia possibile selezionare la versione del protocollo OpenFlow che si desidera usare.

Le impostazioni devono essere configurate ogni volta che si crea una nuova topologia. Se si desidera riusare lo scenario in futuro bisogna salvarlo e per fare ciò basta selezionare "*File*" e scegliere in che modo esportare la topologia. Se si desidera salvare una versione grafica allora selezionare "*Save*" (sarà un file "**.mn*"), in questo modo, per modificare la topologia in futuro basterà aprire il file "**.mn*" tramite *File → Open.* Se invece si desidera salvare solo uno script Python eseguibile da CLI allora selezionare "*Export Level 2 Script*" (sarà un file "**.py*"); quando si vorrà avviare la topologia salvata bisognerà eseguire nella directory con la topologia desiderata il comando :

```
sudo python X.py
```

dove *X* è il nome che si è dato alla topologia.

3.4 Controller POX

Nel contesto di Mininet si trova preinstallato il controller POX, scritto in Python.
Il controller in Mininet può essere visto come un Computer: senza i suoi componenti non è molto utile; se si avvia il controller senza alcun componente, lui non sarà in grado di svolgere alcuna operazione dunque più componenti si aggiungono, più le sue funzionalità aumentano.
Quando si avvia o si crea una topologia di rete personalizzata su Mininet, il controller è impostato su "default" (OpenFlow Reference), ovvero viene usato un controller generico con componenti di base, dunque è possibile utilizzare solamente reti lineari (ad esempio una rete con dei cicli non funzionerebbe).
Invece impostando il controller come "remoto" (Remote Controller), è possibile gestire i componenti (detti components nella struttura di POX) del controller, decidendo quali attivare e quali no, tutto tramite CLI.
Alcune components dei controller POX (vedi ACTION di OF):

- **forwarding.hub**: trasforma gli OF switch in classici switch, e per fare ciò imposta nelle flow table di ogni switch l'azione FLOOD (Non-OpenFlow

flood) per qualsiasi pacchetto ricevuto, ovvero viene inoltrato su tutte le interfacce tranne quella di ricezione. Per verificare il funzionamento è sufficiente eseguire un ping tra 2 host e osservare che il primo ICMP Echo-Reply è immediato, quindi non passa per il controller;

- **forwarding.l2_learning**: permette agli OF switch di agire come uno switch L2 e di apprendere la posizione (e quindi gli indirizzi MAC) dei componenti della rete tramite il procedimento di "learning";
- **fowrarding.l2_pairs**: permette agli OF switch di agire come nel caso precedente, ma nel modo più semplice possibile, ovvero basandosi esclusivamente sugli indirizzi MAC e installando regole per ciascuna coppia di indirizzi di origine e destinazione;
- **forwarding.l3_learning**: permette agli OF switch di imparare dove si trovano gli IP, dunque gestisce una tabella con le corrispondenze da IP a MAC e la relativa interfaccia (ha senso solo nel caso in cui si abbiano più lan);
- **forwarding.l2_multi**: permette agli OF switch di agire come nel caso del component "l2_learning", con l'unica differenza che in più utilizza "openflow.discovery" per apprendere la topologia di tutta la rete: non appena uno switch scopre dove si trova un indirizzo MAC, lo fanno tutti (viene usato per trovare il percorso più breve tra 2 dispositivi). Si noti che il component "openflow.discovery" è indispensabile al funzionamento (deve essere attivato della riga di comando);
- **forwarding.l2_nx**: permette agli OF switch di inoltrare i pacchetti sulla base degli indirizzi MAC, e a differenza di "l2_pair", si usano due tabelle per ogni switch: una per gli indirizzi di origine e una per gli indirizzi di destinazione, in questo modo se arriva un pacchetto da un dispositivo ignoto, viene inoltrato al controller per poter essere appreso, e se la destinazione è nota allora si inoltra sull'interfaccia interessata, altrimenti si inoltra su tutte le interfacce tranne quella del mittente. Si noti che è necessario includere "openflow.nicira" nella riga di comando;
- **openflow.spanning_tree**: usa il componente "discovery" per analizzare l'intera rete in modo da poter costruire lo "spanning tree", e disabilita le interfacce degli switch che creano dei cicli nella rete, ponendole nello stato

"NO_FLOOD"; per un corretto funzionamento viene consigliato di usarlo con le opzioni "--no-flood" e "--hold-down";

- **info.packet_dump**: mostra le informazioni relative ai "packet_in";
- **samples.pretty_log**: mostra le informazioni nel log in maniera più chiara ed evidente.

Dunque, per far funzionare una rete complessa munita di controller si segue il seguente procedimento:

- create una topologia tramite MiniEdit:
  `mininet> sudo mininet/examples/miniedit.py`");
- collocare i propri dispositivi nella rete ed impostare il controller come remoto;
- Aprire un altro terminale connesso alla macchina virtuale tramite SSH;
- Avviare il controller POX ("`$ sudo ~/pox/pox.py`") con tutti i componenti che si desiderano (forwarding.hub, forwarding.l2_learning, forwarding.l3_learning, ecc.);
- Avviare la rete da MiniEdit.

3.5 Componenti personalizzati

In questo paragrafo si andrà ad analizzare com'è possibile definire un component e quali proprietà deve seguire, con particolare attenzione alla gestione degli eventi.

3.5.1 Come scrivere e avviare un component

I components non sono altro che dei programmi in Python, dunque per prima cosa ci si può chiedere dove deve trovarsi il codice: esso può essere posizionato in una qualsiasi cartella a piacere, purché POX possa trovarlo (la ricerca avviene nelle directory specificate nella sua variabile PYTHONPATH). In generale però si consiglia di costruire i propri component (detti anche moduli) nella cartella top-level chiamata "`ext`" poiché POX la aggiunge automaticamente al percorso di ricerca di Python.

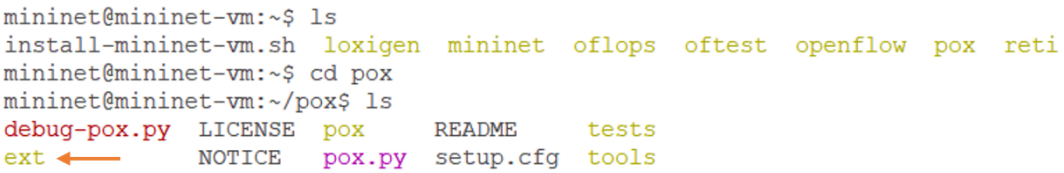


Figura 3.9: Sottocartelle di POX

Un modo comune per iniziare a creare il proprio modulo POX, è quello di copiare un modulo esistente (ad esempio uno di quelli visti in precedenza come `forwardingl2_learning.py`) nella directory "`ext`" (ad esempio `ext/my_component.py`). In seguito, si potrà modificare il nuovo file e potrà essere avviato tramite il comando "`./pox.py my_component`".

Ogni component necessita di una funzione di avvio, la quale viene invocata da POX per dire al component di inizializzarsi. Solitamente questa funzione viene chiamata "launch" e grazie ad essa gli argomenti della riga di comando vengono effettivamente passati al component come parametri.

Per capire al meglio la gestione dei parametri specificata dal CLI segue un esempio: si noti che nell'invocazione del modulo, è presente un insieme (opzionale) di parametri che gli vengono passati:

```
$ ./pox.py moduloprova  --a=3  --b  --c
```

Si osservi la funzione "launch" del modulo e si facciano delle osservazioni, sapendo che per una peculiarità, tutti gli argomenti vengono passati alla funzione come stringhe, dunque in questo caso "a" riceverà la stringa "3"; se si volesse assegnare loro altri tipi, bisognerebbe convertirli tramite funzioni apposite:

```
def launch (a, b="testo", c=True):
    print "moduloprova:", a, b, c
```

- "`a`" non ha alcun valore di default, e ciò rende tale argomento non opzionale; se lanciassimo il nostro modulo senza parametri riceveremmo un errore causato dalla mancanza di un valore per "`a`";
- nella definizione della funzione, l'unica variabile che accetta una stringa è "`b`", ma da CLI (riga di comando) non le viene passato alcun valore, di conseguenza nello script Python riceverà il valore "`True`" di default. Ovviamente se non fosse stata specificata nella riga di comando avrebbe avuto la stringa "`testo`", impostata di default nella funzione;

- “c” ha come valore di default “True”, ma se si volesse passare il valore “False” come si potrebbe fare? Se si usasse “--c=False”, si passerebbe solo la stringa “False” e non il valore vero e proprio. Se si provasse con solo “--c”, si lascerebbe a “c” il suo valore di default “True”. Si noti come non è possibile assegnare il valore desiderato da riga di comando, quindi questo è uno di quei casi in cui si deve convertire esplicitamente il valore da una stringa nel tipo che si desidera effettivamente: per convertire un valore intero o un valore in virgola mobile, è possibile semplicemente usare “int()” o “float()”, integrati in Python. Per i booleani si consiglia di usare la funzione “str_to_bool()” contenuta nella libreria “pox.lib.util”.

3.5.2 Attivazione di component

POX ha un elemento chiamato ”core”, che funge da punto centrale per la gran parte delle API di POX. Il suo scopo è quello di fornire un punto di raccolta tra i componenti: invece di importare un componente all’interno di un altro per farli interagire, i componenti si “registrano” sull’oggetto “core”, il quale può essere usato da tramite per la comunicazione. Per far sì che un component possa sfruttarlo, deve importare la libreria tramite la seguente dichiarazione : “from pox.core import core”.
In generale, per registrare un componente si utilizza il metodo “core.register()”, il quale prende due argomenti: il primo è il nome che vogliamo usare per il nuovo component, il secondo è l’oggetto che desideriamo registrare.

```
class MyComponent (object):
  def __init__ (self, an_arg):
    self.arg = an_arg
    print "MyComponent instance registered with arg:", self.arg

  def foo (self):
    print "MyComponent with arg:", self.arg


def launch ():
  component = MyComponent("spam")
  core.register("thing", component)
  core.thing.foo() # prints "MyComponent with arg: spam"
```

Figura 3.10: Registrazione di un componente come “core.thing”

La gestione degli eventi di POX si adatta al paradigma di publish/subscribe, ovvero: alcuni oggetti pubblicano eventi ed altri possono iscriversi ad essi per poterli ascoltare. In questo modo quando si verifica l'evento di interesse si possono attivare delle parti del codice. Una classe che genera eventi li dichiara in una variabile chiamata "`_eventMixin_Events`".

```
class Chef (EventMixin):
  """
  Class modeling a world class chef

  This chef only knows how to prepare spam, but we assume does it really well.
  """
  _eventMixin_events = set([
    SpamStarted,
    SpamFinished,
  ])
```

Figura 3.11: Esempio di una classe che genera due eventi

Dato che è di particolare interesse sapere quando un evento si è concluso, si ha bisogno di un metodo che "ascolti". Tali metodi prendono sempre un singolo argomento che è l'evento stesso. Supponendo che "SpamFinished" sia un evento, in figura 3.12 è indicata una possibile gestione.

```
def spam_ready (event):
  print "Spam is ready!  Smells delicious!"
```

Figura 3.12: Gestione di un evento

Bisogna inoltre impostare la funzione "`spam_ready()`" come "ascoltatore" per l'evento "SpamFinished":

```
chef.addListener(SpamFinished, spam_ready)
```

Figura 3.13: Attivazione di una funzione al lancio di un evento

3.5.3 Scheletro di un component

Per iniziare a scrivere un component, un'alternativa al modificare uno script già esistente in POX, è quella di usare uno scheletro, che è possibile trovare direttamente in POX seguendo il percorso `pox/ext/skeleton.py`.

Anche nello scheletro, come in ogni script, dopo una breve specifica di cosa esso faccia, si devono importare le librerie necessarie per il corretto funzionamento di un qualsiasi componente.

```
# Import some POX stuff
from pox.core import core                     # Main POX object
import pox.openflow.libopenflow_01 as of      # OpenFlow 1.0 library
import pox.lib.packet as pkt                  # Packet parsing/construction
from pox.lib.addresses import EthAddr, IPAddr # Address types
import pox.lib.util as poxutil                # Various util functions
import pox.lib.revent as revent               # Event library
import pox.lib.recoco as recoco               # Multitasking library

# Create a logger for this component
log = core.getLogger()
```

Figura 3.14: Librerie dello scheletro

Si necessita anche di un gestore degli eventi che, in questo caso, notifica che il componente è stato correttamente caricato ed è pronoto a svolgere il suo compito. Tale metodo viene invocato quando POX passa allo stato attivo (si ascolta l'evento del metodo "`launch()`").

```
def _go_up (event):
  log.info("Skeleton application ready (to do nothing).")
```

Si passa all'analisi della funzione "`launch()`": come già detto in precedenza, quando il nome del component è specificato nel CLI, POX chiama automaticamente questa funzione; si possono aggiungere tutti i parametri che si ritengono necessari. Nello scheletro della funzione, per esempio, si hanno due argomenti, dove il primo è obbligatorio (in quanto non è specificato un valore di default), mentre il secondo è opzionale. Si può inoltre notare che il metodo stampa delle informazioni all'utente esplicitando i valori dei parametri che gli sono stati passati, ed infine viene impostato il metodo "`_go_up()`" come "ascoltatore" dell'evento "UpEvent" in modo da notificare all'utente l'effettiva attivazione del componente.

```
@poxutil.eval_args
def launch (foo, bar = False):

  log.warn("Foo: %s (%s)", foo, type(foo))
  log.warn("Bar: %s (%s)", bar, type(bar))

  core.addListenerByName("UpEvent", _go_up)
```

Infine, si possono dichiarare tutte i metodi necessari al corretto funzionamento del componente, senza alcun vincolo. Per invocare una funzione dalla riga di comando si può usare la seguente sintassi:

```
def breakfast ():

  import random
  items = "egg,bacon,sausage,baked beans,tomato".split(',')
  random.shuffle(items)
  breakfast = items[:random.randint(0,len(items))]
  breakfast += ['spam'] * random.randint(0,len(breakfast)+1)
  random.shuffle(breakfast)
  if len(breakfast) == 0: breakfast = ["lobster thermidor aux crevettes"

  log.warn("Breakfast is served:")
  log.warn("%s and spam", ", ".join(breakfast))
```

```
./pox.py skeleton:breakfast
```

Capitolo 4

Primi esperimenti

4.1 Funzionamento delle reti e del controller

Grazie all'analisi delle tecnologie, è ora possibile creare delle reti personalizzate e osservare come si comportano i vari dispositivi presenti.

Per agevolare la comprensione si inizierà da una semplice topologia, la quale verrà poi arricchita con l'aggiunta di switch. Si noti che in tutti gli esempi presenti, per verificare la comunicazione tra gli host, si farà uso dei ping.

Il primo esempio si basa sulla topologia in figura 4.1 creata con MiniEdit, dove il controller è stato impostato su "remoto". Ovviamente se si esegue subito un ping, esso fallirà, ovvero si otterrà un messaggio ICMP "Host Ureachable": ciò avviene in quanto il controller non è stato avviato, dunque lo switch non può inoltrargli il pacchetto.

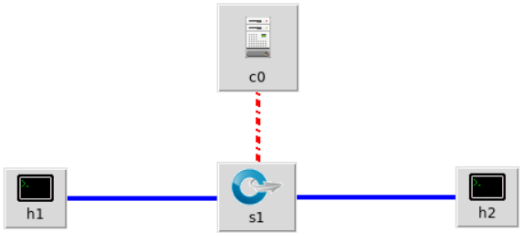


Figura 4.1: Topologia 1

Per avviare il controller si fa uso di un ulteriore connessione SSH, come visto nel capitolo precedente; si suggerisce di avviarlo con il comando :

```
$ sudo ~/pox/pox.py forwarding.l2_learning
info.packet_dump samples.pretty_log log.level --DEBUG
```

Si noti che si fa uso di alcuni component particolari (gli ultimi 3 e --DEBUG) che sono necessari nella fase di apprendimento delle funzionalità del controller: essi infatti permettono la stampa di messaggi sulla schermata del controller, per ogni sua operazione (saranno approfonditi a fine esempio).

Dopo aver attivato il controller, se si verifica lo stato della tabella dello switch s1 tramite il comando `dpctl dump-flows`, si può notare come essa sia inizialmente vuota. Se si riesegue il ping tra h1 ed h2, andrà a buon fine grazie al lavoro del

controller, che avrà installato le regole all'interno della tabella dello switch: per verificarlo è sufficiente riusare il comando `dpctl dump-flows`.

Si passi ora all'analisi dei log presenti nella schermata del controller con particolare attenzione anche ai messaggi OpenFlow, analizzati grazie a WireShark.

In seguito all' attivazione del controller, si può osservare come esso si sia connesso allo switch s1 e per fare ciò c'è stato uno scambio di pacchetti tra il controller e lo switch interessato. Si noti che tale procedimento viene eseguito per ogni switch nella rete.

```
mininet@mininet-vm:~$ sudo ~/pox/pox.py forwarding.l2_learning info.packet_dump samples.pretty_lo
g log.level --DEBUG
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:info.packet_dump:Packet dumper running
[core                  ] POX 0.2.0 (carp) going up...
[core                  ] Running on CPython (2.7.6/Oct 26 2016 20:30:19)
[core                  ] Platform is Linux-4.2.0-27-generic-x86_64-with-Ubuntu-14.04-trusty
[core                  ] POX 0.2.0 (carp) is up.
[openflow.of_01        ] Listening on 0.0.0.0:6633
[openflow.of_01        ] [00-00-00-00-00-01 1] connected
[forwarding.l2_learning ] Connection [00-00-00-00-00-01 1]
```

Figura 4.2: Connessione di uno switch dalla schermata del controller

Nella figura 4.3 le colonne indicano: IP mittente, IP destinatario, Protocollo, Lunghezza e Info.

Dato che mittente e destinatario hanno lo stesso IP, analizzando il pacchetto è possibile trovare la loro porta e capire la direzione del messaggio (dalla figura 4.2 si può osservare che la porta usata dal controller è la 6633).

127.0.0.1	127.0.0.1	OF 1.0	76	of_hello
127.0.0.1	127.0.0.1	OF 1.0	76	of_hello
127.0.0.1	127.0.0.1	OF 1.0	76	of_features_request
127.0.0.1	127.0.0.1	OF 1.0	244	of_features_reply
127.0.0.1	127.0.0.1	OF 1.0	80	of_set_config
127.0.0.1	127.0.0.1	OF 1.0 + OF 1.0	148	of_flow_delete + of_barrier_request
127.0.0.1	127.0.0.1	OF 1.0	76	of_barrier_reply

Figura 4.3: Inziale scambio di pacchetti tra controller e switch s1

Il primo messaggio, ovvero "Hello" viene inviato dal controller durante l'impostazione della connessione per la negoziazione della versione, con conseguente risposta dello switch. Se la negoziazione non riesce, viene inviato un messaggio di errore di tipo HelloFailed.

In seguito, il controller invia un FeatureReq allo switch sul canale di trasporto, chiedendogli le sue "capacità", ed esso risponderà con un FeautureRes.

Il messaggio SetConfig viene mandato dal controller per impostare nello switch il tipo di frammentazione da eseguire.

Infine, il controller manda un messaggio FlowMod per azzerare la tabella dello switch, insieme ad un BarrierReq che imposta un punto di sincronizzazione, indicando che tutti i messaggi di stato precedenti siano completati; segue la risposta dello switch.

Eseguendo il ping tramite il comando `h1 ping -c1 h2`, i log sulla schermata del controller vengo aggiornati ancora una volta, mostrando i tipi di pacchetti che il controller vede passare (ARP e ICMP) e l'installazione delle regole nello switch da un host all'altro. Si noti che nella schermata viene usato il MAC Address degli host: per avere una corrispondenza con gli IP è sufficiente usare il comando `X ifconfig`, dove X e il nome dell'host (quindi h1 o h2), nel CLI di Mininet.

```
[dump:00-00-00-00-00-01 ] [ethernet][arp]
[dump:00-00-00-00-00-01 ] [ethernet][arp]
[forwarding.l2_learning ] installing flow for 9a:2f:6a:d8:98:0c.2 -> 7a:4a:d4:7e:ac:c8.1
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 7a:4a:d4:7e:ac:c8.1 -> 9a:2f:6a:d8:98:0c.2
[dump:00-00-00-00-00-01 ] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning ] installing flow for 9a:2f:6a:d8:98:0c.2 -> 7a:4a:d4:7e:ac:c8.1
[dump:00-00-00-00-00-01 ] [ethernet][arp]
[forwarding.l2_learning ] installing flow for 9a:2f:6a:d8:98:0c.2 -> 7a:4a:d4:7e:ac:c8.1
[dump:00-00-00-00-00-01 ] [ethernet][arp]
[forwarding.l2_learning ] installing flow for 7a:4a:d4:7e:ac:c8.1 -> 9a:2f:6a:d8:98:0c.2
```

Figura 4.4: Invio dei pacchetti dalla schermata del controller

È possibile osservare che in seguito al ping effettuato con successo, la tabella di s1 presenterà delle flow rule.

Analizzando il processo con WireShark, si possono osservare diversi packet_in, packet_out e flow_add: essi indicano lo scambio di messaggi tra lo switch ed il controller (e viceversa) e l'installazione delle regole nella flow table dello switch s1.

886	13.030069000	7a:4a:d4:7e:ac:c8	Broadcast	OF 1.0	128	of_packet_in
896	13.041660000	127.0.0.1	127.0.0.1	OF 1.0	92	of_packet_out
900	13.042039000	9a:2f:6a:d8:98:0c	7a:4a:d4:7e:ac:c8	OF 1.0	128	of_packet_in
909	13.044918000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
912	13.045273000	10.0.0.1	10.0.0.2	OF 1.0	184	of_packet_in
921	13.048286000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
924	13.048624000	10.0.0.2	10.0.0.1	OF 1.0	184	of_packet_in
933	13.051499000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
2089	18.062593000	9a:2f:6a:d8:98:0c	7a:4a:d4:7e:ac:c8	OF 1.0	128	of_packet_in
2174	18.094967000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
2178	18.095330000	7a:4a:d4:7e:ac:c8	9a:2f:6a:d8:98:0c	OF 1.0	128	of_packet_in
2193	18.097678000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add

Figura 4.5: Scambio di pacchetti per consentire la comunicazione

Nel secondo esempio si fa uso della topologia in figura 4.6 e del controller con gli stessi component dell'esempio precedente.

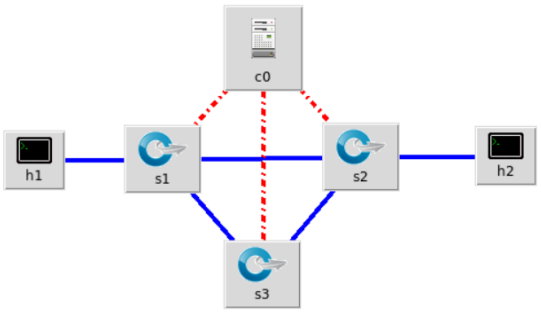


Figura 4.6: Topologia 2

Effettuando il ping tra i due host si nota come i pacchetti non arrivino a destinazione.

Ciò avviene perché nella rete è presente un loop (s1-s2-s3) dunque si ha un ciclo infinito di pacchetti, causato dal fatto che il controller di default non applica il protocollo spanning tree il quale mantiene inattive alcune interfacce degli switch in modo da garantire che la rete rimanga connessa, ma priva di loop.

Per attivare tale protocollo si fa uno del componete "**openflow.spanning_tree"**, introdotto nel capitolo 3, paragrafo 4; per attivare il controller con l'aggiunta di quest'ultimo component si usa:

```
$ sudo ~/pox/pox.py forwarding.l2_learning
info.packet_dump
samples.pretty_log log.level --DEBUG samples.pretty_log
openflow.spanning_tree --no-flood --hold-down
openflow.discovery host_tracker
```

Osservando i log si può notare grazie a "openflow.discovery", il controller apprende la topologia della rete e con l'uso di "openflow.spanning_tree" ne rimuove i loop, trovando il percorso più breve.

Per verificare quali interfacce sono state disabilitate si fa uso del comando:

```
mininet> dpctl dump-ports-desc
```

il quale indica per ogni dispositivo le sue interfacce, e per ognuna di esse, il suo stato, la capacità ed il tipo; il campo "config" indica se la porta è attiva o meno: se è impostata su NO_FLOOD allora è disabilitata, ovvero i pacchetti non vengono inoltrati su quella interfaccia.

```
*** s1 ----------------------------------
OFPST_PORT_DESC reply (xid=0x2):
 1(s1-eth1): addr:5a:8a:dc:52:27:f5
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:f6:db:51:89:5a:1d
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s1-eth3): addr:aa:9e:d4:4e:45:7d
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:32:0f:bd:52:b9:45
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

```
*** s2 ----------------------------------
OFPST_PORT_DESC reply (xid=0x2):
 1(s2-eth1): addr:0a:bb:81:c0:5f:3e
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s2-eth2): addr:4e:a4:0c:1a:4f:e2
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s2-eth3): addr:e6:c3:13:70:5b:e2
     config:     NO_FLOOD
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s2): addr:06:82:90:14:2b:48
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

Figura 4.7: Differenza tra un'interfaccia attiva ed una disattiva

Effettuando ora un ping tra gli host è possibile osservare che effettivamente la comunicazione avviene e che le flow table vengono riempite con le regole che ci si aspettano: i pacchetti passeranno tra s1 e s2, mentre s3 sarà tenuto fuori in quanto formerebbe un percorso più lungo.

4.2 Gestione manuale dei flussi

Dopo aver appreso come lavora il controller, è possibile provare a installare automaticamente le regole nelle flow table degli switch, in ottica di una gestione futura per la realizzazione di un component.

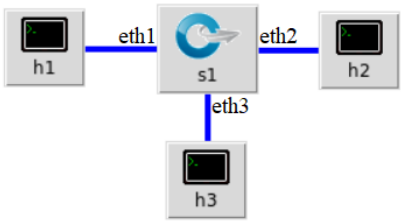


Figura 4.8: Topologia 3

La gestione può avvenire a 3 diversi livelli: rete, MAC e IP.

Nei prossimi sotto paragrafi verrà usata la topologia in figura 4.8, creata con il comando: `$ sudo mn --topo=single,3 --controller=none --mac`.

Eth1, eth2 ed eth3, rappresentano rispettivamente l'interfaccia 1, la 2 e la 3 dello switch s1.

In seguito all'avvio della rete, è possibile verificare che la flow table di s1 sia inizialmente vuota (grazie al comando `dpctl dump-flows`) e che gli host non riescano a comunicare (eseguendo il comando `pingall`).

Tramite il comando `mininet> sh ovs-ofctl add-flow s1 action=normal` si configura l' OF switch per funzionare come uno switch tradizionale (lo si fa aggiungendo una flow rule nella flow table dello switch): si noti che la flow table ora dispone della regola appena inserita ed inoltre che ora gli host possono comunicare.
Ma dato che l'obiettivo è quello della gestione manuale dei flussi, questa scorciatoia non è utile ai fini dell'apprendimento, dunque deve essere evitata; per cancellare la regola si usi il comando `mininet> sh ovs-ofctl del-flows s1`. Si noti come ora gli host non riescano più a comunicare.

4.2.1 Gestione dei flussi a livello 1 (rete)

A questo livello di gestione ci si interessa solo della rete fisica, si andranno quindi a usare le porte degli switch per guidare i flussi.
Data la rete in figura 4.8, si supponga di voler far comunicare gli host h1 ed h2: dovendo gestire le interfacce, l'idea è quella di inoltrare tutti i pacchetti ricevuti su eth1, ad eth2, e viceversa. In questo modo si è sicuri che tutti i pacchetti arrivino al destinatario: per fare ciò si usano i comandi

```
mininet> sh ovs-ofctl add-flow s1 priority=500,in_port=1,actions=output:2
mininet> sh ovs-ofctl add-flow s1 priority=500,in_port=2,actions=output:1
```

In talo modo si indica di aggiungere allo switch s1 due regole caratterizzate dal `in_port`, il quale indica la porta su cui arriva il pacchetto e da `output:x`, che indica l'inoltro del pacchetto sulla porta `x` dello switch. Il campo "priority" viene usato quando si devono gestire tanti flussi, e si desidera dare un ordine di priorità tra tutti quelli che effettuano il "match", ovvero la corrispondenza. Si noti che nelle rule appena installare non ci sono campi su cui effettuare il match dunque le regole valgono per tutti i pacchetti.
Segue un esempio per comprendere al meglio la gestione della priorità: aggiungendo il comando `mininet> sh ovs-ofctl add-flow s1 priority=32768,action=drop`

Rieseguendo un ping tra h1 ed h2 è possibile osservare come essi non riescano a comunicare, nonostante le flow rule siano impostate. Questo accade perché è stato dato un livello di priorità (a quest'ultima regola) maggiore di quello dato alle altre, quindi essa viene eseguita sempre per prima (anche essa effettua il match con tutti i pacchetti) e di conseguenza tutti i pacchetti vengono scartati. Notare che priority=32768 è un valore di default.
Per eliminare quest'ultima regola basta usare il comando
`sh ovs-ofctl del-flows s1 –strict`, il quale elimina tutte le rules con parametri di default.

È possibile notare che l'host h3 non riesce a comunicare con nessuno, in quanto nelle flow table è stato specificato come gestire i flussi delle prime due interfacce, ma non è stato detto nulla della terza.
Si potrebbe provare ad aggiungere le regole relative all'interfaccia dello switch collegato al terzo host:

```
mininet>sh ovs-ofctl add-flow s1 priority=501,in_port=1,actions=output:3
mininet>sh ovs-ofctl add-flow s1 priority=501,in_port=3,actions=output:1
```

Si noti come sia stata impostata una priorità più alta a queste due nuove regola, altrimenti si sarebbero sovrascritte le precedenti rule in quanto viene usata la stessa in_port (ovvero la 1).
Grazie a queste regole è possibile ora far comunicare h1 con h3 (si può verificare con un ping) ma per via della priorità(501>500) i pacchetti spediti da h1 vengono inoltrati sull'interfaccia 3 e non più sulla 2.
Si è dunque giunti alla conclusione che se si utilizza un matching confinato solo alle porte fisiche degli switch, non si possono avere regole per nodi con grado maggiore di due, dunque in questo caso non è possibile far comunica tutti gli host tra di loro.

4.2.2 Gestione dei flussi a livello 2 (indirizzi MAC)

A questo livello di gestione ci si interessa sia della rete fisica che degli indirizzi MAC degli host, quindi in base al MAC del destinatario si andranno a inoltrare pacchetti su diverse interfacce.

Dunque, si per rendere possibile la comunicazione tra h1 ed h2 bisogna specificare che se il pacchetto ha come mittente h1, dunque si ha un match sul suo MAC Address, allora deve essere inoltrato sull'interfaccia 2, ovvero quella di h2, e viceversa:

```
mininet>sh ovs-ofctl add-flow s1
dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,actions
=output:1
mininet> sh ovs-ofctl add-flow s1
dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,actions
=output:2
```

dove con `dl_src` e dl_dst indicano rispettivamente l' indirizzo MAC del mittente e quello del destinatario.

Eseguendo un `pingall` si può notare che nonostante siano state aggiunge le regole per la comunicazione tra h1 e h2, nessun host riesce a comunicare con gli altri: questo è dato dal fatto che prima dello scambio di pacchetti ICMP, c'è lo scambio di pacchetti ARP, i quali non eseguono il match con alcuna regola (i pacchetti ARP Request hanno come destinatario ff:ff:ff:ff:ff:ff).

Per far si che i pacchetti ARP vengano inoltrare a tutte le porte (quindi sia un messaggio broadcast) si usa il comando:

```
mininet>sh ovs-ofctl add-flow s1
dl_type=0x806,nw_proto=1,action=flood
```

In questo modo si indica che tutti gli Ethernet frames (`dl_type=0x806`) di tipo ARP Request (`nw_proto=1`) devono essere inoltrati su tutte le interfacce dello switch tranne quella da cui è arrivato il pacchetto (`action=flood`).

Si noti come ora h1 ed h2 riescano a comunicare.

Con la stessa logica delle regole precedenti, è possibile far comunicare anche h3 con gli altri host:

```
mininet>sh ovs-ofctl add-flow s1
dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,actions
=output:3
mininet>sh ovs-ofctl add-flow s1
dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,actions
=output:1
mininet>sh ovs-ofctl add-flow s1
dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03,actions
=output:3
mininet>sh ovs-ofctl add-flow s1
dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02,actions
=output:2
```

Eseguendo nuovamente `pingall` è possibile verificare il corretto funzionamento della rete.

4.2.3 Gestione dei flussi a livello 3 (indirizzi IP)

A questo livello di gestione si è interessati sia della rete fisica che degli indirizzi IP degli host.

La logica è la stessa applicata nel paragrafo precedente, dunque conoscendo gli IP è possibile inoltrare i pacchetti sulle porte dello switch corrette:

```
mininet>sh ovs-ofctl add-flow s1
dl_type=0x800,nw_src=10.0.0.1,nw_dst=10.0.0.2,actions=output:2
mininet>sh ovs-ofctl add-flow s1
dl_type=0x800,nw_src=10.0.0.2,nw_dst=10.0.0.1,actions=output:1
```

In questo modo si specifica che i pacchetti IP (`dl_type=0x800`) che hanno come mittente `x` (`nw_src=x`) e come destinatario `y` (`nw_dst=y`), devono essere inoltrati su una certa porta.

Si noti che però i pacchetti ARP, anche in questo caso, non riescono ad effettuare il match, dunque per permettere il funzionamento si aggiungono:

```
mininet>sh ovs-ofctl add-flow s1
arp,nw_dst=10.0.0.1,actions=output:1
```

```
mininet>sh ovs-ofctl add-flow s1
arp,nw_dst=10.0.0.2,actions=output:2
```

Si noti come ora i due host riescano a comunicare, dunque è possibile applicare le regole anche per l'host h3:

```
mininet>sh ovs-ofctl add-flow s1
dl_type=0x800,nw_src=10.0.0.1,nw_dst=10.0.0.3,actions=output:3
mininet>sh ovs-ofctl add-flow s1
dl_type=0x800,nw_src=10.0.0.3,nw_dst=10.0.0.1,actions=output:1
mininet>sh ovs-ofctl add-flow s1
dl_type=0x800,nw_src=10.0.0.2,nw_dst=10.0.0.3,actions=output:3
mininet>sh ovs-ofctl add-flow s1
dl_type=0x800,nw_src=10.0.0.3,nw_dst=10.0.0.2,actions=output:2
mininet>sh ovs-ofctl add-flow s1
arp,nw_dst=10.0.0.3,actions=output:3
```

È possible verificare il complete funzionamento dell rete.

Capitolo 5

Realizzazione del component

5.1 Obiettivo

Dopo aver assunto le capacità di base sulla gestione dei flussi e sull'utilizzo delle librerie Python dedicate a Mininet, si può passare allo sviluppo di un component per il controller POX.

Nelle prossime pagine si analizzeranno le procedure logiche che hanno portato allo sviluppo del component idoneo a far comunicare tra di loro tutti i dispositivi presenti in una rete, indipendentemente dalla topologia e in maniera totalmente automatica, con particolare attenzione al percorso seguito dai flussi, in quanto deve essere il più breve per la destinazione.

Le procedure sviluppate si possono dividere in quattro sezioni:

- Inizializzazione delle strutture dati;
- Gestione delle ARP e della comunicazione;
- Calcolo del percorso più breve (shortest path);
- Installazione delle regole per gestire il routing di flussi.

Si noti che non viene utilizzato alcun spanning tree, dunque per evitare il loop di pacchetti nella rete, le interfacce degli switch che non comunicano direttamente con un host vengono impostate su NO FLOOD. In questo modo lo switch inoltrerà i pacchetti in automatico solo agli host e non ad altri switch.

Inoltre, per la gestione dell'ARP, viene creata una tabella, utilizzata esclusivamente dal controller, per tenere traccia delle posizioni degli host. Questa tabella è inizialmente vuota.

5.2 Inizializzazione delle strutture dati

Per poter gestire una qualsiasi rete bisogna prima conoscere come è fatta, a tale proposito si noti la necessità di usare il component "openflow.discovery", oppure la funzione "Discovery" nel codice Python.

All'avvio di una rete con Mininet in cui viene usato Discovery, partirà una fase di inizializzazione in cui il controller rileva la presenza di tutti gli switch (tramite gli eventi ConnectionUp) e dei collegamenti tra essi (tramite gli eventi LinkEvent), grazie ai pacchetti LLDP (Link Layer Discovery Protocol) che permette ai dispositivi connessi di comunicare la propria identità e caratteristiche.

Affinché il component sia completamente automatizzato, tutte le preziose informazioni che riceve devono essere salvate in adeguate strutture dati, in modo da poter essere sfruttate semplicemente nel momento del bisogno (per esempio nel calcolo dello shortest path bisogna certamente conoscere tutti gli switch della rete e come sono connessi!).

5.2.1 Strutture dati e gestione degli eventi

Le strutture dati necessarie sono:

- `lista_connessioni`, necessaria per poter avere un riferimento a tutti gli switch;
- `lista_link`, per poter conoscere quali switch sono connessi tra di loro, e quali sono le porte coinvolte in ogni collegamento. Si noti che sarà una lista formata da quadruple che specifica i due switch e le due porte interessate;
- `nodi`, mappa che identifica per ogni nodo il suo DPID;
- `disattiva`, mappa necessaria per poter "disattivare" le porte degli switch che si interfacciano ad altri switch, al fine di evitare il loop dei pacchetti in quanto non viene usato alcun Spanning Tree. Si noti che le chiavi sono i DPID degli switch, e i valori sono delle liste composte dalle interfacce che non connettono ad un host.
- `lista_host`, per avere una visuale completa su tutti gli host della rete;

- `host_switch`, mappa che ha come chiave un intero (un valore non significativo) e come valore una quadrupla composta da:
  (MAC host, IP host, DPID switch, interfaccia dello switch a cui è connesso l'host)
- `adiacenza`, una matrice, o meglio una lista di liste, usata come matrice di adiacenza per il calcolo del percorso con Dijkstra;
- `tabellaC`, ovvero una mappa usata dal controller per la gestione delle ARP che si analizzerà nel prossimo paragrafo. Le chiavi sono gli IP degli host, e i valori sono coppie composte dal MAC dell'host e dal DPID dello switch più vicino (all'host). Questi dati possono essere estrapolati direttamente dal pacchetto ricevuto dal controller.

Come specificato nei capitoli precedenti, la gestione di una rete Mininet si basa su eventi, dunque si necessita di funzioni che permettano di "ascoltarli":

- "ConnectionUp", generato quando si attiva il collegamento tra uno switch e il controller, tramite la funzione `_handle_ConnectionUp` è possibile gestire l'evento per salvare le informazioni sul singolo switch nella struttura "lista_connessioni" e "nodi".
- Discovery fa uso di "LinkEvent" che segnala quando un collegamento si crea o si distrugge, dunque per ascoltarlo si necessita di una funzione `_handle_LinkEvent`, nella quale si possono salvare i dati sui collegamenti, ovvero quali switch e con quali interfacce sono connessi (struttura "lista_link"). Inoltre, la funzione può essere sfruttata per salvare le informazioni sulle interfacce da disabilitare, all'interno di "disabilita".
- Quando uno switch non trova una corrispondenza nella propria tabella, oppure se la voce corrispondente indica di inviare il pacchetto al controller, quest'ultimo riceverà un messaggio OpenFlow (Packet in) , di conseguenza viene attivato l'evento PacketIn, gestito dalla funzione `_handle_PacketIn`, la quale ottiene il pacchetto legato all'evento e assegna alla funzione "`gestisci`" la responsabilità di gestirlo.

5.3 Gestione delle ARP e della comunicazione

La funzione "`gestisci`" definisce il comportamento del controller in base al tipo di pacchetto.
I pacchetti possono essere divisi in due tipi: i pacchetti ARP e quelli non ARP.

5.3.1 Cos'è un pacchetto ARP

Agli host presenti in una rete sono associati due tipi di indirizzi:

- Indirizzi IPv4, a cui fanno riferimento i protocolli di livello superiore;
- Indirizzi MAC, associati all'hardware delle macchine stesse.

Per inviare un pacchetto IP ad un calcolatore della stessa sottorete, è necessario incapsularlo in un pacchetto di livello datalink, che dovrà avere come indirizzo destinazione il MAC Address del calcolatore a cui lo si vuole inviare. Per ottenere questo indirizzo si usa il protocollo ARP (Address Resolution Protocol), che, come specificato da RFC 826, è un protocollo di rete appartenente alla suite del protocollo internet (IP) versione 4 e operante a livello di accesso alla rete.
Se il pacchetto deve essere inviato ad un calcolatore di un'altra sottorete, ARP viene utilizzato per scoprire il MAC Address del gateway o del router.
L'host che vuol conoscere il MAC address di un altro host, di cui conosce l'indirizzo IP, invia in broadcast una richiesta ARP (pacchetto di ARP Request) contenente il proprio indirizzo MAC e l'indirizzo IP del destinatario di cui si vuole conoscere il MAC Address. Tutti i calcolatori della sottorete ricevono la richiesta: in ciascuno di essi il protocollo ARP verifica, confrontando l'IP proprio con quello inviato, se viene richiesto il proprio MAC Address. L'host di destinazione che riconoscerà il proprio indirizzo IP nel pacchetto di ARP-request, provvederà ad inviare una risposta (ARP Reply) contenente il proprio MAC direttamente all'host mittente (quindi in unicast).

I pacchetti ARP request/reply sono incapsulati all'interno di un frame Ethernet.
L' *Ethernet destination address* viene impostato a *ff:ff:ff:ff:ff:ff* (broadcast per indirizzi ARP) se viene effettuata un'ARP Request e il campo *type* viene impostato al valore 0x0806 (mentre per il RARP 0x8035).

I 28 byte per l'ARP Request/Reply sono strutturati in questo modo:

- hardware type: specifica il tipo di interfaccia hardware su cui l'utente cerca una risposta, per l'Ethernet si setta il campo ad 1.
- protocol type: indica il tipo di indirizzo ad alto livello che il mittente ha fornito, per l'IP si setta a 0x0800.
- hardware len e protocol len: consentono di usare ARP su reti arbitrarie perché specificano la lunghezza dell'indirizzo hardware (MAC) e dell'indirizzo del protocollo di alto livello (IP).
- ARP operation: specifica se si tratta di una richiesta ARP (valore 1), risposta ARP (valore 2), richiesta RARP (valore 3) oppure una risposta RARP (valore 4).

5.3.2 Idea da sviluppare

I primi pacchetti che gli switch (e di conseguenza il controller) riceveranno , saranno pacchetti ARP, in quanto senza di essi la comunicazione non può avvenire: si noti quindi che la funzione "`gestisci`" deve gestire il caso particolare in cui il pacchetto sia ti tipo ARP.

Dato che un pacchetto ARP può essere di tipo Request o di tipo Reply, bisogna quindi gestire anche questa differenza.

Il comportamento del controller (e quindi della funzione `gestisci`) può essere diviso in 2 casi diversi: ARP request in cui il controller non "conosce" il destinatario, o ARP request in cui il destinatario è presente nella tabella del controller.

Si supponga di avere una rete complessa a piacere, che la tabella del controller sia vuota e che ogni switch abbia la propria flow table con una sola regola, ovvero quella preinstallata dal controller che permette di fare match con qualsiasi pacchetto e come azione ha quella di inoltrare il pacchetto al controller.

Esempio: un host h1 desidera mandare un pacchetto (per esempio un Echo Request) ad un host h2, ma non conoscendo il MAC address del destinatario manda inizialmente un ARP Request in broadcast; tale pacchetto viene mandato allo switch il quale trova una corrispondenza nella propria flow table con l'unica regola, ovvero quella preinstallata, la quale ha come azione quella di inoltrare il pacchetto al controller.

Il controller appena riceve il pacchetto verifica se l'IP del mittente (h1) (ottenuto scomponendo il pacchetto ricevuto) è presente nella propria tabella e in caso negativo ne aggiunge i dati; in seguito verifica se l'IP del destinatario (h2) è presente nella propria tabella, ovvero se ha già mandato un pacchetto.

Analisi del primo caso:
Se il destinatario non viene trovato nella tabella (d'ora in poi TABLE MISS), il pacchetto viene inoltrato a tutti gli switch della rete: ciò non causerà alcun problema di loop in quanto precedentemente le porte degli switch che non si interfacciano ad un host sono state impostate su NOFLOOD, di conseguenza ogni switch inoltrerà il pacchetto solo alle interfacce collegate ad un host. Quando il pacchetto arriverà ad un host, esso ne controlla il destinatario: se esso non è il destinatario allora il pacchetto viene scartato, altrimenti viene generata un ARP Reply verso h1.

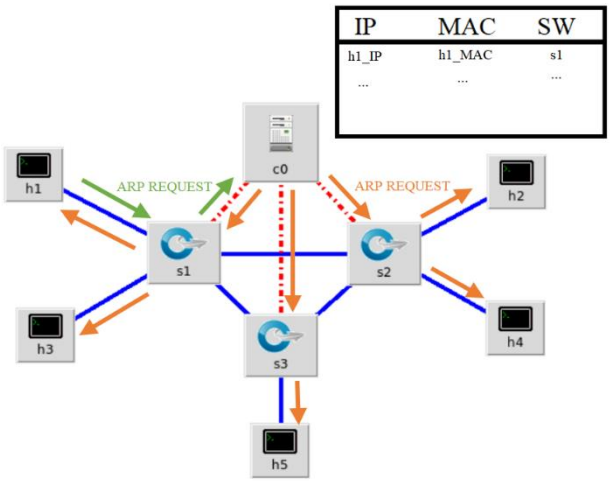


Figura 5.1: Inoltro di ARP Request agli switch causato dal TABLE MISS

L' ARP Reply viene mandata ad uno switch, e come nel caso precedente, viene inoltrato al controller il quale verifica la presenza o meno di h2 nella tabella e si comporterà di conseguenza.
In questo caso il controller conosce il destinatario (TABLE HIT), in quanto è stato salvato precedentemente: il controller dalla sua tabella può ricavare qual è lo switch più vicino all'host e inoltrargli il pacchetto mandato da h2.

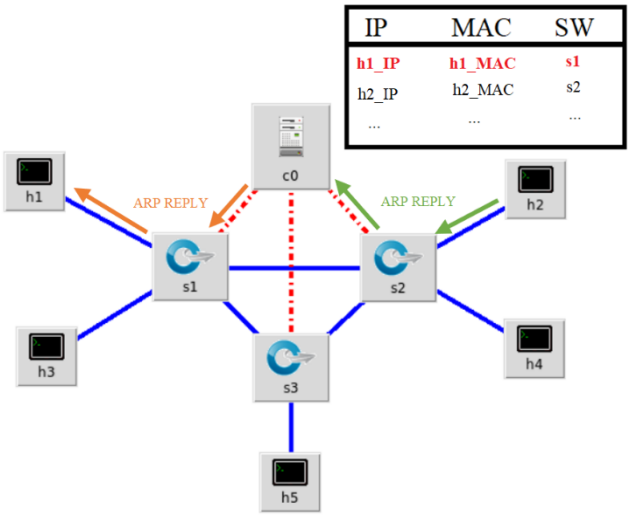


Figura 5.2: Inoltro di ARP Reply direttamente allo switch interessato grazie al TABLE HIT

Lo switch riceve il pacchetto e lo inoltra sulle porte che non sono state impostate su NOFLOOD. L'host h1 riceverà il pacchetto e verrà a conoscenza del MAC address di h2: potrà ora mandare il pacchetto ICMP Echo Request.

Analisi del secondo caso:

Se il destinatario viene trovato nella tabella, allora il controller ne conosce anche il MAC, dunque risponde con un ARP Reply (per conto di h2) direttamente allo switch che ha inoltrato il pacchetto al controller. Quando il pacchetto arriva allo switch, esso lo inoltrerà a tutti gli host a cui è connesso e di conseguenza anche ad h1.

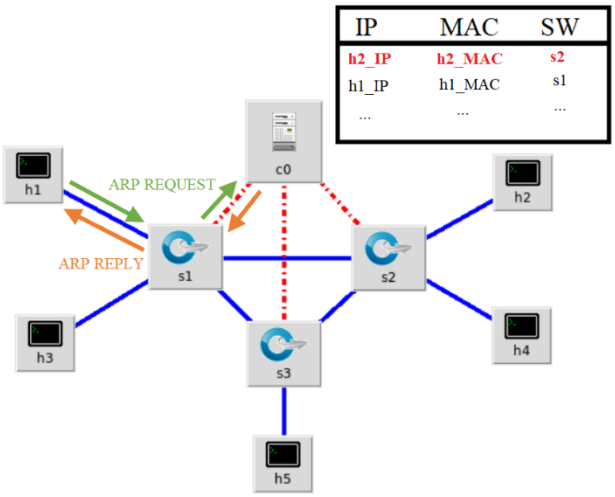


Figura 5.2: ARP Reply generata dal controller

Ora che h1 conosce il MAC di h2 può iniziare a inviare i suoi pacchetti ICMP Echo Request.

E se il pacchetto non è ARP?

Questa possibilità indica che lo scambio dei pacchetti ARP è già avvenuto, di conseguenza il controller ha la propria tabella popolata.

Quando l'host h1 manda l' ICMP Echo Request ad h2, il pacchetto viene ricevuto dallo switch più vicino ad h1, il quale non avendo alcuna nuova regola, invia il pacchetto al controller, il quale sfruttando l'algoritmo di Dijkstra, calcola il percorso più breve tra s1 e lo switch più vicino ad h2 ricavandolo dalla tabella (s2); in aggiunta inoltra il pacchetto a quest'ultimo switch.

Una volta trovato il percorso, si possono installare le regole nelle tabelle degli switch che lo compongono, in ambe le direzioni, in modo che quando h2 risponderà con un ARP Reply, troverà gli switch già predisposti anche alla comunicazione da h2 ad h1: ciò implica l'installazione di due regole per ogni switch interessato.

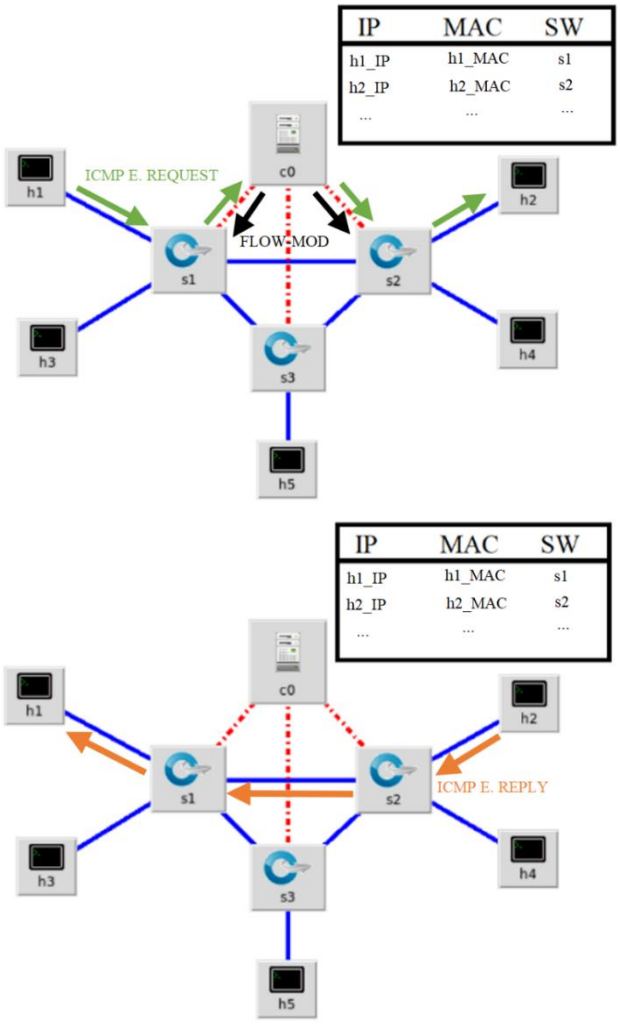


Figura 5.3 e 5.4 : Invio di ICMP Echo Request con installazione delle regole, e risposta

Nel caso di un ulteriore ICMP Echo Request da h1 ad h2, lo switch di h1 troverà una regola che corrisponde con il pacchetto corrente e quindi inoltrerà lo inoltrerà al prossimo switch del percorso calcolato con Dijkstra. Il procedimento si ripete per ogni switch fino allo switch più vicino a h2, che lo inoltra direttamente sull'interfaccia collegata all'host.

Per l' ICMP Echo Reply da h2 ad h1, la procedura si ripete in maniera analoga.

5.4 Calcolo del percorso più breve (shortest path)

Per evitare che ogni singolo pacchetto passi per il controller e aumenti la sua quantità di dati da gestire, si potrebbero trovare dei percorsi nella rete che vengano seguiti dai pacchetti in base al destinatario.

Per portare al massimo l'efficienza del component, invece di trovare un percorso qualsiasi, si è pensato di trovare il percorso più breve dal mittente al destinatario, il cosiddetto shortest path. Esistono molti algoritmi che permettono di determinarlo, ma in questo caso si è scelto di usare Dijkstra in quanto permette una implementazione compatta lavorando sulla matrice di adiacenza.

Si noti che lo shortest path non viene calcolato da un host mittente a uno destinatario, bensì dallo switch che inoltra il pacchetto al controller (switch mittente) allo switch destinatario trovato nella tabella del controller: il grafo della rete comprende solo gli switch e non gli host.

Per costruire la matrice di adiacenza della rete viene usata la funzione `costruisciMatAdiacenza`, nella quale viene inizializzata con tutti valori nulli ed in seguito tramite la struttura `lista_link`, si ricava ogni coppia di switch collegata e si aggiorna il valore della casella corrispondete nella matrice.

Una volta ottenuta la matrice completa è possibile applicare Dijkstra per trovare lo shortest path tra i due switch.

5.4.1 L'algoritmo di Dijkstra

L'algoritmo di Dijkstra è un algoritmo utilizzato per cercare i cammini minimi in un grafo con o senza ordinamento, ciclico e con pesi non negativi sugli archi. I nodi sono gli switch della rete.

Pseudocodice:

```
function Dijkstra(Grafo, sorgente, destinazione):
    For each vertice v in Grafo:                    // Inizializzazione
        dist[v] := infinito ;                       // Distanza iniziale dalla sorgente a v sconosciuta
        precedente[v] := non definita ;             // Nodo precedente di v non definito
    end for
```

dist[*sorgente*] := 0 ; *// La sorgente si trova a distanza 0 dalla sorgente*

visited := L'insieme vuoto; *//Nessuno nodo è stato ancora visitato*

unvisited := L'insieme di tutti i nodi nel *Grafo* *// Tutti i nodi del grafo non sono ancora visitati*

while *unvisited* **non è** vuoto: *// Loop principale*

u := vertice in *unvisited* con la più breve distanza in dist[] ;

rimuovi *u* da *visited* ed aggiungilo a *visited* ;

if dist[*u*] = infinito:

break ; *// tutti i vertici rimanenti sono inaccessibili dal nodo sorgente*

end if

nodi_adiacenti := tutti i nodi adiacenti al nodo u:

For each *nodo v* di *nodi_adiacenti*:

alt := dist[*u*] + dist_tra(*u*, *v*) ; // Calcola la distanza del nodo

if *alt* < dist[*v*]: *//Se è inferiore a quella precedente allora è stato trovato un nodo più vicino*

dist[*v*] := *alt* ; // *Aggiorna i dati*

precedente[*v*] := *u* ;

end if

end for

end while

path := percorso trovato usando *dist* e *precedente; // Funzione esterna per trovare la lista di switch*

return path; //Path è la lista di switch

Si noti che si necessita di diverse altre funzioni:

- `minDist`, per trovare un nodo che minimizza la distanza dal nodo corrente;
- `adiacenti`, necessaria per avere l'insieme di nodi adiacenti al nodo corrente;
- `shortest`, per ricavare la lista di switch che compongono il percorso, sfruttando le strutture dati manipolate nella funzione Dijkstra.

5.5 Installazione delle regole per gestire il routing dei flussi

Nella funzione in cui si gestisce il pacchetto, dopo aver trovato il percorso più breve tra i due switch grazie a Dijkstra, bisogna installare le regole nelle tabelle degli switch interessati.

Per fare ciò si utilizza la funzione `avvia_instal` che grazie ai parametri che le vengono passati, ovvero la lista degli switch dello shortest path, l'IP del mittente e l'IP del destinatario, installa sugli switch le regole per consentire la comunicazione tra i due host interessati.

La logica dietro alla funzione è molto semplice: avendo una lista di switch ordinata, è possibile prendere gli elementi due alla volta e, grazie alla struttura dati `lista_link`, identificare da quale interfaccia del primo switch sono connessi. Bisogna però fare attenzione al caso particolare in cui sta analizzando l'ultimo elemento della lista, il quale non avendo un nodo successivo, significa che è interfacciato con l'host interessato: con l'ausilio della struttura `host_switch`, è possibile trovare su quale interfaccia dello switch è collegato l'host destinatario; in questo modo è possibile installare anche sull'ultimo switch della lista la regola per mandare il pacchetto direttamente all'host interessato, e non sfruttare più le impostazioni di NOFLOOD che generavano traffico inutile.

```
#installa le regole su ogni switch nel percorso
def avvia_instal(self, percorso, ipSrc, ipDest):
        #per ogni nodo del percorso
        for count in range(0,len(percorso)):
                src = percorso[count]
                #se sei arrivato all'ultimo nodo intermedio
                if count == (len(percorso)-1):
                        #cerca l'interfaccia dello switch verso ipDest
                        for data in self.host_switch.values():
                                if data[1]==src and data[3]==ipDest:
                                        out_port = data[2]
                #altrimenti cerca la porta verso il prossimo nodo del percorso
                else:
                        dst = percorso[count+1]  #il prossimo nodo è la destinazione corrente
                        for lista in self.lista_link:
                                if lista[0]==src and lista[2]==dst:
                                        out_port = lista[1]
                #installa la regola su ogni switch
                self.installa_flow_rule(src, out_port, ipSrc, ipDest)
        print "FLOW RULE ISTALLATE"

#installa la regola di instradamento su uno switch
def installa_flow_rule(self, id_switch, out_port, ip_src, ip_dst):
        msg = of.ofp_flow_mod()
        msg.priority = 1000
        msg.match.dl_type = 0x0800
        msg.match.nw_src = ip_src
        msg.match.nw_dst = ip_dst
        msg.actions.append(of.ofp_action_output(port = out_port))
        core.openflow.sendToDPID(id_switch, msg)
```

Figura 5.5: Implementazione dell'installazione delle regole nelle tabelle degli switch del percorso

Capitolo 6

Realizzazione del component

6.1 Condizioni iniziali

Dopo aver realizzato il componente, bisogna testarlo e osservare i risultati da esso prodotti.

L'obiettivo desiderato è di avere un component che, in maniera totalmente automatica, permetta il funzionamento di una rete anche molto complessa.

Negli esperimenti che seguono, viene usata sempre la stessa topologia, ovvero quella della Deutsche Telekom, illustrata in figura 6.1, e realizzata con MiniEdit in figura 6.2.

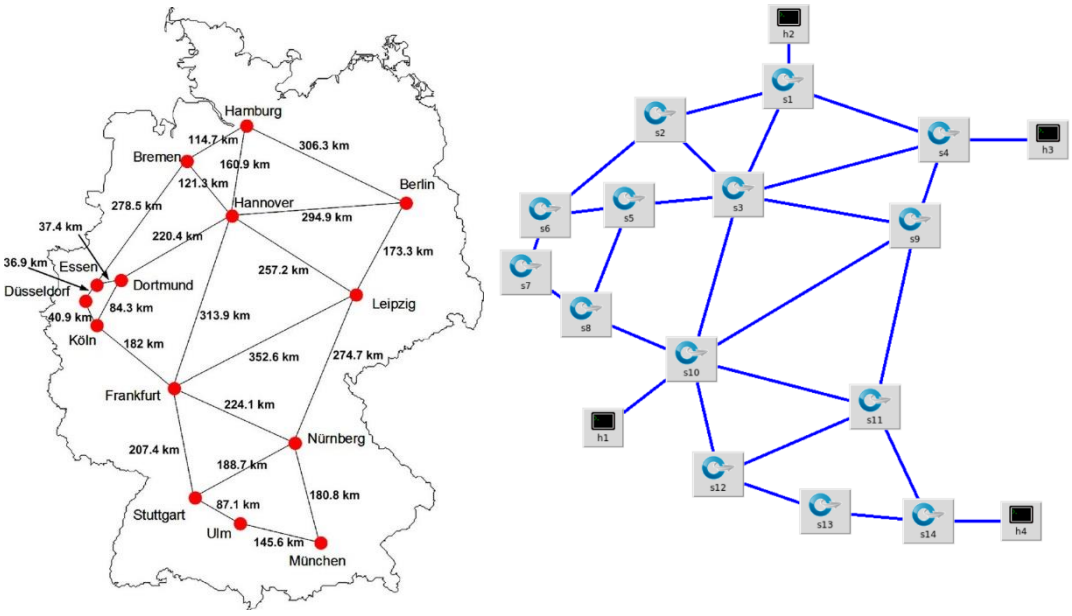


Figura 6.1 e 6.2: Topologia Deutsche Telekom e realizzazione con MiniEdit

Si noti che in figura 6.2 non è presente il controller che deve essere connesso ad ogni switch, per non appesantire l'immagine.

Gli host, necessari per testare il componente, sono stati posizionati in punti strategici in modo da avere percorsi tra gli host il più diversi possibili. Essi si trovano nelle città di Amburgo (Hamburg), Berlino (Berlin), Monaco (München) e Francoforte (Frankfurt).

I test consistono nel verificare i tempi di convergenza della rete, e per fare ciò vengono eseguiti 10 ping, ciascuno composto da 10 pacchetti ICMP, per 3 esperimenti. Si noti che ogni esperimento è legato a un percorso della rete diverso dagli altri.

6.2 Esperimento 1

Il primo esperimento è stato eseguito coinvolgendo gli host h2 ed h3, in modo da avere un primo semplice percorso.

Dato che la funzione principale del component è quella di trovare il percorso più breve, ci si aspetta che esso calcoli il percorso composto da due hops, ovvero che passi per gli switch s1 e s4.

Avviando il controller e la topologia è possibile verificare che il percorso scelto dal controller è proprio quello più breve, di conseguenza è possibile iniziare a testare la rete.

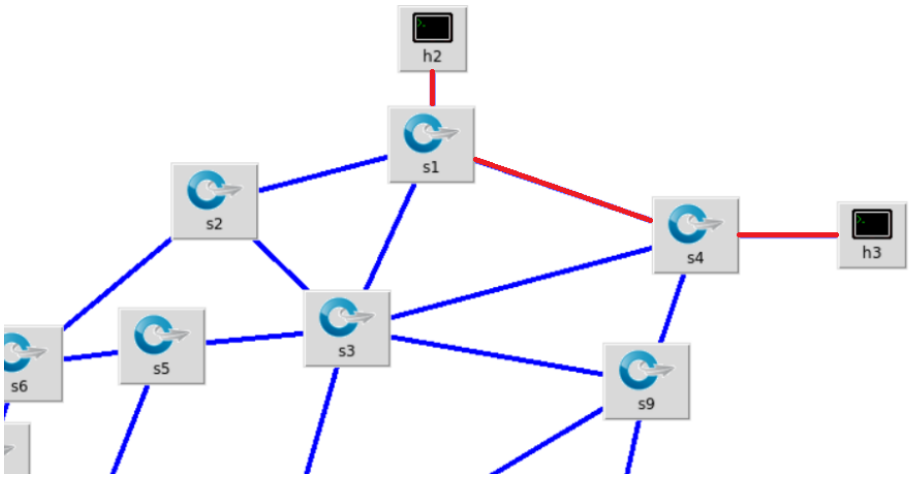


Figura 6.3: Porzione di rete interessata per l'esperimento 1

Seguono i risultati dei test.

TEST 1	1 PACK	2 PACK	3 PACK	4 PACK	5 PACK	6 PACK	7 PACK	8 PACK	9 PACK	10 PACK
1 PING	1054	55,4	0,423	0,083	0,063	0,077	0,081	0,04	0,049	0,074
2 PING	1050	77,8	0,496	0,082	0,078	0,084	0,068	0,082	0,062	0,081
3 PING	1065	63,2	0,421	0,155	0,039	0,083	0,082	0,081	0,037	0,071
4 PING	1043	44,3	0,364	0,071	0,078	0,083	0,072	0,074	0,067	0,052
5 PING	1044	80,6	0,308	0,09	0,067	0,083	0,07	0,102	0,087	0,049
6 PING	1030	67	0,425	0,085	0,076	0,084	0,085	0,083	0,08	0,082
7 PING	1054	46	0,439	0,096	0,08	0,082	0,056	0,084	0,082	0,082
8 PING	1069	69,5	0,351	0,084	0,074	0,078	0,075	0,084	0,064	0,086
9 PING	1045	37,9	0,282	0,129	0,069	0,081	0,156	0,084	0,084	0,082
10 PING	1064	63,6	0,527	0,085	0,086	0,079	0,078	0,084	0,083	0,084
MEDIA	1051,8	60,53	0,4036	0,096	0,071	0,0814	0,0823	0,0798	0,0695	0,0743
VARIANZA	129,16	185,2901	0,0054416	0,0005982	0,0001546	5,84E-06	0,0006666	0,0002198	0,0002535	0,0001602

Figura 6.4: Risultati dei test per l'esperimento 1, tempi espressi in millisecondi

Analizzando i risultati è possibile osservare come i primi due pacchetti, abbiano dei tempi di risposta molto elevati rispetto agli altri: questo è dovuto al fatto che gli switch inizialmente non hanno alcuna regola nelle loro tabelle, dunque deve essere eseguito tutto il procedimento per la gestione delle ARP, per la ricerca del cammino e per l'installazione delle rule nelle flow table.
Una volta eseguiti tutti i procedimenti ed installate le regole negli switch, i tempi di risposta e la varianza diminuiscono notevolmente.

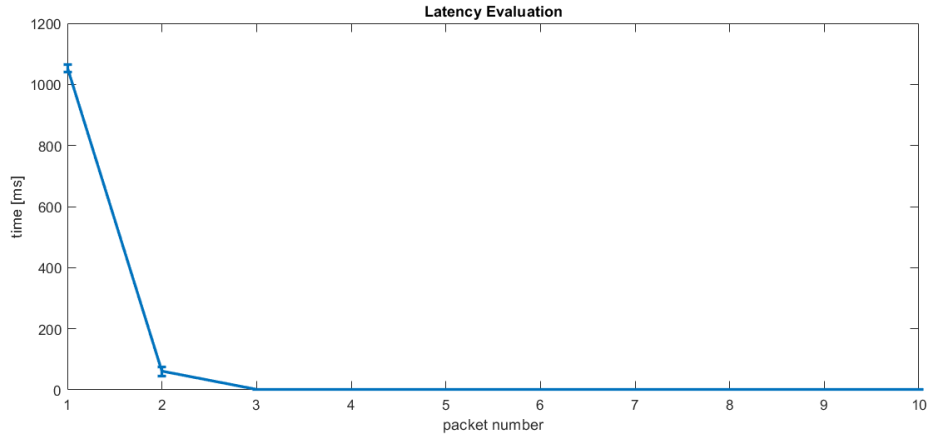


Figura 6.5: Grafico dell'esperimento 1

6.3 Esperimento 2

Nel secondo esperimento si analizzano i tempi di convergenza della rete, coinvolgendo gli host h1 ed h4.
Lo shortest path, eseguito anche dal component è composto da 3 hop: s10 – s11 – s14.

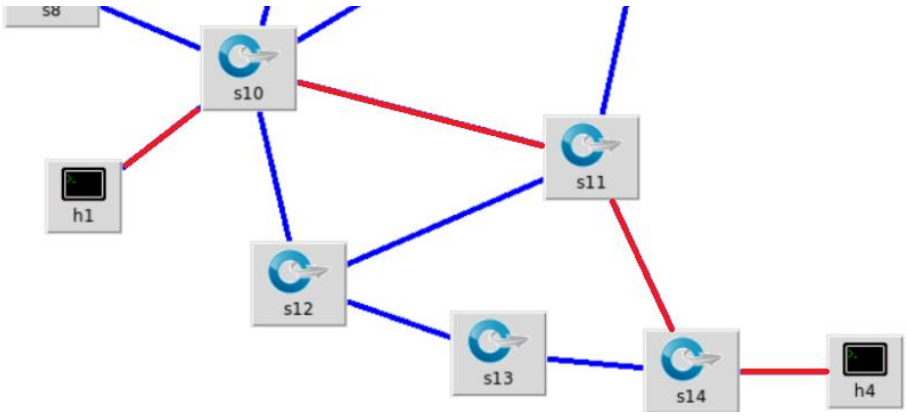


Figura 6.6: Porzione di rete interessata per l'esperimento 2.

Eseguendo i test si ottengono risultati molto simili all'esperimento 1, nonostante ci sia un nodo intermedio in più.

TEST 2	1 PACK	2 PACK	3 PACK	4 PACK	5 PACK	6 PACK	7 PACK	8 PACK	9 PACK	10 PACK
1 PING	1061	60,7	0,746	0,09	0,09	0,08	0,045	0,0135	0,064	0,054
2 PING	1052	52,3	0,321	0,046	0,079	0,038	0,079	0,041	0,035	0,041
3 PING	1059	58,8	0,367	0,031	0,072	0,06	0,085	0,083	0,053	0,105
4 PING	1049	58,5	0,56	0,086	0,058	0,073	0,079	0,051	0,057	0,093
5 PING	1089	81	0,411	0,081	0,086	0,109	0,089	0,086	0,083	0,083
6 PING	1044	43,9	0,321	0,037	0,059	0,092	0,061	0,083	0,041	0,094
7 PING	1060	53,1	0,41	0,091	0,096	0,081	0,027	0,027	0,076	0,055
8 PING	1079	80,8	0,513	0,066	0,063	0,063	0,048	0,077	0,069	0,133
9 PING	1055	55,3	0,547	0,033	0,046	0,089	0,09	0,056	0,028	0,094
10 PING	1059	50,6	0,473	0,043	0,077	0,023	0,091	0,075	0,069	0,036
MEDIA	1060,7	59,5	0,4669	0,0604	0,0726	0,0708	0,0694	0,05925	0,0575	0,0788
VARIANZA	166,61	135,188	0,0154059	0,0005596	0,0002288	0,0005972	0,0004624	0,0005952	0,0002969	0,0008748

Figura 6.7: Risultati dei test per l'esperimento 2, tempi espressi in millisecondi

Anche in questo caso i tempi di risposta dei primi pacchetti sono più alti, per lo stesso motivo dell'esperimento 1.

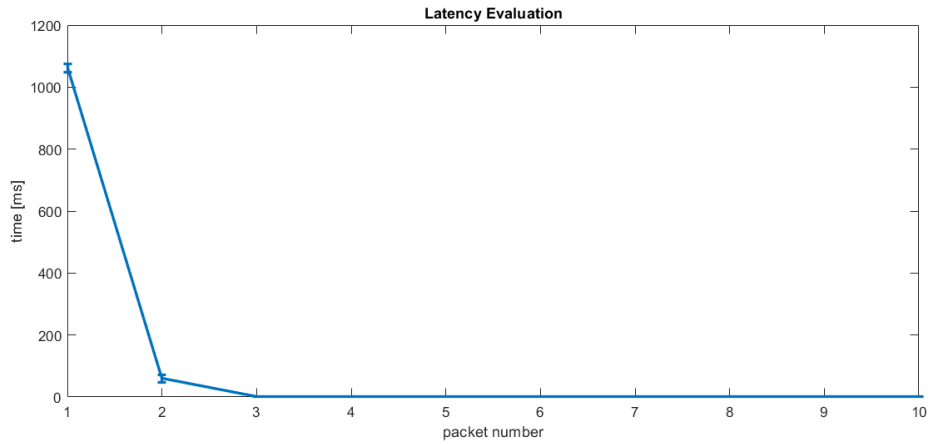


Figura 6.8: Grafico dell'esperimento 2

6.4 Esperimento 3

Nel terzo ed ultimo esperimento, si è scelto un percorso con più hop di quelli precedenti: esso è definito per il percorso da h3 ad h4 ed è composto da 4 hop ovvero s4, s9, s11, s14.

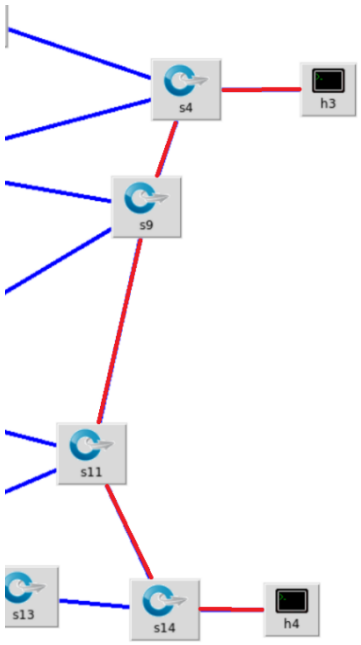


Figura 6.9: Porzione di rete interessata per l'esperimento 3.

TEST 3	1 PACK	2 PACK	3 PACK	4 PACK	5 PACK	6 PACK	7 PACK	8 PACK	9 PACK	10 PACK
1 PING	1064	58,1	0,945	0,098	0,048	0,052	0,044	0,097	0,036	0,04
2 PING	1046	54,1	0,894	0,038	0,094	0,075	0,026	0,061	0,105	0,049
3 PING	1071	76,2	0,174	0,05	0,045	0,032	0,07	0,044	0,07	0,097
4 PING	1071	70,6	0,824	0,095	0,068	0,065	0,054	0,055	0,154	0,038
5 PING	1068	67,7	0,188	0,134	0,033	0,042	0,099	0,092	0,074	0,041
6 PING	1027	41,4	0,149	0,035	0,074	0,084	0,085	0,068	0,064	0,056
7 PING	1060	60,6	0,855	0,064	0,095	0,086	0,063	0,09	0,074	0,081
8 PING	1057	49,3	0,48	0,063	0,069	0,131	0,062	0,069	0,08	0,068
9 PING	1062	62,2	0,459	0,062	0,039	0,072	0,058	0,102	0,042	0,067
10 PING	1091	82,6	1,32	0,07	0,025	0,063	0,065	0,86	0,075	0,087
MEDIA	1061,7	62,28	0,6288	0,0709	0,059	0,0702	0,0626	0,1538	0,0774	0,0624
VARIANZA	255,21	139,6936	0,141967	0,0008295	0,0005476	0,0006828	0,0003668	0,05575	0,0009866	0,0003976

Figura 6.10: Risultati dei test per l'esperimento 3, tempi espressi in millisecondi

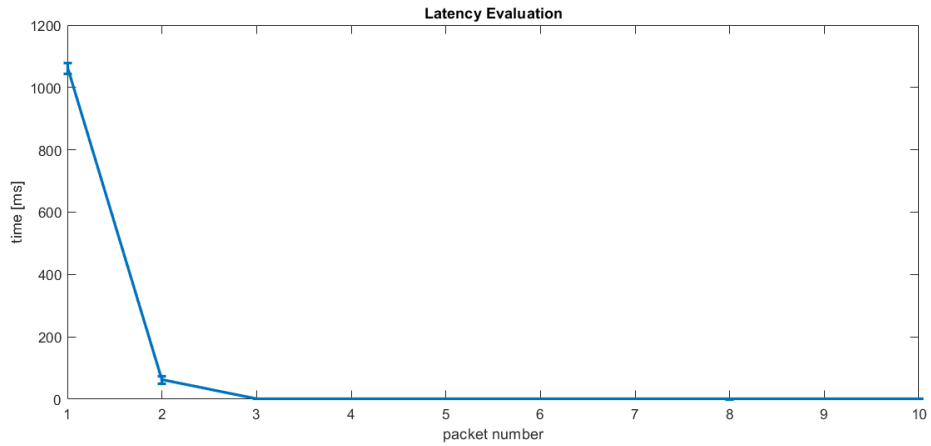


Figura 6.11: Grafico dell'esperimento 3

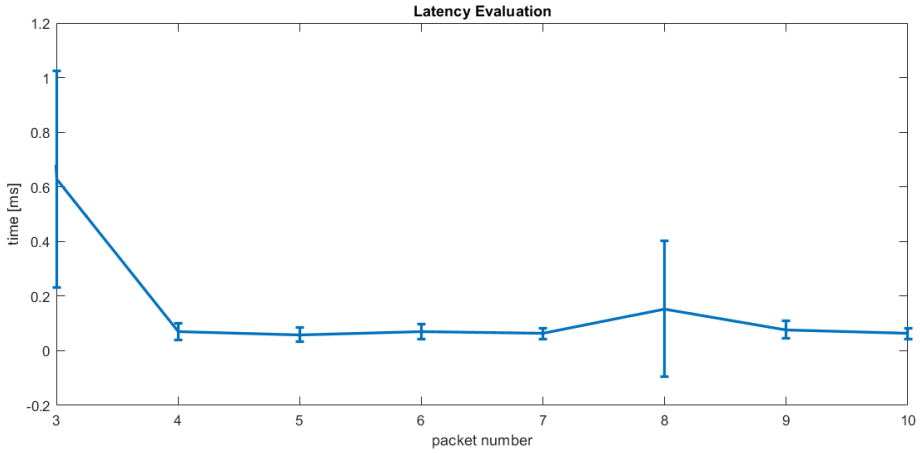


Figura 6.12: Grafico dell'esperimento 3 nel dettaglio

Si noti come anche in questo esperimento, i risultati siano paragonabili ai risultati ottenuti in quelli precedenti.

Capitolo 7
Conclusioni e sviluppi futuri

In questi capitoli sono state affrontate le tematiche SDN tramite la piattaforma di emulazione Mininet ed è stato implementato da 0 un componente che permettesse il traffico di dati in una rete complessa a piacere, in maniera completamente automatica, con un controller "out-of-band", ovvero fuori dal piano dati.
Analizzando i risultati è possibile osservare che tutti gli esperimenti hanno avuto il medesimo esito, cioè tempi di convergenza molto simili, dunque si è arrivati alla conclusione che essi sono indipendenti dal numero di hop, e quindi dal numero di nodi intermedi.
Ciò avviene ogni volta che si ha una nuova destinazione, in quanto deve essere gestito lo scambio di pacchetti ARP, deve essere usato Dijkstra per il calcolo dello shortest path, ed infine devono essere installate le regole sugli switch.

Ovviamente in tutti i test i primi pacchetti avevano una latenza più alta, causati dal controller: difatti anche se esso era "out-of-band", ovvero era direttamente connesso ad ogni singolo switch, esso ha dei tempi di processamento più lunghi di quelli degli switch.
Dunque, i rallentamenti causati direttamente dal controller per via della gestione dei pacchetti ARP, dal calcolo dello shortest path e dallo scambio di pacchetti con il data path portano a risultati poco appetibili, ma una delle proprietà di SDN è proprio quella di investire delle risorse inizialmente, per poi avere dei risultati molto vantaggiosi.
Per esempio, il calcolo dello shortest path richiede tempo al controller, ma può essere visto come un investimento, in quanto grazie ad esso, i pacchetti eseguiranno sempre il percorso più breve, portando così a tempi molto ridotti
Difatti i pacchetti successivi ai primi hanno dei tempi di risposta particolarmente brevi, grazie al fatto che sono state installate le regole all'interno delle flow table degli switch, in modo che non fosse più necessario comunicare con il controller per quel terminato flusso, per cui, nel caso di una grande mole di traffico, le tempistiche dei

primi pacchetti potrebbero essere trascurate, avendo così una rete programmabile e veloce.

Si noti come i tempi di risposta dei primi pacchetti aumenterebbero ancora di più nel caso in cui il controller fosse “in-band”, ovvero fosse all’interno del piano dati, come un nodo intermedio: i tempi aumenterebbero in quanto si andrebbe ad aggiungere una maggiore quantità di tempo per la comunicazione, ovvero, anche i messaggi mandati dal controller seguirebbero dei percorsi, come dei pacchetti normali; ciò non avviene con un controller “out-of-band” in quanto ogni switch ha un canale dedicato alla comunicazione con il controller, quindi è come se avvenisse in tempo particolarmente piccolo se paragonato al primo caso di controller.
È importante però osservare che nel caso di controller “in-band” vengono influenzati solo i primi pacchetti perché quelli successivi, trovando le regole nelle flow table degli switch, non vengono influenzati.

Per aumentare la resistenza ai guasti potrebbe essere implementato anche l’algoritmo di Yen, il quale si occupa del problema del KPath, ossia la ricerca in un grafo di shortest path multipli tra due nodi, e ciò permetterebbe di trovare diversi percorsi di backup, nel caso di problemi con un link della rete.
Inoltre, si potrebbero definire degli algoritmi che permettano di abbattere i tempi di risposta dei primi pacchetti preinstallando delle regole negli switch.

Tutto il materiale è disponibile al link:
https://github.com/alexandrurotariu/ComponentPOX

Bibliografia

[1] SDN architecture Issue 1 June, 2014 ONF TR-502

[2] An SDN Perspective to Mitigate the Energy Consumption of Core Networks – GEANT2

[3] A Network in a Laptop: Rapid Prototyping for Softwere-Defined Networks

[4] Introduction to Mininet - https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

[5]Download/Get Started With Mininet - http://mininet.org/download/

[6]Macchina virtuale - https://it.wikipedia.org/wiki/Macchina_virtuale

[7]Secure Shell - https://it.wikipedia.org/wiki/Secure_Shell

[8] PuTTY, Wikipedia: https://it.wikipedia.org/wiki/PuTTY

[9] How to use MiniEdit, Mininet's graphical user interface - http://www.brianlinkletter.com/how-to-use-miniedit-mininets-graphical-user-interface