UNIVERSITÀ DEGLI STUDI DI PADOVA

Scuola di Ingegneria

Corso di Laurea in Ingegneria dell'Informazione

Studio e implementazione di una Software Defined Network basata su OpenFlow

Laureando

Matteo Maso

matr. 1067486

Relatore

Prof. Andrea Zanella

Anno Accademico 2015-2016

RINGRAZIAMENTI

A tutte le persone che mi hanno spinto ad arrivare fin qui...
ed a tutte quelle che credevano non ci riuscissi.

Sommario

Nell'approccio tradizionale al networking, la maggior parte delle funzionalità di rete vengono svolte da apparati dedicati come switch o router. A differenza degli switch, i router sono in grado di svolgere praticamente qualsiasi funzione sui pacchetti in arrivo, risultando però molto più lenti e costosi.

Tramite OpenFlow si intende standardizzare le interfacce di rete utilizzate, rendere gli apparati di rete mero hardware e spostare l'intero controllo logico delle funzionalità di rete su uno o più apparecchi dedicati chiamati "controller", andando così a formare una SDN.

Lo studio che ne risulta, ha principalmente tre obiettivi: illustrare il concetto di SDN, con tutti i vantaggi che esso introdurrebbe, mostrare poi come OpenFlow contribuisca all'evoluzione delle applicazioni pratiche e dimostrare, tramite un'esperienza concreta le differenze tra una rete tradizionale ed una SDN.

Indice

Elenco delle figure

Capitolo 1

Introduzione

1.1 La rete

Internet è una rete di telecomunicazione a pacchetto, con una topologia molto complessa e articolata, che viene comunemente definita una rete di reti. Essa permette l'interconnessione a livello mondiale di un numero di utenti in costante crescita. La rete internet dal punto di vista fisico è composta dai seguenti elementi:

- **Hardware:** macchine collegate alla rete che forniscono servizi all'utenza e terminali attraverso i quali gli utenti possono accedere a questi servizi. Inoltre fanno parte di questa categoria tutti gli apparati come per esempio switch e router, dedicati al veicolamento dei pacchetti nella giusta direzione all'interno della rete.

- **Collegamenti:** tutti i mezzi fisici che permettono di creare un canale di comunicazione tra i vari elementi hardware della rete, in modo da ottenere un unica rete connessa.

- **Protocolli:** con il termine protocollo intendiamo la definizione formale a priori delle modalità di interazione che due o più apparecchiature elettroniche collegate tra loro devono rispettare. Essendo presenti ele-

menti eterogenei, all'interno della rete sono presenti più protocolli con funzioni diverse che agiscono a vari livelli e tra gli elementi della rete.

Possiamo modellizzare la rete attraverso l'Open System Interconnection (meglio conosciuto come modello ISO/OSI), uno standard di riferimento per reti di calcolatori, stabilito nel 1978 dall'International Organization for Standardization (ISO). L'architettura protocollare ISO/OSI definisce una struttura a strati composta da 7 livelli, che coprono tutte le funzionalità della rete, seguendo un modello logico-gerarchico. A livello implementativo lo standard *de facto* per l'architettura di rete a livelli è il modello TCP/IP, che riprende solo in parte la suddivisione logica dello standard ISO/OSI.

Un'ulteriore caratteristica importante dell'intera struttura di rete, sia a livello logico che a livello organizzativo, è che tutti i suoi elementi sono organizzati in modo gerarchico. La gerarchia tra elementi è stata introdotta per due principali ragioni:

- **Scalabilità**: se l'architettura fosse piatta, non solo ogni elemento della rete dovrebbe conoscere l'intera topologia per comunicare in modo efficente, ma le tabelle in cui andrebbero memorizzati i dati riguardo la struttura, risulterebbero computazionalmente complesse da elaborare. La mole di dati da immagazzinare inoltre aumenterebbe esponenzialmente rispetto un aumento lineare degli elementi. L'hardware adibito all'instradamento dei pacchetti a questo punto si troverebbe ad elaborare una grande mole di dati per svolgere le sue funzionalità, comportando così un rallentamento significativo all'interno dell'intera struttura.

- **Autonomia amministrativa**: un instradamento non gerarchico imporrebbe l'utilizzo di un'unica entita globale di controllo o perlomeno di un algoritmo decisionale univoco al quale tutti gli apparati facciano riferimento. Se la prima opzione non è fisicamente realizzabile, la seconda potrebbe esserlo, ma comporterebbe un ingente lavoro di aggiornamento sia ad ogni piccola modifica del software, sia per ogni piccolo intervento nella topologia di rete.

Negli ultimi anni entrambi i problemi sono stati risolti aggregando i router in "regioni" o "sistemi autonomi" (*Autonomous Systems* - AS). I router che gestiscono l'instradamento all'interno di un AS sono detti "router interni" e processano algoritmi di instradamento che appartengono alla categoria degli *Interior Gateway Protocol* (IGP). D'altra parte i router che instradano i pacchetti verso destinazioni al di fuori dell'AS sono detti "router esterni" o "border gateway" e utilizzano un algoritmo di instradamento appartenente alla categoria degli *Exterior Gateway Protocols* (EGP).

1.2 Evoluzione della rete e delle sue necessità

Nell'ultimo decennio abbiamo assistito a un cambiamento radicale per quanto riguarda la tipologia di traffico che transita nella rete. Come si evince dalla Figura 1.1 a pagina 5, nel 2009 il traffico dati ha superato quello della voce.
Tuttavia oltre alla tipologia di dati, è mutato notevolmente anche il tipo di utenza: se inizialmente la rete era stata sviluppata all'interno di Università con fini accademici, industriali e militari, ora, più di un sesto della popolazione mondiale con le esigenze più varie, desidera avere un accesso ad Internet. Questi cambiamenti hanno portato i fornitori di servizi della rete (Internet Service Provider - ISP),[1] al continuo potenziamento e perfezionamento di un'infrastruttura preesistente. Negli anni gli ISP sono ricorsi ad hardware sempre più performanti per due aspetti fondamentali: da un lato il bacino di utenza in crescita causa un incremento del traffico da veicolare, ma dall'altra parte l'aggiunta e l'evoluzione delle funzionalità che la rete fornisce richiedono capacità di calcolo sempre maggiori.

Tutto questo ha contribuito alla *"Ossificazione della rete"*, fenomeno per il quale la rete è diventata una struttura sempre più rigida e complessa, nella quale anche un minimo cambiamento richiede ingenti risorse.

[1] Gli ISP sono aziende che posseggono l'infrastruttura di internet, e che sotto la stipula di un contratto forniscono l'accesso ad internet alle singole utenze.

L'entità del traffico e la velocità del suo sviluppo aveva permesso in passato la nascita di aziende incentrare univocamente sulla fornitura di connettività. Negli ultimi anni tuttavia sono nati alcuni fenomeni i cui bisogni sono molto più complessi ed articolati, necessitando quindi di una rete più snella e configurabile per essere soddisfatti. Questi fattori fanno si che la rete intesa come un entità rigida sia ormai alla fine dei suoi giorni, c'é bisogno quindi di un nuovo tipo di rete, dinamica, facilmente configurabile e riprogrammabile.

In un primo momento le aziende del settore hanno cercato di ovviare a questi problemi sviluppando nuove apparecchiature hardware da fornire agli ISP, dotate di un "Control-plane" ossia un'interfaccia software per controllare il flusso di dati in modo da velocizzare le operaziondi di aggiornamento o riconfigurazione di elementi topologici.

Un altro aspetto fondamentale nello studio dell'evoluzione della rete riguarda la localizzazione delle connessioni in quanto si è compreso che una grossa fetta di traffico rimane confinata all'interno dei data-center stessi (Figura 1.2 a pagina 6). È proprio per questo motivo che vediamo i data center al primo posto per l'introduzione e lo sviluppo della tecnologia SDN. Essi hanno incrementato vertiginosamente la virtulizzazione al loro interno, in modo da implementare simultaneamente sempre più funzionalità, tuttavia per mantenere alte le prestazioni sono ricorsi all'introduzione di un controllore SDN e stanno sviluppando sempre più questa tecnologia.
Ispirati dalle parole di Marc Andreesen,[2] *"software is eating the world"*, il mondo accademico ha per primo realizzato come il Software Defined Networking (SDN) con la virtualizzazione delle funzioni di rete (Network Function Virtualization - NFV) siano due strumenti in grado di far evolvere la rete in modo da soddisfare e supportare le nuove esigenze del mondo IT.

[2]Marc Andreessen (Cedar Falls, 9 luglio 1971) è un informatico e imprenditore statunitense. È anche conosciuto come coautore di Mosaic, il primo web browser a essere largamente utilizzato, e cofondatore di Netscape Communications Corporation.

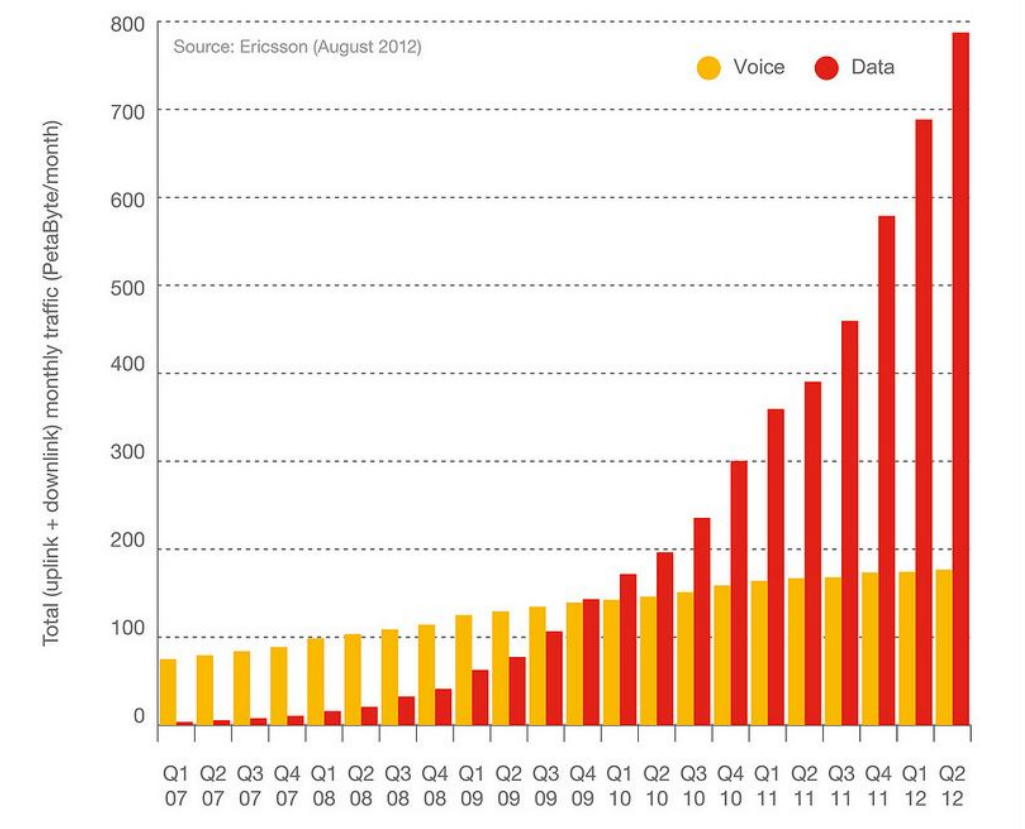


Figura 1.1: *Andamanto temporale del traffico dati e voce a confronto* [3]

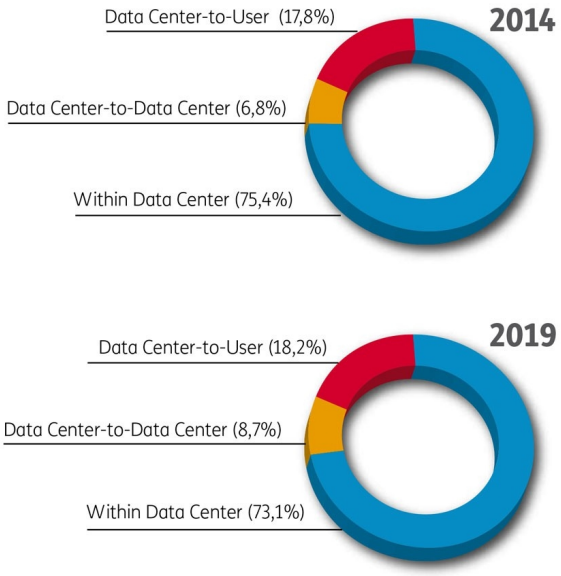


Figura 1.2: *Traffico di rete in relazione ai Data-Center* [2]

1.3 Software Defined Network e Virtualizzazione

Il concetto di SDN è nato nel 2005 quando A. Greeberg e G. Hjalmtysson scrissero un articolo nel quale parlavano di una tabula-rasa dei concetti fondamentali di networking, in vista di nuovo approccio a favore della programmabilità e del controllo.

SDN nell'accezione più diffusa del termina si propone come metodo per rendere programmabili via software gli elementi adibiti all'instradamento dei pacchetti all'interno della rete. SDN entra in gioco introducendo opportuni livelli di astrazione, ai quali accedere attraverso l'uso di interfacce di controllo (Application Program Interface - API).

Le prime implementazioni pratiche si sono viste all'interno delle università, le quali hanno cominciato a sviluppare l'idea di SDN per soddisfare un'esigenza ben precisa: studiare nuove tecnologie di rete in modo agile e veloce. Per questo motivo, era necessaria una rete facilmente riconfigurabile,

c'era bisogno quindi di sviluppare una serie di tecnologie atte a rendere ogni elemento agilmente configurabile via software.

Per far fronte agli stessi bisogni si sviluppa contemporaneamente il concetto di virtualizzazione, ossia l'idea di creare delle partizioni virtuali all'interno della stessa infrastruttura fisica. Questo aspetto, che prende il nome di Virtualizzazione delle funzioni di rete (Network Function Virtualizatio - NFV), viene tuttora largamente sviluppato a livello industriale.

Le funzioni di rete come instradamento, contabilizzazione degli accessi e del traffico utenti, sicurezza e molte altre funzioni ancora, fino a qualche anno fà erano implementate tramite apparati harware dedicati. L'utilizzo di macchine dedicate rende complessa la scalabilità, la manutenzione e l'intera gestione della rete. I Service provider (SP), hanno quindi cercato di sviluppare la tecnologia NFV, in modo da ottenere le stesse funzionalità di prima, implementandole però tramite software in modo da sfruttare apparati generici svincolandosi quindi dalla compatibilità hardware.

Nella ricerca per lo sviluppo dei Data-Center, troviamo per la prima volta l'unione di SDN con il concetto di NFV, in modo da sviluppare una sottorete fluida, facilmente riconfigurabile e in grado di soddisfare le continue esigenze dinamiche di mercato. Tuttavia la simbiosi tra SDN e NFV non è ancora una tecnologia ben consolidata, c'è ancora bisogno di un grosso lavoro di standardizzazione in modo da poter implementarla su scala globale.

Capitolo 2

Struttura di una rete basata sul paradigma SDN

Software Defined Networking (SDN) è un architettura di rete emergente che si propone di essere dinamica, pratica, vantaggiosa e flessibile. Quest'architettura si presta così ad essere l'ideale per la larghezza di banda e la natura dinamica delle moderne applicazioni. SDN disaccoppia il controllo della rete dalle funzioni di inoltro offrendo la possibilità al piano di controllo[1] di diventare direttamente programmabile e rendere astratta l'implementazione dell'infrastruttura alle applicazioni e servizi di rete. In questo modo l'Open Network Foundation[2] definisce SDN.

Essendo SDN la miglior tecnologia attuale per far evolvere la propria rete a un livello superiore abbassando i tempi e i costi di gestione, ed essendo le aziende di Data Center sempre in cerca di sviluppare i propri servizi in modo da renderli competitivi, questa tecnologia è stata in-primis introdotta proprio all'interno dei Data Center.

La principale novità sta nel disaccoppiare il luogo in cui vengono prese decisioni riguardo la direzione di inoltro delle informazioni (il piano di controllo - control plane), dal luogo in cui le informazioni vengono spedite fisicamente

[1] Control Plane `https://en.wikipedia.org/wiki/Control_plane`

[2] ONF sito ufficiale `https://www.opennetworking.org/about/onf-overview`

attraverso i link preselezionati (il piano dati - data plane).

2.1 Architettura fondamentale SDN

La struttura logica di SDN prevede la suddivisione in tre livelli di astrazione, come si può vedere nella Figura 2.1 nella pagina seguente.

Nel primo livello (Application layer) troviamo una serie di appicazioni, 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 efficente 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.

2.1.1 Livello applicazione

Idealmente SDN permette alle applicazioni di specificare alla rete le risorse di cui hanno bisogno attraverso l'uso di semplici APIs. L'interfaccia tra un Controllore SDN e il piano applicazioni è chiamata "application-controller plane interface" (A-CPI). Ogni applicazione SDN può inoltre implementare al suo interno un agente A-CPI che permette l'interazione gerarchica con altre applicazioni SDN.

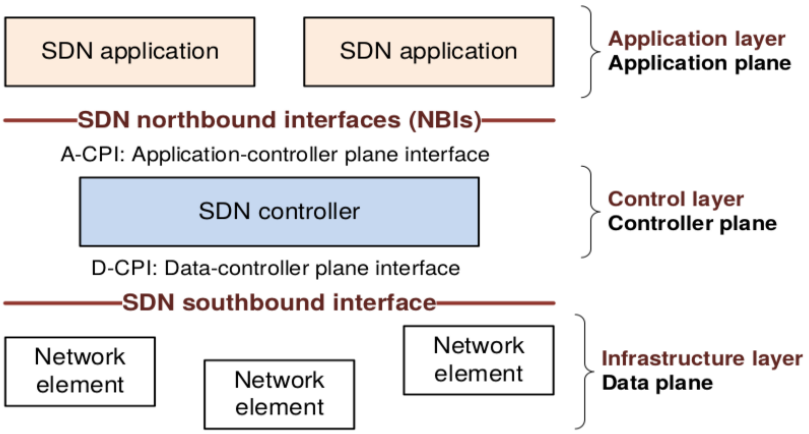


Figura 2.1: *Logica base dell'architettura SDN* [7]

2.1.2 Livello di controllo

Sebbene il controllo sia esercitato a vari gradi anche su altri livelli, il piano di controllo SDN è stato pensato come il raggruppamento di uno o più controllori SDN. Vedremo quindi le componenti funzionali che il livello di controllo SDN deve implementare.

L'architettura non specifica la progettazione interna, ma offre solamente una prospettiva funzionale della stessa. Dal punto di vista logico, il controllore implementato da SDN è centralizzato.

- Un controllore ha tipicamente una visione parziale della rete, tuttavia molto più vasta della visione che potrebbe avere un singolo elemento di rete.

- Non ci sono risorse condivise tra il controllo e altre entità; il controllore SDN vede se stesso come unico fruitore delle risorse virtuali che gli sono state assegnate in fase di configurazione.

Tra le funzionalità di base presenti nel controllo si prevede la possibilità di accedere dall'esterno a tutte le informazioni che esso possiede per modellizzare e gestire la rete; non è esclusa tuttavia l'implementazione di altre funzionalità. Il livello di controllo deve inoltre essere in grado di gestire nu-

merose risorse in relazione tra loro, sebbene esse siano spesso dislocate su piattaforme diverse.

CONTROLLORE

A differenza delle reti tradizionali composte da switch e router, nelle reti SDN è presente un elemento in più, ovvero il controllore SDN. Esso permette di gestire gli altri elementi di rete attraverso il software, rendendo gli altri apparati mero hardware.
Nella letteratura non è specificato se esso debba essere un singolo programma monolitico, piuttosto che un'interazione tra vari software funzionanti su piattaforme diverse, o un unico software replicato e distribuito su tutta la rete.
Il funzionamento di questa architettura è stato comparato a quello di un sistema operativo, in cui il controllore fa da fulcro centrale. Esso permette l'interazione tra i dispositivi di rete che si trovano nel livello inferiore e le applicazioni software situate al livello superiore.
A livello funzionale quando un nuovo pacchetto raggiunge uno switch OpenFlow, quest'ultimo verifica se l'intestazione del pacchetto è presente tra i campi delle tabelle (dette tabelle di flusso) presenti al suo interno: se non lo è, il pacchetto viene inoltrato al controllore. A questo punto il controllore SDN ha pieno potere decisionale riguardo il percorso su cui veicolare il pacchetto per farlo arrivare a destinazione, esso provvede quindi a comunicare allo switch se scartare il pacchetto oppure se inoltrarlo inserendo una nuova voce in tabella.

Questo approccio potrebbe portare ad una congestione della rete, infatti se il controllore non fosse in grado di soddisfare tutte le richieste che gli vengono inoltrare dai vari switch contemporaneamente, esso potrebbe collassare portando al blocco delle comunicazioni. Tale problema si traduce nel decidere la densità e la distribuzione con cui posizionare i controllori all'interno della rete, questo aspetto è ancora in fase di ricerca, ma sono state sviluppate

varie teorie a riguardo.

2.1.3 Piano Dati

Questo livello comprende le risorse e gli apparati fisici il cui scopo è inoltrare e processare specifiche porzioni di traffico all'interno della rete. Questi apparati elaborano direttamente i pacchetti dati prodotti dagli utenti, devono quindi garantire un appropriato livelo di virtualizzazione, sicurezza, disponibilità e qualità nel servizio. In principio gli elementi di rete non hanno libertà decisionale, tuttavia possono essere configurati per rispondere in modo automatico a particolari esigenze, come per esempio connessioni fallite o particolari protocolli come LLDP,[3] STP,[4] BFD[5] e ICMP.[6]

L2 SWITCH

In una rete tradizionale questi dispositivi operano a livello di collegamento, essi svolgono il compito di filtrare, inoltrare e instradare pacchetti dati all'interno di una sottorete. Per poter effettuare queste operazioni, ogni switch deve avere una tabella di inoltro (forwording table), contenente gli indirizzi MAC[7] dei nodi conosciuti, associati alle interfacce su cui i dati devono essere

[3]Il Link Layer Discovery Protocol (LLDP), è un protocollo a livello link utilizzato dai dispositivi di rete per far conoscere agli altri dispositivi all'interno di una rete locale, sia la loro identità che altre informazioni.

[4]Lo Spanning Tree Protocol (STP), viene utilizzato per definire un albero tra gli switch all'interno di una rete in modo da bloccare alcune interfacce per impedire loop sgradevoli nei percorsi.

[5]Bidirectional Forwording Detection (BFD), è un protocollo di rete utilizzato per individuare i guasti nei link di connessione tra dispositivi di inoltro.

[6]L'internet Control Message Protocol (ICMP), è un protocollo di servizio per reti a pacchetto che si occupa di trasmettere informazioni riguardanti malfunzionamenti e controlli tra componenti di una rete internet.

[7]L'indirizzo MAC, detto anche indirizzo fisico, è un codice di 48 bit assegnato in modo univoco dal produttore ad ogni scheda di rete prodotta al mondo, esso è tuttavia modificabile via software.

inoltrati per raggiungere quella destinazione.

Tradizionalmente la tabella di inoltro viene popolata man mano che arrivano pacchetti agli switch: ogni qualvolta un pacchetto trasporta un indirizzo di destinazione non presente in tabella, esso viene inoltrato in broadcast verso tutti i dispositivi connessi. Lo switch deve poi aspettare la risposta da parte del destinatario del pacchetto. Per identificare la rotta tramite la quale inoltrare i pacchetti successivi con ugual destinazione .

In una rete basata su SDN invece, ogni qualvolta un nuovo flusso di pacchetti giunge allo switch, esso provvede all'inoltro del primo pacchetto verso il controllore, il quale dopo aver elaborato il pacchetto, tramite algoritmi di instradamento determina il percorso migliore con cui instradarlo e provvederà all'inserimento delle voci nelle tabelle di flusso dei vari switch.
In una rete basata su SDN, gli switch possono essere di due tipi: puri o ibridi. Gli switch puri OpenFlow non hanno funzionalità decisionali, e fanno completo affidamento sul controller per le decisioni di inoltro. D'altra parte la maggior parte degli switch presenti in commercio sono ibridi, essi supportano il protocollo OpenFlow oltre alle operazioni e protocolli che ogni switch deve implementare per essere definito tale.

2.2 Logica e Interfacciamento

2.2.1 Interfacce di livello superiore

Le Northbound Interfaces (NBI) hanno l'intento di sgravare gli sviluppatori di software dai problemi legati alle diverse implementazioni pratiche attuate dai Service Provider e di rendere le comunicazioni uomo/macchina il più semplice possibili. Le NBI non sono un'unica interfaccia, ma rappresentano tutte le interfacce che si interpongono tra il controller e le applicazioni.

Per quanto riguarda la standardizzazione, diversamente dalle interfaccie di livello superiore, solo di recente è stato stabilito un modello standard [9]. Un importante aspetto del paradigma NBI è l'introduzione del concetto di rimappaggio delle funzionalità. Questo concetto permette ai service provider,

di decidere quali operazioni effettuare in base al sistema specifico su cui si va a lavorare, e ai distributori di connettività di interpretare le risposte del controllore in base alle loro esigenze di distribuzione.

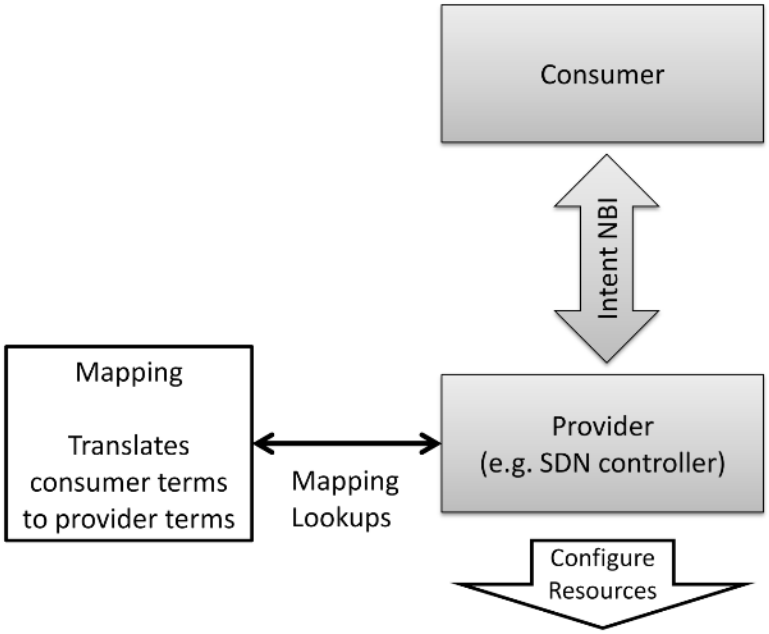


Figura 2.2: *Rappresentazione degli scopi nelle NBI e del rimappaggio* [9]

2.2.2 Interfacce di livello inferiore

Le Interfacce di livello inferiore (Southbound Interfaces), o Control to Data-Plane Interfaces (D-CPI), sono utilizzate per instaurare una comunicazione tra il controllore SDN e gli switch o router della rete. Esse collegano logicamente il piano di controllo al piano dati. A differenza delle interfacce di livello superiore, in questo caso è presente un'implementazione standard universalmente riconosciuta nel protocollo OpenFlow [10].

2.3 Atteggiamento collettivo e protocolli diffusi

I vantaggi generati dal passaggio da una rete tradizionale ad una rete basata su SDN sono molteplici, tuttavia le opinioni e gli atteggiamente dei vari enti, organizzazioni e aziende sono talvolta discordanti, influenzati spesso dal loro ruolo e dai loro interessi economici più che dal reale valore tecnico che l'introduzione di questa tecnologia porterebbe.

Attualmente molti enti di ricerca, prime fra tutti le università, si stanno adoperando per il passaggio a hardware che supportano SDN e OpenFlow. Adottare questa tecnologia offre la possibilità di abbattere i costi, in quanto viene per la maggior parte rilasciata sotto licenza Open Source. Inoltre grazie alla standardizzazione introdotta da OpenFlow le caratteristiche che guideranno la scelta d'acquisto di un dispositivo pittosto che un altro non sarà più la compatibilità a livello di protocolli utilizzati tra gli altri elementi della propria infrastruttura, ma potrà essere il costo e la qualità del dispositivo stesso.

Alcuni fra i principali produttori di hardware hanno interpretato l'avvento di SDN come la rovina per il loro mercato, in quanto la loro forza stava proprio nel produrre hardware con software proprietari altamente competitivi rispetto alle altre aziende. Con l'avvento di SDN invece gli elementi di rete come switch e router dovrebbero diventare semplici apparati hardware a basso costo, in cui il marchio di fabbrica non dovrebbe fare differenza. In risposta a tutto ciò alcune aziende hanno modificato i loro software implementando di fatto soluzioni proprietarie la cui filosofia riprende il concetto SDN, altri produttori invece stanno acquisendo start-up e piccole aziende emergenti nel settore, in modo da far evolvere la loro azienda modificando il loro modello di business per restare competitivi nel settore.

Le aziende nate dal web come Facebook, Google, Amazon e Yahoo, oltre ad aver già implementato le tecnologie SDN e OpenFlow all'interno dei loro Data Center, fanno anche parte dell'organizzazione più attiva nella ricerca e

lo sviluppo di tecnologie per lo sviluppo della rete programmabile: la Open Network Foundation (ONF) [13].

Da parte degli sviluppatori di software e dei produttori di hardware generici c'è un grosso entusiasmo, in quanto per i primi c'è la possibilità, grazie all'introduzione di una nuova tecnologia nel settore, di poter lavorare molto implementando software innovativi, mentre per i secondi c'è la possibilità di produrre elementi hardware senza particolari esigenze funzionali, vendibili su larga scala.

In fine, vediamo come gli operatori di rete stiano ancora cercando di capire quale sarà lo standard da adottare e come si evolverà l'intero scenario a livello globale.

Controller Per quanto riguarda i dispositivi hardware su cui è possibile far girare il software del controllo per una rete SDN, troviamo alcuni produttori principali, come: Brocade, Cisco, HPE, Juniper, Nuage e NEC.[11] Sono inoltre presenti controller open source come OpenDaylight, OpenContrail e ONOS.
Per quanto riguarda il software invece troviamo i seguenti elementi nel mercato:

- **NOX:** sviluppato dalla Nicira Networks, è stato il primo controller SDN in grado di supportare il protocollo OpenFlow. Inizialmente NOX era partito con un interfaccia basata su OpenFlow scritta in linguaggio C++, mentre successivamente è stata rilasciata una versione più recente di NOX, ovvero POX, il quale a differenza del suo predecessore implementa l'interfaccia programmabile in python [12].

- **Beacon:** è stato creato dalla Stanford University nel 2010. A differenza di NOX esso è scritto in Java, la sua potenzialità risiede nella stabilità e nel fatto che è possibile aggiornare parti di codice senza dover interrompere necessariamente il programma in esecuzione.

- altri controller sono **Maestro** e **Floodlight**, scritti entrabi in Java, tra i meno popolari troviamo invece **Trema** scritto in C e Ruby dalla NEC.

Standard per D-CPI Per quanto riguarda le interfaccie tra controller e switch a livello industriale si è affermato il protocollo OpenFlow, sviluppato dall'Open Networking Foundation (ONF) [10]. Attualmente ci sono molti produttori di switch e router che hanno annunciato il loro supporto e la compatibilità dei loro apparecchi verso il protocollo OpenFlow. Tra i produttori di harware di rete compatibili col protocollo troviamo: Cisco, Juniper, Big Switch Networks, Brocade, Arista, Extreme Networks, IBM, Dell, NoviFlow, HP, NEC, e molti altri ancora.
Malgrado OpenFlow sia il protocollo più diffuso, non è il solo disponibile.
Il Network Configuration Protocol (`NetConf`) utilizza un Extensible Markup Language (XML) per comunicare con switch e router in modo da apportare modifiche e cambiamenti.[8]
Lisp, anch'esso promosso dalla ONF, è disponibile per il supporto al mappaggio dei flussi.[9]
Inoltre sono disponibili altri protocolli nello scenario SDN come OSPF, MPLS, BGP e IS-IS.

[8] approfondimenti su NetConf `https://en.wikipedia.org/wiki/NETCONF`

[9] approfondimenti su Lisp `https://wiki.opendaylight.org/view/OpenDaylight_Lisp_Flow_Mapping:Integration_Tests`

Capitolo 3

OpenFlow

3.1 Introduzione

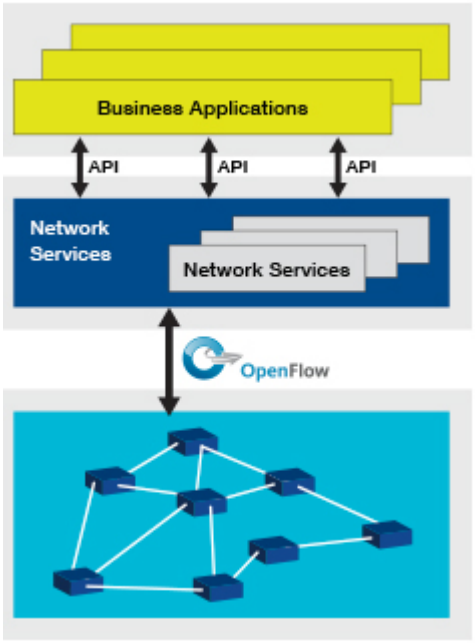
All'interno di questo capitolo troviamo una breve introduzione del protocollo OpenFlow. Nella prima parte viene descritto l'argomento in modo illustrativo, introducendo alcune note riguardo l'evoluzione temporale, mentre nella seconda parte troviamo una descrizione più tecnica e dettagliata.

3.1.1 Definizione

OpenFlow (OF) è il primo protocollo di comunicazione definito tra il livello di controllo e quello di inoltro in un architettura SDN. OpenFlow permette l'accesso diretto e la modifica del piano di inoltro di un dispositivo (switch o router), sia fisico che virtuale, attraverso la rete [16].

Le tecnologie SDN basate su OpenFlow permettono all'IT di supportare la banda larga e la natura dinamica delle applicazioni odierne, di adattare la rete ai bisogni del business in continuo cambiamento, e di ridurre in modo significativo la complessità di gestione e delle operazioni.

- **livello applicativo**
- **livello di controllo**
- **livello fisico**



Architettura logica OpenFlow [16]

3.1.2 Nascita e sviluppo

Le origini di OpenFlow risalgono al 2006 quando M. Casado, uno studente PhD dell'Università di Stanford, svilluppò Ethane, una nuova architettura di rete, la cui idea era quella di centralizzare la gestione delle funzionionalità di rete basandosi sul concetto di flusso. Quest'idea ha portato, grazie a successive ricerche da parte di un team all'interno della stessa università, alla definizione concettuale di OpenFlow così come la conosciamo oggi.

Nel 2011 è stata fondata l'Open Networking Foundation (ONF), un'associazione no profit, il cui scopo è quello di innovare la rete tramite le tecnologie SDN e standardizzare il protocollo OpenFlow con le relative tecnologie.

La prima versione che è stata standardizzata dall'ONF è la v1.0.0, rilasciata nel dicembre 2009, da quel momento in poi sono state rilasciate nuove versioni con cadenza circa annuale. Attualmente la versione più recente è la v1.5.1 approvata nel marzo 2015 [15].

3.2 Protocollo

In questa sezione vediamo quali sono le parti principali che compongono uno switch basato su OpenFlow e come si sviluppa il protocollo a livello pratico.

Uno Switch logico che supporta OpenFlow, in genere contiene una o più *tabelle di flusso* (flow table) e una *tabella di gruppo* (gropus table), tramite le quali realizza il controllo e l'inoltro dei pacchetti. Inoltre in ogni switch sono presenti uno o più canali di comunicazione OpenFLow, che collegano lo switch al controller (Figura 3.1 nella pagina successiva) [15].

Attraverso il protocollo OF, il controller può aggiungere, aggiornare, o eliminare le voci delle tabelle, in maniera reattiva (ossia in risposta ai pacchetti in arrivo), oppure proattiva. Ogni tabella presente nello switch è composta da varie voci di flusso, le quali sono composte a loro volta da più campi, contatori e varie istruzioni associate ad ogni corrispondenza dei pacchetti. I pacchetti vengono analizzati inizialmente cercando corrispondenze tra le voci della prima tabella, nella quale se è presente un'istruzione da associare al pacchetto, essa viene eseguita, altrimenti vengono eseguite delle operazioni di default basate sulla configurazione dello switch, e così via proseguendo con le tabelle successive.

3.2.1 Pipeline OpenFlow e Tabelle

Un pacchetto ricevuto da uno switch OpenFlow viene elaborato dalla pipeline OpenFlow. Sul pacchetto vengono eseguite una serie di operazioni di confronto tra i propri campi e quelli presenti nelle tabelle di flusso e gruppo dello switch, il cui scopo è trovare la serie di istruzioni che andranno poi eseguite per inoltrare correttamente il pacchetto. Le istruzioni che vengono collezionate durante il processamento, ad ogni corrispondenza trovata, vengono salvate all'interno del Action-Set.

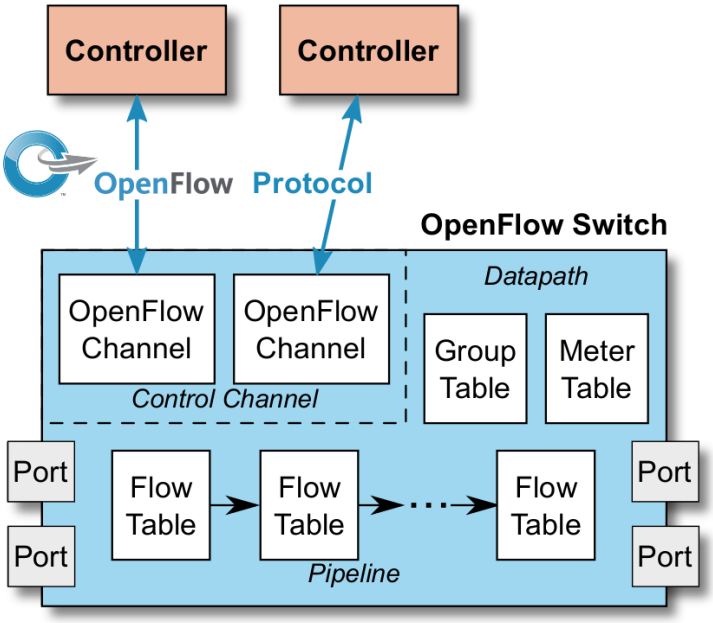


Figura 3.1: *Componenti principali di uno switch OpenFlow* [15]

.

Tabelle di Flusso

Principali componenti di una singola voce in una tabella di flusso:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

- `match fields:` sono i valori con cui i pacchetti vengono confrontati per cercare corrispondenze. Contengono la porta d'ingresso, l'intestazione del pacchetto, e altri campi opzionali;
- `priority:` precedenza nel caso di voci con più corrispondenze;
- `counter:` incrementato ogni qualvolta che viene riscontrata la corrispondenza con un pacchetto;
- `istructions:` istruzioni da aggiungere all'*Action-Set* o modifiche al flusso della pipeline;

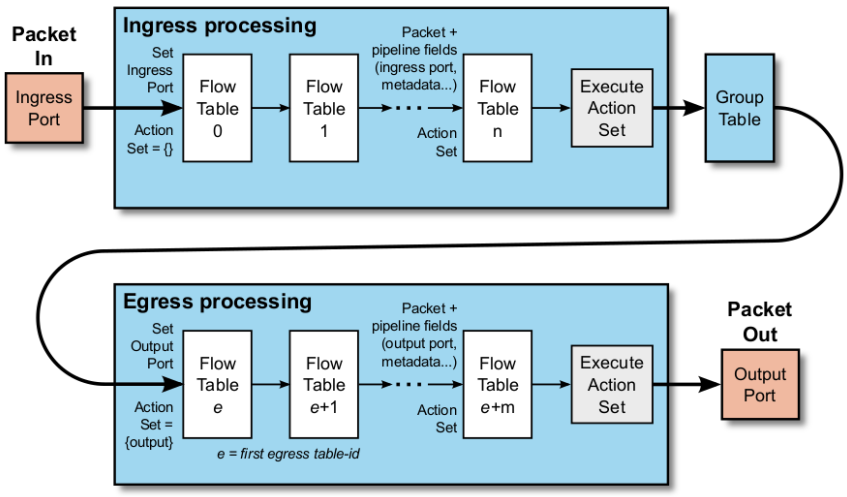


Figura 3.2: *Flusso di un pacchetto all'interno della pipeline OpenFlow* [15]

.

- `timeouts:` massimo tempo per cui lo switch tiene in memoria questa voce, altre il quale la voce viene scartata;
- `cookie:` valori utilizzati per statistiche riguardo la singola voce, quindi non vengono considerati nell'elaborazione dei pacchetti dati;
- `flags:` utilizzate per differenziare la gestione della voce;

Ogni voce presente in tabella è univocamente determinata dal campo *match fields* e *priority*; sono ammesse voci con lo stesso *match fields* a patto che abbiano priorità diverse.

Tabella di Gruppo

Diversamente dalla tabella di flusso, la tabella di flusso raggruppa due o più voci con caratteristiche analoghe alle quali applicare le stesse istruzioni. Ogni singola voce sarà quindi composta nel seguente modo:

Group Identifier	Group Type	Counters	Action Buckets

- `group identifier:` valore intero e univoco che contraddistingue un singolo gruppo;
- `group type:` necessario per eseguire gruppi di istruzioni dedicati al tipo di pacchetti rappresentati;
- `counters:` campo equivalente a quello presente nelle tabelle di flusso;
- `action buckets:` lista di insiemi di istruzioni da applicare;

3.2.2 Porte Switch OpenFlow

Le porte OpenFlow sono le interfacce di rete necessarie per il transito dei pacchetti a partire dalla loro elaborazione, verso il resto della rete. Per consentire l'elaborazione dei pacchetti gli switch OpenFlow allocano un determinato numero di porte logiche che può variare in base alle caratteristiche fisiche del dispositivo e alle mansioni alle quali esso deve adempiere. Ogni switch può essere collegato ad uno o più di altri switch tramite porte specifiche. I pacchetti vengono ricevuti tramite le porte di ingresso e processati secondo la pipeline OpenFlow, per poi essere inoltrati tramite le porte di uscita.
Ogni switch OF deve supportare tre tipologie di porte:[15]

- **porte fisiche:** porte che corrispondono a interfacce hardware appartenenti allo switch. A volte possono essere solamente virtualizzate, rappresentando solo parte dell'interfaccia fisica corrispondente;
- **porte logiche:** a differenza delle porte fisiche, quelle logiche non corrispondono direttamente ad un interfaccia fisica hardware, ma sono

un'astrazione delle stesse implementate in un livello superiore indipendente dal protocollo OF. Esse vengono rimappate in porte fisiche per permettere il loro funzionamento. L'unica differenza tra porte logiche e fisiche nel protocollo OF è che le porte logiche possono avere un capo extra nella pipeline chiamato *Tunnel_ID* associato ad essa, e quando un pacchetto è ricevuto da una porta logica, esso viene spedito al controller insieme alle informazioni che lo riconducono ad entrambe le porte, siano esse fisiche che logiche;

- **porte riservate:** definiscono azioni di inoltro generiche, come per esempio l'invio al controller o modalità di inoltro tramite metodi non OpenFlow;

  Tra le porte riservate, quelle necessarie sono le seguenti:

  - **ALL**: rappresenta tutte le porte che lo switch può usare per inoltrare messaggi, una copia del messaggio verrà inoltrata su tutte le porte ad escluzione delle porte OpenFlow in ingresso e di quelle del tipo `OFPPC_NO_FWD`.
  - **COTROLLER**: rappresenta il canale di comunicazione con il controller, essa può essere utilizzata sia come porta d'ingresso che come porta d'uscita. Nel primo caso i pacchetti vengono contrassegnati come ricevuti dal controller, mentre nel secondo caso è necessario incapsulare i messaggi inviati tramite protocollo OF.
  - **TABLE**: questa porta può essere utilizzata solo come uscite a permette di indirizzare un determinato messaggio verso il confronto con la prima tabella in modo da farlo processare correttamente.
  - **IN_PORT**: rappresenta la porta d'ingresso del pacchetto e permette di rispedire un pacchetto verso la sua sorgente, utilizzando la porta solo come uscita.
  - **ANY**: valore speciale, viene utilizzato nel caso in cui un'operazione può essere eseguita su una qualsiasi porta.

– **UNSET**: valore speciale utilizzato quando la porta d'uscita non è stata impostata nell'*Action-Set*.[1]

Ulteriori porte come `LOCAL`, `NORMAL` e `FLOOD` sono opzionali.

3.2.3 Canale di comunicazione e messaggi utilizzati

Il canale OpenFlow è l'interfaccia che connette ogni switch logico OF con il controller OF. Attraverso questa interfaccia il controller può configurare e gestire lo switch, in modo da inoltrare e ricevere pacchetti tra i due dispositivi. Il canale di controllo di uno switch può supportare uno o più canali di comunicazione OF con uno o più controller. Il canale di comunicazione è spesso crittografato secondo una tecnologia specifica chiamata TLS, [2] tuttavia può essere utilizzado direttamente il protocollo TCP.

Per quanto riguarda i messaggi scambiati, il protocollo OpenFlow supporta tre tipologie di messaggi:

- **controller-to-switch:** una comunicazione con questa tipologia di messaggi è sempre iniziata dal controller, ed è utilizzata per configurare o interrogare lo stato dello switch. Normalmente non sempre richiedono una risposta da parte dello switch. Spesso in questo modo il controller interroga lo switch riguardo le sue funzionalità di base, le specifiche tecniche e la modifica delle voci all'interno delle tabelle.

- **asincroni:** questi messaggi sono inviati dallo switch senza previa richiesta del controller. All'interno di questa tipologia troviamo i messag-

[1] Ad ogni pacchetto è associato un *Action-Set*, esso è costituito da una lista di istruzioni collezionate via via che il pacchetto viene processato attraverso le tabelle di flusso, una volta terminata l'analisi tutte le operazioni vengono eseguite in successione.

[2] Il protocollo TLS consente alle applicazioni client/server di comunicare attraverso una rete in modo tale da prevenire il "tampering" (manomissione) dei dati, la falsificazione e l'intercettazione. È un protocollo standard IETF che, nella sua ultima versione, è definito nella RFC 5246, sviluppata sulla base del precedente protocollo SSL da Netscape Communications.

gi inviati dallo switch all'arrivo di nuovi pacchetti non ancora indicizzati nelle tabelle, e messaggi comunicanti eventuali cambiamenti di stato o oltre notifiche di avvenuta modifica di particolari voci nelle tabelle, da parte dello switch. In questa categoria sono presenti: `Packet-in, Packet-out o Flow-mod, Flow-Removed, Port-status, Role-status, Controller-Status, Flow-monitor.`

- **simmetrici:** i messaggi di questo tipo sono bilaterali e vengono inoltrati senza sollecitazioni. Essi comprendono messaggi per instaurare una nuova connessione come per esempio `Hello` presente all'inizio di una connessione, oppure messaggi per verificare lo stato della connessione come per esempio `Echo`.

Capitolo 4

Sperimentazione

4.1 Ambienti di sviluppo SDN basati su OF

Per realizzare l'intera esperienza è stata utilizzata la macchina virtuale preconfigurata disponibile al seguente sito:
`http://sdnhub.org/tutorials/sdn-tutorial-vm/`
Questa VM ideata per permettere lo studio e lo sviluppo di tecnologie SDN, contiene il seguente materiale:

- 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 Openflow 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.

Mininet: è il software che permette la simulazione della rete fisica. Esso è in grado di simulare un kernel reale, degli switch e dei codici applicativi

[1] visitare `http://openvswitch.org/` per maggiori informazioni.

all'interno della stessa macchina virtuale. Mininet è molto veloce e permette di creare topologie di rete anche molto complesse tramite l'uso di semplici comandi. Si può interagire con gli elementi virtuali creati tramite CLI (e API). Mininet inoltre supporta nativamente OpenFlow. I motivi elencati rendono Mininet un ottimo strumento usato nella didattica, ricerca e sviluppo di sistemi basati sul Software Defined Networking [17]. Il software è attivamente sviluppato e rilasciato sotto licenza BSD.[2]
Il sito officiale è presente al link: `http://mininet.org/`
Un breve tutorial in cui si può comprenedere meglio l'ambiente e conoscere tutti i comandi a disposizione è invece disponibile al seguente sito:
`http://mininet.org/walkthrough/#interact-with-hosts-and-switches`

Miniedit: è un estensione di Mininet che permette la creazione e la configurazione di topologie di rete tramite l'uso di un ambiente grafico.
E' possibile trovare una breve guida sull'utilizzo del software al seguente link:
`http://www.brianlinkletter.com/how-to-use-miniedit-mininets-graphical-user-interface/`

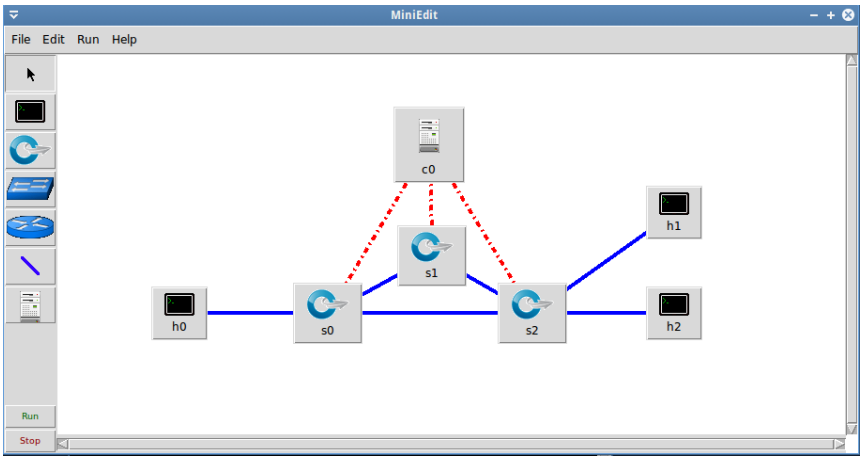


Figura 4.1: *Ambiente grafico di Miniedit.*

[2]Le licenze BSD sono una famiglia di licenze permissive, senza copyleft, per software.
`https://it.wikipedia.org/wiki/Licenze_BSD`

POX: è il controller che andiamo ad utilizzare. Il software, scritto in *Python*, è basato su SDN e supporta OpenFlow. POX è nato come controller OpenFlow, ma ora può essere anche utilizzato come semplice switch OF. Il software funziona in ambienti Linux, Mac OS, e Windows, e utilizza la versione di *Python* 2.7.
Il programma viene lanciato tramite il software `pox.py` il quale analizza i parametri passati e procede all'inizializzazione del controller. Inizialmente tramite `pox.py` vengono caricati tutti i moduli presenti all'interno della cartella d'installazione e il codice presente nelle sotto cartelle pox ed ext, all'interno delle quali si può caricare il software sviluppato. Per testare il controller è possibile utilizzare il seguente comando:

```
./pox.py my_controller --address=127.0.0.1 --port=6633
```

`--address:` consente di impostare l'indirizzo al quale sarà raggiungibile il controller, al quale dovranno collegarsi gli switch;

`--port:` porta sulla quale il controller resta in ascolto di connessioni da parte degli switch in entrata;

Al seguente link è presente una guida dettagliata riguardo tutte le funzionalità offerte.
`https://openflow.stanford.edu/display/ONL/POX+Wiki`.
Mentre il repository ufficiale si trova al link `https://github.com/noxrepo/pox`, dove si possono trovare i codici sorgenti e molti programmi dimostrativi.

4.2 Introduzione

L'esperienza di seguito descritta ha l'intento di mostrare alcuni fra i principali vantaggi che l'introduzione di un approccio SDN può comportare su una rete tradizionale. Per farlo viene mostrato come l'introduzione di un semplice controller in grado di gestire in maniera dinamica un determinato tipo di flusso possa portare grossi vantaggi anche in uno scenario apparentemente banale.

4.2.1 Struttura della rete

Dal punto di vista topologico la rete su cui è stata svolta l'esperienza si presenta come in Figura 4.2. Dalla figura è possibile vedere che sono presenti 2 host, 3 switch OpenFlow e un server in serie ad un NAT.[3]

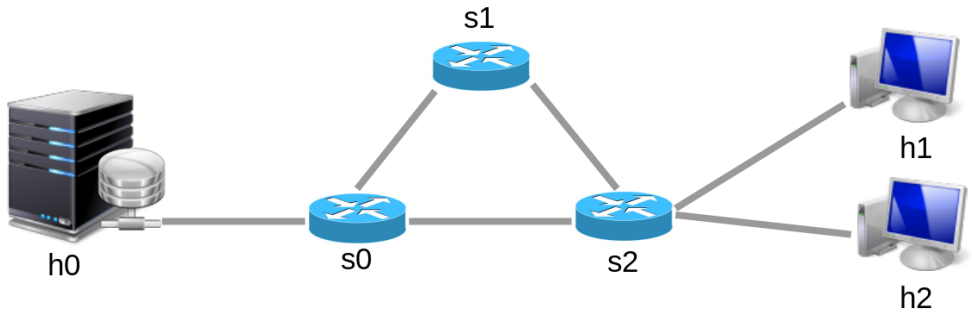


Figura 4.2: *Topologia della rete*

I vari elementi sono connessi tramite link. Ogni link è stato configurato in modo da introdurre un certo *delay* e offrire una *bandwidth* limitata; le caratteristiche di ogni connessione sono riportate nella Tab. 4.1.

[3]Nel campo delle reti telematiche, il network address translation o NAT, ovvero traduzione degli indirizzi di rete, conosciuto anche come network masquerading, native address translation, è una tecnica che consiste nel modificare gli indirizzi IP dei pacchetti in transito su un sistema che agisce da router all'interno di una comunicazione tra due o più host.

Link	bitrate	loss	ritardo
h0-s0	B=10Mb/s	L=0%	d=50ms
s0-s2	B=2Mb/s	L=0%	d=20ms
s0-s1	B=1Mb/s	L=0%	d=10ms
s1-s2	B=1Mb/s	L=0%	d=10ms
s2-h1	B=5Mb/s	L=0%	d=20ms
s2-h2	B=5Mb/s	L=0%	d=20ms

Tabella 4.1: Parametri di configurazione dei link.

4.2.2 Scenario

Lo scenario considerato per la dimostrazione è il seguente: due utenti tramite i loro device [h1, h2] intendono vedere un film presente nel sito www.film.it ospitato dal server h0; tuttavia l'utente su h1 dopo aver iniziato la visione del film vuole scaricare un file importante nel minor tempo possibile, ma senza compromettere la qualità del flusso video. Il file è presente nel medesimo server da cui entrambi gli utenti stanno prelevando il contenuto video e pertanto, deve transitare in uno dei due link possibili.
Una premessa doverosa per la comprensione dell'esperienza è che lo streaming video in questo caso è un flusso di pacchetti TCP mentre il file viene scaricato tramite un flusso UDP.
Dal punto di vista della rete il traffico può essere riassunto nel seguente modo:

- il flusso TCP1 dal server `h0` all'host `h1`
- il flusso TCP2 dal server `h0` all'host `h2`
- il flusso UDP dal server `h0` all'host `h1`

I due flussi TCP non hanno quindi priorità particolari tuttavia è bene offrire loro la larghezza di banda massima disponibile.

Il flusso UDP invece ha la massima priorità, trasporta in totale 1.95 Mbytes e dev'essere inoltrato nel minor tempo possibile.

4.3 Soluzione

Per svolgere l'esperienza è stata simulata una rete virtuale basata sulla topologia in Figura 4.2 utilizzando l'ambiente Mininet.
Per far si che i dati transitino nella giusta direzione all'interno della rete è stato poi necessario programmare il controller in modo da istruire gli switch correttamente. Come simulatore del controller è stato utilizzato POX programmato in python.
Nella prima fase il controller viene implementato in modo da simulare il comportamento di una rete tradizionale, nella seconda fase invece è stato simulato il comportamento di una rete SDN.

Per la comprensione del codice di seguito riportato è necessario apprendere la disposizione delle porte tramite le quali ogni switch interagisce nella rete, a tal proposito è riporta la Figura 4.3.

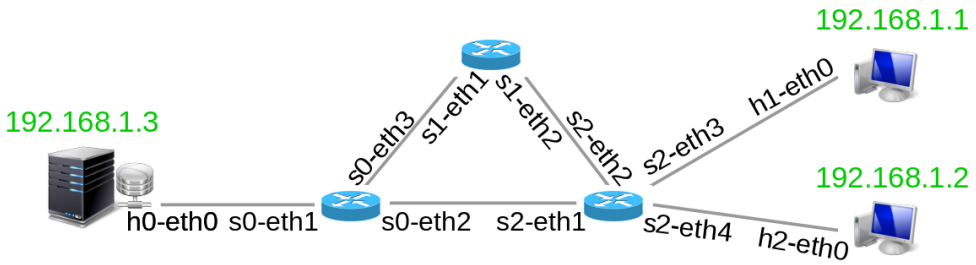


Figura 4.3: *Interfacce e indirizzi utilizzati.*

4.3.1 Rete tradizionale

La caratteristica principale che si vuole mettere in luce è che nelle soluzioni tradizionali le regole di instradamento all'interno degli switch vengono inserite un'unica volta in fase di configurazione, con i problemi che ne derivano

Per simulare tale comportamento è stato programmato il controller in modo tale da inserire tutte le regole necessarie all'interno di uno switch appena tale dispositivo si connette.

Struttura principale del codice

Ogni volta che uno switch manda un messaggio al controller, viene generato un oggetto `event` il quale contiene tutte le informazioni necessarie.
L'oggetto `event` deve essere catturato da funzioni chiamate rispettivamente `_handle_''funzionalità''`, le quali saranno in grado di esaudire la richiesta dello switch. Queste funzioni devono essere tuttavia abilitate alla ricezione dell'oggetto `event` tramite la funzione `launch()`: righe [14-17].

```
1  #!/usr/bin/python
   from pox.core import core
3  import pox.openflow.libopenflow_01 as of
   from pox.lib.util import dpidToStr
5
   log = core.getLogger()
7
   def _handle_ConnectionUp (event):
9    [...]

11 def _handle_PacketIn (event):
     [...]
13
   def launch ():
15   core.openflow.addListenerByName("ConnectionUp",
       _handle_ConnectionUp)
     core.openflow.addListenerByName("PacketIn",_handle_PacketIn)
```

_handle_ConnectionUp

Ogni switch è identificato da un numero `dpid` univoco. Tramite questa fun-

zione viene memorizzato il `dpid` dello switch, in una stringa di più facile comprensione, durante la prima connessione di ogni switch.

```
s0_dpid=0
s1_dpid=0
s2_dpid=0

def _handle_ConnectionUp (event):
  global s0_dpid, s1_dpid, s2_dpid

  for m in event.connection.features.ports:
    if m.name == "s0-eth1":
      s0_dpid = event.connection.dpid
    elif m.name == "s1-eth1":
      s1_dpid = event.connection.dpid
  ...
```

_handle_PacketIn

Questa funzione viene chiamata ogni qual'volta uno switch invia un pacchetto la cui struttura non è stata trovata all'interno delle tabelle di inoltro.[4]

[4] Tramite l'istruzione `event.connection.dpid==s0_dpid` si verifica lo switch da cui è arrivata la richiesta.

[6-12] Per inserire una regola nello switch viene prima creato un oggetto di tipo `of.ofp_flow_mod`, si impostano i parametri desiderati e la si inoltra allo switch con l'istruzione `event.connection.send(msg)`.

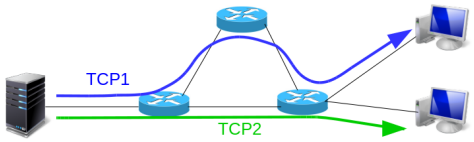
```
def _handle_PacketIn (event):
  global s0_dpid, s1_dpid, s2_dpid
  print "PacketIn: ", dpidToStr(event.connection.dpid)
  if event.connection.dpid==s0_dpid:

    msg = of.ofp_flow_mod()
    msg.priority = 100          #priorita'
    msg.match.in_port = 1       #porta d'ingresso nello switch
    msg.match.dl_type = 0x806   #ARP reques
    #porta su cui inoltrare il pacchetto
    msg.actions.append(of.ofp_action_output(port = 3))
    event.connection.send(msg)
    ...
    msg = of.ofp_flow_mod()
```

[4] Esiste il caso in cui i pacchetti di un determinato tipo vengono per regola inoltrati direttamente al controller.

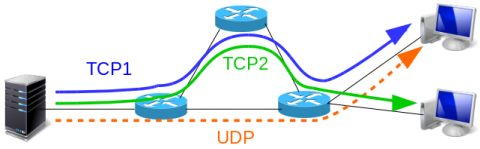
```
15      msg.match.dl_type = 0x800  #Livello 3
        msg.match.nw_dst = "192.168.1.3"
17      msg.match.nw_proto = 1     #ICMP request
        msg.actions.append(of.ofp_action_output(port = 1))
19      event.connection.send(msg)

21      msg = of.ofp_flow_mod()
        msg.match.dl_type = 0x800  #Livello 3
23      msg.match.nw_dst = "192.168.1.2"
        msg.match.nw_proto = 1     #ICMP request
25      msg.actions.append(of.ofp_action_output(port = 3))
        event.connection.send(msg)
27      ...
        msg = of.ofp_flow_mod()
29      msg.match.dl_type = 0x800  #Livello 3
        msg.match.nw_dst = "192.168.1.1"
31      msg.match.nw_proto = 17    #UDP
        msg.actions.append(of.ofp_action_output(port = 2))
33      event.connection.send(msg)

35      msg = of.ofp_flow_mod()
        msg.match.dl_type = 0x800 #Livello 3
37      msg.match.nw_dst = "192.168.1.1"
        msg.match.nw_proto = 6     #TCP
39      msg.actions.append(of.ofp_action_output(port = 3))
        event.connection.send(msg)
41
        msg = of.ofp_flow_mod()
43      msg.match.dl_type = 0x800 #Livello 3
        msg.match.nw_dst = "192.168.1.2"
45      msg.match.nw_proto = 6     #TCP
        msg.actions.append(of.ofp_action_output(port = 2))
47      event.connection.send(msg)

49    elif event.connection.dpid==s1_dpid:

51      [... "semplice inoltro dei pacchetti dalla porta 1 alla
       porta 2 e viceversa" ]

53    elif event.connection.dpid==s2_dpid:

55      [... "i pacchetti di tipo IP vengono inoltrati in base all'
       indirizzo di destinazione mentre i pacchetti di tipo ARP
       vengono inoltrati in broadcast" ]
```

4.3.2 Rete SDN

In questa parte dell'esperienza si possono vedere alcune fra le molte potenzialità del controller.
L'obiettivo è quello di spedire il pacchetto UDP nel minor tempo possibile, senza compromettere lo streaming TCP dei due utenti. Il controller POX è stato quindi programmato in modo tale da:



Far confluire il flusso TCP1 nella rotta superiore mentre il flusso TCP2 in quella inferiore, fintantochè il flusso UDP non è presente.



Instradare entrambi i flussi TCP nella rotta superiore in modo da liberare il canale inferiore per il transito del flusso UDP.

Il codice utilizzato presenta la stessa struttura di base illustrata nella soluzione tradizionale.
Ciò che rende la rete dinamica è la funzione `_handle_PacketIn` che in questo caso libera dinamicamente il canale tra s0 e s2 dirottando i flussi TCP verso lo switch s1 all'arrivo di pacchetti UDP. In questa sezione di codice sono state introdotte alcune operazioni aggiuntive riguardo l'analisi dei pacchetti. Come possiamo vedere all'interno del programma ogni pacchetto viene elaborato in modo differente secondo il tipo di appartenenza.
[5-8] Tramite `packet = event.parsed` viene copiato il messaggio arrivato allo switch, privato dell'involucro creato dal protocollo OpenFlow. In questo modo il pacchetto può essere elaborato ricavando diverse informazioni, mentre se il pacchetto contiene qualche tipo di errore viene scartato.
Tramite la riga `packet_in = event.ofp` il messaggio di tipo OpenFlow che arriva al controller viene interamente copiato nell'oggetto `packet_in` [15-28] In queste righe viene controllata la tipologia del pacchetto, se è di tipo ARP, viene creato un nuovo messaggio di tipo OF [19]. Vengono poi copiati i dati

del messaggio originale all'interno del nuovo messaggio OF [20], e spedito il nuovo messaggio allo switch. [28] Il nuovo messaggio contiene l'istruzione di inoltro dei pacchetti provenienti dalla porta 1 sulla porta 3 altrimenti viceversa.
[39-57] Di default se arriva un pacchetto TCP allo switch, il controller installa le rotte in modo da smistare i flussi TCP come rappresentato nella prima immagine.
[59-94] Non appena arriva un pacchetto UDP vengono sovrascritte le regole precedenti introducendo le nuove, le quali scadono ogni 2 secondi [es 79].

```
def _handle_PacketIn (event):
  global s0_dpid, s1_dpid, s2_dpid
  global flusso_UDP = 0

  packet = event.parsed # This is the parsed packet data.
  if not packet.parsed:
    print "Ignoring incomplete packet"
    return

  packet_in = event.ofp # The actual ofp_packet_in message.

  ipv4_packet = packet.find('ipv4')
  icmp_packet = packet.find('icmp')

  if event.connection.dpid==s0_dpid:

    if packet.type == packet.ARP_TYPE:  #analisi pacchetti ARP
      print "arp packet received nello switch s0"
      msg = of.ofp_packet_out()
      msg.data = packet_in
      if msg.in_port == 1:
        action = of.ofp_action_output(port = 3)

      elif msg.in_port == 3:
        action = of.ofp_action_output(port = 1)

      msg.actions.append(action)
      event.connection.send(msg)


    elif packet.type == packet.IP_TYPE:
      ip_packet = packet.payload  #tolto involucro OF

      if ip_packet.protocol == 1: #messaggio ICMP
       [ ... codice analogo a quello per pacchetto ARP ]

      elif ip_packet.protocol == 6:
```

```
39          msg = of.ofp_flow_mod()      #TCP
            msg.priority = 1000
41          msg.match.dl_type = 0x800 #Livello 3
            msg.match.nw_dst = "192.168.1.1"
43          msg.match.nw_proto = 6 #TCP
            msg.idle_timeout = 1
45          msg.actions.append(of.ofp_action_output(port = 3))
            event.connection.send(msg)
47
            msg = of.ofp_flow_mod()      #TCP
49          msg.priority = 1000
            msg.match.dl_type = 0x800 #Livello 3
51          msg.match.nw_dst = "192.168.1.2"
            msg.match.nw_proto = 6 #TCP
53          msg.idle_timeout = 1
            msg.actions.append(of.ofp_action_output(port = 2))
55          event.connection.send(msg)

57          [...]

59        elif ip_packet.protocol == 17: #messaggio UDP
            print "pacchetto UDP arrivato"
61          flusso_UDP = 1 #indica che e' presente un flusso UDP

63          msg = of.ofp_flow_mod()      #UDP
            msg.priority = 1000
65          msg.match.dl_type = 0x800 #Livello 3
            msg.match.nw_dst = "192.168.1.1"
67          msg.match.nw_proto = 17 #UDP
            msg.idle_timeout = 1
69          msg.hard_timeout = 2
            msg.actions.append(of.ofp_action_output(port = 2))
71          event.connection.send(msg)

73          msg = of.ofp_flow_mod()      #TCP
            msg.priority = 1000
75          msg.match.dl_type = 0x800 #Livello 3
            msg.match.nw_dst = "192.168.1.1"
77          msg.match.nw_proto = 6 #TCP
            msg.idle_timeout = 1
79          msg.hard_timeout = 2
            msg.actions.append(of.ofp_action_output(port = 3))
81          event.connection.send(msg)

83          msg = of.ofp_flow_mod()      #TCP
            msg.priority = 1000
85          msg.match.dl_type = 0x800 #Livello 3
            msg.match.nw_dst = "192.168.1.2"
87          msg.match.nw_proto = 6 #TCP
            msg.idle_timeout = 1
89          msg.hard_timeout = 2
            msg.actions.append(of.ofp_action_output(port = 3))
91          event.connection.send(msg)
```

```
93          [...]

95    elif event.connection.dpid==s1_dpid:
        [... "inoltro semplice dei pacchetti"]
97
      elif event.connection.dpid==s2_dpid:
99      [... "analogo allo switch s0"]
```

4.3.3 Test

Per testare le due soluzioni proposte è stato utilizzato il codice seguente:

9-10 creazione della topologia di rete;

21-25 avvio della simulazione tramite Mininet;

26-29 assegnazione indirizzi IP agli host;

32-33 esecuzione di un test di connettività;

36-44 impostazione host in modo da poter eseguire il test tramite *-iperf*;

40-41 qui il server invia 5Mbytes tramite TCP verso gli host h1 e h2;

42-43 dopo 4 secondi viene inviato un file (UDP) dal server all'host h1 di 1.95Mbytes;

```
class POXcontroller1( Controller):
2   def start(self):
      self.pox='%s/pox/pox.py ' %os.environ['HOME']
4     self.cmd(self.pox, "project_controller &")
    def stop(self):
6     self.cmd('kill %'+self.pox)
controllers = { 'poxcontroller1': POXcontroller1}
8
class MyTopo(Topo):
10  def __init__(self, n=2,**opts):
      Topo.__init__(self, **opts)
12    s0 = self.addSwitch('s0')
      s1 = self.addSwitch('s1')
14    s2 = self.addSwitch('s2')
      h0=self.addHost('h0', cpu=.5/n)
16    h1=self.addHost('h1', cpu=.5/n)
```

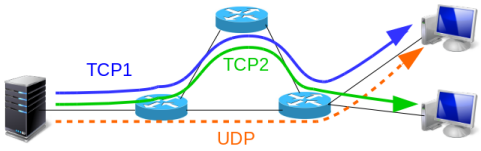
```
    h2=self.addHost('h2', cpu=.5/n)
    self.addLink(h0, s0, bw=10, delay='50ms',loss=0)
    self.addLink(s0, s2, bw=2,  delay='20ms',loss=0)
    [...]

def perfTest():
    topo = MyTopo(n=3)
    net = Mininet(topo=topo, host=CPULimitedHost, link=TCLink,
    controller=POXcontroller1)
    net.start()
    h0, h1, h2 = net.get('h0', 'h1', 'h2')
    h0.setIP( '192.168.1.3/24' )
    h1.setIP( '192.168.1.1/24' )
    h2.setIP( '192.168.1.2/24' )

    #Testa la connettivita' della rete
    dumpNodeConnections(net.hosts)
    net.pingAll()

    print "Starting simulation"
    h1.cmd('iperf -s -u -i 2 > ./udp_traffic0.txt &')
    h1.cmd('iperf -s -i 2    > ./tcp_traffic1.txt &')
    h2.cmd('iperf -s -i 2    > ./tcp_traffic2.txt &')

    print h0.cmd('iperf -c 192.168.1.1 -n 6000K &')
    print h0.cmd('iperf -c 192.168.1.2 -n 6000K &')
    time.sleep(4)
    print h0.cmd('iperf -c 192.168.1.1 -u -n 2000K ')
    [...]
    CLI(net)
    net.stop()

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

4.4 Risultati

Dalle varie simulazioni viene ricavato il tempo che i 3 flussi impiegano e la bandwidth media sfruttata.

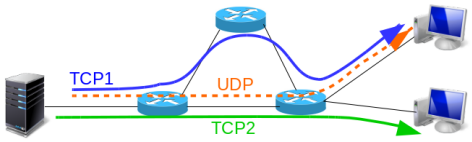
Rete tradizionale

La simulazione è stata eseguita sulla rete provando la combinazione di varie regole di instradamento, di seguito troviamo i risultati delle due più significative:



	tempo	bandwidth
TCP1	84.8s	B=581Kb/s
TCP2	103.1s	B=478Kb/s
UDP	15.6s	B=1.05Mb/s

In questa soluzione si può vedere come il file UDP venga scaricato in soli 15.6 secondi, tuttavia sono molto penalizzati i flussi TCP i quali ottengono rispettivamente una banda di 0.5Mb/s l'uno, causando tempi di scaricamento nell'ordine dei 90s.



	tempo	bandwidth
TCP1	51.8s	B=952Kb/s
TCP2	34.5s	B=1.43Mb/s
UDP	19.6s	B=835Kb/s

Questa soluzione presenta un miglior sfruttamento dei canali, offrendo una banda media maggiore ad entrambi i flussi TCP. Viene tuttavia penalizzato il flusso UDP, che nel transito del canale deve contendersi la banda con il flusso TCP2, causando un ritardo di circa 4s in più rispetto alla soluzione precedente.

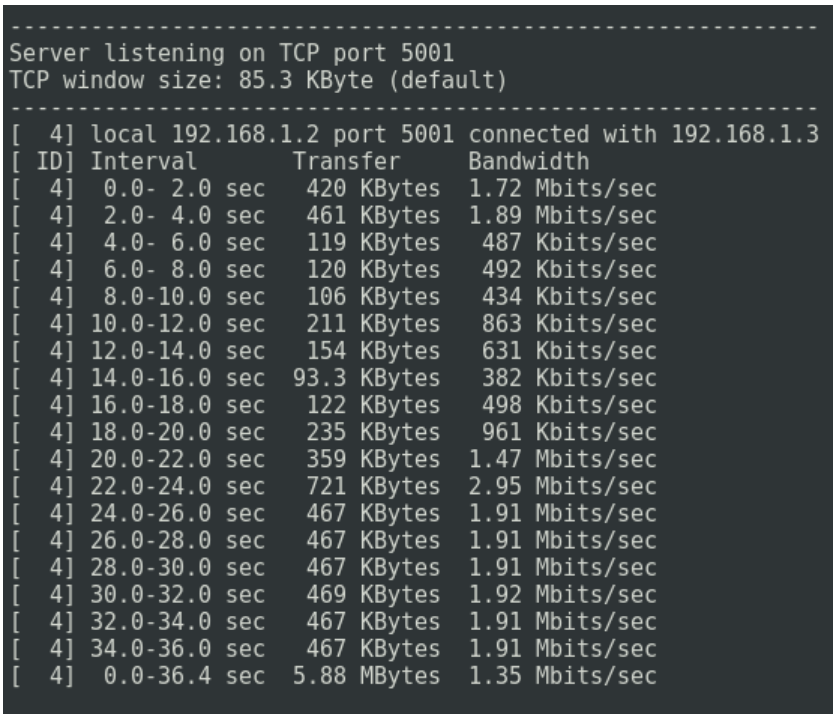
Rete SDN

Tramite l'introduzione di regole dinamiche invece sono stati ottenuti i seguenti risultati:

	tempo	**bandwidth**
TCP1	62.2s	B=792Kb/s
TCP2	36.04s	B=1.35Mb/s
UDP	15.4s	B=1.03Mb/s

Il file UDP è stato scaricato nel minor tempo possibile, mantenendo allo stesso tempo alte le prestazioni per entrambi i flussi TCP.

In quest'immagine possiamo vedere come il flusso TCP2 venga rallentato per 16s ovvero dal secondo 4 al 20 durante lo scaricamento del flusso UDP.



```
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.1.2 port 5001 connected with 192.168.1.3
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 2.0 sec   420 KBytes  1.72 Mbits/sec
[  4]  2.0- 4.0 sec   461 KBytes  1.89 Mbits/sec
[  4]  4.0- 6.0 sec   119 KBytes   487 Kbits/sec
[  4]  6.0- 8.0 sec   120 KBytes   492 Kbits/sec
[  4]  8.0-10.0 sec   106 KBytes   434 Kbits/sec
[  4] 10.0-12.0 sec   211 KBytes   863 Kbits/sec
[  4] 12.0-14.0 sec   154 KBytes   631 Kbits/sec
[  4] 14.0-16.0 sec  93.3 KBytes   382 Kbits/sec
[  4] 16.0-18.0 sec   122 KBytes   498 Kbits/sec
[  4] 18.0-20.0 sec   235 KBytes   961 Kbits/sec
[  4] 20.0-22.0 sec   359 KBytes  1.47 Mbits/sec
[  4] 22.0-24.0 sec   721 KBytes  2.95 Mbits/sec
[  4] 24.0-26.0 sec   467 KBytes  1.91 Mbits/sec
[  4] 26.0-28.0 sec   467 KBytes  1.91 Mbits/sec
[  4] 28.0-30.0 sec   467 KBytes  1.91 Mbits/sec
[  4] 30.0-32.0 sec   469 KBytes  1.92 Mbits/sec
[  4] 32.0-34.0 sec   467 KBytes  1.91 Mbits/sec
[  4] 34.0-36.0 sec   467 KBytes  1.91 Mbits/sec
[  4]  0.0-36.4 sec  5.88 MBytes  1.35 Mbits/sec
```

Di seguito è possibile vedere invece l'adamento temporale del flusso UDP, e vedere come l'output restituito ad ogni simulazione eseguita viene presentato.

```
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.1 port 5001 connected with 192.168.1.3 port 41217
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 2.0 sec   268 KBytes  1.10 Mbits/sec   1.026 ms    9/  196 (4.6%)
[  3]  2.0- 4.0 sec   251 KBytes  1.03 Mbits/sec   0.779 ms    2/  177 (1.1%)
[  3]  4.0- 6.0 sec   238 KBytes   976 Kbits/sec   6.014 ms    7/  173 (4%)
[  3]  6.0- 8.0 sec   256 KBytes  1.05 Mbits/sec   1.844 ms    7/  185 (3.8%)
[  3]  8.0-10.0 sec   245 KBytes  1.01 Mbits/sec   0.666 ms    8/  179 (4.5%)
[  3] 10.0-12.0 sec   223 KBytes   911 Kbits/sec   7.787 ms    7/  162 (4.3%)
[  3] 12.0-14.0 sec   254 KBytes  1.04 Mbits/sec   6.771 ms    7/  184 (3.8%)
[  3]  0.0-15.4 sec  1.89 MBytes  1.03 Mbits/sec   1.303 ms   49/ 1394 (3.5%)
```

Con questa breve esperienza si può vedere come la semplice introduzione di una regola dinamica possa migliorare notevolmente le prestazioni all'interno di uno scenario anche banale. L'applicabilità di questo approccio è resa possibile solo grazie all'introduzione del controller, il quale 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.

Bibliografia

[1] A. Manzolini, V. Vercellone, M. Ullio. *Notiziariotecnico*. Telecom Italia, 1/2013.

[2] P. Fasano, M. Ullio, V. Vercellone. *Notiziariotecnico*. Telecom Italia, 1/2016.

[3] `www.ericsson.com/traffic-market-report` Agosto 2012.

[4] M. Jammal, T. Sinch, A. Shami, R. Asal, Y. Li "*Software defined Networking: State of the art and research challenges*", (`http://dx.doi.org/10.1016/j.comnet.2014.07.004`)

[5] A. Gokhale, P. Berthou, D. C. Schmidt, T. Gayraud "*Software-Defined Networking: Challenges and research opportunities for the Future Internet*", (`http://dx.doi.org/10.1016/j.comnet.2014.10.015`)

[6] IEEE Software Defined Network (`http://sdn.ieee.org/`) [consultato il 02-ott-2016].

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

[8] ONF Northbound Interfaces working group, NB-API-Charter-v1_1 (`https://www.opennetworking.org/images/stories/downloads/working-groups/charter-nbi.pdf`) [giu-2013].

[9] ONF Intent NBI -Definition and Principles (`https://www.opennetworking.org/images/stories/downloads/sdn-resources/`

technical-reports/TR-523_Intent_Definition_Principles.pdf), [ott-2016].

[10] https://www.sdxcentral.com/sdn/definitions/southbound-interface-api/.

[11] https://www.sdxcentral.com/sdn/definitions/sdn-controllers/ [consultato il 15-ott-2016].

[12] M. McBride: SDN Security Considerations in the Data Center, Open Networking Foundation, 2013.

[13] ONF https://www.opennetworking.org/.

[14] Kate Green feb 2009 (http://www2.technologyreview.com/news/412194/tr10-software-defined-networking/) [consultato il 21-ott-2016].

[15] Specifiche tecniche di uno Switch basato su OpenFlow v1.5.1 (https://www.opennetworking.org/sdn-resources/technical-library).

[16] https://www.opennetworking.org/sdn-resources/openflow [consultato il 10-ott-2016].

[17] Documentazione Mininet (https://github.com/mininet/mininet/wiki/Documentation) [consultato il 18-ott-2016].

[18] POX wiki (https://openflow.stanford.edu/display/ONL/POX+Wiki) [consultato il 20-ott-2016].

[19] NAT https://it.wikipedia.org/wiki/Network_address_translation [consultato il 1-nov-2016].