SELINUS UNIVERSITY
OF SCIENCES AND LITERATURE

SOFTWARE DEFINED NETWORK: L'EVOLUZIONE DELLA RETE

By
Emiliano Canneva

Supervised by
Prof. Salvatore Fava Ph.D

A DISSERTATION

Presented to the Department of Information Technology
program at Selinus University

Faculty of Computer Science
in fulfillment of the requirements
for the degree of Bachelor of Science in
Information Technology

2019

INTRODUZIONE

Nell'ultimo decennio abbiamo assistito ad un numero sempre crescente di dispositivi e servizi connessi alla rete Internet, rendendola sempre più estesa e complessa, non adatta alle esigenze del utente finale che richiede una gestione dinamica, veloce e sicura. Tutto ciò ha portato l'industria del networking a riesaminare completamente l'architettura di rete e nel 2011 venne fondata la Open Networking Foundation (ONF) che si fece promotrice di una nuova proposta per la crescita e l'evoluzione della attuale rete di interconnessione, il Software Defined Networking (SDN).

In questo documento esamineremo il background attuale e le motivazioni ce hanno portato alla nascita di SDN. SDN nell'accezione più diffusa del termine si propone come metodo per rendere programmabile via software gli elementi di rete adibiti all'instradamento dei pacchetti, rendendo la progettazione, distribuzione e la manutenzione della architettura di rete più semplice. In particolare analizzeremo nel dettaglio questo emergente paradigma di rete evidenziandone i punti di forza, le potenzialità ed i benefici. Inoltre, approfondiremo lo standard OpenFlow, sostenuto sempre da ONF, che si è imposto come base per lo sviluppo di SDN. Tramite lo standard OpenFlow si è riusciti ad avere un accesso diretto ed a manipolare il forwarding plane dei dispositivi di rete quali router e switch.

Dopo aver studiato gli elementi fondamentali di SDN e OpenFlow, parleremo di Mininet (simulatore di rete), che utilizzeremo per creare rapidamente una network basata su SDN e del controller

SDN, RYU, di cui analizzeremo la struttura ed i principali componenti per comprenderne il funzionamento.

Infine, si è proceduti alla creazione ed alla verifica, del funzionamento di una rete virtuale emulata da Mininet in comunicazione con il controller RYU, tutto ciò con lo scopo di capire come gestire e monitorare una rete logicamente centralizzata.

INDICE

1. SOFTWARE DEFINED NETWORKING

In questo capitolo, tratteremo del paradigma SDN, studiandone i suoi componenti e la sua architettura. Inoltre, analizzeremo le motivazioni che hanno portato alla nascita ed i relativi benefici.

1.1. LA RETE TRADIZIONALE ED I SUI LIMITI

Dalle origini ad oggi, la rete attuale non ha subito significative modifiche, nei dispositivi utilizzati dalla rete tradizionale, come da 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 migliore che i pacchetti utilizzeranno per raggiungere la destinazione. Essa 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.

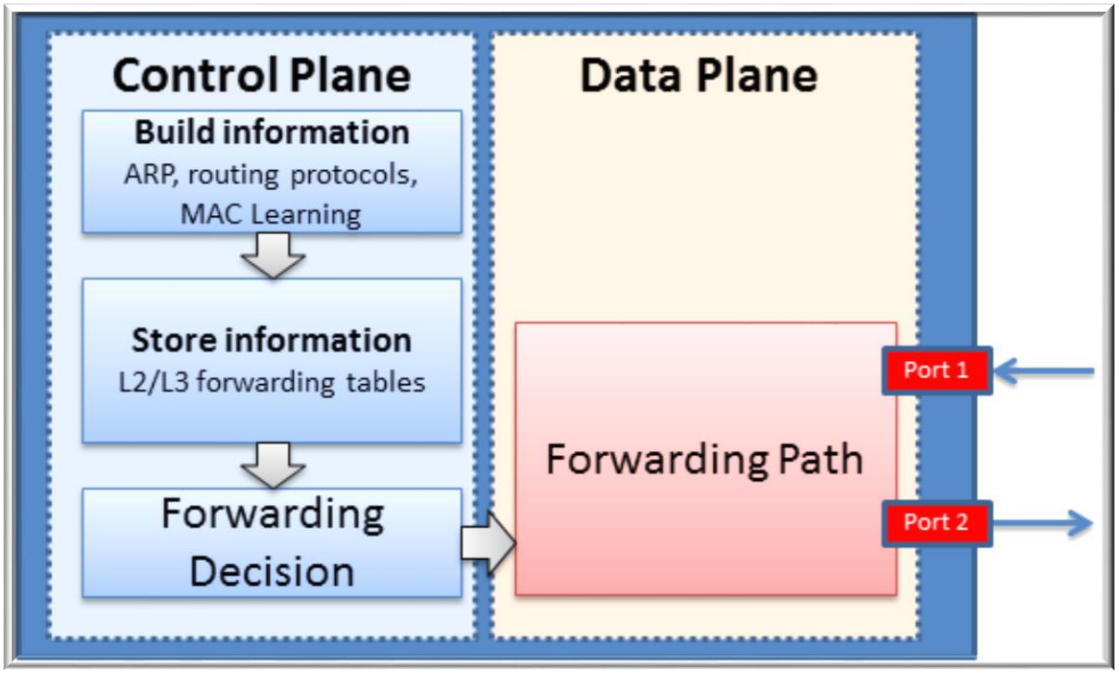

Figura 1: Componenti di uno switch tradizionale

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

L'introduzione della virtualizzazione e l'incremento dei servizi cloud infatti, hanno causato un aumento di complessità di gestione e di funzionamento della rete che l'attuale struttura, costituita per un utilizzo gerarchico a più livelli, non è in grado di gestire, mostrando i suoi limiti e le sue debolezze.

I principali fattori che hanno richiesto nuove architetture strutturali sono:

- **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.

- **Limitata scalabilità:**

  La rete tradizionale non permette di avere una rete scalabile, ovvero una rete che abbia le stesse prestazioni senza degradi nel momento in cui la rete si evolve. Ad esempio, se una azienda ha la necessità di ampliare la sua rete dovrà in maniera manuale lavorare sulle singole macchine e di conseguenza anche la più piccola modifica può richiedere uno spreco di risorse e di tempo. 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 imprevedibile.

- **Flessibilità limitata:**

  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 grandi quantità di dati. Il traffico intermedio fra queste applicazioni distribuite è altamente dinamico, ed questo aspetto che 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 può richiedere molto tempo. 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:**

  Gestire una grande quantità di dati comporta un maggiore utilizzo di banda per gestire più processi in parallelo di migliaia di server collegati tra loro. 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 di rete, può essere necessario la configurazione di migliaia di dispositivi e sistemi. La complessità delle reti rende molto difficile l'applicazione di un insieme di regole di accesso, di sicurezza, di Quality of Service (QoS), 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.

1.2. INTRODUZIONE AL SOFTWARE DEFINED NETWORKING

Il concetto di Software Defined Networking definito da ONF costituisce un nuovo approccio in ottica cloud computing alle architetture di rete, che ne facilita l'amministrazione e la configurazione al fine di migliorarne performance e facilitarne il monitoring.

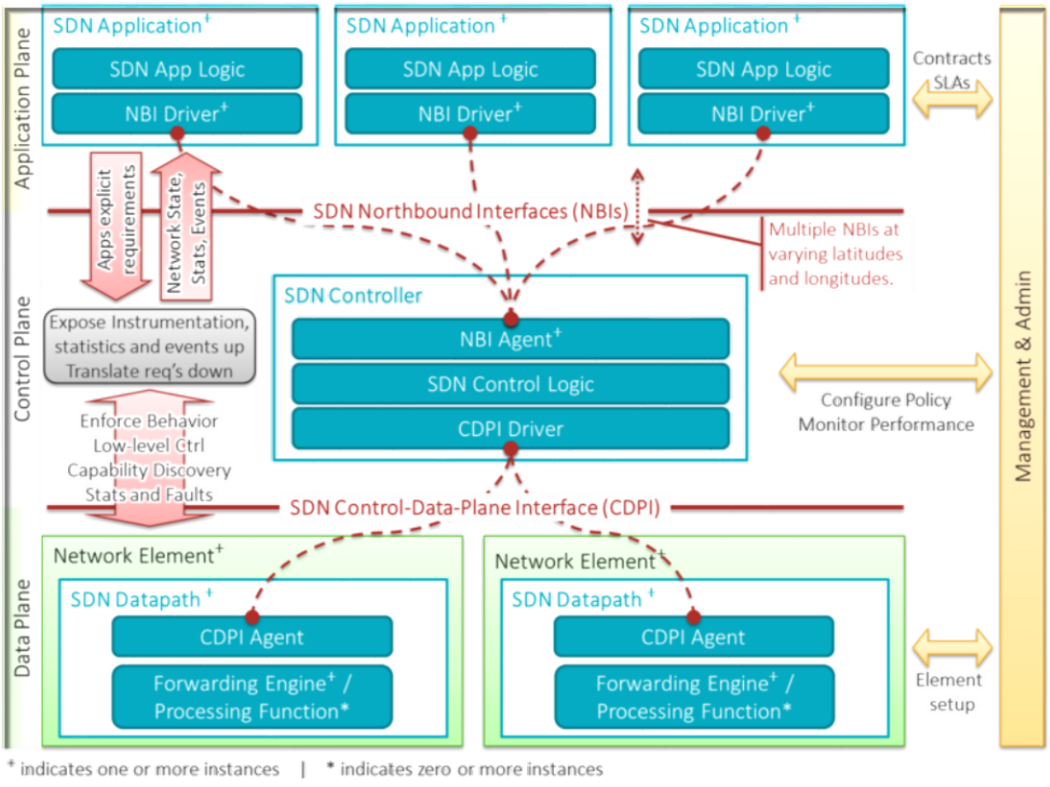

Figura 2: Descrizione ad alto livello di un'architettura SDN

L'architettura SDN nasce con l'intento di essere dinamica, gestibile, economicamente efficiente e adattabile, cercando di essere funzionale per la natura dinamica e ad alto consumo di banda delle applicazioni odierne. Le architetture SDN disaccoppiano il controllo di rete e le funzioni di forwarding, dando la possibilità al controllo di rete di divenire direttamente programmabile e alla sottostante infrastruttura di essere astratta dalle applicazioni e dai servizi di rete.

Le principali caratteristiche della tecnologia SDN sono:

- **Direttamente programmabile:**

il controllo della rete è direttamente programmabile perché è disaccoppiato dalle funzioni di forwarding.

- **Agile:**

L'astrazione del controllo dal forwarding permette agli amministratori di modificare dinamicamente il flflusso di traffico nell'intera rete per soddisfare le necessità di cambiamento.

- **Gestita centralmente:**

L'intelligenza della rete è (logicamente) centralizzata all'interno dei controller SDN che mantengono una vista globale della rete, la quale appare alle applicazioni ed ai motori di policy come un unico switch logico.

- **Configurazione programmata:**

SDN permette ai gestori della rete di configurare, gestire, rendere sicure e ottimizzare le risorse di rete in maniera molto veloce

attraverso script automatizzati che essi stessi possono scrivere, in quanto non devono basarsi su software proprietario.

- **Basata su standard aperti e indipendente dai vendor:**

Se implementata attraverso standard aperti, SDN semplifica il disegno e manutenzione della rete poiché le istruzioni sono fornite dai controller SDN invece che da molteplici dispositivi e protocolli proprietari.

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.

1.3. RETI A CONFRONTO – RETE TRADIZIONALE VS SDN

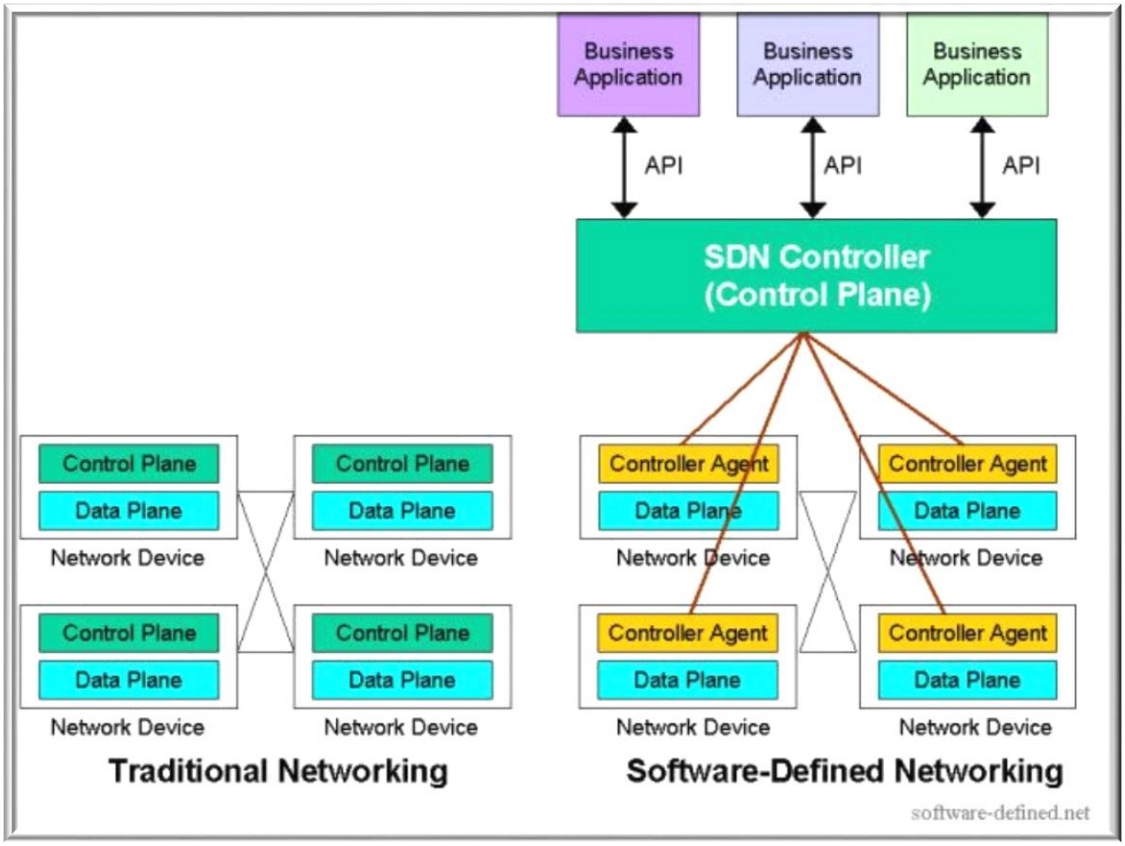

Figura 3: Rete tradizionale e SDN a confronto

La rete attuale si basa su un'architettura distribuita dove il controllo è confinato nei singoli 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 finali. In contrapposizione a questa, come illustrato nella Figura 3, con SDN il controllo della rete è disaccoppiato dall'hardware e posto in un controller, un software logicamente centralizzato, che libera l'utente

dalla singola gestione dei dispositivi, riducendo così la complessità operativa e architetturale.

Il controller, grazie alla logica centralizzata, 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 in base alle 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.

1.4. ARCHITETTURA

La struttura logica di SDN prevede la suddivisione in tre livelli di astrazione, come si può vedere nella Figura 4.

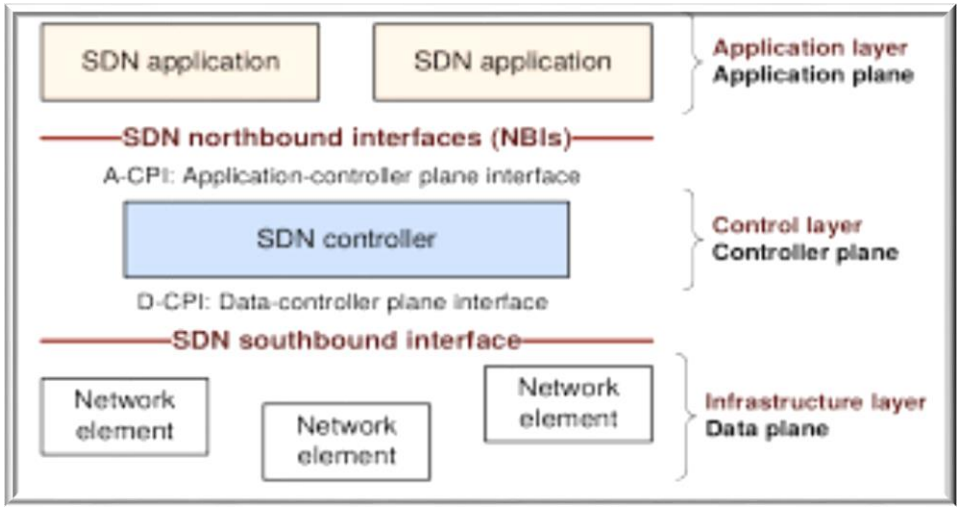

Figura 4: Architettura SDN

Nel primo livello (Application layer) troviamo una serie di applicazioni, tramite le quali gli sviluppatori devono interfacciarsi per usufruire della rete e delle sue funzionalità.

Il secondo è il livello di controllo (Control layer), all'interno del quale risiede la capacità computazionale dell'intera rete. Il controllo viene attuato tramite una serie di software opportunamente sviluppati, funzionanti su dispositivi dedicati, in grado di monitorare l'intera topologia di rete in modo da veicolare e analizzare i flussi del traffico dati in modo più avanzato ed efficiente rispetto alle reti tradizionali. Il software è in grado di interagire con le tabelle decisionali dei dispositivi, in modo da instradare fisicamente i flussi dati, in base al modello elaborato dal controllore.

All'ultimo livello (Infrastructure layer) si colloca l'infrastruttura fisica realizzata tramite hardware dedicati. Gli elementi che compongono questo livello sono in grado di svolgere solamente funzioni basilari sui pacchetti, ma il loro scopo principale è quello di inoltrare fisicamente i pacchetti realizzando dei collegamenti in maniera estremamente stabile e veloce.

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.

1.4.1. 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, parleremo delle 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.

1.4.2. 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 2. 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. Essa rappresenta un punto critico dell'architettura SDN, in quanto non solo permette al controller di gestire in maniera dinamica il traffico di una rete, ma ne incrementa l'efficienza in termini di traffico e richieste di transito.

1.4.3. 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 4, 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. Successivamente, nel capitolo 3, viene studiato e

analizzato nel dettaglio il controller RYU, il quale viene utilizzato per la centralizzazione logica e la gestione della rete.

1.4.4. NORTHBOUND INTERFACE

Le 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.

1.4.5. 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.

1.5. VANTAGGI

La scelta di suddividere il control plane dal data plan e di centralizzare la gestione nel SDN controller ha portato ai seguenti vantaggi:

- consente l'astrazione dell'infrastruttura sottostante;
- le business application ed i servizi di rete percepiscono il network come una entità logica;
- si realizza un'indipendenza dai vendor e dalle specificità dei dispositivi;
- viene semplificata sia la progettazione sia la gestione operativa dell'intera rete;
- permette di rendere più semplice sia le tecnologie di costruzione di router e switch, sia la loro configurazione, poiché il dispositivo svolge solo funzioni di forwarding del flusso e si richiede l'utilizzo del solo protocollo di comunicazione con il controller SDN;
- il piano di controllo viene arricchito con nuove funzionalità;
- la logica della rete è definibile direttamente dagli utilizzatori; è possibile instradare in tempo reale il traffico in base allo stato dell'intera infrastruttura IT, programmando adeguatamente il controller;
- velocizza i processi d'innovazione e di automatizzazione consentendo la creazione di nuove funzionalità e servizi di rete, in modo più semplice e rapido senza dover configurare i singoli dispositivi o modificarne il firmware;

- incrementa l’affidabilità e la sicurezza della rete avendo centralizzato ed automatizzato la gestione dei network device;
- consente di controllare la rete ed applicare policy con diversi livelli di granularità, sessione, utente, device ed applicazione.
- l’architettura SDN fornisce un set di API che consentono di implementare servizi di rete comuni, come routing, multicast, security, access control, bandwidth management, traffic engineering, quality of service, processor e storage optimization, energy usage, policy management.

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

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

Tuttavia la genericità dell’architettura SDN crea un po’ di ambiguità, i confini tra le business application ed il controller non sono ben definiti, in termini di programmazione di alto livello della rete. In

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

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

2. OPENFLOW

In questo capitolo parleremo dello 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.

2.1. 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.

Figura 5: 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 dei singoli 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 redirezione 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 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 soprattutto di ottenere una reattività alle modifiche da parte dell'utente in tempo reale.

2.2. 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: il control plane viene completamente rimosso dal dispositivo di rete, lasciando al dispositivo la sola funzione di trasporto dei 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 6, 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, che rappresneta l'interfaccia di 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.

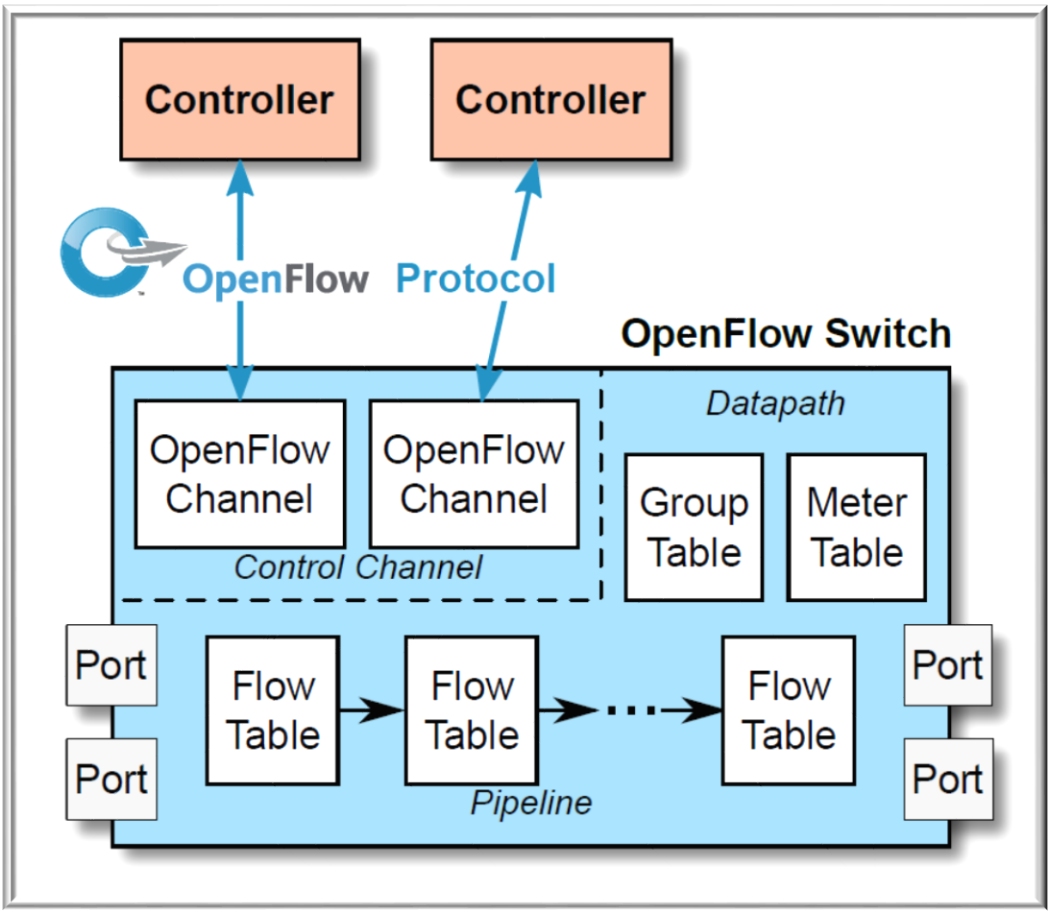

Figura 6: Principali componenti di uno switch OpenFlow

2.3. 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 7) all'interno delle quali a ogni sua voce è associata un'azione. Le azioni previste sono:

- 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.
- 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.

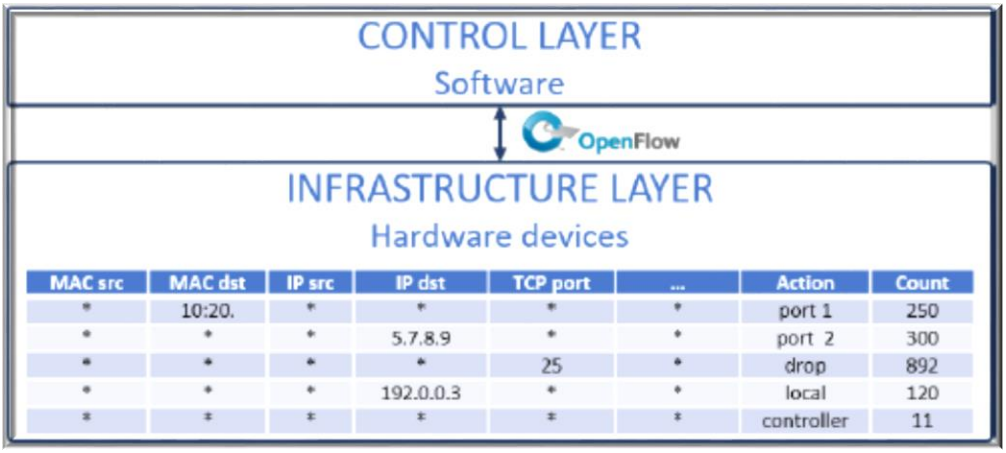

Figura 7: Esempio di una flow table

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 (es.: indirizzo IP, indirizzo MAC, 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 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 7).

Ogni record della flow table contiene:

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

- Un'azione, anch'essa programmabile dal controller. Definisce come il pacchetto deve essere instradato lungo la rete, in conformità a parametri legati a specifici patter, applicazioni e risorse.
- Delle statistiche relative al conteggio dei pacchetti corrispondenti ad ogni regola.

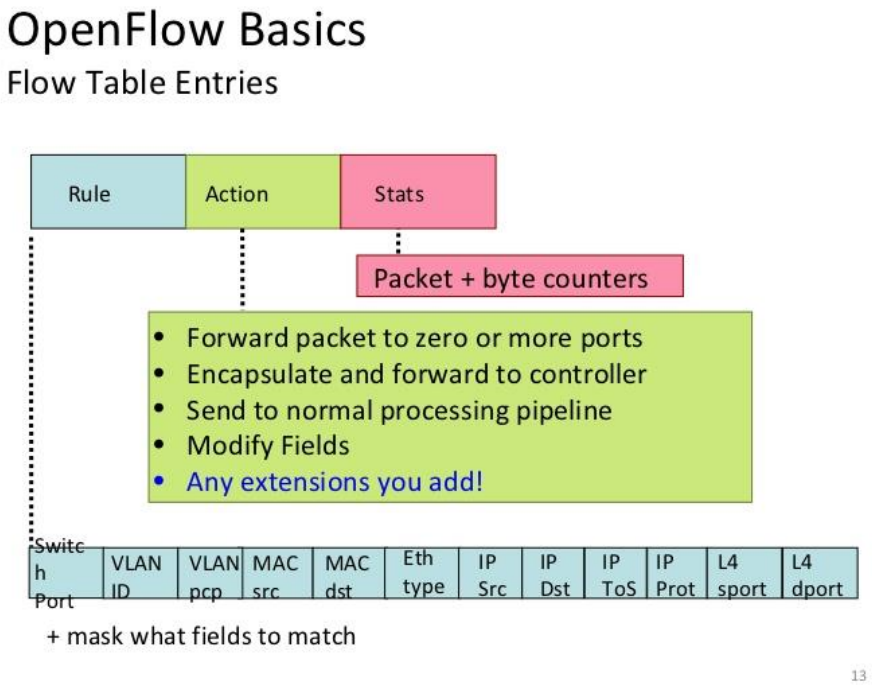

Figura 8: Descrizione dei record che costituiscono le ow table.

Il funzionamento è semplice: tramite OpenFlow secure channel, avviene una comunicazione sicura, tra switch e controller che 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 nuovo pacchetto 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, così da poter rispondere, in tempo reale, ai cambiamenti che si verificano nell'intera rete.

2.4. BENEFICI

Il protocollo OpenFlow fin da subito si rivela un ottimo strumento per la facilità con cui si possono ideare e anche testare il funzionamento di nuovi protocolli, in poco tempo su una determitata rete si riesce a raccogliere tutte le informazioni e le relative criticità, e si può agire su di essa con modifiche strutturali che nelle reti tradizionali richiederebbero un gran dispendio di risorse e di tempo.

Con OpenFlow otteniamo molteplici vantaggi: aumento di velocità delle reti, maggiore dinamicità e agevolazione nella personalizzazione della rete, efficienza energetica, maggiore sicurezza e ottimizzazione del sistema. Esso 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 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 notevolmente la gestione.

3. CONTROLLER SDN: RYU

In questo capitolo viene introdotto il controller Ryu uno dei principali tra i controller SDN. Inoltre viene descritta la sua struttura e analizzate alcune delle sue principali funzionalità.

3.1. INTRODUZIONE AL CONTROLLER

Ryu è un software SDN open-source, completamente scritto in Python che fornisce un insieme di componenti che permettono agli sviluppatori di creare nuove applicazioni per la gestione e il controllo della rete.

Figura 9: Logo del controller Ryu.

Ryu, che in giapponese significa sia drago, che flusso, è un software SDN open-source, implementato interamente in Python, utilizzato come controller nei sistemi SDN. Come gli altri controller SDN, Ryu fornisce componenti software con API ben definite che permettono agli sviluppatori di creare nuove applicazioni di gestione e controllo della rete. Uno dei punti di forza di Ryu è la sua capacità di supportare svariati protocolli di southbound per la gestione dei dispositivi, ad esempio Openow, Netconf, OF-config, eccetera.

3.2. ARCHITETTURA

Come ogni controller SDN, Ryu può creare ed inviare dei messaggi OpenFlow, rilevare eventi asincroni, tra i quali la rimozione di un flusso, e gestire ed analizzare pacchetti in arrivo al controller.
La Figura 10 sottostante rappresenta l'architettura del framework del controller Ryu:

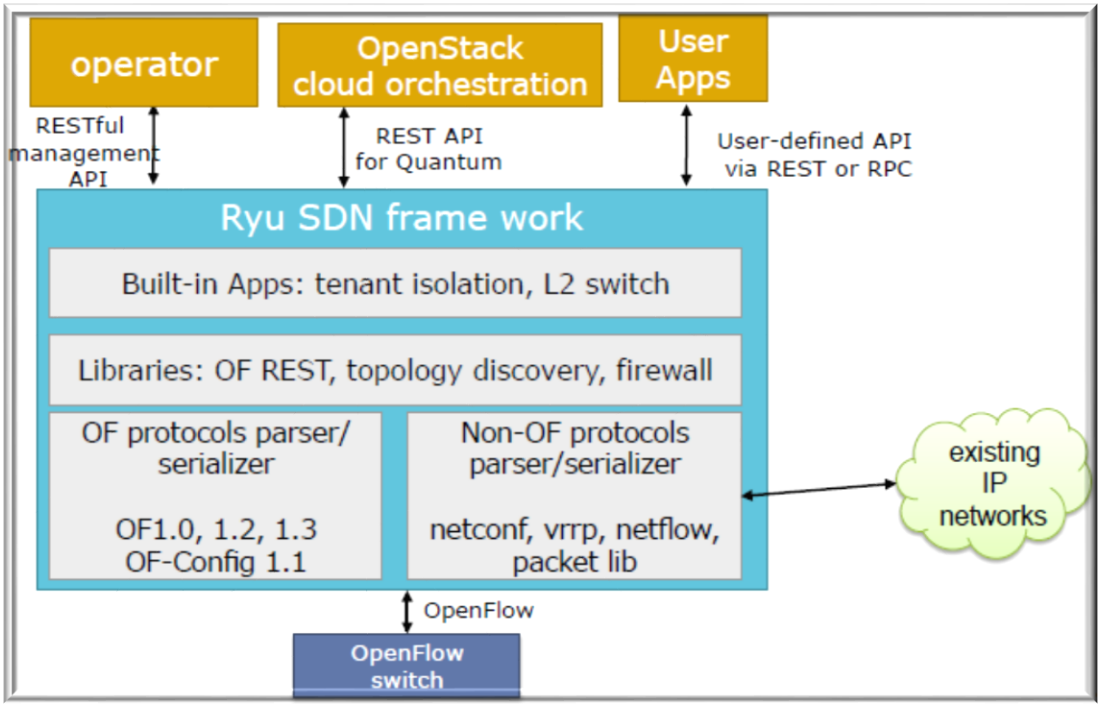

Figura 10: Architettura RYU

3.3. LIBRERIE

Ryu dispone di un'impressionante quantità di librerie, spaziando dal supporto verso più protocolli di southbound, a numerose operazioni per processare i pacchetti di rete.

Per quanto riguarda i protocolli southbound, Ryu supporta OF-Config, Open vSwitch Database Management Protocol (OVSDB), NETCONF, XFlow (Netow e Sow) ed altri protocolli di terze parti. La libreria di Ryu riferita ai pacchetti aiuta nell'analisi e nella realizzazione di vari pacchetti dei protocolli, come VLAN, MPLS, GRE, eccetera.

3.4. PROTOCOLLO E CONTROLLER OPENFLOW

Ryu include una libreria di codifica e decodifica del protocollo OpenFlow, che supporta fino alla versione 1.4 .

Uno dei componenti chiave dell'architettura di Ryu è il controller Open-Flow, che è il responsabile della gestione degli switch OpenFlow utilizzati per configurare i flussi, gestire gli eventi, eccetera. Il controller OpenFlow è una delle sorgenti di eventi interne nell'architettura di Ryu.

La tabella seguente, in Figura 11, riassume i messaggi, la struttura e le corrispondenti API del protocollo OpenFlow di Ryu.

Controller to Switch Messages	Asynchronous Messages	Symmetric Messages	Structures
Handshake, switch-config, flow-table-config, modify/read state, queue-config, packet-out, barrier, role-request	Packet-in, flow-removed, port-satus, and Error.	Hello, Echo-Request & Reply, Error, experimenter	Flow-match
send_msg API and packet builder APIs	set_ev_cls API and packet parser APIs	Both Send and Event APIs	

Figura 11: Messaggi, struttura e API di OpenFlow in Ryu.

3.5. RYU MANAGER

Il Ryu manager è l'eseguibile principale. Quando viene avviato, ascolta tramite un indirizzo IP specificato (ad esempio 0.0.0.0) ed una porta specificata (6633 di default). Osservando ciò, ogni switch OpenFlow (hardware o Open vSwitch oppure OVS) può connettersi al Ryu manager. Il manager è il componente fondamentale di tutte le applicazioni di Ryu, che ne ereditano sempre la classe app manager.

3.6. RYU NORTHBOUND

Al livello API, Ryu include un plug-in di Neutron, per quanto riguarda Open-Stack, che supporta le configurazioni VLAN e GRE-based. Ryu è inoltre predisposto di una interfaccia REST per le relative operazioni di OpenFlow.

3.7. APPLICAZIONI DI RYU

Ryu è distribuito con svariate applicazioni come semplici switch, router, firewall, GRE tunnel, VLAN, eccetera. Le applicazioni sono entità single-threaded, ovvero eseguite singolarmente una alla volta, che implementano varie funzionalità. Per comunicare tra loro, tali applicazioni si inviano vicendevolmente eventi asincroni.

L'architettura funzionale di un'applicazione Ryu è mostrata in Figura 3.4

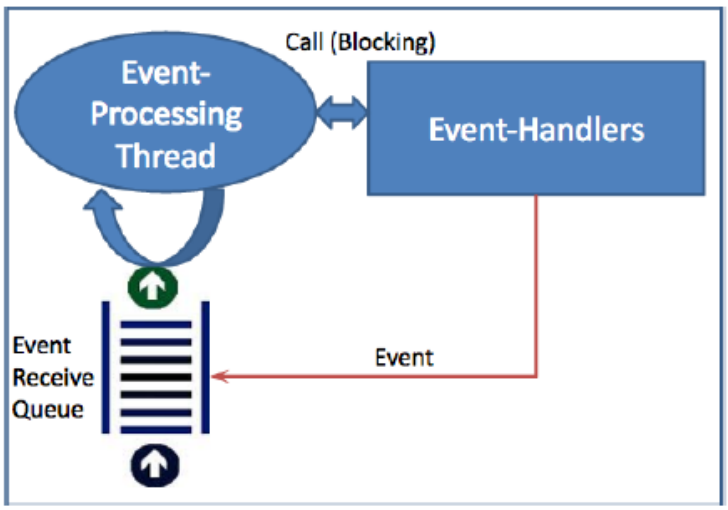

Figura 12: Architettura funzionale di un'applicazione di Ryu.

4. MININET

In questo capitolo si vuole introdurre il software Mininet, capace di creare rapidamente, efficientemente e con risorse limitate una rete virtuale basata sul concetto di SDN.

4.1. IL SIMULATORE

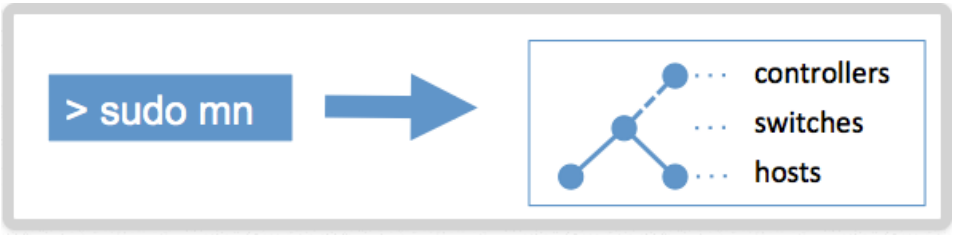

FIGURA 13: Logo Mininet

Nell'ambito dello sviluppo di OpenFlow e di SDN, si è manifestata presto l'esigenza di effettuare dei test, così 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 è stato sviluppato con lo scopo di riprodurre il funzionamento di una rete attraverso la virtualizzazione, con la quale è possibile eseguire test estesi utilizzando risorse limitate.

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 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.

4.2. VANTAGGI

L’utilizzo di Mininet offre sicuramente una serie di vantaggi:

- **Velocità:** l'avvio di una semplice rete richiede pochi secondi.
- **Possibilità di creare topologie personalizzate:** un unico switch, topologie più ampie simili a internet, un centro dati o qualsiasi altra cosa.
- **Possibilità di eseguire programmi veri e propri:** tutto ciò che funziona su linux è disponibile per l’esecuzione sui singoli switch e host creati.
- **Possibilità di personalizzare l’inoltro dei pacchetti:** gli switch di Mininet sono programmabili utilizzando il protocollo

Openflow e le funzionalità testate su di essi possono essere facilmente trasferite in switch reali.

- Mininet può essere eseguito anche su un semplice computer portatile, su un server, su una virtual machine e anche su macchine native Linux.
- **Possibilità di condividere e replicare il codice:** chiunque possegga un computer ha la possibilità di riprodurre il codice.
- **La facilità di utilizzo:** si possono creare ed eseguire esperimenti creando semplici, o anche complessi, script in linguaggio python.
- **Il codice è Open Source:** si può esaminare e modificare il codice sorgente scaricabile al sito https://github.com/mininet
- **Mininet è in fase di sviluppo attivo:** è possibile interagire direttamente con la comunità di sviluppatori

4.3. LIMITAZIONI

Anche se pieno di vantaggi, Mininet presenta anche alcune possibili limitazioni:

- Eseguire una rete su un unico sistema è comodo ma impone alcune limitazioni: bilanciamento di risorse.

- Mininet utilizza un unico kernel Linux per tutti gli host virtuali; questo significa che non è possibile eseguire software che dipende da BSD, Windows, o altri kernel differenti.
- Mininet non crea il controller, se si necessità di un controller personalizzato bisognerà implementarlo per poi poterlo utilizzarlo.
- Per impostazioni predefinite, La rete Mininet è isolata dalla LAN e da internet.
- Per impostazione predefinita, tutti gli host Mininet condividono il file host del sistema e lo spazio PID..
- A differenza di un simulatore, Mininet non ha una nozione forte di tempo virtuale.

Con piccole eccezioni, la maggior parte di queste limitazioni si possono eiliminare con una corretta gestione del codice.

4.4. POTENZIALITÀ

Mininet, con le sue innumerevoli 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.

Grazie all'utilizzo di altre tecnologie, come un semplice analizzatore di protocolli di rete, strumenti di base 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 nell'ambito di reti SDN.

5. IMPELEMTAZIONE

In questo capitolo dopo aver affrontato tutti i concetti fondamentali di SDN e il funzionamento del protocollo OpenFlow, eseguiremo dei test pratici, in un ambiente virtuale, con lo scopo di capire come gestire e monitorare una rete centralizzata SDN con il controller RYU, con l'obbiettivo di rendere la rete efficiente ed ottimizzare le sue performance condizionando il flusso dei pacchetti su percorsi diversi

5.1. AMBIENTI DI SVILUPPO SDN BASATI SU OPENFLOW

Per realizzare i test è stata utilizzata la macchina virtuale pre-configurata disponibile al seguente sito:

http://sdnhub.org/tutorials/sdn-tutorial-vm/

Questa VM contiene il seguente materiale, che permette lo studio e lo sviluppo di tecnologie SDN:

- Controller SDN: OpenDaylight, ONOS, RYU, Floodlight, Floodlight-OF 1.3, POX, e Trema.
- Codici di esempio per Hub, L2 learning switch, monitoraggio del traffico e altre applicazioni.
- Open vSwitch 2.3.0 col supporto per Openow 1.2 - 4.1.

- Mininet: per creare e simulare topologie di reti.
- Wireshark 1.12.1 con il supporto nativo per il filtraggio dei pacchetti OpenFlow.

5.2. . CREAZIONE DI UNA RETE VIRTUALE

In questo esempio si vuole creare, attraverso l'emulatore di rete Mininet, partendo dalla tipologia di rete (figura x) una semplice rete composta da 4 switch e 8 host.

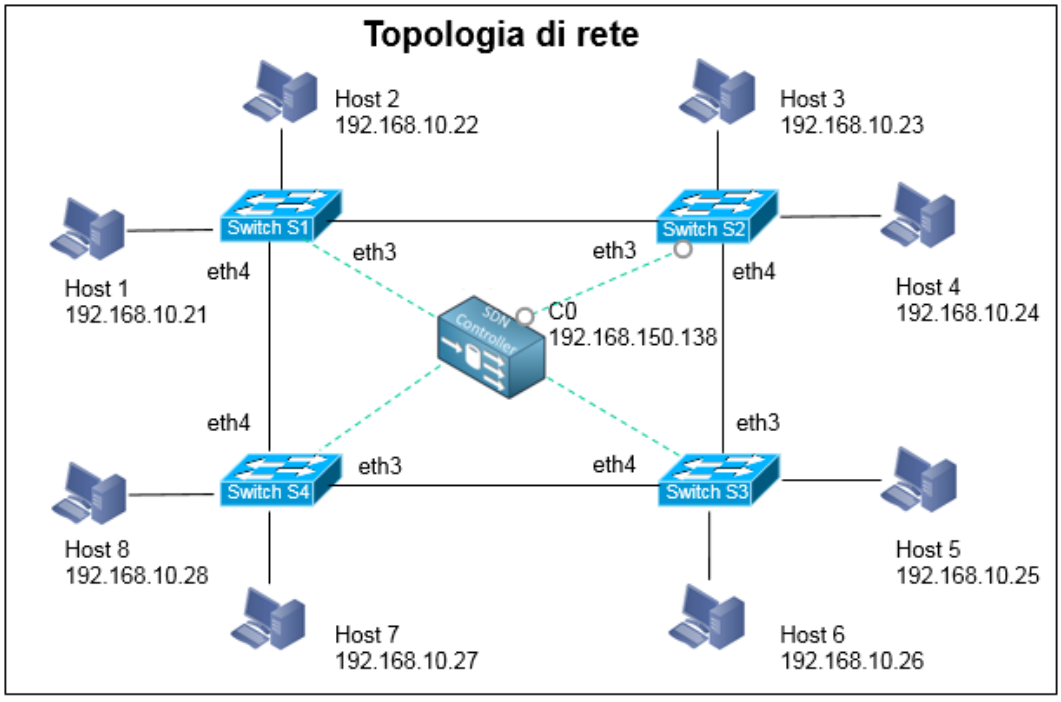

FIGURA 14: Topologia di rete

Per rendere più veloce il processo di implementazione, si è effettuata la creazione della topologia realizzando uno script di codice Python, sfruttando le librerie già presenti in Mininet.

Di seguito lo script utilizzato:

- Import delle librerie API:

```
#!/usr/bin/python

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

- Definizione della classe principale (myNetwork)

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

- Utilizzo del Controller Remoto:

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

- Definizione di Switch Virtuali:

```
info( '*** Add switches\n')
s1 = net.addSwitch('s1')
s2 = net.addSwitch('s2')
s3 = net.addSwitch('s3')
```

```
s4 = net.addSwitch('s4')
```

- Creazione di 8 Host con i relativi ip (rete utilizzata 192.168.10.0/24)

```
info('*** Add hosts\n')
h1 = net.addHost('h1', ip='192.168.10.1')
h2 = net.addHost('h2', ip='192.168.10.2')
h3 = net.addHost('h3', ip='192.168.10.3')
h4 = net.addHost('h4', ip='192.168.10.4')
h5 = net.addHost('h5', ip='192.168.10.5')
h6 = net.addHost('h6', ip='192.168.10.6')
h7 = net.addHost('h7', ip='192.168.10.7')
h8 = net.addHost('h8', ip='192.168.10.8')
```

- Definizione delle connessioni tra Host e switch con relativa banda di collegamento.

```
info('*** Add links\n')
net.addLink(h1, s1, bw=10)
net.addLink(h2, s1, bw=10)
net.addLink(h3, s2, bw=10)
net.addLink(h4, s2, bw=10)
net.addLink(h5, s3, bw=10)
net.addLink(h6, s3, bw=10)
net.addLink(h7, s4, bw=10)
net.addLink(h8, s4, bw=10)
net.addLink(s1, s2, bw=10)
net.addLink(s2, s3, bw=10)
net.addLink(s3, s4, bw=10)
net.addLink(s4, s1, bw=10)
```

- Avvio della rete e della Command line:

```
info('*** Starting network\n')
net.start()
CLI(net)
net.stop()
if __name__ == '__main__':
setLogLevel('info')
myNetwork()
```

Una volta terminato lo script, è sufficiente salvarlo con estensione .py (il nostro file è stato chiamato test_newnet.py) e lanciarlo con il seguente comando:

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

Una volta eseguito, se tutto è andato a buon fine e non sono presenti errori, da Command Line Interface di Mininet, possiamo verificare la correttezza dei nodi creati, nonché delle relative connessioni della Rete con i comandi nodes, net e dump (figura 15)

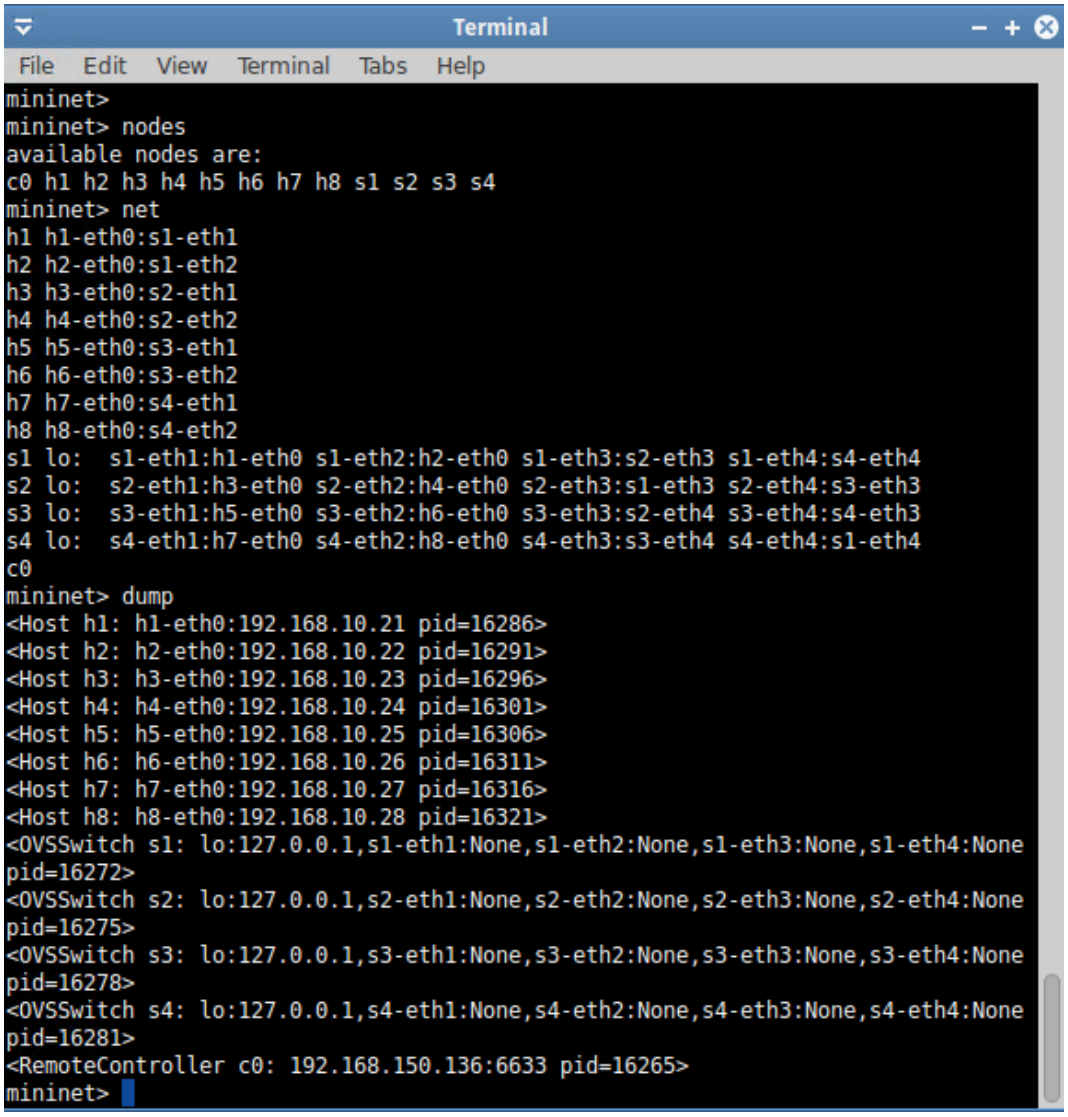

Figura 15: Comandi nodes net e dump di mininet

Per valutare il funzionamento della rete creata, utilizziamo i comandi ping e pingall e considerando che gli switch creati simulano in tutto e per tutto il funzionamento di quelli reali, ci aspettiamo che i test ping tragli host falliscano. Infatti, come da figura 14 si può notare il 100% dei ping droppati.

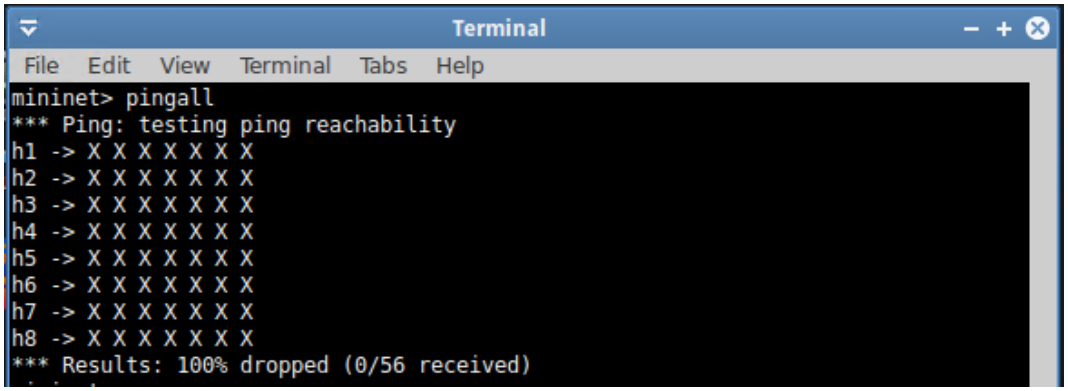

Figura 16: Esempio di pingall

Tale risultato è dovuto al fatto che i nostri switch sono collegati a forma di anello e la mancanza di un protocollo di controllo causa la moltiplicazione dei pacchetti di broadcast generati dalle ARP request dei comandi ping, degenerando in broadcast storm che bloccheranno del tutto la rete.

Come accennato in precedenza essendo gli switch virtuali simili agli apparati reali si è deciso di attivare il protocollo spanning tree (STP), utilizzando i seguenti comandi (openVswitch) nel seguente modo:

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

avviato il protocollo dopo qualche minuto, una volta che l'STP si è creato una alberatura della rete priva di loop i ping hanno dato esito positivo (figura 17).

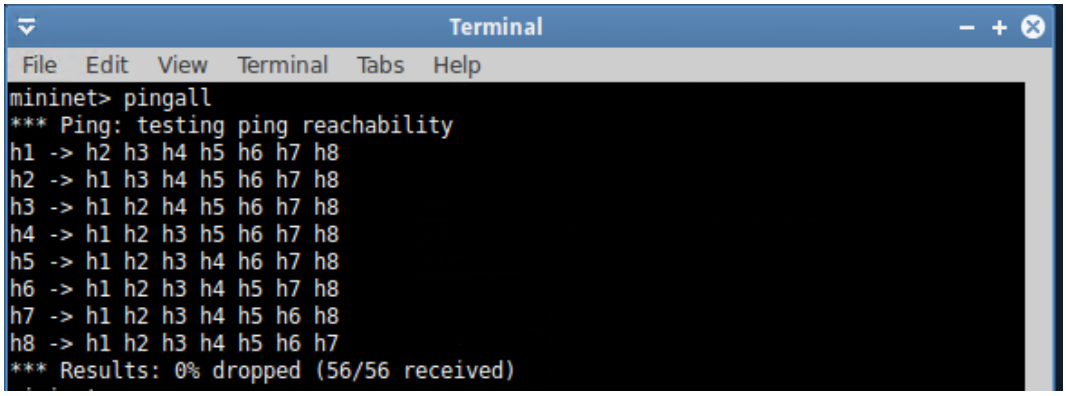

Figura 17: pingall lanciato dopo aver attivato il protocollo STP

Ciò si ottiene mediante la creazione di una gerarchia di bridge. Un bridge viene individuato come radice dell'albero coprente ("root bridge"), e una parte dei collegamenti tra bridge disponibili viene messa in standby, portando in stato "BLOCKING" alcune delle porte dei bridge, denominate alternate port (dall'inglese: porta alternativa)(AP).

Nel caso in cui un nodo diventi irraggiungibile, oppure cambi il costo di connessione, il bridge cercherà di arrivare al nodo attivando i percorsi alternativi (AP) che sono in stand-by, ripristinando in questo modo la connettività completa della rete (se possibile).

Una volta verificata la raggiungibilità tra i vari Host, verifichiamo la banda disponibile e per farlo utilizziamo iperf con il quale andiamo a misurare la velocità di trasferimento di dati tra due host. Con il comando xterm, avviamo due finestre per due host H1 H2 (terminali) dalle quali possiamo eseguire i comandi.

Facciamo una prima prova lanciando un iperf tra due soli host, uno connesso allo switch 1 e l'altro allo switch 3. Essendo la rete priva di traffico, ci aspettiamo che la velocità di trasferimento sia all'incirca quella fisicamente disponibile (10Mbits/sec).

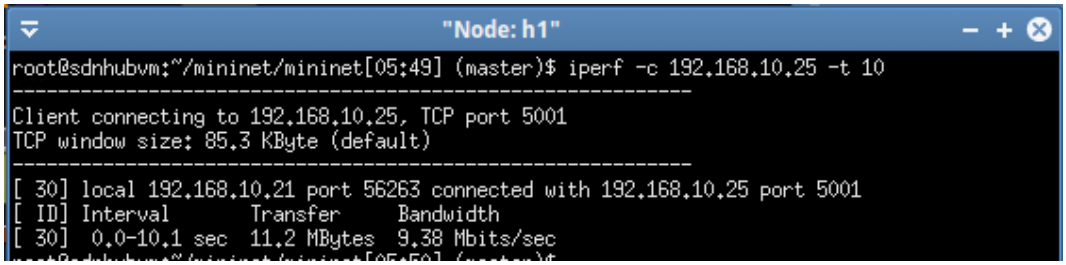

Figura 18: Utilizzo del comando iperf dal Host1.

Infatti, come da figura 18 otteniamo un trasferimento dati da H1 ad H5 con una media di utilizzo di banda di circa 9,38Mbit/sec.

Se, invece proviamo ad effettuare un test di banda simultanio tra H1 vs H5 e tra H2 vs H6, otteniamo i seguenti risultati:

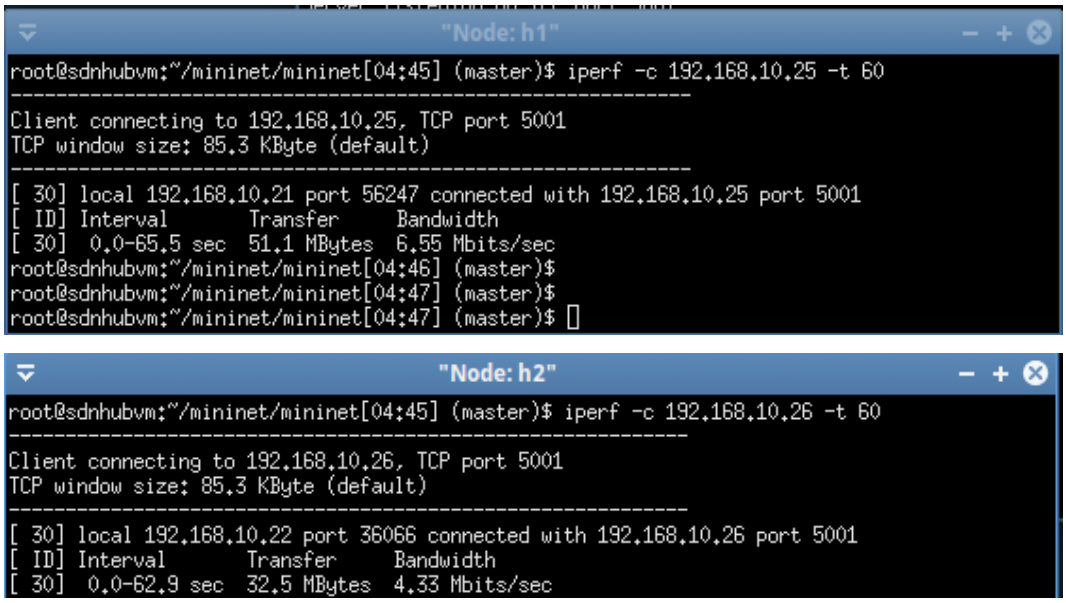

Figura 19: Comando iperf lanciato simultaneamente.

Sommando i risultati ottenuti 6,55 e 4,33 otteniamo all'incira una banda di circa 10Mbit/sec, ciò dimostra che hanno utilizzato lo stesso percorso di rete.

Questo risultato è dovuto al utilizzo del protocollo STP che per evitare loop di rete disabilità alcune porte, generando una gerarchia tra gli

switch, designandone uno come root bridge, per verificalo utilizzamo il seguente comando:

ovs-vsctl list Bridge

da cui ricaviamo le seguenti informazioni, necessarie per stabilire il funzionamento del protocollo STP.

Per i nostri switch otteniamo:

S1

```
status              : {stp_bridge_id="8000.362767077a44", stp_designated_root="8000.362767077a44", stp_root_path_cost="0"}
stp_enable          : true
```

S2

```
status              : {stp_bridge_id="8000.a69c20257f4c", stp_designated_root="8000.362767077a44", stp_root_path_cost="2"}
stp_enable          : true
```

S3

```
status              : {stp_bridge_id="8000.b68bf75e9144", stp_designated_root="8000.362767077a44", stp_root_path_cost="4"}
stp_enable          : true
```

S4

```
status              : {stp_bridge_id="8000.e6c1233cdc48", stp_designated_root="8000.362767077a44", stp_root_path_cost="2"}
stp_enable          : true
```

Dagli show raccolti viene individuato come designed root lo switch S1 (il valore id:8000.362767077a44 corrisponde proprio allo switch 1) che ha il path_cost più basso (0) mentre, agli switch S2 ed S4 viene assegnato un valore pari a 2 e per ultimo sui cui vengono disabilitate le porte un valore pari a 4.
Per risolvere la problematica potremmo agire direttamente sui singoli switch, effettuando una configurazione manuale sulle singole porte, in maniera però non ottimale, generando uno spreco di tempo e di risorse,

proprio in questo caso arriva in nostro soccorso il protocollo Openflow il quale, permettendo la comunicazione tra switch e controller, è in grado di semplificarci il lavoro delegando al software la scelta dei percorsi migliori, per poi comunicarli agli switch istruendoli con apposite tabelle, e lasciando a noi il solo compito di programmarlo nel modo più opportuno.
Si potrebbe agire sul controller, andando ad inserire i seguenti comandi:

```
sudo ovs-ofctl add-flow Sx ip,nw_dst=192.168.10..X,actions=output:Y
sudo ovs-ofctl add-flow Sx arp,nw_dst=192.168.10.X,actions=output:Y
```

dove con Sx s'indica il nome dello switch, con X l'ip del host e con Y la porta verso cui trasmettere i pacchetti di tipo ARP o IP destinati all'indirizzo 192.168.10.X. In questo caso le regole verrebbero aggiunte mediante protocollo openflow ma in pratica manualmente nei singoli switch.
Però, il nostro scopo, è quello di creare un Controller a cui delegare il compito di inviare le regole nel momento in cui gli giunge una richiesta. In pratica, non appena lo switch riceve un pacchetto in ingresso, lo inoltra al controller mediante il secure channel dedicato al protocollo Openflow e genera un messaggio OpenFlow Protocol (OFP) di tipo PaketIn ed Il controller, a sua volta elabora il messaggio, inviando un messaggio di tipo PacketOut oppure di tipo Flow mod con il quale istruisce lo switch.

5.3. IMPLEMENTAZIONE DEL CONTROLLER DI RETE

Prefissata la nostra topologia di rete, si è deciso di implementare un controller che permetta, oltre al corretto raggiungimento di tutti i nostri host, anche la possibilità di utilizzare tutti gli switch sfruttandone la massima capacità. Come già accennato nel paragrafo precedente, si vuole riuscire ad eseguire due connessioni H1 vs H5 ed H2 vs H6, ottimizzandone la capacità di trasferimento.

La soluzione migliore è quello di instradare il flusso dati delle due connesioni su percorsi diversi istruendo gli switch in modo tale, ad esempio, di fare passare la prima connessione direttamente dal collegamento tra S1,S2 e S3, la seconda transitando per S1,S4 per poi arrivare ad S3.

In figura 20 viene mostrato il percorso dei flussi di dati:

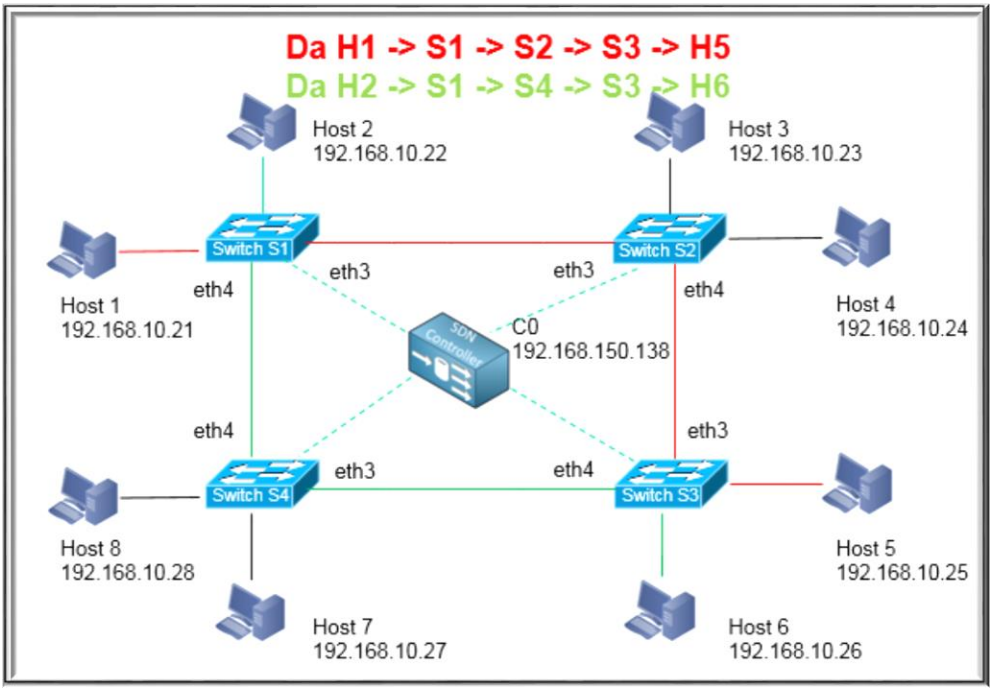

Figura 20: flusso dati.

Tutto questo è stato possibile grazie al Controller Ryu ed al l'utilizzo dei file ofctl_REST.py e simple_switch_13.py, che opportunamente modificati hanno consentito di poter instradare i pacchetti verso porte diverse in base alla destinazione ip, ottenendo il risultato di poter utilizzare al meglio le risorse.

Avviato il controller con le istruzioni opportuanmente modificate ripetiamo il test di invio dati simultiano da due Host differenti, ottenendo il seguente risultato:

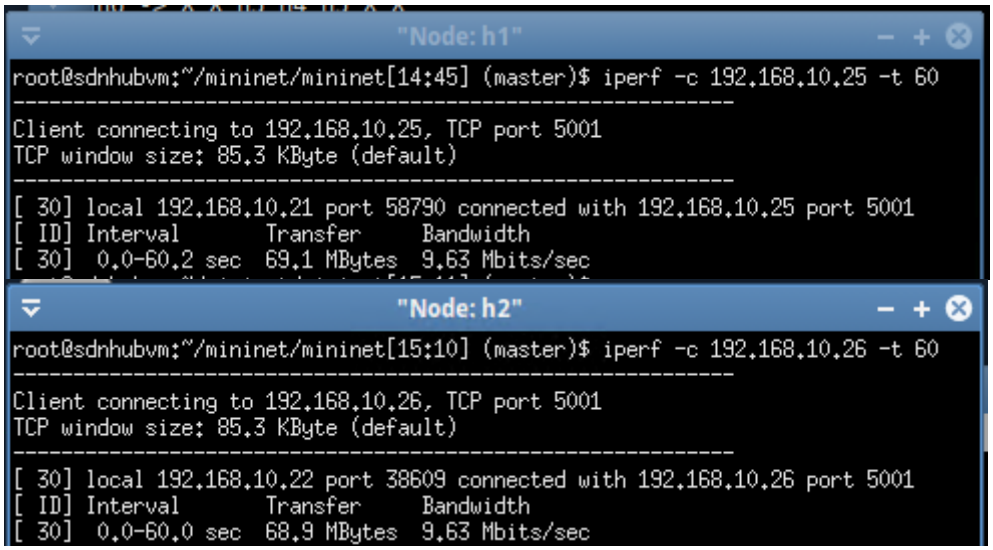

Figura 21: Comando iperf lanciato simultaneamente.

Come si evince dagli screenshot, gli switch una volta istruiti consento agli host di utilizzare tutta la banda disponibile, instradando il flusso dati attraverso percorsi diversi.

5.4. MONITORAGGIO DI RETE

Grazie alle grandi potenzialità di Python è possibile realizzare delle applicazioni che ci consentono di visualizzare la topologia di rete, di monitorarla, gestire il bilanciamento di carico dei server e gestire le statistiche sul traffico generato.

Per verificare quanto indicato, utilizziamo il Kit messo a disposizione da SDNHUB:

http://sdnhub.org/releases/sdn-starter-kit-ryu/

come indicato in precedenza il tutto viene realizzato in Python per la piattaforma del controller Ryu. Per procedere alla sperimentazione sulla VM su cui abbiamo effettuato i test, andiamo a scaricare il seguente kit:

```
$ git clone https://github.com/osrg/ryu
$ cd ryu/ryu/app
$ git clone https://bitbucket.org/sdnhub/ryu-starter-kit
sdnhub_apps
```

Una volta completato il download per eseguire il controller e le applicazioni procediamo nel seguente modo:

```
$ cd ~ / ryu
$ ./ryu/app/sdnhub_apps/run_sdnhub_apps.sh
```

Per effettuare i primi test, se il tutto è stato installato ed avviato correttamente si può accedere al GUI dal link: http://ip-address-of-controller: 8080.

Ecco alcune schermate delle pagine di configurazione:

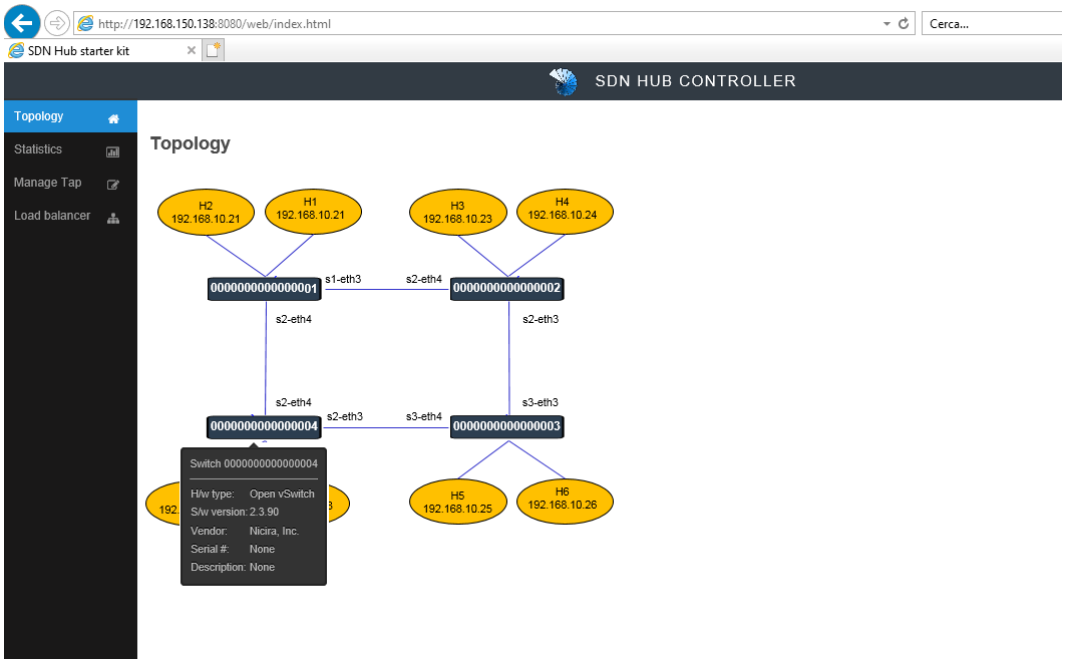

Figura 22: topologia di rete

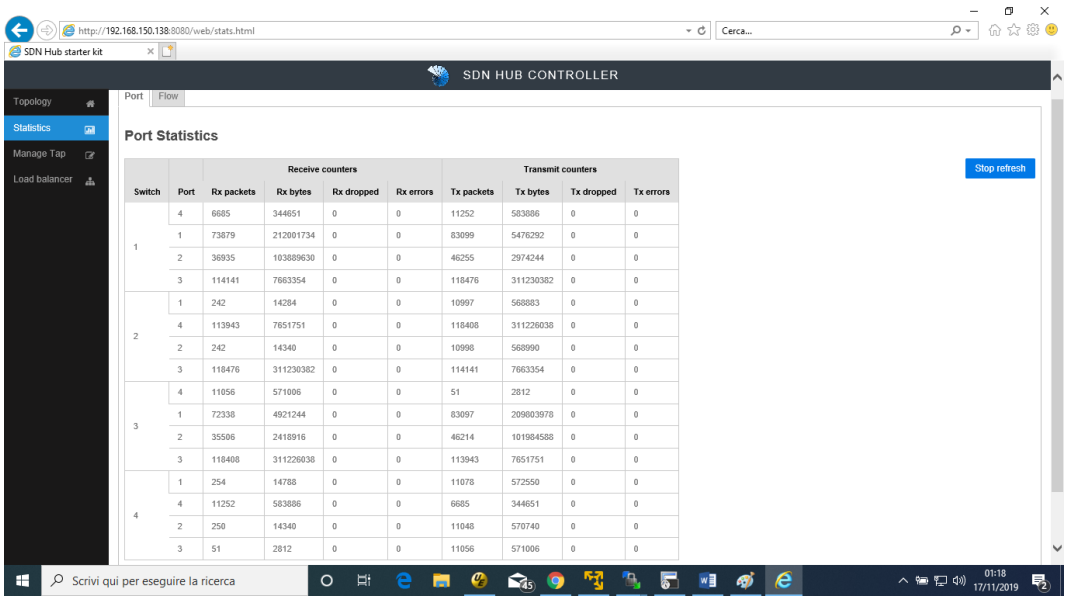

Figura 23: statistiche di utilizzo

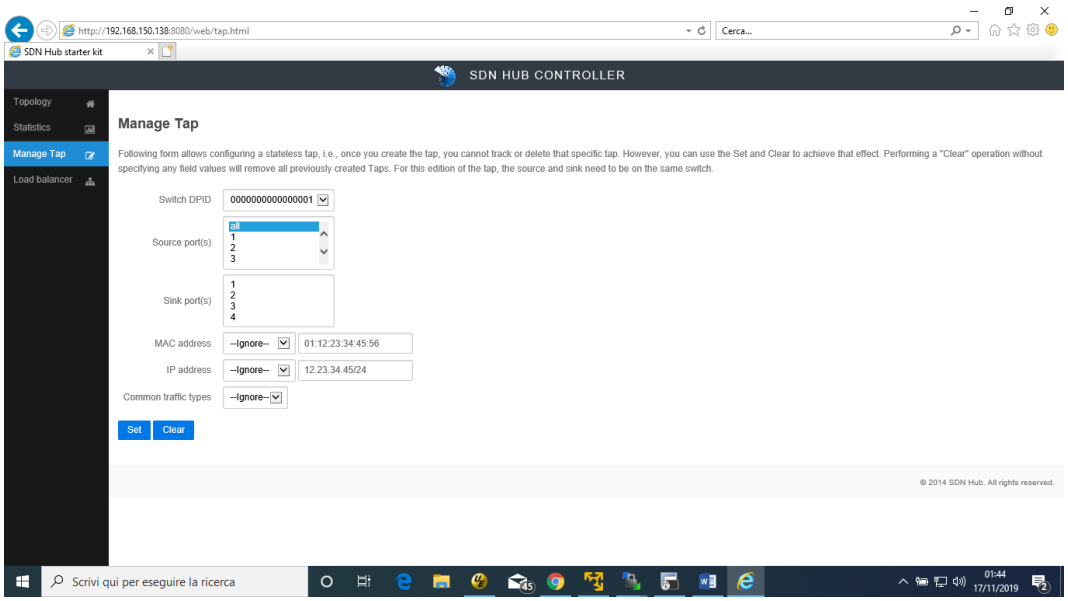

Figura 24:Manage Tap

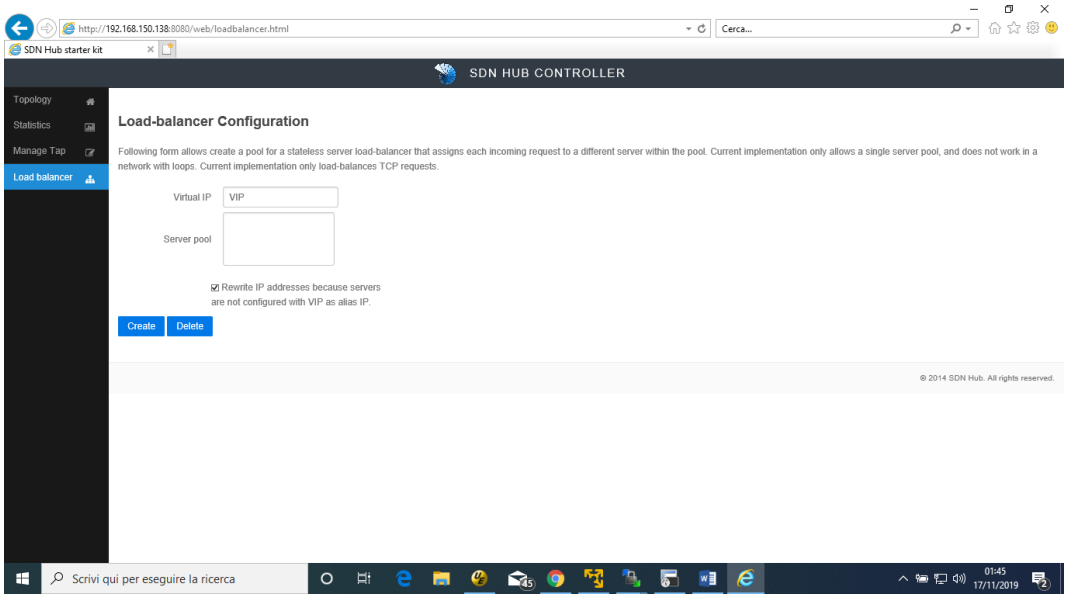

Figura 25: Load Balancer

CONCLUSIONI

Utilizzare una struttura basata sul paradigma SDN compatibile con il protocollo OpenFlow rende la gestione della rete, in termini economici, estremamente vantaggioso, infatti, offre all'utilizzatore finale la possibilità di migliore le performance di banda, e di avere una gestione semplificata sia dal lato implementativo che di manutenzione dinamica.

Tutto il lavoro svolto durante questa tesi, hanno permesso di dimostrare come sia possibile utilizzare efficientemente la nuova architettura SDN ed il protocollo Open-Flow. Utilizzando il software Mininet e il controller Ryu, ci hanno permesso di emulare una topologia di rete funzionante in maniera rapida su un semplice PC, su cui abbiamo effettuato dei semplici test di raggiungibilità e di performance, ottenendone risultati esportabili in un ambiente reale.

Problemi come loop, broadcast storm o manipolazione del instradamento dei flussi dati, come da lavoro svolto, possono essere affrontati e risolti in maniera molto semplice con un approccio software. Il paradigma SDN comporterà nel prossimo futuro una fase di forte innovazione, già molte aziende del settore stanno provvedendo all'aggiornamento dei device in modo da poter utilizzare il protocollo OpenFlow.

INDICE DELLE FIGURE

BIBLIOGRAFIA

https://en.wikipedia.org/wiki/Software-defined_networking

http://www.networxsecurity.org

https://www.citrix.com/content/dam/citrix/en_us/documents/oth/sdn-101-an-introduction-to-software-defined-networking-it.pdf

https://www.sdxcentral.com/articles/contributed/sdn-openflow-tcamneed-to-know/2012/07/

http://home.deib.polimi.it/cesana/teaching/FIR2016-2017/lezioni/SDN.pdf

http://archive.openflow.org

http://openvswitch.org

http://mininet.org/download/

http://mininet.org

https://www.virtualbox.org

https://www.sdxcentral.com/sdn/definitions/sdn-controllers

http://searchsdn.techtarget.com/feature/A-primer-on-northbound-APIs-Their-role-in-a-software-defined-network

https://it.wikipedia.org/wiki/Web_Server_Gateway_Interface

https://www.getpostman.com

https://www.mamp.info/en/

http://ryu.readthedocs.io/en/latest/app/ofctl_rest.html

http://sdnhub.org/

http://osrg.github.io/ryu/