ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA – SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA E DELLE TELECOMUNICAZIONI

EMULAZIONE DISTRIBUITA DI RETI DI TELECOMUNICAZIONI SU PIATTAFORMA MININET

Tesi di laurea in

Laboratorio di Reti di Telecomunicazioni L-A

CANDIDATO:
Francesco Cristiano

RELATORE:
Prof. Walter Cerroni

CORRELATORE:
Ing. Chiara Contoli

Anno Accademico 2013/2014

Sessione III

PAROLE CHIAVE

SDN
Mininet
Openflow
Reti

INDICE

INTRODUZIONE

Le reti di telecomunicazioni sono diventate uno strumento essenziale per la comunicazione mondiale, anche grazie alle prestazioni migliorate durante gli anni. 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.

I nuovi servizi e le attuali applicazioni informatiche richiedono una Rete di Telecomunicazione sempre più dinamica e capace di adattarsi alle esigenze variabili degli utenti e dei gestori in tempi brevi. Tuttociò 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 (o forwarding) 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 e con un approccio tramite software. Questo è reso possibile grazie all'introduzione di nuovi protocolli uno dei quali, ed anche il primo, è il protocollo Openflow, sostenuto dalla Open Networking Foundation.

Utilizzando il simulatore Mininet, con il quale è possibile simulare reti anche complesse su apparecchi relativamente semplici, come un PC-Desktop o un Laptop, si vuole verificare la possibilità, che le SDN mettono a disposizione, di poter definire tramite software una o più reti distribuite, dimostrandone la flessibilità, la programmabilità e l'effettiva funzionalità mediante alcuni Test.

Dopo aver studiato le parti fondamentali del nuovo approccio SDN, le basi del protocollo Openflow e il funzionamento dell'ambiente di simulazione Mininet, si è proceduto quindi allo studio delle diverse Application Programming Interface (API)

presenti online, ovvero un insieme di procedure disponibili al programmatore, messe a disposizione per la creazione di una Rete virtuale e di un controller ad essa adeguato. Nel nostro caso, il controller, è stato creato utilizzato il Framework Pox, basato come Mininet su linguaggio Python.

Infine si è proceduto alla creazione ed alla verifica, eseguendo vari test ed anche graficandone alcuni risultati, del funzionamento di una topologia di Rete distribuita su due macchine virtuali.

CAPITOLO 1

SOFTWARE DEFINED NETWORKING

1.1 La necessità di una nuova Architettura

In questi ultimi anni l'evolversi dei servizi cloud[1] ma anche la diffusione sempre maggiore di apparecchi mobili per l'accesso alle reti di calcolatori, hanno evidenziato alcuni dei limiti dell' architettura di rete classica. Fino ad oggi, le reti sono state organizzate con l'uso di dispositivi di comunicazione quali switch, bridge, router e gateway. Sebbene questa struttura, fino a qualche anno fa, sia stata adeguata, recentemente ha mostrato le sue debolezze [1].

I principali fattori che hanno richiesto nuove architetture strutturali sono:

- Una differente *Tipologia di traffico*: nei data center, dove la maggior parte delle comunicazioni avvengono tra un client e un server in un nuovo modello orizzontale, nel quale le applicazioni accedono a diversi database e server per infine arrivare all'utilizzatore finale, sia utilizzando cloud pubblici che privati.
- Un sempre maggiore utilizzo di *dispositivi mobili* per accedere ai dati e alle reti aziendali in qualsiasi momento ed in qualsiasi luogo, che richiedono nuove funzionalità per la protezione dei dati stessi, per garantirne la compatibilità e migliorarne l'accesso.
- Una modalità *on demand* sempre più richiesta dalle aziende, per l'accesso alle applicazioni e ad altre risorse *Information Tecnology*[2] (IT), che richiede una maggiore sicurezza e flessibilità.

[1] si indica un insieme di tecnologie che permettono, tipicamente sotto forma di un servizio offerto da un provider al cliente, di memorizzare/archiviare e/o elaborare dati (tramite CPU o software) grazie all'utilizzo di risorse hardware/software distribuite e virtualizzate in Rete in un'architettura tipica client-server.

[2] Sono applicazioni di computer e device di telecomunicazioni, dedicate alla memorizzazione , al recupero alla trasmissione ed alla manipolazione di dati.

- L'utilizzo di grandi data Set[3] per elaborazioni massicce su server interoperanti che può richiedere un estensione delle capacità di rete in modo dinamico e talvolta imprevedibile.
- L'incremento dei servizi cloud e la virtualizzazione dei server
- L'offerta di servizi IT *as service* (come servizio*)*

Il modello architetturale attuale non è in grado di far fronte alle esigenze di dinamicità e di gestione dei requisiti delle richieste IT. Ulteriori complicazioni sono dovute all'evoluzione dei protocolli di rete dei device. Nei quali i setup necessitano di un arduo lavoro di riconfigurazione degli apparati switch e router, dei firewall, degli accessi ACL (Access Control List), della sicurezza e della qualità del servizio QoS, richiedendo molto tempo per la loro esecuzione ed impattando sulla topologia di rete e sull'aspetto *Vendor-Dependent.* La staticità delle reti attualmente in uso, rischia di contrapporsi all'esigenza IT di minimizzare il rischio d'interruzione dei servizi.

Tutto questo crea uno scenario fortemente dipendente dai fornitori di apparati di telecomunicazioni e di soluzioni di rete (*Vendor), dove,* gli operatori ed anche le imprese necessitano di fornire nuove funzionalità e nuovi servizi per rispondere alle diverse esigenze aziendali, dell'utente finale e quindi anche del mercato. Questa capacità di realizzazione è determinata dai soli *vendor,* i quali hanno, però, un ciclo di vita del prodotto molto lungo e non sono in grado, in tempi ragionevoli, di fornire i prodotti richiesti per assicurare l'interoperabilità e la personalizzazione necessarie a questi ambienti specifici. Quando un nuovo protocollo viene standardizzato, prima di poter essere introdotto nelle reti reali, deve essere implementato nei dispositivi e come risultato si ottiene un rallentamento dello sviluppo della rete. Infatti, i cicli di sviluppo dei costruttori sono solitamente molto più lenti dei cicli di sviluppo di un normale software e possono perdurare anche alcuni anni.

Un'altra problematica è la natura dinamica e l'ampia richiesta di banda delle applicazioni odierne, quali ad esempio servizi cloud e la sempre maggiore diffusione di apparati Mobili (Cellulari, Tablet ecc), che richiedono una migliore ottimizzazione delle risorse e della gestione della Rete. Inoltre, nei nuovi servizi di Information Tecnology le trasformazioni avvengono con frequenza ravvicinata, la longevità delle connessioni

[3] Sono grandi collezioni di dati.

diventa fattore critico e occorrono strumenti in grado di definire le regole e di procedere il più possibile automaticamente agli aggiornamenti.
Tutto ciò ha portato alla creazione di nuove soluzioni architetturali come il *Software Defined Networking (SDN),* che ha come obiettivo una rapida ed automatica configurazione del traffico e dell'instradamento, attraverso un approccio di *astrazione* delle risorse di rete. La possibilità di definire dinamicamente via software il comportamento di una rete fisica offre agli amministratori la flessibilità di adattare la rete esistente a nuove richieste, senza onerose operazioni di riconfigurazione. Inoltre, regole ben definite e condivise generano in questo modo una visione unificata non *vendor-dependent* di questa soluzione.

1.2 Struttura delle Software Defined Networking.

Software Defined Networking è una nuova architettura di rete standardizzata dall'Open Networking Foundation (ONF), rappresenta un approccio alla costruzione di apparecchiature di rete e software che separa e astrae gli elementi di tali sistemi.
L'architettura può essere rapresentata logicamente da 3 strati (layer) [1]:

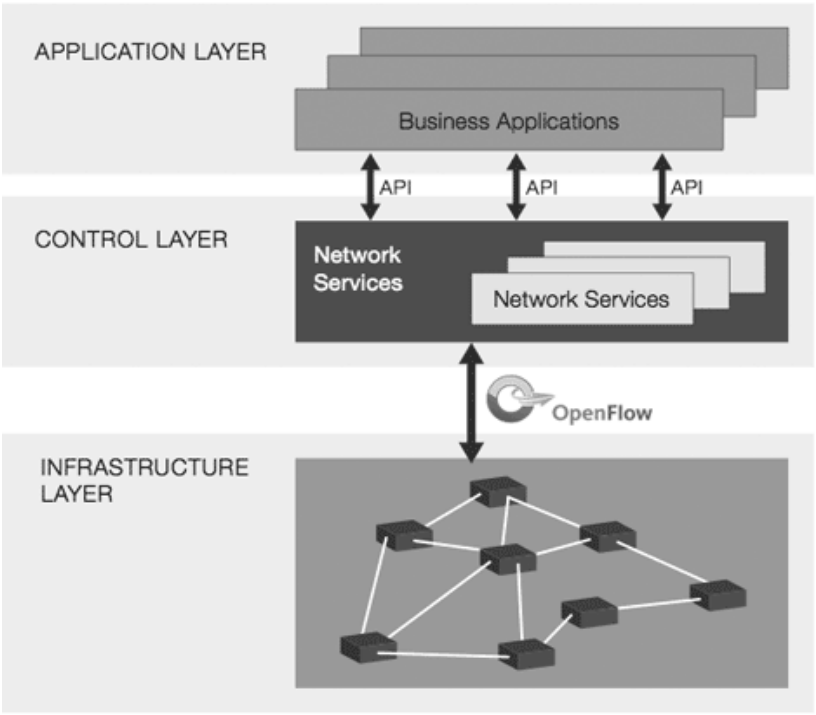


Figura1: architetturaSDN [https://www.opennetworking.org]

-Application Layer

-Control Layer

-Physical Layer

Il **Physical Layer (o infrastucture Layer)** è il livello più basso che comprende quindi i dispositivi fisici e rappresenta l'infrastruttura di rete; si utilizza il termine "switch" per evidenziare come il modello SDN cambia la logica di lavoro degli switch ethernet; è possibile includere nell'infrastruttura di rete anche Virtual switch.

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 *layer di trasporto*, al contrario dell'architettura attuale. In quest' ultima, il controllo, infatti, 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*.

L'**Application Layer** include le business application[4] che rappresentano le cosiddette funzioni usabili dell'architettura SDN, erogano i servizi di networking agli utilizzatori finali; di questo livello ad esempio fan parte la maggior parte delle feature più innovative del Networking. Nell'Application Layer, ciascun'applicazione potrebbe sviluppare una *view* dei flussi di una rete di device (*flow rule*) e poi inviarla al controller per la programmazione diretta degli switch.

[4] è un qualsiasi software o un insieme di programmi per computer che vengono utilizzati dagli utenti aziendali per eseguire varie funzioni aziendali

Molti vendor forniscono entrambi i layer (Application e Controller) in un singolo prodotto.

Caratteristica principale di quest'architettura è il disaccoppiamento tra il controllo della rete e la specificità dei dispositivi fisici, rendendo il primo direttamente programmabile. L'instradamento dei pacchetti viene direttamente deciso dal controller tramite la programmazione delle *flow table* degli switch, nelle quali vengono specificate anche le esatte interfacce cui inoltrare i pacchetti ricevuti. L'astrazione viene introdotta dal fatto che la struttura della *flow table* non viene modificata, così come le operazioni eseguibili dal controller su di essa, il tutto indipendentemente dal vendor. Quindi la selezione della destinazione di un pacchetto è distinta dall'invio del pacchetto sull'opportuna interfaccia di rete e non è più eseguita interamente dall'hardware del singolo dispositivo. L'instradamento e il suo controllo sono assegnati ad un controller, che logicamente è un'unità e può trovarsi su un qualsiasi host[5] della rete, può essere realizzato con un approccio distribuito ed anche essere programmato per gestire flussi dati diversi in modo diverso, in maniera del tutto trasparente alle applicazioni che utilizzano la rete. Gli amministratori di rete in questo modo possono aggiornare e aggiungere funzionalità, modificando il software di controllo e quindi garantendo una maggiore flessibilità ed una rapida evoluzione della rete stessa.

Definendo un'Application Programming Interface (API) fra controller SDN e le applicazioni che utilizzano la rete è anche possibile, da parte di queste ultime, utilizzare un'astrazione della rete fornita dal SDN, potendo tralasciare i dettagli della topologia e concentrandosi sui servizi. Questo rende le reti *application-customized* e le applicazioni *network-capability-aware*.

Inoltre, l'introduzione di un controller programmabile e di un'API a disposizione delle applicazioni di rete, permette di implementare i nuovi servizi e le nuove funzionalità senza bisogno di dover intervenire manualmente su ogni singolo dispositivo di rete. Il controller può essere programmato per adattarsi alla dinamicità delle richieste che giungono alla rete, semplificando e centralizzando la gestione della stessa, consentendo nel contempo una rapida evoluzione. Tutto ciò non più rallentato dalla necessità di attendere il rilascio di nuove versioni del firmware da parte dei costruttori dei dispositivi

[5] indica ogni terminale collegato, attraverso link di comunicazione, ad una rete informatica

e garantendo così uno sviluppo a velocità del software piuttosto che uno sviluppo a velocità degli standard, com'è stato fino ad oggi.

1.3 Vantaggi della nuova architettura

La scelta di spostare la rete al di fuori del layer fisico, e quindi di centralizzare gran parte delle funzionalità nel SDN controller, porta ai seguenti vantaggi [2]:

- consente l'*astrazione* dell'infrastruttura sottostante;

- le *business application* ed i servizi di rete percepiscono il network come una *entità logica*;

- si realizza un'indipendenza dai *vendor* e dalle specificità dei dispositivi;

- viene semplificata sia la progettazione sia la gestione operativa dell'intera rete;

- permette di rendere più *semplice* sia le tecnologie di costruzione di router e *switch*, sia la loro configurazione, poiché il dispositivo svolge solo funzioni di *forwarding*[6] del flusso e si richiede l'utilizzo del solo protocollo di comunicazione con il controller SDN;

- il piano di controllo viene arricchito con *nuove funzionalità*;

- la logica della rete è *definibile* direttamente dagli utilizzatori; è possibile instradare in tempo reale il traffico in base allo stato dell'intera infrastruttura IT, programmando adeguatamente il controller;

- *velocizza* i processi d'innovazione e di automatizzazione consentendo la creazione di nuove funzionalità e servizi di rete, in modo più semplice e rapido senza dover configurare i singoli dispositivi o modificarne il firmware;

- incrementa l'*affidabilità* e la *sicurezza* della rete avendo centralizzato ed automatizzato la gestione dei network device;

- consente di *controllare* la rete ed applicare *policy* con diversi livelli di granularità, sessione, utente, device ed applicazione.

[6] permette il trasferimento dei dati (*forwarding*) da un computer ad un altro tramite una specifica porta di comunicazione

- l'architettura SDN fornisce un *set di API* che consentono di implementare servizi di rete comuni, come routing, multicast, security, access control, bandwidth management, traffic engineering, quality of service, processor e storage optimization, energy usage, policy management.

Questa serie di vantaggi ha portato all' interessamento nei confronti di questo nuovo tipo di approccio da parte di molte aziende del settore. Infatti IDC, primo gruppo mondiale in ricerche di mercato, ha recentemente previsto che il mercato globale SDN è destinato a crescere da 960 milioni di dollari nel 2014 a più di 8 bilioni di dollari nel 2018, quindi non è solo una moda passeggera. I primi ad adoperare questa tecnologia hanno visto un incremento nella loro agilità ed efficienza di rete, mettendo letteralmente a sedere i loro concorrenti [3].

Non solo grazie al SDN è possibile rendere la rete più efficiente, ma è possibile abbattere i costi operativi, siccome le imprese non necessitano più di hardware vendor specifico, né tantomeno di fare investimenti tecnologici aggiuntivi per rendere la propria rete adeguata. I primi a investire su questo nuovo tipo di approccio sono stati i Datacenter e i fornitori di servizi i quali hanno riscontrato migliore servizio al cliente finale e tempi di risposta più rapidi.

Tuttavia la genericità dell'architettura SDN crea un po' di ambiguità, i confini tra le *business application* ed il *controller* non sono ben definiti, in termini di programmazione di alto livello della rete. In alcuni casi, le business application potrebbero sviluppare una *view* dei flussi di una rete di device (*flow rule*) da inviare, poi, al controller per la programmazione diretta degli switch, senza così tralasciare la conoscenza (anche se di alto livello) della rete [1]. Inoltre, diverse soluzioni implementative del SDN forniscono entrambi i layer, application e controller, in un singolo prodotto; non solo, ma gran parte della feature più innovative del networking sono incluse proprio nel livello *application*.

Per superare tale ambiguità nei dispositivi di rete dovrà essere implementato un supporto alla nuova architettura e ogni dispositivo di rete dovrà presentare la stessa interfaccia verso il controller. Per fare questo è stato sviluppato e proposto dalla Open Networking Foundation, consorzio di aziende e istituzioni, lo standard *OpenFlow* in quanto risulta necessario definire alcune regole.

CAPITOLO 2

LO STANDARD OPENFLOW

2.1 Definizione

Figura 2.1 Simbolo openflow [https://www.opennetworking.org]

OpenFlow è il primo protocollo standardizzato per l'architettura SDN, considerato dalla maggior parte dei produttori hardware e software come l'alternativa al **Multi Protocol Label Switching (MPLS)**[7], superando la classica divisione a strati (layer) dei protocolli.

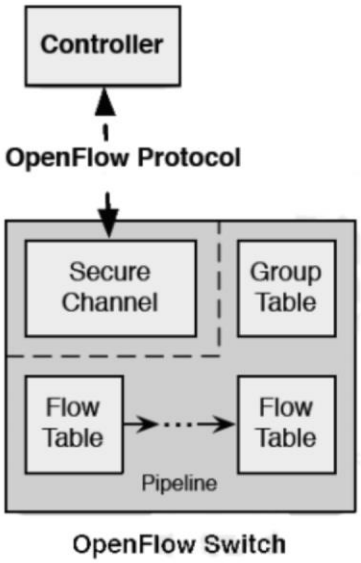


Figura 2.2 : OpenFlow

[7] è una tecnologia per reti IP che permette di instradare flussi di traffico multiprotocollo tra nodo di origine e nodo di destinazione tramite l'utilizzo di identificativi (*label*) tra coppie di router adiacenti e semplici operazioni sulle etichette stesse.

La specifica OpenFlow descrive in quale modo deve essere implementato uno switch affinchè supporti l'architettura SDN ed un protocollo per la gestione del dispositivo da parte di un controller. L'implementazione è infatti pensata per sostituirsi o affiancarsi al normale metodo d'inoltro dei pacchetti da parte del dispositivo, fino ad oggi basato su tabelle associative hardware.

OpenFlow viene definita dalla Open Networking Foundation(ONF), organizzazione dedita alla promozione e all'adozione di SDN, come la prima interfaccia di comunicazione standard tra il *control plane* ed il *data plane* di un architettura SDN.

Infatti proprio l'assenza di un'interfaccia dedicata alla gestione del *forwarding plane* ha portato alla caratterizzazione degli odierni dispositivi di rete come monolitici e chiusi [4]. Un protocollo è necessario affinchè il controllo della rete non sia più concentrato su switch proprietari ma sia decentrato in un software di controllo open source e a gestione locale. Lo standard consente quindi di accedere direttamente al piano di inoltro (forwarding plane) dei dispositivi di rete quali router e switch, sia fisici che virtuali, consentendone anche la modifica.

In definitiva OpenFlow permette ad un software, che può essere istanziato su più router (dei quali almeno due hanno il ruolo di osservatori), di decidere il percorso di rete dei vari pacchetti. La separazione del control plane dal forwarding plane permette una gestione dell'inoltro di pacchetti più sofisticata rispetto a quella possibile utilizzando solamente le access control list ed i classici protocolli di routing.

I suoi inventori considerano OpenFlow come un abilitatore del modello SDN.

2.2 Lo scopo del protocollo

L'idea di base per OpenFlow è quella di rendere programmabili le tabelle di classificazione ed instradamento dei pacchetti presenti negli apparati di networking (router o switch). In questo modo il contenuto, ovvero le cosiddette entry, può essere configurato direttamente dalle applicazioni, tramite un piano di controllo esterno al dispositivo e mediante un'opportuna interfaccia [5].

OpenFlow quindi rimuove l'intero piano di controllo dal dispositivo di rete, rendendo il device dedito al solo trasporto dati. Tutte le funzioni di controllo (*discovery, path computation, path setup*, ecc.) verranno implementate in un'entità esterna alla rete.
Il protocollo si applica principalmente alle reti TLC/WAN, ma è anche finalizzato a contesti di data center ed ambienti virtuali.
Tramite Openflow viene definita la comunicazione tra il layer di controllo e quello di trasporto mediante un set di primitive che consentono l'accesso diretto alle cosiddette *flow tables* dei dispositivi di rete. Viene utilizzato il concetto di flusso per determinare e gestire il traffico di rete, dove per flusso viene inteso una sequenza di pacchetti identificabili da uno o più etichette comuni (quali ad esempio: l'indirizzo IP, il MAC address, il numero di porta, ecc.). Ogni record della tabella dei flussi contiene:

- una serie di regole che permettono di identificare i pacchetti, e quindi i flussi. Le regole possono essere programmate staticamente o dinamicamente dal controller;

- un'azione, anch'essa programmabile dal controller. Definisce come il pacchetto deve essere instradato lungo la rete, in conformità a parametri legati a specifici patter, applicazioni e risorse. In particolare, l'azione può essere del tipo *send-out-port*, *modify-field* o *drop*;

- delle statistiche relative al conteggio dei pacchetti corrispondenti ad ogni regola.

Quando uno switch OpenFlow riceve un nuovo pacchetto per il quale non vi sono regole attive nella propria tabella, lo invia al controller che decide in che modo gestirlo; il controller può infatti decidere di scartarlo oppure di aggiungere un nuovo record alla tabella dello switch, configurando le azioni da applicare a tutti i pacchetti analoghi. Inoltre, in base alle configurazioni impostate dal controller, una regola d'indirizzamento può scadere dopo un certo intervallo o persistere fino allo spegnimento del device. Con tali meccanismi, OpenFlow consente di fornire un controllo estremamente granulare, in modo tale da rispondere, in tempo reale, ai cambiamenti che si verificano nell' intera rete.
Scopo del protocollo OpenFlow è, quindi, quello di presentare all'esterno un modello di nodo generale e unificato, rendendo gli strati più alti dell'architettura di rete SDN

indipendenti dall'implementazione del particolare vendor delle tecnologie impiegate nel piano di forwarding [5].

2.3 Lo Switch Openflow

L'idea di base è semplice: si sfrutta il fatto che la maggior parte di switch e router moderni contengono tabelle di flusso (*flow-table* appunto) e che, pur essendo diverse per ogni costruttore di device, sono state identificate un interessante set comune di funzioni che sono utilizzate in molti di questi apparecchi. OpenFlow sfrutta questo set comune di funzioni, fornendo un protocollo aperto per programmare le flow-table di diversi switch e router.
L'unità di elaborazione di uno switch Openflow è associata alla tabella di flusso, e all'azione indicata a ciascun flusso in ingresso. Le tre parti principali di uno swich Openflow sono:

1- Una o più flow table, con le relative azioni associate ai flussi entranti.
2- Un canale sicuro (Secure Channel) il quale connette lo switch al controller remoto, permettendo ai comandi e ai pacchetti di essere inviati utilizzando il protocollo Openflow.
3- Il protocollo Openflow stesso che consente una comunicazione standard tra controller e switch.

Il protocollo quindi evita di dover programmare i singoli switch.
Risulta utile categorizzare gli Switch in switch Openflow dedicati, i quali quindi non supportano i layer 2 e 3, e switch Ethernet di uso generale abilitati all'Openflow, per i quali il protocollo e le interfacce sono stati aggiunti come nuova funzionalità [6].

2.3.1 Switch Openflow dedicati

Uno switch Openflow dedicato è un elemento che inoltra i pacchetti tra porte, in modo definito da un processo di controllo remoto.

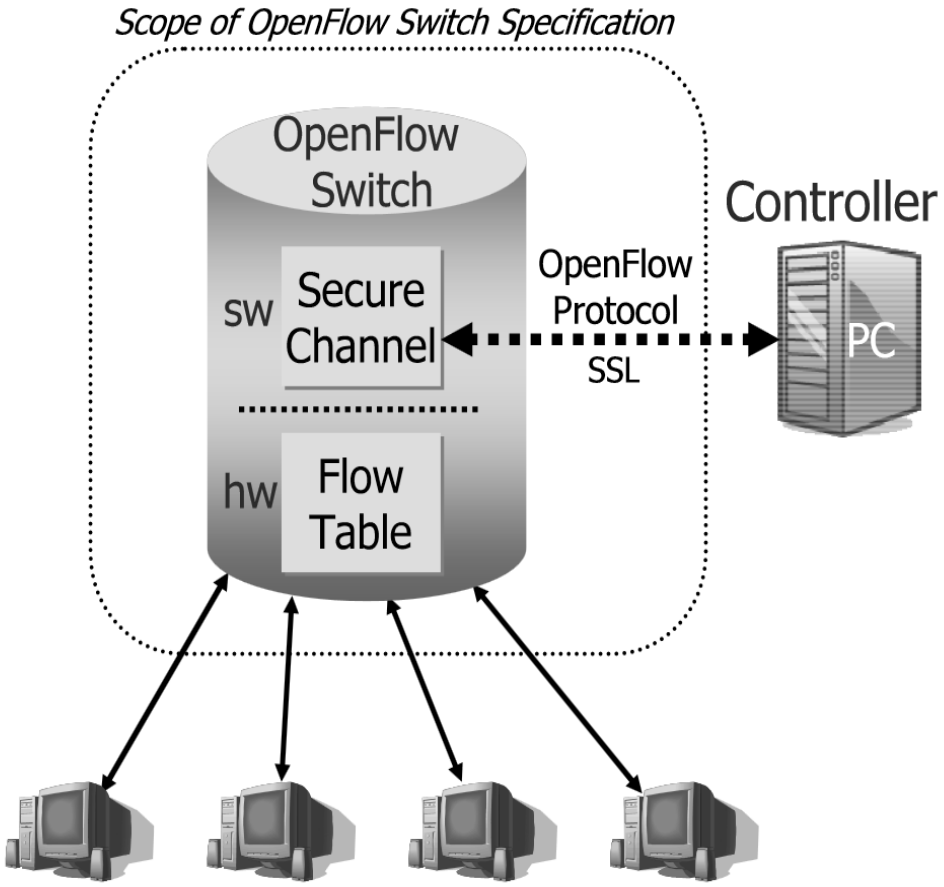


Figura 2.3 Struttura OpenFlow [6]

In questo contesto i flussi sono ampiamente definiti, e sono limitati solamente dalla particolare implementazione della flow-table. Un flusso potrebbe essere ad esempio una connessione TCP, oppure tutti i pacchetti provenienti da un determinato indirizzo MAC o IP, o anche i pacchetti entranti da una determinata porta dello switch. I flussi possono, quindi, essere scelti in determinati modi, che saranno poi decisi dal programmatore stesso in base alle esigenze, e ciascuno di questi flussi ha una semplice azione associata. Le tre azioni base che uno switch Openflow deve supportare sono:

1- Inoltrare i pacchetti del flusso di una o più determinate porte, permettenndo agli stessi di essere instradati attraverso la rete.
2- Inoltrare i pacchetti del flusso a un Controller. Il pacchetto viene consegnato al Secure channell, dove viene incapsulato, per poi essere inviato 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 flow table del relativo switch.

3- Eliminari i pacchetti provenienti dal flusso. Può essere usato per la sicurezza, per evitare attacchi Denial of Service[8], o per ridurre il traffico spurio dovuto ai broadcast[9] da parte degli end-host.

Una voce nella tabella di flusso ha tre campi: un pacchetto intestazione che definisce il flusso, l'azione, che definisce come devono essere trattati i pacchetti, e le statistiche, che tengono conto dei pacchetti e dei byte di ciascun flusso e quando per l'ultima volta un pacchetto è stato abbinato a quel flusso [6].
I requisiti dettagliati di uno switch open flow sono poi definiti nelle specifiche Openflow consultabili sul sito http://OpenFlowSwitch.org.

2.3.2 Switch abilitati Openflow

Alcuni switch, router e access point commerciali saranno rafforzati con le funzionalità OpenFlow, aggiungendo le flow tables, il Secure channel e quindi il protocollo OpenFlow. In genere, la tabella dei flussi riutilizzerà l'hardware esistente, ad esempio un TCAM[10], mentre il Canale sicuro e il protocollo saranno modificati per funzionare con il sistema operativo dello switch.

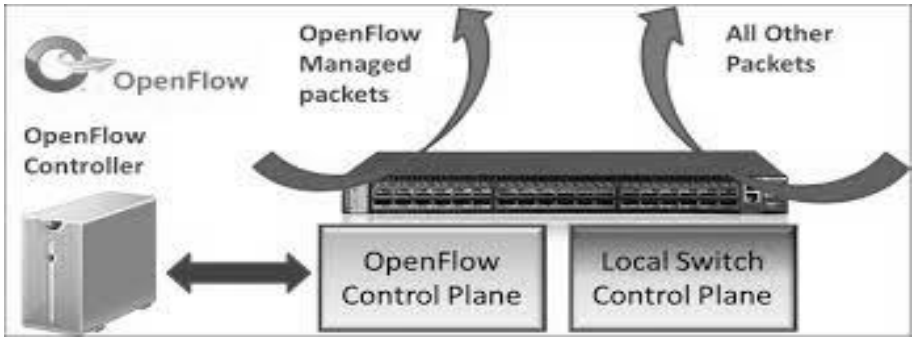


Figura 2.4 Switch abilitato Openflow
[http://www.mellanox.com/blog/tag/openflow/#sthash.iuARFQPj.dpbs]

[8] Indica malfunzionamento dovuto ad un attacco informatico in cui si esauriscono deliberatamente le risorse di un sistema informatico che fornisce un servizio ai client fino a renderlo non più in grado di erogare il servizio ai client richiedenti.
[9] Il termine broadcast indica una modalità di instradamento per la quale un pacchetto dati inviato ad un indirizzo particolare (detto appunto di *broadcast*) verrà consegnato a tutti i computer collegati alla rete
[10] è un tipo specializzato di memoria ad alta velocità che cerca l' intero contenuto in un unico ciclo di clock.

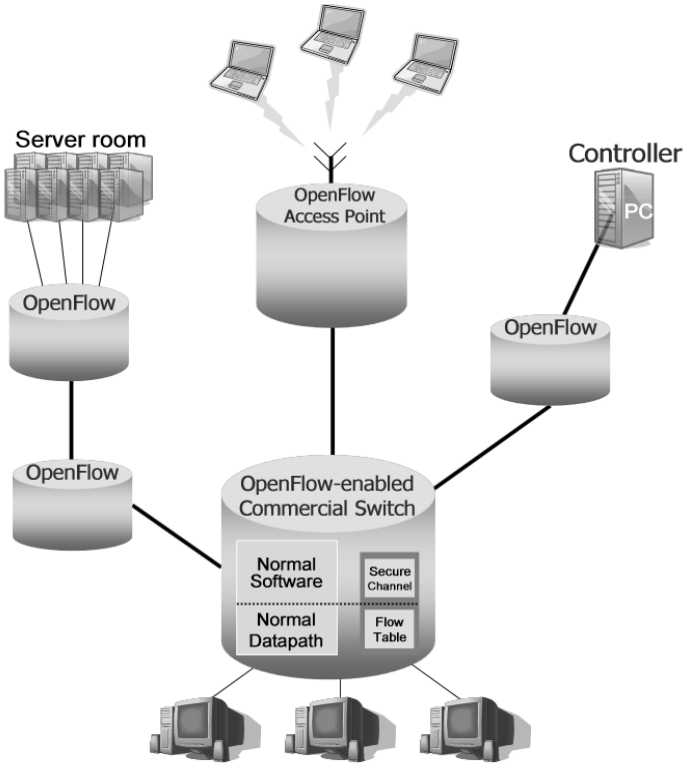


Figura 2.5 Switch OpenFlow abilitato[6]

Nell'esempio mostrato in figura 2.4, tutto il flusso e le flow table sono gestiti dallo stesso controller, ma è possibile migliorare le prestazioni e soprattutto la robustezza della rete permettendo ad uno switch di essere controllato da 2 o più controller. Lo scopo di abilitare gli switch, già esistenti, all'Openflow è quello di sperimentare in modo parallelo in reti già attive, quindi isolando il traffico sperimentale (elaborato dalla flow table) dal traffico che deve essere elaborato dai normali strati 2 e 3 degli switch. Ci sono due modi per realizzare questa separazione. Uno è quello di aggiungere una quarta azione (rispetto alle 3 descritte nello swich dedicato):

4- inoltrare i pacchetti del flusso attraverso la normale struttura di elaborazione dello switch.

L'altro è quello di definire gruppi separati di VLAN per le vie parallele. Entrambi gli approcci consentono al traffico normale, che non fa parte della sperimentazione, di essere trattato al solito modo dallo switch. Tutti gli switch abilitati OpenFlow sono tenuti a sostenere o l'uno o l'altro approccio; mentre altri supporteranno entrambi [6].

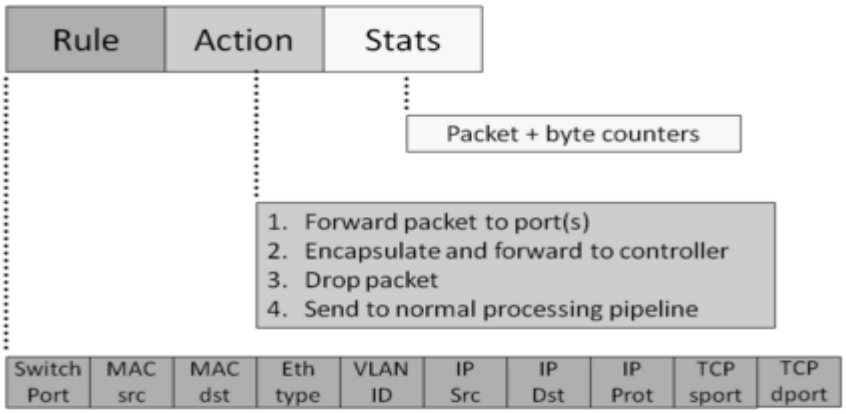


Figura 2.6 Tabella di flusso [http://yuba.stanford.edu/cs244wiki/index.php/Overview]

2.4 Il 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, ai dispositivi della rete indicando loro il miglior percorso da scegliere per il traffico generato dalle applicazioni. Essendo un software il controller può essere dislocato ovunque all' interno della rete, ma può essere anche collocato esternamente ad essa, 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

-Beacon basato su linguaggio java

-Floodlight in java

Molti altri ne sono disponibili e può essere scelto il più comodo al programmatore per l'implementazione.

In definitiva il controller può essere visto come la mente della rete, e risponde alle richieste degli switch definendo l'intera struttura e risolvendo le diverse problematiche.

2.5 Utilizzo di Openflow

Come semplice esempio si consideri il caso in cui si voglia testare un nuovo protocollo d'instradamento in una topologia di rete composta da end-host e da switch OpenFlow. Eseguendo l'esempio all'interno di un PC desktop che funge da controller, in modo tale da evitare di interferire nella rete. Ogni volta che un nuovo flusso applicativo viene generato, il protocollo, che si vuole testare, sceglie un percorso attraverso una serie di switch OpenFlow e aggiunge una flow entry in ognuno, lungo il percorso scelto. Per il traffico in entrata nella rete OpenFlow si definisce un flusso in modo che tutto il traffico in ingresso allo switch passi attraverso la porta connessa al PC e si aggiunge una entry con l'azione che incapsula e spedisce tutti i pacchetti al controller. I pacchetti quindi giungono al controller che grazie al protocollo sceglie un percorso e aggiunge una nuova flow entry per ogni switch lungo tutta la traiettoria. I pacchetti che giungeranno successivamente allo switch, verranno invece processati velocemente attraverso le flow table (le regole per gestirli saranno già presenti). Il funzionamento quindi è semplice avviene una comunicazione sicura (tramite secur channel) tra switch e controller i quali si scambiano una serie di messaggi grazie ai quali il controller può decidere e poi istruire lo switch su come agire. Lo switch non appena riceve un pacchetto dalla Rete invia un messaggio del tipo PaketIn al controller il quale una volta elaborato opportunamente decide come instradarlo ed istruisce lo switch inviandogli un messaggio di tipo PaketOut o FlowMod, nel quale è inserita il tipo di regola da applicare. Quindi per ogni pacchetto di un nuovo flusso entrante nello switch avviene uno scambio di messaggi tramite il protocollo.

Ci sono alcune domande legittime da porsi circa le prestazioni, l'affidabilità e la scalabilità di un controller, che aggiunge e rimuove in modo dinamico flussi:

- Può un controller centralizzato essere abbastanza veloce per elaborare nuovi flussi e programmare i flussi negli switch?
- Cosa succede quando un controller si guasta?

Per rispondere a queste domande, è stata testata con Ethane [7], architettura di switch della stanford University, una semplice topologia che ha utilizzato dei semplici switch ed un controller centrale. I risultati preliminari mostrano che un controller come questo e in grado di elaborare oltre10.000 nuovi flussi al secondo, sufficienti per un grande

campus universitario. Naturalmente, il tasso con il quale i nuovi flussi possono essere elaborati dipende dalla complessità della computazione richiesta dagli esperimenti.
Se ci spostassimo in una rete di più larga scala, nella quale il protocollo che si vuole testare viene eseguito in una rete utilizzata da molte altre persone, sarebbe desiderabile che la rete avesse due caratteristiche aggiuntive:

1. Pacchetti che appartengono a utenti dovrebbero essere instradati tramite un protocollo standard d'instradamento eseguito nello switch o nel router;
2. Colui che gestisce la rete dovrebbe essere in grado di aggiungere voci di flusso per il suo traffico, in modo tale da consentirgli di controllare i flussi e l'integrità della rete.

La prima caratteristica si ottiene dall'abilitazione degli switch OpenFlow; è possibile installare al loro interno delle regole che permettano la gestione di determinati pacchetti appartenenti ad un singolo utente. La seconda invece dipende dal controller; esso dovrebbe essere visto come una piattaforma che permette ai ricercatori di implementare i loro esperimenti. Le restrizioni della seconda caratteristica possono essere ottenute con un appropriato uso dei permessi, limitando il potere degli altri utenti sul controllo delle entry delle flow tables [6].
Martin Casado, l'inventore di OpenFlow, ha creato questa tecnologia pensando a OpenFlow come uno strumento per semplificare la ricerca nel campo delle reti. Tramite OpenFlow si possono infatti programmare i dispositivi di rete, permettendo la gestione di diverse tipologie di traffico. Inoltre si può sfruttare una rete normalmente già utilizzata per le attività di tutti i giorni e quindi con normale traffico di dati (ad esempio una rete aziendale, oppure la rete di un campus universitario) per avviare sperimentazioni, semplicemente partizionando le due tipologie di traffico, senza avere interferenze ne tantomeno perdita di performance. Questo dovuto al fatto che può essere decisa la quantità di risorse da allocare alla tipologia specifica.
Le sperimentazioni mediante protocollo Openflow oltre che in apparati reali e quindi in reti reali, che richiedono grande disponibilità di risorse, possono essere condotte anche in ambito virtuale, sicuramente meno dispendioso. Grazie a software di simulazione di reti di telecomunicazioni, come ad esempio Mininet, è possibile infatti condurre vari esperimenti e test, come quello precedentemente citato, avvicinandosi quindi al mondo delle SDN, ed il tutto semplicemente utilizzando dei Pc-Desktop o dei Laptop.

CAPITOLO 3

MININET

In questo capitolo si vuole introdurre il software Mininet e alcune delle sue funzioni principali, accennandone poi alcuni dei comandi, in quanto attore principale della parte pratica svolta in questa Tesi.

3.1 Il simulatore

Nell'ambito dello sviluppo di OpenFlow e delle SDN, si è verificata presto l'esigenza di prevedere dei *testbed*[11] in modo da poter ottenere un riscontro alle idee che mano a mano si andavano aggiungendo e quindi di verificarle sul campo. I più diffusi, utilizzano delle macchine virtuali, consentendo di emulare i vari dispositivi di rete e i vari nodi ognuno avviato su una di esse. Sebbene questo tipo di test sia abbastanza valido, in quanto si possono ottenere dati simili alla realtà, mostra dei problemi dovuti alla velocità ed anche alla parte economica. Infatti, pur essendo le macchine virtuali meno costose delle macchine reali, il loro overhead[12] in memoria, limita la possibilità di avviarne un numero troppo elevato all' interno della stessa macchina (dipenderà dalla potenza stessa della macchina su cui vengono lanciate). Inoltre prima di avviare un esperimento che prevede l'utilizzo di VM possono essere necessari diversi minuti, ovvero il tempo che occorre a tutte le macchine virtuali, che fanno parte della rete da testare, ad avviarsi. I test eseguiti in questo modo, di conseguenza, possono riguardare solamente topologie di rete abbastanza limitate, avendo a disposizione risorse limitate. Il software di simulazione Mininet è stato sviluppato proprio con lo scopo di poter eseguire test più estesi utilizzando anche risorse molto limitate, quali ad esempio un PC Desktop o un laptop.

[11] Letteralmente banchi di prova, test per verificare il funzionamento.
[12] In informatica definisce le risorse accessorie, richieste in sovrappiù rispetto a quelle strettamente necessarie per ottenere un determinato scopo.

Mininet gestisce un insieme di terminali di rete (host), switch, router ed anche i vari collegamenti su un unico ambiente Linux ed è in grado di simulare un' intera rete grazie ad una virtualizzazione leggera avvalendosi di tecnologie implementate nel kernel Linux e soprattutto dei *network namespaces*[13], consentendo l'avvio di interfacce virtuali, connesse da cavi virtuali, nell'ambito dell'esecuzione di un singolo sistema operativo.

In questo modo si può ottenere un miglioramento sia nel consumo di memoria che nel tempo di realizzazione e avvio degli esperimenti, riuscendo a realizzare test anche con topologie molto complesse.

Infine il trasferimento sui nodi reali può avvenire praticamente in maniera quasi automatica, permettendo un approccio di prototipazione rapida alle reti di calcolatori.

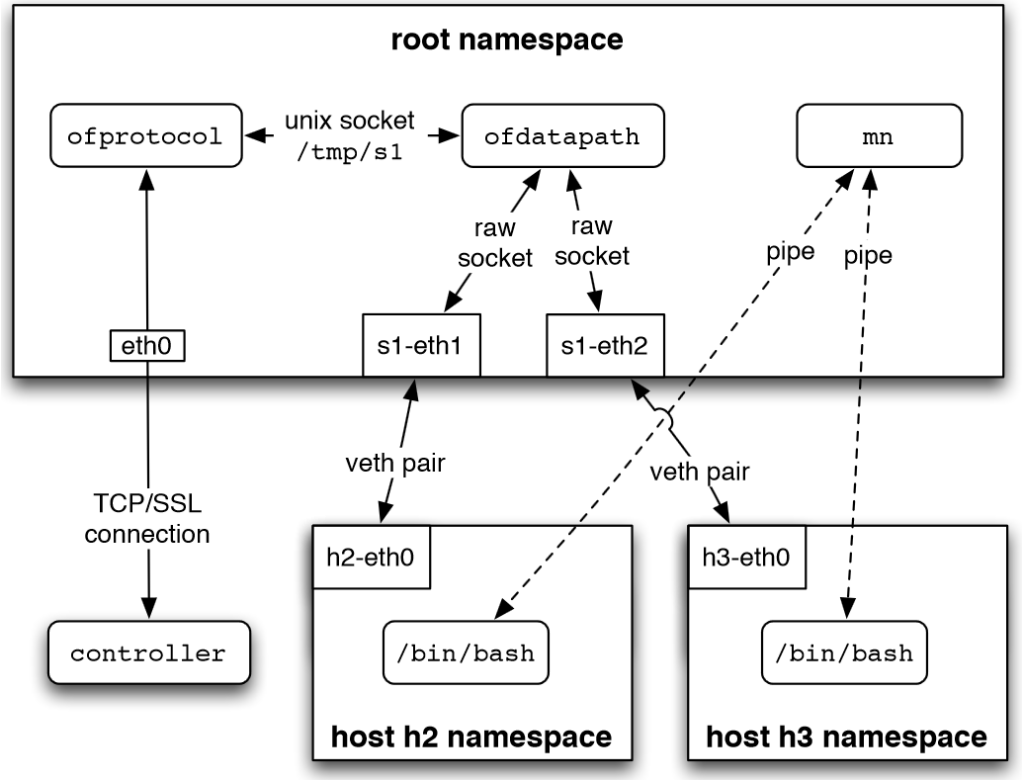


Figura 3.1 Approccio Mininet [8]

[13] permettono di avere differenti e separate istanze delle interfacce di rete e delle tabelle di routing che operano indipendentemente dalle altre.

3.2 Vantaggi

L’utilizzo di Mininet offre sicuramente una serie di vantaggi [9]:

1-Velocità: l'avvio di una semplice rete richiede pochi secondi. Ciò significa che il ciclo di run-edit-debug può essere molto veloce.

2- Possibilità di creare topologie personalizzate: un unico switch, topologie più ampie simili a internet, un centro dati o qualsiasi altra cosa.

3- Possibilità di eseguire programmi veri e propri: tutto ciò che funziona su linux è disponibile per l’esecuzione sui singoli switch e host creati.

4-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.

5- Mininet può essere eseguito anche su un semplice computer portatile, su un server, su una virtual machine e anche su macchine native Linux (Mininet è stato introdotto in UBUNTU dalla versione 12.10 in su).

6- Possibilità di condividere e replicare il codice: chiunque possegga un computer ha la possibilità di eseguire il codice una volta copiato opportunamente.

7- La facilità di utilizzo: si possono creare ed eseguire esperimenti creando semplici, o anche complessi, script in linguaggio python.

8- Il codice è Open Source: si può esaminare e modificare il codice sorgente scaricabile al sito https://github.com/mininet

9- Mininet è in fase di sviluppo attivo: è possibile interagire direttamente con la comunità di sviluppatori.

3.3 Limitazioni

Oltre all'innumerevole serie di vantaggi Mininet presenta anche alcune possibili limitazioni [9]:

1- Eseguire una rete su un unico sistema è comodo ma impone alcune limitazioni: le risorse dovranno essere bilanciate tra gli host della rete.

2- Mininet utilizza un unico kernel Linux per tutti gli host virtuali; questo significa che non è possibile eseguire software che dipende da BSD, Windows, o altri kernel differenti (anche se è possibile collegare macchine virtuali).

3- Mininet non crea il controller, se si necessità di un controller personalizzato bisognerà implementarlo per poi poterlo utilizzarlo.

4- Per impostazioni predefinite, La rete Mininet è isolata dalla LAN e da internet: di solito è una buona cosa per evitare inconvenienti nella rete. Tuttavia, ci sono diversi modi per poter connettere alla rete esterna la rete creata.

5- Per impostazione predefinita, tutti gli host Mininet condividono il file host del sistema e lo spazio PID: bisogna stare attenti quando si eseguono determinati demoni o programmi in /etc/ a non terminare i processi necessari.

6- 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, e che emularle con maggiore velocità non sarà semplice (es: rete a 100 Gbps).

Con poche eccezioni, la maggior parte di queste limitazioni non sono intrinseche di Mininet ed eliminarle è semplicemente una questione di codice, vedremo come, ad esempio, sono state eliminate la terza e la quarta limitazione nel capitolo inerente lo sviluppo di una Rete.

3.4 Lavorare con mininet

Nei seguenti paragrafi si entrerà nell'aspetto pratico di Mininet, mostrandone la semplicità d'installazione, ed elencandone alcune delle principali funzioni per creare delle topologie di rete e quindi verificarne la corretta funzionalità.

3.4.1 Installazione di Mininet [10][11]

Il metodo più rapido e sicuro per installare Mininet su un calcolatore è avvalersi della Virtual Machines (VM) messa a disposizione dagli sviluppatori stessi. Si tratta di un pacchetto contenente un'installazione di Ubuntu Linux 14.04 con Mininet, tutti i tools Openflow preinstallati e alcune modifiche al kernel per poter supportare anche le reti più complesse.
Sarà necessario utilizzare un sistema di virtualizzazione, come ad esempio WMware workstation (uno dei più famosi e potenti ma non gratuito) o Virtual box (gratuito e altrettanto valido) i quali funzionano sia su macchine linux che windows.
Una volta scaricato il pacchetto sarà sufficiente aprirlo con il software di virtualizzazione ed avviare la VM, quindi dopo avere effettuato il login, l'interfaccia si presenterà nel modo seguente:

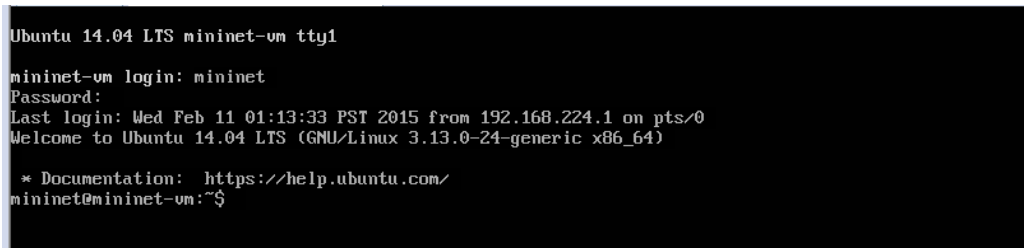


Figura 3.2 interfaccia VM

è un'interfaccia a linea di comando ed il metodo consigliato per interagire con essa è quello di avviare una sessione SSH (secur shell), utilizzando le funzioni di *networking* messe a disposizione dal software di virtualizzazione utilizzato.
Mininet può essere installato anche direttamente su un calcolatore con sistema operativo Linux, ed è presente nelle versioni di Ubuntu successive alla versione 12.10.

3.4.2 creare una topologia di rete [10]

Una volta eseguita la corretta installazione, con Mininet è possibile creare una rete utilizzando un singolo comando e digitandolo sul terminale della VM. Ad esempio:

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

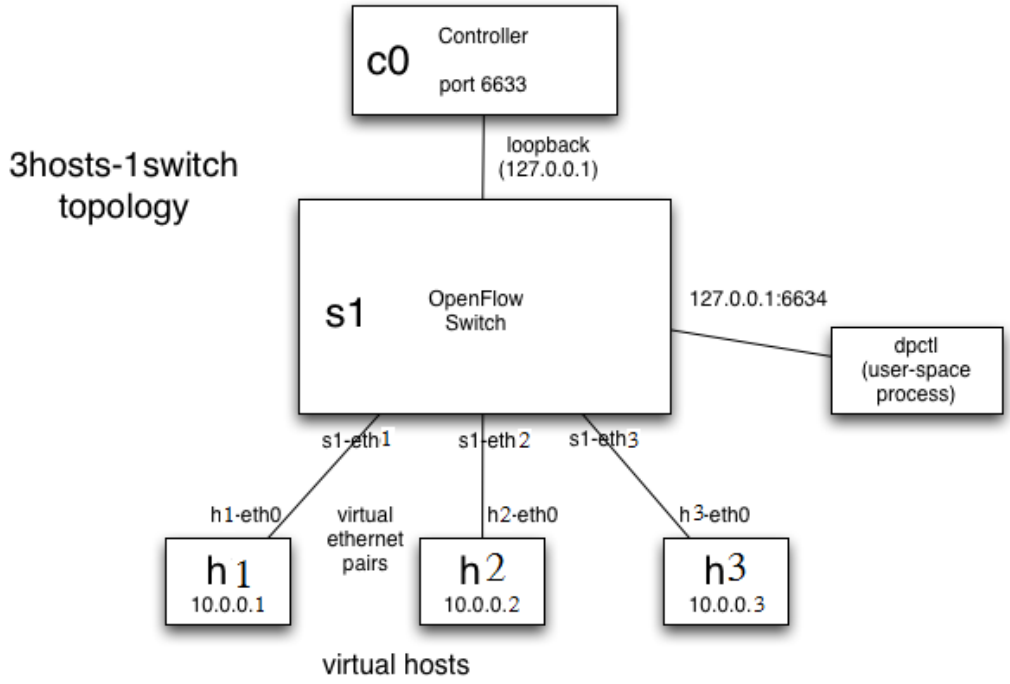


Figura 3.3 Topologia di rete[10]

In questo caso si chiede a Mininet (comando mn) utilizzando i permessi di amministratore (comando sudo di Linux) di creare la topologia rappresentata in figura con le seguenti istruzioni:

1 -- *topo single,3* : crea effettivamente la topologia di rete, uno switch singolo e tre host collegati ad esso.

2 -- *mac* : con questo comando si chiede che ad ogni interfaccia venga assegnato un indirizzo MAC equivalente all'indirizzo IP, invece di un indirizzo casuale.

3 -- *switch* : permette di scegliere quale tipo d'implementazione di switch debba essere utilizzata per la simulazione. Quella scelta, cioè

OVSwitchKernel (ovsk), è un'implementazione di switch OpenFlow compliant come modulo del kernel Linux, che è l'implementazione più efficiente.

4 --*controller* con questo parametro si chiede che gli switch si connettano al controller specificato, invece che a quello predefinito offerto da Mininet. Utilizzando remote senza specificare un indirizzo IP, gli switch si connetteranno all'indirizzo 127.0.0.1 di loopback porta TCP: 6633.

La topologia più semplice da creare, con la quale può anche essere testato il funzionamento del programma è quella realizzata dal semplice comando:

```
$ sudo mn
```

senza specificare opzioni, il quale creerà una semplicissima e minimale topologia con 2 host, 1 switch ed un controller. Molte altre opzioni, oltre a quelle utilizzate nel primo esempio, sono disponibili per creare topologie di reti più o meno complesse e sono visualizzabili con il comando:

```
$ sudo mn -h
```

Un altro metodo per creare topologie di rete con il nostro simulatore è quello di realizzare degli script[14] in codice Python, utilizzando classi, metodi, funzioni e variabili che sono a disposizione in API per Mininet, di seguito se ne elencano alcune delle principali:

-*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 dello switch

[14] file testuali contenenti un insieme organico di istruzioni

-*addLink()* : aggiunge link bidirezionali alla topologia

-*start()*: inizializza la rete

-*stop()*: ferma la rete.

Nella creazione di una rete può essere specificato un controller remoto. Mininet mette a disposizione anche un controller di riferimento che può essere lanciato sulla macchina in cui si vuole avviare, con il comando:

controller ptcp: port

specificando la porta alla quale può essere raggiunto.
Per importare le API in uno script è sufficiente digitare, prima della definizione delle classi principali, il comando from seguito dalla cartella dove si trovano, il comando import ed il nome delle API specifiche, ad esempio:

from mininet.net import Mininet

grazie al quale s'importano le classi principali per la creazione della rete.

L'elenco completo delle API Mininet si può trovare sul sito http://mininet.org/api. Utilizzando gli script si possono creare e salvare topologie per poterle riutilizzare in seguito, in maniera molto rapida e semplice. Come esempio semplicissimo di codice python, si mostra uno script che crea una rete con 4 host e 3 switch ed esegue direttamente un ping da un host ad un altro per la verifica del funzionamento:

```
1 from mininet.net import Mininet
2 from mininet.topolib import TreeTopo
3 tree4 = TreeTopo(depth=2,fanout=2)
4 net = Mininet(topo=tree4)
5 net.start()
6 h1, h4  = net.hosts[0], net.hosts[3]
7 print h1.cmd('ping -c1 %s' % h4.IP())
8 net.stop()
```

3.4.3 Interagire con la rete

Una volta creata la topologia desiderata, l'ambiente Mininet è avviato. Sarà possibile interagire con esso tramite linea di comando CLI (Command line interface), avendo a disposizione una serie di strumenti per analizzare la Rete. L'interfaccia ora si presenterà in questo modo:

```
mininet>
```

Indicandoci che ora siamo in ambiente Mininet, sarà quindi possibile eseguire dei test, ad esempio sulla topologia di rete mostrata nel paragrafo 3.3.2, digitando dei semplici comandi, il primo dei quali potrebbe essere *nodes* che ci permette di visualizzare l'elenco dei nodi creati:

```
mininet> nodes
available nodes are:
 c0 h1 h2 h3 s1
```

come atteso, il comando mostra 3 host (h1 h2 h3) uno switch (s1) ed un ontroller (c0). Per visualizzare tutti i collegamenti tra i vari nodi si usa il comando *net*:

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0
c0
```

ci mostra i nodi e le connessioni associate alle relative porte.

Invece, se come primo comando viene indicato il nome di uno dei nodi, il comando successivo verrà eseguito sul nodo stesso. Come esempio, se vogliamo visualizzare le interfacce del nodo h1 basterà digitare il seguente comando: *h1 ifconfig*

```
mininet> h1 ifconfig
h1-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:01
      inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
      inet6 addr: fe80::200:ff:fe00:1/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:13 errors:0 dropped:0 overruns:0 frame:0
      TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
```

```
        collisions:0 txqueuelen:1000
        RX bytes:1026 (1.0 KB)  TX bytes:816 (816.0 B)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        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)
```

Si può notare che vengono visualizzate le interfacce del nodo di nostro interesse.
Infine per poter visualizzare tutti i comandi a nostra disposizione basterà scrivere il comando *help*, qui di seguito elenchiamo alcuni dei più importanti oltre a quelli già utilizzati:

- *pingall*: con il quale si eseguono dei ping tra tutti gli host connessi alla rete, molto utile per valutare l'effettivo funzionamento dell'intera rete creata.
- *dump*: stampa a viedo informazioni relative a tutti i nodi creati.
- *iperf* il quale esegue un test di banda tra 2 degli host della rete, se utilizzato da interfaccia Mininet senza parametri, viene eseguito tra il primo e l'ultimo.
- *exit* con il quale è possibile uscire dalla rete e tornare all' interfaccia Linux
- *xterm* con il quale è possibile avviare terminali relativi ai vari nodi. Es: xterm h1 avvia un terminale relativo all'host1 nel quale è possibile utilizzare i comandi linux come se fosse una macchina reale.

Oltre ai comandi che possono essere digitati nella CLI vi è la possibilità di accedere ai singoli host, sfruttando il comando xterm, per poter utilizzare i normali strumenti di rete messi a disposizione da Linux, con tutti i comandi che possono essere utili per testare la nostra rete.
La giuda completa con tutti i comandi ed anche le diverse API utili alla creazione di una topologia, da noi consultate e semplicemente introdotte in questo paragrafo, si possono trovare sul sito mininet http://mininet.org/.

3.5 Le potenzialità

La facilità di utilizzo e la molteplicità di funzioni, che mette a disposizione Mininet, consentono di creare reti anche molto complesse e di effettuare numerosi test su di esse. Tutto ciò in un ambiente virtuale, permettendo lo sviluppo di nuovi sistemi di reti e la verifica della loro funzionalità, all'interno di una singola macchina.
Grazie all'utilizzo di altre tecnologie, alcune delle quali verranno descritte brevemente nel Capitolo successivo, come un semplice analizzatore di protocolli di rete, strumenti base di linux, alcune specifiche degli switch virtuali Openflow e la conoscenza di alcune nozioni di python, si possono avviare sperimentazioni sui nuovi concetti di SDN che, negli anni a venire, saranno centro dell'attenzione nell'ambito delle telecomunicazioni.

CAPITOLO 4

SOFTWARE E TECNOLOGIE UTILI PER LO SVILUPPO DI UNA RETE VIRTUALE

In questo capitolo verranno elencate una buona parte di tecnologie utili, insieme a Mininet, allo sviluppo di una rete SDN virtuale e a valutarne l'effettiva funzionalità.

4.1 Software di virtualizzazione

Oggi chiunque può virtualizzare un sistema operativo, senza grosse difficoltà tecniche. La tecnologia di virtualizzazione viene utilizzata per espandere la capacità delle risorse hardware senza dover spostare software e dati da un computer o server ad un altro, è utilizzata anche per abbassare i costi dell'hardware così da poter installare più server virtuali su un'unica macchina. In poche parole consente a due o più sistemi operativi di convivere su una stessa macchina condividendo le risorse fisiche, riuscendo così a sfruttare le potenzialità di entrambi allo stesso tempo.

Avere un ambiente separato consente di effettuare test senza rischiare di compromettere il proprio sistema operativo principale. Solitamente è presente un host, in altre parole il computer principale, dove è installato il programma di gestione, ed un guest, il sistema operativo che viene eseguito nel software di virtualizzazione.

I software di virtualizzazione mettono a disposizione differenti opzioni da poter settare all'avvio delle macchine virtuali, dando l'opportunità anche ai sistemi operativi guest di accedere alla rete ed a tutte le risorse disponibili all'host, utilizzando delle connessioni, anch'esse virtuali, tra host e guest, che a noi torneranno molto utili. Sul mercato sono presenti diversi software per effettuare la virtualizzazione, molti dei quali gratuiti e grazie ai quali la sperimentazione di nuove tecnologie è stata resa anche più sicura.

Nel nostro caso, che verrà presentato nel capitolo successivo, si è voluto virtualizzare Linux contenente Mininet (guest), su un host Windows 7. Importando nel programma il

file messo a disposizione dagli sviluppatori di Mininet, si sono potuti effettuare test, in tutta sicurezza, su topologie di reti da noi create.

4.2 Python

Python è un linguaggio di programmazione dinamico orientato agli oggetti utilizzabile per molti tipi di sviluppo software. Offre un forte supporto all'integrazione con altri linguaggi e programmi, è fornito di un'estesa libreria standard e può essere imparato in pochi giorni [12].

Supporta diversi paradigmi di programmazione, come quello object-oriented (compresa l'ereditarietà multipla), quello imperativo e quello funzionale. La presenza di una libreria estremamente ricca unitamente alla gestione automatica della memoria e a robusti costrutti per la gestione delle eccezioni lo rende uno dei linguaggi più ricchi e comodi da usare, ma allo stesso tempo semplice da imparare. I blocchi logici vengono costruiti semplicemente allineando righe allo stesso modo, utilizzando una sintassi pulita e snella così come i suoi costrutti. Nasce per essere un linguaggio facilmente intuibile e utilizzabile da chiunque.

Python è un linguaggio pseudocompilato, non esiste quindi una fase di compilazione separata che genera un file eseguibile (come avviene in altri linguaggi: C, java): l'interprete si occupa semplicemente di analizzare il codice sorgente (salvato in file testuali con estensione .py) e di eseguirlo. Una volta scritto un sorgente, esso può essere interpretato ed eseguito sulla gran parte delle piattaforme attualmente utilizzate, siano esse di casa Apple (Mac), che Windows o GNU/Linux.

Infine può essere scaricato e utilizzato gratuitamente per applicazioni proprie, oltre ad essere modificato e aggiornato sfruttandone la licenza open-source. Tutte queste caratteristiche lo hanno reso protagonista in diversi ambiti in quanto garanticsce uno sviluppo rapido e a volte anche divertente in tutti i contesti: dal desktop al web, passando anche attraverso lo sviluppo di videogiochi e infine allo scripting di sistema.

Abbiamo voluto presentare python in quanto Mininet è stato implementato con questo linguaggio (su basi di C), e vi è anche la possibilità di implementare Controller per reti SDN. Controller POX, NOX, e Ryu ad esempio sono implementati in python. Sono

disponibili numerose guide e tutorial sull' utilizzo di questo linguaggio (ne sono state consultate alcune messe a disposizione sul sito www.python.org e sul sito italiano www.python.it.) per apprendere il funzionamento base di programmazione e riuscire ad utilizzarlo per creare un Controller Pox ed anche delle Topologie di rete, sfruttando le varie librerie e API messe a disposizione online e già presenti nella nostra versione di Mininet.

4.3 Controller Pox

Pox è un framework[15] che consente il rapido sviluppo di un software di controllo per una rete e la scittura di Controller Openflow. Pox è basato interamente su linguaggio python.

La versione Linux nella quale è presente Mininet è completa già al suo interno del framework, quindi è possibile scrivere, consultando anche delle API messe a disposizione (POX API) e ovviamente tenendo conto delle specifiche del protocollo Openflow, un controller in linguaggio python per poterlo poi direttamente utilizzare per la rete creata. Le specifiche sono visionabili sul sito www.opennetworking.org nella sezione appropriata.

Per avviare Pox è sufficiente lanciare il comando *pox.py* dalla cartella in cui è installato, oppure nel caso in cui si è in fase di sviluppo si può lanciare in modalità di debug utilizzando *debug-pox.py* in modo che verranno visualizzati gli errori e gli imprevisti opportunamente settati in fase di sviluppo.

Ci sono alcune opzioni che possono essere aggiunte nella fase di lancio di pox:

--verbose: mostra informazioni extra

--no-cli: non fa partire la shell interattiva

--no-openflow: non si mette in ascolto automaticamente per le connessioni Openflow.

[15] In informatica, e specificatamente nello sviluppo software, un framework è un'architettura (o più impropriamente struttura) logica di supporto (spesso un'implementazione logica di un particolare design pattern) su cui un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore.

In Mininet sono già presenti alcuni esempi di controller di base tra i quali l'implementazione di un hub e di alcuni learning switch, grazie ai quali è possibile effettuare alcuni test di base o può essere preso spunto per crearne degli altri.
Nel capitolo successivo verrà implementato un controller POX per la topologia di rete da noi voluta, dove saranno brevemente illustrate alcune funzioni utilizzate.
Per maggiori dettagli sul funzionamento del framework si rimanda alla guida on-line da noi consultata: https://openflow.stanford.edu/display/ONL/POX+Wiki.

4.4 ssh, putty, xterm

Secur shell (ssh) è un protocollo di rete che permette di stabilire una sessione remota cifrata tramite interfaccia a riga di comando con un altro host di una rete informatica [13]. Verrà utilizzato per interfacciarci con la macchina virtuale direttamente in da Windows, con la possibilità di lanciare i comandi e semplificarci l'utilizzo di Mininet, dandoci la possibilità di aprire anche più terminali contemporaneamente.
Per poter utilizzare il protocollo ssh in windows, è necessario installare un programma apposito in quanto, al contrario di linux, non è gia implementato sul sistema operativo.
La nostra scelta è ricaduta su putty (http://www.putty.org/), programma gratuito, che permette in maniera molto semplice, mediante un'interfaccia grafica, di avviare una sessione ssh. Nel nostro caso utilizzato tra windows e Linux dove Mininet è avviato, basterà semplicemente indicare l'indirizzo IP per potersi connettere inserendo, poi, utente e password di accesso.

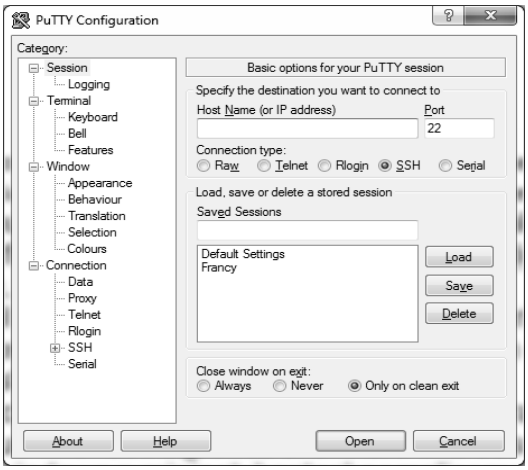


Figura 4.1 Interfaccia di putty

Una volta connessi verrà visualizzata l'interfaccia di Mininet, o meglio di linux nel quale quest'ultimo è avviato, e si presenterà in questo modo:

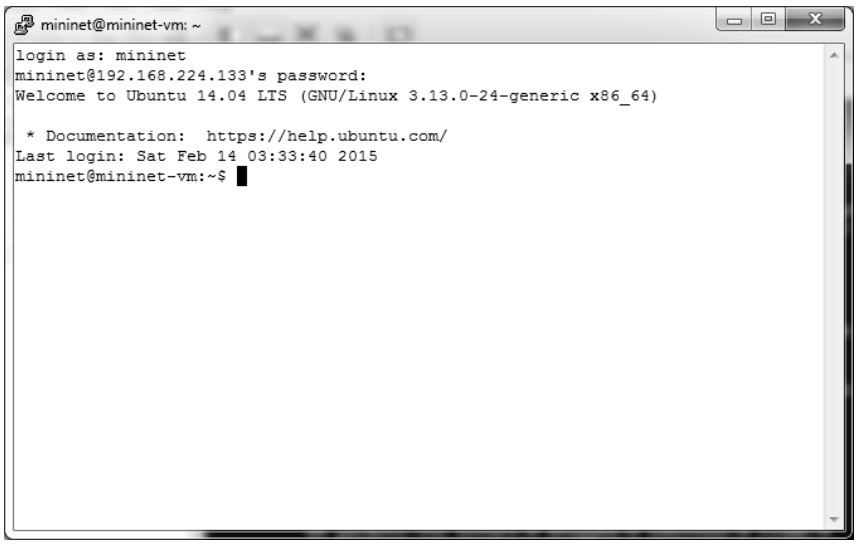


Figura 4.2 Interfaccia Mininet tramite protocollo ssh

Una volta interfacciati abbiamo la possibilità di lavorare con Mininet e creare la nostra rete utilizzando tutti i comandi a disposizione.

Infine, per poter interagire con i diversi host creati nella nostra rete emulata o per avviare programmi grafici tramite connessione ssh, necessitiamo di un programma che consenta il tunneling delle applicazioni grafiche tra Linux e Windows. In questo caso si è utilizzato Xming (http://www.straightrunning.com/XmingNotes/), anch'esso gratuito, il quale abilita X forwarding e crea un X server per Windows consentendo il passaggio di applicazioni grafiche tra i terminali delle 2 macchine connesse. In questo modo è possibile sfruttare la possibilità, che linux mette a disposizione, di eseguire più emulatori di terminale e avviarne molteplici sessioni ognuna delle quali fornisce un sistema di input-output per i processi lanciati.

Una volta settati i giusti parametri per putty e Xming, e stabilita la connessione ssh, è possibile lavorare con mininet direttamente da Windows.

4.5 Spanning Tree Protocol

La creazione di reti complesse, soprattutto a livello fisico, comporta alcune problematiche. Tra queste vi è quella di gestire le richieste di broadcast inviate dagli end host agli switch, che le replicano verso tutte le porte di uscita, eccetto quella da cui proviene, generando cosi un effetto valanga all'interno della rete, il cosiddetto broadcast storm, limitando e saturando la rete. Questo problema si presenta in reti nelle quali sono presenti delle connessioni ad anello ovvero tanti switch o bridge connessi tra loro fino a formare dei percorsi chiusi, ad anello appunto.
Una soluzione a questa problematica è il protocollo spanning tree (STP). Quest'ultimo è un algoritmo distribuito, che opera su tutti gli switch e bridge, in modo tale che, in ogni istante, la rete sia connessa ma priva di cicli, ovvero che il grafo dei collegamenti disponibili sia "coperto" da un albero.
Tutto ciò richiede una generazione di traffico tra gli switch che devono comunicare tra loro per creare una visione generale della rete (albero appunto) e gestire, quindi, il problema, andando ad agire sulle porte e limitando di conseguenza la ridondanza della rete.
La topologia da noi utilizzata in seguito richiederà la risoluzione del problema dei loop, generati appunto dai broadcast, e, in uno dei vari test, la soluzione presa in considerazione sarà proprio l'utilizzo del protocollo STP.

4.6 Wireshark

Wireshark è un analizzatore di protocolli di rete gratuito, ed è stato utilizzato per effettuare la verifica del corretto inoltro dei pacchetti, sia del protocollo Openflow che dei normali protocolli di rete.
Il programma dispone di un interfaccia grafica che, grazie a putty e xming, abbiamo potuto gestire da windows lanciandolo su mininet e sui vari terminali inerenti i nodi della rete. L'utilizzo di wireshark è molto semplice, infatti, una volta scelta l'interfaccia da cui prendere le informazioni, basta avviare la cattura per poi poterle comodamente

consultare una volta terminata. Tutto ciò ci ha aiutato ad osservare il traffico delle interfacce di nostro interesse.
Le ultime versioni sono in grado di analizzare anche pacchetti relativi al protocollo OpenFlow. Una di queste è già presente nel pacchetto della Macchina virtuale contenente mininet. Avviare il programma non è complicato, sarà sufficiente, dall'interfaccia a linea di comando, digitarne il nome, utilizzando i permessi di amministratore. In caso in cui il software non fosse gia presente nel pacchetto Mininet, vi è la possibilità di scaricarlo direttamente dal sito: https://www.wireshark.org/.

4.7 Emacs

Emacs è un editor di testo, presente in molte macchine Linux, è utilizzabile sia tramite terminale che tramite interfaccia grafica, supporta molti dei linguaggi di programmazione, aiutando in parte, nella correzione degli errori. L'editor è stato da noi utilizzato per creare degli script eseguibili, avviandolo e salvando i file, direttamente sulla macchina virtuale.

4.8 Alcuni comandi utili

Nel ultimo paragrafo di questo capitolo si vogliono elencare alcuni comandi che saranno utilizzati frequentemente, alcuni dei quali utili alla valutazione di una rete e disponibili in ambiente Linux, altri utili nella gestione degli switch virtuali.

4.8.1 Ping

Ping (Packet internet goper) è un'utility di amministrazione per reti di computer usata per misurare il tempo, espresso in millisecondi, impiegato da uno o più pacchetti ICMP a raggiungere un dispositivo di rete (attraverso una qualsiasi rete informatica basata su IP) e a ritornare indietro all'origine [14]. È solitamente utilizzato per verificare la

presenza e la raggiungibilità di un altro computer connesso in rete e per misurare le latenze di trasmissione di rete. In pratica tramite ping viene inviato un pacchetto ICMP di tipo *echo request* e si rimane in attesa di un pacchetto ICMP di tipo *echo reply* in risposta. Infatti, solitamente la parte di sistema operativo dedicata alla gestione delle reti è programmata per rispondere in maniera automatica con un pacchetto di tipo *echo reply* alla ricezione di un pacchetto di tipo *echo request*.

Ping , una volta lanciato, visualizza sullo *standard output*[16] il numero di pacchetti inviati e ricevuti, la loro dimensione, il tempo trascorso tra l'invio di ogni pacchetto e la ricezione della risposta corrispondente, la media dei tempi e la percentuale di risposte ottenute.

Mediante l'utilizzo di questo comando è stato possibile verificare, durante i nostri test, la corretta creazione della topologia di rete da noi voluta e l'effettiva raggiungibilità di tutti gli host ad essa connessi.

4.8.2 Iperf

Iperf è un' utility di uso comune nei test di rete, utilizzato per misurare la velocità di trasmissione di flussi di dati TCP e UDP.

Iperf consente all'utente di impostare vari parametri i quali possono essere utilizzati per testare una rete, o in alternativa per l'ottimizzazione della stessa. Presenta anche una funzionalità di client e server che consente di misurare la velocità tra 2 estremi della rete sia in modalità unidirezionale che bidirezionale.

Un' uscita tipica del comando iperf contiene un rapporto, con data e ora, della quantità di dati traferiti e la relativa velocità di trasmissione.

Il modo più comune per eseguire il comando è quello di usarlo, in modalità TCP, tra 2 host in uno dei quali ci si mette in ascolto come server con il comando

iperf –s

mentre dal secondo host ci si connette al primo, specificandone l'indirizzo ip, in modalità client

iperf –c *ipadress –option*

[16] Lo standard output, abbreviato in stdout, è quello che usano i programmi per scrivere le informazioni all'utente.

Si possono specificare varie opzioni, una delle più utilizzate è –t che permette di indicare quanto deve durare la trasmissione di dati.
Nel nostro caso il comando è stato utilizzato più volte, in modalità TCP, per dei test e, nell'ultima parte, per generare del traffico dati all'interno della nostra rete, dandoci l'opportunità di misurare alcune performance di banda tra le nostre connessioni.

4.8.3 cut e grep

Questi due comandi sono stati utilizzati all'interno di alcuni script shell (programmi eseguibili in ambiente shell linux), per aiutarci ad isolare alcune variabili di nostro interesse.
Cut (letteralmente taglia) è un comando dei sistemi operativi Unix e Unix-like, che legge uno o più file di testo (o lo *standard input*) estraendo da ogni linea delle sezioni, definite in termini di byte oppure caratteri oppure campi, le quali sono poi mostrate sullo *standard output*. Può essere considerato un tipo di filtro.
Grep anche esso è un comando dei sistemi operativi Unix , che ricerca in uno o più file di testo le linee che corrispondono ad uno o più modelli specificati con espressioni regolari o stringhe letterali, e produce un elenco delle linee (o anche dei soli nomi di file) per cui è stata trovata corrispondenza. Anche grep può essere utilizzato per filtrare determinate caratteristiche. A disposizione dei comandi Cut e grep ci sono svariate opzioni grazie alle quali è possibile isolare alcuni caratteri o numeri, all'interno di file ma anche tra le uscite di alcuni comandi.
Si vuole fare un esempio, nel caso in cui volessimo isolare l'indirizzo ip del interfaccia eth0 del nostro pc prendendola dall'uscita del comando ifconfig il quale ci mostra:

```
mininet@mininet-vm:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:a9:51:7f
          inet addr:192.168.224.133  Bcast:192.168.224.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:276 errors:0 dropped:0 overruns:0 frame:0
          TX packets:240 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:29816 (29.8 KB)  TX bytes:29228 (29.2 KB)
```

é possibile farlo isolando prima la riga di nostro interesse con il comando grep seguito da un riferimento per trovare la linea giusta (nel nostro caso dell'esempio è stato utilizzato "adrr:")

```
mininet@mininet-vm:~$ ifconfig |grep addr:
      inet addr:192.168.224.133  Bcast:192.168.224.255  Mask:255.255.255.0
```

per poi andare a isolare l'indirizzo ip con il comando cut specificando l'opzione –d , con il quale si indica un riferimento per tagliare e l'opzione –f con la quale si indica la colonna da selezionare, nel nostro esempio è stato utilizzato il riferimento ":" per l'opzione –d e quindi la nostra stringa verrà divisa cosi

colonna1 colonna 2 colonna3 colonna4

inet addr : 192.168.224.133 Bcast : 192.168.224.255 Mask : 255.255.255.0

utilizzando poi l'opzione –f 2 andiamo a selezionare la colonna 2 ottenendo

192.168.224.133 Bcast

sul quale poi viene riutilizzato cut questa volta usando il carattere spazio " " come divisore per poi selezionare la colonna 1.

```
mininet@mininet-vm:~$ ifconfig |grep addr:|cut -d: -f2|cut -d' ' -f1
192.168.224.133
```

I comandi grep e cut sono stati utilizzati in sequenza, e quindi sullo stesso output, grazie all' introduzione del carattere "|".
Questi comandi verranno utilizzati come filtro per ricavare dati di nostro interesse, e poterli quindi isolare ed elaborare in maniera appropriata per i nostri scopi.

4.8.4 Comandi per la gestione degli switch virtuali openflow

Per la creazione di una rete virtuale con Mininet è fondamentale l'utilizzo di switch virtuali, e open vSwitch è il programma incaricato per questo scopo. Per la gestione

degli switch virtuali Openflow (ovs-Switch) abbiamo a disposizione alcuni comandi tra i quali ovs-ofctl e ovs-vsctl .
Qui di seguito verranno elencate una serie di funzioni del comando *ovs-ofctl* che permettono la gestione, la visualizzazione e anche la modifica delle tabelle di flusso degli switch Openflow da linea di comando:

-*ovs-ofctl* per visualizzare le impostazioni e le flow table degli switch e si utilizzano le seguenti opzioni:

- show s: stampa a video le informazioni riguardanti lo switch s, incluse le informazioni delle Flow-Table e delle porte. Esempio:
$ sudo ovs-ofctl show s1

- *dump-ports s [port]* : stampa delle statistiche a console per i dispositivi di rete connessi con lo switch s. è possibile specificare anche il numero di porta dal quale si vuole ricavare le informazioni. Esempio:
$ sudo ovs-ofctl dump-ports s1 3

- *dump-flows s [flows]*: visualizza nella console tutte le flow entry della flow table dello switch s che corrispondono alla sintassi del flusso indicato. Se la seconda opzione viene omessa, vengono restituite tutte le flow entry dello switch indicato.

-*ovs-ofctl* per la gestione delle flow table all'interno di uno switch con le opzioni:

- *add-flow switch flow*: permette di aggiungere una flow ad uno switch.

-*del-flow switch*: permette di eliminare una flow ad un switch.

-*mod-flow switch flow*: consente di modificare le azioni nelle entry delle tabelle di switch che corrispondono ai flussi specificati.

Nei precedenti comandi *flow* specifica i vari flussi e quindi le rispettive entry che possono essere modificate, sono rispettivamente:
- *in port* port: specifica i flussi entranti dalla porta port;
- *nw src*=ip: indica i flussi che hanno come sorgente l'indirizzo specificato

- *nw dst*=ip: indica i flussi che hanno destinazione l'indirizzo specificato
- *tp src*=port: indica la porta sorgente dalla quale il flusso è stato spedito
  (numero compreso tra 0 e 65535);
- *tp dst*=port: indica la porta destinazione con lo stesso range di valori
  del campo precedente;
- *dl type*=ethertype: ethertype è un campo composto da due ottetti
  del frame ethernet utilizzato per incapsulare il protocollo utilizzato nel payload del frame. I più utilizzati sono i seguenti:
  arp : 0x0806
  icmp : 0x0800, nw proto = 1
  tcp : 0x0800, nw proto = 6
  udp : 0x0800, nw proto = 17
  dove il campo nw proto corrisponde al numero di protocollo IP utilizzato nel campo protocol del header del pacchetto IP. Quindi è utilizzato per riconoscere il particolare protocollo utilizzato nel caso in cui il dl type sia lo stesso.
- *idle timeout*= seconds: specifica il numero di secondi dopo i quali
  l'azione, se non utilizzata in questo lasso di tempo, viene eliminata dalla flow table.
- *priority*= value: specifica la priorità che ha una entry, se ad esempio sono
  presenti più regole per uno stesso flusso, verrà applicata quella a priorità maggiore.

Mentre con il comando ovs-ofctl è interamente dedicato alla gestione dello switch, il comando ovs-vsctl comprende opzioni anche per le interfacce ed il controller, alcune delle quali ci ritorneranno utili in seguito:

-*ovs-vsctl add* port : consente di aggiungere allo switch virtuale una nuova porta.

-*ovs-vsctl set Bridge* switch *stp_enable= false/true* : consente di abilitare o disabilitare il protocollo spanningtree all' interno dello switch specificato.

-*ovs-vsctl list Bridge* switch: ci mostra le informazioni relative allo switch, come nome, stato, se è attivo o meno il protocollo STP, e molte altre informazioni.

Grazie a questi comandi è possibile configurare uno switch Openflow settando i parametri e le opzioni più opportuni o analizzare dati relativi ai flussi.
Per l'elenco completo dei comandi e delle specifiche degli switch Openflow si può visionare il sito *http://openvswitch.org* dal quale si è preso spunto per la realizzazione di questo paragrafo.

CAPITOLO 5

REALIZZAZIONE E TEST DI UNA RETE VIRTUALE DISTRIBUITA

Dopo avere appreso i concetti fondamentali di SDN e il funzionamento del protocollo Openflow, illustrati in precedenza, si è voluto metterli in pratica realizzando una serie di prove, mediante piattaforma Mininet e l'utilizzo degli strumenti a nostra disposizione, per poter infine creare una topologia di Rete distribuita su più macchine virtuali (VM) e dimostrarne il corretto funzionamento.
Nel nostro caso si è deciso di utilizzare due VM avviate sulla stessa macchina, nelle quali sono state implementate inizialmente 2 reti separate ma con la stessa topologia.

5.1 Descrizione della Topologia di Rete utilizzata

Una volta stabilito il traguardo da raggiungere, ossia quello di creare 2 reti separate per poi connetterle e realizzarne una unica, si è decisa la Topologia da utilizzare per le singole reti e la nostra scelta è ricaduta su una rete di minima ridondata come quella mostrata in figura 5.1. Si tratta di una rete, in versione ridotta, tipica di un data center nella quale tutti gli host, nel nostro caso 6, sono connessi a 2 switch principali S1 e S2, mentre gli switch S4 e S5 forniscono percorsi aggiuntivi al fine di migliorare le prestazioni.
La rete è stata creata prima su una Macchina Virtuale e, dopo averne verificato il corretto funzionamento ed i vantaggi forniti dall'utilizzo del protocollo Openflow, è stata replicata su una seconda VM (figura 5.2) per poter, infine, procedere al collegamento delle due grazie anche alle possibilità di Networking fornite dal software di virtualizzazione.

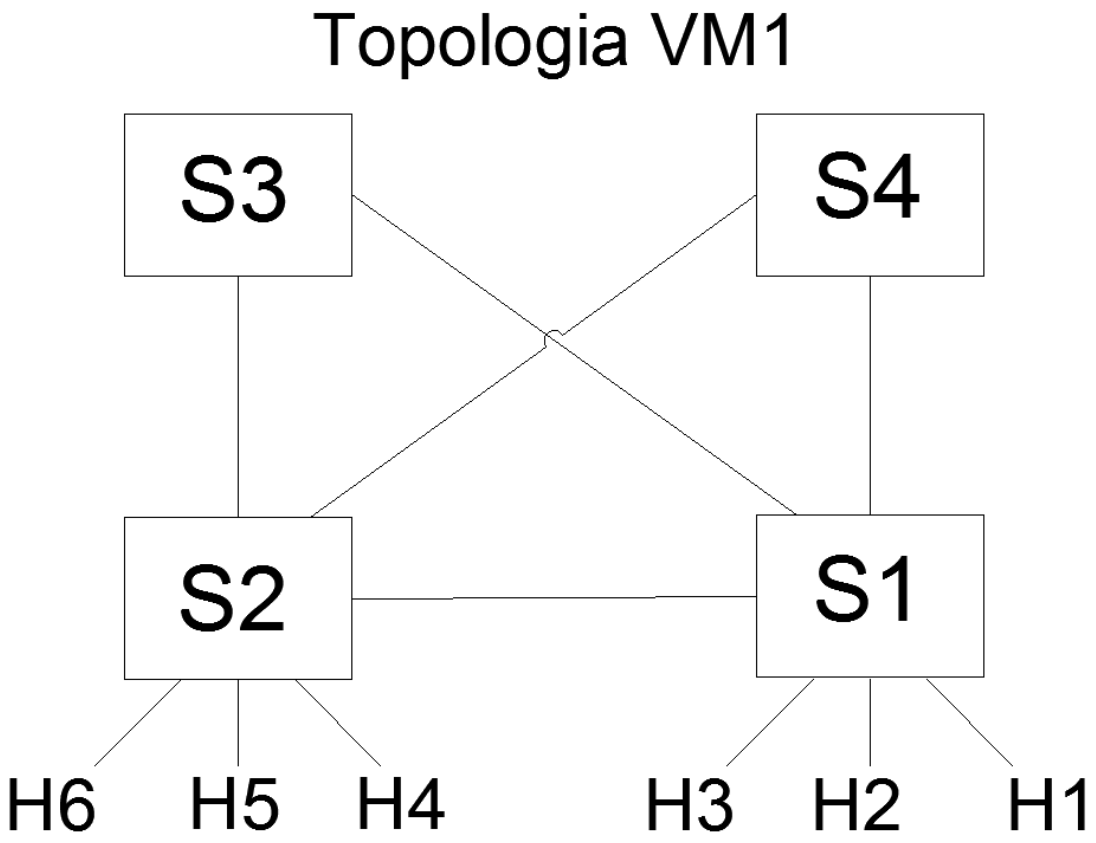


Figura 5.1 Topologia VM1

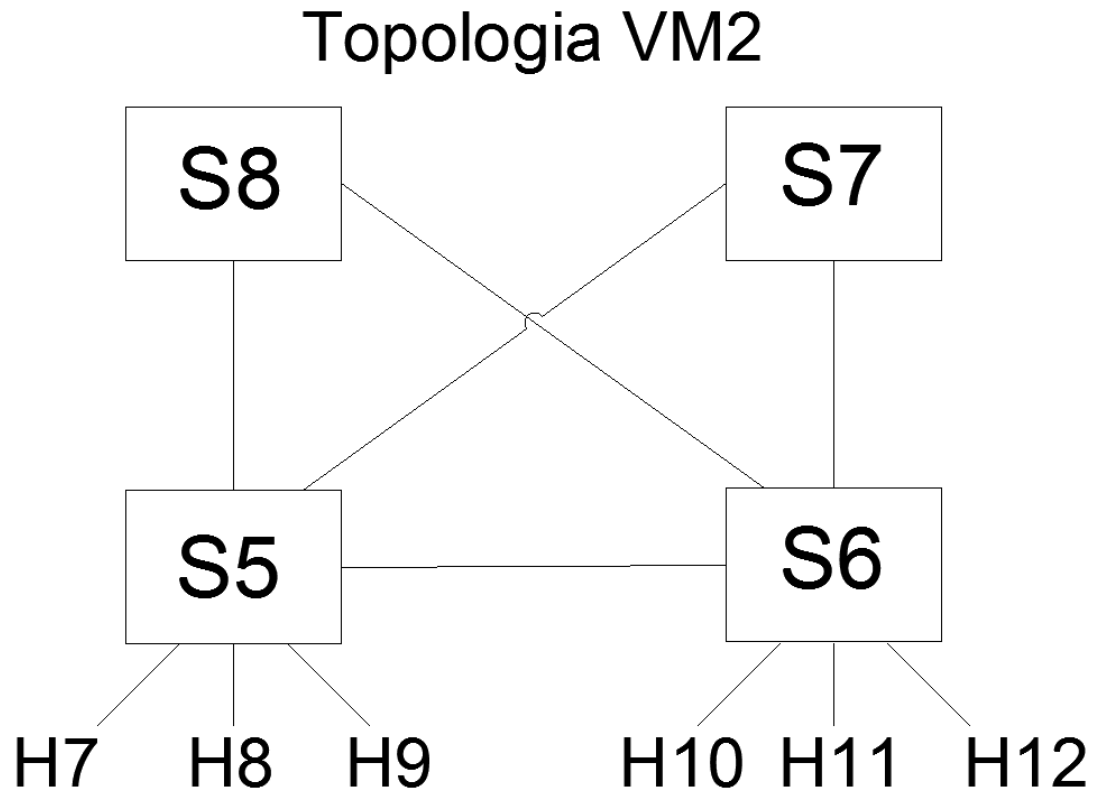


Figura 5.2 Topologia VM2

La nostra rete finale, distribuita su ambedue le VM, risulterà come quella mostrata nella figura 5.3 di seguito:

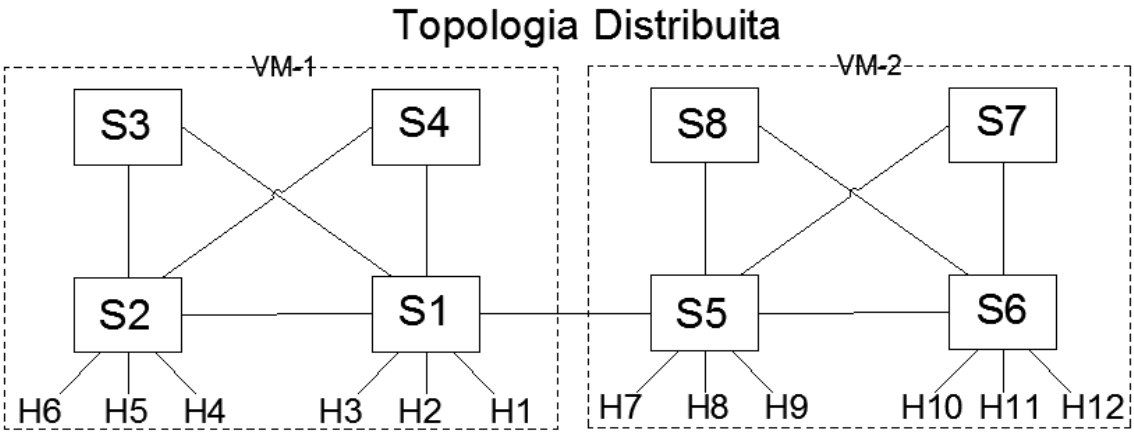


Figura 5.3 Topologia Rete Distribuita

Una volta definita sulla carta si è passati alla pratica iniziando da una singola VM.

5.2 Inizializzazione dei software utilizzati

Per poter procedere con la creazione della topologia di rete si è reso prima necessario scaricare i software utili alla realizzazione: software di virtualizzazione, il pacchetto contenente Mininet e, per semplificarci il lavoro, anche Putty e Xming.

Una volta installato il software di virtualizzazione si è avviata la macchina virtuale contenente Mininet e, come prima cosa, se ne è verificata la corretta installazione eseguendo i comandi base introdotti nel relativo capitolo. Si è poi passati alla configurazione di Xming e Putty per accedere, tramite protocollo ssh, alla macchina virtuale e poter avviare applicazioni grafiche con la possibilità di utilizzare più terminali per l'esecuzione contemporanea di molteplici comandi.

L'accesso da remoto tramite ssh è reso possibile anche grazie ad alcune opzioni del software di virtualizzazione. Quest'ultimo consente la creazione di connessioni virtuali di diverso tipo all'interno della stessa macchina su cui è avviato. Nel nostro caso si è realizzata una LAN con indirizzamento IP 192.168.224.0/24 dove la nostra prima

macchina virtuale, contenente Mininet, ha come indirizzo 192.168.224.133 associato alla porta eth0, ed il nostro host principale, con sistema operativo Windows 7, occupa la posizione di gateway con indirizzo 192.168.224.1.
La possibilità di creare nuove connessioni con il software di virtualizzazione tornerà molto utile anche in seguito.
Una volta verificato l'indirizzo della nostra VM è stato sufficiente inserirlo nel programma Putty per avviare il terminale da remoto e poter interagire con Linux e, di conseguenza, con Mininet da ambiente Windows 7.
Eseguiti tutti i passaggi si è potuta iniziare l'implementazione della rete.

5.3 Creazione della topologia minimale

Innanzi tutto si è reso necessario studiare l'implementazione di una rete Mininet e consultare molte delle API messe a nostra disposizione dagli sviluppatori direttamente sul loro sito o contenute all'interno della macchina stessa ed accessibili attraverso il classico comando *help*.
Onde evitare di dover implementare ogni volta la rete e per averla subito a disposizione, salvandola in maniera opportuna, si è effettuata la creazione della topologia realizzando uno script di codice Python ed utilizzando le librerie già presenti in Mininet, importandole adeguatamente.
Una volta creato un nuovo file di testo, avvalendosi di emacs, come prima riga è stato scritto il percorso dell'interprete (Python), necessario alla corretta esecuzione dello script, seguito nelle righe successive da tutte le librerie API importate e a noi necessarie:

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController
from mininet.cli import CLI
from mininet.link import Intf
from mininet.log import setLogLevel, info
```

from mininet.link import TCLink

dopo di che si è definita la classe principale, myNetwork, con la quale viene creata la rete mininet senza impostare la topologia, che andremo noi a definire, e impostando le connessioni di tipo TCLink, utili in seguito per impostare la banda del canale a loro dedicata,

```
net = Mininet(topo=None,
              build=False, link=TCLink)
```

per poi aggiungere rispettivamente i componenti, richiamandoli con l'istruzione nett.add*Elemento,* che sono:

- un controller (nominato c0); si è optato per un controller remoto, o meglio, un controller che poi verrà avviato nella macchina con indirizzo 192.168.224.133 alla porta 6633

```
net.addController(name='c0',
                  controller=RemoteController,
                  ip='192.168.224.133',
                  port=6633)
```

-gli switch virtuali (s1,s2,s3,s4),

```
s1 = net.addSwitch('s1')
s2 = net.addSwitch('s2')
s3 = net.addSwitch('s3')
s4 = net.addSwitch('s4')
```

-un interfaccia aggiuntiva allo s1, tornerà utile in seguito

```
Intf('eth1',node = s1)
```

-i 6 host con i relativi indirizzi IP , si è utilizzata una rete 192.168.2.0/24

```
h1 = net.addHost('h1', ip='192.168.2.21')
h2 = net.addHost('h2', ip='192.168.2.22')
h3 = net.addHost('h3', ip='192.168.2.23')
```

h4 = net.addHost('h4', ip='192.168.2.24')
h5 = net.addHost('h5', ip='192.168.2.25')
h6 = net.addHost('h6', ip='192.168.2.26')

- infine le connessioni tra i diversi host e switch specificando la banda a loro dedicata nel nostro caso sono stati utilizzati link a 10Mb/s (l'utilizzo di link del tipo TCLink impostati all' inizio permette di aggiungere alcuni parametri relativi alle interconnessioni virtuali come appunto la banda) :

net.addLink(h1, s1, bw=10)
net.addLink(h2, s1, bw=10)
net.addLink(h3, s1, bw=10)

comandi con i quali si connettono gli host h1, h2 ,h3 allo switch 1

net.addLink(s1, s2, bw=10)
net.addLink(s1, s3, bw=10)
net.addLink(s1, s4, bw=10)
net.addLink(s2, s3, bw=10
net.addLink(s2, s4, bw=10)

comandi con i quali gli switch vengono connessi tra loro

net.addLink(h4, s2, bw=10)
net.addLink(h5, s2, bw=10)
net.addLink(h6, s2, bw=10)

comandi con i quali gli host h4, h5 h6 vengono collegati allo switch 2.
Aggiunti tutti i componenti rimane solamente da avviare la rete

net.start()

e la Command Line Interface per poter comunicare con gli host da linea di comando Mininet.

CLI(net)

Per la corretta esecuzione dello script l'interprete richiede di indicare la Main class, ovvero la classe principale, quindi come ultime righe di codice vanno inserite le seguenti:

if __name__ == '__main__':

myNetwork()

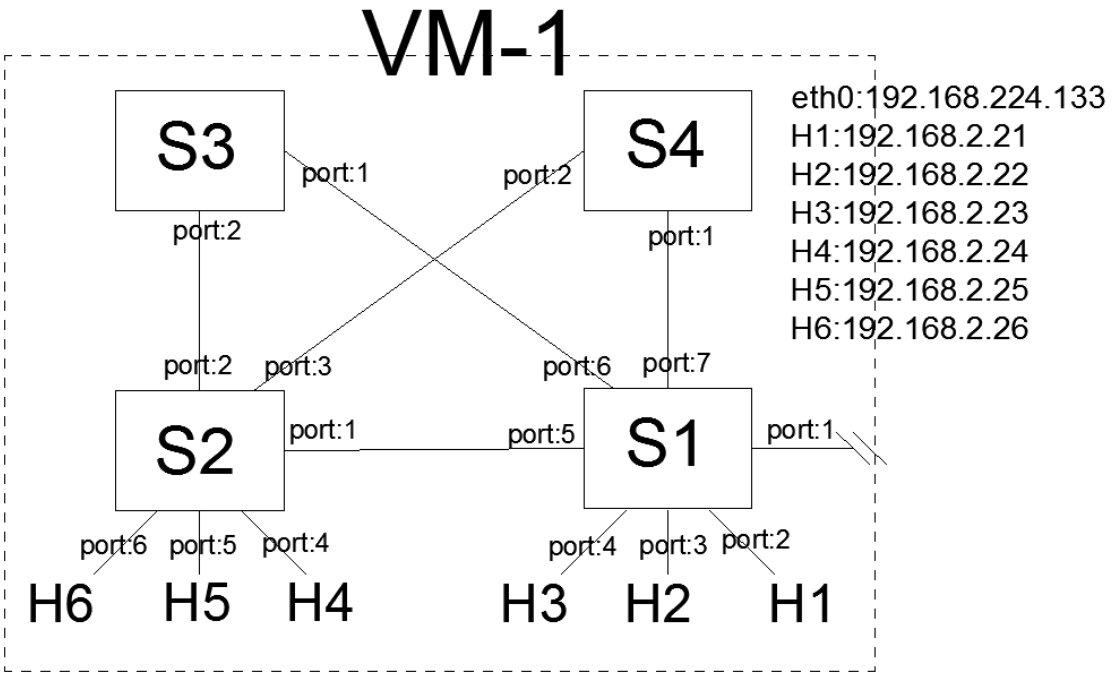


Figura 5.4 Topologia VM1 con relative porte e indirizzi associati ai nodi

Una volta terminato lo script, la cui versione completa si trova in appendice, è sufficiente salvarlo con estensione .py (il nostro file è stato chiamato Net4switch6hostControllerRemote.py) e lanciarlo per valutarne la corretta struttura con il seguente comando:

mininet@mininet-vm:~$sudo python Net4switch6hostConttrollerRemote.py

Una volta eseguito, se tutto è andato a buon fine e non sono presenti errori, mostrerà, prima, tutti gli elementi inseriti e, poi, la Command Line Interface di Mininet come in figura 5.6 (pagina seguente).

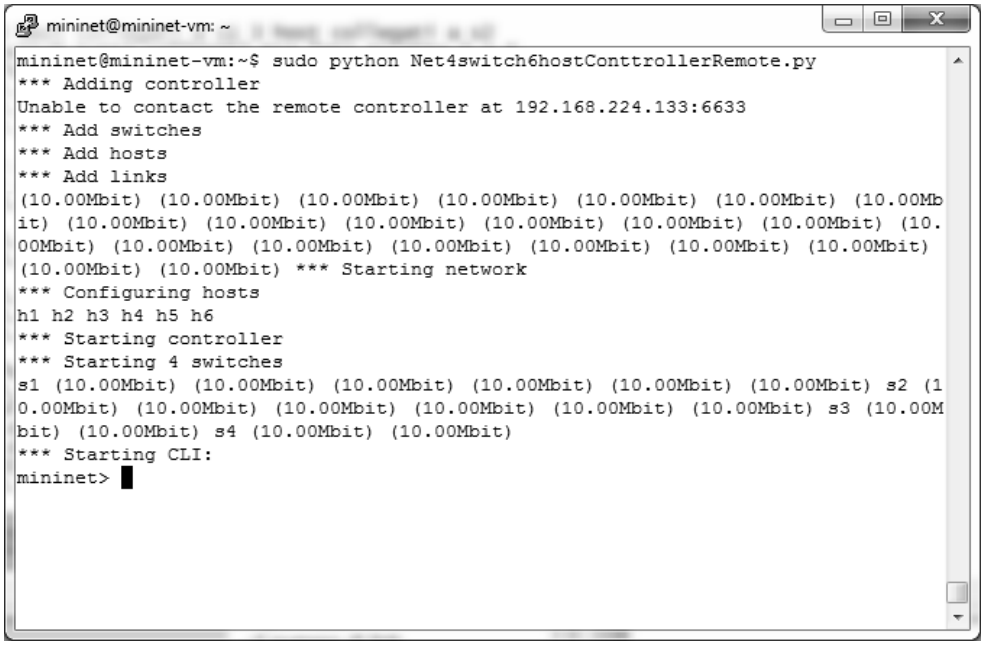


Figura 5.6 Avvio topologia VM1

Tramite i comandi *nodes*, *net* e *dump* (figura 5.7) possiamo verificare la correttezza dei nodi creati, nonchè delle relative connessioni della Rete e annotare il numero delle porte eth degli switch in quanto torneranno utili.

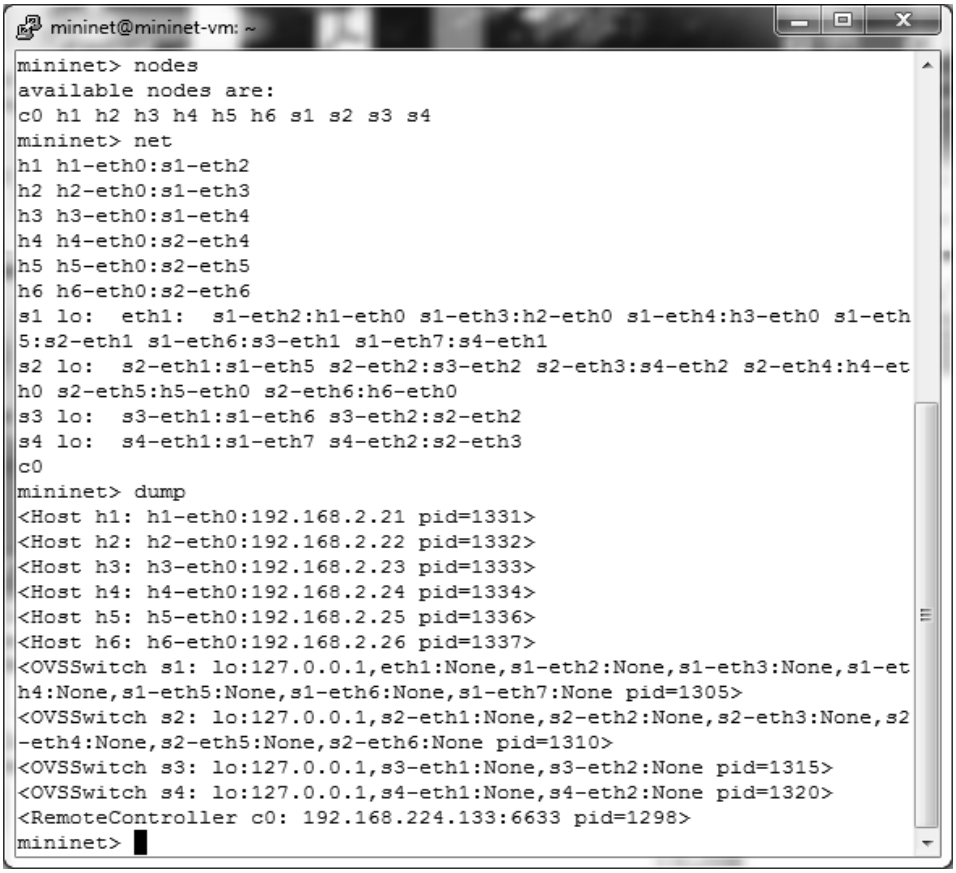


Figura 5.7 Comandi nodes net e dump di mininet

Una volta stabilita la correttezza della rete possiamo finalmente verificarne la funzionalità avviando, per prima cosa, un controller senza il quale i nostri switch virtuali non avrebbero utilità.
Prima di cimentarci nella realizzazione di un controller appositamente studiato si è presa dimestichezza con il software, utilizzando un controller di riferimento già presente in Mininet. Quest'ultimo simula il funzionamento di switch di tipo learning, i più utilizzati attualmente, in grado di replicare ed instradare, verso tutte le porte o solo alcune di esse, i pacchetti entranti discriminandoli in base ai MAC address dei dispositvi collegati.
Grazie all'utilizzo di questi switch (o meglio, grazie al controller che ne simula il comportamento) simulanti il comportamento di switch presenti anche in reti attuali, sarà reso evidente quale vantaggio potrebbe apportare il protocollo Openflow.
Si vuole chiarire che, utilizzando un Controller, già si farà uso dei concetti di SDN e si utilizzerà il protocollo Openflow, ma, in questi primi casi, verranno utilizzati in modo da simulare una rete attuale e quindi senza sfruttarne i veri vantaggi. Praticamente il controller agirà sugli switch semplicemente gestendoli come dei normali hub o learning switch.

5.3.1 Valutazione della rete

Per valutare il funzionamento della rete creata e, quindi, la corretta raggiungibilità di tutti gli host, sono stati utilizzati inizialmente i semplici comandi ping e pingall sfruttando la possibilità di lanciarli direttamente dall'interfaccia CLI di Mininet.
Considerando che il controller da noi avviato nella macchina virtuale con il comando

$ sudo controller –v ptcp:6633

simula il comportamento di switch comuni, ci aspettiamo che eseguendo dei ping tra i diversi host nessuno di questi vada a buon fine. Infatti, come si può notare in figura 5.8, il 100% dei ping fallisce (le X indicano che il ping non è andato a buon fine).

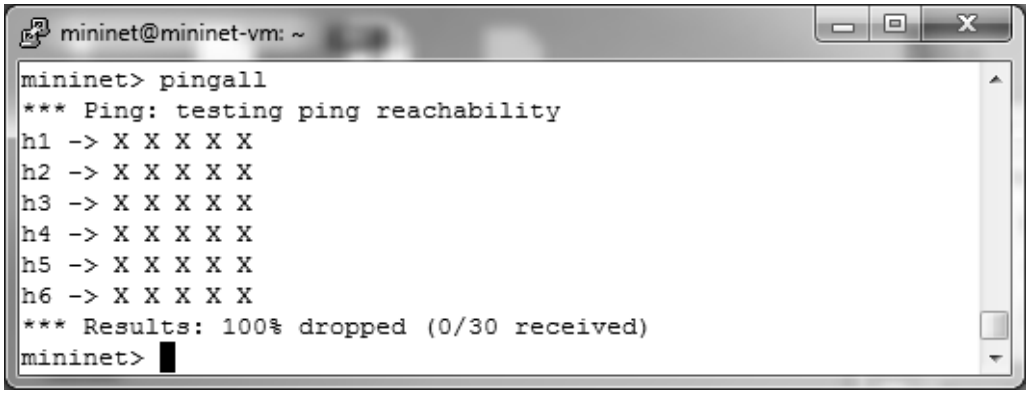


Figura 5.8 Esempio di pingall eseguito dopo aver avviato il controller di riferimento

Vi chiederete il perché di ciò e che senso abbia. La risposta sta nel fatto che la nostra rete è composta da switch connessi a formare uno o più anelli ma la mancanza di un protocollo di controllo causa la moltiplicazione dei pacchetti di broadcast generati dalle ARP request dei comandi ping. Tutto ciò genera all'interno della rete una tempesta di pacchetti (broadcast storm) che ben presto saturerà la rete senza permetterne il corretto funzionamento né tantomeno la raggiungibilità tra host.

Per ovviare a questo problema, come prima soluzione, si potrebbero scollegare gli switch 3 e 4 perdendo però i vantaggi forniti dalla possibilità di avere più connessioni allo stesso tempo ad una velocità più elevata. Ad esempio, considerando uno scambio di dati in contemporanea tra h1 e h4, h2 e h5, h3 e h6, si avrebbe l'opportunità di utilizzare tre strade differenti sfruttando a pieno la banda delle connessioni fisiche disponibili (nel nostro caso a 10Mbit/s). Invece, se si eliminassero i due switch questa possibilità non esisterebbe e le tre trasmissioni sarebbero costrette a dividersi la banda a disposizione avendo come unica via possibile quella tra lo switch 1 e lo switch 2.

Scartata dunque questa ipotesi, come seconda soluzione si è provato ad attivare il protocollo spanning tree (STP) nei singoli switch, utilizzando i comandi per gli switch virtuali (openVswitch) nel seguente modo:

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

Una volta attivato il protocollo si è nuovamente lanciato il comando pingall dal terminale di Mininet e dopo qualche minuto se n'è potuto verificare l'effettivo funzionamento. Infatti, i primi pingall non sono andati a buon fine in quanto il

protocollo STP richiede un po' di tempo per creare un albero della rete e poter essere funzionale. Tuttavia, una volta terminata la sua procedura tutti gli host risultano raggiungibili.

Una problematica in una grande rete potrebbe essere il tempo iniziale necessario alla creazione dell'albero di instradamento.

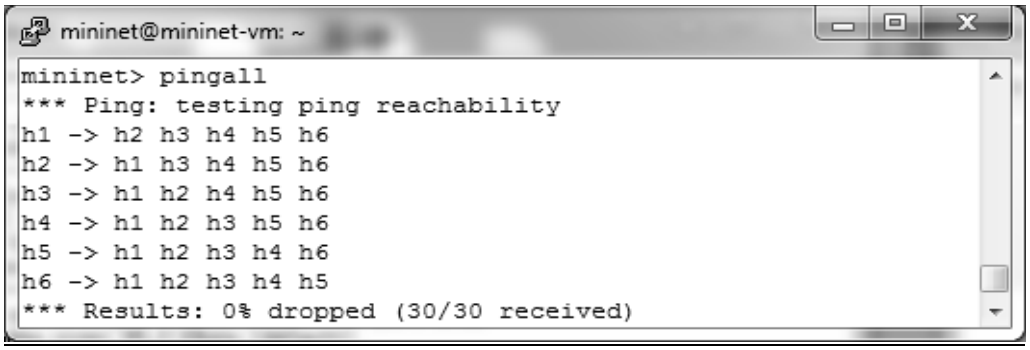


Figura 5.9 pingall lanciato dopo aver attivato il protocollo STP

Il protcocollo STP, mediante un apposito algoritmo, individua la presenza di anelli nella rete e li interrompe bloccando le corrispondenti porte degli switch. In questo secondo caso si ottiene un miglioramento nella robustezza della rete. Pertanto, nel caso in cui uno switch si guastasse, o cadesse un collegamento, il protocollo ricalcolerà i percorsi passando per altre vie. In questo caso non sarebbero risolti il problema delle tre connessioni in contemporanea e quello dell'ottimizzazione della rete e ciò è verificabile utilizzando il comando iperf con il quale andiamo a misurare la velocità di trasferimento di dati tra due host. Per fare ciò, apriamo una finestra xterm per ogni host, dunque sei finestre (terminali) dalle quali possiamo eseguire i comandi.

Facciamo una prima prova lanciando un iperf tra due soli host, uno connesso allo switch 1 e l'altro allo switch 2. Ci attendiamo che la velocità di trasferimento sia all'incirca quella messa a disposizione dalle nostre connessioni fisiche ovvero, come da noi impostato inizialmente, 10Mbits/sec.

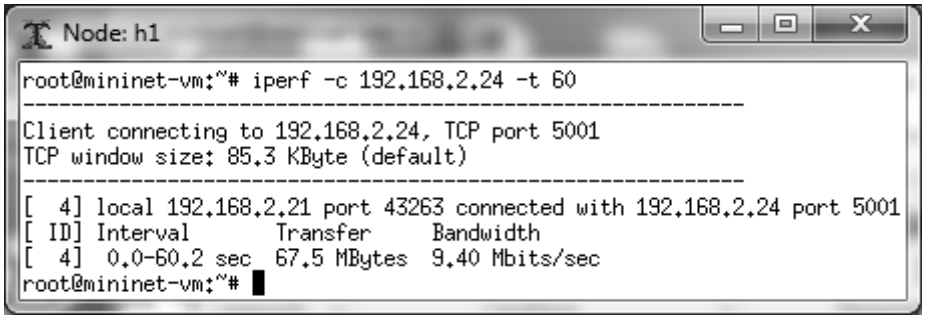


Figura 5.10 iperf lanciato tra due host della rete.

Come ipotizzato, utilizzando iperf per 60 secondi tra h1 e h4 otteniamo 9,40 Mbits/sec, risultato più che accettabile.
Ora però avviamo contemporaneamente tre comandi iperf, ossia da h1 a h4, da h2 a h5 e da h3 a h6, e, quindi, tre trasferimenti di dati.

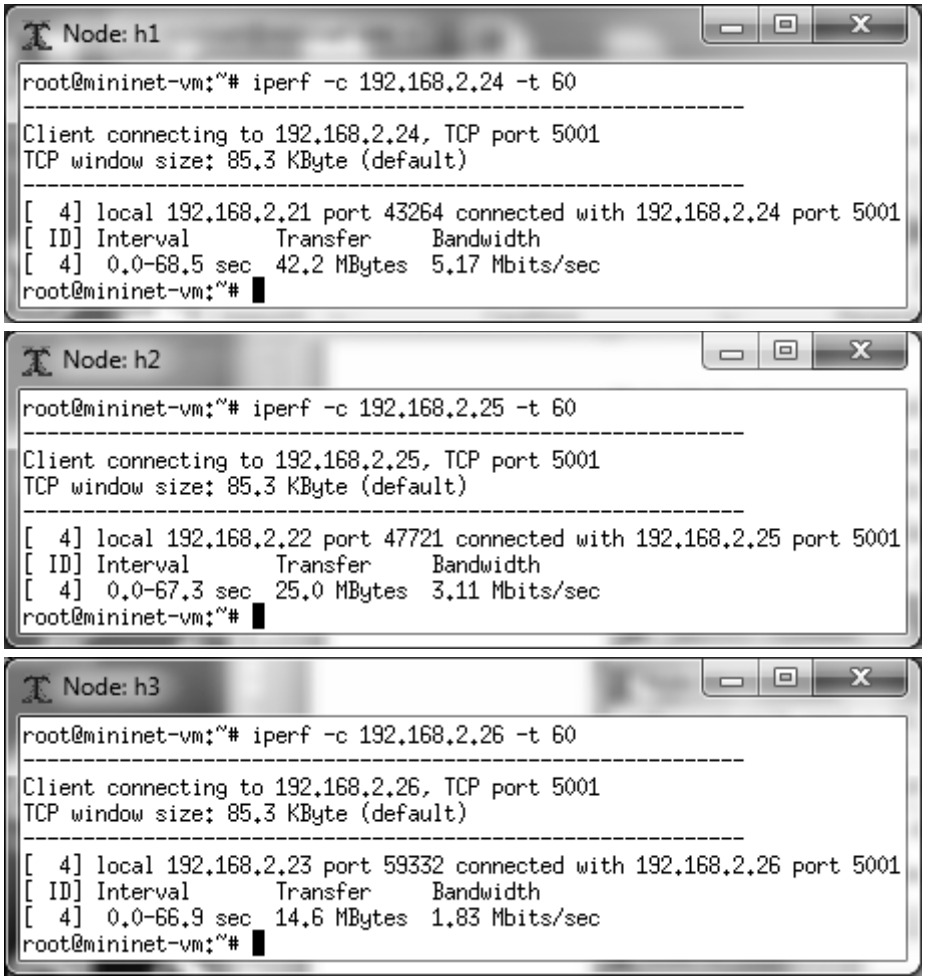


Figura 5.11 iperf lanciati in contemporanea su 3 host differenti.

Notiamo che la velocità delle tre trasmissioni, rispettivamente 5,17, 3,11 e 1,83 Mbits/sec, è nettamente inferiori alla capacità del nostro canale. Ciò dimostra che hanno condiviso la stessa strada, infatti, la loro somma è circa 10 Mbits/sec (nel nostro caso è di poco maggiore in quanto i comandi iperf non sono stati lanciati proprio in contemporanea ma uno di seguito all'altro).
Questo risultato è attendibile in quanto il protocollo STP agisce disabilitando alcune porte per evitare anelli e generando una gerarchia tra gli switch, designandone uno come

root bridge[17]. Possiamo verificare quest'ultimo e, di conseguenza, stabilire com'è stato instradato il traffico utilizzando il comando

ovs-vsctl list Bridge switch

Lanciandolo, specificando gli switch, in una finestra xterm aperta per il controller c0 o, nel nostro caso, da terminale della nostra VM nella quale è stato avviato anche il controller. Le informazioni di nostro interesse, che ci mostra il comando, sono alle voci *status* e *stp_enable* inerenti lo stato e l'attivazione del protocollo STP. Per i nostri switch otteniamo (sono state isolate solo le righe di nostro interesse):

-Switch s1

status : {stp_bridge_id="8000.000c29a95189", stp_designated_root="8000.000c29a95189", stp_root_path_cost="0"}
stp_enable : true

-Switch s2

status : {stp_bridge_id="8000.be47a9081e48", stp_designated_root="8000.000c29a95189", stp_root_path_cost="2"}
stp_enable : true

-Switch s3

status : {stp_bridge_id="8000.0ad5a753214c", stp_designated_root="8000.000c29a95189", stp_root_path_cost="2"}
stp_enable : true

-Switch s4

status : {stp_bridge_id="8000.869712679c4f", stp_designated_root="8000.000c29a95189", stp_root_path_cost="2"}
stp_enable : true

Notiamo che è stato individuato come designed root lo switch S1(il valore id:8000.000c29a95189 corrisponde proprio allo switch 1) ed i costi associati dal protocollo (stp_root_path_cost) a tutti gli altri sono pari a 2; ciò significa che vengono praticamente disabilitate le connessioni tra s3 e s2 e tra s4 e s2 dando la stessa priorità a

[17] Un bridge viene individuato come radice dell'albero coprente ("root bridge"), e una parte dei collegamenti tra bridge disponibili viene messa in standby, portando in stato "BLOCKING" alcune delle porte dei bridge.

tutti gli switch e rendendo, nel nostro caso, praticamente inutili gli switch 3 e 4. In figura 5.12 viene mostrata l'azione del protocollo STP.

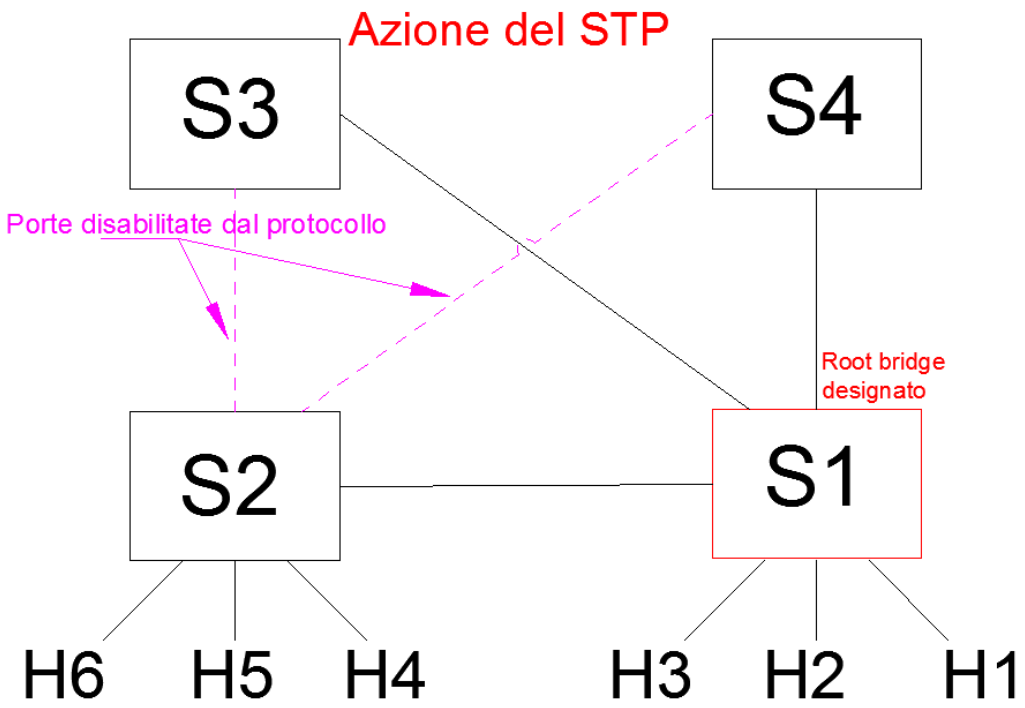


Figura 5.12 azione dello STP protocol

Utilizzando degli switch comuni il problema, dunque, potrebbe essere risolto solamente andando ad impostare e definendo i diversi percorsi manualmente, agendo sulle porte, in maniera però non ottimale, oppure avvalendosi di componenti che non agiscono solo a livello di datalink (collegamento), ma anche a livello di rete come dei router.

Proprio in questo caso entra in gioco il protocollo Openflow il quale, permettendo la comunicazione tra switch e controller, è in grado di semplificarci il lavoro delegando al software di controllo la scelta dei percorsi migliori, per poi comunicarli agli switch istruendoli con apposite tabelle, e lasciando a noi il solo compito di programmarlo nel modo più opportuno.

Si potrebbe agire sul controller da noi già avviato, andando ad inserire, utilizzando determinati comandi, tutte le regole ciascuna con la seguente sintassi digitata dal terminale dove il controller openflow è attivo:

sudo ovs-ofctl add-flow **Sx** ip,nw_dst=**192.168.2..X**,actions=output:**Y**
sudo ovs-ofctl add-flow **Sx** arp,nw_dst=**192.168.2.X**,actions=output:**Y**

dove con Sx s'indica il nome dello switch (s1 s2 s3 o s4), con X il valore corrispondente al relativo host e con Y la porta verso cui trasmettere i pacchetti di tipo ARP o IP destinati all'indirizzo 192.168.2.X. In questo caso le regole verrebbero aggiunte mediante protocollo openflow ma in pratica manualmente nei singoli switch.

Il nostro scopo, però, è di creare un Controller delegandogli il compito di installare le regole nel momento in cui gli giunge una richiesta da parte di uno switch. In sostanza, non appena lo switch riceve un pacchetto in ingresso, lo inoltra al controller mediante il secure channel dedicato al protocollo Openflow e genera un messaggio OpenFlow Protocol (OFP) di tipo PaketIn. Il controller, una volta elaborato il messaggio, manda un messaggio di tipo PacketOut oppure di tipo Flow mod con il quale istruisce lo switch sul da farsi in quel momento, oppure installa sullo switch una regola relativa a quel flusso.

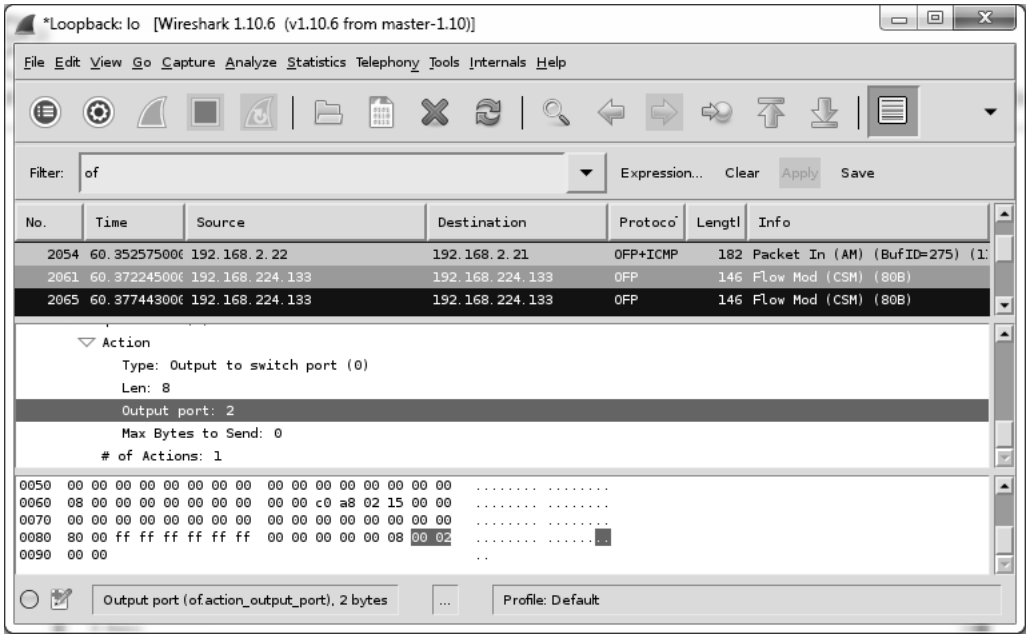


Figura 5.13 visualizzazione mediante Wireshark di pacchetti Openflow

In figura 5.13 viene mostrata una schermata dell'analizzatore di rete wireshark nel quale si mostra uno scambio di pacchetti OFP dopo un ping tra l'host con indirizzo 192.168.2.22 (h2) e quello con indirizzo 192.168.2.21 (h1). Viene evidenziata, pertanto, la regola mandata dal controller allo switch con l'azione del tipo output to switch port e come porta di destinazione la 2 che è in effetti quella relativa al nostro host h1(si può verificare la correttezza della porta visionando la figura 5.4).

Le regole inviate dal controller agli switch sono mantenute in apposite tabelle, FlowTable, consultate dagli switch ogni volta che viene ricevuto un pacchetto, in modo analogo alle attuali tabelle di inoltro impostate manualmente in odierni switch. Ogni regola, come è stato illustrato nel Capitolo inerente al Protocollo OpenFlow, presenta diverse entry mediante le quali è possibile discriminare diversi pacchetti ed inoltrarli in diversi modi, avendo anche l'opportunità di settare delle scadenze alle regole stesse.
Nel prossimo paragrafo verrà mostrata l'implementazione di un Controller OpenFlow mediante il quale si cercherà di ottimizzare la rete, concentrandoci sull'opportunità di consentire contemporaneamente le tre principali connessioni, h1 con h4, h2 con h5 e h3 con h6, e di sfruttare la massima capacità di trasferimento dati tra stesse.

5.4 Implementazione del Controller di Rete

Stabilita la nostra topologia di rete, si è deciso di implementare un controller che permetta, oltre al corretto raggiungimento di tutti i nostri host, anche la possibilità di utilizzare i quattro switch per sfruttare al massimo la capacità dei canali. Come già accennato nel paragrafo precedente, si vuole riuscire ad eseguire le tre connessioni h1 con h4, h2 con h5 e h3 con h6, ottimizzandone la capacità di trasferimento. Il modo migliore per far ciò è quello di far prendere tre strade diverse alle tre connessioni istruendo gli switch in modo tale, ad esempio, da fare passare la prima connessione direttamente dal collegamento tra S1 e S2, la seconda transitando per S3 per poi arrivare ad S2, e la terza transitando per S4 giungendo poi ad S2.
In figura 5.14 viene mostrato come abbiamo deciso di gestire i flussi di dati:
-tutto ciò che è destinato ad h4 e passa per lo switch 1 verrà spedito mediante la porta5 di S1 collegata direttamente alla porta1 di S2 (che poi consegnerà direttamente ad h4), viceversa tutto ciò che transita per S2 ed è destinato ad h1 compirà il percorso inverso.
-tutto ciò che è destinato ad h5 e passa per lo switch 1 verrà spedito mediante la porta6 di S1 quindi passando per S3 e poi giungere alla porta2 di S2 (che poi consegnerà direttamente ad h5), viceversa tutto ciò che transita per S2 ed è destinato ad h2 compirà il percorso inverso.

-tutto ciò che è destinato ad h6 e passa per lo switch 1 verrà spedito mediante la porta6 di S1 quindi passando per S4 e poi giungere alla porta3 di S2 (che poi consegnerà direttamente ad h6), viceversa tutto ciò che transita per S2 ed è destinato ad h3 compirà il percorso inverso.

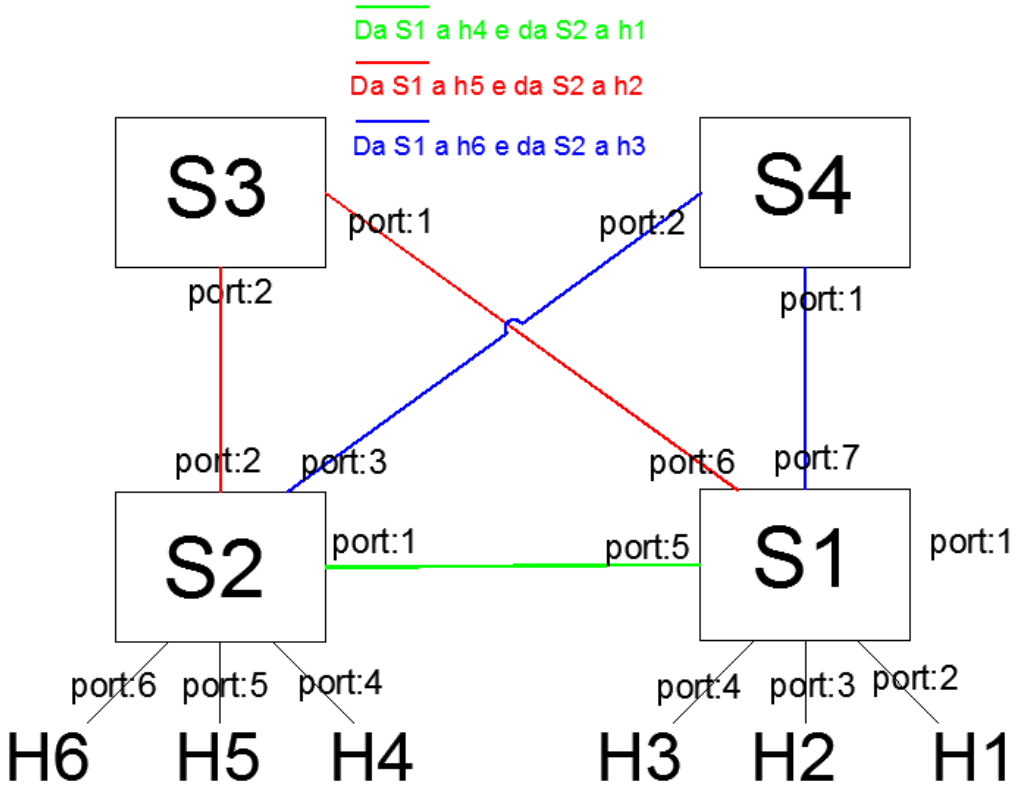


Figura 5.14 3 vie diverse

È stata data, quindi, priorità alle tre connessioni stabilite, ma è possibile raggiungere tutti gli host appartenenti alla rete.

Tutto questo è reso possibile grazie all'utilizzo di un Controller appositamente studiato. La sua realizzazione è stata eseguita mediante il framework Pox ed, anche in questo caso, alle relative API a disposizione, creando uno script di codice Python e sfruttando la struttura del linguaggio.

Dopo aver importato le librerie API a noi necessarie, precedentemente consultate e studiate sul sito internet di Mininet (nella sezione tutorial è presente anche una parte inerente la creazione di Controller POX) e sui diversi tutorial POX online, come prima cosa, si è creata la classe principale

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
```

from pox.lib.util import dpid_to_str
import pox.lib.packet as pkt
import os
import string

class Controller4switch (object):

che crea un oggetto Controller4switch per ogni switch connesso.
Dopo di che sono state richiamate e create alcune funzioni:

- **_init_(self, connection)** : con la quale si tiene traccia della connessione allo switch e come ingressi richiede l'oggetto[18] Controller4switch (self) e un oggetto di tipo connection, questa funzione assocerà l'oggetto di tipo connessione al nostro oggetto principale.

- **install_flow (self, packet_in, match, dst_ip, out_port)**: questa funzione è stata da noi creata e permette di installare le regole nello switch, creandole e poi inviandole in modo automatico. Come ingressi oltre all'oggetto principale, vuole il pacchetto ricevuto, il destinatario cui dovrà essere inviato e la porta relativa, oltre all'oggetto match nel quale sono presenti diverse caratteristiche riguardanti il pacchetto. All'interno di questa funzione viene creato un messaggio di tipo Flow_mod (of.ofp_flowmod) dove vengono inserite le azioni da compiere utilizzando i campi a disposizione

msg = of.ofp_flow_mod(action = of.ofp_action_output(port = out_port),
match=of.ofp_match(dl_type=match.dl_type, nw_dst=dst_ip))

dopo di che viene inviato allo switch nel quale sarà installata la regola nella relativa tabella di flusso.

self.connection.send(msg)

- **_hendle_PaketIn (self, event):** è la funzione principale ed è la vera mente del nostro Controller, gestisce i pacchetti provenienti dagli switch. All'interno di questa funzione si è reso necessario discriminare da quale switch il messaggio di tipo PacketIN provenisse in quanto al nostro controller verranno connessi ben 4 switch e per

[18] In python gli oggetti sono delle istanze contenenti diversi dati (attributi), possono essere usati come parametri delle funzioni.

diversificare la provenienza dei messaggi si è utilizzato il data path ID che identifica i singoli dispositivi.

Il data path ID è composto da 16 cifre esadecimali e Mininet associa ad ogni switch la cifra corrispondente al numero del suo nome (ad esempio S1 associato a 00:00:00:00:00:00:00:01 S2 a 00:00:00:00:00:00:00:02 e cosi via). Si sfrutta, pertanto, questa caratteristica, alcune funzioni disponibili nelle API ed una funzione creata dall'ing. Chiara Contoli per effettuare la discriminazione degli switch e poter quindi inviare le regole correttamente.

Oltre all'oggetto principale la funzione _handle_PacketIn richiede in ingresso l'oggetto event il quale è correlato all'oggetto connection e permette di determinare l'arrivo di un pacchetto, il data path ID di provenienza ed altre opzioni da noi non utilizzate. Sono state create, poi, sempre all'interno della funzione, diverse variabili che saranno sfruttate nella gestione generale:

packet = event.parsed

con i dati del pacchetto

packet_in = event.ofp

contenente l'attuale messaggio packet_in

src_dpid = dpid_to_str(event.dpid)

con il DPID dello switch, convertito in stringa grazie alla funzione dpid_to_str trovata tra le API

int_dpid = convertToInt(src_dpid)

Con il DPID convertito in numero intero grazie alla funzione messa a disposizione dall'ing. Chiara Contoli convertToInt la quale in ingresso vuole una stringa esadecimale e la converte in numero intero.

match = of.ofp_match.from_packet(packet)

contenente le corrispondenze (matching) provenienti dal pacchetto.

Sempre all'interno della funzione handle _paket_in è stato infine utilizzato il costrutto if-else di python più volte, per poter determinare le regole e poi installarle con la funzione, creata in precedenza, install_flow.

Come prima condizione, per le regole della rete, si è imposta quella che i pacchetti fossero di tipo ARP o IP, nel caso fosse verificata vengono create altre 6 variabili dst_ip, dst_ip_str, dst_ip_split, src_ip, src_ip_st,r src_ip_split

```
 if match.dl_type == pkt.ethernet.ARP_TYPE or match.dl_type ==
pkt.ethernet.IP_TYPE :# IP O ARP
    log.debug("IP or ARP packet")
    dst_ip = match.nw_dst
    dst_ip_str = dst_ip.toStr()
    dst_ip_split = string.split(dst_ip_str, '.')

    src_ip = match.nw_src
    src_ip_str = src_ip.toStr()
    src_ip_split = string.split(src_ip_str, '.')
```

le quali indicano rispettivamente l'indirizzo di destinazione, l'indirizzo di destinazione convertito in stringa ed infine un array contenete l'indirizzo di destinazione costruito utilizzando la funzione string.split la quale prende in ingresso una stringa ed un separatore e in uscita restituisce un array, ad esempio:
192.168.2.21 viene convertito in un array cosi formato [192,168,2,21] dove 192 occupa la posizione 0, 168 la 1, 2 la 2 e 21 la 3, questo array a noi servirà a selezionare solo l'ultima cifra dell'indirizzo ip. Le stesse considerazioni possono essere fatte sulle variabili src che poi non sono state utilizzate, ma potrebbero tornare utili per sviluppi futuri.
Come seconda discriminante si è considerato il dpid mediante il quale si è potuta capire la provenienza del pacchetto Packet_IN

```
if int_dpid == n:
```

dove con n si indica il DPID dello switch

Infine, come terza discriminante, nel nostro caso, si è utilizzata l'ultima cifra dell'indirizzo ip con il quale è possibile definire con certezza l'host e quindi inviare la regola.

```
if dst_ip_split[3] == 'm':
  self.install_flow(packet_in, match, dst_ip, p)
```

dove con 'm' si indica l'ultimo campo dell'indirizzo IP di un host e con P la porta che verrà settata nella regola di flusso e alla quale varrà inviato il pacchetto.(per semplicità è stato utilizzato solo l'ultimo campo del indirizzo IP ma in reti con più host sarà necessario ampliare questa discriminante).

Creando praticamente una condizione generale del tipo:

“Se è un pacchetto di tipo IP o ARP e se proviene dallo switch n e se è destinato all’indirizzo 192.168.2.m, allora installa la regola con porta destinazione p”

dove n indica lo switch corrispondente al dpid, m l’ IP dell’host di destinazione e p la porta dove verrà inoltrato il flusso.

Dopo aver verificato la correttezza delle porte dei vari switch, si è potuto completare il codice inserendo, ad una ad una, le istruzioni relative ad ogni destinazione visionabili nel listato completo presente in Appendice (A.3).

Negli switch 3 e 4 è stato sufficiente impostare solamente la regola che impone a tutti i flussi entranti da una porta di uscire dall’altra, in quanto utilizzano solamente 2 porte.

```
elif int_dpid == 3:
  msg = of.ofp_flow_mod()
  msg.match.in_port = 1
  msg.actions.append(of.ofp_action_output(port=2))# IN port:1 Out port:2
  self.connection.send(msg)
  msg = of.ofp_flow_mod()
  msg.match.in_port = 2
  msg.actions.append(of.ofp_action_output(port=1))# IN port:2 Out port:1
  self.connection.send(msg)
```

Ricapitolando se siamo nello switch S1, se il data pathID quindi è 1, e come destinazione abbiamo h1 (*if dst_ip_split[3] == '21':*) il flusso uscirà dalla porta 2, se come destinazione abbiamo h2 uscirà dalla porta 3, h3 dalla 4, h4 dalla 5, h5 dalla 6 ed infine h6 dalla 7 in tutti gli altri casi dalla porta 1; per S2 il funzionamento sarà analogo e se come destinazione abbiamo h4 , h5 o h6, rispettivamente i flussi usciranno dalle porte 4, 5, o 6 , invece per ottenere 3 vie di comunicazione quando i destinatari dei flussi sono h1, h2 e h3 saranno inoltrati dalle porte 1, 2 e 3, mentre in tutti gli altri casi verranno inoltrati verso lo switch 1 passando dalla porta 1. Possiamo verificare tutte le porte di destinazione aiutandoci con la figura 5.14 dove sono mostrate le nostre connessioni principali. Infine per S3 e S4 tutto ciò che entra da una porta esce dall’ altra.

Come accennato, nel caso in cui l’indirizzo di destinazione non appartenga alla nostra rete, verrà gestito da S2, con un “else” finale mandando il pacchetto ad S1 e quindi presso la porta 1,

```
else:
  self.install_flow(packet_in, match, dst_ip, 1)
```

da S1, mandandolo in uscita all’interfaccia aggiuntiva quindi anche qui verso la porta1.

```
else:
  self.install_flow(packet_in, match, dst_ip, 1)
```

Una volta stabilite le regole il nostro controller viene completato aggiungendo allo script le seguenti funzioni

```
def launch ():
  def start_switch (event):
    log.debug("Controlling %s" % (event.connection,))
    Controller4switch(event.connection)
  core.openflow.addListenerByName("ConnectionUp", start_switch)
```

con le quali vengono inizializzati i componenti.
Salvato il file inerente il controller in formato .py, (il nostro sarà Controller4switchflowmod.py) sarà sufficiente avviarlo con la seguente sintassi dalla cartella in cui è presente il framework pox

```
./pox.py log.level --DEBUG misc.Controller4switchflowmod
```

Infine non resta che verificarne la corretta esecuzione.

5.4.1 Test del Controller

Per verificare il corretto funzionamento del controller riprendiamo la nostra topologia, ma questa volta, invece di avviare il Controller di riferimento di Mininet, avviamo il controller da noi creato in modalità debug che ci mostrerà alcuni commenti da noi inseriti nel codice.

```
./pox.py log.level --DEBUG misc.Controller4switchflowmod
```

Se tutto funzionerà correttamente, il controller si metterà in ascolto alla porta 6633 che è anche la porta specificata da noi, quando abbiamo creato la topologia, per il controller Remoto.

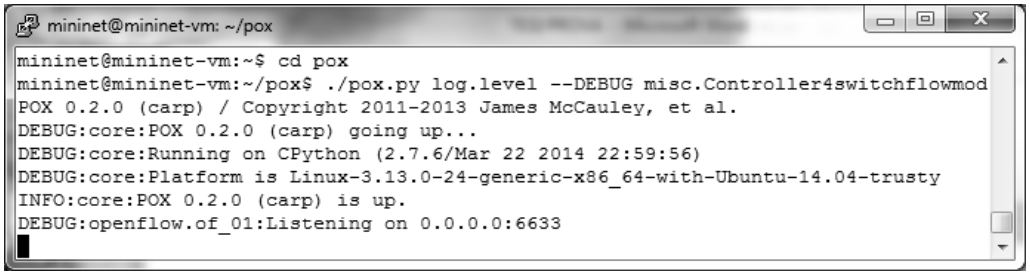


Figura 5.15 Lancio del Controller

Una volta avviata la rete, il Controller inizierà a comunicare con i quattro switch connettendosi a loro e, avvenuta la connessione, possiamo andare a verificare le tabelle di flusso degli switch aprendo una finestra xterm per il nodo c0 oppure dal terminale della VM utilizzando il comando:

ovs-ofctl dump-flows switch

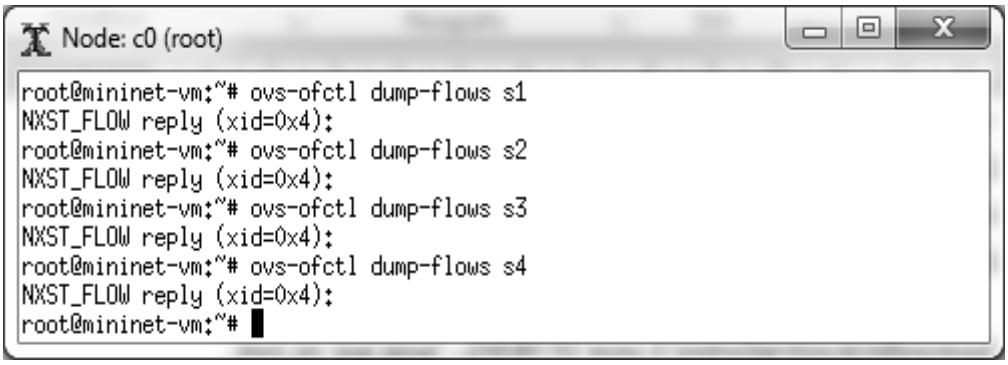


Figura 5.16 visualizzazione regole di flusso

Si può notare che non è presente nessuna regola.

Ora per verificare la corretta installazione di regole, proviamo ad eseguire un ping tra due host, ad esempio h3 e h6. Ci aspettiamo che vengano installate nello switch 1, nello switch 2 ed anche nello switch 4, in quanto è una delle tre trasmissioni che abbiamo voluto impostare inizialmente (si puo verificare il flusso con l'aiuto della figura 5.14 nella quale è raffigurato in colore Blu).

Dopo aver eseguito il ping, il quale va a buon fine solo dopo qualche istante, rieseguiamo i comandi per verificare le tabelle ottenendo i dati seguenti (Figura 5.17 pagina seguente):

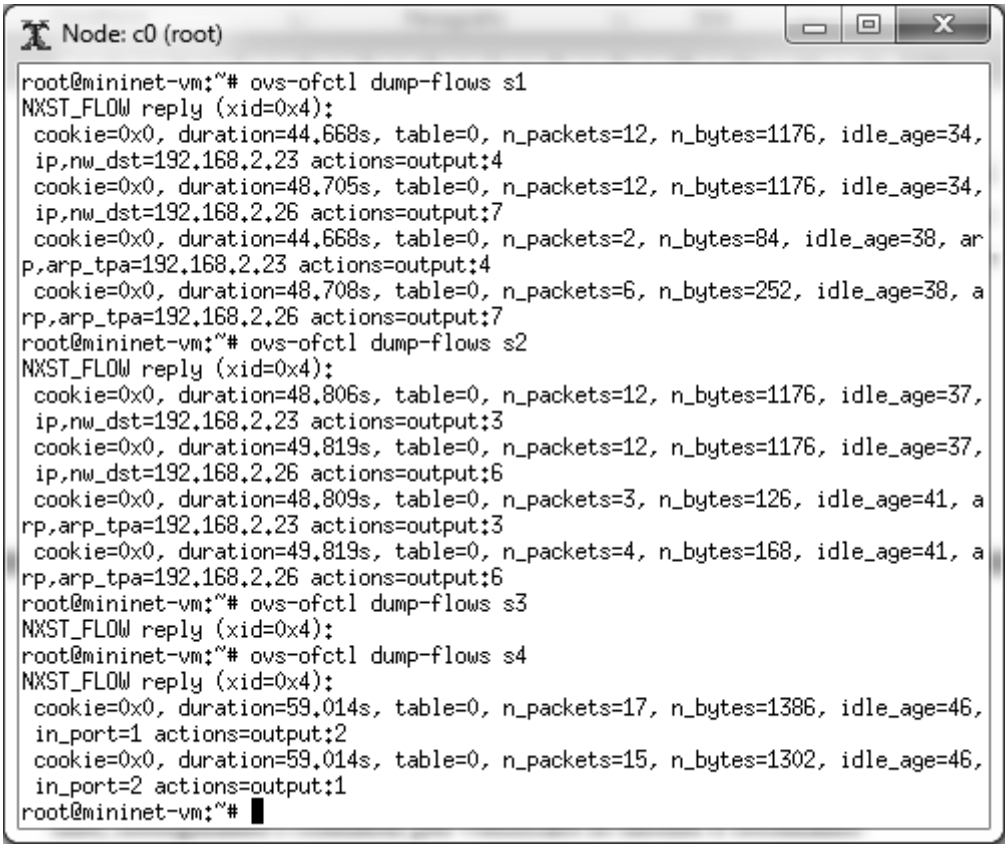


Figura 5.17 visualizzazione regole di flusso dopo l'esecuzione di alcuni ping

Come atteso possiamo notare il corretto inserimento delle regole in tutti gli switch eccetto il terzo. Tali regole impongono, come da noi voluto per la connessione tra h3 e h6, di passare per lo switch 4.

Si può anche verificare l'esattezza delle regole inserite notando che nello switch 1, come nello switch 2, sono state inserite per l'indirizzo di destinazione di H3 e per l'indirizzo di destinazione H6 (ping richiede anche una risposta), mentre nello switch 4 sono state inserite le regole che inoltrano dalla porta 1 alla 2 e viceversa.

Il controller, comunicando con gli switch, installa le regole automaticamente e gestisce il flusso generato dal comando ping in maniera appropriata.

Dopo aver verificato la raggiungibilità di tutti gli host e, di conseguenza, anche la correttezza di tutte le regole e del funzionamento del controller creato, si è voluto anche dimostrare che, con il suo utilizzo, si riesce ad ottimizzare l'utilizzo delle tre connessioni da noi imposte all'inizio per migliorare la rete (tra h1---h4, h2---h5 e h3---h6).

Utilizzando nuovamente come test il comando iperf lanciato dalle finestre xterm relative agli host:

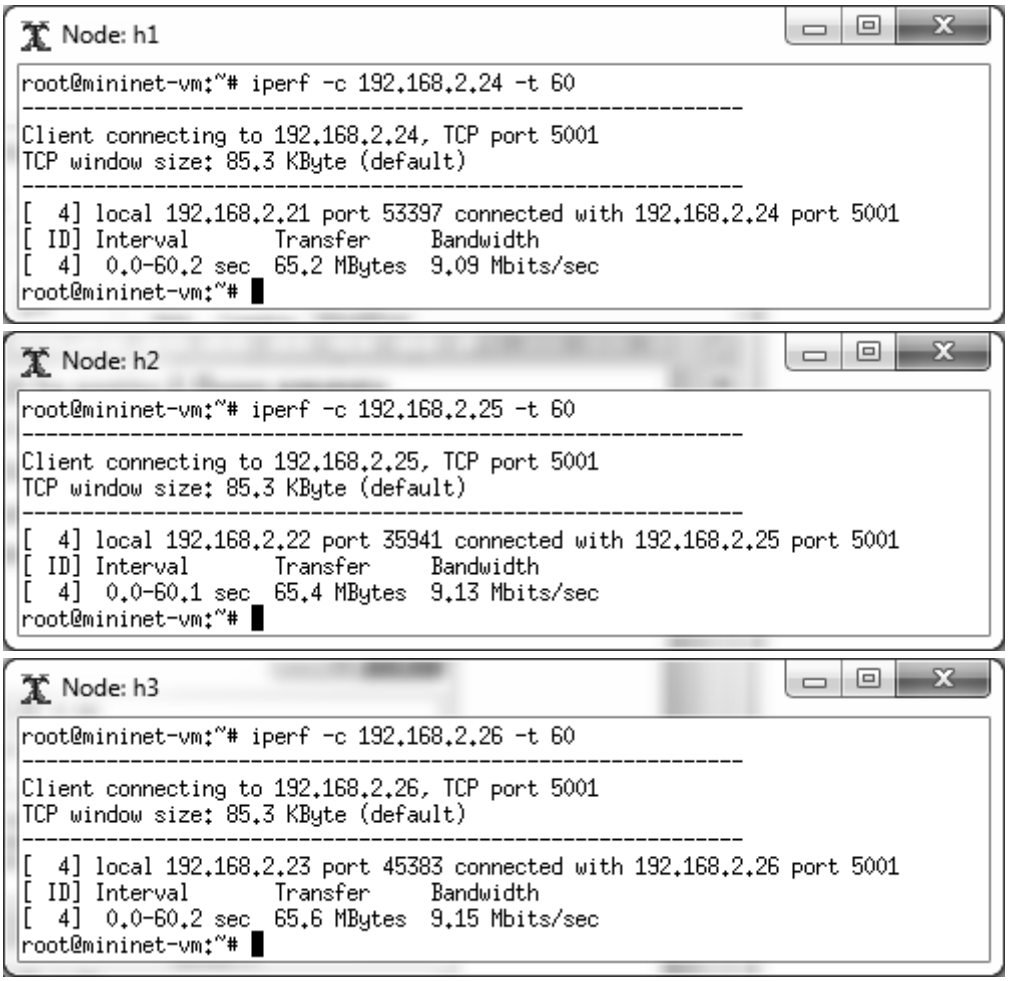


Figura 5.18 iperf lanciati in contemporanea su tre host differenti dopo aver avviato il nostro Controller

Notiamo un notevole miglioramento rispetto allo stesso test effettuato attivando il protocollo spanning tree; infatti, tutte e tre le connessioni viaggiano ad una velocità del tutto accettabile maggiore di 9 Mbits/sec, sfruttando tutti gli switch e le connessioni a nostra disposizione.

Dato che il Controller esegue il suo compito a dovere e la rete reagisce in maniera opportuna grazie al protocollo OpenFlow, si vuole ora tentare di espandere quest'ultima.

5.5 Rete Distribuita

Presa dimestichezza con la rete minimale, si è voluto provare ad espanderla utilizzando una seconda VM. Per fare ciò si è installata nel programma di virtualizzazione una seconda VM contenete anche essa Mininet e vi si è creata una seconda topologia identica alla prima ma con altri nodi, attribuendo agli host (da h7 a h12) indirizzi dal 192.168.2.31 al 192.168.2.36 e numerando gli switch da S5 a S8; il tutto come schematizzato in figura 5.19 e in pratica simmetricamente alla VM1.

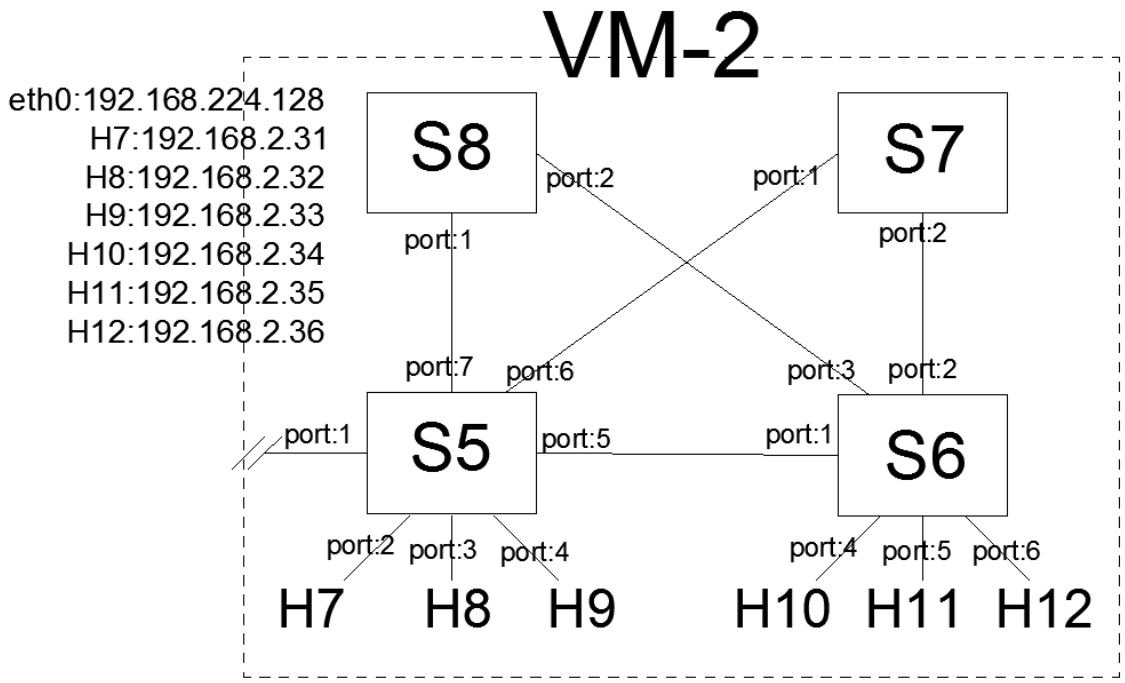


Figura 5.19 Schema della rete sulla seconda VM

Ci troviamo, ora, con due reti funzionanti su due macchine separate e la nostra idea è quella di connetterle creandone una unica. Il nostro scopo fin dall'inizio era quello di distribuire la rete su più VM e, nella creazione della rete minimale, avevamo inserito questa riga di codice

Intf('eth1',node = s1)

con la quale si era aggiunta un interfaccia allo switch 1 per futuri sviluppi.
Per connettere le 2 reti dobbiamo fare in modo che questa interfaccia possa essere collegata all'interfaccia aggiunta simmetricamente ad S5 anche nella seconda topologia.

Affinché ciò sia possibile, creeremo una seconda connessione tra le due macchine virtuali utilizzando il nostro software di virtualizzazione. La prima connessione, che è anche quella utilizzata dal protocollo ssh, collega le due VM e l'host Windows 7 in una LAN 192.168.224.0/24 mediante le loro porte eth0, mentre la seconda connessione collegherà le 2 VM nella LAN 10.0.0.0/24 mediante le porte eth1. Il software di virtualizzazione, infatti, consente di aggiungere nuove connessioni tra le macchine virtuali permettendone la comunicazione tra loro o anche, volendo, con la rete esterna.
Per far si che anche la rete implementata possa connettersi attraverso l'interfaccia eth1, bisogna inserire le seguenti righe di codice nello script della topologia di rete

```
s1.cmd('ovs-vsctl add-port s1 eth1')
s1.cmd('ifconfig s1 10.0.0.2')
```

Con queste viene aggiunta una porta allo switch s1 e gli viene assegnato un indirizzo ip, in modo tale da consentire la comunicazione con la seconda Rete nella quale dovranno essere inserite le stesse righe di codice, ovviamente cambiando nome allo switch ed assegnandogli un altro indirizzo appartenente alla medesima rete. Questi comandi si potrebbero digitare anche direttamente da terminale in quanto sono comandi relativi agli openVswitch e non hanno a che fare propriamente con la topologia di rete. Tuttavia, onde evitare di doverli ogni volta settare, sono stati inseriti nello script della nostra topologia.
In concreto la nostra rete una volta aggiunte le righe di codice, sarà capace di comunicare con l'esterno attraverso l'interfaccia eth1 della nostra VM consentendo un'espansione della Rete stessa.
La struttura della rete complessiva tra le macchine virtuali, una volta modificati gli script, sarà come quella mostrata in figura 5.20.
Le due VM possono comunicare attraverso le due LAN 192.168.224.0/24 e 10.0.0.0/24, mentre le due reti create sulle macchine possono comunicare grazie alle interfacce eth1 come se fossero direttamente collegati gli switch 1 e 5, divenendo un'unica rete.

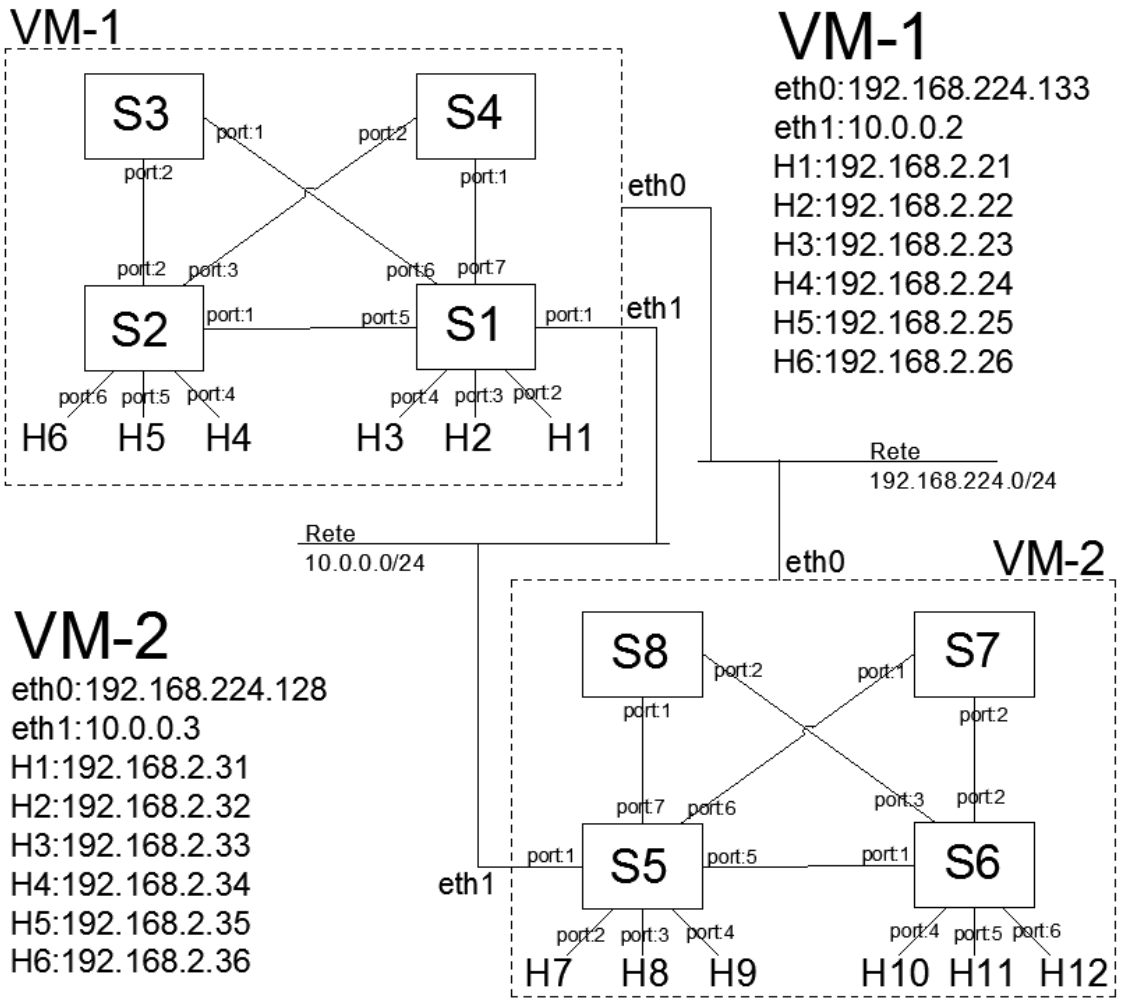


Figura 5.20 schema della rete distribuita su 2 VM

L'intera rete può essere controllata mediante un unico controller al quale le singole Mininet possono collegarsi tramite interfaccia eth0, senza quindi interferire con i dati scambiati all'interno della rete che comunica attraverso eth1. Sarà sufficiente espandere le funzionalità del Controller precedentemente creato aggiungendo delle regole per gli switch 5, 6, 7 e 8, le quali saranno identiche a quelle utilizzate rispettivamente per gli switch 1, 2, 3 e 4, a differenza degli indirizzi degli host, che nella prima rete vanno da 21 a 26 e nella seconda rete da 31 a 36, e dei numeri degli switch e degli host che sarà sufficiente sostituire. Anche nella seconda VM quindi le connessioni tra h6 e h10, h7 e h11 ed h8 h12 verranno gestite prendendo 3 strade differenti, per chiarezza si può visionare la figura 5.21 a pagina seguente.

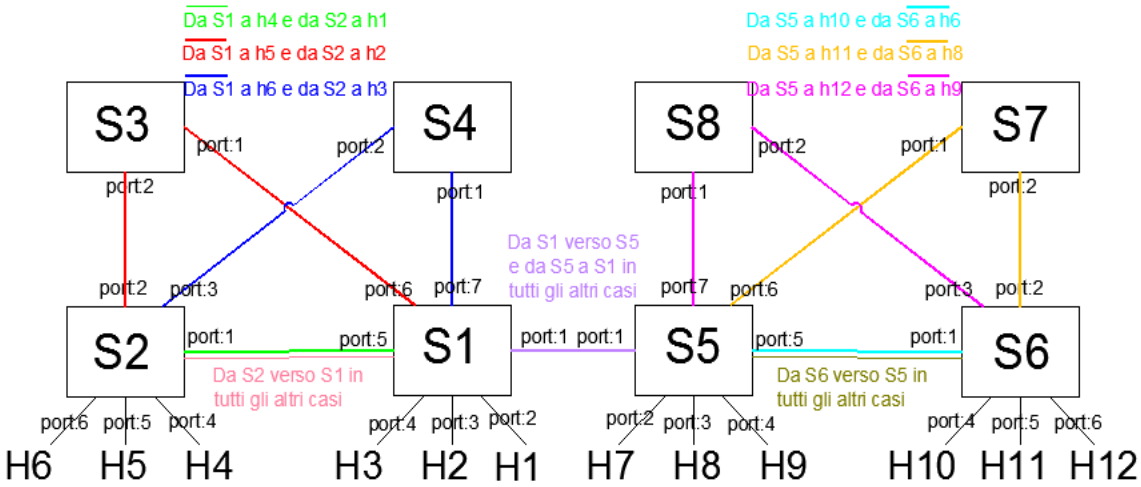


Figura 5.21 Rete distribuita e gestione dei flussi da parte degli switch delle 2 reti

Per la gestione dell'intera rete si sarebbero potuti anche utilizzare due Controller, uno per ogni VM, ottenendo lo stesso risultato, ma si è voluta dimostrare la possibilità di dislocare il controller ovunque all'interno della rete o anche esternamente ad essa. In realtà, in reti complesse la gestione OpenFlow viene affidata a più di un controller affinchè non ci siano problemi dovuti al malfunzionamento di uno degli apparati sui quali il controller stesso viene lanciato; pertanto, il protocollo contempla anche l'utilizzo di più Controller, ma questa trattazione esula i nostri scopi.

In appendice si può trovare il listato (parte aggiuntiva rispetto al Controller per 4 switch) del controller dell'intera rete Controller8switchflowmod.py notando che è un'estensione di quello per quattro Switch con l'aggiunta delle regole per gli altri.

5.5.1 Test di base sulla Rete distribuita

Per verificare l'effettiva raggiungibilità tra gli host appartenenti alle due reti, e, quindi, la comunicazione tra le due mediante interfaccia eth1, si è innanzi tutto avviato il Controller e, dopo aver lanciato le due topologie nelle due VM, si è notata l'effettiva connessione tra gli switch e quest'ultimo.

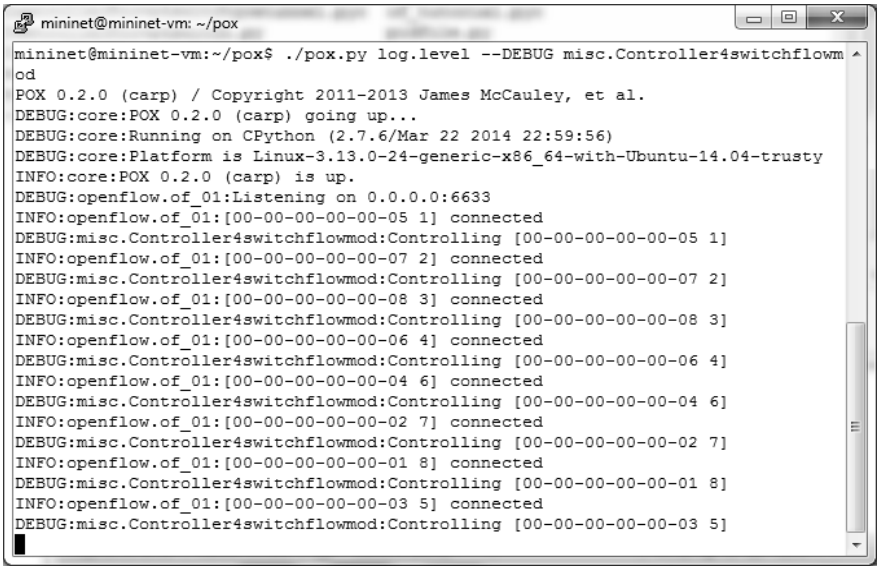


Figura 5.22 Avvio controller per 8 switch.

Nuovamente, utilizzando il comando pingall in entrambe le Mininet, si è controllata la raggiungibilità di tutti gli host all'interno delle singole VM; infatti, con il comando pingall, le mininet inviano dei ping solamente tra gli host creati nella loro topologia, quindi per la rete VM1: h1 h2 h3 h4 h5 h6 e per la rete VM2: h7 h8 h9 h10 h11 h12. (pingall è un comando Mininet)

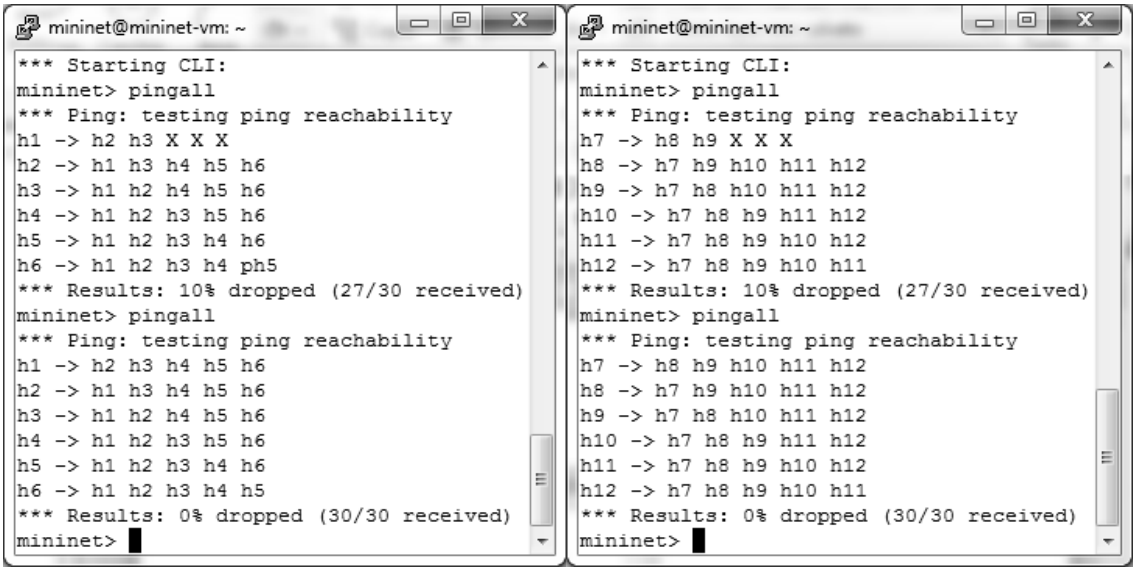


Figura 5.23 Comando pingall eseguito sulle 2 Mininet

Si può notare come, un primo tentativo di pingall, abbia fallito in qualche connessione tra host (nel nostro caso sono 3 le connessioni fallite in entrambe le reti). Questo è semplicemente dovuto al fatto che gli switch devono comunicare con il controller prima

di avere a disposizione le regole corrette; infatti il secondo tentativo è andato a buon fine immediatamente, avendo gli switch già appreso e installato gran parte delle regole.
Una volta stabilito che le due reti separatamente funzionano correttamente, sono stati effettuati alcuni comandi ping tra host delle due macchine; ad esempio, da xterm di h1(Rete VM1) si può provare a lanciare un ping verso h12 (Rete VM2) verificando o meno se le due reti sono comunicanti (figura 5.24).

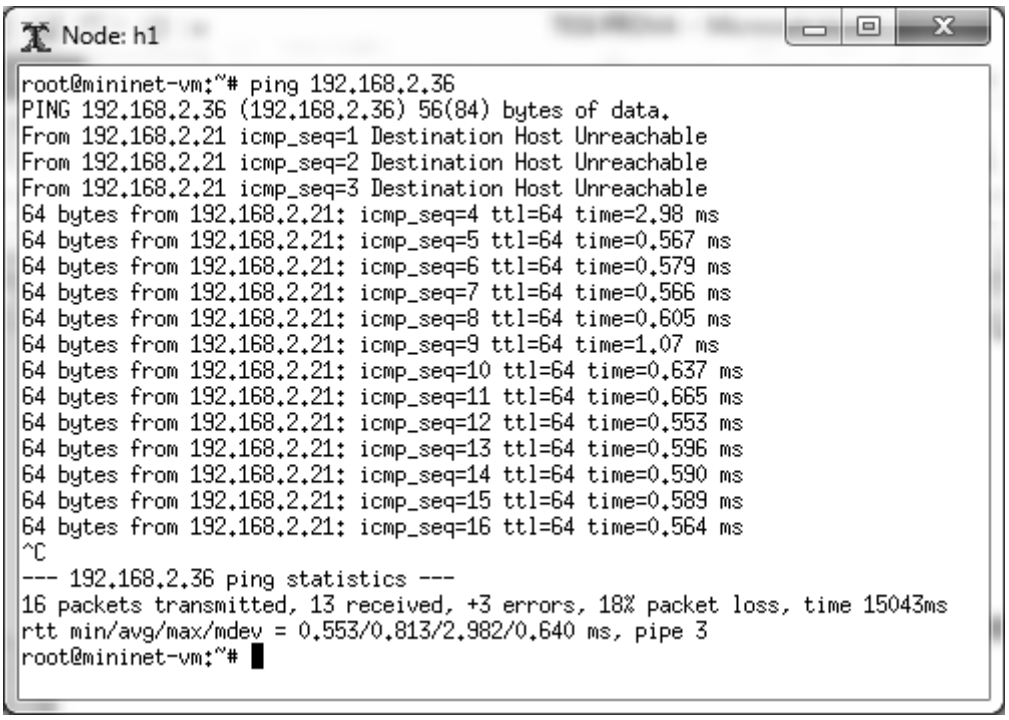


Figura 5.24 ping eseguito da h1(VM1) ad h2(VM2)

Notiamo che, anche in questo caso, i primi tre ping non sono andati a buon fine perché, come spiegato anche in precedenza, le regole negli switch ancora non erano presenti, ma successivamente la destinazione è stata raggiunta. Si può infine notare come, ripetendo nuovamente il comando, vada subito a buon fine (figura 5.25) dato che le regole rimangono negli switch una volta installate a meno di eventuali opzioni.
Si è proceduto a verificare il corretto raggiungimento di tutti gli host ed a constatare la correttezza del nostro controller, oltre a quella dell'intera rete.
Stabilita la raggiungibilità di tutti gli host, si è voluta eseguire una valutazione più approfondita della nostra rete andando ad analizzare i flussi tra le due reti per assicurarci il corretto inoltro dei dati.

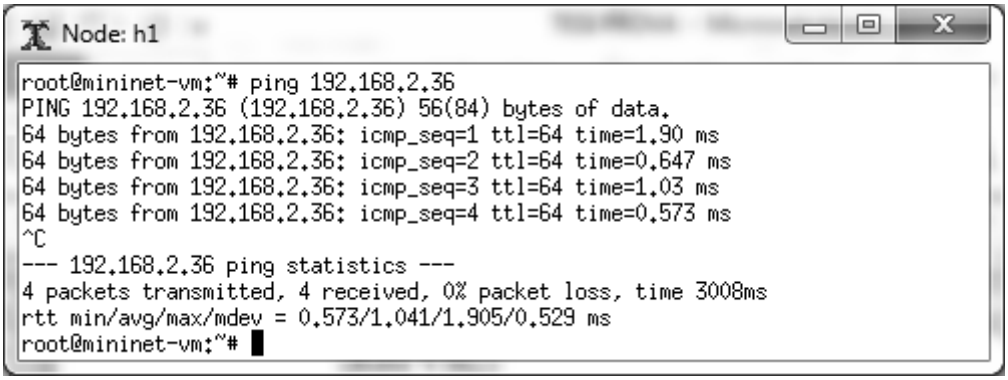


Figura 5.25 nuovo ping eseguito da h1(VM1) ad h2(VM2)

5.6 Valutazione della Rete Distribuita.

Eseguiti i semplici test di base, utili a constatare la raggiungibilità di tutti gli host appartenenti alla rete implementata, si è voluto eseguire alcuni test con i quali si mette in evidenza il corretto instradamento dei flussi di dati con l'ausilio di alcuni grafici inerenti ai throughput delle varie porte appartenenti ai diversi switch.
Per creare grafici appropriati si sono utilizzati i dati forniti dal comando lanciato da terminale della VM (oppure da terminale del controller c0):

ovs-ofctl dump-ports switch

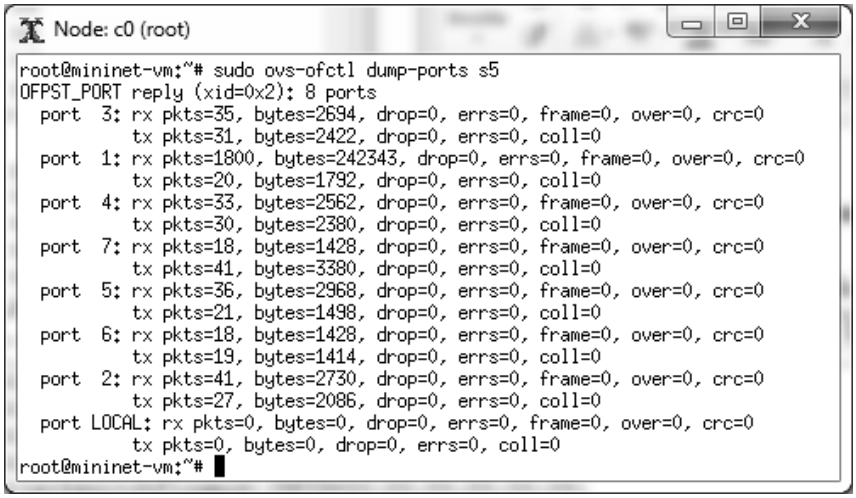


Figura 5.26 comando ovs-ofctl dump-ports lanciato per lo switch s5

in uscita mostra delle statistiche relative alle porte dei dispositivi di rete indicati (in figura 5.24, come esempio sono mostrati quelli relativi allo switch S5) come il numero

di pacchetti transitati in ingresso ed in uscita, i dati transitati sia in ingresso che in uscita ed altre informazioni.
Si è deciso di graficare i throughput di tutte le porte sia in entranta che in uscita. Il throughput è la quantità di dati trasmessi in un unità di tempo che indica la capacità di trasmissione effettivamente utilizzata; possiamo calcolarlo, ad esempio, lanciando 2 volte il commando sopra menzionato a distanza di n secondi e utilizzare i campi byte ricevuti e inviati per eseguirne le differenze. In poche parole, si utilizza la formula

Mbit/s= (B2-B1)*8/(t2-t1)

dove B2 indica i byte visualizzati dopo il secondo lancio del comando, B1 quelli indicati al primo lancio e t2-t1 è l'intervallo di tempo trascorso tra i 2 comandi, la differenza tra byte viene moltiplicata per 8 in quanto i troughput sono misurati solitamente in bit/s e non in byte/s.
Questa operazione dovrebbe essere eseguita per ogni switch e per ciascuna porta.
Per creare un grafico adeguato, che ci mostri l'andamento in un periodo di tempo ragionevole, avremmo bisogno di ripetere il comando più volte andando ogni volta a selezionare i campi di nostro interesse e rendendo il tutto un'impresa ardua. Per aiutarci ad ottenere le catture dei dati in maniera automatica sono, quindi, stati creati alcuni script shell eseguibili.

5.6.1 Implementazione Script per per la cattura dei dati di interesse.

Dopo avere stabilito i campi del comando *ovs-ofctl dump-ports* a noi necessari e aver studiato il funzionamento dei comandi grep e cut di linux, si sono potuti creare degli script eseguibili all'interno dei quali è stato richiamato anche un semplice programma scritto in C, gettimestamp.c (Appendice), il quale una volta eseguito stampa in uscita il tempo con una precisione del microsecondo. Il file scritto in C dovrà essere presente nella stessa cartella del file shell (.sh) che andremo a implementare.
Nel nostro file quindi verranno create delle variabili all' interno di un istruzione while che genera un loop continuo

```
while true
do
        sw1p1rx=`sudo ovs-ofctl dump-ports s1|grep -A 1 1:|grep rx|cut -d= -f3|cut -d, -f1`
```

nelle quali verrà inserito il risultato del commando *ovs-ofctl dump-ports* opportunamente tagliato e filtrato con i due comandi cut e grep.

Aggiunte tutte le variabili (nel comando sopra se ne è mostrata solo una: sw1p1rx ma saranno 2 per ogni porta dello switch) di nostro interesse, ovvero 2 per ogni porta contenenti byte trasmessi e ricevuti, verranno stampate a video grazie al comando echo insieme al tempo t ricavato dalla funzione gettimestamp.

```
        t=`./gettimestamp`
        echo –n "t"
        echo –n "sw1p1rx"
```

La serie di comandi echo, di cui sopra, stampa a video t seguito dalla variabile sw1p1rx (grazie all' opzione –n vengono stampate, senza andare a capo, a seguire sulla stessa riga). Infine l'utilizzo del comando *sleep*

```
        sleep $period
```

e di una variabile creata inizialmente

```
        period=n
```

permettono ai comandi elencati nel nostro script file di essere eseguiti ogni n secondi, fintanto che non lo fermeremo con ctrl-c.

Lanciato il nostro Controller e avviate le nostre reti è quindi possibile eseguire anche i nostri script file, precedentemente opportunamente salvati, per ricavare i dati ed iniziare le catture. L'esecuzione di uno script shell è molto semplice, basta lanciarlo ponendogli davanti il comando sh, ed usare i permessi di amministratore.

```
sudo sh scriptdaeseguire.sh
```

Ad esecuzione avvenuta nel nostro terminale possiamo notare un susseguirsi, ad intervalli regolari (o quasi, noteremo poi il perchè) di n secondi circa, di tutti i dati filtrati a partire dal tempo con, a seguire sulla stessa riga, i byte ricevuti e trasmessi di tutte le porte. Nel caso dello switch 1 ad esempio i dati sono 15 per ogni riga stampata a video, mentre per gli altri switch la mole di dati sarà minore (S1 come S5 ha 7 porte e per ogni porta ricaviamo 2 dati con in più anche il tempo).

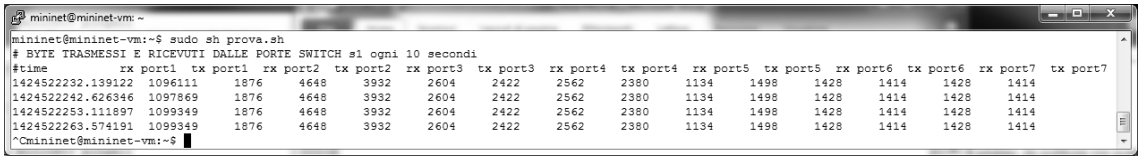


Figura 5.27 esecusione dello script per lo switch s1

Verificato l'effettivo funzionamento dello script, si è proceduto creandone altri 7 uno per ogni switch.

I dati ricavati in uscita, però, non ci consentono di determinare direttamente i grafici dei flussi relativi alle porte ma richiedono una elaborazione, che seppur semplice, deve essere eseguita su una grande quantità di dati.

5.6.2 Elaborazione dei dati

Una volta eseguiti tutti gli script in contemporanea, o quasi (sono stati eseguiti uno di seguito all' altro), ci troviamo a disposizione un enorme quantità di dati. Come esempio, solo per lo switch 1, effettuando una cattura di dati ogni 5 secondi per 5 minuti, avendo 2 dati per porta e 7 porte, ci troveremmo con circa 900 numeri da dover elaborare, considerando che gli switch a nostra disposizione sono 8, effettuare i calcoli manualmente diventerebbe complicato perlomeno in termini di tempo. Per ovviare al problema i dati in uscita visualizzati sul nostro terminale sono stati copiati su un foglio di calcolo excel. L'importazione è stata resa semplice grazie alla divisione dei dati da noi impostata in maniera ordinata: in righe (in quanto ogni cattura genera una riga) e in colonne (in quanto i dati sono divisi da spaziature). Una volta copiati i dati è sufficiente utilizzare delle semplici funzioni per elaborarli, generando nuove colonne con i risultati da noi desiderati, ossia i bits/sec di ogni porta e le differenze tra tempi successivi, per poter infine generare dei grafici (con ascissa il tempo, e ordinata i bit/s).

Per essere più chiari facciamo un esempio:

Consideriamo 2 catture consecutive di sole 2 porte (per semplicità) che nel nostro terminale verranno visualizzate come segue (potrebbe essere il caso degli switch secondari, s3, s4,s7,s8 che possiedono solo 2 porte):

Time	port1 rx	port1 tx	port2 rx	port2 tx
1424522232.139122	0	0	0	0
1424522242.626346	15000	2500	26000	1520

Dove con time viene visualizzato il tempo corrente preso dal PC mediante la funzione creata, port rx i byte ricevuti port tx quelli trasmessi ricavati dal comando citato all' inizio del paragrafo 5.6.1. Sarà quindi sufficiente effettuare i calcoli relativi alla formula relativa il calcolo del throghputh:
la differenza tra i tempi delle 2 righe, la differenza tra i byte delle 2 righe, dopo di che moltiplicare la differenza di byte per 8, trovando i bit, ed infine dividere per la differenza di tempo, trovando il dato di nostro interesse.

T2-T1=1424522232.139122-1424522242.626346 = 10,487224
notiamo che pur avendo impostato nel nostro script come intervallo di cattura n=10 secondi la differenza tra le 2 catture e maggiore di 10 secondi. Per questo è stata utilizzato il file gettimestamp e non è stato semplicemente utilizzato 10 come dato, ottenendo maggiore precisione.

B2-B1 = 15000 -0 = 15000
Dati ricevuti dalla porta 1 in byte

(B2-B1)*8=150000
Dati ricevuti dalla porta 1 in bit (1 byte = 8bit)

Trovando infine il throughput in ingresso sulla porta 1:
(B2-B1)*8/(T2-T1)=150000/10,487224=14303 bit/s circa 14,3 kb/s.

Le stesse operazioni possono essere eseguite per i byte trasmessi e per tutte le altre porte. Ottenuti tutti i valori e disposti in colonne ordinate è possibile graficarli semplicemente utilizzando le funzioni grafighe del nostro foglio di calcolo Excel.

5.6.3 Visualizzazione Grafica dei dati elaborati

Dopo aver avviato le reti e tutti gli script di cattura relativi ai diversi switch, per poter creare dei grafici inerenti i flussi di dati passanti attraverso le porte, è richiesto innanzi tutto un traffico di dati. Per generarlo è stato utilizzato il comando iperf che, infatti, genera traffico dati di tipo TCP, come potrebbe essere quello di una connessione internet e sul quale si basano gran parte delle applicazioni. Il comando verrà eseguito più volte, lanciandolo da host diversi e verso diversi host, in modo da simulare uno scambio di informazioni attraverso la rete.

Le catture di dati sono state mantenute per circa 7 minuti, con un' intervallo di 10 secondi, nei quali abbiamo utilizzato diverse volte il comando iperf per tempi di 120, 30 e 60 secondi (utilizzando l'opzione –t). Terminata la cattura si è passati all'elaborazione dei valori trasportandoli in excel, grazie al quale è stato anche possibile visualizzarne i risultati grafici.

L'esperimento è stato svolto come segue: abbiamo lanciato in tempi casuali, dopo un pingall iniziale in entrambe le reti, 2 volte un iperf dall' host h10(client) all' host h2 (server) della durata di 120 secondi, 2 volte iperf della durata di 30 secondi tra h1 e h6 ed infine 2 volte iperf della durata di 60 secondi tra h5 e h9 tutto ciò nel arco dei 7 minuti.

Utilizzando i comandi sopra citati, come conseguenza, ci attendiamo un instradamento dei flussi del tipo raffigurato in figura 5.26 dove notiamo come dovrebbero transitare i dati in base alle regole da noi impostate:

-Linea rossa: Iperf tra h10 e l'host h2 genera un invio di dati da h10 ad h2 quindi per prima cosa i dati dovranno entrare dalla porta 4 dello switch6 per poi uscire dalla 1 dello stesso switch, transitare per lo switch 5 entrando dalla porta5 e uscendo dalla porta1, raggiungere lo switch 4 entrando dalla porta1 e giungere mediante porta 3 all' host 2.

-Linea verde: l'iperf tra h1 e h6, genera un invio di dati a partire da h1 verso h6, quindi lo switch1 nel quale entrano dalla porta2 instraderà verso la porta1 dello switch4, per poi uscire dalla porta2 dello stesso e giungere mediante porta3 allo switch2 che provederà alla consegna diretta ad H6 (porta6).

-Linea azzurra: iperf tra h5 e h9 , genera un invio di dati a partire da h5 quindi porta 5 dello switch 2, per poi passare in uscita dalla porta 1, e giungere allo switch 1 presso la porta 5, per arrivare allo switch 5 transitando in uscita dalla porta1 di S1 e in ingresso ad S5 dalla sua porta1 ed essere infine consegnato tramite porta dedicata all' host 9 (porta4).

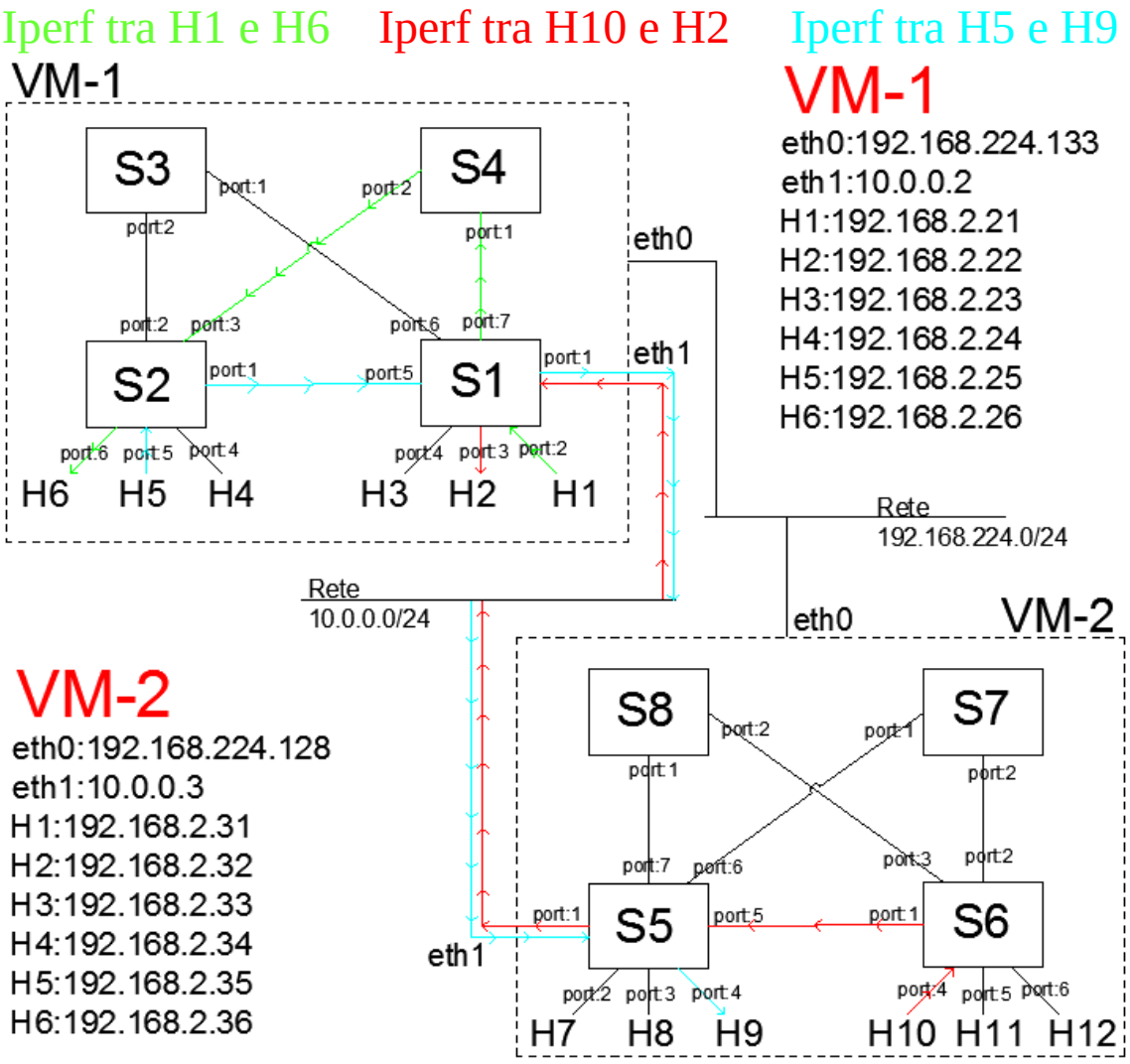
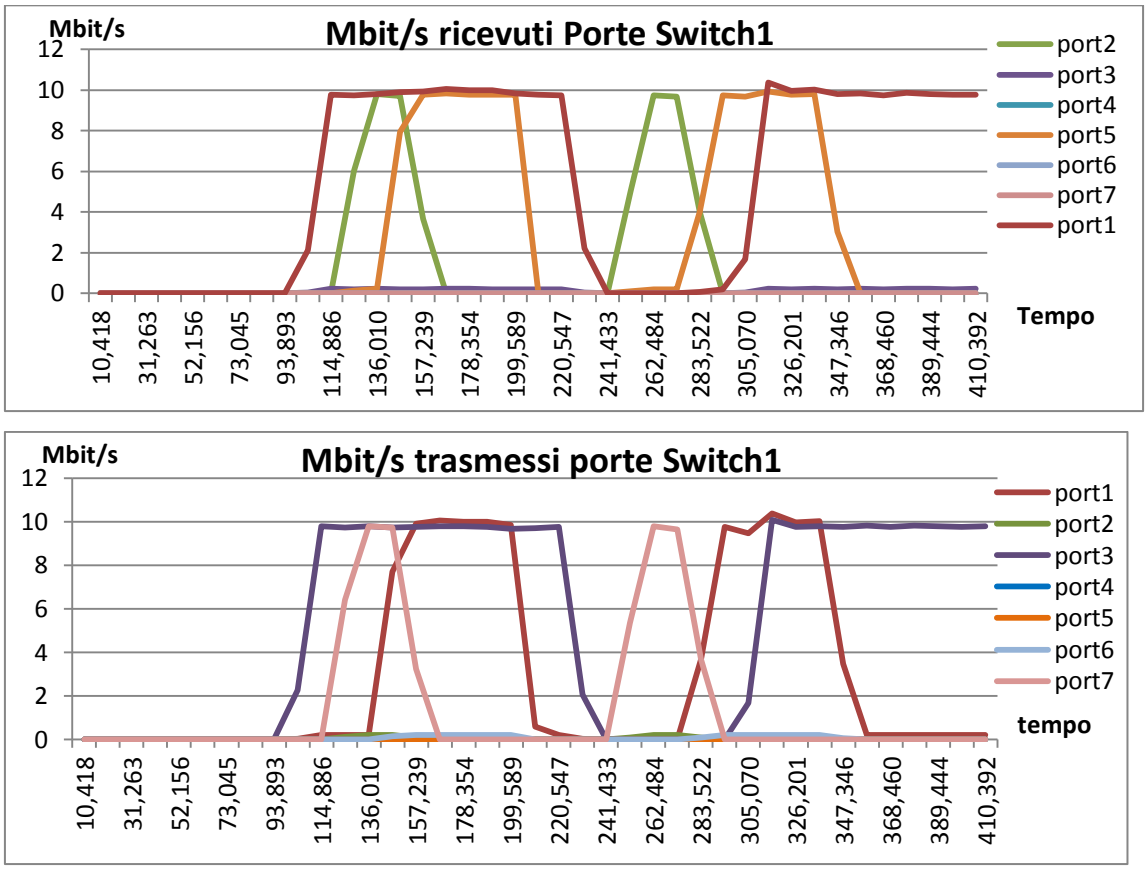


Figura 5.28 Flussi generati dai comandi Iperf.

Terminate le catture dei dati relativi a tutti gli switch è stato possibile elaborarli e generarne dei grafici, mediante i quali si ha la possibilità di verificare la correttezza dei flussi confrontandoli con le regole sopra descritte:

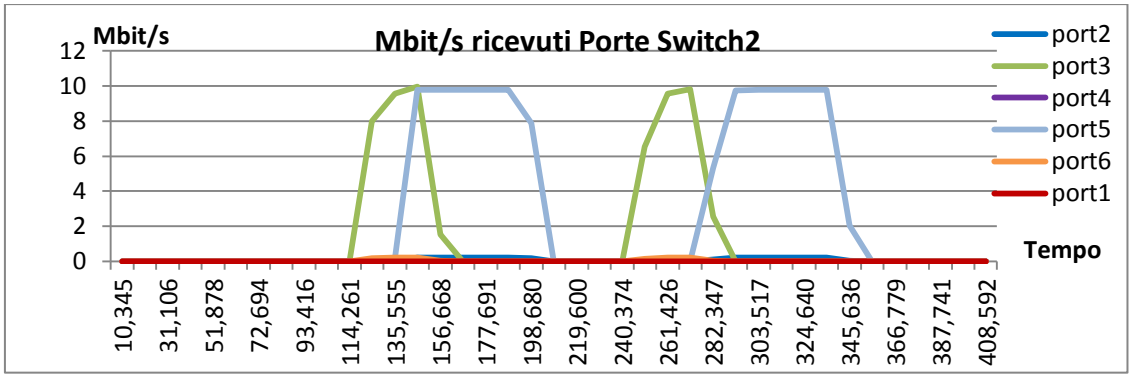
-Switch S1

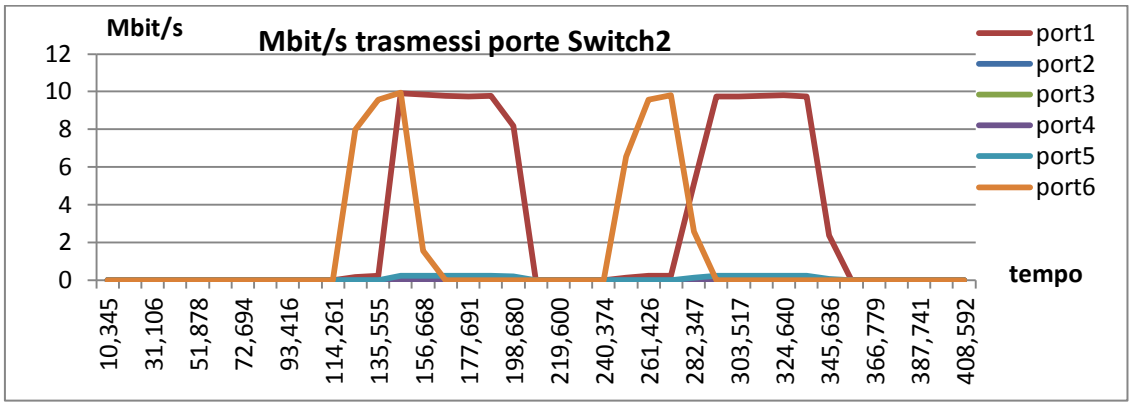


Possiamo notare 3 diversi flussi, i dati ricevuti dalla porta1 vengono trasmessi alla porta 3 quella relativa all' host 2, dimostrando che l'iperf lanciato da h10 viene correttamente instradato. Il flusso ricevuto alla porta 2 viene inoltrato alla porta 7 e quindi giustamente l'iperf lanciato da h1 ad h6 passa per lo switch S4(collegato alla porta 7). Infine l'ultimo viene giustamente ricevuto dalla porta 5(collegata allo switch 2) ed inoltrato verso la porta 1 collegata alla seconda rete infatti come destinazione abbiamo h9.

Notiamo anche che 2 flussi, uno entrante e uno uscente dalla porta 1, per un certo periodo di tempo avvengono in contemporanea senza diminuzione di banda, questo è douvuto al fatto che gli switch 5 e 6 sono stati connessi tramite software di virtualizzazione creando un canale con capacità maggiore di 10 Mb/s e questo consente il passaggio di entrambi i flussi senza avere cali di prestazione.

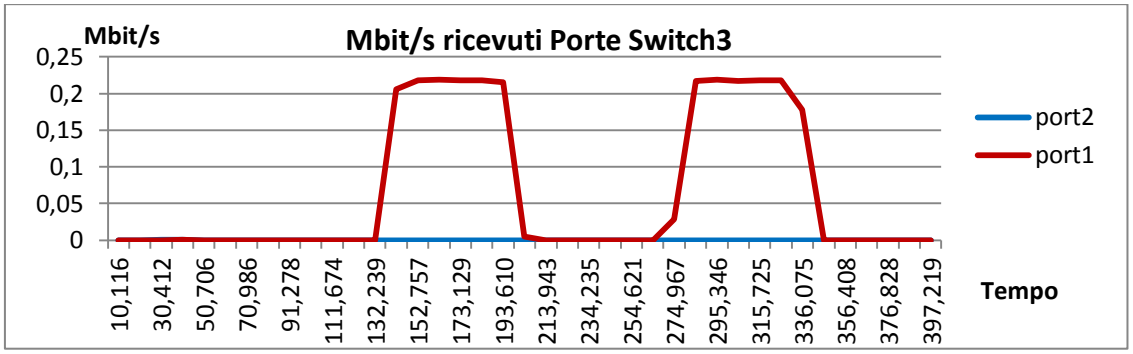
-Switch S2

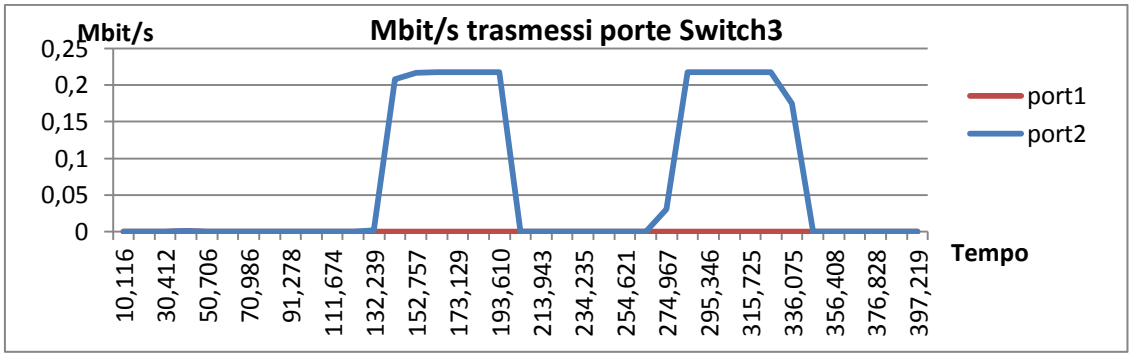




Dal grafico possiamo notare come i dati ricevuti dalla porta 3 vengano mandati in uscita presso la porta 6, infatti generati dall'iperf è lanciato da h1 verso h6. I dati ricevuti dalla porta 5, quindi iperf lanciato da h5, vengono inoltrati alla porta 1, infatti tutte le connessioni destinate ad indirizzi diversi da quelli della rete interna alla VM1 vengono spediti verso la porta 1 allo switch S5. Nel nostro caso il destinatario è appunto l'host 9 che si trova nella seconda VM. La presenza dei 2 flussi risulta quindi corretta.

-Switch S3

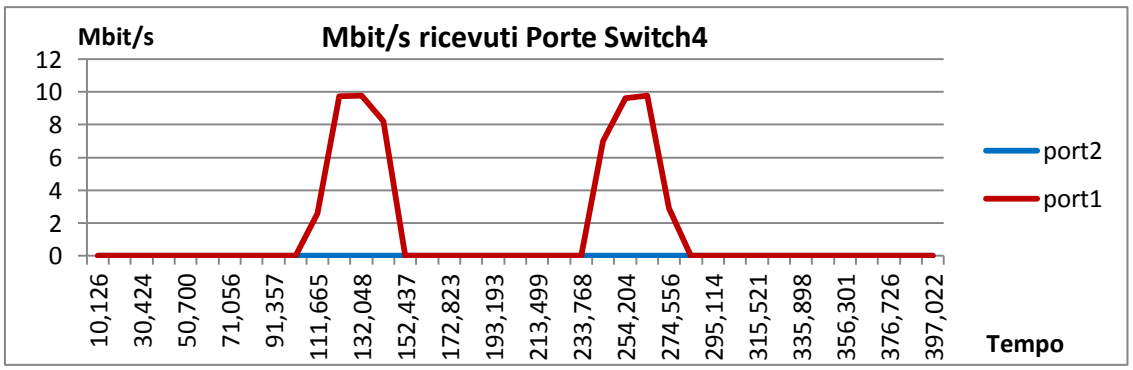


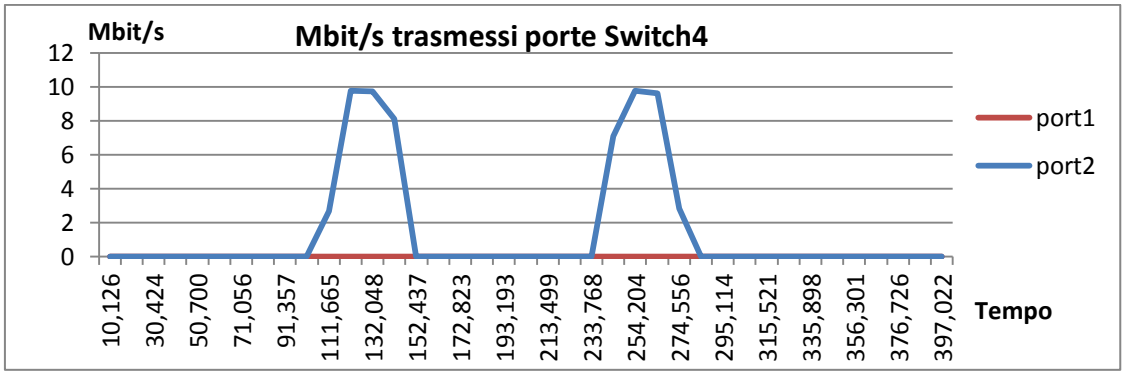


Dal grafico dello switch3 si può dedurre che una piccola quantità di dati viene inviata anche dal server per mantenere la connessione attiva. Infatti, si nota la presenza di un flusso poco superiore ai 0.2Mbit/s dovuto all' iperf lanciato da H5. Il destinatario (H9) risponde ad H5 e, come percorso per raggiungerlo, i pacchetti una volta arrivati ad S1 vengono instradati verso S3 per poi giungere ad S2 e quindi a H5 come da noi impostato nelle regole del controller.

La connessione di tipo TCP tra i due host (iperf) richiede delle risposte da parte, in questo caso, di H9 per segnalare la corretta ricezione di un pacchetto dati, e questa viene data mediante degli acknowledge. Questi piccoli trasferimenti che partono dai server si possono notare in particolar modo in questo grafico ma, pur non essendo stati fino ad ora considerati, sono presenti anche negli altri, dove possono essere notati lievemente. Tutto ciò dimostra il corretto funzionamento degli script di cattura da noi creati, e consente di controllare il corretto instradamento anche dei flussi delle risposte date dai server, verificandoli ,ottenendo riscontro positivo, con le regole inserite nel controller.

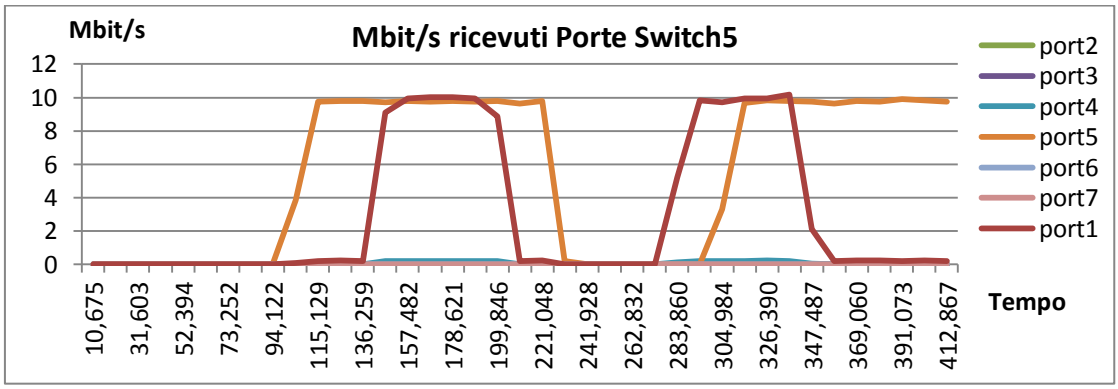
-Switch S4

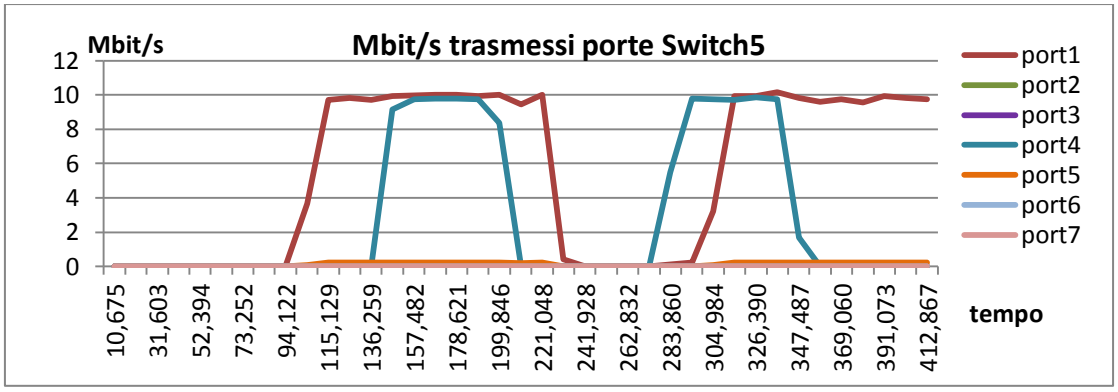




Attraverso lo switch 4 come da noi atteso passa un solo flusso ed è quello proveniente da h1 e destinato ad h6.

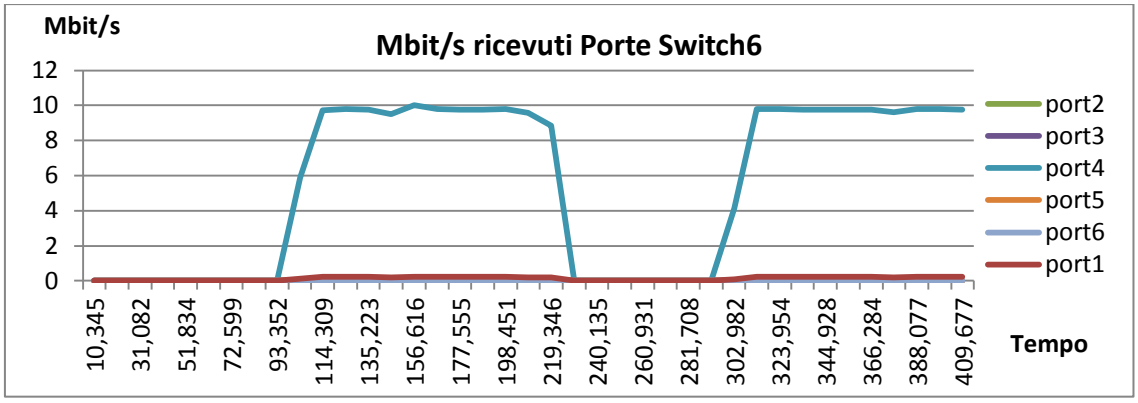
-Switch S5

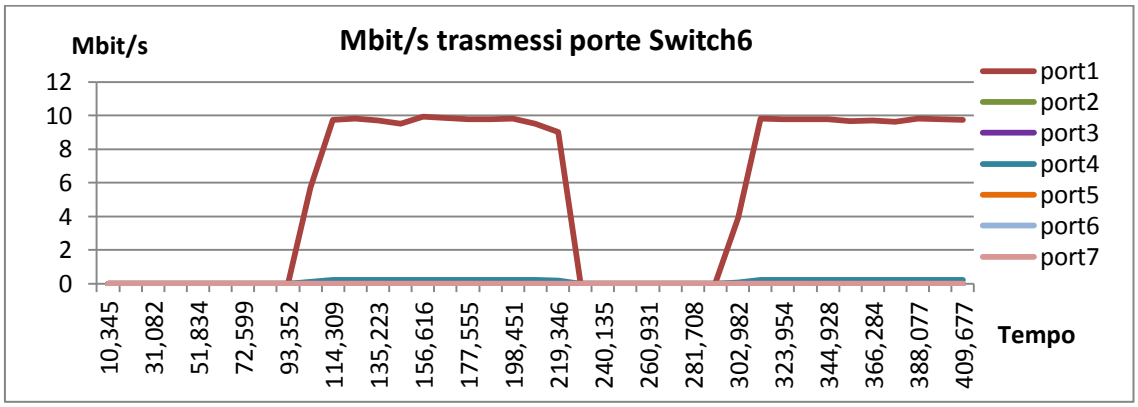




Ci troviamo ora nella VM2 dove il flusso in arrivo dalla VM1 (generato dall' iperf lanciato da h5) presso la porta 1 di S5 viene correttamente inoltrato al destinatario h9 collegato alla porta 4 del medesimo switch. Il flusso generato da h10 invece arriva ad S5 passando per la porta 5 e viene inoltrato verso la VM1 dalla porta 1. Di tutto ciò si può avere riscontro positivo visionando il grafico.

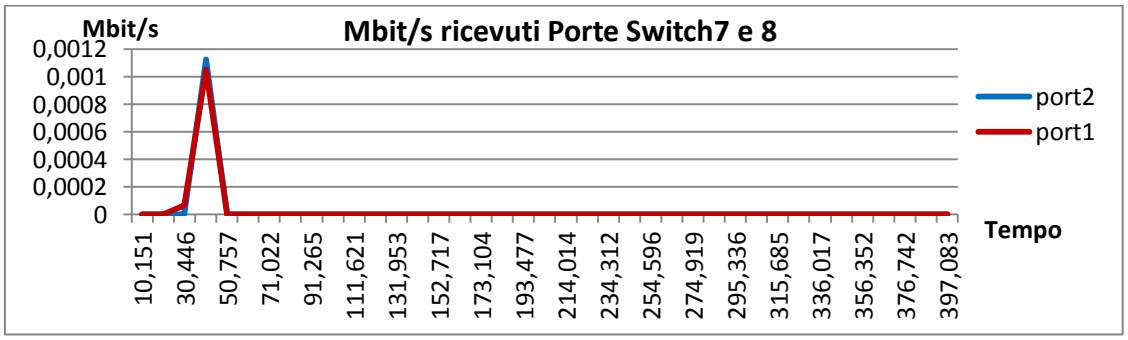
-Switch S6

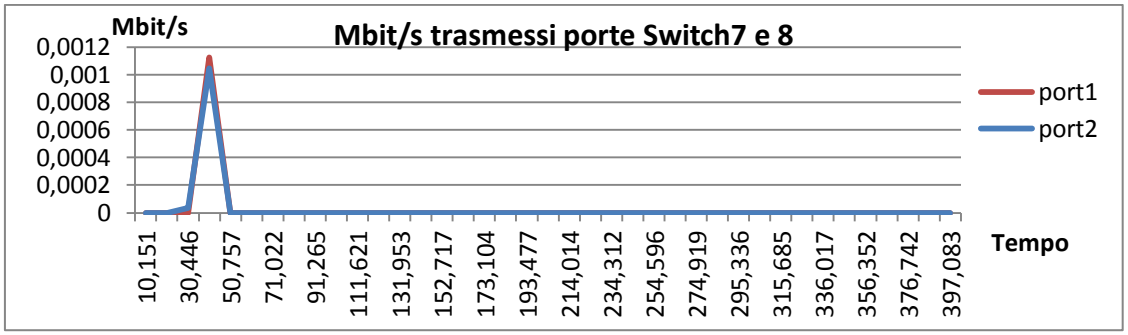




Nello switch 6 possiamo notare solamente la presenza del flusso generato da h10 e quindi entrante dalla porta 4 e inoltrato verso lo switch S5 dalla porta 1, in quanto il destinatario appartiene alla rete sulla VM1.

-Switch7 e Switch8





Infine negli ultimi 2 grafici relativi agli switch 7 e 8 (sono identici) si può notare che non transita alcun flusso, eccetto quello iniziale generato dai comandi pingall, che pur essendo quasi nullo, non è sfuggito alle nostre catture. Anche in questo caso questi piccolissimi flussi sono presenti in tutti i grafici, ma la scala, da noi mostrata negli altri, non li rende evidenti come per questi ultimi due switch.

Come ultime considerazioni notiamo che tutti i trasferimenti avvengono alla massima velocità prossima ai 10Mbit/s. Quindi almeno per quanto riguarda i tre flussi generati in questo test non si hanno link condivisi, eccetto il collegamento tra le 2VM il quale però come già accennato è stato impostato con una capacità maggiore di 10Mbit/s quindi non crea un effetto “collo di bottiglia” per le connessioni.
Abbiamo potuto constatare la correttezza della nostra rete ed anche il corretto instradamento dei flussi grazie a questa serie di grafici. La nostra rete è funzionante e comunica in modo appropriato con il Controller, gestendo i flussi come da noi programmato. Inoltre gli script, da noi creati per analizzare i dati, risultano molto efficaci, come si può notare vengono mostrati anche flussi molto piccoli di dati, quindi possono risultare un valido strumento per la verifica di una rete emulata ma eventualmente anche reale. Le valutazioni da noi effettuate oltre alla correttezza della rete possono quindi affermare la funzionalità del protocollo Openflow. Mediante il protocollo, infatti, siamo riusciti a gestire al meglio buona parte delle connessioni, ottimizzandole come inizialmente richiesto.

CONCLUSIONI

Con questa tesi si è voluto sperimentare utilizzando la nuova architettura Software Defined Networking ed il protocollo OpenFlow ad essa dedicato.
Tutto è stato reso possibile grazie al software di simulazione Mininet e ad uno dei framework per la creazione di un Controller (POX), che permettono di emulare un prototipo di rete funzionante in breve tempo e con a disposizione un semplice PC. La valutazione della rete è stata fatta su test di raggiungibilità e di performance che, pur essendo abbastanza semplici, aprono la possibilità a test molto più complessi ed esegubili su reti molto più amplie. Inoltre, il corretto funzionamento della rete su software Mininet ne garantisce il funzionamento in un eventuale implementazione su apparati reali che supportino OpenFlow. In una rete reale eventuali test di performance saranno probabilmente meno ottimali ed andrebbero valutati in maniera opportuna.
Problematiche come quella dei broadcast storm possono essere affrontate e in alcuni casi risolte in maniera semplice con un approccio software, come si è potuto anche osservare nel nostro elaborato. Infatti Mininet è un ottimo strumento per testare nuove implementazioni e fornisce la possibilità di applicare un agile metodo di sviluppo al Networking, come appunto quello che fino ad ora era pertinenza dei soli software.
Infine la possibilità di ampliare le reti e quindi distribuirle su più macchine, in cui è presente Mininet, potrebbe agevolare lo sviluppo di soluzioni per reti anche molto complesse. In quest'utlime è fondamentale eseguire dei test sulla fattibilità e sulla qualità delle prestazioni e, tutto ciò, non sarebbe realizzabile su una singola macchina. Nel nostro caso sono state collegate due macchine virtuali, ma nulla vieta che se ne possano connetere ulteriori o utilizzare macchine reali opportunamente collegate mediente le loro interfaccie.
Tutto questo potrebbe ampliare lo sviluppo di reti, delegandone parti diverse a diversi programmatori per poi essere assemblate, potrebbero essere create reti portatili allo scopo di connetterle a reti reali consentendo alcuni test, o semplicemente portebbe essere emulato il comportamento di un intera rete esistente utilizzando più computer.

Le tecnologie di rete in un futuro molto prossimo, e già adesso in alcuni casi, conosceranno una fase di forte innovazione, soprattutto, grazie all' interesse di molte grandi aziende. Le quali stanno provvedendo all' aggiornamento dei loro device in modo da poter lavorare anche con protocollo OpenFlow. Tutto ciò mostra un sempre maggiore interesse nei confronti dell' approccio SDN e di conseguenza programmi per la prototipazione delle reti come Mininet avranno un ruolo fondamentale negli anni a venire.

APPENDICE

In quest' appendice è possibile trovare tutti i codici, con anche alcuni commenti che ne facilitano la comprensione, utilizzati nei test eseguiti e nelle varie implementazioni.

A.1 Codice topologia avviata sulla prima macchina virtuale.

Il codice seguente è quello realizzato in uno script, per creare la topologia di rete minimale nella Virtual Machine contenente il software Mininet. I commenti sono presenti nel listato a seguito del segno cancelletto (#) eccezion fatta per la prima riga di codice.

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController
from mininet.cli import CLI
from mininet.link import Intf
from mininet.log import setLogLevel, info
from mininet.link import TCLink

#Definisco la rete
def myNetwork():

    net = Mininet(topo=None,
                  build=False, link=TCLink)

    #Aggiungo il Controller Remoto
    info('*** Adding controller\n') #con il comando info si stampano a video alcune scritte
    net.addController(name='c0',
                      controller=RemoteController,
                      ip='192.168.224.133',
                      port=6633)

    # aggiungo 4 SWITCH
    info( '*** Add switches\n')
    s1 = net.addSwitch('s1')
    s2 = net.addSwitch('s2')
    s3 = net.addSwitch('s3')
    s4 = net.addSwitch('s4')
    Intf('eth1',node = s1)
```

```
    #aggiungo 6 HOST
    info('*** Add hosts\n')
    h1 = net.addHost('h1', ip='192.168.2.21')
    h2 = net.addHost('h2', ip='192.168.2.22')
    h3 = net.addHost('h3', ip='192.168.2.23')
    h4 = net.addHost('h4', ip='192.168.2.24')
    h5 = net.addHost('h5', ip='192.168.2.25')
    h6 = net.addHost('h6', ip='192.168.2.26')


    # 3 host collegati a s1 3 host collegati a s2
    # switch tutti collegati tra loro eccetto 3 a 4
    info('*** Add links\n')
    net.addLink(h1, s1, bw=10)
    net.addLink(h2, s1, bw=10)
    net.addLink(h3, s1, bw=10)

    net.addLink(s1, s2, bw=10)
    net.addLink(s1, s3, bw=10)
    net.addLink(s1, s4, bw=10)

    net.addLink(s2, s3, bw=10)
    net.addLink(s2, s4, bw=10)

    net.addLink(h4, s2, bw=10)
    net.addLink(h5, s2, bw=10)
    net.addLink(h6, s2, bw=10)

    info('*** Starting network\n')
    #avvio la rete
    net.start()
    #Collego lo switch s1 alla porta eth1 della VM
    s1.cmd('ovs-vsctl add-port s1 eth1')
    s1.cmd('ifconfig s1 10.0.0.2')
   #Avvio la CLI per la rete
   CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    myNetwork()
```

A.2 Codice topologia avviata sulla seconda macchina virtuale.

Come si potrà notare lo script seguente è identico a quello realizzato per la prima Virtual machine, in quanto le 2 topologie sono simmetriche.

```
#!/usr/bin/python
```

```
import os
from mininet.net import Mininet
from mininet.node import Controller, RemoteController
from mininet.cli import CLI
from mininet.link import Intf
from mininet.log import setLogLevel, info
from mininet.link import TCLink

def myNetwork():

    net = Mininet(topo=None,
                  build=False, link=TCLink)

    #Aggiungo il Controller Remoto
    info('*** Adding controller\n')
    net.addController(name='c0',
                controller=RemoteController,
                ip='192.168.224.133',
                port=6633)

    # 4 SWITCH
    info( '*** Add switches\n')
    s5 = net.addSwitch('s5')
    s6 = net.addSwitch('s6')
    s7 = net.addSwitch('s7')
    s8 = net.addSwitch('s8')
    Intf('eth1',node = s5)

    #6 HOST
    info('*** Add hosts\n')
    h7 = net.addHost('h7', ip='192.168.2.31')
    h8 = net.addHost('h8', ip='192.168.2.32')
    h9 = net.addHost('h9', ip='192.168.2.33')
    h10 = net.addHost('h10', ip='192.168.2.34')
    h11 = net.addHost('h11', ip='192.168.2.35')
    h12 = net.addHost('h12', ip='192.168.2.36')

    info('*** Add links\n')
    net.addLink(h7, s5, bw=10)
    net.addLink(h8, s5, bw=10)
    net.addLink(h9, s5, bw=10)

    net.addLink(s5, s6, bw=10)
    net.addLink(s5, s7, bw=10)
    net.addLink(s5, s8, bw=10)

    net.addLink(s6, s7, bw=10)
    net.addLink(s6, s8, bw=10)

    net.addLink(h10, s6, bw=10)
    net.addLink(h11, s6, bw=10)
    net.addLink(h12, s6, bw=10)
```

```
    info('*** Starting network\n')
    net.start()
    #Collego lo switch s5 alla porta eth1 della VM
    s5.cmd('ovs-vsctl add-port s5 eth1')
    s5.cmd('ifconfig s5 10.0.0.3')
    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    myNetwork()
```

A.3 Codice del Controller per la topologia su una singola VM (4switch)

Il File è contenuto nella cartella misc di pox: Controller4switchflowmod.py. In questo listato si possono notare tutte le regole da noi impostate nei relativi switch. I commenti sono presenti nel listato a seguito del segno cancelletto (#) e alcuni tra le virgolette (’’’’’’).

```
"""
CONTROLLER PER 4 SWITCH
"""

from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
import pox.lib.packet as pkt
import os
import string

log = core.getLogger()

class Controller4switch (object):
  """
  Un oggetto Controller4switch viene creato per ogni switch che si connette
  L'oggetto Connection  per questo switch viene passato alla funzione __init__ .
  """
  def __init__ (self, connection):
    # Tiene traccia della connessione allo switch in modo da poter inviare i messaggi

    self.connection = connection
```

```
    # Lega il nostro ascoltatore di PacketIn event
    connection.addListeners(self)

  def install_flow (self, packet_in, match,dst_ip, out_port):
    """
    Installa una regola nell' relativo switch
    """
    msg = of.ofp_flow_mod(action = of.ofp_action_output(port = out_port),
match=of.ofp_match(dl_type=match.dl_type, nw_dst=dst_ip))
    self.connection.send(msg)
    msg = of.ofp_flow_mod(action = of.ofp_action_output(port = out_port),
match=of.ofp_match(dl_type=pkt.ethernet.IP_TYPE, nw_dst=dst_ip))
    self.connection.send(msg)

  def _handle_PacketIn (self, event):
    """
    Gestisce i PacketIn provenienti dagli Switch.
    """
  #Convert an Hex to Dec
    def convertToInt(hexval):
          dpid_split = string.split(hexval,'-')
          concat_hex = ''
          for elem in dpid_split:
            concat_hex+=elem
          intval=int(concat_hex,16)
          return intval

    packet = event.parsed # Contiene I dati del pacchetto analizzato.
    if not packet.parsed:
      log.warning("Ignoring incomplete packet")
      return

    packet_in = event.ofp # L'attuale messaggio ofp_packet_in.

    src_dpid = dpid_to_str(event.dpid)
    int_dpid = convertToInt(src_dpid)

    log.debug("receiving paket from switch s%d", int_dpid)
    log.debug("(DPID=%s)", src_dpid)



    match = of.ofp_match.from_packet(packet)#l'oggetto match contiene alcuni parametric del
pacchetto

    if match.dl_type == pkt.ethernet.ARP_TYPE or match.dl_type == pkt.ethernet.IP_TYPE :
# se il pacchetto è di tipo IP O ARP creo le seguenti variabili
      log.debug("IP or ARP packet")
      dst_ip = match.nw_dst
      dst_ip_str = dst_ip.toStr()
      dst_ip_split = string.split(dst_ip_str, '.')

      src_ip = match.nw_src
```

```
    src_ip_str = src_ip.toStr()
    src_ip_split = string.split(src_ip_str, '.')
#Switch 1(se il dpid è 1)
    if int_dpid == 1:
      log.debug("SWITCH S1")
      if dst_ip_split[3] == '21':
        self.install_flow(packet_in, match, dst_ip, 2)
# se il destinatario è il 192.168.2.21 allora inoltra alla porta 2

      elif dst_ip_split[3] == '22':
        self.install_flow(packet_in, match, dst_ip, 3)
# se il destinatario è il 192.168.2.22 allora inoltra alla porta 3

      elif dst_ip_split[3] == '23':
        self.install_flow(packet_in, match, dst_ip, 4)
# se il destinatario è il 192.168.2.23 allora inoltra alla porta 4

      elif dst_ip_split[3] == '24':
        self.install_flow(packet_in, match, dst_ip, 5)
        log.debug("inviato a 192.168.2.24!!")
# se il destinatario è il 192.168.2.24 allora inoltra alla porta 5

      elif dst_ip_split[3] == '25':
        self.install_flow(packet_in, match, dst_ip, 6)
        log.debug("inviato a 192.168.2.25!!")
# se il destinatario è il 192.168.2.25 allora inoltra alla porta 6

      elif dst_ip_split[3] == '26':
        self.install_flow(packet_in, match, dst_ip, 7)
        log.debug("inviato a 192.168.2.26!!")
# se il destinatario è il 192.168.2.26 allora inoltra alla porta 7
      else:
        self.install_flow(packet_in, match, dst_ip, 1)
# in tutti gli altri casi manda in uscita sulla porta 1

#Switch 2 valgono le stesse considerazione effettuate per lo switch 1
    elif int_dpid == 2:
      log.debug("SWITCH S2")
      if dst_ip_split[3] == '24':
        self.install_flow(packet_in, match, dst_ip, 4)

      elif dst_ip_split[3] == '25':
        self.install_flow(packet_in, match, dst_ip, 5)

      elif dst_ip_split[3] == '26':
        self.install_flow(packet_in, match, dst_ip, 6)

      elif dst_ip_split[3] == '23':
        self.install_flow(packet_in, match, dst_ip, 3)

      elif dst_ip_split[3] == '22':
        self.install_flow(packet_in, match, dst_ip, 2)
```

```
        elif dst_ip_split[3] == '21':
          self.install_flow(packet_in, match, dst_ip, 1)

        else:
          self.install_flow(packet_in, match, dst_ip, 1)

#Switch 3
      elif int_dpid == 3:
        log.debug("SWITCH S3")
        msg = of.ofp_flow_mod()
        msg.match.in_port = 1
        msg.actions.append(of.ofp_action_output(port=2))# IN port:1 Out port:2
        self.connection.send(msg)
        msg = of.ofp_flow_mod()
        msg.match.in_port = 2
        msg.actions.append(of.ofp_action_output(port=1))# IN port:2 Out port:1
        self.connection.send(msg)
#Switch 4
      elif int_dpid == 4:
        log.debug("SWITCH S4")
        msg = of.ofp_flow_mod()
        msg.match.in_port = 1
        msg.actions.append(of.ofp_action_output(port=2))# IN port:1 Out port:2
        self.connection.send(msg)
        msg = of.ofp_flow_mod()
        msg.match.in_port = 2
        msg.actions.append(of.ofp_action_output(port=1))# IN port:2 Out port:1
        self.connection.send(msg)
#switch 3 e 4 , tutto quello che entra dalla porta 1 viene mandato in uscita sulla porta 2 e viceversa.


def launch ():
  """
  Inizializza i componenti
  """
  def start_switch (event):
    log.debug("Controlling %s" % (event.connection,))
    Controller4switch(event.connection)
  core.openflow.addListenerByName("ConnectionUp", start_switch)
```

A.4 Controller per la Rete Distribuita (8 switch)

Per brevità non verrà inserito l'intero codice ma solamente la parte aggiunta rispetto al Controller per 4 switch, ovvero quella inerente gli switch 5, 6, 7 e 8, che è inserita subito dopo lo switch 4 nel listato presente nel paragrafo A.3.

Il file integrale è Controller8switchflowmod.py contenuto nella cartella misc di pox.
La parte aggiunta rispetto a Controller4switchflowmod.py è la seguente:

```
#Switch 5
    elif int_dpid == 5:
      log.debug("SWITCH S5")
      if dst_ip_split[3] == '31':
        self.install_flow(packet_in, match, dst_ip, 2)
        log.debug("inviato a 192.168.2.31!!")

      elif dst_ip_split[3] == '32':
        self.install_flow(packet_in, match, dst_ip, 3)
        log.debug("inviato a 192.168.2.32!!")

      elif dst_ip_split[3] == '33':
        self.install_flow(packet_in, match, dst_ip, 4)
        log.debug("inviato a 192.168.2.33!!")

      elif dst_ip_split[3] == '34':
        self.install_flow(packet_in, match, dst_ip, 5)
        log.debug("inviato a 192.168.2.34!!")

      elif dst_ip_split[3] == '35':
        self.install_flow(packet_in, match, dst_ip, 6)
        log.debug("inviato a 192.168.2.35!!")

      elif dst_ip_split[3] == '36':
        self.install_flow(packet_in, match, dst_ip, 7)
        log.debug("inviato a 192.168.2.36!!")

      else:
        self.install_flow(packet_in, match, dst_ip, 1)
        log.debug("If not in my net, go to s1")

#Switch 6
    elif int_dpid == 6:
      log.debug("SWITCH S6")
      if dst_ip_split[3] == '34':
        self.install_flow(packet_in, match, dst_ip, 4)

      elif dst_ip_split[3] == '35':
        self.install_flow(packet_in, match, dst_ip, 5)

      elif dst_ip_split[3] == '36':
        self.install_flow(packet_in, match, dst_ip, 6)
```

```
        elif dst_ip_split[3] == '33':
          self.install_flow(packet_in, match, dst_ip, 3)

        elif dst_ip_split[3] == '32':
          self.install_flow(packet_in, match, dst_ip, 2)

        elif dst_ip_split[3] == '31':
          self.install_flow(packet_in, match, dst_ip, 1)

        else:
          self.install_flow(packet_in, match, dst_ip, 1)
          log.debug("If not in my net, go to s5")

#Switch 7
      elif int_dpid == 7:
        log.debug("SWITCH S7")
        msg = of.ofp_flow_mod()
        msg.match.in_port = 1
        msg.actions.append(of.ofp_action_output(port=2))# IN port:1 Out port:2
        self.connection.send(msg)
        msg = of.ofp_flow_mod()
        msg.match.in_port = 2
        msg.actions.append(of.ofp_action_output(port=1))# IN port:2 Out port:1
        self.connection.send(msg)

#Switch 8
      elif int_dpid == 8:
        log.debug("SWITCH S8")
        msg = of.ofp_flow_mod()
        msg.match.in_port = 1
        msg.actions.append(of.ofp_action_output(port=2)) # IN port:1 Out port:2
        self.connection.send(msg)
        msg = of.ofp_flow_mod()
        msg.match.in_port = 2
        msg.actions.append(of.ofp_action_output(port=1)) # IN port:2 Out port:1
        self.connection.send(msg)

#se il datapath ID non è presente nel elenco allora lo switch non è della mia rete.
      else:
        log.debug("The swich isn't in my net")
```

A.5 script C gettimestamp.c

File : gettimestamp.c

```
#include <sys/time.h>
#include <stdio.h>
```

```
main (void) {

        struct timeval currentTime;

        gettimeofday(&currentTime,NULL);
        printf("%.6f\n",(double)currentTime.tv_sec+currentTime.tv_usec/1000000.0);

}
```

A.6 Sript shell per la cattura dei dati.

Vengono di seguito riportati gli script eseguiti per poter catturare i dati relativi alle porte degli switch e poter quindi creare dei grafici inerenti ai troughput. I listati sono identici per i diversi switch ad eccezione del comando ovs-ofctl che viene eseguito sul relativo switch e il numero di comandi che è funzione delle porte presenti.

-Switch1:

File Switch1.sh:

```
#! /bin/bash

period=10
echo "# BYTE TRASMESSI E RICEVUTI DALLE PORTE SWITCH s1 ogni $period secondi"
echo "#time          rx port1  tx port1  rx port2  tx port2  rx port3  tx port3  rx port4  tx port4  rx
port5  tx port5  rx port6  tx port6  rx port7  tx port7"
while true
do

      sw1p1rx=`sudo ovs-ofctl dump-ports s1|grep -A 1 1:|grep rx|cut -d= -f3|cut -d, -f1`
      sw1p1tx=`sudo ovs-ofctl dump-ports s1|grep -A 1 1:|grep tx|cut -d= -f3|cut -d, -f1`
      sw1p2rx=`sudo ovs-ofctl dump-ports s1|grep -A 1 2:|grep rx|cut -d= -f3|cut -d, -f1`
      sw1p2tx=`sudo ovs-ofctl dump-ports s1|grep -A 1 2:|grep tx|cut -d= -f3|cut -d, -f1`
      sw1p3rx=`sudo ovs-ofctl dump-ports s1|grep -A 1 3:|grep rx|cut -d= -f3|cut -d, -f1`
      sw1p3tx=`sudo ovs-ofctl dump-ports s1|grep -A 1 3:|grep tx|cut -d= -f3|cut -d, -f1`
      sw1p4rx=`sudo ovs-ofctl dump-ports s1|grep -A 1 4:|grep rx|cut -d= -f3|cut -d, -f1`
      sw1p4tx=`sudo ovs-ofctl dump-ports s1|grep -A 1 4:|grep tx|cut -d= -f3|cut -d, -f1`
      sw1p5rx=`sudo ovs-ofctl dump-ports s1|grep -A 1 5:|grep rx|cut -d= -f3|cut -d, -f1`
      sw1p5tx=`sudo ovs-ofctl dump-ports s1|grep -A 1 5:|grep tx|cut -d= -f3|cut -d, -f1`
      sw1p6rx=`sudo ovs-ofctl dump-ports s1|grep -A 1 6:|grep rx|cut -d= -f3|cut -d, -f1`
      sw1p6tx=`sudo ovs-ofctl dump-ports s1|grep -A 1 6:|grep tx|cut -d= -f3|cut -d, -f1`
      sw1p7rx=`sudo ovs-ofctl dump-ports s1|grep -A 1 7:|grep rx|cut -d= -f3|cut -d, -f1`
      sw1p7tx=`sudo ovs-ofctl dump-ports s1|grep -A 1 7:|grep tx|cut -d= -f3|cut -d, -f1`

      t=`./gettimestamp`

      echo -n "$t  "
```

```
        echo -n "$sw1p1rx     "
        echo -n "$sw1p1tx     "
        echo -n "$sw1p2rx     "
        echo -n "$sw1p2tx     "
        echo -n "$sw1p3rx     "
        echo -n "$sw1p3tx     "
        echo -n "$sw1p4rx     "
        echo -n "$sw1p4tx     "
        echo -n "$sw1p5rx     "
        echo -n "$sw1p5tx     "
        echo -n "$sw1p6rx     "
        echo -n "$sw1p6tx     "
        echo -n "$sw1p7rx     "
        echo "$sw1p7tx"


        sleep $period
done
```

-Switch2:

File Switch2.sh

```
#! /bin/bash

period=2
echo "# BYTE TRASMESSI E RICEVUTI DALLE PORTE SWITCH s2 ogni $period secondi"
echo "#time          rx port1  tx port1  rx port2  tx port2  rx port3  tx port3  rx port4  tx port4  rx
port5  tx port5  rx port6  tx port6"
while true
do

    sw1p1rx=`sudo ovs-ofctl dump-ports s2|grep -A 1 1:|grep rx|cut -d= -f3|cut -d, -f1`
    sw1p1tx=`sudo ovs-ofctl dump-ports s2|grep -A 1 1:|grep tx|cut -d= -f3|cut -d, -f1`
    sw1p2rx=`sudo ovs-ofctl dump-ports s2|grep -A 1 2:|grep rx|cut -d= -f3|cut -d, -f1`
    sw1p2tx=`sudo ovs-ofctl dump-ports s2|grep -A 1 2:|grep tx|cut -d= -f3|cut -d, -f1`
    sw1p3rx=`sudo ovs-ofctl dump-ports s2|grep -A 1 3:|grep rx|cut -d= -f3|cut -d, -f1`
    sw1p3tx=`sudo ovs-ofctl dump-ports s2|grep -A 1 3:|grep tx|cut -d= -f3|cut -d, -f1`
    sw1p4rx=`sudo ovs-ofctl dump-ports s2|grep -A 1 4:|grep rx|cut -d= -f3|cut -d, -f1`
    sw1p4tx=`sudo ovs-ofctl dump-ports s2|grep -A 1 4:|grep tx|cut -d= -f3|cut -d, -f1`
    sw1p5rx=`sudo ovs-ofctl dump-ports s2|grep -A 1 5:|grep rx|cut -d= -f3|cut -d, -f1`
    sw1p5tx=`sudo ovs-ofctl dump-ports s2|grep -A 1 5:|grep tx|cut -d= -f3|cut -d, -f1`
    sw1p6rx=`sudo ovs-ofctl dump-ports s2|grep -A 1 6:|grep rx|cut -d= -f3|cut -d, -f1`
    sw1p6tx=`sudo ovs-ofctl dump-ports s2|grep -A 1 6:|grep tx|cut -d= -f3|cut -d, -f1`

    t=`./gettimestamp`

    echo -n "$t  "
        echo -n "$sw1p1rx     "
        echo -n "$sw1p1tx     "
        echo -n "$sw1p2rx     "
```

```
        echo -n "$sw1p2tx     "
        echo -n "$sw1p3rx     "
        echo -n "$sw1p3tx     "
        echo -n "$sw1p4rx     "
        echo -n "$sw1p4tx     "
        echo -n "$sw1p5rx     "
        echo -n "$sw1p5tx     "
        echo -n "$sw1p6rx     "
        echo "$sw1p6tx"


        sleep $period
done
```

-Switch3:

File Switch3.sh

```
#! /bin/bash

period=2
echo "# BYTE TRASMESSI E RICEVUTI DALLE PORTE SWITCH s3 ogni $period secondi"
echo "#time         rx port1  tx port1  rx port2  tx port2"
while true
do

    sw1p1rx=`sudo ovs-ofctl dump-ports s3|grep -A 1 1:|grep rx|cut -d= -f3|cut -d, -f1`
    sw1p1tx=`sudo ovs-ofctl dump-ports s3|grep -A 1 1:|grep tx|cut -d= -f3|cut -d, -f1`
    sw1p2rx=`sudo ovs-ofctl dump-ports s3|grep -A 1 2:|grep rx|cut -d= -f3|cut -d, -f1`
    sw1p2tx=`sudo ovs-ofctl dump-ports s3|grep -A 1 2:|grep tx|cut -d= -f3|cut -d, -f1`

    t=`./gettimestamp`

    echo -n "$t  "
        echo -n "$sw1p1rx     "
        echo -n "$sw1p1tx     "
        echo -n "$sw1p2rx     "
        echo "$sw1p2tx"


        sleep $period
done
```

-Switch4:

File Switch4.sh

```
#! /bin/bash

period=2
echo "# BYTE TRASMESSI E RICEVUTI DALLE PORTE SWITCH s4 ogni $period secondi"
```

```
echo "#time          rx port1  tx port1  rx port2  tx port2"
while true
do

      sw4p1rx=`sudo ovs-ofctl dump-ports s4|grep -A 1 1:|grep rx|cut -d= -f3|cut -d, -f1`
      sw4p1tx=`sudo ovs-ofctl dump-ports s4|grep -A 1 1:|grep tx|cut -d= -f3|cut -d, -f1`
      sw4p2rx=`sudo ovs-ofctl dump-ports s4|grep -A 1 2:|grep rx|cut -d= -f3|cut -d, -f1`
      sw4p2tx=`sudo ovs-ofctl dump-ports s4|grep -A 1 2:|grep tx|cut -d= -f3|cut -d, -f1`

      t=`./gettimestamp`

      echo -n "$t  "
         echo -n "$sw4p1rx     "
         echo -n "$sw4p1tx     "
         echo -n "$sw4p2rx     "
         echo "$sw4p2tx"


         sleep $period
done
```

Switch 5, 6, 7 e 8:

per brevità gli script relativi a questi switch non vengono riportati ma sono identici a quelli per gli script degli switch 1, 2, 3 e 4 nei quali si è cambiato semplicemente in tutti i comandi il nome dello switch, da s1 a s5 da s2 a s6 da s3 a s7 e da s4 a s8.

BIBLIOGRAFIA

[1] Castellano Marcello, Perfetti Salvatore, Simone Vincenzo, Andriola Vito: Software defined networking , seminario expertis 2013 http://www.marcellocastellano.it/web/didattica/seminari/

[2] ONF White Paper 13 aprile, 2012 https://www.opennetworking.org/images/stories/downloads/sdn-resources /white-papers/wp-sdn-newnorm.pdf

[3] SDN, Markus Nispel http://www.itproportal.com/2015/02/05/sdn-101-everything-need-know/

[4] Openflow, Wikipedia http://it.wikipedia.org/wiki/OpenFlow

[5]Antonio Manzalini, Vinicio Vercellone and Mario Ullio, "Software Defined Networking: sfide e opportunità per le reti del futuro", Notiziario tecnico Telecom Italia, No. 1, pp. 30-43, 2013.

[6] Openflow : http://archive.openflow.org/documents/openflow-wp-latest.pdf

[7] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, Scott Shenker. "Ethane: Taking Control of the Enterprise," ACM SIGCOMM '07, August 2007, Kyoto, Japan.

[8] Brandon Heller et al. 2010 "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks" *ACM SIGCOMM Hotnets Conference, Nov 2010.*

[9] Introduzione a mininet: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

[10] Openflow tutorial:

http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial

[11] Mininet, sito ufficiale : http://mininet.org

[12] Linguaggio di programmazione Python : http://www.python.it

[13] Secure shell , Wikipedia : http://it.wikipedia.org/wiki/Secure_shell

[14] Comando ping, Wikipedia : http://it.wikipedia.org/wiki/Ping

[15] Comando iperf, Wikipedia: http://en.wikipedia.org/wiki/Iperf