Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea in Informatica

Improving network management with Software-Defined Networking

Elaborato Finale

Laureanda

Benedetta Contigiani

Relatore

Prof. Fausto Marcantoni

Correlatore

Fabrizio Ippoliti

Anno Accademico 2015/2016

Indice

1 Introduzione

Il numero sempre crescente di utenti e servizi che fanno uso della rete Internet ha fatto sì che quest'ultima divenisse costantemente più estesa e complessa. Il problema principale è rappresentato dalla staticità dell'architettura attuale che si contrappone ai bisogni degli utilizzatori, i quali richiedono una gestione dinamica, veloce e sicura della rete. Tutto ciò ha portato l'industria del networking a riesaminare completamente la struttura della tradizionale architettura di rete. Negli ultimi anni, infatti, anche il mondo della ricerca ha concentrato i suoi sforzi per cercare di migliorare questa struttura, ma con scarsi risultati data la complessità attuale di Internet e la difficoltà nell'apportarvi delle modifiche. Qualche anno fa, però la *Open Networking Foundation* (ONF) ha avanzato una proposta che sembra costituire una vera svolta per la crescita e l'evoluzione dell'attuale rete di intercomunicazione, il *Software-Defined Networking* (SDN).

In questo documento viene esaminato il background attuale e le motivazioni che hanno portato alla definizione di SDN. SDN è un approccio a una nuova architettura dinamica, programmabile e adattabile in grado di rappresentare una soluzione ideale per le applicazioni odierne. Questa architettura ha lo scopo di semplificare la complessità nella progettazione, distribuzione e manutenzione delle moderne reti riducendo la necessità di stratificazione all'interno delle reti e permettendo di apportare modifiche a ogni livello senza influenzare il successivo. In particolare viene analizzato nel dettaglio questo emergente paradigma di rete evidenziandone i punti di forza, le potenzialità e i benefici, nonché eventuali punti di debolezza. Inoltre viene approfondito lo standard *OpenFlow*, sostenuto anche esso da ONF, che si è imposto come base per lo sviluppo di SDN. Tramite lo standard OpenFlow si è andato oltre alle black boxes dei produttori, riuscendo a cogliere quelli che sono dei funzionamenti comuni degli switch e ad ottenere delle politiche di inoltro indipendenti dall'hardware sulle quali sono applicate. Dopodiché viene utilizzato *Mininet*, software che simula svariate topologie di reti, per dimostrare come sia possibile creare rapidamente ed efficientemente una network basata su SDN capace di adattarsi alla topologia con il minimo sforzo, potendo eseguire regression test automatizzati. Dopo aver studiato gli elementi fondamentali del paradigma SDN, le basi del protocollo OpenFlow e il funzionamento dell'ambiente di simulazione Mininet, si è proceduto allo studio di un controller SDN, *OpenDaylight* (ODL), di cui vengono analizzati la struttura e i principali componenti per comprenderne meglio il suo funzionamento.

Infine si è proceduto alla creazione ed alla verifica, eseguendo vari test, del funzionamento di una topologia di rete virtuale emulata da Mininet in comunicazione con il controller ODL. Tutto ciò con lo scopo di capire come gestire e monitorare una rete logicamente centralizzata, definita da SDN, e come migliorare e ottimizzare le sue prestazioni.

2 Software-Defined Networking

In questo capitolo, si esamina lo stato dell'arte del paradigma SDN, studiandone i suoi componenti e la sua architettura. Inoltre vengono analizzate le motivazioni che hanno portato a questa innovazione e quali sono i benefici apportati dal cambiamento.

2.1 Limiti delle reti tradizionali

Dal 1970 a oggi, la rete attuale non ha subito significative modifiche per quanto riguarda le tecnologie e i dispositivi di rete. Nei dispositivi che compongono la rete tradizionale, come raffigurato nella Figura 1, il *data plane* e il *control plane* coesistono all'interno dello stesso sistema.

Il control plane contiene le funzioni di instradamento o routing. La funzione di routing è quella parte che si occupa di calcolare e determinare il percorso dei pacchetti, ovvero quale direzione deve seguire un pacchetto attraverso la rete per raggiungere la sua destinazione. Tale parte è costituita da algoritmi di routing che calcolano il percorso a seconda della conoscenza della topologia della rete, dai protocolli di routing che si scambiano le informazioni sulla topologia della rete con i nodi vicini e da funzioni di routing che generano le tabelle di routing con le informazioni a disposizione.

Il data plane, anche conosciuto con il nome di forwarding plane, è l'hardware specializzato per l'inoltro dei pacchetti. Questo livello si occupa di inoltrare i pacchetti in arrivo verso il next hop, attraverso il percorso selezionato dalla logica del control plane.

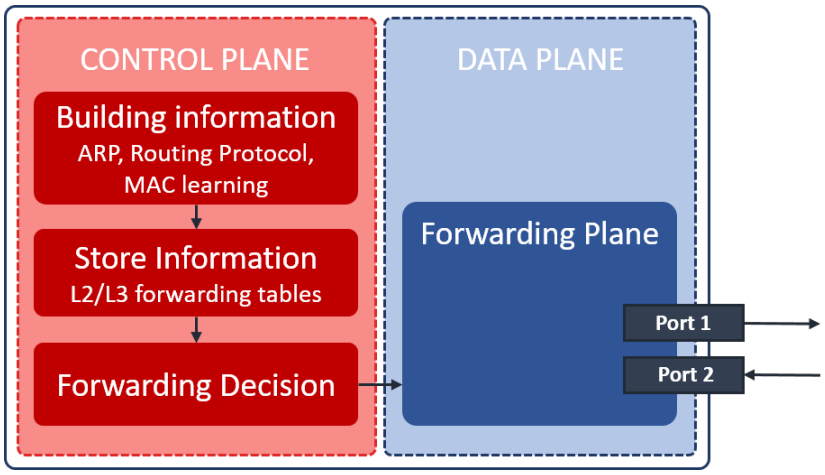


Figura 1: Principali componenti di uno switch tradizionale

Il problema principale delle architetture attuali è quello della staticità che si contrappone ai bisogni degli utilizzatori, i quali richiedono una gestione dinamica, veloce e sicura della rete. Inoltre l'infrastruttura descritta è chiusa alle innovazioni e cresce lentamente facendo si che la sua funzionalità sia limitata dalle caratteristiche offerte e imposte dai fornitori dell'hardware.

L'avvento di nuove tecnologie, l'introduzione della virtualizzazione e l'incremento dei servizi cloud infatti hanno causato un aumento di complessità di amministrazione e di funzionamento della rete che l'attuale struttura, costituita per un utilizzo gerarchico a più livelli, non è in grado di gestire. Sebbene questa struttura, fino a qualche anno fa, sia stata adeguata, recentemente ha mostrato le sue debolezze. Di seguito, nei prossimi sottoparagrafi, vengono elencati alcuni di questi aspetti che hanno evidenziato la necessità di un cambiamento [1].

Complessità strutturale

Con l'introduzione di nuovi dispositivi e l'aumento dei servizi cloud, la richiesta di accesso alla rete è conseguentemente aumentata. La tecnologia di oggi è costituita da un insieme di protocolli progettati per connettere host fra di loro in maniera affidabile su distanze, velocità di connessione e topologie di rete variabili. Per soddisfare i requisiti tecnici e commerciali richiesti i protocolli di rete sono stati perfezionati in modo tale da poter offrire prestazioni migliori e affidabili, garantire una connettività più ampia e assicurare un buon livello di sicurezza.

Tuttavia, tali protocolli solitamente sono definiti in maniera isolata con lo scopo di fornire una soluzione ad un problema di comunicazione tra due o più entità senza però offrire un adeguato livello di astrazione rendendo così le reti più complesse con l'aggregazione di ogni protocollo.

Scalabilità limitata

La rete tradizionale non permette di avere una rete scalabile, ovvero una rete che abbia le stesse prestazioni senza degradazioni nel momento in cui questa cresce. Ad esempio, una piccola azienda dispone inizialmente di una rete modesta, proporzionata alle sue esigenze iniziali; poi l'azienda cresce, e ha la necessità di estendere la rete e aggiungere nuovi dispositivi, i quali sono configurabili attraverso terminale remoto, in maniera manuale, lavorando sulle singole apparecchiature, dall'amministratore della rete. Perciò anche la più semplice modifica alla rete, può richiedere anche giorni di lavoro.

Tale ridimensionamento non può essere affrontato attraverso una configurazione manuale. Si tratta, quindi, di un fattore cruciale per la gestione efficiente delle reti siccome queste possono crescere in maniera sproporzionata e impredicibile.

Scarsa flessibilità

La modalità di circolazione del traffico dati è cambiata. Inizialmente il traffico consisteva nello scambio di pacchetti che si concentrava fra client e server selezionati, ovvero i pacchetti viaggiavano da un capo all'altro della rete senza subire grandi trasformazioni. Invece, oggi le applicazioni sono distribuite fra un gran numero di server, i quali si scambiano flussi di grandi quantità di dati tra di loro. Il traffico intermedio fra queste applicazioni distribuite è altamente dinamico, ovvero non è possibile prevedere in anticipo quali saranno i flussi di traffico fra i vari nodi. Questo nuovo aspetto dinamico è in netto contrasto alla natura statica e poco flessibile del modello convenzionale della rete, il quale resta legato alla staticità delle apparecchiature, incapaci di adeguarsi a tali cambiamenti.

Lentezza

Un nuovo protocollo prima di essere utilizzato deve essere implementato all'interno dei dispositivi e questo passaggio più richiedere molto tempo e il conseguente rallentamento dello sviluppo della rete. Questo è dovuto al fatto che è molto più veloce lo sviluppo di un nuovo software che porta dei miglioramenti alla rete piuttosto che la sua installazione su tutti i dispositivi che ne necessitano.

Banda Limitata

Una grande quantità di dati comporta una maggiore banda e il saper gestire più processi in parallelo di migliaia di server collegati tra loro, rispettando e utilizzando tutti i protocolli sviluppati. Tutto ciò in una rete tradizionale può portare all'esaurimento delle risorse, al decadimento delle prestazioni e nel peggiore delle ipotesi all'interruzione del servizio.

Politiche inconsistenti

Per attuare una politica su tutta la rete, può essere necessario la configurazione di migliaia di dispositivi e sistemi. La complessità delle reti di oggi rende molto difficile l'applicazione di un insieme coerente di regole di accesso, di sicurezza, di Quality of Service (QoS), e altre politiche a una molteplicità di dispositivi.

In conclusione, la mancata corrispondenza tra le esigenze del mercato e la capacità della rete tradizionale ha reso necessario la riprogettazione del network in modo che può fornire le competenze richieste per affrontare i problemi e i limiti delle reti attuali.

2.2 Introduzione al Software-Defined Networking

Il concetto di ***Software-Defined Networking*** definito da ONF rappresenta un'innovazione nell'attuale panorama informatico.

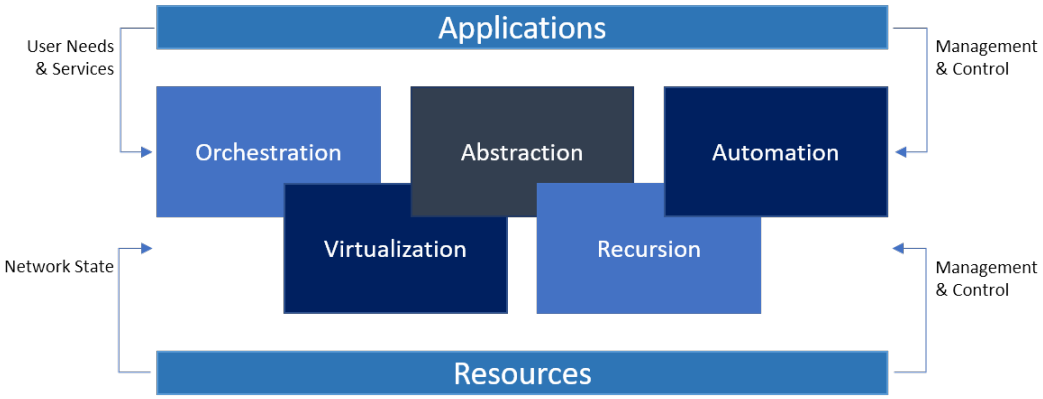


Figura 2: Definizione di SDN

Il paradigma SDN è un concetto innovativo che permette di trasformare i network tradizionali in piattaforme flessibili, intelligenti e programmabili, in grado di rispondere in tempo reale alle esigenze di larghezza di banda e alla natura dinamica delle moderne applicazioni.

L'obiettivo di SDN è ristrutturare l'architettura di networking, introducendo opportuni livelli di astrazione (Figura 2): la separazione logica delle funzioni di control e di forwarding della rete permette da un lato la possibilità di gestire via software tutta la rete da un unico controller, garantendo una maggiore scalabilità e standard di affidabilità e sicurezza della rete, dall'altro quella di utilizzare indifferentemente dispositivi prodotti dalle diverse aziende, senza preoccuparsi dei dettagli e di come funzionano i dispositivi. In questo modo, gli amministratori di rete possono aggiornare e aggiungere funzionalità, modificando il software di controllo e quindi garantendo una maggiore flessibilità ed una rapida evoluzione della rete stessa. Tutto ciò favorisce la nascita di una rete dinamica e non più legata al larghissimo numero di protocolli differenti attualmente utilizzati.

Il seguente elenco definisce le caratteristiche principali di SDN [2] con una breve descrizione.

Direttamente programmabile. Il controllo di rete è programmabile in modo diretto perché è disgiunto dalle funzioni di forwarding.

Agilità. La separazione delle funzioni di control e forwarding consente agli amministratori di regolare dinamicamente il flusso del traffico su tutta la rete per soddisfare le esigenze al loro variare.

Gestione centralizzata. A livello logico, l'intelligenza di rete è centralizzata in controller SDN basati su software che mantengono una visione globale della rete, che alle applicazioni e ai motori di policy appare come un singolo switch logico.

Programmazione configurata. SDN consente ai responsabili di rete di configurare, gestire, proteggere e ottimizzare le risorse di rete molto rapidamente tramite programmi SDN automatici e dinamici, che i responsabili possono scrivere in modo autonomo perché i programmi non dipendono da software proprietari.

Standard aperti e non vincolati dai fornitori. Se implementato attraverso standard aperti, SDN semplifica la progettazione e il funzionamento della rete perché le istruzioni sono fornite dai controller SDN invece che dai molteplici dispositivi e protocolli con specifiche diverse per ciascun fornitore.

Questa innovativa architettura è dinamica, facile da gestire, economicamente vantaggiosa e adattabile in qualsiasi circostanza. Considerate le già discusse limitazioni dell'attuale architettura di rete, SDN è la soluzione che permette l'interazione delle applicazioni con la rete in modo sufficientemente semplice da stimolare una vasta produzione di nuove applicazioni.

2.3 Software-Defined Networking vs Traditional Network

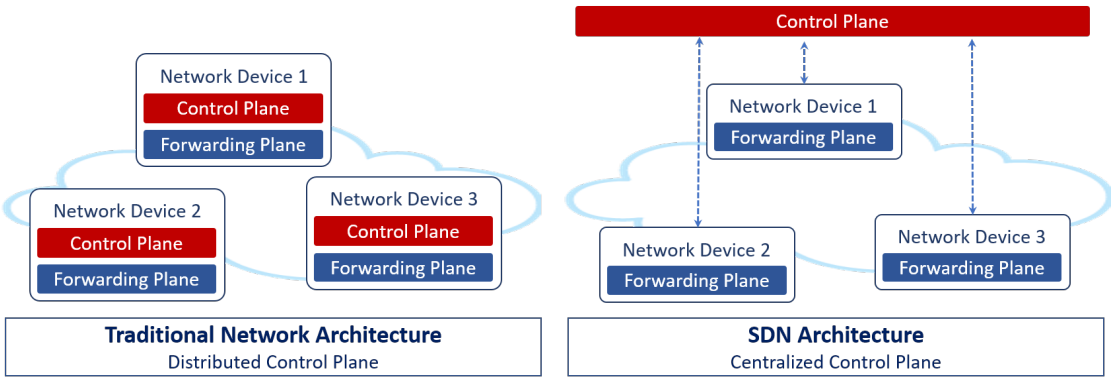


Figura 3: Rete tradizionale e SDN a confronto

La rete attuale si basa su un'architettura distribuita dove il controllo è confinato nei dispositivi di rete. Ciò significa che ogni dispositivo fisico di rete funziona in modo indipendente dagli altri, limitando l'accesso a software esterni e quindi ostacolando sviluppi specifici a supporto degli operatori e degli utilizzatori di rete.

In contrapposizione a questa, come illustrato nella Figura 3, con SDN il controllo della rete è disaccoppiato dall'hardware e posto in un controller, ovvero un software logicamente centralizzato, togliendo il vincolo che l'intelligenza del dispositivo debba risiedere nel dispositivo stesso. Riducendo così la complessità operativa e architetturale. Di conseguenza i nodi non decidono più in autonomia il loro comportamento, ma sono controllati da un'entità esterna.

Grazie alla centralizzazione logica, il controller mantiene una vista globale della rete: ciò permette di rilevare lo stato della rete, di ottimizzare le risorse e di regolare le politiche di forwarding dinamicamente in base ai cambiamenti di stato molto più velocemente rispetto un sistema distribuito. Inoltre mediante piattaforme modulari, interoperabili e basate su standard aperti è possibile controllare la propria strategia di rete e implementare una varietà di policy e cambiarle dinamicamente a seconda del variare dello stato e delle esigenze.

Si noti che il data plane è ancora completamente distribuito, mentre il control plane è logicamente centralizzato, ma può non essere fisicamente centralizzato: per questioni di performance, scalabilità e di affidabilità la logica centralizzata del controller SDN può essere distribuita su più controller fisici che cooperano al controllo della rete e delle applicazioni.

2.4 Architettura

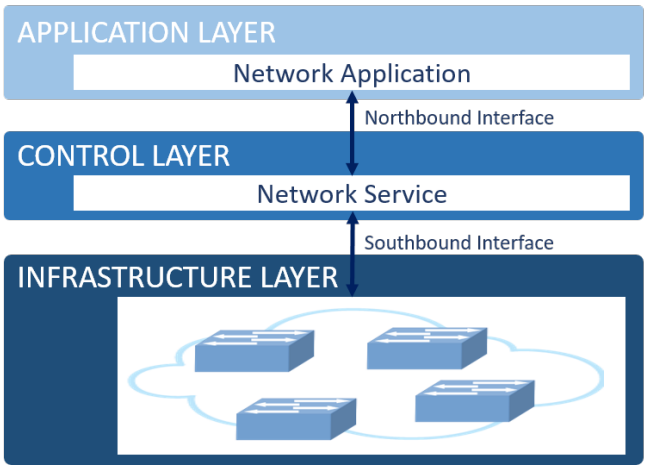


Figura 4: Architettura SDN

Come rappresentato nella Figura 4, dal punto di vista funzionale l'approccio SDN identifica tre componenti fondamentali [3]: *application layer*, *control layer* e *infrastructure layer*.

Alla base troviamo l'infrastruttura di rete, l' infrastructure layer, ovvero la struttura fisica, l'hardware, composta da switch, router, server e cavi di collegamento. Al livello intermedio, il control layer, troviamo il software SDN, che consente la gestione dinamica del network così da adattare il flusso dei dati alle necessità del momento. Infine, il livello superiore, l'application layer, è costituito dai servizi e dalle applicazioni aziendali che agiscono sull'infrastruttura fisica comunicando direttamente con il controller per mezzo di apposite API.

All'interno di questo schema, le applicazioni comunicano esplicitamente, direttamente e in maniera programmata le proprie specifiche di network al controller SDN attraverso un'interfaccia *northbound*, che concettualizza i dettagli di livello inferiore, come dati o funzioni. Mentre il controller interagisce con i nodi della rete fisica utilizzando un'interfaccia *southbound*, attraverso la quale è possibile monitorare inoltri, statistiche e notifiche degli eventi al fine di ottenere un'espressione diretta dei comportamenti e dei requisiti di rete a qualsiasi livello di astrazione.

2.4.1 Infrastructure Layer

L'infrastructure layer è il livello più basso che comprende i dispositivi fisici e rappresenta l'infrastruttura di rete. In seguito, nel capitolo 4, vengono illustrate le caratteristiche dell'emulatore di rete Mininet, il quale viene utilizzato per ricreare l'infrastructure layer in un contesto SDN, con lo scopo di eseguire dei test in un ambiente virtuale.

2.4.2 Southbound Interface

L'interfaccia definita southbound interface ha la funzione di definire la comunicazione tra il controller e i dispositivi hardware di rete. L'implementazione standard di riferimento è il protocollo OpenFlow, il quale viene approfondito nel capitolo 3. Questo protocollo viene implementato sia sul lato del controller che sul lato dei dispositivi e instaura così un canale di comunicazione end-to-end sicuro, grazie ai convenzionali protocolli crittografici come SSL o TLS. Dunque questa interfaccia rappresenta un punto critico dell'architettura SDN, in quanto non solo permette al controller di gestire in maniera dinamica e atomica il traffico di una rete, ma ne incrementa l'efficienza in termini di traffico e richieste di transito.

2.4.3 Control Layer

L'architettura SDN, rispetto a quella tradizionale, è dotata di un elemento aggiuntivo chiamato controller. Il controller è l'elemento intelligente della rete e rappresenta il fulcro di tale architettura. Come raffigurato nella Figura 5, il controller è un entità logica centralizzata che permette la comunicazione tra i dispositivi di rete al livello inferiore e le applicazioni software a livello superiore. Ciò

consente di percepire una rete come un unico sistema centralizzato coordinato dal controller e personalizzato mediante applicazioni utente, indipendentemente dalla topologia fisica di quest'ultima. Successivamente, nel capitolo 5, viene studiato e analizzato nel dettaglio il controller OpenDaylight, il quale viene utilizzato per la centralizzazione logica e la gestione della rete.

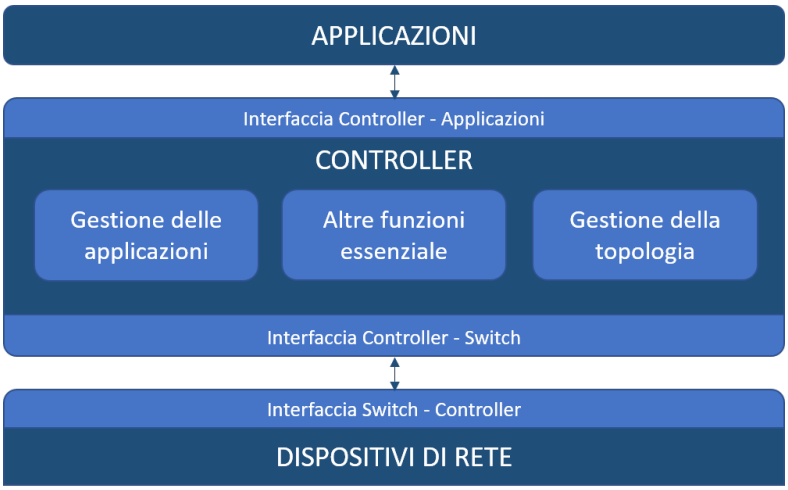


Figura 5: Elementi fondamentali di un controller

2.4.4 Northbound Interface

Le cosiddette northbound interface sono le interfacce di programmazione indispensabili per la comunicazione tra i controller SDN e i software applicativi. Queste sono necessarie in quanto determinate funzioni o servizi hanno il bisogno di acquisire informazione riguardo la struttura e il comportamento della rete,permettendo l'orchestrazione e l'automazione della rete con l'obiettivo di velocizzare l'innovazione. Diversamente dalla comunicazione controller-switch, attualmente non esiste un modello standard di API dominante per le interazioni northbound.

2.4.5 Application Layer

Per quanto riguarda le applicazioni SDN, si tratta di programmi che comunicano dinamicamente, esplicitamente e direttamente i requisiti di rete e il suo stato al controller SDN attraverso le northbound interface. Le applicazioni SDN, grazie alla centralizzazione logica fornita dal controller, hanno una visione globale di tutta la rete e erogano i servizi di networking agli utenti finali.

2.5 Vantaggi

La scelta di spostare la rete al di fuori del layer fisico, e quindi di centralizzare gran parte delle funzionalità nel controller SDN, consente di beneficiare di numerosi vantaggi quali [1]:

1. Controllo e gestione centralizzata dei dispositivi di rete anche provenienti da produttori diversi, realizzando così un'indipendenza dai vendor e dalle specificità dei dispositivi.

2. Possibilità di automatizzare molte delle operazioni di gestione sfruttando unicamente funzionalità software dato che il piano di controllo è svincolato dalla struttura fisica sottostante della rete.

3. Maggiore apertura all'innovazione grazie alla possibilità di aggiungere nuove funzionalità e servizi unicamente aggiornando il software, in modo semplice e rapido senza dover intervenire sui singoli dispositivi fisici per la riconfigurazione.

4. Possibilità di programmare le funzionalità di rete unicamente via software sfruttando linguaggi di programmazione di uso comune nel mondo dell'informatica.

5. Maggiore affidabilità e sicurezza della rete grazie alla possibilità di gestire la rete in modo centralizzato e automatizzato; inoltre diminuzione degli errori di configurazione grazie alla gestione completamente automatizzata.

6. Semplificazione sia delle tecnologie di costruzione di router e switch, sia della loro configurazione, poiché grazie all'astrazione del dispositivo sottostante, il dispositivo svolge solo funzioni di forwarding del flusso e si richiede l'utilizzo del solo protocollo di comunicazione con il controller SDN.

7. Possibilità di gestire la rete in modo granulare applicando politiche di gestione dinamiche, variabili a seconda dell'utente, della sessione o del servizio utilizzato.

8. Miglioramento dell'esperienza utente grazie alla possibilità di offrire servizi di rete personalizzati (routing, multicast, security, access control, bandwidth management, traffic engineering, quality of service, processor e storage optimization, policy managemen ecc.) adattando le funzionalità in run-time al tipo di servizio richiesto.

Questa serie di vantaggi ha portato all'interessamento, da parte di molte aziende del settore, nei confronti di questo nuovo tipo di approccio.

3 OpenFlow

Nel capitolo si esamina lo standard OpenFlow, che si è imposto come base per lo sviluppo di questa architetture di rete definita da SDN. Si definisce il funzionamento del protocollo e i conseguenti miglioramenti che ha apportato alla rete.

3.1 Descrizione del protocollo

Il protocollo ***OpenFlow*** è l'attuale interfaccia di comunicazione standard definita tra il controller e i dispositivi di inoltro che specifica come uno switch sia gestito da un unico dispositivo di controllo [4].

Figura 6: Logo di OpenFlow

Il protocollo OpenFlow definisce un modello di nodo generale e unificato da presentare alle applicazioni esterne, rendendo così gli strati più alti dell'architettura di rete SDN indipendenti dall'implementazione del particolare vendor e dalle tecnologie impiegate nel piano di forwarding. In questo modo è possibile monitorare e gestire il traffico della rete secondo determinate necessità. Inoltre OpenFlow utilizza il concetto di flusso per la re-direzione dei pacchetti. In questo contesto, per flusso si intende una sequenza unidirezionale di pacchetti aventi caratteristiche comuni, che attraversa il nodo entro un intervallo temporale, avendo sorgente e destinazioni fisse.

L'idea di base di OpenFlow è quella di rendere possibile la gestione di più switch attraverso il collegamento al controller, il quale semplifica la gestione dei flussi. In questo modo si introduce la possibilità di controllare la rete a livello atomico e permette, in maniera semplificata e potenzialmente più efficiente, la gestione dell'intera infrastruttura, la definizione delle politiche, la gestione del tipo di traffico e sopratutto di ottenere una reattività alle modifiche da parte dell'utente in tempo reale.

3.2 Switch OpenFlow

A differenza di quanto avviene in uno switch tradizionale, dove il control plane e il data plane coesistono nello stesso sistema, in uno switch basato su OpenFlow queste due funzioni sono separate: l'intero piano di controllo viene rimosso dal dispositivo di rete, rendendo il dispositivo dedito al solo trasporto dati, mentre

le funzioni di controllo (discovery, path computation, path setup, ecc.) vengono implementate in un'entità esterna alla rete.

Uno switch compatibile con OpenFlow, raffigurato nella Figura 7, contiene generalmente una o più *flow table* e una *group table*, le quali si occupano della funzione di inoltro dei pacchetti, e da un *OpenFlow secure channel*, il quale costituisce l'interfaccia che crea una connessione diretta tra switch e controller. Tramite questa interfaccia Il controller agisce sugli switch secondo le regole contenute all'interno delle flow table, che il protocollo installa e configura in base al comportamento che desidera ottenere. Tali regole che di fatto costituiscono la flow table sono dette *flow entry*.

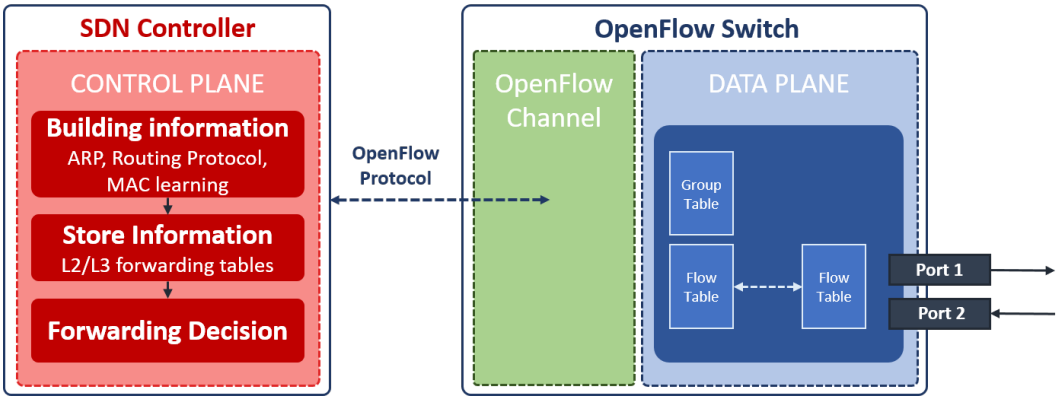


Figura 7: Principali componenti di uno switch OpenFlow

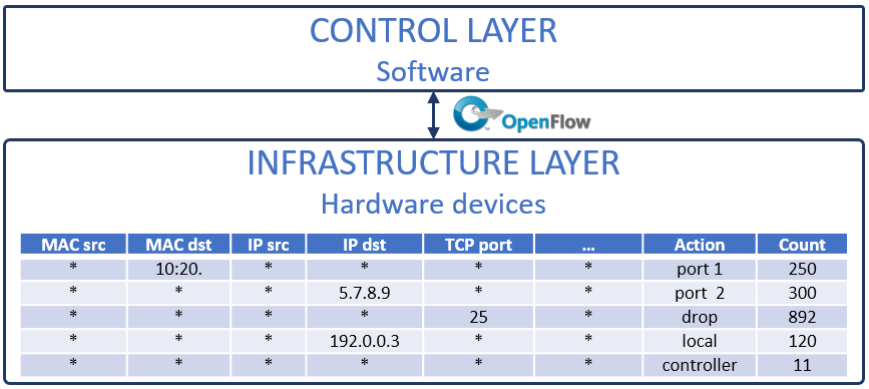
3.3 Funzionamento di OpenFlow

Il principio di funzionamento alla base di OpenFlow è quindi la separazione del software di controllo del traffico dai dispositivi di rete fisici. Attraverso il protocollo viene definita la comunicazione tra il controller e i dispositivi di rete mediante un set di regole primitive che consentono l'accesso diretto alle flow table dei dispositivi di rete per la gestione del forwarding plane.

Il forwarding plane di uno switch OpenFlow è costituito dalle flow table (Figura 8) all'interno delle quali a ogni sua voce è associata un'azione. Le azioni previste sono [5]:

- Inoltrare il flusso di pacchetti a una determinata porta. Ciò consente ai pacchetti di essere instradati attraverso la rete.
- Incapsulare e inoltrare il flusso di pacchetti al controller. I pacchetti sono incanalati nell'OpenFlow secure channel, dal quale sono inviati al controller. Tipicamente viene usato quando è necessario elaborare un nuovo flusso di dati.

- Scartare i pacchetti del flusso. Può essere usato per la sicurezza o ad esempio per frenare attacchi Denial of Service.
- Inoltrare il flusso di pacchetti come classica elaborazione dello switch.



MAC src	MAC dst	IP src	IP dst	TCP port	...	Action	Count
*	10:20.	*	*	*	*	port 1	250
*	*	*	5.7.8.9	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.0.0.3	*	*	local	120
*	*	*	*	*	*	controller	11

Figura 8: Esempio di una flow table

Questo protocollo di rete utilizza un insieme ben definito di regole di comunicazione per classificare il traffico di rete in flussi. Per flusso si intende una sequenza di pacchetti identificabili da uno o più etichette comuni (quali ad esempio: l'indirizzo IP, l'indirizzo MAC, il numero di porta, ecc.).

OpenFlow utilizza il concetto di flusso per determinare e gestire il traffico di rete. Infatti, questo concetto permette al protocollo di identificare specifiche porzioni di traffico con delle regole impostate in precedenza e immagazzinate in flow table in modo da far intraprendere azioni personalizzate.

Utilizzando il protocollo OpenFlow, il controller può aggiungere, aggiornare e cancellare flow entry nelle flow table (Figura 9). Ogni record della flow table contiene [5]:

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

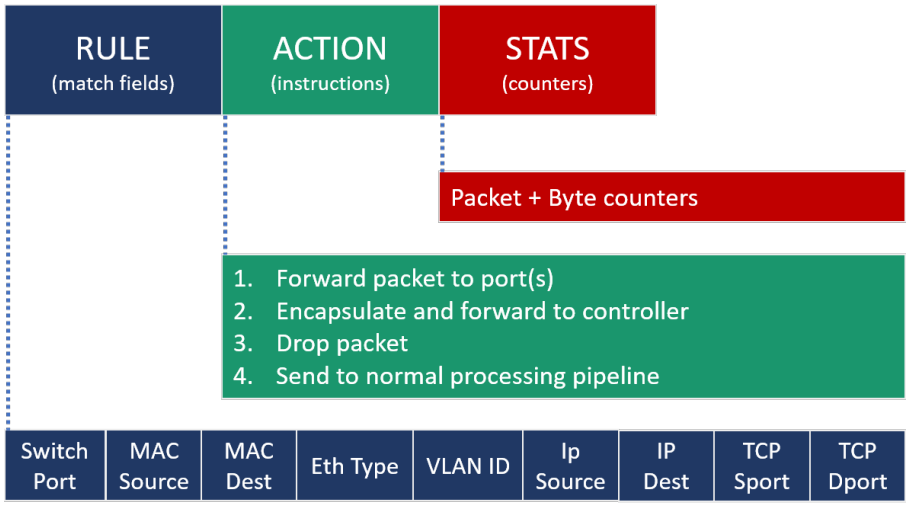


Figura 9: Descrizione dei record che costituiscono le flow table

Il funzionamento è semplice [5]: avviene una comunicazione sicura, tramite OpenFlow secure 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.

Quando uno switch OpenFlow riceve un nuovo pacchetto per il quale non vi sono regole attive nella propria tabella, 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 *FlowMod* o *PaketOut* (Figura 10), nel quale è inserita il tipo di regola da applicare. Infatti, il controller può decidere di scartarlo oppure di aggiungere un nuovo record alla flow table 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 dispositivo di rete.

Quindi per ogni pacchetto di un nuovo flusso entrante nello switch avviene uno scambio di messaggi tra il controller e lo switch, tramite questo protocollo. 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.

3.4 Messaggistica del protocollo

Per quanto riguarda i messaggi del canale controller-switch, il protocollo OpenFlow prevede un modello standard da applicare su ogni dispositivo. Sono supportate tre tipologie di messaggio [5]: *controller-to-switch*, *asynchoronous* e *symmetric*.

I messaggi controller-to-switch sono sempre inviati inizialmente dal controller e di norma non è richiesta necessariamente una risposta da parte dello switch. La maggior parte di questi messaggi hanno lo scopo di interrogare il dispositivo di commutazione riguardo alle proprie capacità di base (*features*) e specifiche tecniche (*read-state*), di modificare alcune impostazioni (*configuration*) o determinate voci nelle tabelle di flusso (*modify-state*), e infine di inviare e inoltrare pacchetti di dati ricevuti in precedenza per essere esaminati (*packet-out*).

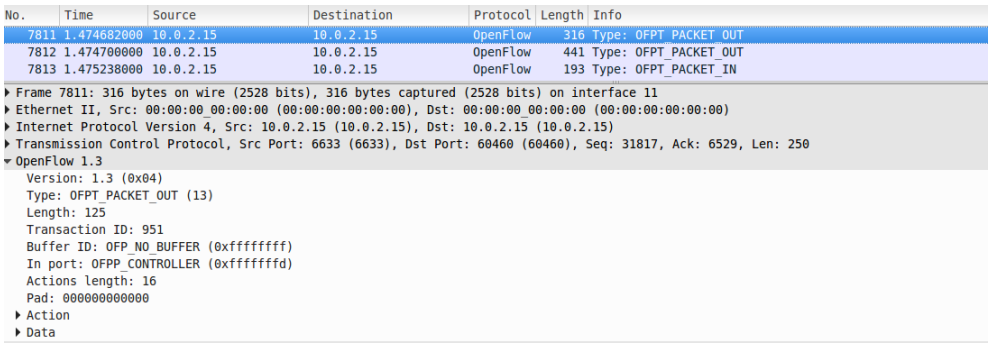


Figura 10: OpenFlow packet-out

I messaggi asynchoronous appartengono alla categoria "push", in altre parole vengono inviati dagli switch al controller senza che questo ne solleciti l'invio. La circostanza più importante riguarda l'invio al controller di un pacchetto per il quale non è stata trovata alcuna corrispondenza nelle tabelle di flusso (*packet-in*). Questo genere di messaggio può comprendere alcuni tipi di notifiche finalizzate a informare il controller di alcune modifiche effettuate dallo switch quali la rimozione di una voce dalla tabella di flusso (*flow-removed*), il cambio di una porta (*port-status*) o semplicemente il manifestarsi di un problema (*error*).

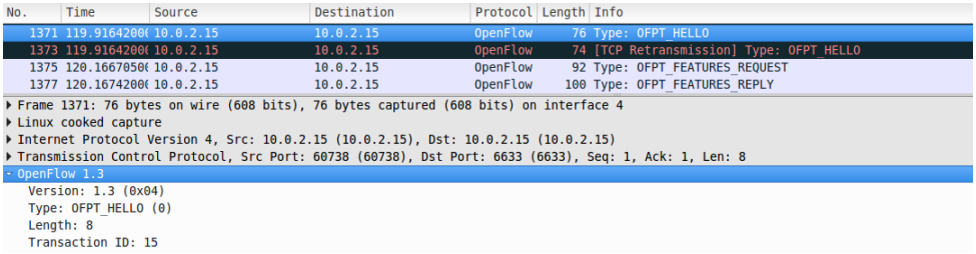


Figura 11: OpenFlow hello

I messaggi symmetric sono bilaterali e vengono inviati senza sollecitazione. Essi comprendono messaggi scambiati al fine di inizializzare la connessione tra i due dispositivi (*hello*), i cui dettagli del pacchetto sono mostrati nella Figura 11, o semplici verifiche che tale connessione sia ancora attiva (*echo*).

3.5 Benefici di una rete basata su OpenFlow

OpenFlow si rivela fin da subito un ottimo strumento per la facilità con cui si possono ideare e anche testare il funzionamento di nuovi protocolli. In poco tempo si riesce a raccogliere i risultati e le criticità di una determinata rete, e si può agire su di essa con modifiche strutturali che nelle reti tradizionali richiederebbero operazioni molto più laboriose mentre nel caso di OpenFlow sarà sufficiente modificare a livello software il controller.

Si hanno così molteplici vantaggi [6]: aumento di velocità delle reti, maggiore dinamicità e agevolazione nella personalizzazione della rete, efficienza energetica, maggiore sicurezza e ottimizzazione del sistema. OpenFlow include svariate funzioni: la modifica e l'automatizzazione delle regole di instradamento, la creazione di una rete virtuale dotata di nodi logici e la possibilità di monitorare il traffico accrescendo la sicurezza della propria rete. Inoltre il controllo del flusso basato su OpenFlow fornisce agli amministratori la possibilità di applicare policy a un livello di granularità molto alto, potendo definire regole per ogni sessione, per utenti diversi, per molteplici dispositivi o per differenti applicazioni, tutto in modo automatico e ad un alto livello di astrazione.

OpenFlow permette un uso flessibile della rete, nasconde la complessità delle singole parti dei dispositivi di rete e ne centralizza il controllo in modo virtualizzato, semplificando quindi notevolmente la gestione della rete.

4 Mininet

In questo capitolo viene esaminato l'emulatore di rete Mininet, capace di creare rapidamente, efficientemente e con risorse limitate una rete virtuale basata sul concetto di SDN.

4.1 Il simulatore

Nell'ambito dello sviluppo di OpenFlow e di SDN, si è manifestata presto l'esigenza di prevedere dei testbed[1] in modo tale da poter ottenere un riscontro alle idee che mano a mano si andavano aggiungendo e quindi di verificarle sul campo.

L'emulatore di rete ***Mininet*** [7] è stato sviluppato con lo scopo di riprodurre il funzionamento di una rete su uno o più calcolatori mediante la virtualizzazione[2], attraverso il quale è possibile eseguire test estesi utilizzando risorse limitate.

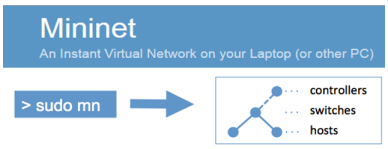


Figura 12: Logo di Mininet

Mininet è un progetto che fornisce API per simulare una rete multi-node su una singola macchina. Utilizza la virtualizzazione basata su processi, grazie a questa riesce a gestire molti host (fino a 4096) e switch su un singolo kernel del sistema operativo. In particolare permette la creazione e la gestione di switch OpenFlow, controller per gestire gli switch, e gli host per comunicare attraverso la rete simulata. L'emulatore Mininet 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[3] consentendo l'avvio di interfacce virtuali, connesse da cavi virtuali, nell'ambito dell'esecuzione di un singolo sistema operativo.

Come viene illustrato nell'appendice A, l'emulatore consente 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.

[1] Letteralmente banchi di prova, test per verificare il funzionamento.

[2] Consiste in un insieme di tecniche hardware, software o ibride che consentono di realizzare una risorsa virtuale identica per funzionalità ad una reale, suddividendo quest'ultima in diversi ambienti operativi isolati dalla macchina fisica sottostante.

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

4.2 Vantaggi

L'utilizzo di Mininet offre una serie di vantaggi [8]:

1. Velocità: l'avvio di una semplice rete richiede solo alcuni istanti, a seconda dell'hardware della macchina su cui viene eseguito.

2. Possibilità di creare topologie personalizzate: unico switch, topologie più ampie simili a internet, un data center, ecc.

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. Personalizzazione ed inoltro di pacchetti: gli switch di Mininet utilizzano il protocollo OpenFlow e quindi sono programmabili.

5. Portabilità: il software può essere eseguito su macchine linux oppure su una Virtual Machine (VM).

6. Facilità di utilizzo: si possono creare ed eseguire reti virtuali creando semplici, o anche complessi, script in linguaggio python.

7. Codice open-source: si può esaminare e modificare il codice sorgente.

8. Crescita continua di Mininet: è possibile interagire con gli sviluppatori per aggiornare il software.

4.3 Limitazioni

Oltre all'innumerevole serie di vantaggi, Mininet presenta anche alcune possibili limitazioni [8]:

1. Limitazioni nell'eseguire una rete su un unico sistema: le risorse dovranno essere bilanciate tra gli host della rete.

2. Mininet utilizza un unico kernel linux per tutti gli host virtuali: non è possibile eseguire software dipendente da kernel differenti.

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

4. 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. 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. 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.

La maggior parte di queste limitazioni non sono intrinseche di Mininet ed eliminarle è semplicemente una questione di codice.

4.4 Le potenzialità

Mininet, attraverso una molteplicità di funzioni e una facilità di utilizzo, permette la creazione di un prototipo funzionante di una rete su cui eseguire tutti i test necessari in maniera rapida e semplice. 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. Il superamento dei test con Mininet garantisce l'implementazione di funzioni anche sui dispositivi fisici.

Grazie all'utilizzo di altre tecnologie, 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, Mininet può essere classificato come un ottimo strumento per testare nuove implementazioni e sperimentazioni nell'ambito delle reti SDN.

5 OpenDaylight

Nel capitolo sono approfondite le caratteristiche di OpenDaylight, considerato uno dei più popolari tra i controller SDN. Inoltre viene descritta la sua struttura e analizzate alcune delle sue principali funzionalità.

5.1 Project

OpenDaylight [15] è un progetto collaborativo open-source, introdotto dalla Linux Foundation, che punta ad aiutare a velocizzare lo sviluppo della tecnologia a disposizione degli utenti con lo scopo di consentire un'adozione maggiormente diffusa di SDN.

Figura 13: Logo di OpenDaylight

Il progetto è un implementazione del concetto di SDN, con al suo interno una piattaforma modulare e flessibile che funge da controller.

Questo progetto svolge la funzione di fornire il controllo centralizzato e programmabile dei dispositivi fisici e virtuali con lo scopo di migliorare e velocizzare le prestazioni di rete. Inoltre, controlla i dispositivi attraverso protocolli standard open-source in modo tale da ridurre la complessità a livello operativo. Infine, ha lo scopo di garantire un alto livello di astrazione delle sue funzionalità, in modo da agevolare sviluppatori e ingegneri di rete nel creare nuove applicazioni per personalizzare l'installazione e la gestione della rete. Tutto ciò fornisce un approccio trasparente che favorisce l'innovazione e semplifica la gestione dell'infrastruttura sottostante.

5.2 Controller OpenDaylight

Il controller ODL [16] è un'infrastruttura open-source, multi-protocollo, scalabile, estendibile, modulare e flessibile che viene utilizzata per l'implementazione di una rete che si basa sul concetto di SDN.

La natura open-source del controller permette agli utenti di ridurre la complessità operativa, estendere il ciclo di vita dei dispositivi hardware esistenti a livello fisico e di abilitare nuovi servizi e funzionalità disponibili solo con SDN. L'infrastruttura del controller fornisce il controllo centralizzato e programmabile dei dispositivi e un alto livello di astrazione delle sue funzionalità, con lo scopo di

semplificare l'installazione e la gestione della rete definita da SDN, permettendo un miglioramento delle sue performace.

5.2.1 Architettura Microservices

L'infrastruttura del controller ODL è caratterizzato da un'*Architettura Microservices*. Per Architettura Microservices si intende una specifica struttura composta da un insieme di particolari servizi o protocolli, indipendenti fra di loro, ciascuno incentrato su un particolare aspetto, che comunicano tra di loro per realizzare applicazioni più complesse. In questo caso, ad esempio un servizio di Learning Switch o di AAA (Authentication, Authorization, Accounting), o un plugin che fornisce la connettività tra i dispositivi di rete via OpenFlow.

In questo documento la versione di ODL analizzata e utilizzata è la release 0.4.4-Beryllium-SR4 la cui struttura viene mostrata nella Figura 14. É stata scelta questa versione poiché offre nuovi servizi di rete, un miglioramento nella gestione di dati, un livello di astrazione elevato, una nuova interfaccia grafica e l'opportunità di integrare servizi e applicazioni esterni, come Grafana.

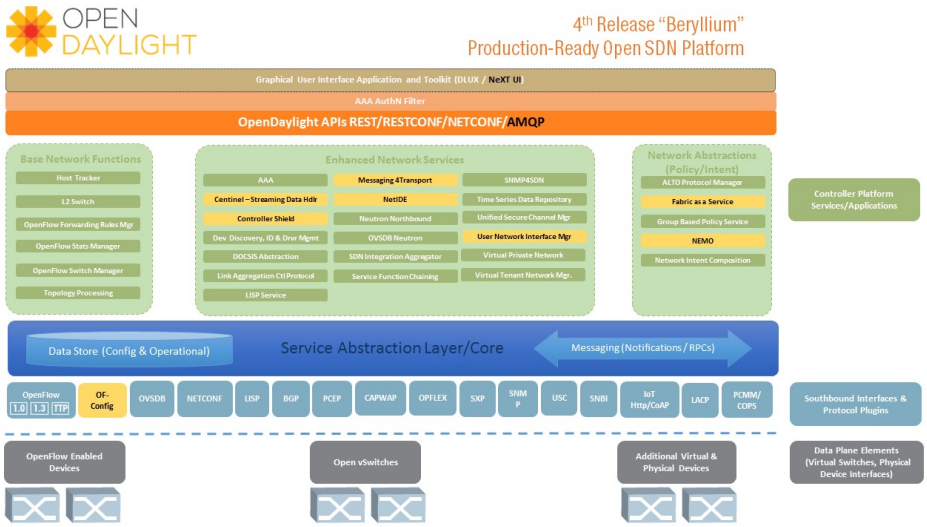


Figura 14: Architettura Microservices di ODL Beryllium

Il controller ODL è una implementazione del concetto di SDN. Infatti, come si può notare nella figura sovrastante, l'infrastruttura del controller segue la struttura a tre livelli di SDN.

Le applicazioni di livello superiore comunicano esplicitamente, direttamente e in maniera programmata le proprie specifiche di network al controller tramite le northbound API, che concettualizzano i dettagli di livello inferiore. Per la comunicazione ODL supporta: il framework Open Service Gateway Initiative (OSGi)

e le bidirezionali API REST. Il framework OSGi viene utilizzato principalmente per le applicazioni che si trovano nella stessa rete del controller, mentre le API REST (web-based) vengono utilizzate per le applicazioni che non sono nella stessa network, ovvero funzionano anche su VM diverse. La logica di business e gli algoritmi fanno parte del livello applicativo. Le applicazioni utilizzano il controller per raccogliere informazioni sulla rete, per eseguire algoritmi e analisi, per la gestione e il monitoraggio della network e quindi per l'orchestrazione delle nuove regole in tutta la rete.

La southbound interface del controller è in grado di supportare diversi protocolli, come plugin indipendenti, tra i quali OpenFlow 1.0, OpenFlow 1.3, BGP, ecc. Questi moduli sono collegati dinamicamente al Service Abstraction Layer (SAL), che è il cuore del design modulare del controller e permette di supportare più protocolli sulla southbound interface. Il livello SAL ha lo scopo di soddisfare il servizio richiesto, indipendentemente dal protocollo di base utilizzato tra il controller e le periferiche di rete, fornendo così un alto livello di astrazione e indipendenza dall'infrastruttura fisica

5.2.2 Installazione

Il controllore ODL è una piattaforma Java Virtual Machine (JVM), quindi può essere installato e eseguito su qualsiasi piattaforma hardware e sistema operativo fintanto che supporti Java. Quindi, per prima cosa, è necessario installare JVM[4]. Successivamente scegliere la release Beryllium dal sito ufficiale [17], scaricare il relativo file zip ed estrarlo. Una volta estratto, possiamo interagire con il controller attraverso la console karaf, mostrata nella Figura 15, la quale è possibile avviare eseguendo le seguenti istruzioni:

```
cd distribution-karaf-0.4.4-Beryllium-SR4
cd bin
./karaf
```

Figura 15: Console karaf di ODL

[4]Si rende necessario l'utilizzo della versione 1.7 se si lavora con la release Beryllium di ODL.

5.3 Feature ODL

Le feature sono servizi e/o protocolli che si possono includere nella piattaforma software del controller. Una volta avviata la console karaf possiamo visualizzare la lista completa delle feature in ordine alfabetico digitando:

```
feature:list
feature:list --installed
```

5.3.1 DLUX

La feature DLUX [18] fornisce un'interfaccia grafica web-based messa a disposizione da ODL per semplificare la visione globale e la gestione della rete tramite il controller. La funzione può essere installata, nel controller ODL, digitando le seguenti istruzioni dalla console karaf:

```
feature:install odl-dlux-core
feature:install odl-dlux-node
feature:install odl-dlux-yangui
feature:install odl-dlux-yangvisualizer
```

L'interfaccia grafica DLUX raccoglie, dal database ODL, le informazioni riguardo la topologia di rete, le statistiche dei flussi e le caratteristiche dei nodi per poi visualizzarle in maniera chiara, semplice e intuitiva.

Per accedere all'interfaccia DLUX, aprire un browser e immettere l'URL e le credenziali di accesso.

```
URL: http://<controller_IP>:8181/index.html
USER: admin
PASSWORD: admin
```

I seguenti sottoparagrafi elencano le sezioni che compongono l'interfaccia DLUX e le rispettive funzioni.

TOPOLOGY

In questa sezione è possibile visualizzare una rappresentazione grafica della topologia di rete su cui si sta lavorando (Figura 16). In aggiunta, posizionando il mouse su un host, un link o uno switch è possibile visualizzare le caratteristiche delle porte di origine e di destinazione di quel determinato elemento.

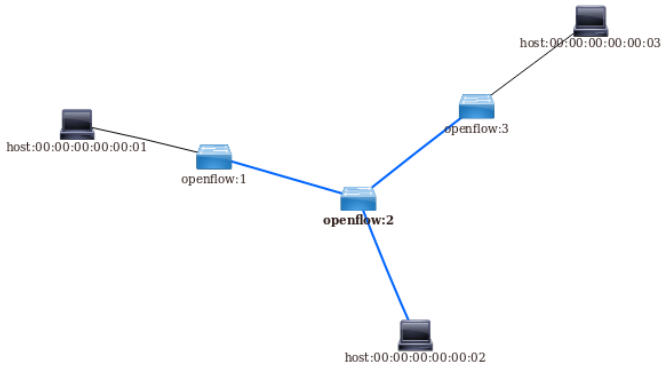


Figura 16: Visualizzazione della rete attraverso l'interfaccia DLUX

NODES

La sezione consente di visualizzare le informazioni relative ai nodi (Figura 17) e alle porte (Figura 18) degli switch presenti nella rete. Inoltre questa sezione consente di visualizzate anche le relative statistiche.

Node Id	Node Name	Node Connectors	Statistics
openflow:2		4	Flows | Node Connectors
openflow:3		3	Flows | Node Connectors
openflow:1		3	Flows | Node Connectors

Figura 17: Lista degli switch appartenenti alla rete

Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
openflow:2:3	165	274	27711	47959	0	0	0	0	0	0	0	0
openflow:2:2	163	272	27094	47702	0	0	0	0	0	0	0	0
openflow:2:1	17	373	1298	67651	0	0	0	0	0	0	0	0
openflow:2:LOCAL	34	8	2652	648	34	0	0	0	0	0	0	0

Figura 18: Informazioni relative alle interfacce di uno switch

YANGUI

In questa sezione è possibile visualizzare un'interfaccia grafica (Figura 19) attraverso la quale è possibile interagire facilmente con il controller, mediante le API messe a disposizione da ODL, per ottenere, modificare e cancellare risorse.

Figura 19: YangUI

5.3.2 L2Switch

Attraverso la feature di L2Switch viene abilitata la funzione di learning switch negli dispositivi che fanno parte della rete monitorata dal controller ODL. La funzione di learning switch è il processo attraverso il quale lo switch gestisce l'instradamento del pacchetto associando l'indirizzo MAC alla porta di provenienza.

Per installare la funzione di L2switch digitare nella console karaf il seguente comando:

```
feature:install odl-l2switch-switch
```

Il funzionamento è semplice: quando lo switch riceve un pacchetto di cui non conosce la destinazione, ovvero non trova nessuna associazione tra la destinazione e un record contenuto nelle flow table, invia il pacchetto al controller attraverso il protocollo OpenFlow. Il controller riceve dallo switch il pacchetto che non ha riscontri nella flow table, lo elabora e decide se aggiungere la flow entry relativa o scartare il pacchetto. Attraverso questa funzione il traffico di rete viene ottimizzato e velocizzato.

5.3.3 Time Series Data Repository

Time Series Data Repository (TSDR) è un framework per la raccolta, l'archiviazione, il mantenimento e l'interrogazione di Time Series Data[5] in ODL. TSDR ha il compito di raccogliere i diversi dati temporali e memorizzare le informazioni ottenute in un datastore TSDR.

[5]Una serie di dati ordinati in ordine cronologico (time stamps) che esprimono la dinamica di un certo fenomeno nel tempo. Questo tipo di dati da all'utente una visione in tempo reale del fenomeno e prevede il suo andamento futuro.

Per abilitare la funzione del framework TSDR nel controller ODL digitare la seguente istruzione:

```
feature:install odl-tsdr-core
```

Quindi, TSDR è un framework in grado di raccogliere informazioni e memorizzarle in un datastore. Attualmente, sono sopportati tre tipi di datastore:

- HSQLDB database relazionale.

  ```
  feature:install odl-tsdr-hsqldb
  ```

- HBase NoSQL database.

  ```
  feature:install odl-tsdr-hbase
  ```

- Cassandra NoSQL database.

  ```
  feature:install odl-tsdr-cassandra
  ```

TSDR offre anche la possibilità di collezionare informazioni relative a diversi tipi di protocolli, alcuni dei quali OpenFlow, NetFlow, SFlow, SNMP e Syslog. Si possono abilitare queste funzionalità attraverso i seguenti comandi:

```
feature:install odl-tsdr-openflow-statistics-collector
feature:install odl-tsdr-netflow-statistics-collector
feature:install odl-tsdr-sflow-statistics-colletor
feature:install odl-tsdr-snmp-data-collector
feature:install odl-tsdr-syslog-collector
```

Infine, per ottenere informazioni, possiamo interrogare il database TSDR attraverso semplici query costituite nel seguente modo:

```
tsdr:list <categoria>
```

Dove `<categoria>` corrisponde a una delle seguenti:

`FLOWSTATS`: raccoglie le informazioni relative a specifici flow.

`PORTSTATS`: colleziona i dati relativi a ogni interfaccia degli switch.

`FLOWTABLESTATS`: raccoglie i dati relativi a determinate flow table.

`NETFLOW`: raccoglie le informazioni relative al protocollo NetFlow.

Con le funzionalità fornite da TSDR, gli amministratori di rete possono utilizzare delle applicazioni integrate a questo framework (ad esempio Grafana, tool per la visualizzazione di dati temporali) che acquisiscono le informazioni ottenute, le elaborano e generano analisi di prestazioni della rete automatizzate.

5.3.4 RESTCONF

RESTCONF [21] è un protocollo che si basa sull'utilizzo di HTTP e fornisce un'interfaccia di programmazione per l'accesso e la configurazione dei dati definiti dal modello YANG[6], utilizzando il concetto di datastore descritto nel protocollo di rete NETCONF[7]. Il protocollo RESTCONF fornisce un framework applicativo che utilizza i metodi HTTP per inviare (creazione e/o aggiornamento), effettuare query, modificare e cancellare risorse[8].

Il controller ODL utilizza questo protocollo per la comunicazione northbound, allo scopo di consentire alle applicazioni web l'accesso alle risorse ODL. Per abilitare questa funzione è necessario digitare nella console karaf la seguente istruzione:

```
feature:install odl-restconf-all
```

Il protocollo RESTCONF [22] fornisce le REST-like API per manipolare il modello dati YANG e fornisce l'accesso al datastore del controller. Nel controller ODL esistono due tipi di datastore:

1. Config: contiene i dati inseriti tramite il controller.

2. Operational: contiene altri tipi di dati mantenuti dal sistema.

I dati di configurazione e di stato mantenuti nei database sono esposti come risorse, che possono essere recuperate attraverso il metodo GET di HTTP. Mentre i dati di configurazione possono essere aggiornati e modificati mediante i metodi PUT, POST e DELETE forniti dal protocollo HTTP. Le risorse RESTCONF sono accessibili tramite un set di URI, descritti brevemente nei seguenti sottoparagrafi. Le richieste e le risposte sono in formato XML o JSON.

[6]Yet Another Next Generation, è un linguaggio di modellazione per la definizione di risorse, utilizzato per modellare sia i dati di configurazione che lo stato degli elementi di rete.

[7]Network Configuration Protocol, è un protocollo basato su XML utilizzato per gestire la configurazione di apparecchiature di rete

[8]Fonti di informazioni, identificate tramite URL logici.

OPTION

Il metodo OPTION rappresenta una richiesta di informazioni inerenti alle opzioni di comunicazione disponibili sul canale definito dal request-URI. Queste opzioni sono riferite alla risorsa da utilizzare o alle capacità del server e sono restituite in formato XML.

```
URI: /restconf
```

GET

Il metodo GET è usato per recuperare le informazioni localizzate dal request-URI e ottenere così il contenuto della risorsa indicata nella richiesta. In questo caso restituisce i dettagli relativi alla risorsa invocata memorizzata nel datastore config. Nella Figura 20 viene rappresentato il processo di tale richiesta.

```
URI: /restconf/config/<moduleName>:<nodeName>
```

In questo altro caso restituisce le informazioni relative alla risorsa richiesta archiviata nel datastore operational.

```
URI: /restconf/operational/<moduleName>:<nodeName>
```

Dove `/restconf/config/` e `/restconf/operational/` identifica il database nel quale si sta lavorando per ottenere le risorse, `<moduleName>` è il nome del modello YANG, mentre `<nodeName>` è il nome di un nodo nel modulo di livello superiore.

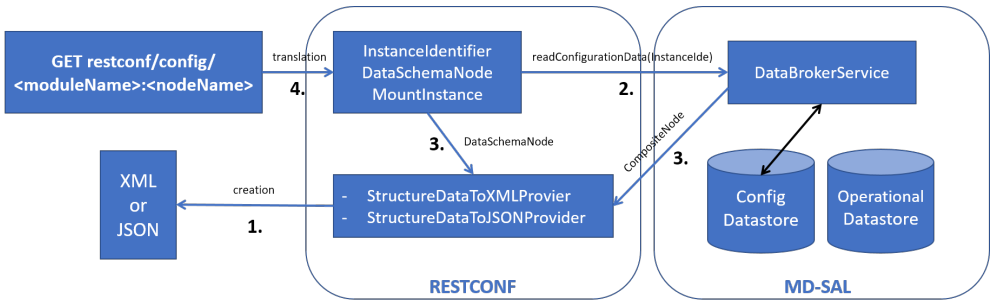


Figura 20: Procedimento per la richiesta di una risorse tramite il metodo GET

PUT

Il metodo PUT di HTTP crea, aggiorna o modifica le risorse in relazione al request-URI nel datastore di configurazione e restituisce lo stato dell'operazione[9]. La Figura 21 mostra il procedimento di questa operazione.

```
URI: /restconf/config/<moduleName>:<nodeName>
```

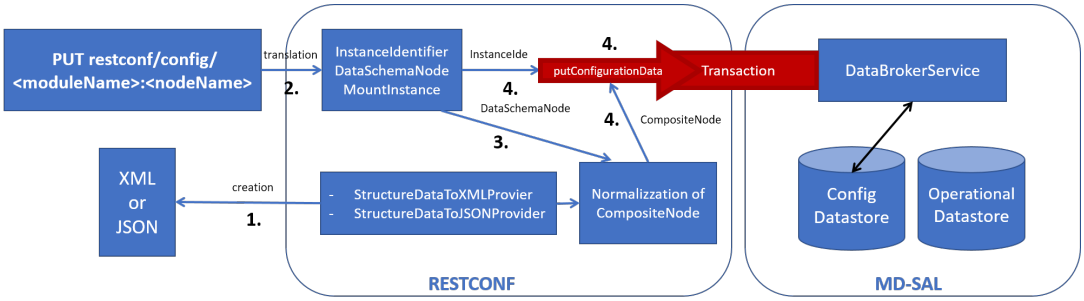


Figura 21: Procedimento per la modifica di una risorsa tramite il metodo PUT

POST

Il metodo POST si utilizza per definire una risorsa o ampliarne una già esistente identificata tramite il request-URI. Dopodiché restituisce lo stato dell'operazione.

```
URI: /restconf/config/<moduleName>:<nodeName>
```

DELETE

Il metodo DELETE richiede l'eliminazione della risorsa specificata dal request-URI, localizzata nel datastore che contiene i dati di configurazione, e infine restituisce lo stato dell'operazione.

```
URI: /restconf/config/<moduleName>:<nodeName>
```

[9]Gli HTTP Status Codes sono utilizzati per avere informazioni sullo stato dell'operazione. Questi hanno la funzione di fornire al client delle informazioni di stato riguardo all'esito della ricezione della richiesta (200 OK, 201 Created, 500 Internal Error, ecc.).

5.4 Potenzialità di OpenDaylight

Con l'introduzione del software ODL si sono manifestati significativi miglioramenti in termini di prestazioni, scalabilità e funzionalità nel mondo del networking. Il controllo logicamente centralizzato e programmabile della rete, la gestione semplificata tramite interfacce web-based, l'utilizzo di protocolli open-source e l'introduzione di un altro livello di astrazione hanno portato a un cambiamento vantaggioso nella gestione di una rete definita da SDN. Infatti, ODL propone nuovi servizi di rete automatizzati, ottimizzazione delle risorse di rete e garantisce la possibilità di integrare facilmente nuovi applicazioni e protocolli esterni. Inoltre il controller ODL offre la visibilità e il controllo logicamente centralizzato di rete. Per di più, è considerata la piattaforma ideale per ottenere molteplici opzioni per la politica e la configurazione interna con lo scopo di rendere la rete un ambiente omogeneo.

In conclusione, la piattaforma ODL, modulare ed estensibile, supporta una vasta gamma di casi d'uso ed elimina significative barriere nell'implementazione di soluzioni SDN.

6 Implementazione

In questo capitolo vengono eseguiti dei test, in un ambiente virtuale, con lo scopo di capire come gestire e monitorare una rete centralizzata SDN con OpenDaylight che funge da controller. L'obiettivo è quello di rendere la rete efficiente, velocizzare il traffico di pacchetti, ottimizzare le sue performance e bloccare il traffico secondo determinate caratteristiche del flusso.

6.1 Impostazioni ambiente virtuale

La VM utilizzata per lo scopo di questo documento ha come sistema operativo Ubuntu 16.10 LTS e possiede le seguenti caratteristiche:

- CPU: 2 Cores
- RAM: 4 GB
- Storage: 30 GB
- JVM: 1.7

Inoltre, su questa macchina, sono istallati i seguenti software: l'emulatore di rete Mininet[10] e il controller ODL[11].

Impostazioni OpenDaylight

Una volta avviato il controller, le feature ODL principali da installare per l'obiettivo di questo esempio sono:

```
feature:install odl-l2switch-switch
feature:install odl-restconf-all
feature:install odl-dlux-all
feature:install odl-tsdr-core
```

6.2 Creazione di una rete virtuale

In questo esempio si vuole creare, attraverso l'emulatore di rete Mininet, una semplice rete lineare composta da dieci switch, ognuno dei quali è collegato a un dispositivo terminale (Figura 22).

[10] Versione Mininet Beta 2.2.2b2.
[11] Versione OpenDaylight 0.4.4-Beryllium-SR4.

Gli switch che compongono la rete devono essere di tipo Open vSwitch[12] (OVS), ovvero devono supportare il protocollo OpenFlow versione 1.3. Inoltre questi devono essere connessi a un controller remoto specificato, in questo caso al controller ODL, con cui comunicano attraverso la porta 6633.

```
sudo mn --topo linear,10 --mac
    --controller=remote,ip=<controllerIP>,port=6633
    --switch ovs,protocols=OpenFlow13
```

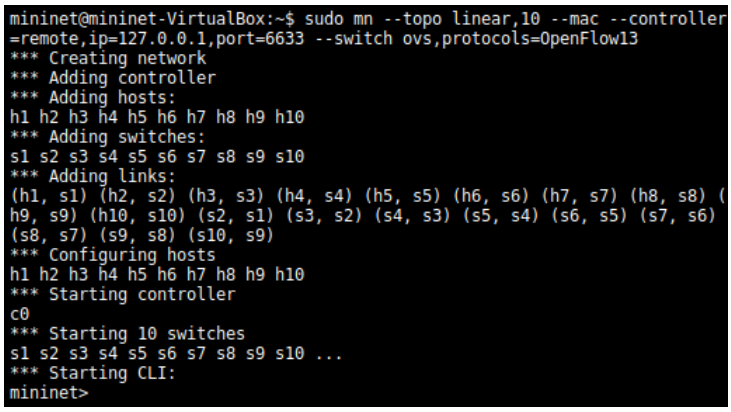


Figura 22: Esempio di una rete virtuale emulata da Mininet

Attraverso il comando *pingall* (Figura 23), inserito dalla console Mininet, se questo non da errore, possiamo verificare la comunicazione tra il controller e gli switch della rete virtuale emulata.

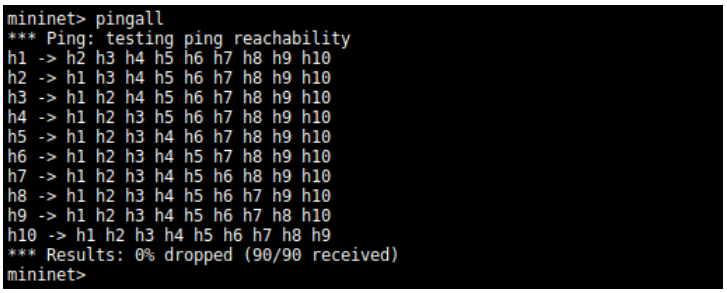


Figura 23: Esecuzione di un ping fra tutti gli host virtuali

[12]Open vSwitch è uno switch virtuale nato per gestire gli ambienti di virtualizzazione multi-server che può essere usato, in concomitanza con OpenFlow, come control stack per gli switch hardware.

6.2.1 Visualizzazione della rete in DLUX

Attraverso l'interfaccia grafica DLUX possiamo visualizzare la rappresentazione grafica della rete virtuale generata nel precedente paragrafo.

```
URL: http://<controller_IP>:8181/index.html
```

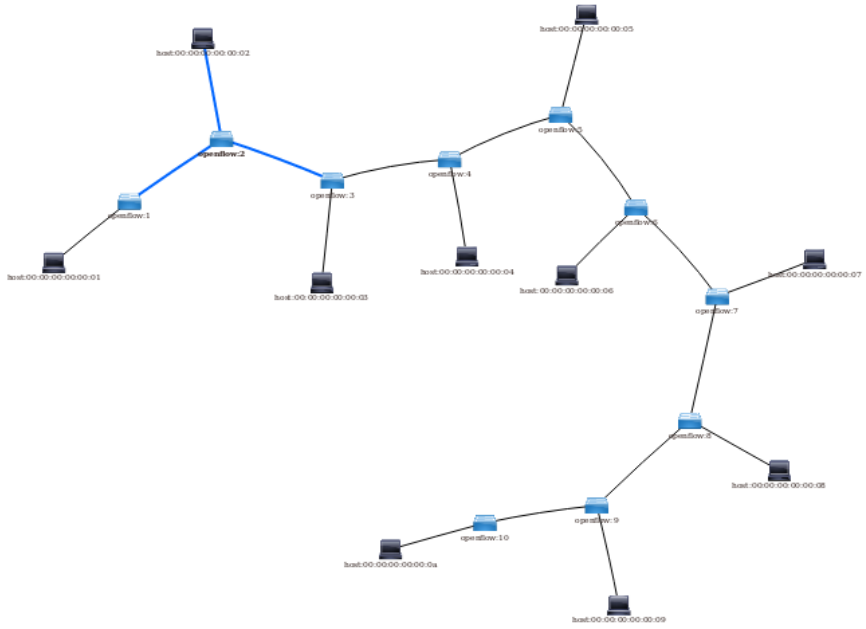


Figura 24: Visualizzazione della topologia di rete virtuale tramite DLUX

6.3 Lavorare con TSDR

TSDR fornisce un framework per la corretta raccolta e memorizzazione di dati temporali in un datastore. Sono offerti da TSDR tre diverse tipologie di datastore: HSQLDB, HBase e Cassandra. In questo esempio si è scelto di utilizzare il datastore HSQLDB[13].

```
feature:install odl-tsdr-hsqldb
```

I file HSQLDB di archiviazione vengono memorizzati automaticamente nella directory `<karaf folder>/tsdr/`. Una volta inizializzato il datastore HSQLDB è possibile testare il suo funzionamento attraverso la console karaf del controller ODL.

[13]Database leggero e facile da utilizzare, il quale non ha bisogno di configurazioni iniziali.

6.3.1 Testare il funzionamento del protocollo OpenFlow

Il funzionamento del protocollo OpenFlow può essere testato attraverso il framework TSDR. Per prima cosa, abilitare la feature di TSDR che permette di collezionare i pacchetti di questo protocollo.

```
feature:install odl-tsdr-openflow-statistics-collector
```

Per visualizzare le informazioni relative ai pacchetti OpenFlow sono disponibili tre diverse query, eseguibili dalla console karaf.

FLOSTATS

Questa query permette di visualizzare la lista delle caratteristiche dei singoli flussi installati, tramite il protocollo OpenFlow, nelle flow table degli switch e ne specifica il comportamento e il lasso temporale. Un esempio è raffigurato nella Figura 25.

```
tsdr:list FLOWSTATS
```

```
opendaylight-user@root>tsdr:list FLOWSTATS
[NID=openflow:2][DC=FLOWSTATS][MN=PacketCount][RK=Node:openflow:2,Table:0,Flow:L2switch-121][TS=1485451000594][44]
[NID=openflow:3][DC=FLOWSTATS][MN=PacketCount][RK=Node:openflow:3,Table:0,Flow:L2switch-315][TS=1485451000604][2]
[NID=openflow:3][DC=FLOWSTATS][MN=PacketCount][RK=Node:openflow:3,Table:0,Flow:L2switch-314][TS=1485451000604][536]
[NID=openflow:1][DC=FLOWSTATS][MN=PacketCount][RK=Node:openflow:1,Table:0,Flow:#UF$TABLE*0-404][TS=1485451000609][205]
[NID=openflow:2][DC=FLOWSTATS][MN=PacketCount][RK=Node:openflow:2,Table:0,Flow:L2switch-311][TS=1485451000594][266]
[NID=openflow:2][DC=FLOWSTATS][MN=PacketCount][RK=Node:openflow:2,Table:0,Flow:L2switch-313][TS=1485451000597][6]
[NID=openflow:2][DC=FLOWSTATS][MN=PacketCount][RK=Node:openflow:2,Table:0,Flow:L2switch-312][TS=1485451000597][271]
[NID=openflow:3][DC=FLOWSTATS][MN=PacketCount][RK=Node:openflow:3,Table:0,Flow:L2switch-120][TS=1485451000604][205]
[NID=openflow:3][DC=FLOWSTATS][MN=ByteCount][RK=Node:openflow:3,Table:0,Flow:L2switch-120][TS=1485451000604][17425]
[NID=openflow:1][DC=FLOWSTATS][MN=ByteCount][RK=Node:openflow:1,Table:0,Flow:L2switch-119][TS=1485451000609][4125]
```

Figura 25: Statistiche relative alla categoria FLOWSTATS

Dove `[NID]`, node identifier, identifica il nome del nodo interessato, `[DC]`, data category, specifica la categoria TSDR su cui si sta lavorando, `[MN]`, metric name, definisce il nome della metrica, `[RK]`, record key, descrive i dettagli dei flussi installati in una determinata tabella e `[TS]`, time stamp, indica l'ordine temporale. Nella Tabella 1 sono esaminate le metriche che sono raccolte da TSDR e salvate nel database per la categoria `FLOWSTATS`.

Metrica	Descizione
ByteCount	Numero di byte per flusso
PacketCount	Numero di pacchetti per flusso

Tabella 1: Metriche relative alla categoria FLOWSTATS

PORTSTATS

La seguente query è utilizzata per visualizzare la lista delle informazioni relative al comportamento di un pacchetto che attraversa una determinata porta di un nodo appartenente alla rete virtuale. Viene mostrato un esempio nella Figura 26.

```
tsdr:list PORTSTATS
```

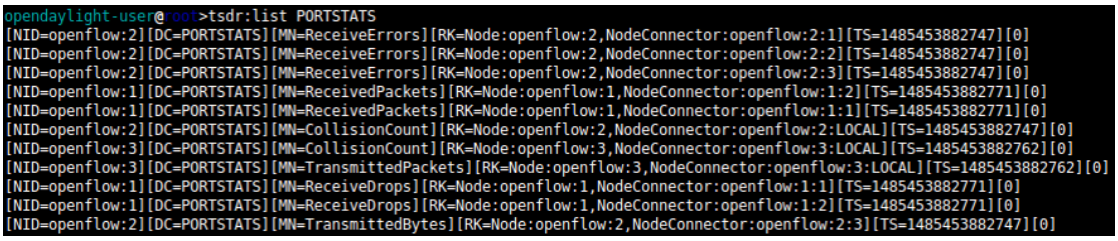


Figura 26: Statistiche relative alla categoria PORTSTATS

Di seguito, nella Tabella 2, sono analizzate le metriche che sono raccolte da TSDR e salvate nel database per la categoria `PORTSTATS`.

Metrica	Descizione
TransmittedPackets	Numero dei pacchetti trasmessi
ReceivedPackets	Numero dei pacchetti ricevuti
TransmittedBytes	Numero dei byte trasmessi
ReceivedBytes	Numero dei byte ricevuti
TransmitErrors	Numero di errori nella trasmissione
ReceiveErrors	Numero degli errori nella ricezione
CollisionCount	Numero di collisioni
TransmitDrops	Numero di pacchetti scartati nella trasmissione
ReceiveDrops	Numero di pacchetti scartati in ricezione

Tabella 2: Metriche relative alla categoria PORTSTATS

FLOWTABLESTATS

Questa utima query è utilizzata per ricevere informazioni relative alle singole tabelle di routing dei nodi presenti nella rete (Figura 27).

```
tsdr:list FLOWTABLESTATS
```

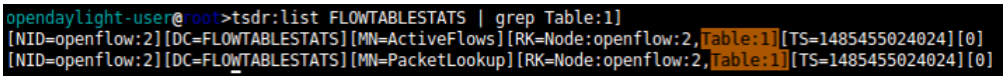


Figura 27: Statistiche relative alla categoria FLOWTABLESTATS

Di seguito, nella Tabella 3, la descrizione delle metriche che sono raccolte da TSDR e salvate nel database per `FLOWTABLESTATS`.

Metrica	Descizione
ActiveFlows	Numero di flussi attivi
PacketMatch	Numero di pacchetti identificati
PacketLookup	Numero di pacchetti non identificati

Tabella 3: Metriche relative alla categoria FLOWTABLESTATS

6.3.2 Verificare il funzionamento di L2Switch

Il funzionamento del processo di learning switch si può verificare attraverso il framework TSDR, il quale raccoglie informazioni inerenti al protocollo NetFlow[14]. In primo luogo è necessario abilitare il protocollo di analisi negli switch che compongono la rete. Inoltre è necessario creare un bridge attraverso il quale vengono inviati i record del protocollo NetFlow dagli switch OVS al controller ODL.

```
ovs-vsctl - - set Bridge br0 netflow=@nf -- --
    id=@nf create NetFlow targets=/"27.0.0.1:2055"/
    active-timeout=60
```

Dopodichè è necessario installare, nella console karaf del controller ODL, la feature che ha la funzione di collezionare le informazioni inerenti a questo determinato protocollo.

```
feature:install odl-tsdr-netflow-statistics-collector
```

Infine, per verificare il giusto comportamento di learning switch, è necessario creare del traffico di rete. Per realizzare ciò, come possiamo vedere in Figura 28, è stato eseguito un *ping* tra due determinati host della rete.

[14]NetFlow è un protocollo per il monitoraggio passivo di flussi di rete, anche ad alta velocità.

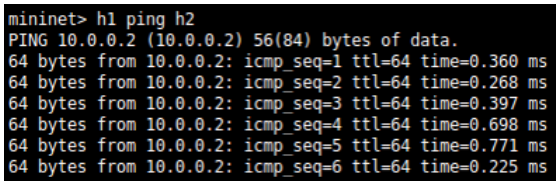


Figura 28: Esecuzione di un ping tra due host della rete

Per visualizzare le informazioni raccolte da TSDR digitare nella console:

```
tsdr:list NETFLOW
```

Attraverso questa query è possibile visualizzare le statistiche relative al protocollo NetFlow. Nella Figura 29 si mette in evidenza lo scambio diretto di pacchetti tra i due host e la velocità con cui viene effettuato.

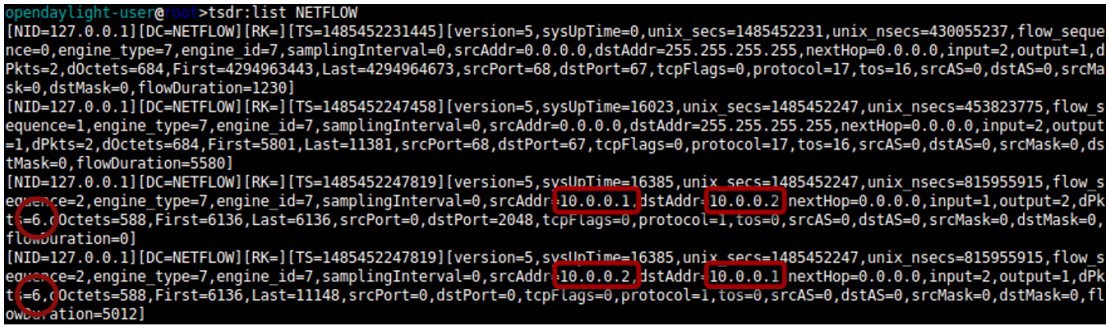


Figura 29: Statistiche relative alla categoria NETFLOW

Nella Tabella 4 sono elencate alcune delle metriche del protocollo NetFlow che sono raccolte da TSDR e memorizzate nel database.

Metrica	Descizione
Flow_sequence	Numero di sequenza dei flussi
Src Addr	Indirizzo IP della sorgente
Dst Addr	Indirizzo IP address della destinazione
Next Hop	Indirizzo IP del next-hop
Src Port	Numero della porta sorgente TCP/UDP
Dst Port	Numero della porta di destinazione TCP/UDP
Protocol	Tipo del protocollo
Src Mask	Subnet Mask della sorgente
Dst Mask	Subnet Mask della destinazione

Tabella 4: Metriche relative alla categoria NetFlow

6.3.3 Grafana

Le funzionalità offerte da TSDR permettono agli amministratori di rete di utilizzare delle applicazioni integrate a questo framework che acquisiscono le informazioni, le elaborano e generano analisi di prestazioni della rete automatizzate. Quindi vi è la possibilità di integrare un tool grafico al controller ODL per visualizzare, graficamente e in tempo reale, i dati temporali raccolti.

In questo esempio è stato utilizzato ***Grafana*** [23], tool grafico per la visualizzazione, l'analisi e il monitoraggio di dati. Come integrare Grafana a ODL viene spiegato nell'appendice B. Innanzitutto, bisogna definire le metriche del grafico che si vuole visualizzare. TSDR utilizza una specifica *TSDRKey* per la definizione di queste:

```
TSDRKey:  [NID=][DC=][MN=][RK=]
```

Per esempio, se si vuole visualizzare le statistiche relative alla metrica *TransmittedBytes* nella porta con ID openflow:1:1, bisogna inserire la seguente query:

```
[NID=openflow:1][DC=PORTSTATS][MN=TransmittedBytes]
[RK=Node:openflow:1,NodeConnector:openflow:1:1]
```

Il risultato della query, mostrato nella Figura 30, è un grafico che prende le informazioni memorizzate nel TSDR datastore, le elabora e mostra le statistiche prodotte in tempo reale.

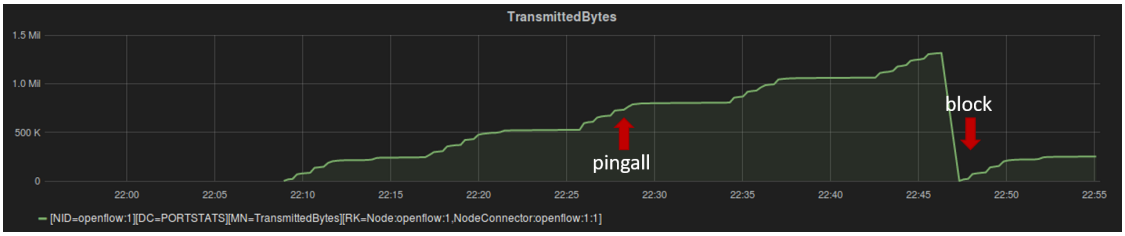


Figura 30: TransmittedBytes

6.4 Utilizzo delle API REST

Come già menzionato nel capitolo 5.3.4 le API REST sono utilizzate per la comunicazione northbound allo scopo di consentire alle applicazioni web l'accesso alle risorse ODL. Mediante queste è possibile richiedere, inviare, aggiornare e cancellare risorse.

Per fare ciò è necessario utilizzare un'applicazione web-client[15] capace di manipolare queste risorse attraverso il protocollo HTTP. Tra le più popolari e facili da utilizzare troviamo:

- Interfaccia YangUI in DLUX
- Postman per Google Chrome
- RESTClient per Firefox
- Comando CURL per Linux
- REST URL in Web Browser

6.4.1 GET: Richiesta di risorse

Il metodo GET ha la funzione di richiede la risorsa associata alla URI specificata, non cambiando lo stato dei dati. In altre parole, è una query per l'interrogazione del database in sola lettura. Negli esempi successivi viene spiegato come poter visualizzare le informazioni relative alla topologia di rete, agli switch e alle flow table mediante le API REST. L'output può essere in formato XML o JSON in base al parametro *Accept* inserito.

Esempio: Network

Nell'esempio che segue si vuole ottenere le informazioni generali relative ai nodi che compongono la rete, su cui si sta lavorando, attraverso una richiesta GET.

```
GET
Accept:application/XML
http://<controller_IP>:8181/restconf/operational/
    network-topology:network-topology/
```

L'output è un insieme di informazioni, in formato XML, che mostra i dettagli relativi alla topologia della rete. Vengono specificati:

- L'ID della topologia di rete su cui si sta lavorando.

```
1 <network-topology>
2    <topology>
3       <topology-id>flow:1</topology-id>
4    </topology>
5 </network-topology>
```

[15]REST client: applicazione capace di accedere allo stato di una risorsa e contribuire a modificarlo per mezzo dei metodi PUT, POST e DELETE dell'HTTP.

- Le informazioni relative agli host che compongono la rete, indicando l'indirizzo IP, l'indirizzo MAC e definendo lo stato dell'interfaccia. Nell'esempio sottostante si mostra i dettagli relativi all'host *h1*.

```
<node>
   <node-id>host:00:00:00:00:00:01</node-id>
   <id>00:00:00:00:00:01</id>
   <addresses>
      <id>1</id>
      <mac>00:00:00:00:00:01</mac>
      <ip>10.0.0.1</ip>
      <first-seen>1490038075913</first-seen>
      <last-seen>1490265831658</last-seen>
   </addresses>
   <attachment-points>
      <tp-id>openflow:1:1</tp-id>
      <active>true</active>
      <corresponding-tp>host:00:00:00:00:00:01</corresponding-tp>
   </attachment-points>
   <termination-point>
      <tp-id>host:00:00:00:00:00:01</tp-id>
   </termination-point>
</node>
```

- Le informazioni relative agli switch, definendo l'ID e le informazioni generali riguardo alle sue porte. In questo esempio vengono mostrati i dettagli dello switch con ID *openflow:1*.

```
<node>
   <node-id>openflow:1</node-id>
   <inventory-node-ref>/a:nodes/a:node[a:id='openflow:1']</inventory-node-ref>
   <termination-point>
      <tp-id>openflow:1:2</tp-id>
      <inventory-node-connector-ref>/a:nodes/a:node[a:id='openflow:1']/a:node-connector
          [a:id='openflow:1:2']</inventory-node-connector-ref>
   </termination-point>
   <termination-point>
      <tp-id>openflow:1:1</tp-id>
      <inventory-node-connector-ref>/a:nodes/a:node[a:id='openflow:1']/a:node-connector
          [a:id='openflow:1:1']</inventory-node-connector-ref>
   </termination-point>
   <termination-point>
      <tp-id>openflow:1:LOCAL</tp-id>
      <inventory-node-connector-ref>/a:nodes/a:node[a:id='openflow:1']/a:node-connector
          [a:id='openflow:1:LOCAL']</inventory-node-connector-ref>
   </termination-point>
</node>
```

- Le informazioni relative ai link, bidirezionali e non orientati, specificando la porta sorgente e quella di destinazione.

```
<link>
   <link-id>openflow:1:2</link-id>
   <source>
      <source-node>openflow:1</source-node>
      <source-tp>openflow:1:2</source-tp>
   </source>
   <destination>
      <dest-node>openflow:2</dest-node>
      <dest-tp>openflow:2:2</dest-tp>
   </destination>
</link>
```

Esempio: Switch

```
GET
Accept:application/XML
http://<controller_IP>:8181/restconf/operational/
     opendaylight-inventory:nodes/node/openflow:2
```

In questo esempio si vogliono visualizzare i dettagli relativi alla risorsa localizzata dall'URI. L'output mostra le informazioni in formato XML riguardo la risorsa indirizzata, in questo caso uno switch con ID *openflow:2*, specificando le caratteristiche dell'hardware:

```
<node>
   <id>openflow:2</id>
   <switch-features>
      <max_buffers>256</max_buffers>
      <max_tables>254</max_tables>
      <capabilities>flow-feature-capability-flow-stats</capabilities>
      <capabilities>flow-feature-capability-queue-stats</capabilities>
      <capabilities>flow-feature-capability-group-stats</capabilities>
      <capabilities>flow-feature-capability-port-stats</capabilities>
      <capabilities>flow-feature-capability-table-stats</capabilities>
   </switch-features>
   <serial-number>None</serial-number>
   <ip-address>127.0.0.1</ip-address>
   <manufacturer>Nicira, Inc.</manufacturer>
   <hardware>Open vSwitch</hardware>
   <software>2.6.0</software>
   <description>None</description>
</node>
```

Oltre a ciò, vengono visualizzati i dati inerenti alle interfacce dello switch indirizzato. Specificando l'ID e il nome dell'interfaccia, lo stato del link, le informazioni del nodo connesso a quella porta (indirizzo IP e MAC) e il numero dei byte, il numero dei pacchetti e la frequenza degli errori trasmessi e ricevuti in quella determinata interfaccia. Segue un esempio della porta *openflow:2:1*.

```
<node-connector>
   <id>openflow:2:1</id>
   <port-number>1</port-number>
   <current-speed>10000000</current-speed>
   <name>s2-eth1</name>
   <supported/>
   <current-feature>copper ten-gb-fd</current-feature>
   <configuration/>
   <peer-features/>
   <maximum-speed>0</maximum-speed>
   <advertised-features/><hardware-address>EE:E7:6A:B0:1B:9E</hardware-address>
   <state>
      <link-down>false</link-down>
      <blocked>false</blocked>
      <live>false</live>
   </state>
   <flow-capable-node-connector-statistics>
      <receive-errors>0</receive-errors>
      <receive-frame-error>0</receive-frame-error>
      <receive-over-run-error>0</receive-over-run-error>
      <receive-crc-error>0</receive-crc-error>
      <bytes>
         <transmitted>179712</transmitted>
         <received>9964</received>
      </bytes>
      <receive-drops>0</receive-drops>
      <duration>
         <second>4997</second>
         <nanosecond>906000000</nanosecond>
      </duration>
      <transmit-errors>0</transmit-errors>
      <collision-count>0</collision-count>
      <packets>
         <transmitted>2229</transmitted>
         <received>134</received>
      </packets>
      <transmit-drops>0</transmit-drops>
   </flow-capable-node-connector-statistics>
   <addresses>
      <id>2</id>
      <ip>10.0.0.2</ip>
      <first-seen>1490038076108</first-seen>
      <last-seen>1490265831658</last-seen>
      <mac>00:00:00:00:00:02</mac>
   </addresses>
</node-connector>
```

Esempio: Flow Table

Nell'esempio seguente viene mostrato come ottenere le informazioni relative a una tabella di routing. In questo caso l'URI viene indirizzato alla tabella con ID *table0* che è localizzata all'interno dello switch *openflow:2*. La risposta, in formato XML, descrive le informazioni generali della tabella e i dettagli dei flussi configurati all'interno di essa.

```
GET
Accept:application/XML
http://<controller_IP>:8181/restconf/operational/
    opendaylight-inventory:nodes/node/openflow:2/table/0
```

```
<table>
   <id>0</id>
   <table-features>
      <table-id>0</table-id>
      <max-entries>1000000</max-entries>
      <metadata-match>18446744073709551615</metadata-match>
      <name>table0</name>
      <metadata-write>18446744073709551615</metadata-write>
   </table-features>
   <flow-table-statistics>
      <packets-matched>1077</packets-matched>
      <packets-looked-up>1081</packets-looked-up>
      <active-flows>5</active-flows>
   </flow-table-statistics>
</table>
```

6.4.2 PUT: Aggiornamento di risorse

Il metodo PUT, visto precedentemente, ha la funzione di creare, aggiornare e cancellare le risorse. Negli esempi che seguono viene spiegato come poter installare e bloccare un flusso di pacchetti tramite le API REST.

Esempio: installare un flusso

Questo esempio fornisce i dettagli per programmare un flusso L2 di pacchetti in uno switch. Fanno parte del flusso con ID "BennyFlow" tutti i frame Ethernet che hanno come indirizzo MAC sorgente 00:00:00:00:00:02 e come indirizzo MAC di destinazione 00:00:00:00:00:01. L'azione prevista è *dec-mpls-ttl* che imposta il campo time-to-live (TTL) a tutti i pacchetti che sono associati a questo flusso, ovvero hanno le caratteristiche precedentemente descritte.

```
PUT
Accept:application/XML
Content-Type:application/xml
http://<controller_IP>:8181/restconf/operational/
    opendaylight-inventory:nodes/node/openflow:2/
    table/25/flow/BennyFlow
Body:
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
   <strict>false</strict>
```

```
4     <instructions>
5        <instruction>
6           <order>0</order>
7           <apply-actions>
8              <action>
9                 <order>0</order>
10                <dec-mpls-ttl/>
11             </action>
12          </apply-actions>
13       </instruction>
14    </instructions>
15    <table_id>25</table_id>
16    <id>BennyFlow</id>
17    <cookie_mask>255</cookie_mask>
18    <installHw>false</installHw>
19    <match>
20       <ethernet-match>
21          <ethernet-type>
22             <type>0x8847</type>
23          </ethernet-type>
24          <ethernet-destination>
25             <address>00:00:00:00:00:01</address>
26          </ethernet-destination>
27          <ethernet-source>
28             <address>00:00:00:00:00:02</address>
29          </ethernet-source>
30       </ethernet-match>
31    </match>
32    <hard-timeout>12</hard-timeout>
33    <cookie>4</cookie>
34    <idle-timeout>34</idle-timeout>
35    <flow-name>BennyFlow</flow-name>
36    <priority>2</priority>
37    <barrier>false</barrier>
38 </flow>
```

Se il flusso è stato installato correttamente nello switch, il controller ODL risponderà con un HTTP Status Code positivo.

Esempio: bloccare un flusso

Questo esempio fornisce i dettagli per programmare il blocco di un flusso di pacchetti L3 in uno switch. In questo caso si vogliono bloccare i pacchetti che attraversano la porta con ID *openflow:1:1*, quindi diretti all'host h1. L'azione da applicare al flusso, denominato *block*, è *drop-action*.

```
PUT
Accept:application/XML
Content-Type:application/xml
http://<controller_IP>:8181/restconf/operational/
    opendaylight-inventory:nodes/node/openflow:1/
    table/0/flow/block
```

Body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
  <priority>32535</priority>
  <flow-name>block</flow-name>
  <match>
    <ethernet-match>
         <ethernet-type>
            <type>2048</type>
         </ethernet-type>
    </ethernet-match>
    <in-port>openflow:1:1</in-port>
  </match>
  <id>block</id>
  <table_id>0</table_id>
  <instructions>
    <instruction>
       <order>0</order>
       <apply-actions>
          <action>
            <order>0</order>
            <drop-action/>
          </action>
       </apply-actions>
    </instruction>
  </instructions>
</flow>
```

Se il flusso è stato programmato nello switch senza errori, il controller ODL risponderà con un HTTP Status Code positivo. Possiamo verificare la corretta attività del flusso eseguendo, nella console Mininet, un *pingall* tra tutti gli host che compongono la rete.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X X X X X
h2 -> X h3 h4 h5 h6 h7 h8 h9 h10
h3 -> X h2 h4 h5 h6 h7 h8 h9 h10
h4 -> X h2 h3 h5 h6 h7 h8 h9 h10
h5 -> X h2 h3 h4 h6 h7 h8 h9 h10
h6 -> X h2 h3 h4 h5 h7 h8 h9 h10
h7 -> X h2 h3 h4 h5 h6 h8 h9 h10
h8 -> X h2 h3 h4 h5 h6 h7 h9 h10
h9 -> X h2 h3 h4 h5 h6 h7 h8 h10
h10 -> X h2 h3 h4 h5 h6 h7 h8 h9
*** Results: 20% dropped (72/90 received)
mininet>
```

Figura 31: Esecuzione di pingall fra gli host virtuali

Nella Figura 31 possiamo notare che i pacchetti che sono inviati e/o ricevuti dall'host *h1*, con indirizzo IP 10.0.0.1, ovvero tutti i pacchetti che attraversano la porta con ID *openflow:1:1*, sono scartati.

6.5 Considerazioni

Come abbiamo visto in questo scenario di implementazione lavorare con il controller OpenDaylight rende possibile ogni tipo di analisi e funzionalità sui pacchetti in transito nella rete, pur mantenendo una visione d'insieme del networking in tempo reale, rendendo così la gestione della rete molto più semplice.

Attraverso questa breve esperienza si può vedere come la semplice introduzione di una regola dinamica possa facilitare la gestione con lo scopo di migliorare notevolmente le prestazioni all'interno di un ambiente di rete anche banale. Difatti il controller ODL è stato programmato per adattarsi alla dinamicità delle richieste che giungono alla rete, semplificando e centralizzando la gestione della stessa, consentendo nel contempo una rapida evoluzione garantendo così uno sviluppo a velocità del software. Inoltre, gli esempi mostrati nei capitoli precedenti, ci mostrano come l'utilizzo dell'interfaccia web permette di interagire con il controller ODL in maniera semplice e rapida e come l'introduzione delle REST API, a disposizione delle applicazioni di rete, hanno reso possibile ai nuovi servizi e alle nuove funzionalità di essere implementate senza bisogno di dover intervenire manualmente su ogni singolo dispositivo di rete.

7 Conclusioni

La scelta e l'impiego di una struttura basata sul paradigma SDN compatibile con il protocollo OpenFlow rende la propria rete, in termini economici, un vantaggio competitivo. La tecnologia SDN, infatti, offre un sostanziale miglioramento delle performance di banda, un adattamento dinamico alla configurazione delle applicazioni e ai bisogni commerciali, il tutto riducendo la complessità delle operazioni di manutenzione.

Gli esperimenti condotti nel corso di questo lavoro di tesi mostrano come sia possibile utilizzare efficientemente la nuova architettura SDN ed il protocollo OpenFlow ad essa dedicato. Tutto è stato reso possibile grazie al software di simulazione Mininet e al controller OpenDaylight, che permettono di emulare un prototipo di rete funzionante in maniera rapida 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 sofisticati, che simulino anche applicazioni reali, ottenendone risultati immediatamente utilizzabili. Le prove evidenziano anche i limiti principali dei software utilizzati, i quali dovranno essere risolti affinché questo nuovo approccio di rete possa essere sfruttato al massimo delle sue potenzialità.

A Utilizzare Mininet

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

A.1 Installazione di Mininet

Abbiamo a disposizione più opzioni per poter installare Mininet [9].

OPZIONE 1

Il metodo più rapido e sicuro per installare Mininet è avvalersi della VM messa a disposizione dagli sviluppatori stessi [10]. Si tratta di un pacchetto contenente un'installazione di Ubuntu 14.04 con Mininet, 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 o VirtualBox i quali funzionano sia su macchine Linux che Windows.

OPZIONE 2

Per installare Mininet, mn e API di python, direttamente su un calcolatore con sistema operativo Linux Ubuntu 12.10+ è necessario eseguire il seguente codice da terminale:

```
sudo apt-get install mininet
```

OPZIONE 3

Installazione del software nativo, tramite repository. Questa opzione è ottimale per VM locali, in quanto è possibile poi apportarne i relativi aggiornamenti. Prima di tutto è necessario installare la variabile di ambiente Git[16].

```
sudo apt-get install git
```

Utilizzare git per scaricare il codice sorgente Mininet.

```
git clone git://github.com/mininet/mininet
```

[16]Git è il sistema di controllo del software utilizzato dal progetto Mininet.

Il comando precedente crea una cartella chiamate Mininet nella home directory. Per trovare l'ultima versione di Mininet eseguire:

```
cd mininet
git tag
git checkout -b <LastVersion>
```

Eseguire lo script di installazione che viene fornito dal progetto Mininet.

```
./mininet/util/install.sh -a
```

Infine, per verificare il corretto funzionamento del software, eseguire il seguente comando che genera un breve scenario Mininet.

```
sudo mn --test pingall
```

A.2 Creare una topologia di rete standard

Una volta eseguita la corretta installazione di Mininet, attraverso questo software è possibile creare qualsiasi tipo di network utilizzando singoli comandi e digitandoli su terminale.

Il seguente comando, come mostra la Figura 32, crea una semplicissima e minimale topologia di rete, composta da due host, uno switch ed un controller, con la quale si testa il funzionamento del programma.

```
sudo mn
```

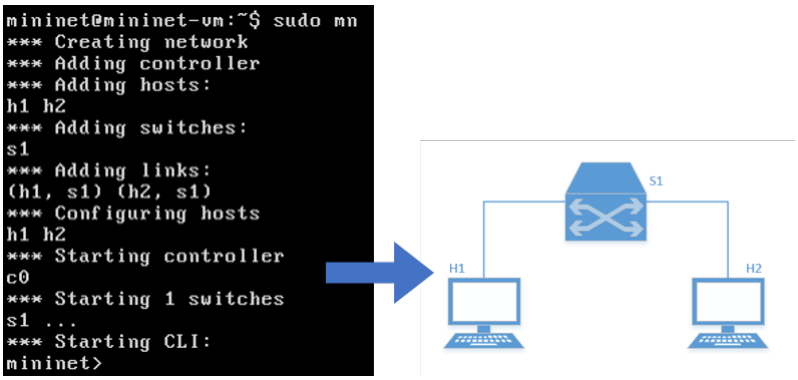


Figura 32: Minimal topology

Infatti, attraverso il comando mn, digitato da terminale, è possibile creare qualsiasi variante di topologia. Ad esempio, il seguente crea una rete composta da un

unico switch, tre host collegati ad esso e un controller remoto (Figura 33).

```
sudo mn --topo single,3 --mac --switch ovsk
```

Figura 33: Single switch topology

Nella Tabella 5 sono illustrate alcune delle varianti per la realizzazione di topologie di rete standard [11].

	Opzioni	Descrizione
--topo	minimal	MinimalTopology: crea una topologia di rete semplice, costituita da due host connessi a uno switch.
	single, <value>	SingleSwitchTopology: crea una topologia di rete composta da un unico switch. Il parametro <value> indica il numero degli host.
	linear, <value>	LinearTopology: crea una topologia di rete lineare, dove a uno switch corrisponde un host. Il parametro <value> indica il numero degli host, che equivale al numero degli switch.
	tree, depth=<value>,fanout=<value>	TreeTopology: crea una topologia di rete ad albero, con profondità e ampiezza indicate.
--switch	ovs,protocol=<value>	OpenvSwitch: è un'implementazione di uno switch basato sul protocollo OpenFlow. Il parametro <value> identifica la versione di OpenFlow da utilizzare.
	ovsk,protocol=<value>	OpenvSwitchKernel: è un'implementazione di uno switch basato sul protocollo OpenFlow con kernel Linux. Il parametro <value> identifica la versione di OpenFlow da utilizzare. Questa viene considerata l'implementazione più efficiente.
	user	UserSpaceSwitch: è un'implementazione di uno switch userspace. Questa tipologia è più soggetta a cali di prestazione dovuti ai context switch.
--controller	default	Gli switch che compongono la rete si connettono a al controller predefinito offerto da Mininet.
	remote,ip=<value>,port=<value>	Gli switch che compongono la rete si connettono a un controller specificato. Utilizzando remote senza specificare un indirizzo IP, gli switch si connetteranno all'indirizzo 127.0.0.1 con porta TCP 6633.
--nat	-	Aggiunge la connessione tra gli host della topologia Mininet alla rete fisica.
--mac	-	Assegna a ogni interfaccia un indirizzo MAC equivalente all'indirizzo IP, invece di un indirizzo casuale.
--arp	-	Imposta le coppie di voci ARP.
--test	pingall	Esegue un ping tra tutti gli host connessi alla rete. Utile per verificare il funzionamento della rete.
	irpef	Esegue un test di banda tra il primo e l'ultimo host della rete.

Tabella 5: Alcune opzioni per creare una topologia di rete standard

A.3 Creare una topologia di rete personalizzata

Finora si è lavorato con una topologia di rete standard che mette a disposizione Mininet da linea di comando tramite le opzioni viste precedentemente.

Un secondo metodo per creare la tipologia di rete che si vuole è quello di realizzare degli script in codice python (estensione *.py) che utilizza classi, metodi, funzioni e variabili che sono a disposizione in *API python* per Mininet [12]. Attraverso l'utilizzo di questi script è possibile creare e salvare topologie per poterle riutilizzare in seguito, in maniera molto rapida e semplice. Di seguito, nella Tabella 6, se ne elencano alcune delle principali:

Classe	Metodi	Descrizione
mininet.node.Node	MAC/setMAC()	Visualizza/Assegna l'indirizzo MAC a una specifica interfaccia.
	Ip/setIP()	Visualizza/Assegna l'indirizzo IP a una specifica interfaccia.
	cmd()	Invia un commando, aspetta la risposta e la visualizza.
mininet.net.Mininet	addHost()	Aggiunge un host alla topologia di rete.
	addSwitch()	Aggiunge uno switch alla rete.
	addLink()	Aggiunge un link bidirezionale alla rete.
	addController()	Aggiunge un controller.
	ping()	Utilizzato per testare la connettività tra gli host.
	start()	Avvia la rete creata.
	stop()	Ferma il controller, gli switche e gli host.
Mininet.topo.Topo	Metodi simili a net.	E.g., addHost, addSwitch, addLink…
	addNode()	Aggiunge un nodo al grafo.
	addPort()	Genera una porta per una nuova interfaccia.
	switches()	Visualizza tutti gli switch preseti nella rete

Tabella 6: Alcune API python per Mininet

Nella Figura 34 viene visualizzata la lista degli script python [13] già formati, messi a disposizione da Mininet, per facilitare il loro utilizzo. Questi si possono trovare nella directory `/mininet/examples/`.

```
mininet@mininet-VirtualBox:~$ cd mininet/examples/
mininet@mininet-VirtualBox:~/mininet/examples$ ls
baresshd.py       cpu.py              multilink.py      scratchnet.py
bind.py           emptynet.py         multiping.py      scratchnetuser.py
clustercli.py     hwintf.py           multipoll.py      simpleperf.py
clusterdemo.py    __init__.py         multitest.py      sshd.py
cluster.py        intfoptions.py      natnet.py         test
clusterSanity.py  limit.py            nat.py            tree1024.py
consoles.py       linearbandwidth.py  numberedports.py  treeping64.py
controllers2.py   linuxrouter.py      popenpoll.py      vlanhost.py
controllers.py    miniedit.py         popen.py
controlnet.py     mobility.py         README.md
mininet@mininet-VirtualBox:~/mininet/examples$
```

Figura 34: Lista degli script python offerti da Mininet

Tra questi i principali sono: *net.py* che mostra come collegare i nodi della rete Mininet a Internet utilizzando la connessione NAT, *bind.py* il quale crea directory private per ogni nodo, *controllers.py* che realizza una rete con più controller remoti e *multitest.py* il quale crea una rete ed esegue più test su di essa.

A.3.1 Esempio: Linear Topology

Come esempio semplicissimo di codice python, la Figura 35 mostra uno script che crea una topologia di rete lineare con quattro host e quattro switch ed esegue un *ping* da un host ad un altro per la verifica del funzionamento.

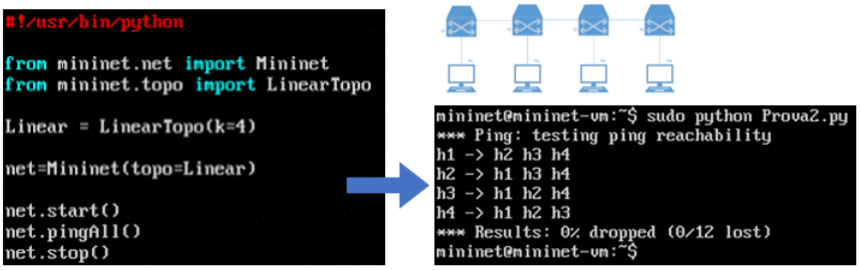


Figura 35: Esempio di script che genera un topologia lineare

A.3.2 Esempio: Tree Topology

La Figura 36 mostra un secondo esempio di script che crea una tipologia di rete ad albero con quattro host e tre switch ed esegue direttamente un *pingall*. Infine la network si blocca.

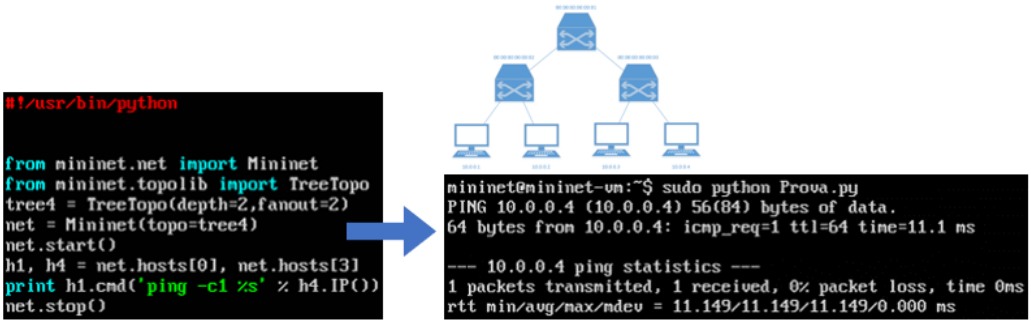


Figura 36: Script python che genera una topologia di rete ad albero

A.3.3 Esempio: Internet Topology

Lo script mostrato in Figura 37 crea una semplice network formata da un host, uno switch e un controller e permette la comunicazione tra l'host e la rete esterna.

Dal momento che Mininet è in esecuzione all'interno ad una VM, bisogna collegare una delle interfacce esterne della VM agli switch OVS in esecuzione in Mininet, nel mio caso l'interfaccia è denominata *eth2*. Inoltre la configurazione degli indirizzi IP degli host presenti nella rete non avviene manualmente, ma attraverso il protocollo DHCP.

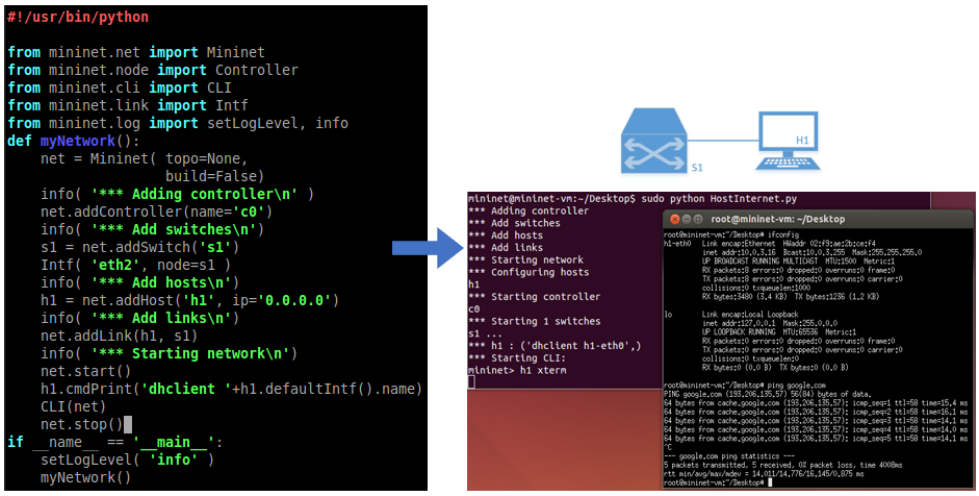


Figura 37: Script che crea una rete con la quale è possibile collegarsi in Internet

A.4 Creare una topologia di rete attraverso MiniEdit

Un altro metodo per creare topologie di rete più o meno complesse in Mininet è per mezzo dell'editor grafico ***MiniEdit*** [14]. MiniEdit è uno tool sperimentale creato per dimostrare come Mininet più essere esteso. Lo script MiniEdit si trova nella directory `/examples/` di Mininet. Per lanciare MiniEdit eseguire il comando:

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

L'interfaccia grafica di MiniEdit, come mostra la Figura 38, è molto semplice, in quanto presenta sul lato sinistro della finestra le varie icone per poter creare la nostra topologia personalizzata ed una barra di menù posizionata verso l'alto.

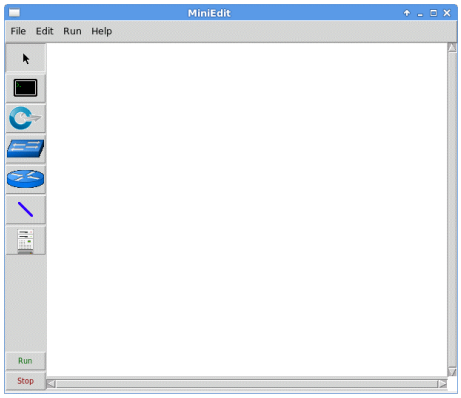


Figura 38: Interfaccia grafica di MiniEdit

Di seguito sono illustrati i vari componenti:

Select tool con un semplice "click e drag" è usato per muovere gli elementi della network sulla schermata.

Host tool crea i nodi che eseguiranno le funzioni degli host.

Switch tool permette di creare degli switch, questi sono previsti per poi avere un collegamento con un controller.

Legacy Switch tool crea un particolare tipo di switch con impostazioni di default. Questa tipologia di switch non è configurabile e opera indipendentemente, ovvero senza un controllore.

Legacy Router tool viene utilizzato per creare un router basilare che opera indipendentemente.

NetLink tool crea un collegamento tra i vari nodi presenti nella rete.

Controller tool consente di creare un controller. Le impostazioni di default implementano un learning switch, ovvero un OpenFlow controller.

Run botton simula lo scenario appena configurato nella finestra di MiniEdit.

Stop botton termina lo stato di esecuzione dello scenario appena avviato.

Attraverso MiniEdit è possibile configurare i vari elementi della rete per poi salvarne la topologia e lanciare l'emulazione attraverso Mininet:

Configurazione controller: di default Mininet usa un controller OpenFlow.

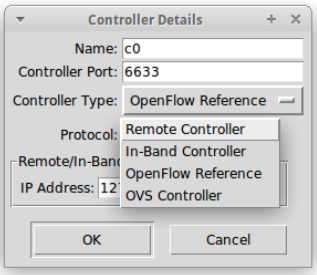


Figura 39: Configurazione del controller

Configurazione preferenze: per impostare le preferenze di MiniEdit è necessario andare nella barra dei menù, sotto la voce modifica.

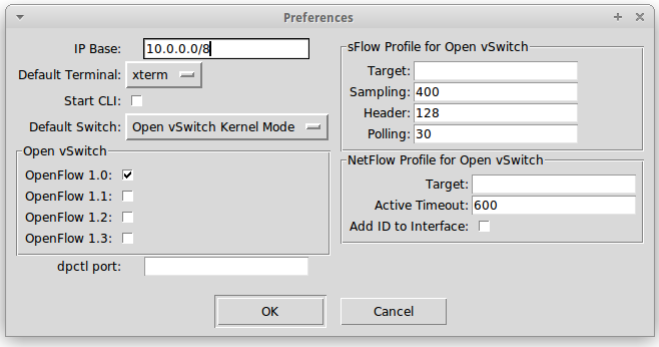


Figura 40: Impostazioni delle preferenze

Configurazione Switch Virtuale: per verificare che tutto sia impostato correttamente bisogna controllare le configurazioni di commutazione nella simulazione di rete. Il comando Show OVS summary mostra un riepilogo delle configurazioni dello switch. In questo caso, si può verificare che ogni interruttore è in ascolto al controller corretto sulla porta corretta

Salvataggio Preferenze: le preferenze di MiniEdit vengono salvate nel file della topologia dello scenario creato. In questo modo si può avere preferenze diverse per ogni scenario memorizzato.

Salvataggio Configurazione: avendo creato e configurato uno scenario di rete, che dovrebbe consentire ad ogni host di comunicare con qualsiasi altro host,

allora possiamo salvare il file della topologia di MiniEdit in modo da poter caricare questo scenario in futuro. Per salvare il file della topologia di Mininet (*.MN), basta cliccare su "File" nella barra dei menù e selezionare "Salva".

Un'altra possibile scelta è quello di esportare uno script in python, in modo da poterlo lanciare in una finestra di terminale. Per salvare il file in pyhton (*.py), bisogna cliccare su "File" nella barra dei menù e selezionare "salva livello 2 script".

Esecuzione Scenario di Rete: per avviare lo scenario di simulazione, basta fare click sul pulsante "Run" sulla GUI di MiniEdit. Nel terminale da cui è stato avviato MiniEdit vengono mostrati alcuni messaggi che mostrano l'avanzamento della simulazione.

A.5 Interazione con la rete

Una volta installato correttamente il software e creata la topologia di rete desiderata, l'ambiente Mininet è avviato. Quindi, è 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 presenta in questo modo:

```
mininet>
```

Indicandoci che ora siamo in ambiente Mininet ed è quindi possibile interagire con la network ed eseguire dei test digitando semplici comandi [11]:

```
mininet> help
```

Per poter visualizzare tutte le istruzioni attraverso le quali è possibile avere informazioni della rete su cui stiamo lavorando.

```
mininet> nodes
```

Questo comando ci permette di visualizzare l'elenco di tutti i nodi virtuali che compongo la rete.

```
mininet> net
```

Ci permette di visualizzare tutti i collegamenti tra i vari nodi, inoltre ci mostra i nodi e le connessioni associate alle relative porte.

```
mininet> <nodeID> ifconfig
```

Ci permette di visualizzare tutti i collegamenti tra i vari nodi, inoltre ci mostra i nodi e le connessioni associate alle relative porte.

```
mininet> <nodeID> route
```

Si utilizza per visualizzare i record della tabella di routing dei nodi che compongono la rete.

```
mininet> pingall
```

Attraverso il quale si eseguono dei ping tra tutti gli host connessi alla rete, molto utile per valutare l'effettivo funzionamento dell'intera rete creata.

```
mininet> <nodeID> ping <nodeID>
```

Questo comando ha la funzione di verificare la raggiungibilità o meno tra due host che compongono la rete virtuali.

```
mininet> dump
```

Il comando stampa a video le informazioni relative a tutti i nodi creati che costituiscono la rete.

```
mininet> <nodeID> xterm
```

Con il quale è possibile avviare da remoto i terminali relativi ai vari nodi, per poter utilizzare i normali strumenti di rete messi a disposizione da linux.

```
mininet> irpef <nodeID> <nodeID>
```

Attraverso il quale si eseguono dei test di banda tra i nodi indicati. Se utilizzato da interfaccia Mininet senza parametri, viene eseguito tra il primo e l'ultimo nodo.

```
mininet> link <nodeID> <nodeID> down
```

Attraverso il quale è possibile disabilitare le interfacce virtuali Ethernet di un link.

```
mininet> link <nodeID> <nodeID> up
```

Attraverso il quale è possibile abilitare le interfacce virtuali Ethernet di un link.

```
mininet> exit
```

Con il quale è possibile uscire dall'interfaccia Mininet.

Come esempio, nella Figura 41 viene mostrato il risultato di alcune di queste istruzioni applicate alla topologia di rete mostrata nel paragrafo A.2, composta da tre host connessi ad un unico switch:

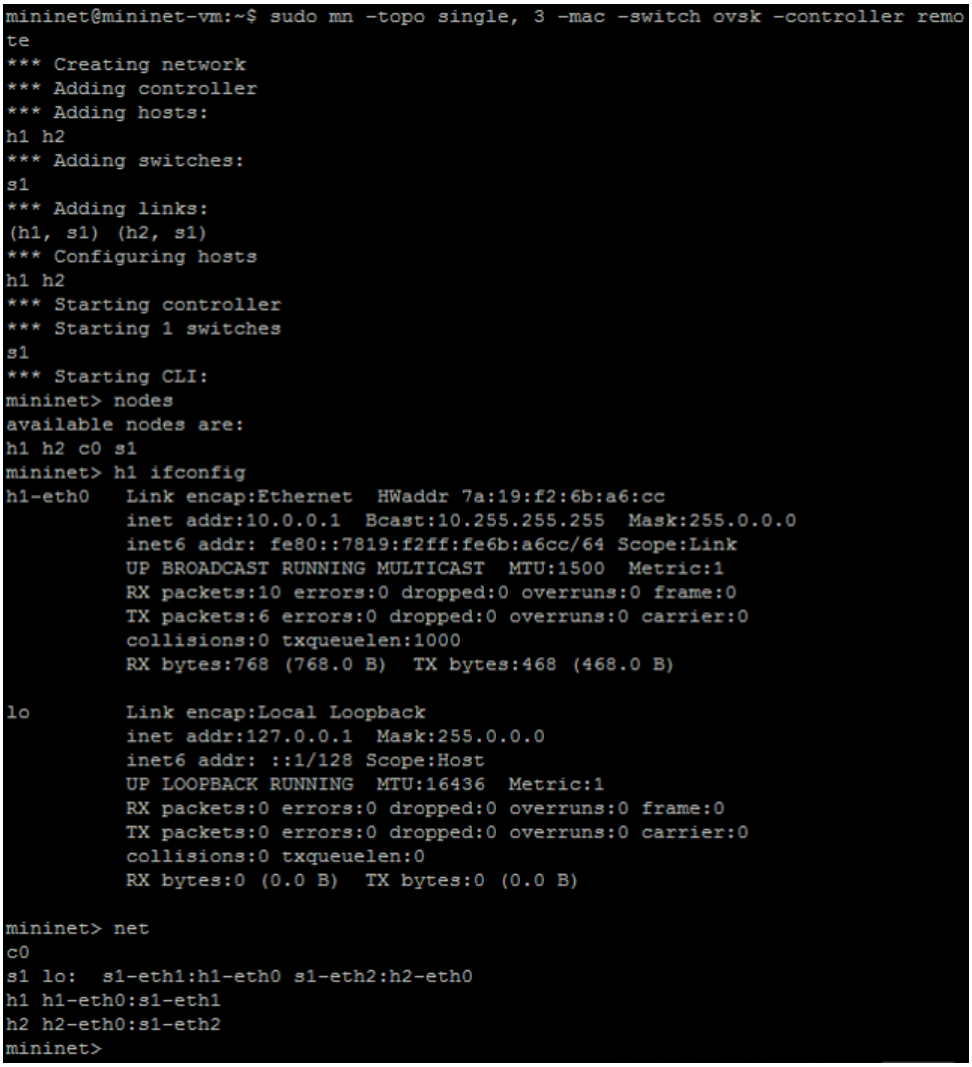


Figura 41: Esempio di come interagire con una rete Mininet virtuale

B Integrazione di Grafana

Grafana è un tool grafico per la visualizzazione, l'analisi e il monitoraggio di dati e, attraverso le seguenti istruzioni, è possibile integrare il software con il controller ODL per visualizzare i dati temporali raccolti da TSDR graficamente e in tempo reale [24].

B.1 Procedimento di installazione

B.1.1 Installazione di Apache

Per lavorare con Grafana è necessaria l'installazione di Apache, HTTP server, attraverso il seguente comando:

```
sudo apt-get install apache2
```

B.1.2 Installazione di Grafana

Scaricare il file zip di Grafana[17] dal sito ufficiale [25] ed estrarlo nella directory `/opt/grafana`.

B.1.3 Configurazione di Apache

Una volta estratto il file, è necessario modificare il file *apache2.conf*, che si trova nella directory `/etc/apache2/` ed aggiungere la seguente sezione:

```
<Directory /opt/grafana/>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

Inoltre è essenziale cambiare i *DocumentRoot* nelle configurazioni di Apache, ciò modificando il file nella directory `/etc/apache2/sites-available/` e includere le seguenti righe:

```
<VirtualHost:*80>
    DocumentRoot /opt/grafana
```

[17] Grafana versione 1.9.1.

B.1.4 Configurazione di Grafana

Infine, modificare il file di configurazione *config.sample.js* presente nella directory `/opt/grafana` e impostare il data source nel seguente modo:

```
datasources:{
        graphite:{
            type:  'graphite',
            url:  "http://127.0.0.1:8181/tsdr/nbi",
            render_method:  'GET'
            }
        }
```

B.2 Utilizzare Grafana

B.2.1 Avviare interfaccia web di Grafana

Innanzitutto, avviare il web server Apache.

```
/etc/init.d/apache2 start
```

Per accedere all'interfaccia GUI web-based di Grafana, aprire il browser e digitare il seguente URL. Nella Figura 42 viene mostrata l'interfaccia iniziale.

```
URL: http://<IP_Address:80/
```

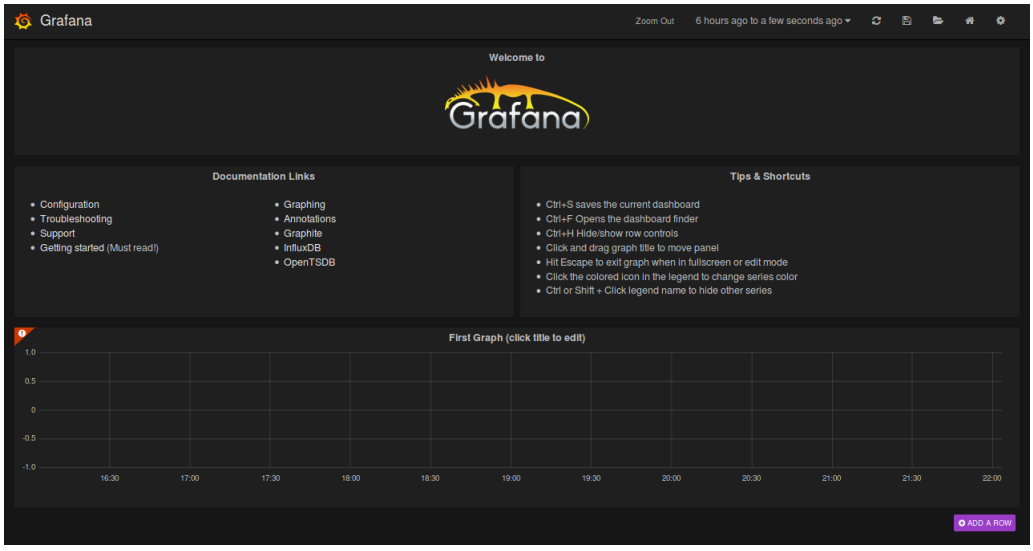


Figura 42: Interfaccia web-based di Grafana

B.2.2 Creazione di grafici

Una volta che il software è stato integrato al framework TSDR del controller ODL, attraverso le configurazioni precedenti, ed è stato avviato è possibile creare dei grafici definendone le metriche. Semplicemente inserendo delle specifiche TSDRKey.

```
TSDRKey:  [NID=][DC=][MN=][RK=]
```

Il risultato delle query sono grafici costruiti mediante le informazioni memorizzate nel datastore TSDR. Nei seguenti sottoparagrafi vengono mostrati alcuni esempi, in riferimento alla topologia creata nel capitolo 6.2:

Esempio: TransmittedBytes

Il grafico mostrato nella Figura 43 raffigura le statistiche relative alla metrica *TransmittedBytes*, ovvero vengono raccolte ed elaborate le informazioni relative ai pacchetti trasmessi dalla porta con ID *openflow:1:1*.

```
[NID=openflow:1][DC=PORTSTATS][MN=TransmittedBytes]
[RK=Node:openflow:1,NodeConnector:openflow:1:1]
```

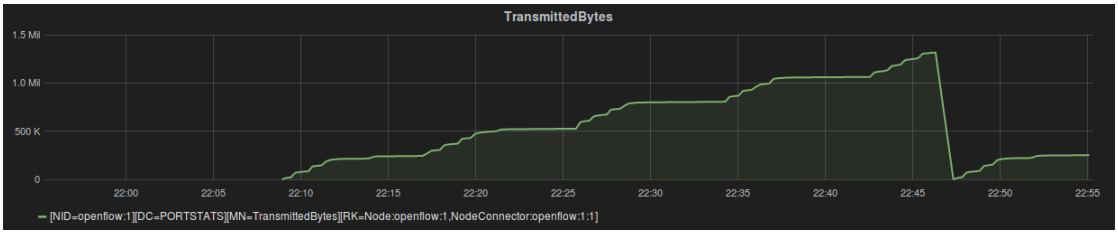


Figura 43: TransmittedBytes

Esempio: ReceiveDrops

In questo secondo esempio vengono raccolte ed elaborate le statistiche inerenti alla metrica *ReceiveDrops*. Come si può notare nella Figura 44, nessun pacchetto perso è stato ricevuto dalla porta con ID *openflow:1:1*.

```
[NID=openflow:1][DC=PORTSTATS][MN=ReceivedBytes]
[RK=Node:openflow:1,NodeConnector:openflow:1:1]
```

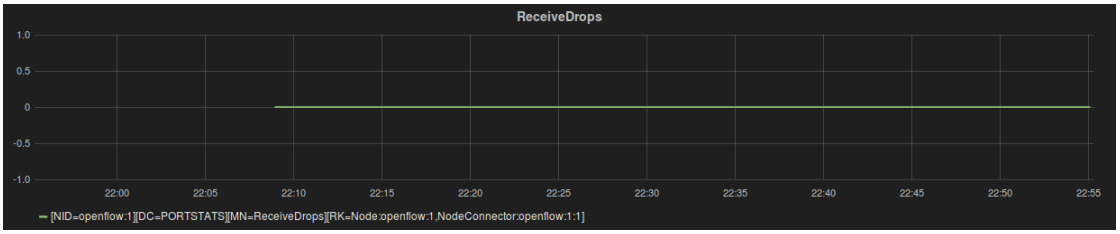


Figura 44: ReceiveDrops

Esempio: ReceivedBytes

In questo caso, come viene mostrato nella Figura 45, sono state elaborate le statistiche relative alla metrica *ReceivedBytes*. Ovvero vengono raccolte le informazioni sui pacchetti ricevuti dalla porta con ID *openflow:1:1*.

```
[NID=openflow:1][DC=PORTSTATS][MN=ReceiveDrops]
[RK=Node:openflow:1,NodeConnector:openflow:1:1]
```

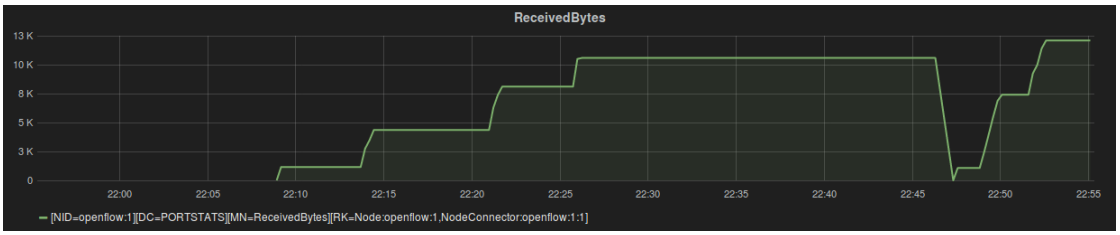


Figura 45: ReceivedBytes

Elenco delle figure

Elenco delle tabelle

Riferimenti bibliografici

1 *Software-Defined Networking: The New Norm for Networks*, Open Networking Foundation, ONF White Paper, 2012, https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

2 *Software-Defined Networking (SDN) Definition*, Open Networking Foundation, https://www.opennetworking.org/sdn-resources/sdn-definition.

3 *Understanding the SDN Architecture*, SdxCentral, https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/.

4 *OpenFlow*, http://archive.openflow.org/.

5 *OpenFlow Switch Specification*, Open Networking Foundation, ONF TS-009, 2013, https://www.opennetworking.org/.

6 *Key Benefits of OpenFlow-Based SDN*, Open Networking Foundation, https://www.opennetworking.org/?p=321&option=com_wordpress&Itemid=164.

7 *Mininet*, http://mininet.org/.

8 *Introduction to Mininet*, https://github.com/mininet/mininet/wiki/Introduction-to-Mininet.

9 *Download/Get Started With Mininet*, http://mininet.org/download/.

10 *Mininet VM Images*, https://github.com/mininet/mininet/wiki/Mininet-VM-Images.

11 *Mininet Walkthrough*, http://mininet.org/walkthrough/.

12 *Mininet Python API Reference Manual*, http://mininet.org/api/.

13 *Mininet Examples*, https://github.com/mininet/tree/examples.

14 *How to use MiniEdit, Mininet's graphical user interface*, http://www.brianlinkletter.com/.

15 *OpenDaylight*, https://www.opendaylight.org/.

16 *OpenDaylight Controller Overview*, http://docs.opendaylight.org/en/stable-boron/user-guide/opendaylight-controller-overview.html.

17 *OpenDaylight Download*, https://www.opendaylight.org/downloads.

18 *Using the OpenDaylight User Interface (DLUX)*, https://wiki.opendaylight.org/view/DluxApps:Main.

19 *L2 Switch:Main*, https://wiki.opendaylight.org/view/L2_Switch:Main.

20 *TSDR:Main*, https://wiki.opendaylight.org/view/TSDR:Main.

21 *RESTCONF Protocol*, RFC 8040, January 2017, https://tools.ietf.org/html/rfc8040.

22 *OpenDaylight Controller:MD-SAL:Restconf*, https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Restconf.
23 *Grafana: The open platform for beautiful analytics and monitoring*, https://grafana.com/.
24 *Grafana Integration with TSDR Step-by-Step*, https://wiki.opendaylight.org/view/Grafana_Integration_with_TSDR_Step-by-Step.
25 *Download Grafana*, https://grafana.com/grafana/download.

Ringraziamenti