Alma Mater Studiorum · Università di Bologna

SCUOLA DI INGEGNERIA E ARCHITETTURA - SEDE CESENA
CORSO DI LAUREA IN INGEGNERIA ELETTRONICA INFORMATICA E DELLE TELECOMUNICAZIONI

SOFTWARE DEFINED NETWORKS: ANALISI DELL'INTERAZIONE FRA NODI DI RETE E CONTROLLER

Relatore:
Chiar.mo Prof.
FRANCO CALLEGATI

Presentata da:
DANIELE SCHIAVI

Prima sessione
Anno Accademico 2016-2017

Indice

Introduzione

Questa tesi ha come tema principale le Software Defined Networking (SDN) ed il protocollo OpenFlow; questi rappresentano una concezione delle reti emergente che cerca di sostituire le architetture di rete tradizionali che hanno notevoli difficoltà a rimanere competitive dato l'enorme aumento dei dispositivi e del traffico scambiato. L'obiettivo è quello di capire le dinamiche di funzionamento dell'architettura SDN e del protocollo OpenFlow, in particolare le interazioni tra gli switch ed i controller in un ambiente multi-controller e la gestione dei ruoli degli stessi. Il primo capitolo parla dell'architettura SDN, introducendo inizialmente le cause che hanno portato alla scoperta di questo nuovo metodo di concepire le reti poi passando alla spiegazione di cosa si tratta ed infine vengono illustrati i componenti base dell'architettura ed i benefici che comporta l'utilizzo della stessa. Il secondo capitolo ha come tema il protocollo OpenFlow, anche in questo caso si è pensato di iniziare con un'introduzione per definire da dove è nato e quali sono state le cause che hanno portato alla sua definizione; in seguito vengono spiegati i concetti ed i componenti fondamentali che costituiscono tale protocollo concludendo tale capitolo con l'architettura multi-controller. I primi due capitoli sono puramente teorici e contengono i concetti base necessari prima di poter procedere ad uno studio dal punto di vista pratico. Il terzo capitolo descrive la tipologia di rete che è stata creata appositamente per effettuare i test per approfondire lo studio delle interazioni tra controller e switch; in particolare vengono presentati brevemente gli strumenti e le tecnologie utilizzate, poi viene descritta la rete ed i test effettuati con lo scopo di capire il funzionamento del protocollo in tali situazioni.

Capitolo 1

Software Defined Networking (SDN)

1.1 Verso una nuova architettura

Negli ultimi anni vi è stato un notevole aumento del numero dei dispositivi mobili connessi in rete, dei contenuti e dei servizi cloud sia pubblici che privati richiesti da utenti ed aziende. Oltre al numero di dispositivi è cambiato anche la dimensione dei dati richiesti e il metodo di utilizzo degli stessi; infatti, mentre fino a qualche tempo fa le applicazioni erano principalmente basate su una tipologia client-server, ora le richieste sono più complesse e possono richiedere l'accesso a più server o database prima di essere soddisfatte. Inoltre gli utenti cercano un modo di accedere alle reti aziendali tramite i dispositivi mobili personali come tablet, smartphone o computer portatili, portando un ulteriore problema dal punto di vista della sicurezza e della protezione dei dati. Le reti di oggi però non sono state progettate per soddisfare queste richieste, infatti presentano svariate limitazioni che vincolano i progettisti di reti; alcune di queste sono le seguenti:

- Fattore economico. Con l'aumento dei dispositivi mobili si ha un aumento del costo delle apparecchiature, per questo si cerca di sfruttare al meglio le reti aziendali utilizzando strumenti di gestione a livello di dispositivo e processi manuali.

- Dipendenza dal venditore. La mancanza di interfacce standard e aperte e i cicli di produzione delle apparecchiature limitano la capacità dei programmatori di adattare la rete ai singoli ambienti, rallentando l'implementazione di nuove funzionalità e servizi utili a soddisfare le esigenze aziendali o degli utenti.

- Complessità. Le reti attuali sono composte in gran parte da insiemi di protocolli utilizzati per connettere gli host ed avere prestazioni ed affidabilità elevate. Negli ultimi anni le esigenze aziendali e tecniche sono aumentate ed è stato necessario aggiungere nuovi protocolli di rete realizzati per risolvere problemi specifici senza fornire alcuna astrazione. Questi protocolli isolati che sono utilizzati per mantenere elevate affidabilità, prestazioni, connettività e sicurezza, hanno portato un notevole aumentato della complessità e di conseguenza ha reso le reti più statiche.

- Scalabilità. La rete tradizionale deve crescere rapidamente per soddisfare le richieste in rapido aumento dei centri di dati. Come detto precedentemente l'aggiunta di molti dispositivi rende la rete molto più complessa e di conseguenza aumenta la difficoltà per la rete di scalare, dovendo configurare e gestire manualmente i dispositivi.

Queste problematiche hanno reso necessaria la creazione di una nuova architettura di rete: Software Defined Networking e gli standard associati.

1.2 Introduzione a SDN

Software Defined Networking (SDN) è un'architettura di rete emergente che ha come scopo quello di fornire un'interfaccia che permetta agli sviluppatori software di ispezionare, modificare e controllare i flussi di rete e le connettività delle risorse disponibili. Questa interfaccia serve per permettere la comunicazione con risorse che provengono da diversi venditori e che quindi utilizzano metodi differenti di comunicazione; permette quindi l'astrazione delle risorse fisiche (switch, router) della rete sottostante in modo da semplificare la modifica, la gestione e la personalizzazione da parte dei programmatori. La caratteristica principale di questo nuovo metodo di concepire le reti è il disaccoppiamento tra il piano del controllo e quello dell'instradamento dei dati. Un altro aspetto importante è la migrazione del controllo dai dispositivi di rete individuali ad uno o più dispositivi accessibili che cooperano, centralizzando logicamente l'intelligenza e lo stato della rete. Le architetture SDN si differenziano rispetto quelle tradizionali sotto alcuni aspetti chiave:

- Nelle architetture tradizionali è quasi sempre necessario un intervento umano per elaborare le esigenze di rete descritte dalle applicazioni solitamente in modo implicito e indiretto.

- Le reti tradizionali non offrono un modo dinamico per esprimere l'intera gamma di requisiti dell'utente, ad esempio velocità di trasmissione, ritardo o disponibilità. Le intestazioni dei pacchetti possono codificare le richieste di priorità, ma i fornitori di rete in genere non si fidano dei contrassegni di traffico degli utenti. Pertanto, alcune reti tentano di dedurre direttamente i requisiti degli utenti (ad esempio attraverso l'analisi del traffico), che possono comportare costi aggiuntivi e talvolta portare a una classificazione errata. SDN offre la possibilità per un utente di specificare completamente le proprie esigenze nel contesto di una relazione di fiducia che può essere monetizzata.

- Le reti tradizionali non espongono le informazioni e lo stato di rete alle applicazioni che li utilizzano, mentre utilizzando un approccio SDN le applicazioni possono richiedere informazioni e monitorare lo stato della rete per adattarsi di conseguenza.

Come già accennato, il piano di controllo è logicamente centralizzato e disaccoppiato dal piano dei dati ma questo non implica che il controller sia centralizzato anche fisicamente; infatti esso può essere distribuito in modo tale che diverse istanze del controller fisico cooperino per controllare la rete e servire le applicazioni, ottimizzando prestazioni, scalabilità e/o affidabilità. Inoltre il controller SDN osserva e notifica lo stato di rete per le applicazioni e traduce le richieste delle stesse in regole di basso livello per la rete sottostante. Con SDN, il piano di controllo agisce come un singolo sistema operativo di rete logico e centralizzato sia in termini di pianificazione che di risoluzione dei conflitti di risorse; ogni controller SDN ha infatti un controllo completo degli elementi di rete e quindi non deve competere con altri componenti del piano di controllo, semplificando in questo modo la pianificazione e l'allocazione delle risorse. Ciò consente alle reti di eseguire politiche complesse e precise per sfruttare maggiormente le risorse di rete e offrire una qualità di servizio migliore.

1.3 Struttura di SDN

Esplorando l'architettura SDN notiamo immediatamente che è divisa in 3 grandi strati (Figura 1.1):

- Application layer, raggruppa tutte le applicazioni SDN che hanno bisogno di interfacciarsi con il controller per utilizzare risorse di rete.
- Control layer, mantiene una vista globale della rete ed è lo strato in cui viene centrata logicamente l'intelligenza.
- Infrastructure layer, contenente i dispositivi di rete e si occupa dell'inoltro dei pacchetti seguendo le regole impresse dal controller.

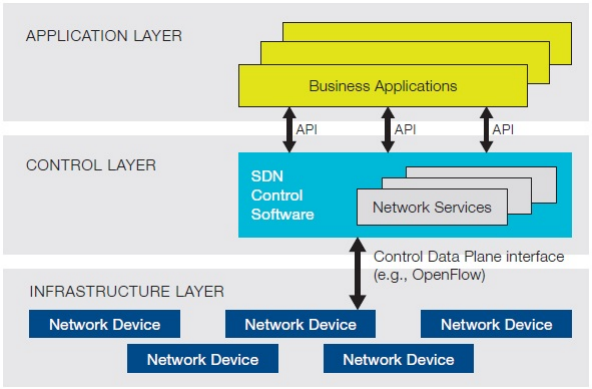


Figura 1.1: Struttura SDN

Dall'immagine (Figura 1.2) si può notare che tra lo strato dell'infrastruttura (Infrastructure layer) e quello di controllo (Control layer) vi è una sezione chiamata Data-Control Plane Interface(D-CPI), questa rappresenta l'interfaccia sud dell'architettura e tramite essa gli elementi di rete devono esporre al piano di controllo le loro capacità. La D-CPI inoltre fornisce programmabilità delle operazioni di instradamento dei dati, la possibilità di riportare statistiche sullo stato della rete e di notificare eventi. Tra lo strato di controllo e quello delle applicazioni (Application layer) è presente un'altra interfaccia chiamata Application-Control Plane Interface(A-CPI), questa contiene le API tramite le quali le applicazioni possono comunicare con i controller per effettuare le loro

richieste; questa viene definita l'interfaccia nord dell'architettura e fornisce la visione di astrazione della rete al piano delle applicazioni. Entrambe le interfacce appena citate sono implementate in modo da essere aperte e indipendenti dai venditori dei dispositivi utilizzati. A fianco di questi tre piani possiamo individuarne un quarto verticale che si occupa della gestione e amministrazione degli altri strati e delle attività statiche che vengono gestite meglio al di fuori degli altri piani. Alcune funzioni presenti in questo strato sono la gestione delle relazioni client-server, l'assegnazione delle risorse ai client, la gestione e la coordinazione delle risorse fisiche e logiche.

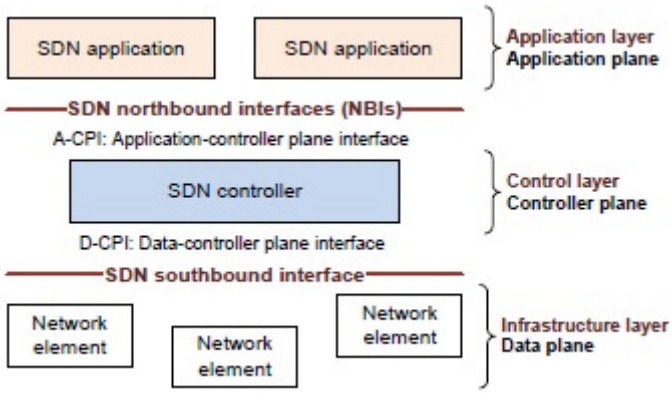


Figura 1.2: Architettura SDN

1.3.1 Componenti principali

Entrando nel dettaglio dell'architettura possiamo individuare tre componenti fondamentali (Figura 1.3):

Controller

Situato nello strato di controllo rappresenta il cervello dell'architettura, in esso viene centralizzata la logica della rete e ha completa visione del modello delle informazioni sotto il suo controllo; esso può operare come singola entità o cooperare con altri controller. Il suo ruolo fondamentale consiste nel coordinare una serie di risorse interconnesse, spesso distribuite su un certo numero di piattaforme subordinate e intanto assicurare l'integrità transazionale come parte del processo; questo processo è chiamato "orchestrazione".

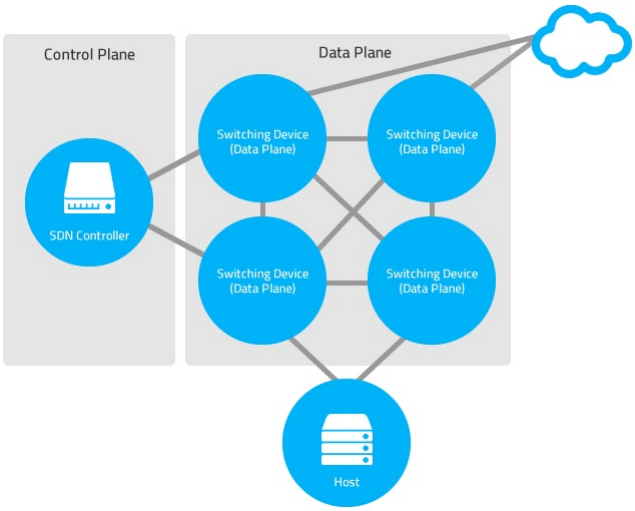


Figura 1.3: Componenti principali

Inoltre si occupa della gestione dei flussi, li controlla e seleziona quelli che possono essere immessi, per questi modifica le tabelle degli switch in modo da fargli seguire il percorso ottimale; per fare ciò il controller deve gestire le tabelle di instradamento degli switch creando o modificando le voci interessate.

Switch

Uno switch (Figura 1.4) è un dispositivo di rete simile ad un hub che si occupa dell'indirizzamento ed instradamento dei flussi di dati mediante l'utilizzo degli indirizzi fisici (MAC), ognuno dei quali corrisponde in modo univoco ad una sola porta. Vi sono 3 tipi diversi di switch: switch tradizionali, switch abilitati ad OpenFlow e switch OpenFlow. Gli switch abilitati ad OpenFlow sono switch tradizionali a cui sono stati aggiunti le tabelle di flusso, il canale protetto (secure channel) e i protocolli OpenFlow; questi supportano quindi l'indirizzamento e l'instradamento dei flussi come uno switch tradizionale ma in aggiunta hanno la possibilità di essere usati come switch OpenFlow. Gli switch OpenFlow invece non supportano i metodi tradizionali ma solo quelli OpenFlow; essi sono composti da tre parti fondamentali: una tabella di flusso (flow table), questa contiene le voci di flusso (flow entry) ed un'azione associata ad ognuna di esse, queste servono per indicare allo switch come elaborare il flusso; un canale protetto (secure channel), è

il collegamento tra uno switch ed un controller, permette la trasmissione di comandi e pacchetti in modo sicuro; il protocollo OpenFlow, è un metodo aperto e standard che utilizza il controller per comunicare con lo switch. Le azioni fondamentali che si possono associare ad una voce nella tabella dei flussi sono quattro:

- Inoltrare i pacchetti del flusso ad una o più porte, questo consente di instradare i pacchetti attraverso la rete;

- Inoltrare i pacchetti del flusso al secure channel dove vengono incapsulati ed inviati al controller, questa azione viene tipicamente effettuata al ricevimento del primo pacchetto di un flusso in modo tale da farlo controllare e fargli decidere se aggiungere il flusso alla flow table;

- Eliminare i pacchetti di un flusso, utilizzata per questioni di sicurezza o per ridurre il traffico falso di discovery degli host;

- Inoltrare i pacchetti del flusso attraverso la pipeline di elaborazione propria dello switch, questa azione è possibile solamente negli switch abilitati ad OpenFlow in quanto gli ultimi non hanno un metodo di indirizzamento tradizionale ma supportano solamente quello di OpenFlow.

End point o Host

Sono i dispositivi che hanno bisogno di scambiare traffico nella rete, cioè i punti di partenza e di arrivo dei dati, essi sono collegati agli switch e rappresentano i punti finali dell'architettura.

1.3.2 Benefici di SDN

I benefici dovuti all'architettura SDN sono molteplici, tra i principali vi sono:

- Il controllo e la gestione delle risorse di rete viene logicamente centralizzato indipendentemente dai fornitori delle risorse fornendo ai programmatori di rete un interfaccia comune.

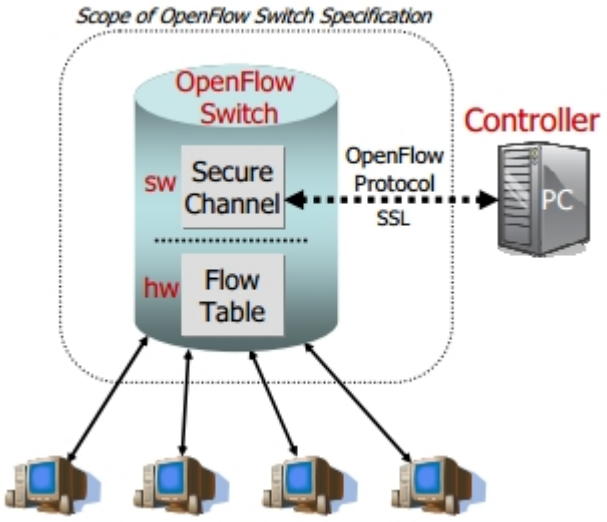


Figura 1.4: Switch OpenFlow

- Fornisce migliore automazione e gestione delle risorse tramite l'utilizzo di API comuni che permettono di astrarre dai dettagli della rete.

- La complessità della rete viene notevolmente ridotta in quanto non è più necessario conoscere migliaia di protocolli ma basterà che i dispositivi accettino le istruzioni dai controller. Inoltre i programmatori non dovranno più lavorare con molte linee di codice sparse su molti dispositivi; si ridurrà quindi anche il tempo per creare nuovi servizi o modificare dinamicamente il comportamento della rete stessa.

- Aumenta la scalabilità e la possibilità di innovare la rete permettendo ai programmatori di operare in tempo reale sulle reti per modificarle in base alle esigenze attuali.

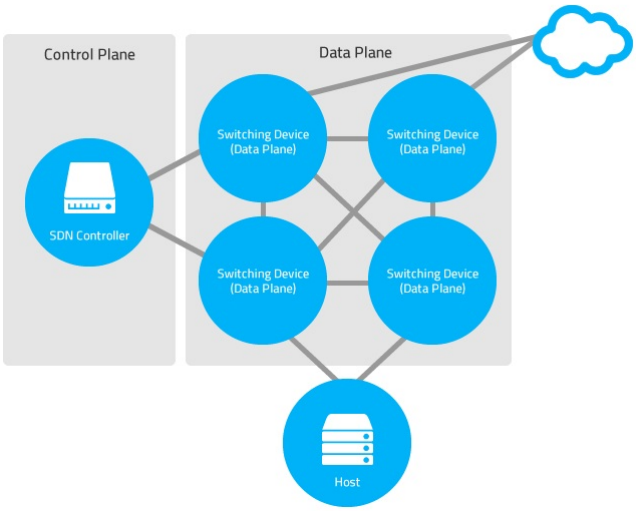


Figura 1.5: Componenti principali

Capitolo 2

OpenFlow

2.1 Introduzione ad OpenFlow

Il problema fondamentale delle reti attuali è dato dalla presenza di una barriera estremamente elevata per l'introduzione di nuove idee creata dall'enorme quantità di apparecchiature e di protocolli, e dalla riluttanza a sperimentare con il traffico; riluttanza data dal fatto che non vi sono modi per testare nuovi protocolli in una rete sufficientemente realistica. Una prima soluzione è lo sviluppo di reti utilizzando switch e router programmabili per poter elaborare contemporaneamente pacchetti per reti multiple, tra cui quelle sperimentali, in modo isolato; l'adozione di queste reti su larga scala però è un processo lungo e costoso. Inoltre, gli switch e i router commerciali solitamente non forniscono una piattaforma software aperta e non permettono la virtualizzazione dei propri dispositivi, ma rilasciano interfacce standard esterne che consentono solamente alcune azioni di base e nascondono le interfacce implementative interne, impedendo quindi ai ricercatori di poterli utilizzare per i propri scopi di sperimentazione. I venditori delle apparecchiature non distribuiscono le interfacce interne dei loro prodotti per motivi di sicurezza, perchè nuovi esperimenti potrebbero far andare in crash le reti, e per motivi economici, in quanto si ridurrebbe la barriera di ingresso di nuovi concorrenti in quel settore. Esistono anche piattaforme aperte ma hanno costo elevato e non soddisfano i requisiti in termini di densità di porte e di prestazioni richieste per testare nuove tecnologie di rete in un ambiente simile a quello reale. Proprio dal bisogno di reti programmabili

nasce l'idea di OpenFlow, un protocollo di rete aperto per la programmazione delle tabelle di flusso degli switch e la prima interfaccia standard di comunicazione tra lo strato di controllo e quello dell'infrastruttura nell'architettura SDN; esso è stato sviluppato inizialmente da docenti delle università di Stanford e Berkeley, ed infine migliorato e standardizzato da Open Networking Foundation (ONF), un'organizzazione senza scopo di lucro che ha come obiettivo principale quello di accelerare l'adozione di SDN. Inizialmente è stato pensato per fornire ai ricercatori un ambiente in cui poter testare i loro protocolli sperimentali, utilizzando una rete composta da switch eterogenei; inoltre, utilizzando questo protocollo, i venditori possono permettere di comunicare con i loro apparecchi senza il bisogno di conoscere i dettagli costruttivi dei loro prodotti. Lo scopo di OpenFlow è quello di creare un modello generale ed unificato di nodo da presentare all'esterno, rendendo le applicazioni indipendenti dall'implementazione delle tecnologie di indirizzamento e di forwarding impiegate dallo specifico venditore. L'idea di base per realizzare questo modello è di rendere possibile la gestione di più switch attraverso il collegamento ad uno o più controller che si occuperanno della definizione delle politiche e della gestione dell'infrastruttura e del traffico, rendendoli più semplici ed efficaci. Il controller avrà il compito di controllare e gestire i flussi andando a modificare le tabelle presenti negli switch; i flussi sono definiti come sequenze di pacchetti unidirezionali con caratteristiche comuni, che devono transitare da una sorgente ad una destinazione fisse attraversando i nodi interessati entro un determinato lasso di tempo. Con questo protocollo le applicazioni potranno configurare il contenuto delle tabelle di classificazione e di instradamento dei pacchetti presenti nei dispositivi tramite le interfacce fornite dal piano di controllo. Riassumendo, OpenFlow permette quindi l'accesso diretto e la manipolazione del piano di forwarding dei dispositivi di rete, che siano switch o router, fisici o virtuali, trasformando così le reti tradizionali in reti programmabili staticamente e dinamicamente.

2.2 Logica di OpenFlow

La logica di OpenFlow rispetta, come accennato precedentemente e come si può notare in Figura 2.1, la struttura dell'architettura SDN, infatti l'ambiente è formato

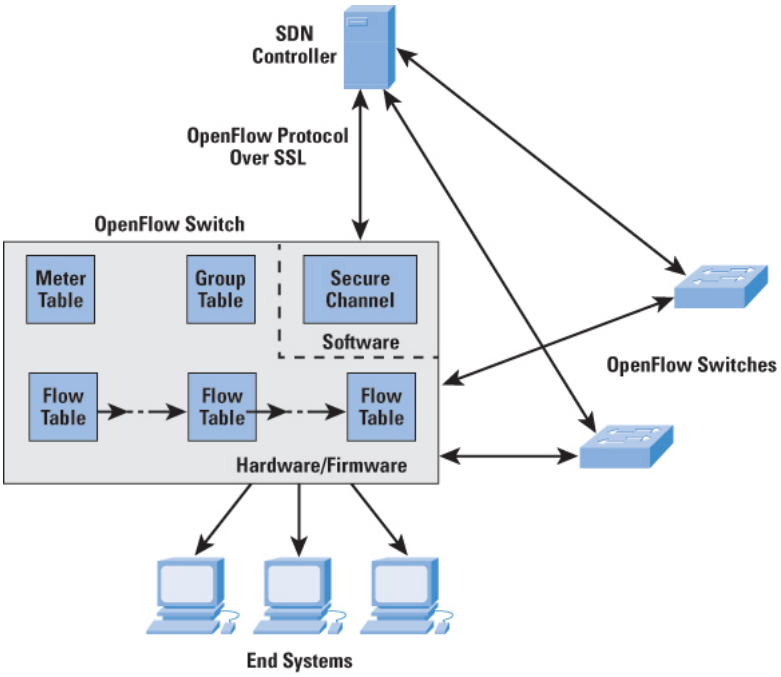


Figura 2.1: Vista generale del protocollo OpenFlow

principalmente di tre componenti: il controller, gli switch e gli end system. Ogni switch è può essere connesso ad altri switch ed a host che rappresentano i punti di partenza ed arrivo dei pacchetti; inoltre può essere connesso ad uno o più controller con cui comunica tramite un canale sicuro, utilizzando il protocollo OpenFlow su SSL (Secure Sockets Layer). Ogni switch è composto da una parte software, che gestisce la connessione coi controller in modo da fornire un canale sicuro con ognuno di essi, ed una composta da una serie di tabelle utilizzate per la gestione dei flussi e che solitamente viene implementata tramite hardware o firmware. Le tabelle definite nelle specifiche di OpenFlow sono di 3 tipologie:

- Flow table, tramite esse si individuano i flussi in ingresso che corrispondono ad una delle voci della tabella, a cui è legata l'azione da svolgere con essi; vi possono essere più flow table connesse per verificare la corrispondenza dei flussi, instradarli o modificarli, questa caratteristica è chiamata pipeline;

- Group table, consentono di individuare un gruppo di flussi ed applicare azioni in base al gruppo di appartenenza;

- Meter table, consente di eseguire azioni su un flusso in base alle prestazioni.

2.2.1 Flow table

L'elemento principale di uno switch è rappresentato dalle flow table in quanto ogni pacchetto in ingresso è obbligato a passare sulla tabella di flusso "zero", ognuna di esse è composta da un numero variabile di voci chiamate flow entries. Ogni entry, come si può notare in Figura 2.2, è formata da 6 campi:

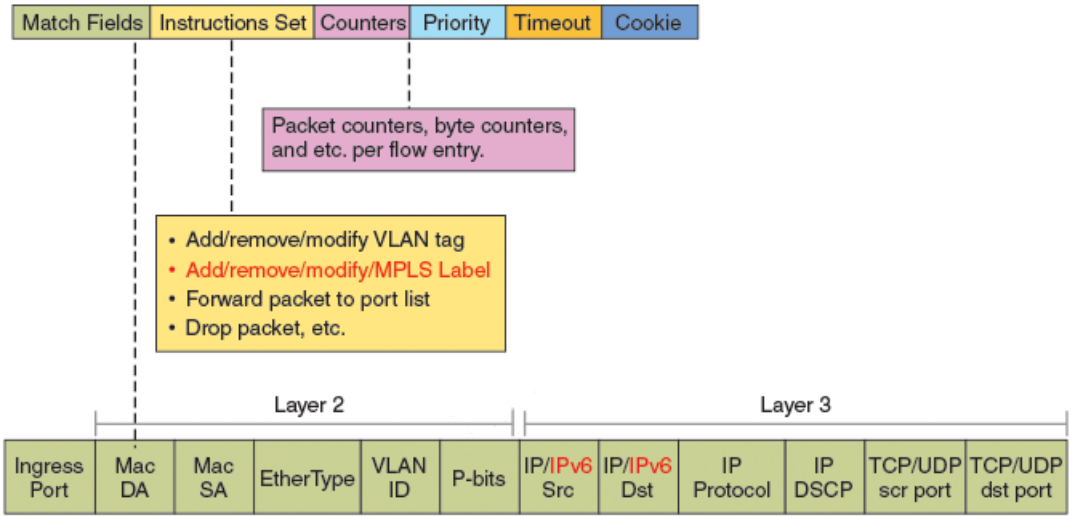


Figura 2.2: Composizione di una flow entries

- Match fields, utilizzato per trovare la corrispondenza dei pacchetti, è composto da più campi che servono per individuare i flussi come la porta di ingresso, l'intestazione del pacchetto ed eventualmente possono esserci metadati di una tabella precedente nella pipeline;

- Instructions set, contiene le istruzioni da eseguire su quel flusso;

- Counters, utilizzato per contare quanti pacchetti corrispondenti a quella entry sono stati elaborati;

- Priority, individua la priorità di tale flow entry;

- Timeout, definisce il tempo massimo di una voce della tabella, dopo tale periodo esse viene cancellata;

- Cookie, dati opachi definiti dal controller, possono essere usati per filtrare le statistiche, le modifiche e le cancellazioni dei flussi.

I campi Match e Priority sono la chiave che identifica in modo univoco una voce nella tabella di flusso, il campo di match è inoltre suddiviso in altri campi che possiamo dividere in obbligatori ed opzionali; tra gli obbligatori troviamo i campi: porta di ingresso, indirizzi ethernet sorgente e destinazione, numero dei protocolli IPv4 o IPv6, indirizzi IPv4 o IPv6 sorgente e destinazione, porte TCP e UDP di sorgente e destinazione; mentre tra gli opzionali troviamo: porta fisica, metadati, tipo di ethernet, ed altri. Le azioni servono per descrivere le operazioni di inoltro di pacchetti, di modifica dei pacchetti, di modifica dell'action set e/o alla pipeline; le azioni possibili definite dalla specifica OpenFlow sono le seguenti:

- Output, inoltra i pacchetti verso la porta in uscita;

- Set-queue, imposta l'ID della coda del pacchetto corrispondente alla porta di uscita e viene usata per lo scheduling e l'inoltro dei pacchetti;

- Drop, è un'azione non esplicita che può essere il risultato di un'instruction set vuoto di una pipiline o di un istruzione clear-actions;

- Group, processa il pacchetto attraverso un gruppo;

- Push-tag/pop-tag, inserisce o toglie il campo tag da un pacchetto VLAN o MPLS.

Un action set è la lista di azioni associata ad un pacchetto, queste vengono aggiunte durante l'elaborazione delle tabelle e vengono eseguite quando il pacchetto esce dal processo della pipeline. Il campo instructions contiene l'insieme di istruzioni da eseguire quando un flusso corrisponde al match; le istruzioni possibili sono le seguenti:

- Indirizzare un pacchetto verso una meter table;

- Eseguire un'azione immediatamente senza aggiungerla all'action set;

- Cancellare le azioni presenti nell'action set;
- Scrivere un'azione nell'action set del flusso;
- Scrivere valori di metadati;
- Indirizzare verso una tabella successiva nella pipeline.

Nel campo instruction può essere presente una sola istruzione per ogni tipo e vengono eseguite seguendo l'ordine utilizzato nell'elenco. Esiste una flow entry particolare chiamata "table-miss" che ha i campi di match settati con un valore jolly che corrisponde a tutti i valori possibili ed il campo priority settato a zero che corrisponde alla priorità minore, questa viene utilizzata per tutti i pacchetti che non hanno una corrispondenza con nessun'altra voce della tabella ed indica come queste debbano essere elaborate.

2.2.2 Pipeline

Uno switch OpenFlow può avere una o più tabelle di flusso; nel caso in cui vi siano più tabelle, ognuna con le proprie voci, viene definito un percorso che i flussi possono seguire in un'unica direzione, questo processo viene effettuato numerando sequenzialmente le flow table e viene denominato pipeline (Figura 2,3). Quando arriva un pacchetto viene elaborato dalla tabella di flusso zero; questa è la prima tabella con cui qualsiasi flusso viene confrontato e deve essere obbligatoriamente presente in ogni switch. Arrivando nella flow table zero i pacchetti avranno metadata e acion set vuoti in quanto non sono ancora stati processati da nessuna tabella, i campi di match e di priorità verranno confrontati con le flow entry per cercare una corrispondenza. Se non è presente nessuna corrispondenza il pacchetto viene eliminato. In caso di corrispondenza con una o più voci viene processata quella con priorità maggiore eseguendo le seguenti azioni:

1. Aggiorna i contatori associati a quella voce;
2. Esegue l'istruzione presente nel campo instruction, questa può essere un aggiornamento dell'action set o di valori dei metadati oppure l'esecuzione di un'azione;
3. Inoltra il pacchetto alla porta d'uscita o ad una tabella che può essere una group table, una meter table o una flow table successiva nella pipeline.

Nel caso l'unica corrispondenza fosse con la voce table-miss, il pacchetto verrebbe processato seguendo l'azione ad essa associata; questa azione potrebbe essere l'inoltro ad una flow table successiva nella pipeline, la cancellazione del pacchetto o l'invio al controller che potrà decidere se creare una nuova entry o cancellare il pacchetto. L'inoltro ad altre flow table è possibile solo verso tabelle successive nella pipeline, cioè con numero maggiore, non è possibile indirizzare un flusso verso una flow table precedente a quella attuale; per l'ultima tabella nella pipeline quindi non sarà possibile inoltrare il pacchetto ad un'altra tabella di flusso. Quando un flusso viene diretto alla porta d'uscita, l'action set del flusso che si è accumulato fino a quel momento viene eseguito ed il pacchetto viene messo in coda per uscire.

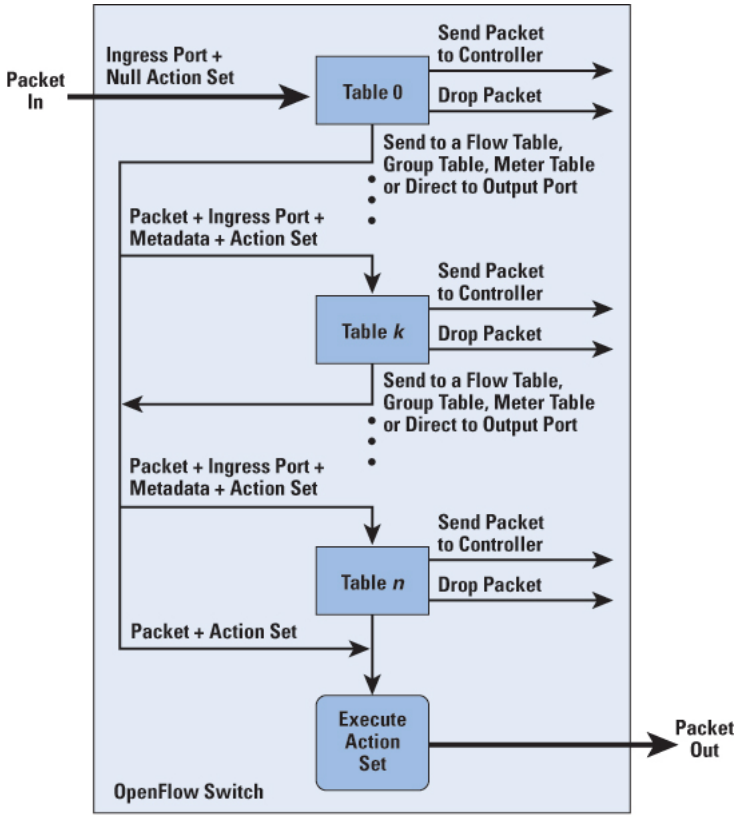


Figura 2.3: Pipeline

2.2.3 Messaggi OpenFlow

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

Controller-To-Switch

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

- Features, attraverso un messaggio features-request il controller chiede l'identità e le capacità di uno switch, lo switch risponde a tale messaggio con un features-response;

- Configuration, con questo messaggio il controller richiede o imposta la configurazione dei parametri dello switch, la risposta viene inviata solo nel caso di una richiesta di informazioni;

- Modify-State, questo messaggio permette al controller di aggiungere, eliminare e modificare le flow-entries o le group-entries delle tabelle e di impostare le proprietà delle porte dello switch;

- Read-State, utilizzato dal controller per richiedere informazioni allo switch come la configurazione corrente, le statistiche e le capacità;

- Packet-out, attraverso questo messaggio il controller può inviare pacchetti in uscita da una specifica porta dello switch o può inoltrare i pacchetti ricevuti dallo switch attraverso un messaggio Packet-in;

- Barrier, utilizzati dal controllore per garantire che le dipendenze dei messaggi siano soddisfatte o per ricevere notifiche quando un operazione viene completata;

- Role-Request, utilizzati dal controller per richiedere o modificare il proprio ruolo;

- Asynchronous-Configuration, attraverso questo messaggio il controller richiede o modifica la lista dei messaggi asincroni che vuole ricevere.

Asincroni

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

- Packet-in, utilizzato dallo switch per trasferire un pacchetto al controller, solitamente utilizzato quando non vi sono voci nella tabella di flusso che corrispondono a quel pacchetto a parte la miss-table;

- Flow-Removed, inviato dallo switch per informare il controller che una voce di flusso è stata rimossa dalla flow table, la causa può essere una richiesta di un controller o l'esaurimento del timeout;

- Port-Status, attraverso questo messaggio lo switch informa il controller del cambiamento della configurazione o dello stato di una porta.

Simmetrici

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

- Hello, scambiato tra controller e switch ad inizio connessione;

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

- Error, viene utilizzato da entrambi per notificare all'altro un avvenuto errore, solitamente inviato dallo switch per indicare al controller un fallimento di una richiesta;

- Experimenter, può essere utilizzato per aggiungere delle funzionalità.

2.2.4 Architettura multi-controller

Un meccanismo per il supporto al failover è rappresentato dall'utilizzo di un'architettura con controller multipli, questo permette di rimpiazzare un controller in caso questo non sia più disponibile fornendo affidabilità ed elasticità al sistema. Oltre a fornire un meccanismo per il recupero dovuto alla perdita di un controller, utilizzare questo tipo di architettura porta vantaggi anche dal punto di vista del bilanciamento del carico. Quando uno switch viene avviato si connette con tutti i controller che sono stati configurati ad esso e cerca di mantenere la connessione con tutti loro contemporaneamente. Più controller possono inviare messaggi controller-to-switch allo switch, le risposte ai comandi vengono inviate sulla connessione relativa al controller che ha effettuato la richiesta; per inviare messaggi asincroni a più controller, il messaggio deve essere duplicato per ogni canale OpenFlow consentito. La presenza di controller multipli oltre ai benefici porta un problema di concorrenza, questo viene gestito attraverso l'utilizzo dei ruoli. Il cambio di ruolo è un meccanismo interamente gestito dai controller e lo switch ha il compito di ricordare i ruoli in modo da aiutarne l'elezione e gestire i messaggi in ingresso ed in uscita da ognuno di essi. I ruoli che un controller può assumere sono tre:

EQUAL È il ruolo di default di un controller all'avvio dello switch; questi controller hanno pieno accesso allo switch, possono inviargli comandi controller-to switch per modificarne lo stato e ricevono tutti i messaggi asincroni. Vi possono essere più controller con questo ruolo.

MASTER I controller master hanno gli stessi privilegi degli equal, l'unica differenza è che lo switch ha il compito di assicurarsi che ci sia solamente un controller con questo ruolo. Nel caso un controller voglia cambiare il suo ruolo in master lo switch deve prima cambiare il ruolo del master attuale in slave e inviargli un messaggio per informarlo, poi eleggerà il nuovo controller.

SLAVE I controller con questo ruolo hanno accesso allo switch solamente in lettura, non possono mandare comandi per modificare lo stato dello switch o per spedire pacchetti; di default non ricevono i messaggi asincroni a parte il messaggio dello stato della porta. Nel caso un controller SLAVE tenti di mandare un comando a lui proibito, lo switch gli risponderà con un messaggio di errore. Vi possono essere più controller con questo ruolo.

Ogni controller può richiedere o modificare il proprio ruolo inviando un messaggio Role-Request allo switch passandogli il ruolo che si vuole assumere ed il generation_id. Il generation_id è un campo numerico e rappresenta un contatore monotono incrementale che viene utilizzato per l'elezione del master; ogni volta che viene effettuata una richiesta per cambiare ruolo in master viene confrontato il generation_id del controller con quello dello switch, nel caso in cui il primo sia inferiore viene considerata una richiesta meno recente e viene scartata inviando al controller che ha richiesto il cambio di ruolo un messaggio di errore, in caso contrario viene effettuata l'elezione del controller a master ed il generation_id dello switch viene incrementato.

Capitolo 3

Verifica del comportamento e delle interazioni tra switch e controller

Questo capitolo illustra le tecnologie ed i software impiegati per sviluppare una rete SDN virtuale che utilizzi il protocollo OpenFlow; su questa rete sono state effettuate modifiche per testare il comportamento e le interazioni tra switch e controller nei casi considerati e spiegati in seguito.

3.1 Tecnologie e software utilizzati

Virtual box e Virtual Machine Manager

Sono software che permettono di create ed eseguire macchine virtuali, in questo modo è possibile condividere le risorse hardware della macchina fisica per poter eseguire sulla macchina virtuale un sistema operativo diverso da quello della macchina fisica. Utilizzando le macchine virtuali è possibile avere diversi sistemi operativi disponibili sulla stessa macchina su cui poter effettuare test senza rischiare di compromettere il sistema operativo principale.
Sito Virtual box: `https://www.virtualbox.org/`
Download Virtual box: `https://www.virtualbox.org/wiki/Downloads`
Documentazione Virtual box: `https://www.virtualbox.org/wiki/Documentation`

Mininet

È un software emulatore che permette di creare e gestire una rete virtuale di host, switch e controller su una sola macchina fisica o virtuale con kernel Linux. Una volta creata la rete è possibile interagire con i componenti creati utilizzando istruzioni a riga di comando. Gli switch di Mininet inoltre supportano il protocollo OpenFlow, questo ci permette di sviluppare ed effettuare test su una rete che utilizza l'architettura SDN.
Sito Mininet: `http://mininet.org/`
Download Mininet: `http://mininet.org/download/`
Documentazione Mininet: `https://github.com/mininet/mininet/wiki/Documentation`

Putty e Xming

Putty è un software utilizzabile su piattaforma Windows che fornisce un'implementazione di un client SSH(Secure SHell). SSH è un protocollo che permette di stabilire una connessione remota cifrata con un altro host della rete; ci consente quindi di comandare una macchina remota utilizzando un'interfaccia a riga di comando. Questo protocollo viene utilizzato per l'amministrazione remota di sistemi UNIX; utilizzando putty come client SSH riusciamo quindi a connetterci ad un sistema Linux per comandarlo in remoto attraverso la shell anche su piattaforma Windows. Xming è un server grafico X open source utilizzabile su piattaforma Windows che permette di visualizzare le applicazioni di Linux da remoto su interfaccia grafica utilizzando il protocollo SSH; essendo solo un display abbiamo bisogno di Putty per avviare le applicazioni remote inoltrando la sessione a Xming. Nel caso si utilizzi un sistema UNIX è possibili utilizzare il protocollo SSH direttamente dalla shell.
Download Putty: `http://www.putty.org/`
Download Xming: `https://sourceforge.net/projects/xming/`

Wireshark

È un software che permette di analizzare i protocolli di rete; attraverso Wireshark è possibile intercettare i pacchetti di una rete per visualizzarne le informazioni, viene solitamente utilizzato per risolvere problemi di rete e per testare software o protocolli

di comunicazione. Nel nostro caso ci permette, attraverso il dissector di OpenFlow, di filtrare i pacchetti con tale protocollo e studiare lo scambio di messaggi nella rete ed in particolare tra switch e controller.
Sito Wireshark: `https://www.wireshark.org/`

Open vSwitch

È una implementazione di switch virtuale multi-server che supporta l'utilizzo di standard e protocolli multipli in una rete; nel nostro caso viene utilizzato come switch virtuale in quanto supporta OpenFlow.
Sito Open vSwitch: `http://openvswitch.org/`
Comandi Open vSwitch: `http://openvswitch.org/support/dist-docs/ovs-vsctl.8.txt`

Controller RYU

È un controller SDN con API ben definite che permettono agli sviluppatori di gestire la rete e controllare le applicazioni con più facilità potendo modificare i componenti o implementandone di nuovi utilizzando python e sfruttando le API.
Documentazione Ryu: `http://ryu.readthedocs.io/en/latest/getting_started.html`

3.2 Descrizione della rete e dei test effettuati

Dopo aver affrontato la parte teorica si è deciso di mettere in pratica gli argomenti studiati in modo da avere un confronto anche dal punto di vista pratico; in una prima parte è stato necessario prendere un po' di dimestichezza con l'ambiente Mininet, effettuando prove sul funzionamento del software su una macchina virtuale locale. Nella seconda parte è stata creata la rete su cui sono state effettuate le verifiche; sono state utilizzate due macchine fisiche in modo da individuare meglio lo scambio di pacchetti distinguendoli tra due indirizzi differenti invece che un solo indirizzo locale. Nella prima macchina fisica con sistema UNIX è stato installato Mininet, con esso sono stati creati uno switch e tre host e sono stati connessi tra loro come si può osservare nella Figura 3.1. Nella seconda macchina fisica sono state create due macchine virtuali contenenti il software Mininet e Ryu-manager con cui poi saranno creati i controller Ryu da connettere

con lo switch presente sull'altra macchina fisica per effettuare i test. I test effettuati si possono dividere in tre casi principali, differenziati per il ruolo dei due controller, questi sono:

1. Un controller Master ed uno Slave;

2. Un controller Master ed un Equal;

2. Il primo controller Master ed il secondo effettua la richiesta di diventare Master.

Attraverso lo studio di questi tre casi possiamo verificare il comportamento e le interazioni tra switch e controller distinti per i ruoli assunti dai controller e verificare come funziona il meccanismo del cambio di ruolo. Nelle sezioni seguenti descriveremo nel dettaglio i passaggi effettuati per svolgere i test e mostreremo i risultati ottenuti per ogni caso di studio.

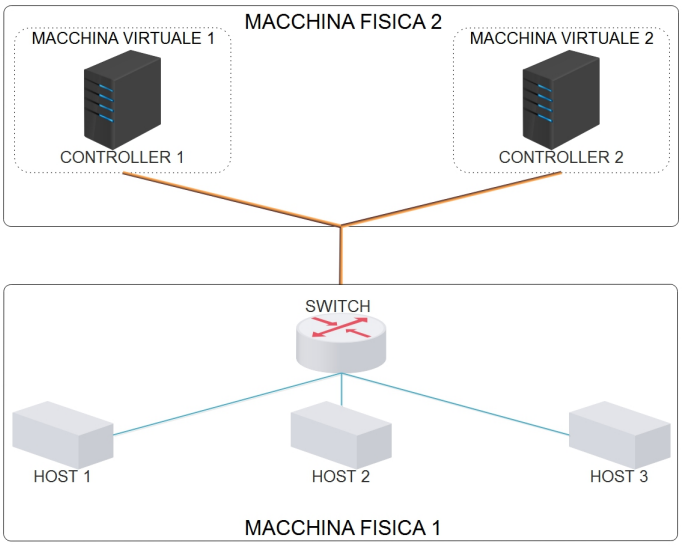


Figura 3.1: Rete in cui sono state effettuate le verifiche

3.3 Preparazione alla fase di verifica

Dopo aver predisposto le macchine fisiche e virtuali installando i software necessari ed impostando la rete in modo che possano comunicare tra loro, abbiamo dovuto iniziare la progettazione della topologia dal lato della prima macchina fisica. Come già accennato precedentemente la macchina deve virtualizzare una rete composta da uno switch e tre host collegati ad esso, inoltre lo switch deve essere predisposto per accettare due controller remoti che si trovano nelle macchine virtuali presenti sul secondo computer. Nella Figura 3.2 troviamo il codice che è stato utilizzato per creare la topologia e che è stato salvato con il nome topomod2.py nella cartella topology; questo codice viene eseguito nella prima macchina fisica utilizzando il seguente comando:

```
sudo python topologiy/topomod2.py
```

Analizzando questo codice notiamo che vi sono due funzioni: init e simpleTest; la prima definisce la struttura della topologia aggiungendo uno switch e tre host attraverso le seguenti istruzioni:

```
switchOVS = self.addSwitch('s1')   <-- aggiunge uno switch
host = self.addHost('h')           <-- aggiunge un host
```

Poi vengono connessi gli host allo switch attraverso l'istruzione:

```
self.addLink(host, switchOVS, bw=10)
```

La seconda funzione chiama la prima per creare la nuova topologia, in seguito crea la rete impostando come topologia quella da noi appena creata, utilizzando l'istruzione:

```
net = Mininet(topo, build=False, link=TCLink, xterm=True)
```

Una volta creata la rete possiamo aggiungerci i controller remoti fornendogli ip e porta in cui andranno a connettersi nel seguente modo:

```
net.addController(name='c0', controller=RemoteController,
                  ip='10.20.20.11', port=6633)
```

Aggiunti i controller la rete è completa e possiamo avviarla attraverso la chiamata della funzione net.start(); di seguito sono stati cambiati gli indirizzi MAC degli host per renderli sequenziali ed è stato utilizzata l'istruzione CLI(net) per poter permettere di comunicare con gli host tramite la linea di comando di mininet.

Una volta creata la rete è stata testata per verificarne il corretto funzionamento; per farlo abbiamo eseguito la topologia sulla prima macchina fisica attraverso il comando vi-

```
from mininet.util import quietRun

from time import sleep
import os
import sys

class MyTopology(Topo):
    def __init__(self, n=2, **opts):
        # Initialize topology and default options
        Topo.__init__(self, **opts)
        #Adding switch
        switchOVS = self.addSwitch('s1')
        #Adding host (host connected to the left switch)
        host1 = self.addHost('h1')
        host2 = self.addHost('h2')
        host3 = self.addHost('h3')
        self.addLink(host1, switchOVS, bw=10)
        self.addLink(host2, switchOVS, bw=10)
        self.addLink(host3, switchOVS, bw=10)

def simpleTest():
    "Create and test simple Network"
    topo = MyTopology(2)
    #1. With following Mininet(), it runs default controller
    #net = Mininet(topo, link=TCLink)
    net = Mininet(topo, build=False, link=TCLink, xterms=True)
    net.addController(name='c0', controller=RemoteController, ip='10.20.20.11', port=6633)

    net.addController(name='c1', controller=RemoteController, ip='10.20.20.12', port=6633)

    net.start()
    print "Network started!"
    print "Chainging MAC address to each host:"
    nhost = len(net.hosts)
    for i in range(nhost):
        h = net.getNodeByName("h%s" % (i + 1 ))
        h.setMAC("00:00:00:00:00:0%s" % (i + 1), h.name + "-eth0" )
        result = h.cmd('sysctl -w net.ipv6.conf.all.disable_ipv6=1')
        print result
        result = h.cmd('sysctl -w net.ipv6.conf.default.disable_ipv6=1')
        print result
        result = h.cmd('sysctl -w net.ipv6.conf.lo.disable_ipv6=1')
        print result

        result = h.cmd('cat /proc/sys/net/ipv6/conf/all/disable_ipv6')

    CLI(net)
    print "Going to stop network!"
    net.stop()

if __name__ == "__main__":
    setLogLevel('info')
    simpleTest()
```

Figura 3.2: Codice della topologia utilizzata

sto precedentemente e sono stati avviati, nelle macchine virtuali, due controller semplici forniti da Ryu, utilizzando il comando:

```
sudo ryu-manager --verbose ryu/ryu/app/simple_switch_13.py
```

Oltre a verificare il corretto funzionamento della topologia è stato utilizzando Wireshark per analizzare i pacchetti scambiati tra i controller e gli switch ad inizio connessione, questi sono riportati nella Figura 3.3. In questa schermata notiamo che ad inizio connessione entrambi mandano un messaggio del tipo OFPT_HELLO per accordarsi sulla versione del protocollo OpenFlow da utilizzare, che coinciderà con la più bassa dei due; dopo il controller manda un messaggio OFPT_FEATURES_REQUEST per richiedere le capacità dello switch, questo risponde con un OFPT_FEATURES_REPLY contenente le informazioni richieste. Successivamente il controller chiede le informazioni sullo stato del datapath ed in particolare la descrizione delle porte utilizzando il messaggio

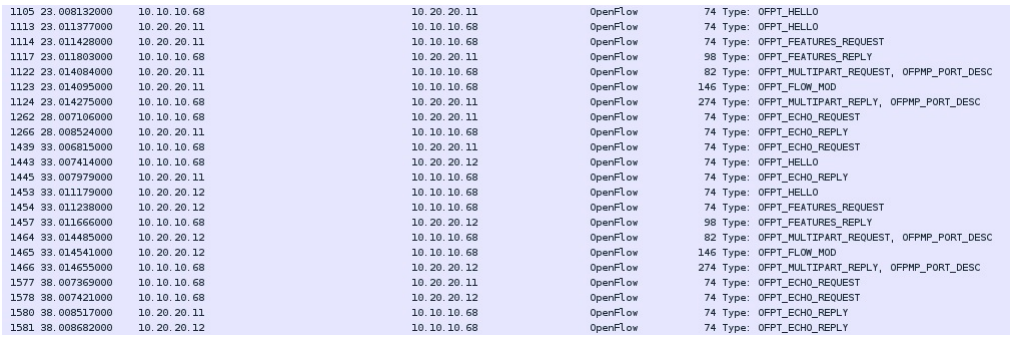


Figura 3.3: Schermata di Wireshark che mostra i pacchetti ad inizio connessione

OFPT_MULTIPART_REQUEST e ne segue la risposta OFPT_MULTIPART_REPLY dallo switch; infine il controller manda il comando OFPT_FLOW_MOD per aggiungere una voce nella flow table dello switch. Dopo aver verificato il corretto funzionamento della topologia si è dovuto pensare ad un modo per effettuare il cambio di ruolo dei controller; l'idea pensata per fare ciò è quella di modificare il controller simple_switch_13.py che abbiamo usato nel test precedente, aggiungendogli le istruzioni per fargli spedire un messaggio di richiesta di cambio ruolo alla ricezione di un Packet_in. Si è pensato di creare una copia del file che poi è stata rinominata in prova.py ed infine modificata per aggiungere il cambio di ruolo. Nel codice del file prova.py sono state fatte due modifiche:

1. Come si pò vedere in Figura 3.4, è stato cambiato il nome passato nella funzione super() e quello della classe, da simple_switch a prova, poi sono stati aggiunti i campi gen_id e cont settati a 0, il primo serve per il cambio di ruolo in Master mentre il secondo è stato utilizzato per inviare solamente un messaggio alla ricezione del primo packet_in.

2. Come si pò vedere in Figura 3.5, sono state aggiunte le istruzioni per preparare e spedire un messaggio del tipo OFPRoleRequest allo switch, utilizzato per richiedere il cambio di ruolo.

Una volta effettuate queste modifiche è possibile verificare che funzionino correttamente utilizzando un comando, nel terminale dello switch, per far mostrare a video le tabelle con le informazioni dei controller relative a quello switch; questo comando è il seguente:

```
ovs-vsctl list Controller
```

```
class prova(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(prova, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self.gen_id = 0
        self.cont = 0
```

Figura 3.4: Codice prima modifica in prova.py

```
if self.cont < 0:
        role = ofproto.OFPCR_ROLE_SLAVE
        mss = datapath.ofproto_parser.OFPRoleRequest(datapath, role, self.gen_id)
        datapath.send_msg(mss)
        role_string = ['NOCHANGE', 'EQUAL', 'MASTER', 'SLAVE']
        print 'Cambio ruolo in %s' % (role_string[role])
        self.cont = self.cont + 1
```

Figura 3.5: Codice seconda modifica in prova.py

Osservando la Figura 3.6(a) possiamo notare che lo switch ha due voci corrispondenti ai due controller che gli sono stati configurati, da questa immagine inoltre possiamo capire che la connessione non è ancora stata effettuata con nessuno dei due controller in quanto, scorrendo gli attributi dei controller, ci accorgiamo che le voci is_connected di entrambi sono impostate a false. Un'altra voce essenziale per le nostre verifiche è role; questo campo può avere tre valori possibili coincidenti con i ruoli di cui abbiamo precedentemente parlato, l'unica differenza è che il ruolo Equal viene chiamato Other. Nella Figura 3.6(b) possiamo vedere i controller una volta connessi, in particolare notiamo che il ruolo di questi ad inizio connessione è Equal dato che non sono state ancora effettuate richieste per il cambio di ruolo. Infine la Figura 3.6(c) mostra le informazioni dei controller dopo che lo switch ha inviato il primo packet_in e di conseguenza i controller hanno richiesto il cambio ruolo; in particolare notiamo che dopo tale richiesta i ruoli dei controller sono cambiati dalla precedente impostazione Other, uno in Master ed uno in Slave. Come ultimo controllo si è scelto di analizzare i pacchetti scambiati tra controller e switch tramite Wireshark per verificare che lo scambio di messaggi coincida con la dinamica da noi aspettata; in Figura 3.7 possiamo notare che dopo aver ricevuto il packet_in i controller rispondono con un packet_out seguito da un messaggio OFPT_ROLE_REQUEST per richiedere il cambio ruolo, esattamente come era stato previsto. A questo punto abbiamo la rete funzionante e siamo in grado di cambiare ruolo ai controller quindi è possibile

iniziare i test dei casi introdotti precedentemente.

```
root@minitower68:~# ovs-vsctl list Controller
_uuid               : 92192a61-7a2f-4efe-86e1-25fc3d1df9f0
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : false
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : other
status              : {last_error="Connection refused", sec_since_disconnect="0"
, state=BACKOFF}
target              : "tcp:10.20.20.12:6633"

_uuid               : 07372fd3-fcfc-4485-b6c6-922831929b4e
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : false
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : other
status              : {last_error="Connection refused", sec_since_disconnect="0"
, state=BACKOFF}
target              : "tcp:10.20.20.11:6633"
```

(a) Prima della connessione

```
root@minitower68:~# ovs-vsctl list Controller
_uuid               : 92192a61-7a2f-4efe-86e1-25fc3d1df9f0
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : true
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : other
status              : {last_error="Connection refused", sec_since_connect="3", s
ec_since_disconnect="4", state=ACTIVE}
target              : "tcp:10.20.20.12:6633"

_uuid               : 07372fd3-fcfc-4485-b6c6-922831929b4e
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : true
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : other
status              : {last_error="Connection refused", sec_since_connect="22",
sec_since_disconnect="23", state=ACTIVE}
target              : "tcp:10.20.20.11:6633"
```

(b) Dopo la connessione

```
root@minitower68:~# ovs-vsctl list Controller
_uuid               : 92192a61-7a2f-4efe-86e1-25fc3d1df9f0
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : true
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : slave
status              : {last_error="Connection refused", sec_since_connect="33",
sec_since_disconnect="34", state=ACTIVE}
target              : "tcp:10.20.20.12:6633"

_uuid               : 07372fd3-fcfc-4485-b6c6-922831929b4e
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : true
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : master
status              : {last_error="Connection refused", sec_since_connect="52",
sec_since_disconnect="53", state=ACTIVE}
target              : "tcp:10.20.20.11:6633"
```

(c) Dopo il cambio di ruolo

Figura 3.6: Lista dei controller vista dallo switch

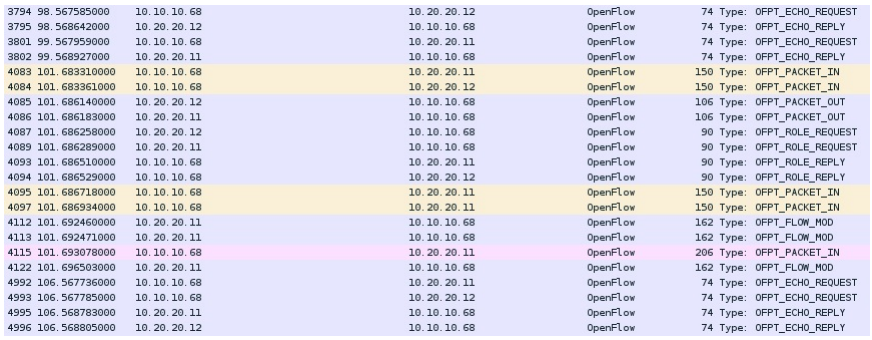


Figura 3.7: Schermata di Wireshark della dinamica del cambio di ruolo

3.3.1 Master-Slave

`Indirizzo switch: 10.10.10.68`

`Indirizzo controller Master: 10.20.20.11`

`Indirizzo controller Slave: 10.20.20.12`

Per prima cosa si sono messi ad eseguire la topologia ed i controller, in questa situazione troviamo i controller connessi ma ancora entrambi con ruolo Other; a questo punto dobbiamo effettuare il cambio di ruolo facendo passare il primo controller a Master e l'altro a Slave. Per farlo eseguiamo un ping dal primo host h1 al secondo h2 (vedi primo ping in Figura 3.8), in questo modo lo switch genera un packet_in e lo invia ai due controller dato che hanno entrambi ruolo Equal; questi lo ricevono e rispondono con un packet_out seguito dalla richiesta di cambio ruolo. A questo punto ci ritroviamo nella situazione con un controller Master ed uno Slave, la prima cosa che notiamo (vedi Figura 3.7) è che lo switch dopo il cambio ruolo non genera più messaggi del tipo packet_in diretti allo Slave ma solamente al Master; invece i messaggi del tipo Echo continuano anche verso lo Slave. Per osservare meglio questa situazione è stato effettuato un ping dal primo host h1 al terzo h3 (vedi secondo ping in Figura 3.8) in modo da generare un altro messaggio packet_in che, come mostrato in Figura 3.9, viene inviato solamente al controller master. In seguito è stato disconnesso il controller Master e, tramite il comando `"ovs-vsctl list Controller"` eseguito nel terminale dello switch, siamo andati ad osservare la lista dei controller configurati con tale switch come possiamo vedere in Figura 3.10(a). Analizzando le Figure 3.10(a) e 3.10(b) notiamo che dopo la disconnessione del Master, lo Slave rimane tale e non effettua nessuna richiesta per cercare di cambiare ruolo;

inoltre lo switch modifica la propria tabella interna in cui mantiene le informazioni sui controller, cambiando il ruolo del controller disconnesso in Other. Infine è stato effettuato un ping dal secondo host h2 al terzo h3 (vedi terzo ping in Figura 3.8) e in questo caso l'unico controller connesso allo switch è quello con ruolo Slave; in questa situazione lo switch non invia alcun messaggio packet_in e il controller Slave non può aggiungere nessuna voce di flusso per gestire il pacchetto, di conseguenza il flusso viene eliminato ed il ping fallisce.

```
mininet> h1 ping -c 1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=14.8 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 14.869/14.869/14.869/0.000 ms
mininet> h1 ping -c 1 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=11.2 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 11.250/11.250/11.250/0.000 ms
mininet> h2 ping -c 1 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

Figura 3.8: Schermata del terminale contenente i ping

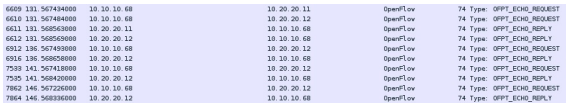
4177	70.029581000	10.10.10.68	10.20.20.11	OpenFlow	74 Type: OFPT_ECHO_REQUEST
4178	70.029633000	10.10.10.68	10.20.20.12	OpenFlow	74 Type: OFPT_ECHO_REQUEST
4179	70.030735000	10.20.20.12	10.10.10.68	OpenFlow	74 Type: OFPT_ECHO_REPLY
4180	70.030765000	10.20.20.11	10.10.10.68	OpenFlow	74 Type: OFPT_ECHO_REPLY
4212	71.838784000	10.10.10.68	10.20.20.11	OpenFlow	150 Type: OFPT_PACKET_IN
4213	71.841242000	10.20.20.11	10.10.10.68	OpenFlow	106 Type: OFPT_PACKET_OUT
4215	71.841800000	10.10.10.68	10.20.20.11	OpenFlow	150 Type: OFPT_PACKET_IN
4222	71.845102000	10.20.20.11	10.10.10.68	OpenFlow	162 Type: OFPT_FLOW_MOD
4223	71.845743000	10.10.10.68	10.20.20.11	OpenFlow	206 Type: OFPT_PACKET_IN
4228	71.848982000	10.20.20.11	10.10.10.68	OpenFlow	162 Type: OFPT_FLOW_MOD
4277	75.029044000	10.10.10.68	10.20.20.12	OpenFlow	74 Type: OFPT_ECHO_REQUEST
4278	75.030120000	10.20.20.12	10.10.10.68	OpenFlow	74 Type: OFPT_ECHO_REPLY
4301	76.029510000	10.10.10.68	10.20.20.11	OpenFlow	74 Type: OFPT_ECHO_REQUEST
4302	76.030425000	10.20.20.11	10.10.10.68	OpenFlow	74 Type: OFPT_ECHO_REPLY

Figura 3.9: Schermata di Wireshark, Packet_in solo al Master

```
root@minitower68:~# ovs-vsctl list Controller
_uuid               : 92192a61-7a2f-4efe-86e1-25fc3d1df9f0
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : true
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : slave
status              : {last_error="Connection refused", sec_since_connect="63",
sec_since_disconnect="64", state=ACTIVE}
target              : "tcp:10.20.20.12:6633"

_uuid               : 07372fd3-fcfc-4485-b6c6-922831929b4e
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : false
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : other
status              : {last_error="Connection refused", sec_since_connect="82",
sec_since_disconnect="0", state=BACKOFF}
target              : "tcp:10.20.20.11:6633"
```

(a) Lista dei controller vista dallo switch



(b) Schermata di Wireshark

Figura 3.10: Master disconnesso

3.3.2 Master-Equal

`Indirizzo switch: 10.10.10.68`

`Indirizzo controller Master: 10.20.20.11`

`Indirizzo controller Equal: 10.20.20.12`

Come nel caso precedente per prima cosa eseguiamo la topologia ed i due controller poi effettuiamo un ping dal primo host h1 al secondo h2 (vedi primo ping in Figura 3.11) per cambiare il ruolo dei controller uno in Master e l'altro in Equal, anche se per il secondo non era necessario dato che di default il ruolo è Equal. A questo punto, come possiamo vedere in Figura 3.12, ci troviamo nella situazione voluta con un Master ed un Equal e possiamo procedere effettuando un ping dal primo host h1 al terzo h3 (vedi secondo ping in Figura 3.12). Analizzando la Figura 3.13 possiamo osservare che in questo caso lo switch spedisce i messaggi di tipo packet_in sia al Master che all'Equal ed entrambi gli rispondono generando pacchetti OFPT_FLOW_MOD per andare a modificare le flow table aggiungendo la voce che gestirà quel flusso. A questo punto viene disconnesso il controller Master e viene effettuato un ping dal secondo host h2 al terzo h3 (vedi terzo ping in Figura 3.12); non essendoci più il Master verranno inviati pacchetti solamente al

controller Equal che, come possiamo vedere in Figura 3.14(b), gestirà il flusso aggiungendo una nuova voce nella flow table dello switch. Dalla Figura 3.14(a) invece possiamo vedere che quando il Master viene disconnesso, il controller Equal rimane tale e non richiede di cambiare ruolo per sostituirlo.

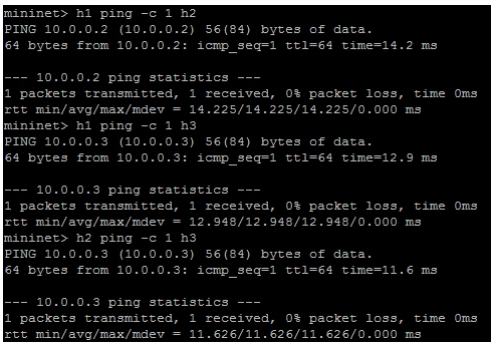


Figura 3.11: Schermata del terminale contenente i ping

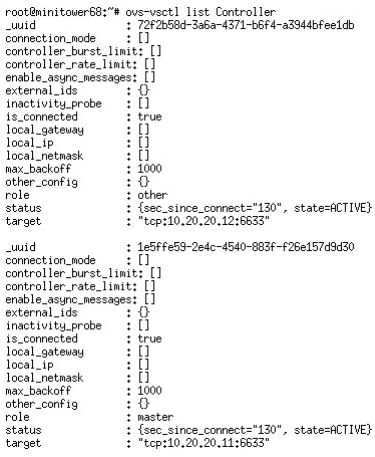
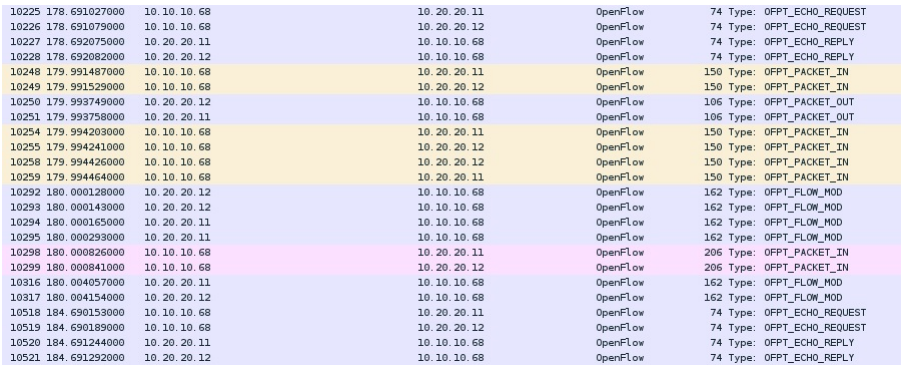


Figura 3.12: Lista dei controller vista dallo switch, un Master ed un Equal



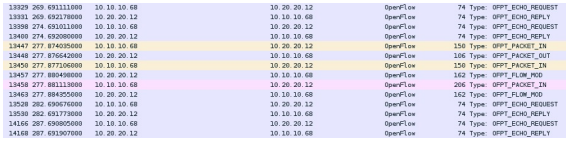
10225	178.691027000	10.10.10.68	10.20.20.11	OpenFlow	74 Type: OFPT_ECHO_REQUEST
10226	178.691079000	10.10.10.68	10.20.20.12	OpenFlow	74 Type: OFPT_ECHO_REQUEST
10227	178.692075000	10.20.20.11	10.10.10.68	OpenFlow	74 Type: OFPT_ECHO_REPLY
10228	178.692082000	10.20.20.12	10.10.10.68	OpenFlow	74 Type: OFPT_ECHO_REPLY
10248	179.991487000	10.10.10.68	10.20.20.11	OpenFlow	150 Type: OFPT_PACKET_IN
10249	179.991529000	10.10.10.68	10.20.20.12	OpenFlow	150 Type: OFPT_PACKET_IN
10250	179.993749000	10.20.20.12	10.10.10.68	OpenFlow	106 Type: OFPT_PACKET_OUT
10251	179.993758000	10.20.20.11	10.10.10.68	OpenFlow	106 Type: OFPT_PACKET_OUT
10254	179.994203000	10.10.10.68	10.20.20.11	OpenFlow	150 Type: OFPT_PACKET_IN
10255	179.994241000	10.10.10.68	10.20.20.12	OpenFlow	150 Type: OFPT_PACKET_IN
10258	179.994426000	10.10.10.68	10.20.20.12	OpenFlow	150 Type: OFPT_PACKET_IN
10259	179.994464000	10.10.10.68	10.20.20.11	OpenFlow	150 Type: OFPT_PACKET_IN
10292	180.000128000	10.20.20.12	10.10.10.68	OpenFlow	162 Type: OFPT_FLOW_MOD
10293	180.000143000	10.20.20.12	10.10.10.68	OpenFlow	162 Type: OFPT_FLOW_MOD
10294	180.000165000	10.20.20.11	10.10.10.68	OpenFlow	162 Type: OFPT_FLOW_MOD
10295	180.000293000	10.20.20.11	10.10.10.68	OpenFlow	162 Type: OFPT_FLOW_MOD
10298	180.000826000	10.10.10.68	10.20.20.11	OpenFlow	206 Type: OFPT_PACKET_IN
10299	180.000841000	10.10.10.68	10.20.20.12	OpenFlow	206 Type: OFPT_PACKET_IN
10316	180.004057000	10.20.20.11	10.10.10.68	OpenFlow	162 Type: OFPT_FLOW_MOD
10317	180.004154000	10.20.20.12	10.10.10.68	OpenFlow	162 Type: OFPT_FLOW_MOD
10518	184.690153000	10.10.10.68	10.20.20.11	OpenFlow	74 Type: OFPT_ECHO_REQUEST
10519	184.690189000	10.10.10.68	10.20.20.12	OpenFlow	74 Type: OFPT_ECHO_REQUEST
10520	184.691244000	10.20.20.11	10.10.10.68	OpenFlow	74 Type: OFPT_ECHO_REPLY
10521	184.691292000	10.20.20.12	10.10.10.68	OpenFlow	74 Type: OFPT_ECHO_REPLY

Figura 3.13: Schermata di Wireshark, Packet_in ad entrambi i controller

```
root@minitower68:~# ovs-vsctl list Controller
_uuid               : aa9f028a-8bc3-41e9-961d-4dec37778f41
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : true
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : other
status              : {sec_since_connect="395", state=ACTIVE}
target              : "tcp:10.20.20.12:6633"

_uuid               : 8ad00306-0834-48c4-9c73-f13a01134a77
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : false
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : other
status              : {last_error="Connection refused", sec_since_connect="395",
 sec_since_disconnect="0", state=BACKOFF}
target              : "tcp:10.20.20.11:6633"
```

(a) Lista dei controller vista dallo switch



13929	269.691111000	10.10.10.68	10.20.20.12	OpenFlow	74 Type: OFPT_ECHO_REQUEST
13931	269.692178000	10.20.20.12	10.10.10.68	OpenFlow	74 Type: OFPT_ECHO_REPLY
13398	274.691011000	10.10.10.68	10.20.20.12	OpenFlow	74 Type: OFPT_ECHO_REQUEST
13400	274.692080000	10.20.20.12	10.10.10.68	OpenFlow	74 Type: OFPT_ECHO_REPLY
13447	277.874035000	10.10.10.68	10.20.20.12	OpenFlow	150 Type: OFPT_PACKET_IN
13448	277.876642000	10.20.20.12	10.10.10.68	OpenFlow	106 Type: OFPT_PACKET_OUT
13450	277.877106000	10.10.10.68	10.20.20.12	OpenFlow	150 Type: OFPT_PACKET_IN
13457	277.880498000	10.20.20.12	10.10.10.68	OpenFlow	162 Type: OFPT_FLOW_MOD
13458	277.881113000	10.10.10.68	10.20.20.12	OpenFlow	206 Type: OFPT_PACKET_IN
13463	277.884355000	10.20.20.12	10.10.10.68	OpenFlow	162 Type: OFPT_FLOW_MOD
13528	282.690676000	10.10.10.68	10.20.20.12	OpenFlow	74 Type: OFPT_ECHO_REQUEST
13530	282.691778000	10.20.20.12	10.10.10.68	OpenFlow	74 Type: OFPT_ECHO_REPLY
14166	287.690805000	10.10.10.68	10.20.20.12	OpenFlow	74 Type: OFPT_ECHO_REQUEST
14168	287.691907000	10.20.20.12	10.10.10.68	OpenFlow	74 Type: OFPT_ECHO_REPLY

(b) Schermata di Wireshark

Figura 3.14: Master disconnesso

3.3.3 Cambio di Master

`Indirizzo switch: 10.10.10.68`

`Indirizzo primo controller Master: 10.20.20.11`

`Indirizzo secondo controller Master: 10.20.20.12`

Eseguiamo la topologia ma in questo caso avviamo solamente un controller ed effettuiamo un ping dal primo host h1 al secondo h2 per fare il cambio di ruolo e farlo diventare

```
root@minitower68:~# ovs-vsctl list Controller
_uuid               : 487cf8f7-fe1b-4b5a-bc9d-e926cf7473e2
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : true
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : other
status              : {last_error="Connection refused", sec_since_connect="6", s
ec_since_disconnect="7", state=ACTIVE}
target              : "tcp:10.20.20.12:6633"

_uuid               : 2be6743b-8c40-40b0-a857-9dcea6a805d4
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : true
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : master
status              : {sec_since_connect="35", state=ACTIVE}
target              : "tcp:10.20.20.11:6633"
```

Figura 3.15: Lista dei controller vista dallo switch, un Master ed un Equal

Master; successivamente possiamo avviare il secondo controller e ci troveremo nella situazione in cui avremo un Master ed un Equal come possiamo notare dalla Figura 3.15. A questo punto eseguiamo un ping dal primo host h1 al terzo h3, in questo modo il secondo controller effettuerà la richiesta di cambio ruolo per diventare il prossimo Master; questo passaggio lo possiamo vedere nel dettaglio analizzando lo scambio dei pacchetti attraverso l'utilizzo di Wireshark riportato in Figura 3.16. In questa immagine è possibile notare in primo luogo i packet_in, questi vengono inviati dallo switch ad entrambi i controller dato che uno ha ruolo Master e l'altro Equal; dopo possiamo individuare il messaggio OFPT_ROLE_REQUEST con cui il secondo controller richiede di passare a Master, seguito poi dalla risposta dello switch che conferma il cambio di ruolo. Successivamente il controller Master viene declassato dallo switch al ruolo Slave; in questi test è stata utilizzata la versione 1.3 del protocollo OpenFlow che non prevede alcun messaggio di notifica al controller Master che viene declassato a Slave, questa sarà aggiunta nella versione 1.4 del protocollo. A questo punto ci troviamo con un controller Master ed uno Slave, entrambi hanno ricevuto il messaggio packet_in e di conseguenza cercheranno di gestire il flusso andando a modificare la flow table dello switch per aggiungere la voce interessata. Per quanto riguarda il Master l'operazione viene eseguita con successo in quanto ha pieno accesso allo switch; situazione differente per il controller Slave che ha ricevuto il packet_in quando il suo ruolo era ancora Master ma essendo stato declassato non ha più i permessi necessari per andare a modificare lo stato dello switch. Infatti,

in risposta al messaggio OFPT_FLOW_MOD dello Slave, lo switch invia un messaggio di errore OFPT_ERROR per notificargli il fallimento dell'operazione; questo messaggio viene mostrato nel terminale del controller Slave, riportato in Figura 3.17, e mostra l'errore segnalato dallo switch. Dall'immagine possiamo vedere che il tipo di errore è OFPET_BAD_REQUEST con codice OFPBRC_IS_SLAVE riferito ad un messaggio del tipo OFPT_FLOW_MOD, che significa che la richiesta è fallita perchè il controller ha ruolo Slave e non ha i permessi per effettuare una modifica della flow table. Infine, in Figura 3.18, possiamo vedere la lista dei controller vista dallo switch che ci conferma che la situazione finale è proprio quella appena descritta.

```
EventOFPErrorMsg received.
version=0x4, msg_type=0x1, msg_len=0x4c, xid=0x4b0f6cdf
 `-- msg_type: OFPT_ERROR(1)
OFPErrorMsg(type=0x1, code=0xa, data=b'\x04\x0e\x00\x60\x4b\x0f\x6c\xdf\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x01\x00\x00\x01\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\
x00\x16\x80\x00\x00\x04\x00\x00\x00\x01\x80\x00\x06\x06')
 |-- type: OFPET_BAD_REQUEST(1)
 |-- code: OFPBRC_IS_SLAVE(10)
 `-- data: version=0x4, msg_type=0xe, msg_len=0x60, xid=0x4b0f6cdf
     `-- msg_type: OFPT_FLOW_MOD(14)
```

Figura 3.16: Schermata del terminale con il messaggio di errore inviato al nuovo Slave

```
root@minitower68:~# ovs-vsctl list Controller
_uuid               : 487cf8f7-fe1b-4b5a-bc9d-e926cf7473e2
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : true
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : master
status              : {last_error="Connection refused", sec_since_connect="36",
sec_since_disconnect="37", state=ACTIVE}
target              : "tcp:10.20.20.12:6633"

_uuid               : 2be6743b-8c40-40b0-a857-9dcea6a805d4
connection_mode     : []
controller_burst_limit: []
controller_rate_limit: []
enable_async_messages: []
external_ids        : {}
inactivity_probe    : []
is_connected        : true
local_gateway       : []
local_ip            : []
local_netmask       : []
max_backoff         : 1000
other_config        : {}
role                : slave
status              : {sec_since_connect="65", state=ACTIVE}
target              : "tcp:10.20.20.11:6633"
```

Figura 3.17: Lista dei controller vista dallo switch, un Master ed uno Slave

Conclusioni

L'obiettivo di questa tesi era la verifica delle interazioni tra switch e controller in un'architettura SDN che utilizza il protocollo OpenFlow, in particolare nel caso multi-controller; inizialmente si sono studiati SDN ed OpenFlow poi è stata costruita una rete virtuale su cui sono stati effettuati i test. Attraverso questi test abbiamo verificato il comportamento e le interazioni tra lo switch ed i controller ad esso connessi differenziandoli in base al ruolo assunto da ogni controller.

Bibliografia

[1] Opening Networking Foundation, *Software-Defined Networking (SDN) Definition*, https://www.opennetworking.org/sdn-resources/sdn-definition.

[2] Opening Networking Foundation, *SDN Architecture Overview version 1.0*, https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf.

[3] Opening Networking Foundation, *Software-Defined Networking: The New Norm for Networks*, https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[4] Opening Networking Foundation, *OpenFlow*, https://www.opennetworking.org/sdn-resources/openflow.

[5] Opening Networking Foundation, *OpenFlow Switch Specification version 1.4*, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

[6] Opening Networking Foundation, *SDN architecture*, https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf.

[7] CISCO, *Software-Defined Networks and OpenFlow - The Internet Protocol Journal, Volume 16, No. 1*, http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.html.

[8] *Mininet Python API Reference Manual*, http://mininet.org/api/index.html.

[9] BROCADE, *Multiple controller connections*, http://www.brocade.com/content/html/en/configuration-guide/netiron-05900-sdnguide/GUID-A2DC26EE-4CD8-440A-9735-9F88C6408CA2.html.

[10] sdxcentral, *What are SDN Controllers (or SDN Controllers Platforms)?*, https://www.sdxcentral.com/sdn/definitions/sdn-controllers/.

[11] SDN Hub, *OpenFlow version 1.3 tutorial*, http://sdnhub.org/tutorials/openflow-1-3/.

[12] *Ryu 4.14 documentation*, http://ryu.readthedocs.io/en/latest/getting_started.html.