ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA - SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA INFORMATICA E DELLE TELECOMUNICAZIONI

PROBLEMATICHE DI SICUREZZA NELLE SOFTWARE DEFINED NETWORKS

Elaborato in

Reti di Calcolatori

Relatore

Prof. FRANCO CALLEGATI

Presentato da

GIULIO CRESTANI

Correlatore

Ing. CHIARA CONTOLI

SESSIONE II

ANNO ACCADEMICO 2013/2014

Alla mia famiglia,
ai miei amici
ed in particolare a
Chiara, Marta e Caterina.

PAROLE CHIAVE

SDN

OpenFlow

Mininet

Security

Indice

Introduzione

Questa tesi ha l'obiettivo di comprendere e valutare se l'approccio al paradigma SDN, che verrà spiegato nel Capitolo 1, può essere utilizzato efficacemente per implementare dei sistemi atti alla protezione e alla sicurezza di una rete più o meno estesa. Oltre ad introdurre il paradigma SDN con i relativi componenti basilari, si introduce il protocollo fondamentale OpenFlow, per la gestione dei vari componenti.

Per ottenere l'obiettivo prestabilito, si sono seguiti alcuni passaggi preliminari. Primo tra tutti si è studiato cos'è l'SDN. Esso introduce una potenziale innovazione nell'utilizzo della rete. La combinazione tra la visione globale di tutta la rete e la programmabilità di essa, rende la gestione del traffico di rete un processo abbastanza complicato in termini di livello applicativo, ma con un risultato alquanto performante in termini di flessibilità. Le alterazioni all'architettura di rete introdotte da SDN devono essere valutate per garantire che la sicurezza di rete sia mantenuta. Le Software Defined Network (come vedremo nei primi capitoli) sono in grado di interagire attraverso tutti i livelli del modello ISO/OSI[1] e questa loro caratteristica può creare problemi. Nelle reti odierne, quando si agisce in un ambiente "confinato", è facile sia prevedere cosa potrebbe accadere, che riuscire a tracciare gli eventi meno facilmente rilevabili. Invece, quando si gestiscono più livelli, la situazione diventa molto più complessa perché si hanno più fattori da gestire, la variabilità dei casi possibili aumenta fortemente e diventa più complicato anche distinguere i casi leciti da quelli illeciti. Sulla base di queste complicazioni, ci si è chiesto se SDN abbia delle problematiche di sicurezza e come potrebbe essere usato per la sicurezza. Per rispondere a questo interrogativo si è fatta una revisione della letteratura a riguardo, indicando, nel Capitolo 3, alcune delle soluzioni che sono state studiate. Successivamente si sono chiariti gli strumenti che vengono

[1] *http://it.wikipedia.org/wiki/Open_ Systems_ Interconnection*

utilizzati per la creazione e la gestione di queste reti (Capitolo 4) ed infine (Capitolo 5) si è provato ad implementare un caso di studio per capire quali sono i problemi da affrontare a livello pratico.

Successivamente verranno descritti tutti i passaggi individuati in maniera dettagliata ed alla fine si terranno alcune conclusioni sulla base dell'esperienza svolta.

Capitolo 1

Introduzione alle Software Defined Network

Fino a qualche anno fa, servizi di archiviazione, elaborazione e risorse di rete, sono stati volutamente separati sia sul piano fisico sia sul piano funzionale. Anche i sistemi utilizzati per gestire questi servizi e queste risorse sono stati divisi tra loro, spesso fisicamente. Applicazioni interagenti con qualunque di questi servizi, come ad esempio un sistema di monitoring, sono state mantenute a debita distanza grazie anche alle politiche d'accesso; tutto questo in nome della sicurezza. È stato solo dopo l'introduzione e l'aumento sostanziale di: potenza di calcolo a basso costo (con la riduzione dello spazio occupato dal sistema fisico cercando di mantenere il sistema più performante possibile), capacità di archiviazione e complessità delle network nei data center, che le organizzazioni sono state costrette ad unificare questi diversi servizi in un unico sistema. Tale cambiamento ha portato tutte le applicazioni, che prima gestivano le singole tipologie di risorse, ad avere un comportamento molto simile tra loro.

I data center erano originariamente progettati in modo da separare fisicamente i tradizionali elementi di computazione (come ad esempio i PC server) dai loro sistemi di archiviazione dei dati e dalle reti che connettono i vari client. La potenza di calcolo che esisteva in questi data center venne concentrata su specifiche funzionalità del server, eseguendo applicazioni come server di posta, di database o altre funzionalità usate per gestire le richieste dei client. In precedenza, le funzionalità venivano offerte dai vari server dipartimentali e, prima ancora, erano gestite da numerosi PC desktop (all'interno, ad esempio, di

un'azienda). Con il passare del tempo, però, i server dipartimentali migrarono verso un unico data center per due motivi: il primo per facilitare la gestione dei dati e la gestione della rete, il secondo per consentire la condivisione tra gli utenti di una stessa organizzazione o di una stessa azienda.

È stato circa 10 anni fa[3] che ha avuto luogo una interessante trasformazione. La società chiamata VMware[1] aveva inventato un'interessante tecnologia che ha permesso ad un sistema operativo (ad esempio Linux) di eseguirne uno o più all'interno di esso (come ad esempio Windows). Quello che VMware ha fatto, non è stato altro che generare un programma tale da creare un ambiente virtuale dove poter installare un vero e proprio sistema operativo, con proprie capacità di calcolo e possibilità di archiviare dati, con la propria scheda di rete, audio e video.

Con l'avvento della virtualizzazione, i server, che di solito gestivano un solo sistema operativo dedicato con le relative applicazioni appositamente sviluppate, sono visti come delle piattaforme di calcolo e di archiviazione di massa. I server sono sempre più in grado di eseguire una moltitudine eterogenea di sistemi operativi in contemporanea, grazie ai continui progressi nelle prestazioni e nelle capacità di archiviazione. VMware ha ampliato la propria visione; da quella di singolo host, ad una più orientata alla centralizzazione dei dati, che è in grado di eseguire e controllare centinaia di macchine virtuali da una singola macchina. In ognuno di questi ambienti virtuali, è possibile salvare il relativo stato di esecuzione, che può essere messo in pausa, trasferito, clonato o salvato in una copia di backup. Inizia così l'era di quello che viene chiamato elastic computing.

Grazie a questa flessibilità si sono creati data center con numerosi sistemi operativi funzionanti. Questo, naturalmente, ha generato un nuovo interrogativo: come è possibile separare le risorse fisiche del data center tra le migliaia di macchine virtuali che sfruttano il sistema? In questo caso, capacità di calcolo, archiviazione di dati e risorse di rete possono essere offerti "a fette" indipendenti ed isolate le une dalle altre. È fondamentale che esse siano mantenute separate; questo pone alcune sfide interessanti che non erano presenti nei data center esistenti prima dell'avvento della virtualizzazione. Si tenga presente che ogni macchina virtuale necessita di un indirizzo di rete univoco, al quale il proprietario e gli utenti esterni possono accedere.

Oltre a queste sfide, ci si è trovati, nell'ultimo decennio, a non avere una sostanziale diminuzione del costo degli apparati di rete intelligenti a causa

[1] `http://www.vmware.com/it`

dell'aumento della complessità del piano di controllo; i dispositivi devono supportare molteplici tecnologie che aumentando rapidamente di numero, ne garantiscono in molti casi la coesistenza.

Intorno al 2008[1] nelle università di Berkeley e Stanford, queste considerazioni portarono alla nascita della visione della ***Software Defined Network (SDN)***, una rete composta da apparati di rete controllati da un controller centrale, e quindi, alla conseguente proposta di un protocollo di comunicazione standard tra il controller e gli apparati: OpenFlow. Oggi, con le dovute differenze, molti vendor hanno deciso di supportare e promuovere l'approccio SDN.

1.1 Definizione

Il Software Defined Networking è un nuovo modo di concepire le reti, verso il quale si sta concentrando un interesse senza precedenti nella storia di Internet. Esso infatti, permette di suddividere il piano di trasporto dei dati da quello di controllo, definito via software. Permette agli operatori di rendere le proprie reti facilmente gestibili, riprogrammabili e personalizzabili, oltre a liberarli dal cosiddetto "vendor lock-in" (ossia la situazione di dipendenza da un unico fornitore). Per tale motivo gli operatori, i produttori di apparati di rete e la comunità accademica stanno dedicando un'attenzione sempre maggiore a questo argomento.

Secondo McKeown[6] l'obiettivo principale di SDN è ristrutturare l'architettura di networking, introducendo opportuni livelli di astrazione in grado di operare una trasformazione simile a quanto già avvenuto nel campo delle architetture elaborative (Figura 1.2). Nell'ambito del computing, infatti, ormai da molto tempo i programmatori sono in grado di implementare sistemi complessi senza dovere gestire le tecnicità dei singoli dispositivi coinvolti o senza interagire in linguaggio macchina, il tutto grazie all'introduzione di opportuni livelli di astrazione nell'architettura (Figura 1.1).

Oltre a questo, una delle principali sfide tecnologiche del paradigma SDN riguarda la centralizzazione della logica del controllo. Ciò abiliterebbe le applicazioni ad acquisire una vista astratta della rete, come se questa fosse governata da un piano di controllo concettualmente centralizzato; diventa quindi possibile implementare logiche di controllo, astraendo dalla complessità fisica della molteplicità degli apparati di rete. Lo strato di controllo ha il compito

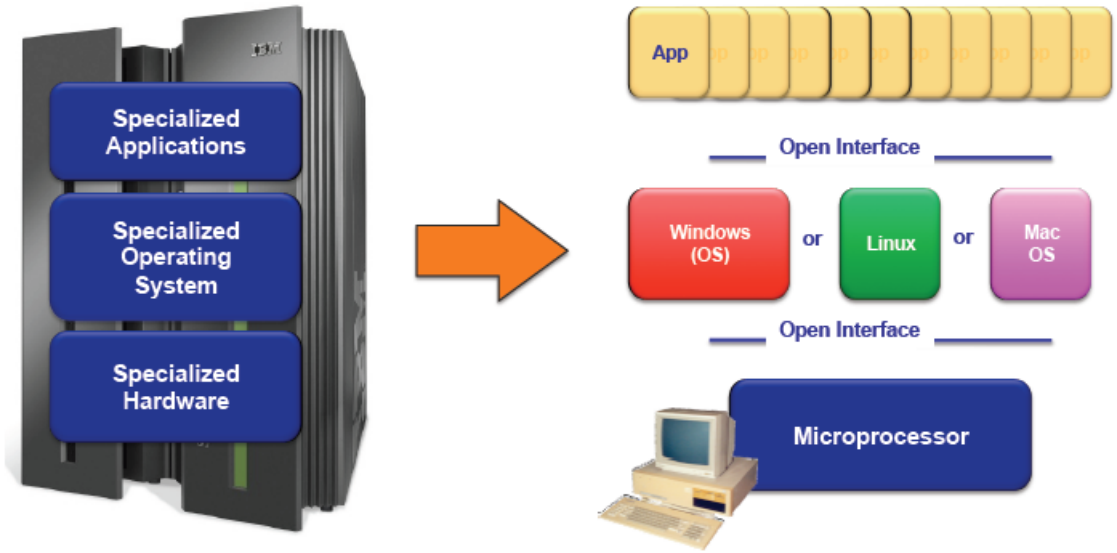


Figura 1.1: Evoluzione del Computing.

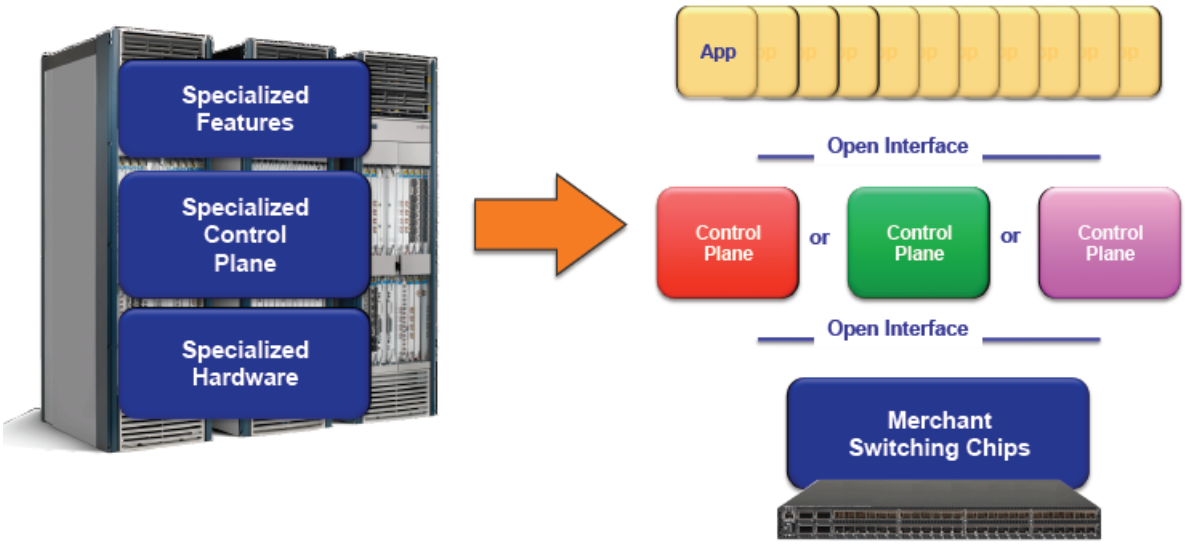


Figura 1.2: Evoluzione del Networking (?).

di presentare una vista unica, globale e logicamente centralizzata, gestendo la topologia fisica della rete e la distribuzione delle informazioni di stato necessarie ad implementare le logiche di servizio. Il cuore della SDN assomiglia dunque ad un ecosistema di moduli software di controllo, interagenti fra loro e capaci di attuare più facilmente azioni di configurazione e ottimizzazione delle risorse di rete. D'altra parte, la stessa centralizzazione logica del controllo potrebbe avere dei punti critici legati a: livelli di prestazioni, affidabilità, scalabilità e stabilità.

Essenzialmente si parte dall'idea di suddividere il piano di controllo e quello dei dati tra dispositivi differenti, in contrasto con quanto accade nelle reti attuali, dove un router si occupa di popolare delle strutture dati con algoritmi e protocolli, che verranno poi utilizzate per l'inoltro dei pacchetti dallo stesso router.

1.2 Architettura

SDN è un framework[2][2] che consente agli amministratori di rete di gestire e controllare automaticamente e dinamicamente un gran numero di dispositivi di rete, di servizi, di topologie, di percorsi di traffico e gestione dei pacchetti, utilizzando linguaggi di alto livello e le API. Questa gestione comprende fornitura, amministrazione, monitoraggio, ottimizzazione e gestione dei guasti in un ambiente multi-tenant[3].

1.2.1 Requisiti

Per progettare e creare una rete di questo tipo devono essere soddisfatti alcuni requisiti fondamentali:

Generalità il sistema deve supportare una vasta gamma di applicazioni per la gestione della rete;

Scalabilità non devono essere presenti limitazioni dovute alla piattaforma sulla quale si appoggia l'intero sistema;

[2]Architettura logica di supporto (spesso un'implementazione logica di un particolare design pattern) su cui un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore.

[3]`http://it.wikipedia.org/wiki/Multi-tenant`

Affidabilità gestione dei fallimenti ottimizzata per non recare danni al sistema;

Prestazioni garantire le performance sufficienti per poter mantenere stabilità.

1.2.2 Componenti fondamentali

L'architettura delle Software Defined Network prevede tre componenti:

Controller il suo ruolo è centrale all'interno dell'architettura; governa la rete controllando se i flussi possono essere immessi e quali devono essere le loro caratteristiche di trasporto. Gestisce le strutture dati, le tabelle di instradamento, le policy di firewalling, la lunghezza e gli algoritmi di gestione delle code. Fisicamente è un'applicazione software e si trova tra i dispositivi di rete e le applicazioni. Tutte le comunicazioni tra applicazioni e dispositivi devono passare attraverso il controller. Esso utilizza anche protocolli come OpenFlow per configurare i dispositivi di rete e scegliere il percorso ottimale per il traffico delle applicazioni. In effetti, il controller SDN funge come una sorta di sistema operativo di rete; prende il piano di controllo fuori dall'hardware di rete e lo esegue come software. In questo modo il controller facilita la gestione della rete automatizzata e rende più facile da integrare e gestire applicazioni di business. L'aspetto più interessante e innovativo sta nel fatto che con questo disaccoppiamento, le applicazioni possono vedere la rete come un'entità singola con la quale si può comunicare con delle API indipendenti dall'hardware.

Switch costituito da una o più tabelle di flusso (Flow Table) e da una Group Table[4], che svolgono ricerche di pacchetti e forwarding e, inoltre, da un canale sicuro che connette lo switch ad un processo remoto (controller). Lo switch comunica con il controller e il controller gestisce lo switch permettendo ai comandi ed ai pacchetti di essere scambiati tramite il protocollo OpenFlow, che fornisce un'interfaccia standard di comunicazione (Figura 1.3).

Il controller, attraverso il protocollo OpenFlow, può aggiungere, rimuovere e aggiornare le entry delle flow table. Ognuna di queste nello switch, contiene un insieme di entry, usate per fare matching e per processare

[4]Un'istruzione può specificare un identificatore di gruppo. Questi gruppi forniscono un modo efficiente per imporre allo stesso insieme di azioni la loro esecuzione su più flussi. Le operazioni del gruppo sono definite all'interno dello switch da voci nella tabella del gruppo.

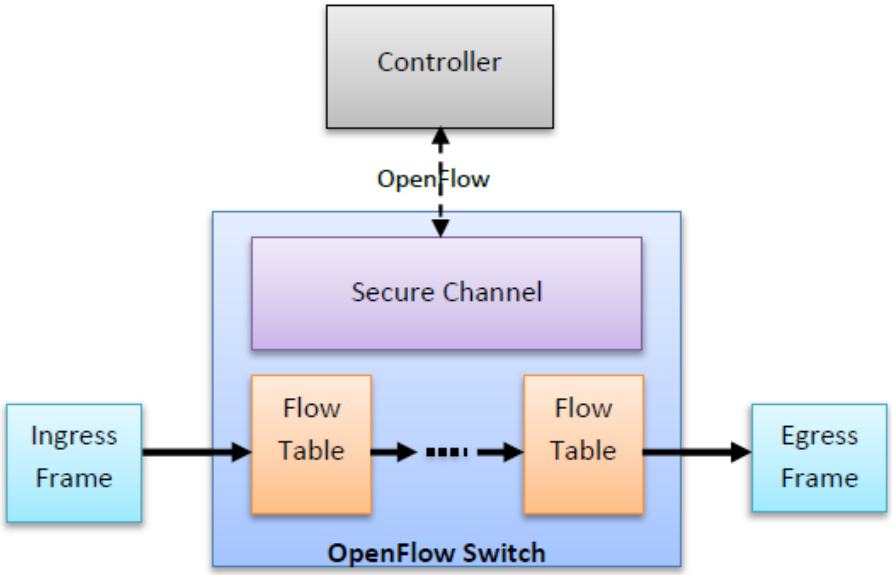


Figura 1.3: OpenSwitch.

i pacchetti che transitano nello switch. Le flow table sono costituite da campi di confronto, contatori e una serie di istruzioni da applicare ai pacchetti corrispondenti. Le azioni di base associate alle flow entry, che il nodo OpenFlow deve supportare, sono:

1. inoltrare il pacchetto verso una determinata porta di uscita; questo permette ai pacchetti di essere instradati attraverso la rete;
2. incapsulare il pacchetto e spedirlo al controller attraverso l'interfaccia di comunicazione. Solitamente questa azione viene applicata al primo pacchetto di un nuovo flusso; il controller decide poi se configurare una nuova entry nella tabella per specificare il trattamento dei pacchetti successivi. Infine nulla impedisce, in casi particolari, l'invio di tutti i pacchetti di un flusso verso il controller per la loro elaborazione;
3. scartare i pacchetti appartenenti al flusso; può essere usato per la sicurezza, per trattenere gli attacchi DoS o per ridurre il traffico non

legittimo di broadcast discovery[5] da parte degli host;

4. trattare i pacchetti secondo le normali procedure di forwarding del nodo.

Host sono dei normali terminali connessi alla rete; più precisamente uno switch è connesso a un certo numero di host e gestisce il traffico da e verso di essi (Figura 1.4).

In più, rispetto alle reti odierne, vi è la capacità di creare delle regole che agiscono tra tutti i livelli del modello ISO-OSI senza guardare più, in modo rigido, a queste separazioni. Quindi si potrebbe contestualmente dire al controller: "Inietta una certa regola nel mio apparato, una regola di instradamento che vale per un certo MAC con un certo IP su una certa porta".

Da diversi anni ormai il mercato è cambiato proponendo hardware, sistemi operativi e API in grado di produrre un mercato vario, concorrenziale e rapido nell'innovazione. SDN si propone lo stesso traguardo con protocolli aperti e fruibili come OpenFlow, disaccoppiando hardware e software, rendendo il piano di controllo ancora una volta una parte software indipendente e interfacciabile con qualunque hardware. In questa visione gli apparati di rete perdono il loro ruolo centrale, diventando meri esecutori delle indicazioni ricevute dal controller.

[5]Quando si utilizza il broadcast discovery, il server di gestione tenta di scoprire oggetti gestibili in rete trasmettendo una singola richiesta a tutta la sottorete su cui si trova il server.

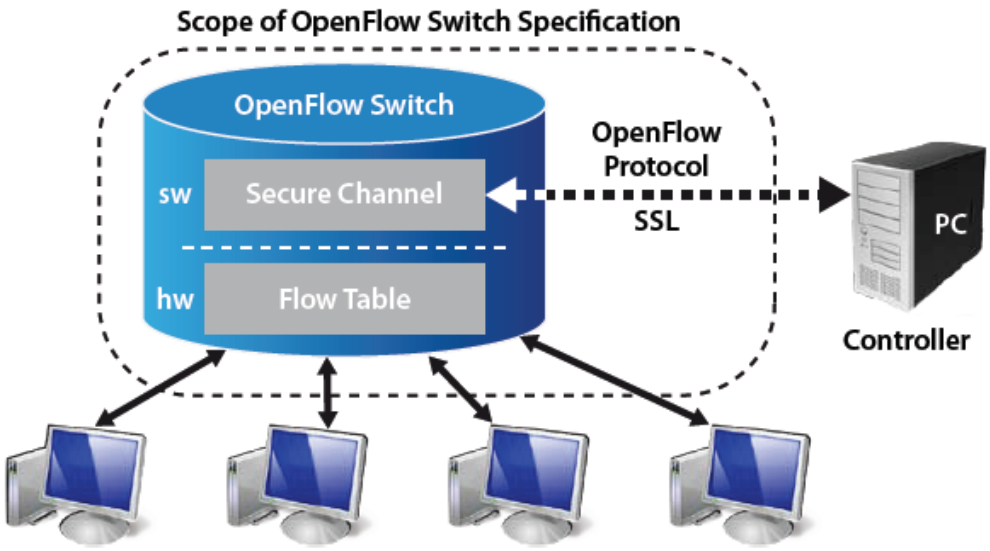


Figura 1.4: Host connessi ad uno Switch OpenFlow.

Capitolo 2

Il protocollo OpenFlow

Si ritiene[4] che il protocollo OpenFlow rappresenti un fattore per realizzare la trasformazione verso i concetti di rete flessibile e programmabile, anche se da solo ovviamente non sufficiente. L'interfaccia realizzata da OpenFlow si colloca al livello più basso di astrazione previsto dall'architettura SDN; essa permette infatti di svincolarsi dall'hardware di inoltro dei pacchetti. Uno degli aspetti fondamentali che sta alla base dell'attività di specifica avviata da OpenFlow, consiste nella definizione di un modello standard dell'hardware di inoltro dei pacchetti che costituisce il nucleo dei diversi dispositivi di networking. Scopo del protocollo OpenFlow è, quindi, quello di presentare all'esterno un modello di nodo generale e unificato, rendendo gli strati più alti dell'architettura di rete SDN, indipendenti dall'implementazione del particolare vendor delle tecnologie impiegate nel piano di forwarding.

Il suo obiettivo è quello di "aprire" il controllo dei nodi di rete per permettere la separazione tra *piano di forwarding* (cioè l'inoltro veloce dei pacchetti), e *piano di controllo* (cioè il livello delle decisioni di instradamento). Ciò garantisce la possibilità di sviluppare nuove funzionalità di rete in un controller esterno e logicamente centralizzato che può essere personalizzato secondo le esigenze del gestore di rete, slegandole dai vincoli imposti dai vendor i quali, generalmente, implementano i propri protocolli in forma proprietaria.

Risalendo alle origini della proposta, l'idea di base di OpenFlow è quella di rendere programmabili in senso generale, le tabelle di classificazione ed instradamento dei pacchetti presenti negli apparati di networking (siano essi router o switch)[1]; in questo modo il contenuto (le cosiddette entry) può essere

[1]Uno switch è un apparato di livello due (della pila ISO/OSI) che permette di fare

configurato dalle applicazioni attraverso un piano di controllo esterno al dispositivo, mediante un'opportuna interfaccia. Quest'ultima, costituita appunto dal protocollo OpenFlow, permette di definirne in modo flessibile il contenuto, in funzione della logica di servizio da realizzare.

Le tabelle utilizzate per la classificazione del traffico all'interno dei router sono generalmente in grado di operare in tempo reale e vengono sfruttate anche per realizzare funzioni aggiuntive al forwarding di base, per esempio: firewall, NAT, QoS. Nel modello del nodo OpenFlow queste tabelle vengono denominate *flow table* e specificano le regole di trattamento associate a ciascun flusso di traffico. L'entità base con cui viene rappresentato e gestito il traffico in OpenFlow è per l'appunto il **flusso** di pacchetti (*flow*); quest'ultimo è definito da una regola di classificazione ottenuta specificando il contenuto di opportuni campi dell'intestazione tramite una entry della flow table. Il protocollo OpenFlow permette quindi al piano di controllo di definire in modo flessibile e dinamico le regole di instradamento e di trattamento dei pacchetti appartenenti ai diversi flussi di traffico.

Normalmente l'implementazione delle tabelle di classificazione dei pacchetti, che gestiscono la definizione dei flussi e le relative regole di inoltro, è una caratteristica proprietaria del particolare apparato di networking. Per superare questo modello, OpenFlow mira ad individuare, specificare e rendere accessibili attraverso il protocollo, un insieme di funzioni supportate dalla maggior parte dei router o switch commerciali. L'obiettivo principale consiste quindi nel definire un modello astratto dell'elemento che esegue il forwarding dei pacchetti, rendendolo programmabile attraverso un'interfaccia aperta e standard. Un amministratore di rete può suddividere il traffico in flussi di produzione e flussi sperimentali. I ricercatori possono controllare i loro flussi scegliendo quali percorsi devono seguire i pacchetti e quale trattamento debbano ricevere. In questo modo possono essere testati nuovi protocolli di instradamento, schemi di indirizzamento innovativi e modelli di sicurezza.

---

commutazione dei frame, ossia di inviarli con consegna diretta verso le destinazioni collegate direttamente ad una sottorete fisica di cui lui stesso fa parte usando l'indirizzo MAC. Un router è invece un apparato di livello 3 che permette principalmente di fare instradamento dei pacchetti, ossia di decidere da quale porta far uscire un pacchetto che gli è arrivato facendolo arrivare a destinazione in base all'indirizzo IP di destinazione.

2.1 Esempi di utilizzo del protocollo OpenFlow

2.1.1 Esempio 1

Come semplice esempio[5] si consideri il caso in cui si voglia testare un nuovo protocollo di instradamento in una topologia di rete composta da end-host e da switch OpenFlow. L'esempio verrà eseguito all'interno di un PC desktop che funge da controller, in modo tale da evitare di interferire nella rete. Ogni volta che un nuovo flusso applicativo viene generato, il protocollo che si vuole testare sceglie un percorso attraverso una serie di switch OpenFlow e aggiunge una flow entry in ognuno lungo il percorso. Per il traffico in entrata nella rete OpenFlow si definisce un flusso in modo che tutto il traffico entrante nello switch passi attraverso la porta connessa al PC e si aggiunge una entry con l'azione incapsula e spedisci tutti i pacchetti al controller. Quando tutti i pacchetti arrivano al controller, il protocollo sceglie un percorso e aggiunge una nuova flow entry per ogni switch lungo tutta la traiettoria. Quando i pacchetti successivi giungono nello switch, vengono processati velocemente attraverso le flow table (essendoci già presenti le regole per gestirli).

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

- Può un controller centralizzato essere abbastanza veloce per elaborare nuovi flussi e programmare gli interruttori di flusso?

- Cosa succede quando un controller si guasta?

Per rispondere a queste domande, in letteratura si è testata una semplice topologia che ha utilizzato dei semplici switch ed un controller centrale. I risultati preliminari mostrano che un controller come questo è in grado di elaborare oltre 10.000 nuovi flussi al secondo, sufficienti per un grande campus universitario. Naturalmente, il tasso al quale i nuovi flussi possono essere elaborati dipende dalla complessità della computazione richiesta dagli esperimenti del ricercatore.

Se ci spostassimo in una rete di più larga scala, nella quale il protocollo che si vuole testare viene testato in una rete utilizzata da molte altre persone, sarebbe desiderabile che la rete avesse due caratteristiche aggiuntive:

1. Pacchetti che appartengono a utenti dovrebbero essere instradati tramite un protocollo standard di instradamento eseguito nello switch o nel router;

2. Colui che gestisce la rete dovrebbe essere in grado di aggiungere voci di flusso per il suo traffico, in modo tale da consentirgli di controllare i flussi e l'integrità della rete.

La prima caratteristica si ottiene dall'abilitazione degli switch OpenFlow; è possibile installare al loro interno delle regole che permettano la gestione di determinati pacchetti appartenenti ad un singolo utente. La seconda invece dipende dal controller; esso dovrebbe essere visto come una piattaforma che permette ai ricercatori di implementare i loro esperimenti. Le restrizioni della seconda caratteristica possono essere ottenute con un appropriato uso dei permessi, limitando il potere degli altri utenti sul controllo delle entry delle flow table.

2.1.2 Esempio 2: VLAN

OpenFlow può facilmente fornire un aiuto agli utenti con la propria rete isolata, proprio come fanno le VLAN[2]. L'approccio più semplice è dichiarare un insieme di flussi che specifica le porte accessibili dal traffico su un determinato ID VLAN. Il traffico identificato come proveniente da un singolo utente (ad esempio, provenienti da porte specifiche o indirizzi MAC) viene etichettato dagli switch (tramite un'azione) con l'ID VLAN appropriato. Quindi quando l'utente genera del traffico, questo traffico ha l'ID della VLAN alla quale l'utente appartiene. Un approccio più dinamico potrebbe utilizzare un controller per gestire l'autenticazione degli utenti e utilizzare la conoscenza delle posizioni degli utenti per la codifica del traffico in fase di esecuzione.

2.1.3 Esempio 3: Mobile wireless VOIP

[3]Per questo esempio si consideri un esperimento di un nuovo meccanismo di chiamata tramite smartphone attraverso la tecnologia Wi-Fi. I client stabilisco-

[2]Virtual LAN: indica un insieme di tecnologie che permettono di segmentare il dominio di broadcast che si crea in una rete locale basata su Switch, in più reti locali logicamente non comunicanti tra loro, ma che condividono globalmente la stessa infrastruttura fisica di rete locale. Le applicazioni di questa tecnologia sono tipicamente legate ad esigenze di separare il traffico in gruppi di lavoro o dipartimenti di un'azienda, per applicare diverse politiche di sicurezza informatica. Fonte: `http://it.wikipedia.org/wiki/VLAN`

[3]Si intende una tecnologia che rende possibile effettuare una conversazione telefonica sfruttando una connessione Internet o una qualsiasi altra rete dedicata a commutazione di pacchetto che utilizzi il protocollo IP senza connessione per il trasporto dati.

no una nuova connessione tramite una rete di switch OpenFlow. Un controller è implementato per tenere traccia della posizione dei clienti, delle connessioni e del re-instradamento, riprogrammando le tabelle di flusso, in base a come gli utenti si muovono attraverso la rete, consentendo il trasferimento da un punto di accesso all'altro. Se un client che sta eseguendo una chiamata cambia punto d'accesso, il controller se ne accorge e riconfigura le flow table in modo da consentire la continuazione della chiamata a fronte di questo cambio.

2.2 Sviluppi recenti

La definizione dell'architettura OpenFlow ha subito delle evoluzioni rispetto alla versione iniziale, da quando è stata fatta oggetto di studio da parte di ONF[4] ed ha visto, parallelamente, un sempre maggiore coinvolgimento da parte dell'industria. A differenza della semplice struttura discussa in precedenza a titolo esemplificativo, il modello di nodo OF, attualmente definito dalla versione più recente della specifica (Versione 1.3.4[5]), prevede la presenza di una sequenza di flow table in cascata al fine di consentire una maggiore flessibilità nel trattamento dei pacchetti (vedi figura 1: switch).

Infine è bene notare che, mentre l'introduzione di un livello di interfacciamento aperto verso i dispositivi di forwarding rimane uno dei capisaldi dell'architettura SDN, comincia ad emergere, all'interno della comunità scientifica e industriale che lavora alla definizione di OpenFlow, l'idea che l'astrazione supportata dalle versioni di OpenFlow, attualmente in corso di specifica, sia soggetta a limiti che possono ostacolare la piena applicabilità della tecnologia. Il principale limite identificato all'interno della stessa ONF, per il modello corrente, consiste nel fatto che la rappresentazione semplificata su cui si basa attualmente OpenFlow, non sia in grado di gestire agevolmente le informazioni sulla logica di forwarding che l'applicazione intende implementare, rendendo difficile, se non talora impossibile, fare leva sulle funzionalità messe a disposizione dall'hardware.

[4]Open Networking Foundation: `https://www.opennetworking.org/`

[5]`https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf`

Capitolo 3

Problemi relativi alla sicurezza

I dispositivi maggiormente utilizzati all'interno delle reti moderne sono i middlebox, cioè dispositivi che manipolano il traffico per fini diversi dall'inoltro dei pacchetti; per esempio i firewall che filtrano traffici inaspettati o dolosi. I middlebox sono utilizzati per fornire le funzioni di sicurezza di rete, ma rendono difficile il rispetto di politiche a livello di rete. Questo perché quando i pacchetti attraversano la rete, le intestazioni e i contenuti possono essere modificati dinamicamente da altri middlebox; ad esempio, i NAT[1] e i load balancer[2] riscrivono le intestazioni. In un contesto SDN, queste modifiche rendono difficile (se non impossibile) garantire che il set di politiche desiderato venga conseguentemente applicato in tutta la rete. Ciò è particolarmente impegnativo perché i middlebox spesso si basano su una logica interna (di proprietà dell'azienda costruttrice) per effettuare le suddette trasformazioni dinamiche del traffico. A tal fine, si identifica il monitoraggio del flusso come una capacità fondamentale per l'applicazione delle policy in presenza di trasformazioni dinamiche del traffico. Cioè, si devono "allegare" delle informazioni aggiuntive ad un flusso di traffico mentre viaggia in rete (come ad esempio, salvare l'utente che ha avviato il pacchetto), anche se le intestazioni dei pacchetti ed i contenuti vengono modificati.

[1]Nel campo delle reti telematiche, il network address translation o NAT, 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.(`http://it.wikipedia.org/wiki/Network\_address\_translation`)

[2]Il load balancing, in italiano bilanciamento del carico, è una tecnica informatica utilizzata nell'ambito dei sistemi informatici che consiste nel distribuire il carico di elaborazione di uno specifico servizio, ad esempio la fornitura di un sito web, tra più server (`http://it.wikipedia.org/wiki/Load\_balancing`)

Sulla base di questa intuizione, si è esteso il paradigma SDN con l'architettura *FlowTags*.

Ritornando al monitoraggio della rete, la sicurezza avviene in due fasi: protezione attiva (firewall, per esempio) ed, appunto, il monitoraggio della rete. Gli amministratori di oggi sono molto interessati al monitoraggio del traffico allo scopo di raccogliere dati statistici, di rilevare delle intrusioni e di fornire prove forensi[3]. In genere, questo controllo è svolto sia tramite l'uso di middlebox collocati direttamente sul percorso dei dati, sia attraverso l'ispezione di copie di traffico nei punti interessanti della rete. Purtroppo, le dimensioni della rete e la sua complessità, possono rendere questo un compito scoraggiante. A parte i problemi di analisi del traffico di rete per queste informazioni (un compito già estremamente difficile in sé), esiste un problema prioritario: come instradare il traffico per l'analisi di rete in modo robusto, altamente performante e che non incida sul traffico normale di rete? Per risolvere questo problema si può pensare di utilizzare il sistema *OpenSAFE*.

Cambiando argomento, ma rimanendo sempre nell'ambito del monitoraggio della rete, passiamo ad analizzare un paradigma che negli ultimi anni, soprattutto con l'avvento del web, ha preso sempre più piede: il cloud computing[4]. Le principali caratteristiche del cloud computing possono essere riassunte come segue:

- si tratta di un ambiente di grandi dimensioni, costituito da molti host fisici e macchine virtuali (VM);

- la configurazione di un ambiente di cloud computing è piuttosto complicata. Per gestire una rete cloud, si dovrebbe considerare il gran numero di diverse macchine (fisiche o virtuali) collegate in rete e il gran numero di diversi consumatori che possono richiedere configurazioni di rete molto diverse.

- è una rete abbastanza dinamica. Una delle funzioni interessanti del cloud computing è il servizio on-demand; questo significa che se un determinato servizio è massicciamente richiesto, un ambiente di cloud computing dovrà

[3] `http://it.wikipedia.org/wiki/Informatica\_forense`

[4] Il cloud computing è un modello per consentire ubiquità, convenienza (se uso le risorse del cloud computing ho un vantaggio dal punto organizzativo ed economico), l'accesso alla rete on-demand ad un insieme condiviso di risorse di calcolo configurabili(dinamicamente) (ad esempio, reti, server, storage, applicazioni e servizi) che possono essere rapidamente fornite e rilasciate con il minimo sforzo di gestione o interazione del fornitore di servizi.

eseguire più macchine virtuali per quel servizio in quel momento. Così, le macchine virtuali in un host fisico possono essere invocate o rimosse in modo dinamico, e possono anche essere migrate ad altri host fisici.

Dal momento che molti host diversi e configurazioni di rete coesistono in una rete cloud, è essenziale proteggere ciascuno di essi dalle minacce. Per fare ciò, esistono già tecniche in grado di impiegare dispositivi di sicurezza di rete ma, applicandole ad una rete cloud, si richiedono ulteriori considerazioni per la sua complessità, il suo dinamismo e la sua diversità. Quindi sorge spontanea la seguente domanda: "È possibile applicare i dispositivi correnti di sicurezza ad un ambiente cloud?". Sì, è possibile, ma date le caratteristiche del cloud computing appena citate, ci sono diversi problemi che non possono essere ignorati quando si distribuiscono i vari dispositivi di sicurezza e quando si fornisce un servizio di monitoraggio nella rete cloud. Poiché una rete cloud è abbastanza complicata da gestire e difficile da riconfigurare, si dovrebbe studiare attentamente il luogo più appropriato dove installare questi dispositivi. Altrimenti potrebbe essere necessario riconfigurarli o spostarli di frequente con conseguente dispendio di risorse. Inoltre, bisogna implementare dispositivi di sicurezza di rete tenendo presente il dinamismo del cloud computing. Si consideri il caso dell'installazione di un NIDS (Network Intrusion Detection System) su un collegamento tra l'host A e l'host B, lasciando che il sistema di rilevamento del traffico di rete prodotto da una macchina virtuale sia in esecuzione sull'host A. Se le macchine virtuali sull'host A migrassero su un altro host C, si avrebbe la necessità di trasferire il sistema di rilevamento come collegamento tra host A e l'host C. Questo tipo di migrazione delle macchine virtuali è abbastanza frequente nel cloud computing. Le reti SDN forniscono una modalità di controllo dei flussi di rete che può essere adattata a questa necessità. Con l'aiuto di questa tecnologia, *CloudWatcher* cambia i percorsi di instradamento per i flussi di rete e fa sì che questi vengano trasmessi attraverso i nodi della rete in cui i dispositivi di sicurezza risiedono.

Infine si cambia prospettiva: si prende in considerazione la sicurezza all'interno di un'azienda. Al giorno d'oggi, la sicurezza all'interno di una rete aziendale è tipicamente reattiva e si basa molto sulla protezione degli host e sull'utilizzo dei middlebox. Gli amministratori utilizzano tecniche ad hoc con l'ausilio di IDS e con un insieme di configurazioni di rete complesse. Ma, nonostante i significativi progressi nella protezione degli host, il crescente numero e le svariate tipologie di dispositivi di rete, che vanno dai PC desktop ai laptop ai palmari, rendono sempre più difficile mettere in sicurezza ogni dispositivo che si

connette alla rete. Questi dispositivi eseguono una varietà di sistemi operativi e sono soggetti ad una serie diversificata di vulnerabilità, perciò si vengono a creare interazioni complesse tra i numerosi protocolli di rete e tra i vari sistemi, che possono provocare risposte lente agli attacchi rendendo il sistema molto vulnerabile. Di fronte a questi problemi, la rete deve autenticare questi nuovi dispositivi quando entrano in essa e deve monitorare il loro comportamento per rilevare le violazioni alle diverse politiche di sicurezza (ad esempio, la presenza di host non autorizzati o compromessi). Per far fronte a ciò si deve introdurre nel livello di rete una modalità di controllo dinamico degli accessi. Invece di riporre fiducia negli end-host o fare affidamento su middlebox di sicurezza, una rete deve offrire meccanismi che controllino direttamente il traffico, in base alle politiche di sicurezza e in risposta all'input di sistemi di monitoring di rete distribuiti. Per fare ciò, *Resonance* utilizza il paradigma SDN il quale permette l'integrazione dei due aspetti fondamentali di questo sistema: il monitoraggio di rete ed il controllo dinamico. Gli switch di rete programmabili permettono un controllo della rete più diretto rispetto ai dispositivi presenti oggigiorno; essi riescono a manipolare il traffico a livelli più bassi. Questi switch utilizzano le azioni (come ad esempio, il drop di un pacchetto, o la ridirezione del traffico) per applicare delle politiche di sicurezza sulla base delle informazioni che giungono dai sistemi di monitoring della rete. Oltre a questo, gli algoritmi distribuiti di monitoring sono in grado di correlare velocemente e accuratamente il traffico da sorgenti distinte per rilevare attacchi coordinati (ad esempio per rilevare botnet[5]).

3.1 FlowTags

L'architettura *FlowTags*[13] propone l'uso di middlebox leggermente modificati che interagiscono con il controller SDN attraverso delle API FlowTags. Per affrontare il problema del rispetto delle politiche a livello di rete descritto nell'introduzione, si cerca di integrarli nella struttura delle reti SDN in modo minimamente invasivo invece di eliminarli o riprogettarli completamente. Questi middlebox sono in grado di aggiungere ***tag*** (etichette) all'interno delle intestazioni dei pacchetti in uscita per fornire un contesto nel quale il pacchetto si trova o delle informazioni. Switch SDN e gli altri middlebox utilizzano le

[5]Rete formata da dispositivi informatici collegati ad Internet e infettati da malware, controllata da un'unica entità.

informazioni di questi tag per prendere decisioni riguardo alle operazioni di instradamento e di elaborazione dei pacchetti o per l'introduzione di nuove politiche di gestione della rete.

Per l'implementazione di FlowTags si è dovuta creare una nuova interfaccia "southbound"[6] tra controller e middlebox per permettere ai controller di configurare la capacità di "taggare" i vari flussi e per il supporto necessario per implementare le funzioni di FlowTags correlate. Oltre ad applicare correttamente le nuove politiche, FlowTags può anche fornire nuove funzionalità per verificare che queste politiche siano attuate correttamente.

3.1.1 Architettura di FlowTags

Nella Figura 3.1 viene mostrata un'immagine riassuntiva dove è possibile osservare dove vengono aggiunte le nuove funzionalità (colorate di rosso). SDN fornisce un'interfaccia tra il controller e gli switch per controllare il comportamento di inoltro dei pacchetti. FlowTags estende questa architettura lungo tre dimensioni chiave (evidenziate in Figura 3.1):

1. Il potenziamento FlowTags dei middlebox che tengono conto dei tag dei pacchetti in ingresso mentre li processano e che possono anche aggiungere nuovi tag basati sul contesto. Gli switch usano i tag per pilotare i pacchetti;

2. Nuove API FlowTags tra controller e middlebox con FlowTags;

3. Nuove applicazioni di controllo che consentono di configurare il comportamento di tagging dei middlebox e degli switch, e che usano i tag per sostenere l'applicazione delle policy e la verifica.

Nella progettazione FlowTags, si impongono tre vincoli fondamentali:

1. Richiedere modifiche minime al middlebox, al fine di stimolare l'adozione tra i fornitori di middlebox;

[6] In SDN l'interfaccia southbound[14] è il protocollo OpenFlow. La sua funzione principale è abilitare la comunicazione tra il controller e i nodi della rete in modo che il router possa scoprire la topologia di rete, definire i flussi della rete e soddisfare le richieste inoltrate ad esso tramite le API northbound. L'interfaccia northbound descrive, invece, l'area di protocolli di comunicazione supportati tra il controller e le applicazioni o programmi di controllo di più alto livello

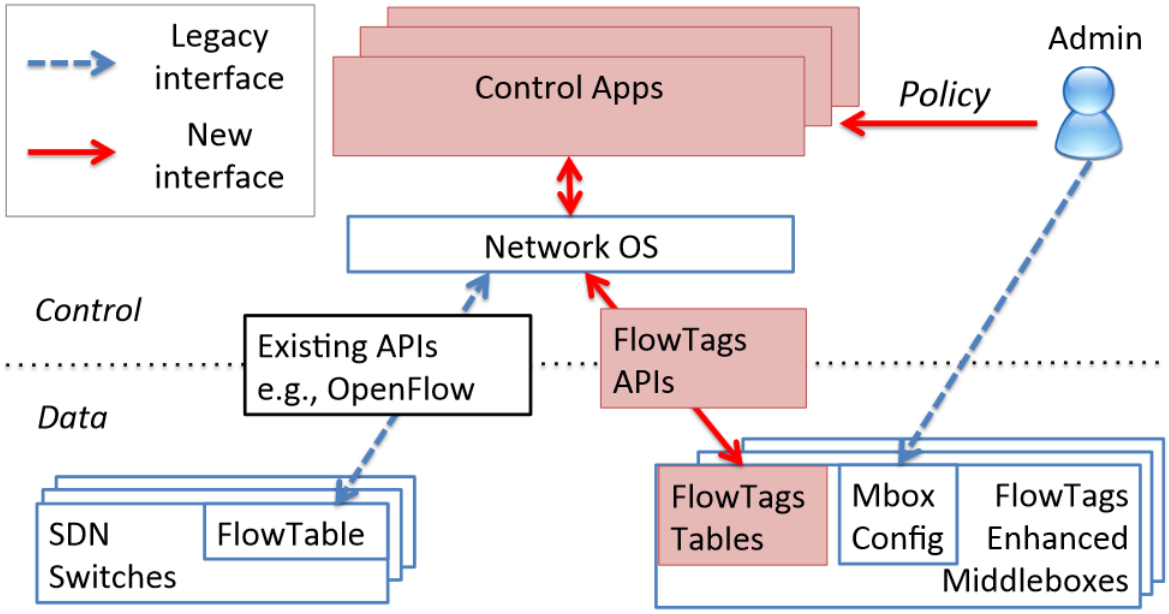


Figura 3.1: Architettura di FlowTags.

2. Mantenere esistenti gli switch e l'interfaccia tra switch e controller (ad esempio, OpenFlow), per mantenere la compatibilità con i fornitori di switch;

3. Evitare interazioni dirette tra i middlebox e switch per disaccoppiare i percorsi evolutivi di queste diverse classi di dispositivi.

Il controller SDN configura le azioni sugli switch e sui middlebox in modo da utilizzare i tag (aggiunti da altri middlebox) come parte della loro attività del piano di dati, al fine di applicare correttamente le politiche a livello di rete. Si noti che FlowTags non impone nuove funzionalità agli switch SDN né richiede la segnalazione diretta tra middlebox e switch; gli switch continuano a utilizzare le API SDN tradizionali.
Nel caso più semplice: quando uno switch vede un pacchetto con un tag per il quale non ha alcuna regola corrispondente, invia (come è tipico in OpenFlow) un messaggio di *packet-in* al controller, che a sua volta risponde con la regola appropriata. Allo stesso modo, quando un middlebox vede tale pacchetto, invia un messaggio analogo al controller per imparare la corretta azione FlowTag da utilizzare. In generale: le azioni che il controller invia ad un middlebox saranno diverse da quelle che invia ad uno switch. Per migliorare l'efficienza, in termini di spazio di tabella e di controllo del traffico, possiamo installare le azioni FlowTags in modo proattivo (anticipatamente, in modo da prevenire i problemi).

3.1.2 API southbound

Si definisce un'interfaccia tra il controller SDN e i middlebox FlowTags al fine di controllare il comportamento relativo ai tag. Si noti che i middlebox sono sia produttori (ad esempio, il NAT deve esporre le mappature degli host e i loro relativi IP pubblici), che consumatori (ad esempio, il rivelatore di scansione IDS deve utilizzare tag per attribuire il traffico agli host) di tag. In corrispondenza di questi due ruoli, si prevedono due tabelle di configurazione:

1. Analogamente alle flow table in OpenFlow, ogni middlebox ha una *TagsFlowTable* per abbinare i modelli di flusso ai tag;

2. Una *TagsActionTable* che associa un pacchetto con specifici tag ad azioni.

3.2 OpenSAFE

OpenSAFE[15] è un sistema per abilitare la direzione arbitraria del traffico per le applicazioni di monitoraggio di sicurezza a line rate. Inoltre, viene descritto *ALARMS*, un linguaggio specifico per la gestione dei flussi che semplifica notevolmente la gestione delle apparecchiature di monitoraggio della rete.

3.2.1 Copia del traffico

Nel monitoraggio di rete è tipico utilizzare le porte SPAN [7] per creare copie di traffico in vari punti di interesse della rete. In genere, i punti di interesse si possono trovare prima o dopo un firewall di rete. Questo permette copie complete di tutti i flussi di rete per essere osservati e controllati. Queste porte SPAN sono in genere dirette verso un singolo computer che esegue una sorta di IDS (Intrusion Detection System), come mostrato in Figura 3.2

Gestire più dispositivi connessi ad una SPAN è un compito difficile. O tutto il traffico viene inoltrato a tutti i dispositivi sulla SPAN, o particolare cura deve essere presa per dirigere i vari sottoinsiemi di traffico nei diversi dispositivi. Questo rende molto più difficile per gli amministratori aggiungere nuovi dispositivi di controllo che possono essere fastidiosi quando gli apparecchi più specializzati sono acquisiti o quando il carico diventa troppo grande per il monitoraggio hardware corrente.

3.2.2 Descrizione

OpenSAFE (Open Security Auditing e Flow Examination) è un sistema unificato per il monitoraggio della rete. Sfruttando la tecnologia di rete aperta come OpenFlow, OpenSAFE può dirigere il traffico di rete in modi arbitrari. OpenSAFE può gestire qualsiasi numero di ingressi di rete e gestire il traffico in modo tale che possa essere utilizzato da molti servizi filtrando pacchetti a line-rate.

OpenSAFE è costituito da tre componenti principali:

[7] Lo Switched Port Analyzer (SPAN) è una proprietà che permette di replicare interamente tutto il traffico generato da/verso una o più vlan/interfacce verso un'interfaccia fisica. Su tale porta di destinazione solitamente troveremo connesso un network analyzer, permettendo di conoscere meglio la rete oltre a creare statistiche e report di vario genere.

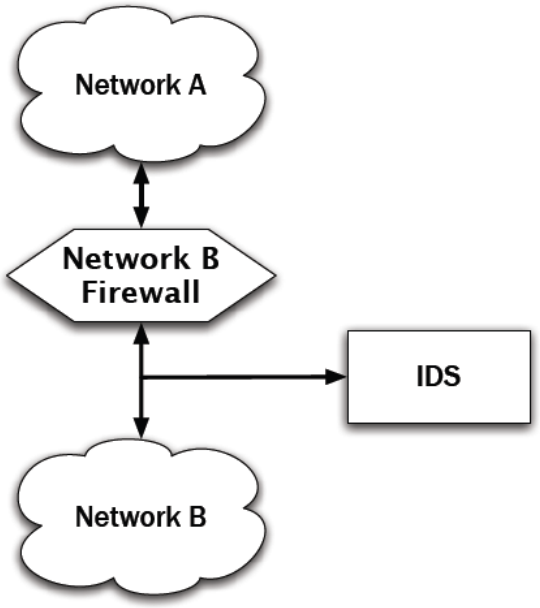


Figura 3.2: Tipica configurazione di network monitoring.

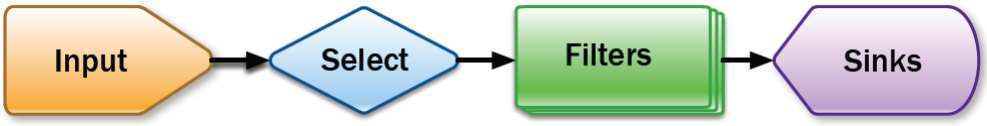


Figura 3.3: Varie parti di un percorso.

1. un insieme di astrazioni di progettazione per pensare al flusso del traffico di rete;
2. *ALARMS (A Language for Arbitrary Route Management for Security)*, un linguaggio di policy per specificare e gestire facilmente percorsi;
3. un componente OpenFlow che implementa la politica.

3.2.3 Design complessivo

Per rendere la gestione dei percorsi per il monitoraggio della rete facile e flessibile, OpenSAFE è stato progettato intorno a diverse primitive semplici. Usiamo la nozione di *paths* (percorsi) come astrazione di base per descrivere la selezione del traffico e del percorso che questo particolare traffico dovrebbe prendere. Fondamentalmente, si vogliono costruire percorsi che consentano al traffico desiderato di accedere al sistema ed essere indirizzato ad uno o più sistemi di monitoring.

In OpenSAFE, l'articolazione dei percorsi avviene in modo incrementale lungo l'itinerario desiderato del percorso. Come mostrato in Figura 3.3, i percorsi sono composti di più parti: ingressi, selezioni, filtri e sinks (letteralmente lavelli).

Ad alto livello, ogni percorso inizia con un ingresso. Si applica un criterio di selezione facoltativo e i percorsi che soddisfano il criterio di selezione passano attraverso zero o più filtri finendo in una o più sinks. Gli ingressi possono produrre solo il traffico, i sinks possono ricevere solo il traffico e i filtri devono fare entrambe le cose.
Un esempio di questo design è il seguente: del traffico HTTP viene fatto scorrere attraverso un dispositivo contatore ed infine viene applicato ad un dispositivo di TCP dump[8]. Se consideriamo la Figura 3.4 e la adattiamo al pattern sopra

[8]È un tool comune per il debug delle reti di computer. Consente all'utente di intercettare

Figura 3.4: Esempio.

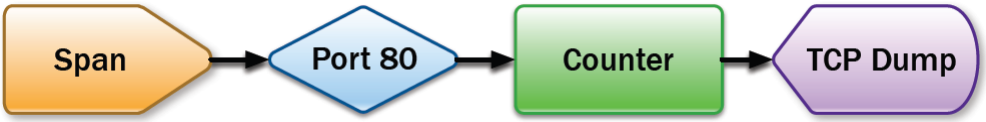


Figura 3.5: Traduzione dell'esempio.

descritto, viene tradotta nella Figura 3.5. Questa mostra il traffico in entrata su una porta SPAN (input), selezionato per la porta 80 (selezione), instradato attraverso un contatore (filtro) ed infine inviato ad un TCP dump (sink).

Il sistema complessivo di OpenSAFE è mostrato nella Figura 3.6

L'input è una connessione che parte dal percorso di SPAN nel punto di aggregazione della rete scelto fino ad una porta dello switch OpenFlow. Un certo numero di filtri sono in uso, assegnati alle varie porte dello switch. Infine, l'output è diretto in un qualche numero di sink.
Per monitorare grandi reti a line-rate, è possibile che un singolo filtro o sink non sia in grado di far fronte a tutto il traffico di rete. Per risolvere questo problema, si permette al traffico di essere indirizzato a più filtri o sink che operano in parallelo all'interno di uno stesso percorso.

3.2.4 ALARMS - A Language for Arbitrary Route Management for Security

Per consentire agli amministratori di rete di gestire e aggiornare la propria infrastruttura di controllo facilmente, viene introdotto *ALARMS*, un linguaggio per la gestione del percorso arbitrario per la sicurezza del traffico. ALARMS

pacchetti e trasmissioni.

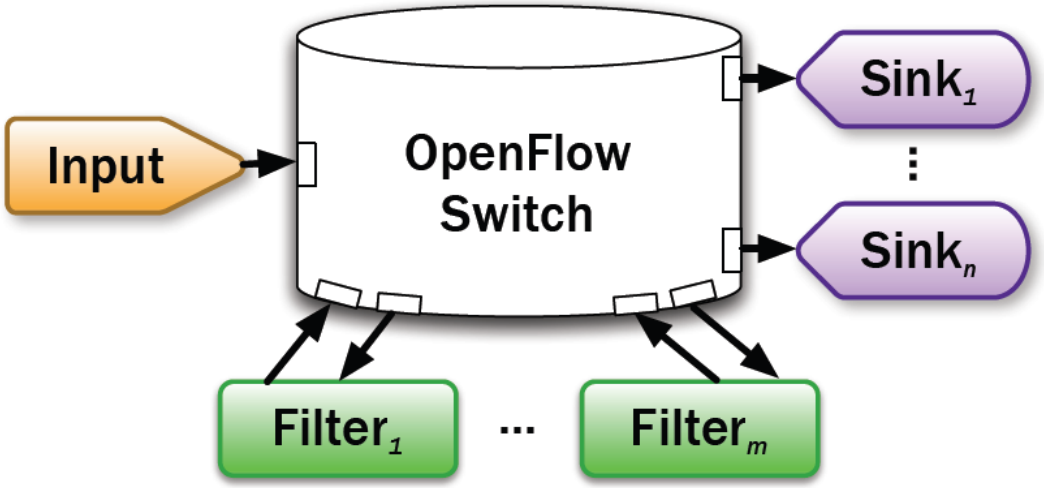


Figura 3.6: Sistema complessivo OpenSAFE.

utilizza le astrazioni citate nel paragrafo precedente (ad esempio quella di *paths*) per creare una semplice sintassi del linguaggio per descrivere e gestire i percorsi. Questi percorsi sono definiti tra i componenti citati (input, select, filter, sink). Ciascun componente può essere soggetto ad una regola di distribuzione nel caso di molteplicità o parallelismo.

ALARMS è un linguaggio di programmazione ad alto livello e si basa su un'interfaccia di programmazione di basso livello per uno switch di rete. Si usa OpenFlow come tecnologia per programmare gli switch. Come regola, i nuovi flussi che non corrispondono a voci esistenti nella Flow Table vengono inviati al controller. In questo modo, il controller può manipolare le tabelle di flusso dinamicamente, basandosi sull'attività di rete.
OpenFlow permette alle voci della tabella dei flussi di basarsi su un massimo di dieci voci per ogni pacchetto (compreso l'indirizzo IP di sorgente/destinazione, porta sorgente/destinazione, e così via). Questo è noto come *OpenFlow 10-tuple*.
ALARMS è implementato con OpenFlow e dovrebbe essere abbastanza generico per gestire qualsiasi livello di rete programmabile.
In ALARMS, tutti i componenti di un percorso sono indicati da un unico tipo e da un unico nome. In particolare sono:

- **OpenFlow switch**: ha un nome univoco che corrisponde al proprio

datapath ID all'interno della propria rete;

- **Input and Sinks**: come mostrato nella Figura 3.3 del design complessivo, input e sink sono semplicemente porte dello switch OpenFlow;

- **Filters** sono middlebox all'interno di una rete OpenSAFE. Un filtro è una combinazione di un sink più i rispettivi ingressi. Come tali, i filtri sono definiti analogamente agli input e ai sink, ma con un po' più di flessibilità in quanto sono in grado di trasmettere e ricevere traffico.

- **Selection**: sono un'istanza del tradizionale *OpenFlow 10-tuple* con una sintassi booleana limitata; in particolare ALARMS permette un sintassi di questo tipo:

$$select\ http\ =\ tp\_src : 80\ ||\ tp\_dst : 80;$$

  che seleziona solo il traffico decrittato HTTP (essendo sulla porta 80). Ogni altro campo che non è specificato nella selezione non viene preso in considerazione;

- **Waypoint**: non sono destinazioni fisiche, esistono solo all'interno del linguaggio ALARMS. Questo permette alle regole di percorso di fare riferimento al waypoint come origine o destinazione.

3.2.5 Regole di distribuzione

Al fine di permettere funzioni di load balancing, ogni porzione di un percorso può essere distribuita tra diversi componenti. Questa distribuzione del traffico tra i componenti è gestita da una delle seguenti regole di distribuzione e applicate sulla base di ogni flusso:

- ALL (Si duplica)

- RR (Round Robin)

- ANY (Random)

- HASH (Si applica una funzione hash)

Queste regole di distribuzione sono rappresentate dal linguaggio di policy nel seguente modo:

$$span[http] \rightarrow \{ALL,\ counter1,\ counter2\} \rightarrow tcpdump;$$

Le prime tre regole sono abbastanza semplici. ALL invia dei flussi in entrata a tutti i componenti nella lista; RR distribuisce i flussi a ciascun componente in un ordine secondo la gestione Round Robin[9]; ANY inoltra i flussi ad uno dei componenti selezionato casualmente. Infine HASH è un caso speciale di ANY; considera un ulteriore parametro: il nome della funzione hash. Questa è fornita con il primo pacchetto del flusso così come l'intero elenco di distribuzione, e ci si aspetta che ritorni il componente in cui dovrebbe essere trasmesso.

3.2.6 Limitazioni degli switch

Gli switch OpenFlow sono switch Ethernet (hardware) in grado di utilizzare il protocollo OpenFlow. Trattandosi di prodotti fisici, spesso hanno limitazioni che variano tra i fornitori e tra i prodotti. Ci sono tre problemi distinti quando si utilizza degli switch OpenFlow:

1. il possibile esaurimento della Flow-Table,
2. la capacità di matching,
3. la latenza nell'inserimento delle entry.

Il numero di voci della tabella di flusso è spesso molto limitata negli switch; in genere circa 1500-3000 voci. Con queste voci spesso non tutti i campi sono in grado di essere abbinati sullo switch. Infine, se nuovi flussi devono essere valutati dal controller quando si inserisce un flusso in entrata allo switch OpenFlow, sono necessari fino a diverse centinaia di millisecondi al pacchetto iniziale del flusso per andare al controller e per inserire una nuova voce nella tabella di flusso.

Per emulare le regole di ALARMS, i flussi devono essere gestiti dinamicamente dal controller. Questo si traduce in un tempo relativamente lungo di

[9]In informatica, la politica di scheduling Round Robin è quella in cui tutti i processi attivi in un sistema ricevono il controllo della CPU secondo un turno, assegnato in maniera circolare.

andata e ritorno al controller per ogni funzione di hashing lungo il percorso che un flusso prenderà.

Un'altra limitazione riguarda il fatto che OpenFlow non ha il concetto di *waypoint* (tappa) quindi si deve ricalcolare qualsiasi flusso contenente dei waypoint. Attraverso la creazione del prodotto incrociato tra le regole di percorso che terminano ed iniziano in un waypoint, viene creato l'insieme di regole rappresentative di OpenFlow.

3.3 CloudWatcher

CloudWatcher[16] è un framework che fornisce servizi di monitoraggio per le reti cloud. Questo, devia automaticamente i pacchetti di rete per essere controllati da dispositivi di sicurezza di rete preinstallati. Tutte queste operazioni possono essere implementate scrivendo un semplice script, quindi, un amministratore di rete cloud è in grado di proteggere facilmente la sua rete cloud.

Esso porta alcuni benefici nelle reti cloud:

- Controlla i flussi di rete garantendo che tutti i pacchetti siano ispezionati da dispositivi di sicurezza appropriati;
- Fornisce un linguaggio semplice di script per aiutare le persone ad utilizzare i servizi forniti.

3.3.1 Architettura

Fondamentalmente CloudWatcher può essere realizzato come applicazione su sistemi operativi di rete, i quali sono utilizzati per controllare i router di rete o gli switch in ambienti SDN. CloudWatcher è formato da tre componenti principali (Figura 3.7):

1. dispositivo gestore delle politiche: gestisce le informazioni dei dispositivi di sicurezza;
2. generatore di regole di instradamento: crea un insieme di regole per gestire ogni flusso;
3. dispositivo che mette in pratica le regole: fa rispettare le regole di flusso generate per gli switch.

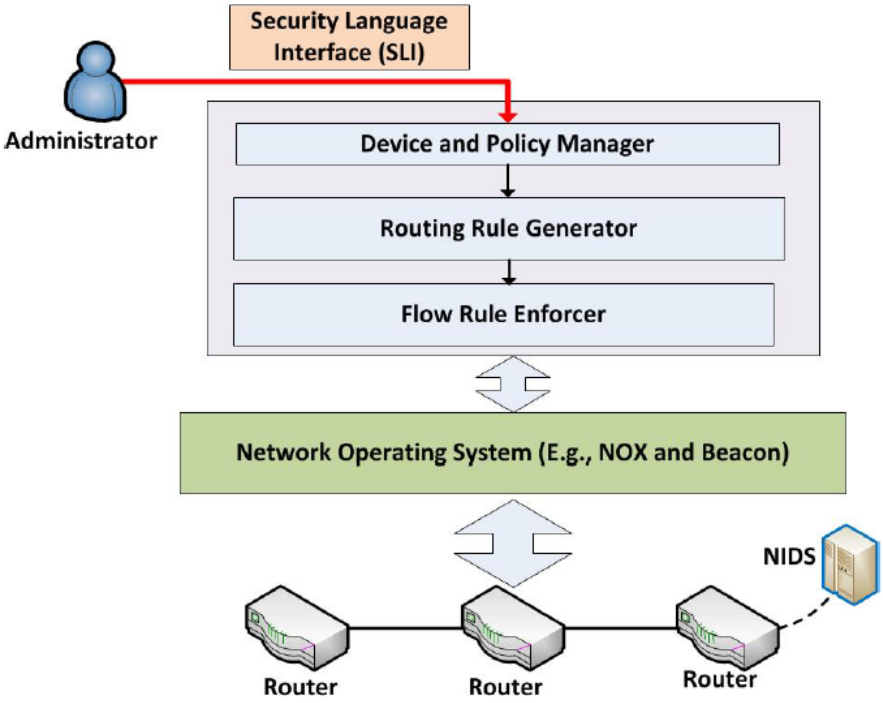


Figura 3.7: Componenti principali di CloudWatcher.

Per usare i dispositivi di sicurezza attraverso CloudWatcher è necessario per prima cosa registrarli. Ciò richiede l'invio di alcune informazioni di base per ogni dispositivo:

ID del dispositivo identificatore univoco;

Tipo del dispositivo sottolinea la funzione principale del dispositivo;

Luogo locazione nella quale il dispositivo è installato;

Modalità d'installazione rappresenta com'è installato;

Funzioni supportate descrivono i tipi di funzioni di sicurezza che vengono fornite dal dispositivo.

Un amministratore, per creare politiche di sicurezza, come ad esempio per un servizio di monitoring utile per ogni esigenza, deve compilare questi due campi:

1. **Flow condition**: rappresenta il flusso che deve essere osservato;
2. **Device set**: mostra i dispositivi di sicurezza necessari per monitorarlo.

Nel campo *Flow condition*, l'amministratore può definire diversi tipi di condizioni che dipendono dai campi corrispondenti che sono supportati dalla specifica SDN. Per esempio nella specifica OpenFlow ci sono 15 di questi campi (ad esempio, porta di destinazione o sorgente, IP destinazione o sorgente) che l'amministratore di rete può usare per impostare il primo dei due campi sopraelencati.

Si può inoltre specificare quali dispositivi di sicurezza sono usati per monitorare i pacchetti di rete; questo viene specificato nel secondo campo sopraelencato. In aggiunta è possibile specificare più dispositivi per il monitoraggio in questo campo.
Per controllare i flussi di rete, CloudWatcher controlla i pacchetti che transitano nella rete e se ne trova che combaciano con una *flow condition* specificata all'interno di una policy, allora li farà deviare in modo che soddisfino le richieste di sicurezza. Quando CloudWatcher devia dei pacchetti di rete, deve considerare le seguenti condizioni:

- i pacchetti di rete devono passare attraverso degli specifici router o collegamenti di rete che specificano a quali dispositivi di sicurezza sono associati;
- i percorsi di instradamento devono essere ottimizzati.

3.4 Resonance

Resonance[17] è un sistema per la protezione delle reti aziendali, dove i dispositivi di rete implementano le politiche di controllo di accesso dinamico basate sia su informazioni raccolte a livello di flusso che su avvisi in tempo reale. Resonance inietta nel livello di rete le funzioni di base per implementare le policy di sicurezza, così come un'interfaccia di controllo permette ai sistemi di monitoraggio di controllare il traffico in base a criteri predefiniti.

Nelle versioni precedenti, Resonance non faceva altro che controllare il traffico usando delle politiche che il controller installava negli switch programmabili. Estendendo questo paradigma si è creato un framework di controllo degli accessi che integra il controller con tutti i vari sottosistemi di monitoring. Questa integrazione permette ad un operatore di specificare come la rete dovrebbe controllare il traffico all'interno dell'azienda in base ai cambiamenti delle condizioni della rete. Per esempio, Resonance può automaticamente mettere in quarantena host o sottoinsiemi di traffico quando viene rilevata una violazione della sicurezza.

3.4.1 Sfide

Nonostante il funzionamento di Resonance appaia promettente e le tendenze recenti potrebbero rendere la sua distribuzione più fattibile, sono presenti alcuni problemi che devono essere affrontati:

1. Resonance deve essere scalabile[10] per un ampio numero di utenti e di flussi di traffico. Il sistema deve fornire flessibilità e controllo dinamico, senza memorizzare un numero troppo elevato di stati all'interno degli switch o introdurre ritardi eccessivi nell'inoltro dei pacchetti;

2. Deve essere sensibile ai vari cambiamenti nella politica di rete: deve rapidamente autenticare gli host di rete e i dispositivi e deve velocemente mettere in quarantena gli host che violano le politiche di sicurezza;

3. Controller e switch OpenFlow devono essere integrati con sistemi di monitoring real-time; il controller deve essere in grado di correlare e sintetizzare velocemente gli allarmi e inviare messaggi di controllo agli switch per eventualmente modificare i flussi di traffico;

[10]Scalabilità: capacità di un sistema di "crescere" o diminuire di scala in funzione delle necessità e delle disponibilità.

4. Il canale di controllo deve essere sicuro: controller e le interfacce degli switch devono essere robusti per resistere agli attacchi, il canale di controllo tra il controller e i dispositivi deve essere disponibile.

3.4.2 Architettura

Di seguito viene descritta l'architettura di Resonance. Le proprietà fondamentali sono:

Policy specification framework i framework tipicamente assegnano ad ogni dispositivo una classe di sicurezza. Siccome il controllo dell'accesso in Resonance è dinamico, ogni dispositivo ha sia una classe di sicurezza che uno stato. Resonance permette agli operatori di rete di specificare una varietà di funzioni basate sulla classe di sicurezza e lo stato del dispositivo. Per esempio, in base alle informazioni di un dispositivo con stato compromesso che sta entrando nella rete, il controller può dare istruzione agli switch di trattare il traffico di rete di conseguenza sulla base di una specifica politica di sicurezza.

Distributed network monitoring invece che fare affidamento sulla sicurezza a livello di host, Resonance assegna questo compito alla rete stessa. In letteratura si dimostra che, con l'utilizzo e lo studio delle informazioni sull'analisi del traffico di rete, si è in grado di svolgere attività di gestione di rete essenziali, come ad esempio il rilevamento di host compromessi, il filtraggio dello spam, e il degrado delle prestazioni. Resonance rende questa attività di monitoring intrinseca all'interno dell'architettura; i dispositivi di rete sono in grado di inoltrare i report attraverso la rete al controller centralizzato per migliorare lo studio delle informazioni di traffico.

Dynamic control with programmable switches a differenza delle reti odierne che disaccoppiano l'attività di monitoring dal controllo del traffico a livello inferiore, Resonance permette agli switch di rete di rimappare dinamicamente i client in base ad altri input (per esempio, allarmi dal sistema di monitoring). I sistemi di allerta controllano il traffico inviando messaggi al controller che, a sua volta, controlla il comportamento degli switch tramite lo standard OpenFlow. L'accoppiamento tra Resonance e i sistemi di allarme basati sullo studio delle informazioni ricevute dai

dispositivi di rete consente il controllo di accesso dinamico per cui i dispositivi di rete possono trattare il traffico in modo diverso in base alla classe di sicurezza ed allo stato attuale di un host che vede il controller.

3.4.3 Policy

A differenza dei framework esistenti per il controllo dell'accesso, il controllo degli accessi di Resonance permette di avere delle policy dinamiche. Queste politiche sono essenzialmente basate sul *lattice-based access control*[11] tranne che ogni dispositivo ha sia una classe di sicurezza che uno stato dove quest'ultimo può cambiare con il passare del tempo in accordo ad un insieme di transizioni definite dalle policy. La politica indica in modo efficace quali azioni uno switch dovrebbe assumere sul traffico da e verso un host che è di una particolare classe di sicurezza e con un certo stato.

Le politiche dinamiche consentono agli switch di rete di modificare il modo in cui loro controllano il traffico di un host in base al cambiamento delle condizioni della rete. Resonance facilita queste politiche attraverso delle specifiche che determinano:

- le possibili classi e membri per ciascun dispositivo;
- le corrispondenti politiche di controllo accessi;
- le azioni che gli elementi di rete dovrebbero adottare per applicare la politica;
- le indicazioni di come gli host possono passare da uno stato all'altro.

3.4.4 Classi di sicurezza

Come nei modelli di controllo d'accesso tradizionali, in Resonance i dispositivi in una rete hanno le classi di sicurezza che determinano il livello di accesso che essi (e il loro traffico) hanno sulle altre risorse della rete. In definitiva, possiamo estendere Resonance in modo che ogni risorsa (ad esempio, un dispositivo collegato a una porta switch) abbia una classe di sicurezza, e il *lattice-based access control* detterà come il traffico può scorrere verso una risorsa o altro.

[11] È un modello di controllo di accesso complesso che si basa sull'interazione tra qualsiasi combinazione di oggetti (come le risorse, computer e applicazioni) e soggetti (come individui, gruppi o organizzazioni).

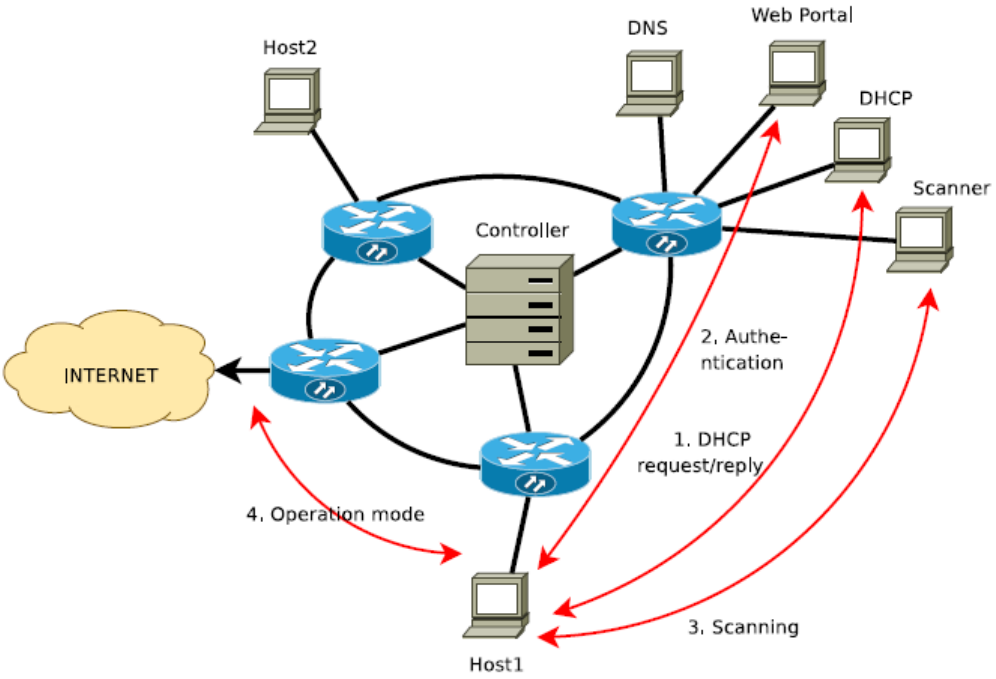


Figura 3.8: Rete utilizzata per l'esempio.

3.4.5 Stati e transizioni

Ogni classe di sicurezza ha un insieme predefinito di stati che determinano cosa è consentito fare ad un dispositivo di quella classe, quali transizioni sono permesse e ciò che provoca queste transizioni tra gli stati. Ogni stato può anche avere delle azioni che devono essere prese per certi sottoinsiemi di traffico (dove i tipi di traffico sono identificati dagli attributi di flusso). I diversi avvisi da sistemi di monitoraggio distribuiti possono causare una serie di transizioni da uno stato ad un altro.

3.4.6 Esempio di utilizzo di Resonance

In questo paragrafo si vede come Resonance lavora step-by-step quando un host si connette alla rete. Si considera una rete semplice composta da quattro switch OpenFlow, un controller, due host e quattro server come in Figura 3.8.

Registration State	
Match	**Action**
Ethernet Type ARP (0x806)	FLOOD
UDP src_port=68 dst_port=67 (DHCP ports)	FLOOD
UDP src_port=67 dst_port=68 (DHCP ports)	FLOOD
TCP Dst port 80/443/8080	REDIRECT (to web portal: 192.168.1.3)
*	DROP
Operation State	
Match	**Action**
*	FORWARD/DROP (according to policy lookup)

Figura 3.9: Entry per gli stati di Registrazione e Funzionamento.

Per prima cosa gli switch OpenFlow stabiliscono una connessione con il controller tramite un canale sicuro. Quando un nuovo host viene introdotto nella rete, in primo luogo trasmette un messaggio DHCP sulla porta 67 UDP[12] di discover (primo dei messaggi del protocollo DHCP); quando questo primo pacchetto DHCP viene ricevuto dallo switch collegato all'host, invia il pacchetto al controllore su un canale sicuro. In accordo con le politiche della tabella nella Figura 3.9. Il controller:

- genera una entry nella Flow–Table per consentire la comunicazione DHCP e ARP con l'host;

- aggiunge l'host al suo database di host e segna il suo stato come "Registrazione".

In questo stato, se un host avvia qualsiasi traffico che non sia DHCP o ARP, il controller installa una nuova entry nella Flow–Table degli switch con action = "DROP", a meno che il traffico sia http (porta 80), nel qual caso il controller

[12]Dynamic Host Configuration Protocol (DHCP) è un protocollo di rete di livello applicativo che permette ai dispositivi o terminali di una certa rete locale di ricevere dinamicamente ad ogni richiesta di accesso a una rete IP (quale Internet) la configurazione IP necessaria per stabilire una connessione e operare su una rete più ampia basata su Internet Protocol (`http://it.wikipedia.org/wiki/Dynamic\_Host\_Configuration\_Protocol`)

Authenticated State	
Match	**Action**
Ethernet Type ARP (0x806)	FLOOD
Dst_IP=192.168.1.4 (Scanner's IP)	FORWARD
Src_IP=192.168.1.4 (Scanner's IP)	FORWARD
TCP dst_IP=update sites	FORWARD
TCP src_IP=update sites	FORWARD
UDP dst_port=53 (DNS port)	FORWARD
UDP src_port=53 (DNS port)	FORWARD
*	DROP

Quarantined State	
Match	**Action**
TCP dst_port=80/443/8080	REDIRECT (to quarantine web page)
*	DROP

Figura 3.10: Entry per gli stati di Autenticazione e Quarantena.

installa una entry per reindirizzare il traffico al portale, che reindirizza l'utente al sito Web di autenticazione.
Un dispositivo nello stato di registrazione o messo in quarantena non può avviare una connessione, ma può sempre ricevere pacchetti da una macchina nello stato di "Funzionamento" (*Operation*).

Il portale Web consente all'utente di autenticarsi e informa il controller dello stato di autenticazione tramite una connessione separata. Dopo l'autenticazione, il controller sposta l'host dallo stato di "Registrazione" a quello di "Autenticato" o di "Quarantena". Quindi, elimina tutte le entry vecchie dalla Flow–Table corrispondente all'indirizzo MAC dell'host e installa una nuova serie di entry, come indicato nella tabella della Figura 3.10. L'unica modifica apportata dallo stato di "Registrazione" a quello "Autenticato" è che l'host può comunicare con il sito di scanner e di aggiornamento. Lo scanner esegue la scansione del computer per le potenziali vulnerabilità.
Se la macchina si rivela vulnerabile, viene reindirizzata verso siti di aggiornamento per correggere le potenziali vulnerabilità. Una volta che sono state applicate le patch di aggiornamento, lo scanner lo comunica al controller che poi trasferisce l'host allo stato di "Funzionamento"(*Operation*) e aggiorna le voci della Flow–Table di conseguenza. Una volta in questo stato, l'host può

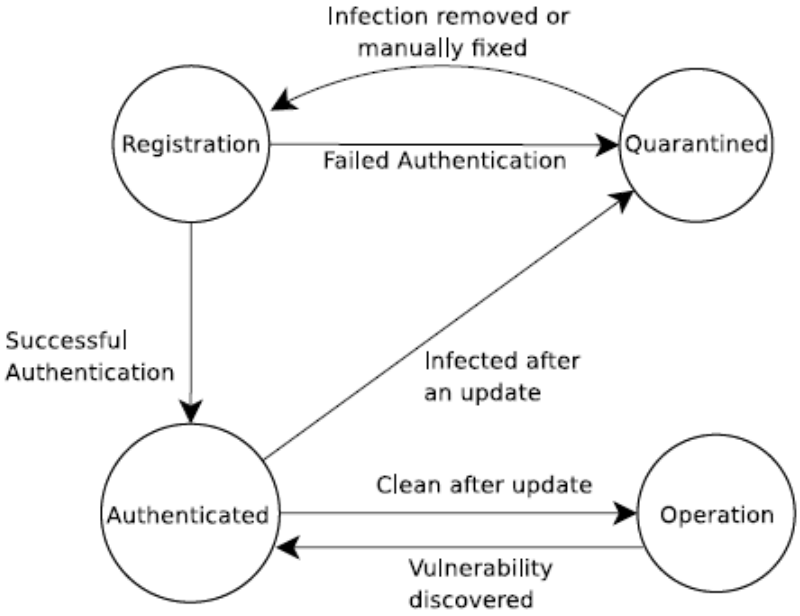


Figura 3.11: Transazioni di stato dell'host.

connettersi ad ogni destinazione di internet. Durante il normale funzionamento, l'host può compromettersi. Se l'allarme di rete informa il controller di questo evento, il controller può poi spostare l'host allo stato "Autenticato".

Capitolo 4

Tecnologie per lo sviluppo di reti SDN

Una volta chiarito il funzionamento del paradigma di Software Defined Networking e dopo aver largamente discusso sui vari componenti fondamentali di funzionamento, il loro ruolo nella rete e come interagiscono tra loro, si passa ora un po' più sul pratico. Questo significa che abbandoniamo tutte le discussioni teoriche per passare alla descrizione di quelli che sono gli strumenti che vengono messi a disposizione per sviluppare e testare nuove topologie di rete, nelle quali è possibile eseguire controller personalizzati, applicare nuovi protocolli e cercare di risolvere i problemi di sicurezza (molti dei quali tuttora in fase di sviluppo o completamente sulla carta).

4.1 Piattaforma Mininet

Mininet[7][8] è un emulatore di rete. Gestisce un insieme di terminali di rete, switch, router, e collegamenti su un unico kernel Linux[1]; quindi è un ottimo modo per sviluppare e sperimentare attraverso il protocollo OpenFlow i sistemi di Software-Defined Networking. Mininet usa una virtualizzazione leggera (infatti è solo a riga di comando) per rendere un singolo sistema capace di simulare una rete completa, tutto all'interno dello stesso kernel. Un host Mininet si comporta proprio come una vera e propria macchina; si può utilizzare il

[1]Nel seguente link sono presenti le immagini da montare su una Virtual Machine: `https://github.com/mininet/mininet/wiki/Mininet-VM-Images`

protocollo ssh (descritto brevemente in uno dei prossimi paragrafi) al suo interno ed eseguire programmi arbitrari (tra cui tutto ciò che è installato sul sistema Linux sottostante). I programmi eseguiti possono inviare pacchetti attraverso quello che sembra una vera e propria interfaccia ethernet, con una certa velocità di collegamento e con un certo ritardo di propagazione e i pacchetti contenenti i dati vengono elaborati da quello che sembra un vero e proprio switch ethernet, router, o middlebox.

In breve, gli host virtuali, gli switch e i controller di Mininet sono una cosa reale; sono solo creati utilizzando il software piuttosto che l'hardware e per la maggior parte, il loro comportamento è simile ad elementi hardware discreti.

4.1.1 Vantaggi

Di seguito vengono indicate diverse proprietà che caratterizzano l'emulatore Mininet:

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

Personalizzazione I è possibile[9] creare topologie personalizzate: un singolo switch, grandi topologie come quella di internet, un data center, o qualsiasi altra cosa.

Esecuzione di programmi è possibile eseguire programmi veri e propri: tutto ciò che gira su Linux è disponibile per l'esecuzione; da un server web a strumenti di monitoraggio di rete come Wireshark[2].

Personalizzazione II è possibile personalizzare l'inoltro dei pacchetti: gli switch di Mininet sono programmabili utilizzando il protocollo OpenFlow. Progetti di reti SDN che girano su Mininet possono essere facilmente trasferiti a switch OpenFlow per inoltrare i pacchetti a line-rate.

Portabilità è possibile eseguire Mininet sul computer portatile, su un server, in una Virtual Machine o su una macchina nativa Linux (Mininet è incluso con Ubuntu 12.10).

Condivisione e replicazione è possibile condividere e replicare i risultati: chiunque con un computer può eseguire il codice una volta compresso.

[2] `https://www.wireshark.org/`

Semplicità si può usare facilmente: è possibile creare ed eseguire esperimenti Mininet scrivendo semplici (o complessi se necessario) script in Python.

Open source è un progetto open source, quindi è possibile esaminare il suo codice sorgente, modificare, correggere bug o inviare patch.

4.1.2 Limitazioni

Purtroppo Mininet, pur essendo un buon strumento di sviluppo, presenta delle limitazioni. Per esempio:

Limitazione delle risorse in esecuzione su un singolo sistema è conveniente, ma impone dei limiti alle risorse: se il server dispone di 3 GHz di CPU e può far passare circa 3 Gbps di traffico simulato, queste risorse dovranno essere equilibrate e condivise tra gli host virtuali e gli switch.

Unicità Mininet utilizza un unico kernel Linux per tutti gli host virtuali; questo significa che non è possibile eseguire il software che dipende da Windows o altri kernel del sistema operativo (anche se è possibile collegare macchine virtuali a Mininet).

Niente aiuti Mininet non scriverà il controller OpenFlow per coloro che lo vogliono utilizzare; se si ha bisogno di un instradamento personalizzato o un comportamento di commutazione specifico, è necessario trovare o sviluppare un controller con le caratteristiche desiderate.

4.2 Lavorare con Mininet

In questa sezione vedremo come interagire, tramite linea di comando, con la macchina virtuale dove si è installato Mininet. Nella Figura 4.1 viene mostrata l'interfaccia minimale della macchina.

4.2.1 Creazione di topologie ed interazione tra host e switch

Di seguito verranno elencati alcuni comandi da utilizzare in Linux o direttamente in Mininet per creare ex novo delle topologie di rete o per avere informazioni circa la topologia già implementata.

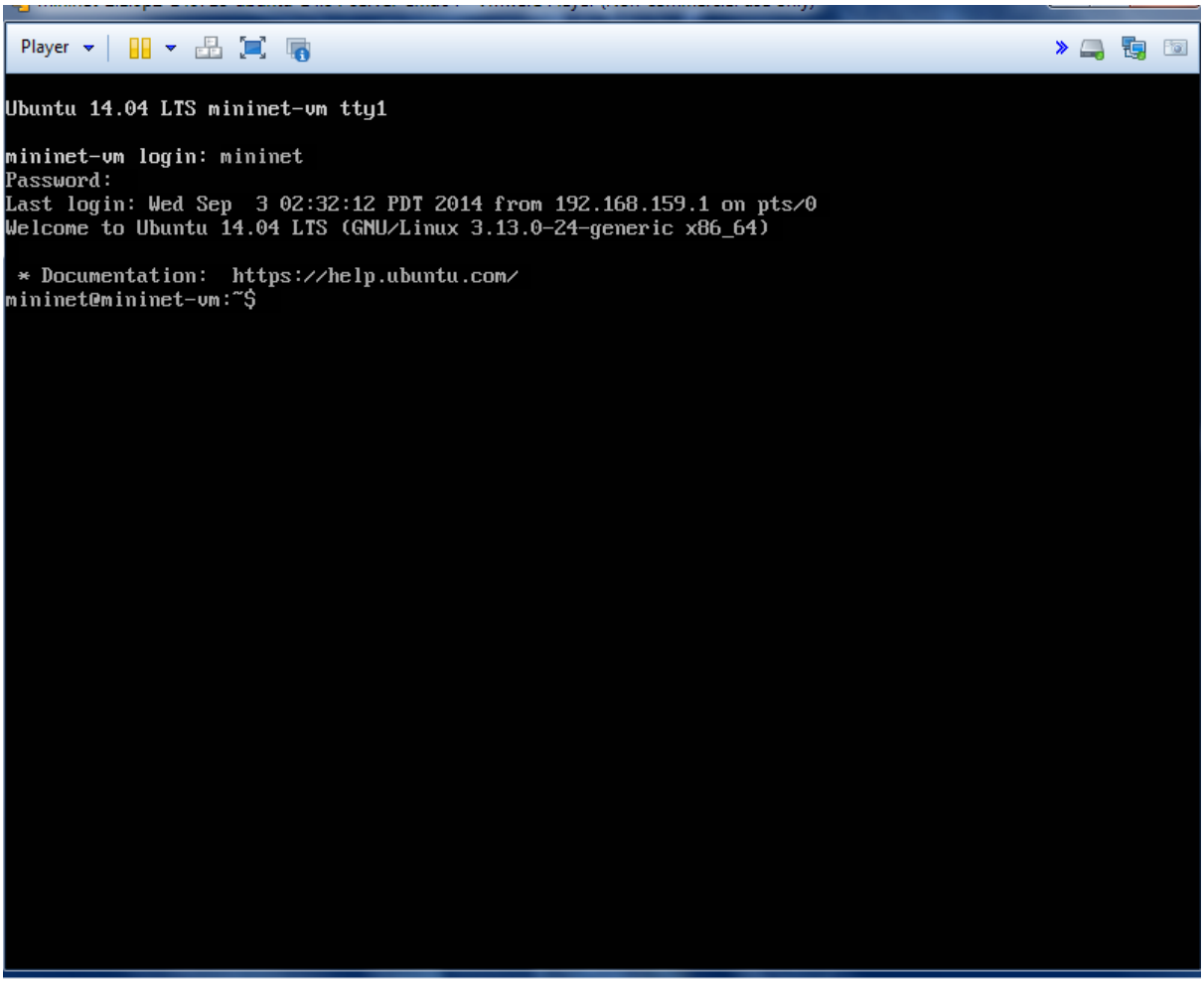


Figura 4.1: Interfaccia di Mininet.

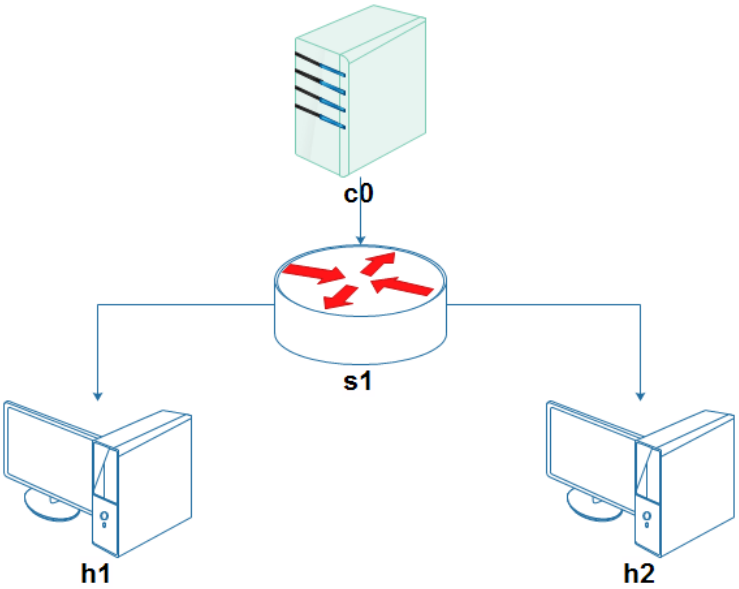


Figura 4.2: Topologia creata tramite il comando *sudo mn*

Il comando per creare una topologia minimale di rete formata da un controller, uno switch e due host è il seguente:

$ sudo mn

[3] e la topologia che viene creata è quella in Figura Figura 4.2:

Questa topologia può essere creata anche specificando, attraverso command line, l'opzione *–topo=minimal*. Altre topologie sono disponibili attraverso l'uso di questa opzione[4]

Per visualizzare tutti i possibili comandi di Mininet digitare:

mininet > help

[3] Quando si antepone il simbolo '$' significa che il comando è lanciato dalla shell di Linux mentre se è lanciato dalla console di mininet verrà visualizzato il testo:

mininet >

.

[4] Utilizzare il comando *sudo mn -h* per visualizzare tutte le varianti.

mentre per la lista di tutti i nomi dei nodi della rete (compreso il controller) si utilizza il seguente comando:

$$mininet > \ nodes$$

e invece per visualizzare tutti i collegamenti tra i vari nodi si usa:

$$mininet > \ net$$

.

Successivamente, se vogliamo andare più nello specifico di un nodo, possiamo utilizzare:

$$mininet > \ h1 \ ifconfig$$

oppure

$$mininet > s1 \ ifconfig$$

per visualizzare le interfacce di rete proprie dell'host *h1* o dello switch *s1*. Se la prima stringa scritta nella console di Mininet è il nome di un nodo della topologia allora il comando che viene scritto successivamente si riferisce a quel nodo.

4.2.2 Testare la connettività tra gli host

Per testare la connettività tra gli host, ora che abbiamo creato la topologia di rete, si utilizza il comando *ping* nella seguente maniera:

$$mininet > \ h1 \ \ ping \ \ -c \ \ 3 \ \ h2$$

Questo verifica la raggiungibilità o meno dell'host *h2* partendo da *h1*. Il primo host (sorgente) invia una trama broadcast (ARP request) contenente l'indirizzo IP del nodo destinazione. Questo causa l'invio di un messaggio di tipo packet_ in per il controller; quest'ultimo invia un messaggio di tipo packet_ out allo switch. Il secondo host vede l'ARP Request ed invia un ARP Reply broadcast in risposta. Questa risposta giunge al controller che rimanda la Reply al primo host installando nello switch una flow entry. Ora, il primo host conosce l'indirizzo IP del secondo e può inviare i suoi ping attraverso il protocollo ICMP. Sia la richiesta che la risposta del ICMP vengono processate dal controller e verrà installato un nuovo flusso sempre nella flow table.

Se si va a considerare l'output di questo comando, si può osservare (Figura 4.3) che il tempo del primo ping è nettamente superiore a quello degli altri. Questo appunto perché il primo ha aspettato che si installassero nello switch le regole per gestire questo tipo di azioni.

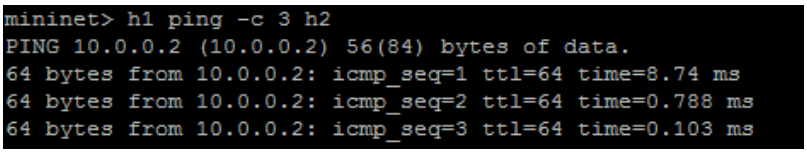


Figura 4.3: Screenshot di ping.

4.2.3 Altri comandi utili

Completiamo questa lista di comandi di Mininet con questi ultimi.

$$mininet > \ pingall$$

utilizzato per testare da tutti gli host il ping verso tutti gli altri;

$$mininet > \ exit$$

se si vuole uscire dalla particolare topologia e ritornare alla linea di comando di Linux; ed infine:

$$\$ \ sudo \ mn \ -c$$

utilizzato da Linux per 'ripulire' la memoria. Viene utilizzato nel caso in cui succede un crash per un qualche motivo inaspettato.

4.3 SSH: Trasferimento file

Secure Shell (SSH) è un protocollo di rete che permette di stabilire una sessione remota cifrata tramite interfaccia a riga di comando con un altro host di una rete. Si è utilizzato questo protocollo per trasferire file da Windows alla macchina virtuale con sopra installato Mininet e per simulare l'interfaccia della macchina virtuale direttamente in una finestra di Windows, con la possibilità di creare, per ogni nodo della rete, una propria finestra pop-up.

4.3.1 PSCP

PSCP[5] è un client SCP (Secure CoPy) il quale è un mezzo per trasferire in modo sicuro un file tra un computer locale ed un host remoto o tra due host remoti, usando il protocollo Secure Shell.

L'utilizzo di questo client da Windows è abbastanza semplice. Una volta aperta la console di Windows ci si deve spostare nella cartella dove è presente il file *pscp.exe* e digitare il seguente comando:

pscp.exe pathname_ file_ da_ traferire mininet@"indirizzo_ IP_ VM":"pathname_ cartella_ destinazione"

esempio:

pscp.exe C:\Users\Utente\Cartella\Prova.txt mininet@192.168.120.154:/home/mininet/prova[6]

e successivamente viene chiesta la password della macchina virtuale.

Nel progetto si è utilizzato questo metodo per copiare dei file Python e degli script da Windows alla macchina virtuale di Mininet.

4.3.2 PuTTY

PuTTY implementa il lato client di una sessione SSH utilizzato per l'esecuzione di una sessione remota su un computer, tramite una rete.
In termini molto semplici: si esegue PuTTY su una macchina Windows dicendogli di connettersi (per esempio) a una macchina Unix. PuTTY apre una finestra. Qualsiasi cosa si digiti in quella finestra viene inviato direttamente alla macchina Unix, e tutto ciò che la macchina Unix rimanda viene visualizzato nella finestra. Così si può lavorare sulla macchina Unix come se si fosse seduti direttamente davanti alla macchina, mentre in realtà si è seduti da qualche altra parte.
Si è utilizzato, in questo modo, PuTTY per interagire direttamente da Windows con la macchina virtuale con sopra presente Mininet. In una finestra si è implementata la topologia di rete mentre nell'altra si sono eseguiti script utili che verranno argomentati tra poco.

[5]Scaricabile dal sito `http://www.chiark.greenend.org.uk/\textasciitildesgtatham/putty/download.html`

[6]Tutto un unico comando.

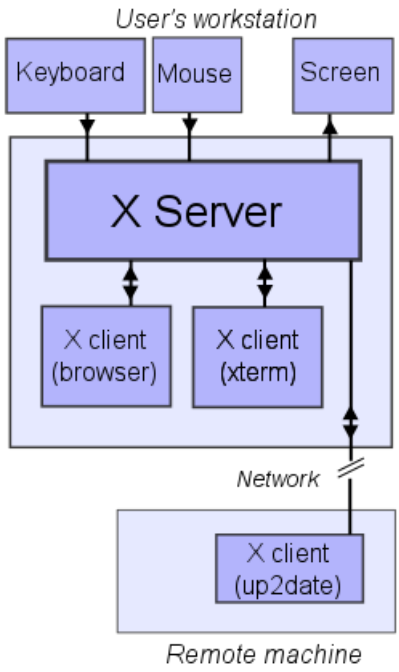


Figura 4.4: Funzionamento xterm.

4.4 Xterm

xterm è l'emulatore standard di terminale in ambiente Unix-like. Un utente può avere più sessioni di xterm avviate su uno o più display, le quali forniscono un sistema di input/output per i processi lanciati[7].

Il funzionamento di questo, è riassunto nella Figura 4.4.

La "remote machine" nel caso specifico sarebbe la macchina virtuale di Mininet che, connessa tramite la rete locale al X Server, è in grado di interagire con i vari client presenti su Windows.
Per ulteriori debug più complessi, è possibile avviare Mininet in modo che generi uno o più xterm. Per fare sì che anche lato Windows sia possibile questo proliferare di finestre, si ha bisogno di installare un programma che abiliti l'X forwarding; si è scelto di utilizzare il software gratuito Xming[8]. Poi, una

[7] Da Wikipedia.
[8] `http://www.straightrunning.com/XmingNotes/`

volta installato, prima di connettersi con PuTTY alla macchina virtuale, si deve abilitare l'X forwarding, un'opzione che si trova nella finestra iniziale di PuTTY.

Per creare un xterm per ogni nodo della topologia connesso alla rete si utilizza il seguente comando:

$$\$ \; sudo \; mn \; -x$$

mentre dall'interno di Mininet si può creare un xterm di uno dei nodi che si vuole semplicemente scrivendo:

$$mininet > \; xterm \;\; h1$$

per aprire la finestra di h1. In questo modo, per ogni nodo che si apre con xterm, si possono eseguire tutti i comandi di un normale Linux a linea di comando ed anche quelli di Mininet come ad esempio *ping*(Figura 4.5). Nel caso del comando ping però, si deve specificare non più il nome del particolare nodo ma bensì il suo indirizzo IP.

4.4.1 Iperf

Tramite l'uso di xterm si agevola l'utilizzo e la gestione dei singoli nodi della topologia di rete, facilitando e consentendo l'utilizzo di diversi comandi. Tra tutti, uno che si è testato è *iperf*. Questo è uno strumento per misurare le prestazioni in termini di larghezza di banda di una connessione tra due host; più precisamente per misurare il limite massimo di larghezza di banda.
Per funzionare, questo comando ha bisogno di un host che faccia da server e di uno o che sia il client. Il server non farà altro che eseguire sul suo terminale il comando:

$$iperf \; -s$$

mentre il client, oltre a specificare che vuole effettivamente esserlo, deve anche specificare l'indirizzo IP del server iperf con il quale vuole testare la larghezza di banda; in questo modo:

$$iperf \; -c \; 10.0.0.1$$

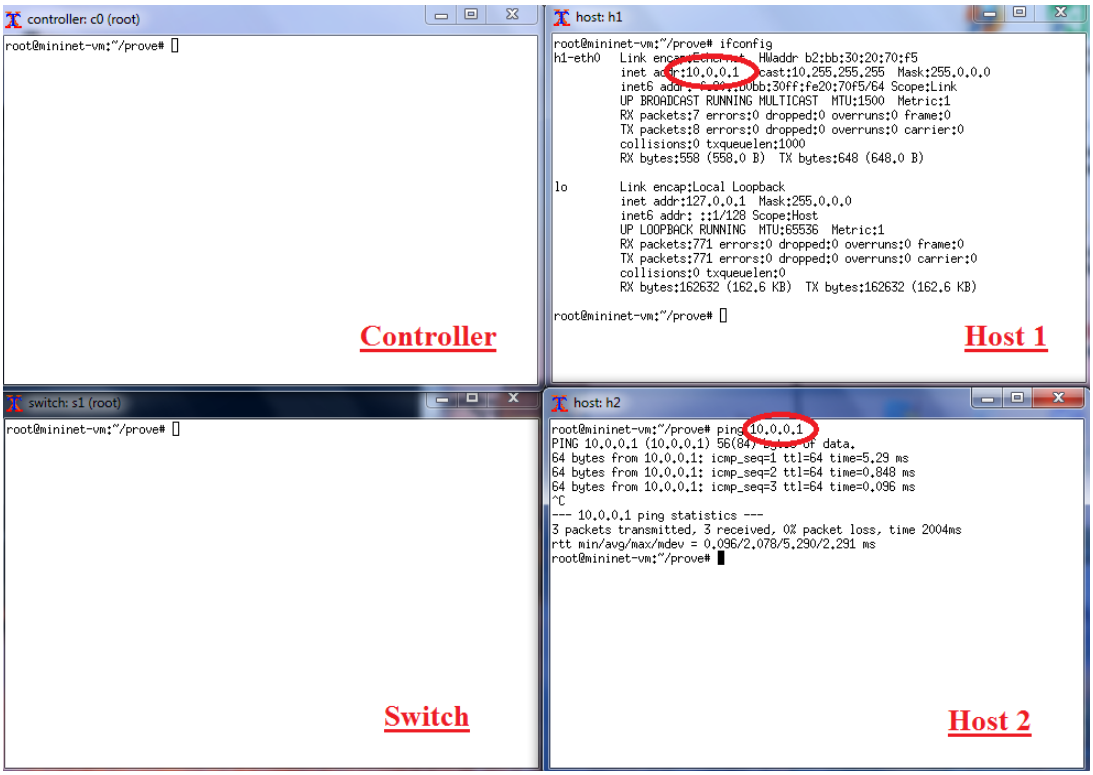


Figura 4.5: Finestre xterm ed esempio di ping.

Così facendo, parte il test e, dopo un intervallo di 10 secondi[9] si visualizzano le specifiche del test su entrambi gli host. Più precisamente si possono osservare l'intervallo di tempo trascorso, la quantità di dati scambiata e la larghezza di banda media che è stata usata.

È anche possibile, al momento della creazione della topologia, specificare la massima larghezza di banda nei link tra host e switch. L'opzione per specificarlo è:

$$\$ \ sudo \ \ mn \ \ --link \ \ tc, bw = 10$$

In questo modo si va a specificare che i link tra host e switch devono essere a 10 Mb.

4.4.2 Wireshark

Per visualizzare e controllare il traffico che scorre nella rete, sulla macchina virtuale è già installato il noto tool di monitoraggio di rete Wireshark. Per poter aprire una finestra di questo si utilizza PuTTY connettendo la macchina virtuale a Windows con l'X forwarding abilitato. Successivamente è sufficiente digitare il seguente comando e verrà aperta la finestra desiderata:

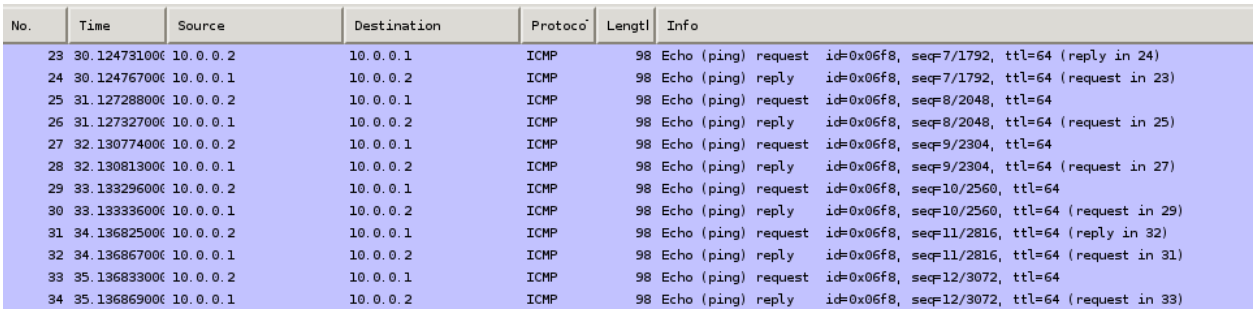
$ sudo wireshark &

Ora, se si vuole visualizzare il traffico che scorre attraverso un'interfaccia ethernet dello switch, si seleziona l'interfaccia nella pagina iniziale e si clicca su "Start". D'ora in poi Wireshark catturerà tutti i pacchetti transitanti questa interfaccia. Se si prova ad eseguire un ping da due host connessi allo switch che si sta osservando con Wireshark, si visualizzeranno tutti i pacchetti ICMP che si scambiano i due host (Figura 4.6)

4.5 Creazione topologia personalizzata

Finora si è lavorato con una topologia di rete standard che mette a disposizione Mininet da linea di comando tramite l'opzione *–topo*. Oltre a queste, è possibile creare, tramite uno script esterno, la topologia che si vuole. Lo script da implementare è in Python (estensione .py) e di seguito verranno descritte le principali direttive per creare switch e host e per aggiungere i collegamenti tra di essi.

[9]Intervallo standard, se lo si vuole modificare bisogna che lo si specifichi tramite l'opzione -t.



No.	Time	Source	Destination	Protocol	Length	Info
23	30.124731000	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x06f8, seq=7/1792, ttl=64 (reply in 24)
24	30.124767000	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id=0x06f8, seq=7/1792, ttl=64 (request in 23)
25	31.127288000	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x06f8, seq=8/2048, ttl=64
26	31.127327000	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id=0x06f8, seq=8/2048, ttl=64 (request in 25)
27	32.130774000	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x06f8, seq=9/2304, ttl=64
28	32.130813000	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id=0x06f8, seq=9/2304, ttl=64 (request in 27)
29	33.133296000	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x06f8, seq=10/2560, ttl=64
30	33.133336000	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id=0x06f8, seq=10/2560, ttl=64 (request in 29)
31	34.136825000	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x06f8, seq=11/2816, ttl=64 (reply in 32)
32	34.136867000	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id=0x06f8, seq=11/2816, ttl=64 (request in 31)
33	35.136833000	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x06f8, seq=12/3072, ttl=64
34	35.136869000	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id=0x06f8, seq=12/3072, ttl=64 (request in 33)

Figura 4.6: Pacchetti in Wireshark.

4.5.1 Direttive Python

Dopo aver eseguito gli import (presenti nell'esempio del prossimo paragrafo) e dopo aver creato una classe apposita, per aggiungere alla topologia (per ora vuota) uno switch chiamato *s1*, si utilizza il seguente comando:

```
switch1 = self.addSwitch('s1')
```

mentre per aggiungere un host, chiamato *h1*, si utilizza questo:

```
host1 = self.addHost('h1')
```

ed infine, per creare un collegamento, che sia esso tra un host e uno switch o tra due switch, basta digitare:

```
self.addLink(host1, switch1)
```

oppure

```
self.addLink(switch1, switch2)
```

Per creare la topologia personalizzata che si vuole, dopo aver creato lo script, basta aggiungere un'opzione al comando di creazione delle topologie standard:

$$\$ \; sudo \;\; mn \;\; --custom \;\; topology.py \;\; --topo = mytopology$$

dove *topology.py* è il file Python creato con le direttive appena descritte e *mytopology* è il nome che si è dato, per ultima istruzione prima della fine dello script, alla topologia.

4.5.2 Esempio di topologia

Di seguito si riporta lo script Python per la creazione di una topologia personalizzata di rete. Essa ha 4 switch e 6 host.

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class MyTopology(Topo):

        def __init__(self):
                #Initialize topology
                Topo.__init__(self)
                #Adding switch
                switchLeft = self.addSwitch('s1')
                switchRight = self.addSwitch('s2')
                switchTopLeft = self.addSwitch('s3')
                switchTopRight = self.addSwitch('s4')

                #Adding host (host connected to the left
                    switch)
                h1Left = self.addHost('h11')
                h2Left = self.addHost('h12')
                h3Left = self.addHost('h13')

                #Adding host (host connected to the
                   right switch)
                h1Right = self.addHost('h21')
                h2Right = self.addHost('h22')
                h3Right = self.addHost('h23')

                #Adding connection (connection host to
                   switch and switch to top switch)
```

```
                    #Connectig 3 host to the left switch
                    self.addLink(h1Left, switchLeft)
                    self.addLink(h2Left, switchLeft)
                    self.addLink(h3Left, switchLeft)

                    #Connecting 3 host to the right switch
                    self.addLink(h1Right, switchRight)
                    self.addLink(h2Right, switchRight)
                    self.addLink(h3Right, switchRight)

                    #Connecting switch (topLeft to all the
                        others)
                    self.addLink(switchLeft, switchRight)
                    self.addLink(switchLeft, switchTopLeft)
                    self.addLink(switchLeft, switchTopRight)

                    #Connecting switch (topRight to just s3
                        and s4)
                    self.addLink(switchRight, switchTopLeft)
                    self.addLink(switchRight, switchTopRight
                        )
topos = {'mytopology': ( lambda: MyTopology() )}
)
```

Questo codice crea rete in Figura 4.7.

4.5.3 Problema dei loop - Spanning Tree Protocol

Quando si creano reti più complesse[10][11], contenenti diversi switch collegati tra loro (ad esempio quella appena mostrata) possono nascere alcuni problemi. Uno di questi problemi sono le ridondanze: esistono più percorsi possibili verso una destinazione. È qui che possono insorgere dei loop; uno switch, infatti, conosce gli indirizzi MAC degli host connessi su ogni segmento, ma se riceve un pacchetto con destinazione sconosciuta, o un pacchetto broadcast, lo invia su tutti i segmenti, tranne che su quello di provenienza. Se esiste un ciclo nella rete, il pacchetto raggiungerà nuovamente il segmento da cui è partito, venendo nuovamente replicato. Questo porterebbe alla proliferazione di infinite copie dello stesso pacchetto sulla rete, e quindi alla saturazione della rete stessa.

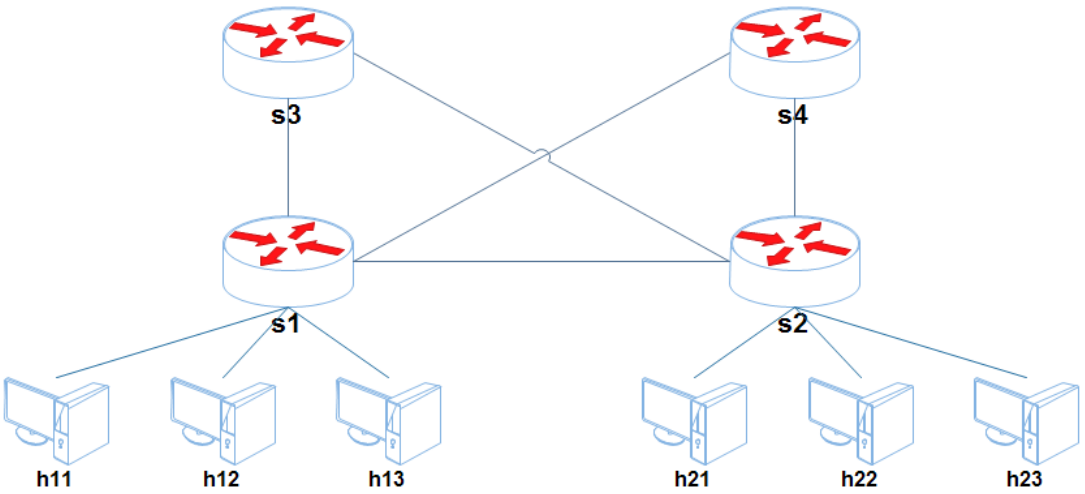


Figura 4.7: Topologia specificata nell'esempio.

Lo scopo del protocollo STP è di evitare l'insorgere di loop all'interno di una LAN. Il principio di funzionamento è quello di bloccare alcune porte in modo da utilizzare un unico percorso per ogni destinazione[10].

Una volta creata la rete, per scongiurare la nascita di loop e quindi per attivare lo STP viene eseguito, per ogni switch, un comando apposito:

$ sudo ovs-vsctl set Bridge s1 stp_ enable=true

che nel caso in esempio riguarda lo switch denominato *s1*. Oppure si può creare uno script per la shell così si possono eseguire tutti i comandi per attivare lo STP in tutti gli switch in modo automatico. Una volta creato un file vuoto con estensione *.sh* basta che sulla prima riga si inserisca l'intestazione *#!/bin/bash*[11] e per ogni altra riga si metta un singolo comando; uno per ogni switch. Ed infine usare il comando:

$ sudo sh ./enable_ stp.sh

per eseguire tutti i comandi scritti nello script in una volta sola.

[10] `http://it.wikipedia.org/wiki/Spanning\_tree\_(networking)`

[11] Che indica alla shell quale programma utilizzare per interpretare lo script una volta eseguito.

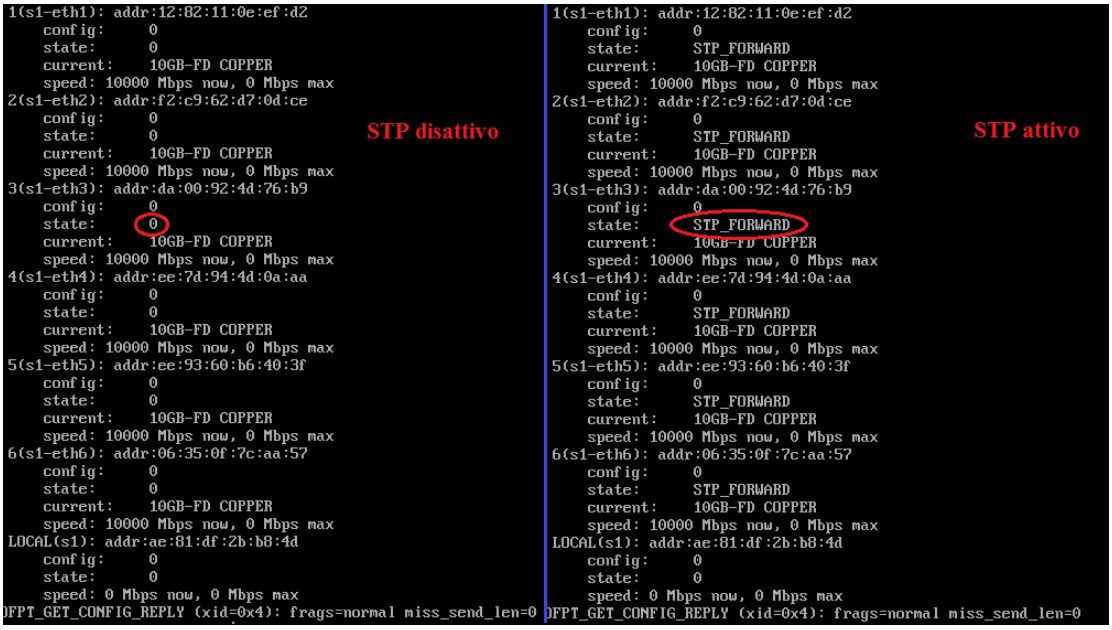


Figura 4.8: Spanning Tree Protocol disattivato e attivato.

Nella pratica, il problema dei loop si può testare direttamente sulla macchina virtuale con Mininet. Dopo aver creato la topologia di rete e dopo aver aperto un end-host a caso della rete, si osserva che, se si prova ad usare il comando ping, esso non funziona e ritorna la stringa: *Destination Host Unreachable*. Questo per il fatto appena descritto. Se invece si esegue lo script e si attende qualche secondo, per poi riprovare il ping, si osserva che tutto funziona come dovrebbe; il ping va a buon fine.

Un altro modo per verificare se il protocollo è attivo o meno è tramite l'utilizzo del comando

$ sudo ovs-ofctl show s1

Questo è uno strumento a riga di comando per il monitoraggio e la gestione degli switch OpenFlow. Nella Figura 4.8 viene riportato l'output nel caso di STP attivo e non.

Prima di lanciare lo script che abiliti lo STP si osserva che nello stato delle interfacce dello switch non compare nulla; successivamente assume lo stato

di *STP_ LEARN* ed infine lo stato *STP_ FORWARD* una volta abilitato il protocollo.

4.5.4 Scelta del controller

Se non viene specificato da linea di comando, alla rete viene associato il controller di default che non fa altro che rendere gli switch dei MAC learning ma, a differenza degli switch ethernet tradizionali, non hanno lo Spanning Tree Protocol abilitato.

Se invece si volesse creare un Controller personalizzato, lo si deve posizionare nella cartella */home/mininet/pox/pox/forwarding*, scriverlo in Python e, in questo caso a differenza delle direttive per la creazione di una topologia, le direttive sono molto più numerose e molto più articolate[12].
Tramite queste direttive è possibile verificare l'IP destinazione o l'IP sorgente di un pacchetto, filtrarlo, duplicarlo inoltrando una copia ad un altro terminale, ecc..

Dopo che si è creato il controller, si crea tutta la topologia con il comando suddetto e con altre due nuove opzioni:

$ sudo mn - -custom topology.py - -topo=mytopology - -controller remote - -switch ovsk[13]

dove:

- l'opzione - *-controller remote* indica che il controller da utilizzare per la nuova topologia non è quello di default ma è uno esterno;
- l'opzione - *-switch ovsk* indica che gli switch da implementare sono gli OpenVSwitch cioè quelli da programmare esternamente.

Successivamente ci si sposta nella cartella */home/mininet/pox/* e si esegue il controller con questo comando:

$ ./pox log.level –DEBUG pox.forwarding.myController

nel caso in cui il file contenente il controller si chiami *myController.py*.

[12]Per informazioni sulle varie direttive e sulle strutture utilizzate si rimanda alla specifica 1.4.0 di OpenFlow: `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf`

[13]I meno del comando sono attaccati.

4.5.5 Alternativa al controller

Invece di utilizzare un controller scritto in Python come appena descritto, è possibile specificare direttamente le azioni che ogni switch deve eseguire sul traffico, direttamente da linea di comando. Grazie a ciò si possono iniettare direttamente le regole, magari con l'ausilio di uno script, evitando così di scrivere del codice. Innanzitutto, prima di scrivere da linea di comando le regole, ci si deve assicurare che la topologia di rete sia attiva altrimenti ritornerà un errore del tipo: "Non so cosa sia il dispositivo s1". Dopo aver attivato la topologia (ad esempio su Windows attraverso PuTTY) è possibile scrivere il codice direttamente all'interno della macchina virtuale di Mininet. Per ogni comando si ha bisogno dei privilegi di super-user quindi si antepone ad ogni comando, *sudo*.

Come prima cosa si deve richiamare il programma per il monitoraggio e la gestione degli switch OpenFlow. Questo tramite il seguente comando (con relativa sinossi):

$ sudo ovs-ofctl [options] command [switch] [args...]

Tramite esso è possibile mostrare lo stato attuale di uno switch OpenFlow, le funzioni e le entry della Flow-Table.
N.B: I comandi che verranno specificati di seguito sono quelli che si sono utilizzati per realizzare il progetto o che si ritiene siano importanti ai fini dell'elaborato.

4.5.6 ovs-ofctl - Gestione degli switch

Di seguito sono indicati i comandi per la gestione degli switch[18]:

- **show** *s*: stampa a video le informazioni riguardanti lo switch *s*, incluse le informazioni delle Flow-Table e delle porte. Esempio:

  $ sudo ovs–ofctl show s1

- **dump-ports** *s [port]*: stampa delle statistiche a console per i dispositivi di rete connessi con lo switch *s*. È possibile specificare anche il numero di porta specifico dal quale si vuole ricavare informazioni. Esempio:

$ sudo ovs–ofctl dump–ports s1 3

- **dump-flows** *s [flows]*: visualizza nella console tutte le flow entry della flow table dello switch *s* che corrispondono alla sintassi del flusso *flows* indicato. Se la seconda opzione viene omessa, vengono restituite tutte le flow entry dello switch indicato. Per vedere come specificare una certa tipologia di flusso, guardare il Paragrafo 4.5.8.

4.5.7 ovs-ofctl - Gestione delle Flow-Table

Questi comandi gestiscono le Flow-Table all'interno di uno switch. In questo caso *flow* specifica una Flow-Entry descritte nel prossimo paragrafo.

- **add-flow** *switch flow*: aggiunge una flow-entry allo switch *switch*. Esempio:

  $ sudo ovs–ofctl add–flow s1 (flow)

- **del-flow** *switch*: elimina una flow-entry allo switch *switch*.

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

4.5.8 ovs-ofctl - Sintassi dei flussi

In questo paragrafo si descrivono i vari flussi. Queste descrizioni comprendono una serie di assegnazioni del tipo *field=value* separate da una virgola o spazi bianchi. Le seguenti assegnazioni dei campi descrivono come un flusso corrisponde a un pacchetto. Se una di queste assegnazioni è omessa dalla sintassi del flusso, al campo viene assegnato un valore di default.

- **in_ port** *port*: specifica i flussi che entranti dalla porta *port*;

- **nw_ src**=*ip*: indica i flussi i quali hanno come *network source* (IP sorgente) il dispositivo avente *ip* come IP, che può essere specificato tramite indirizzo IP o tramite il nome dell'host (ad esempio: "www.google.it");

- **nw_ dst**=*ip*: indica i flussi che hanno come *network destination* (IP di destinazione) il dispositivo avente *ip* come IP, che può essere specificato tramite indirizzo IP o tramite il nome dell'host (ad esempio: "www.google.it");

- **tp_ src**=*port*: indica la porta sorgente dalla quale il flusso è stato spedito (numero compreso tra 0 e 65535);

- **tp_ dst**=*port*: indica la porta destinazione con lo stesso range di valori del campo precedente;

- **dl_ type**=*ethertype*: *ethertype* è un campo composto da due ottetti del frame ethernet utilizzato per incapsulare il protocollo utilizzato nel payload del frame. I più utilizzati sono i seguenti:

  arp : 0x0806

  icmp : 0x0800, *nw_ proto* = 1

  tcp : 0x0800, *nw_ proto* = 6

  udp : 0x0800, *nw_ proto* = 17
  dove il campo *nw_ proto* corrisponde al numero di protocollo IP utilizzato nel campo *protocol* del header del pacchetto IP. Quindi è utilizzato per riconoscere il particolare protocollo utilizzato nel caso in cui il *dl_ type* sia lo stesso.

- è possibile specificare il protocollo che i flussi devono soddisfare per poter essere selezionati; alcuni di questi protocolli sono: ip, tcp, udp, arp;

- **idle_ timeout**=*seconds*: specifica il numero di secondi dopo i quali l'azione, se non utilizzata in questo lasso di tempo, viene eliminata dalla flow table.

- **priority**=*value*: specifica la priorità che ha una entry. Se sono presenti due entry all'interno della flow table che corrispondono allo stesso flusso (ma magari con azioni diverse), sarà applicata l'azione con priorità maggiore. Il range delle priorità va da 0 a 65535 compresi. Se ad una entry non si specifica la priorità, essa viene inserita di default con valore 32768. Se due entry corrispondono allo stesso flusso ma una di queste contiene meno caratteri jolly, ad essa corrisponderà il valore di priorità 65535 perché fa match con più campi. Nel caso in cui due entry abbiano la stessa priorità e corrispondano allo stesso flusso con gli stessi campi jolly, la specifica OpenFlow indica che verrà applicato un comportamento indefinito nella gestione di ciò.

Quindi se si volesse aggiungere un flusso tra quelli presenti nella Flow-Table dello switch *s1*, che soddisfano il protocollo ARP e che provengono dall'host avente IP 10.0.0.3 si deve scrivere il seguente comando:

$ sudo ovs–ofctl add–flows s1 arp,nw_ src=10.0.0.3

Però si vuole anche aggiungere una qualche azione per i flussi che abbiamo appena specificato con i suddetti comandi. Per fare ciò si definiscono le azioni da svolgere su questi flussi, subito dopo averli selezionati aggiungendo l'opzione "actions=". Riprendendo l'esempio appena descritto, si dovrebbe aggiungere:

sudo ovs–ofctl add–flows s1 arp,nw_ src=10.0.0.3,actions=...

Una lista delle possibili azioni viene indicata nel prossimo paragrafo.

4.5.9 ovs-ofctl - Azioni

Alcune delle azioni che si possono svolgere, e quelle utilizzate nel caso di studio sono le seguenti:

- **output:** ***port***: per indicare da che porta dello switch deve uscire il pacchetto.
- **in_ port**: fa uscire il pacchetto dalla porta che lo ha ricevuto.
- **mod_ nw_ src:*IP***: indica che l'indirizzo IP sorgente è *IP*.
- **mod_ nw_ dst:*IP***: indica che l'indirizzo IP di destinazione è *IP*.
- **drop**: scarta il pacchetto in modo tale che non possa essere processato o inoltrato nella rete.

Se invece di lavorare con indirizzi IP si volesse lavorare a livello 2 con indirizzi MAC allora non si deve fare altro che modificare da "nw" a "dl" nelle varie condizioni, specificando ovviamente gli indirizzi MAC:

- **mod_ dl_ src:*MAC***: indica che l'indirizzo MAC sorgente è *MAC*.
- **mod_ dl_ dst:*MAC***: indica che l'indirizzo MAC di destinazione è *MAC*.

Oppure, se si volesse lavorare a livello 4 con i protocolli TCP e UDP:

- **mod_ tp_ src:*port*** indica che la porta sorgente è *port*.
- **mod_ tp_ dst:*port*** indica che la porta di destinazione è *port*.

Esempio: se si volesse aggiungere un flusso tra quelli presenti nella Flow-Table dello switch *s1*, che soddisfano il protocollo IP, che va verso l'host avente IP 10.0.0.4 e che, come azione, esca dalla porta numero 6 dello switch, allora si dovrebbe scrivere il seguente comando:

sudo ovs–ofctl add–flows s1 ip,nw_ dst=10.0.0.4,actions=output:6

Per ulteriori informazioni sui comandi indicati e per conoscerne tanti altri che è possibile utilizzare, è possibile visitare questo utile sito: `http://openvswitch.org/cgi-bin/ovsman.cgi?page=utilities\%2Fovs-ofctl.8`.

Capitolo 5

Caso di studio

Una volta arrivati a questo punto, si ha ben chiaro quelli che sono gli aspetti fondamentali dell'elaborato, cioè:

- il paradigma di Software Defined Networking con la relativa architettura, i componenti principali con il loro ruolo all'interno di queste reti;
- il protocollo OpenFlow il quale permette di programmare gli switch di rete mostrando anche alcuni esempi di utilizzo;
- alcuni aspetti delle problematiche riguardo alla sicurezza all'interno di reti con questo approccio;
- gli strumenti che si sono studiati, le piattaforme utilizzate e tutti i vari comandi da avvalersi per svolgere determinate funzioni.

In questo capitolo si passa all'aspetto più pratico. Si cerca di implementare una semplice topologia di rete all'interno della quale si vuole attuare un sistema di sicurezza minimale. Per fare ciò si è utilizzata la macchina virtuale di Mininet per poter usufruire delle sue funzionalità circa la programmabilità (tramite OpenFlow) degli switch di rete con tutti i relativi comandi per creare topologie personalizzate.

5.1 Descrizione

Il sistema deve "discriminare" il comportamento dello switch di rete sulla base del tipo di traffico che giunge in prima istanza ad esso. Si parte dalla situazione

nella quale si ha un OpenVSwitch (quindi programmabili con i comandi o tramite controller) a cui da un lato è connessa una sorgente del traffico e dall'altro lato abbiamo un host che funge da destinazione. L'idea è quella di avere, all'interno della stessa rete una macchina che svolge il compito di DPI[1] sulla quale vengono ridiretti i pacchetti appartenenti ad un flusso che, ad esempio, lo switch non ha mai visto. Quindi, a fronte di un nuovo flusso, lo switch invia il pacchetto al DPI per controllare se il traffico che sta per passare è consono o meno al traffico che l'amministratore di rete ritiene corretto.

L'obiettivo è quello di riconoscere i flussi di traffico del protocollo SSH e riuscire a dropparli, installando sullo switch una regola che esegua questo.

5.1.1 Esempio dimostrativo: reti peer-to-peer

Supponiamo di essere in un contesto di lavoro in cui il traffico tipico che passa attraverso lo switch sia inerente alle azioni normali che svolge un utente nell'azienda. In questo caso l'azione che viene fatta sul pacchetto sarà quella di mandarlo effettivamente dalla sorgente alla destinazione. Invece, si suppone che il traffico in arrivo venga riconosciuto come estraneo e che lo switch non lo abbia mai visto. Per esempio, se il dipendente sta cercando di scaricare un film da torrent, allora, quando il traffico arriva al DPI, questo riconosce in qualche modo che è una comunicazione peer-to-peer (p2p) perché magari osserva che utilizza determinate porte (ad esempio, potrebbero essere le porte che utilizza eMule) e, siccome è un traffico che non si vuole far passare, si decide di scartare il pacchetto. Quindi l'azione da applicare a questo tipo di pacchetti sarà *drop*.

5.1.2 Topologia creata

La topologia di rete che è stata utilizzata per testare la funzionalità del sistema è semplice. Essa è composta da tre host collegati ad uno switch al quale vengono iniettate le azioni che il controller gli indica attraverso il protocollo OpenFlow. Nella Figura 5.1 viene mostrata la semplice topologia.

[1]Deep Packet Inspection: è una forma di filtraggio dei pacchetti di rete che esamina la parte di dati (ed eventualmente anche l'intestazione) di un pacchetto che passa un punto di controllo, alla ricerca di virus, spam, intrusioni, o criteri per decidere se il pacchetto può passare, se ha bisogno di essere indirizzato a una destinazione diversa o per lo scopo di raccogliere informazioni statistiche.

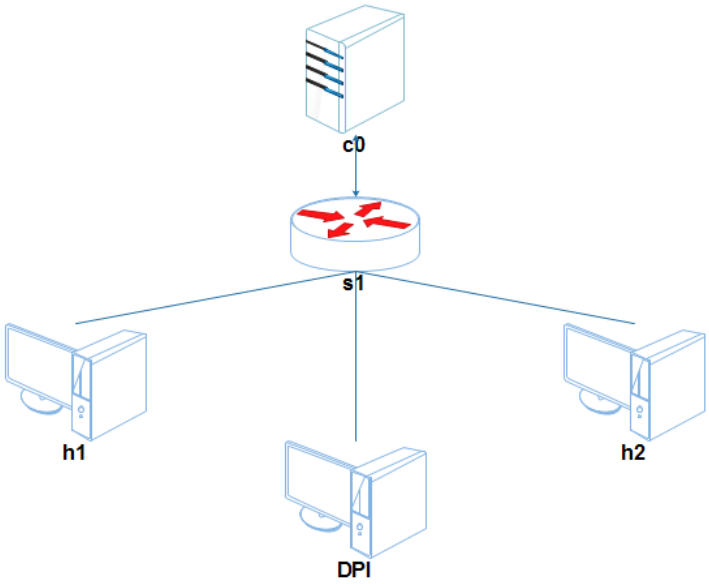


Figura 5.1: Topologia realizzata per il caso di studio.

h1 e *h2* sono due semplici host che si scambiano dei dati con i vari protocolli, mentre *DPI* è un particolare host che, grazie alle azioni installate nello switch, è in grado di ricevere gran parte dei dati che gli altri due host connessi si scambiano. Il codice per realizzare questa topologia è in Appendice B.1.
Dopo aver testato il corretto funzionamento di ogni cosa, simulando la topologia sulla macchina virtuale di Mininet con il controller di default, si è passati a specializzare il DPI per consentirgli di svolgere il suo compito.

5.2 DPI e nDPI

All'interno della macchina virtuale di Mininet si è installato un tool per consentire al DPI di osservare il traffico: *nDPI*[21][22]. nDPI consente il rilevamento, a livello applicativo, dei protocolli, indipendentemente dalla porta utilizzata. Ciò significa che è possibile rilevare sia protocolli noti su porte non standard (come ad esempio il protocollo HTTP su una porta diversa da 80) sia rilevare anche il contrario (ad esempio, il rilevamento di traffico Skype sulla porta 80).

5.2.1 Download ed installazione

Se si vuole scaricare ed installare nDPI sulla propria macchina, si devono seguire pochi semplici passi. Da linea di comando di Mininet si fa il download della cartella contenente tutti i file che servono:

svn co `https://svn.ntop.org/svn/ntop/trunk/nDPI`

. Una volta terminato il download, ci si sposta nella cartella appena creata (*cd nDPI*) per poter lanciare il file *configure*. Prima però lo si deve generare attraverso l'utilizzo di un altro file chiamato *configure.ac*. Senza troppe spiegazioni in merito, si crea il file che ci interessa attraverso l'utilizzo del seguente comando:

autoreconf -ivf

in tutte le cartelle che si andranno ad utilizzare. Nel nostro caso si andrà ad utilizzare solo un'altra cartella: *nDPI/example*. Successivamente, eseguiamo tutti i file *configure* che si sono generati:

./configure -with-pic

e

make

.

5.2.2 Utilizzo

Dopo aver installato il tool, ci si sposta nella cartella creata dopo il download; più precisamente nella cartella nDPI/example. Sempre da linea di comando si digiti

sudo ./ndpiReader

per poter visualizzare tutte le opzioni disponibili che si possono utilizzare per categorizzare il traffico come meglio si vuole. Le opzioni che si sono usate nel caso di studio sono le seguenti:

* ***-i device*** utilizzata per specificare da quale interfaccia di rete controllare il traffico dei dati (nel caso specifico è l'interfaccia del DPI connessa allo switch);

* ***-s n_ seconds*** che indica per quanti secondi si vuole che il tool funzioni;

* ***-v2*** significa che si vuole l'output del programma molto approfondito;

* ***-j file.json*** utilizzato per salvare l'output che viene generato in un file json, quindi gestibile facilmente da programmi in Python.

Quindi, il comando che viene utilizzato all'interno del DPI è il seguente:

sudo ./ndpiReader -i DPI–eth0 -s 6 -v2 -j /home/mininet/prove/file.json

Il tool viene fatto funzionare per 6 secondi dopodiché l'output che si genera viene salvato in un file json. Questo file viene aperto con Python (importando il modulo chiamato *json*) e vengono letti tutti i pacchetti, salvando, in ogni riga di in un file di testo (chiamato *pacchettiDPI.txt*), i dati relativi ad un flusso. Più precisamente, ogni riga contiene i seguenti campi:

1. Il protocollo di rete utilizzato (ad esempio, TCP);

2. Il protocollo che nDPI ha categorizzato (tra quelli che riesce a supportare, sono circa 170), come ad esempio Skype;

3. Host sorgente del flusso di dati;

4. Porta dell'host sorgente dalla quale proviene il flusso;

5. Host di destinazione di tale flusso;

6. Porta dell'host di destinazione alla quale il flusso dovrebbe giungere.

Ogni campo viene separato dal successivo con due punti ":". Un esempio di riga del file di testo è questa:

TCP:Skype:192.168.159.137:31099:157.56.126.204:37466

Tutto ciò viene ripetuto per 60 secondi dopodiché il programma termina. Questo programma è possibile trovarlo in Appendice B.2.

5.3 Ruolo del controller

Il ruolo fondamentale di tutto il progetto lo assume il controller. Il suo codice si può trovare in Appendice B.3. Inizialmente quello che fa è semplice: fintanto che i pacchetti scambiati sono degli ICMP (quindi dei ping), il controller inietta allo switch una regola che permette a questi pacchetti di giungere a destinazione. Invece, se sono di un'altra natura, viene iniettata l'azione che fa in modo di deviare i pacchetti verso l'host DPI. Questa deviazione si può eseguire perché il controller conosce le porte a cui gli host sono connessi; in particolare nella porta 1 è presente h1, nella 2 è presente h2 e nella 3 è presente il DPI. Quindi per deviare un pacchetto che ha come destinazione, per esempio, h1, basta che venga iniettata la regola che impone a quel traffico di non essere più inoltrato verso la porta 1 ma verso la porta 3. Successivamente, dopo che il controller ha deviato i flussi sospetti verso il DPI, quest'ultimo li elabora come descritto precedentemente.

5.3.1 Il thread al suo interno

Oltre a questo, all'interno del controller viene generato e fatto partire un *thread* (generato dalla classe *MyThread*) che svolge un'azione molto importante: inserisce delle nuove regole nella flow table dello switch. Più precisamente, il thread esegue queste operazioni:

1. Per prima cosa, dopo aver atteso 6 secondi, apre il file di testo *pacchettiDPI.txt* che ha generato il DPI durante le sue osservazioni del traffico;

2. Se nel file di testo sono presenti dati utili, il thread legge una riga alla volta e salva i vari campi che incontra in un array. Per fare ciò utilizza la funziona *string.split(str, :)* specificata in Appendice A.4.

3. Se i dati che legge sono utili, cioè se i vari flussi utilizzano dei protocolli per i quali si vogliono aggiungere delle azioni, allora si prosegue altrimenti il thread rincomincia il suo ciclo.

4. Successivamente viene gestito il file di testo *action.txt*. Questo file viene utilizzato per interagire direttamente con il controller. Al suo interno sono presenti delle regole che il thread ha creato e che il controller ha iniettato nello switch *s1* fino a quel momento. Anche in questo file, come

in *pacchettiDPI.txt* viene specificata, in ogni riga, un'azione con tutti i campi che servono. In questo caso i campi sono separati da uno spazio vuoto invece che da ":". Ogni riga contiene i seguenti campi:

a. Azione svolta sui flussi di quel tipo: può essere "DROP" o "FORWARD";
b. Il protocollo utilizzato dei flussi: ad esempio "SSH";
c. Host sorgente rappresentato dal suo IP;
d. Porta dell'host sorgente (la stessa specificata nel file *pacchettiDPI.txt* descritto in precedenza);
e. Host destinazione rappresentato nello stesso modo di quello sorgente;
f. Porta dell'host destinazione.

Se nel file *action.txt* non è presente nessun dato, non si fa altro che scrivere al suo interno la nuova azione che si vuole iniettare. Invece, se sono già presenti delle azioni, il thread deve controllare se quel tipo di azione tra quei due host è già presente o meno. Se è presente, non modifica il file e non inietta nessuna azione mentre se non è presente, scrive in modalità *append* la nuova azione, rispettando il formato descritto.

5. Dopo che si sono scritte le regole nel file *action.txt* le si vogliono rendere effettivamente operative all'interno dello switch. Per fare ciò si è creata una semplice funzione che viene richiamata ogni volta che si vuole aggiungere un'azione: *inject_ action*. I parametri che si passano a questa funzione sono l'host sorgente, l'host destinazione, l'azione che si deve iniettare e le due porte. Questa funzione non fa altro che eseguire l'azione inviandola allo switch. Nel caso l'azione fosse "DROP", si fa in modo che l'azione iniettata non faccia arrivare a destinazione il flusso. Questo viene fatto indicando come porta di output dello switch per quel flusso, non la porta collegata all'host di destinazione, ma una porta alla quale non è connesso nessun host. Questo però non è abbastanza perché, così facendo, si hanno due azioni che vogliono gestire lo stesso flusso e si viene a creare un problema. Serve un modo per indicare allo switch quale di queste azioni eseguire. Per fare in modo che una entry della flow table prevalga su un'altra, si è impostato un livello maggiore di priorità (campo *priority* spiegato nel Capitolo 4.5.8) cosicché quando arriva un'azione con più alto valore di priorità, essa verrà eseguita e l'altra, con priorità minore, non

viene eseguita. Viene impostato anche il valore del campo *idle_ timeout* per i flussi che vengono deviati verso il DPI; questo fa in modo che, nel caso in cui il thread installasse delle nuove azioni (con priorità maggiore) su questi stessi flussi, le azioni con il timeout, dopo un certo numero di secondi, vengano rimosse.

5.3.2 Problemi riscontrati e risolti

Durante la fase di sviluppo del codice, ci si è imbattuti in qualche problema.

Uno di questi problemi è il fatto che il thread all'interno del controller non è in grado di "vedere" le variabili contenute all'interno del flusso principale del controller appunto. Per rimediare a ciò, si è deciso semplicemente di rendere globali quelle variabili e quei valori che servono al thread per eseguire correttamente il suo compito. Questi parametri non si possono passare come argomenti perché variano con il tempo e si possono passare una volta sola.

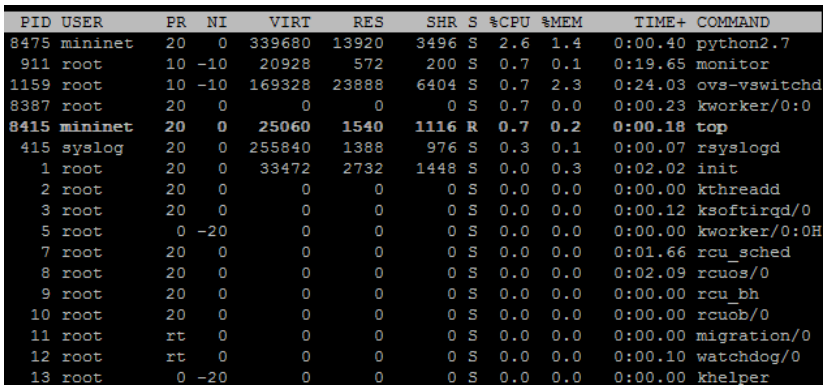
Un altro problema si è riscontrato quando si è cercato di aprire un file del quale non si hanno i permessi. Per questo si è utilizzata la funzionalità contenuta all'interno del modulo *os*:

os.chmod(/home/mininet/prove/action.txt, 0777)

In questo modo si attribuiscono i permessi di lettura, scrittura ed esecuzione a tutte le tipologie di utente.

Un problema, si è riscontrato nella funzione del thread *inject_ action(...)* per il fatto che, nel campo *nw_ dst* (o anche *nw_ src*) del messaggio che si prepara per mandare allo switch, il tipo di variabile richiesto è *IPAddr*. Per risolvere questo problema si è deciso di creare due array (*dst_ ip_ array* e *src_ ip_ array*) che contenessero al loro interno i valori *IPAddr* degli host, in modo da usarli quando si deve preparare un messaggio.

Infine l'ultimo problema si è riscontrato nell'utilizzo del tool *ndpiReader*. Nel file json che viene generato, contenente l'output con tutti i flussi rilevati, che poi verrà analizzato dal thread del controller, non viene specificato quale, tra il primo ed il secondo IP di un flusso, è la sorgente e quale la destinazione. Quindi non si è in grado di trovare immediatamente quale è la sorgente. Per rimediare a questo problema si è deciso di fare una scelta mirata per il nostro caso specifico, ovvero per i flussi SSH. Si è deciso di osservare le porte sorgente e destinazione. Se la porta è la numero 22 (porta standard SSH), allora sicuramente l'IP associato a quella porta sarà la destinazione della richiesta SSH. In questo caso si dovranno droppare i flussi che hanno come destinazione questo indirizzo IP.



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
8475	mininet	20	0	339680	13920	3496	S	2.6	1.4	0:00.40	python2.7
911	root	10	-10	20928	572	200	S	0.7	0.1	0:19.65	monitor
1159	root	10	-10	169328	23888	6404	S	0.7	2.3	0:24.03	ovs-vswitchd
8387	root	20	0	0	0	0	S	0.7	0.0	0:00.23	kworker/0:0
8415	**mininet**	**20**	**0**	**25060**	**1540**	**1116**	**R**	**0.7**	**0.2**	**0:00.18**	**top**
415	syslog	20	0	255840	1388	976	S	0.3	0.1	0:00.07	rsyslogd
1	root	20	0	33472	2732	1448	S	0.0	0.3	0:02.02	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.12	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
7	root	20	0	0	0	0	S	0.0	0.0	0:01.66	rcu_sched
8	root	20	0	0	0	0	S	0.0	0.0	0:02.09	rcuos/0
9	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
10	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuob/0
11	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
12	root	rt	0	0	0	0	S	0.0	0.0	0:00.10	watchdog/0
13	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	khelper

Figura 5.2: Utilizzo della cpu tramite il comando *top*.

5.4 Osservazioni

- Come si può notare dalla Figura 5.2, il consumo della CPU è molto ridotto; questo è dovuto al fatto dell'utilizzo del thread.

- Invece nella Figura 5.3 e nella Figura 5.4 è possibile notare, attraverso l'uso del comando

  sudo ovs–ofctl dump–flows s1

  che mostra tutte le flow entry dello switch *s1*, i due momenti di interesse. Nella prima figura si osserva che, quella evidenziata di colore rosso, è la flow entry che verrà eliminata dopo un timeout di 200 secondi nel caso in cui rimanga inattiva. La prima flow entry della prima figura evidenzia l'azione di drop per i flussi che hanno come destinazione l'IP 10.0.0.3 (*h2*); per dropparla si è deciso di fare uscire quel flusso dalla porta 5 dello switch, che non è collegata a nessun host. Nella seconda figura invece, si osserva che sono trascorsi più di 200 secondi (che è il timeout scelto per questo esempio) e come si può notare, la flow entry che aveva quel timeout è scomparsa.

- Una domanda che può sorgere, riguardante il motivo dell'utilizzo di un file di testo per salvare tutte le regole che il controller ha installato sullo switch *s1*, è la seguente: non può il controller, tenere traccia al suo

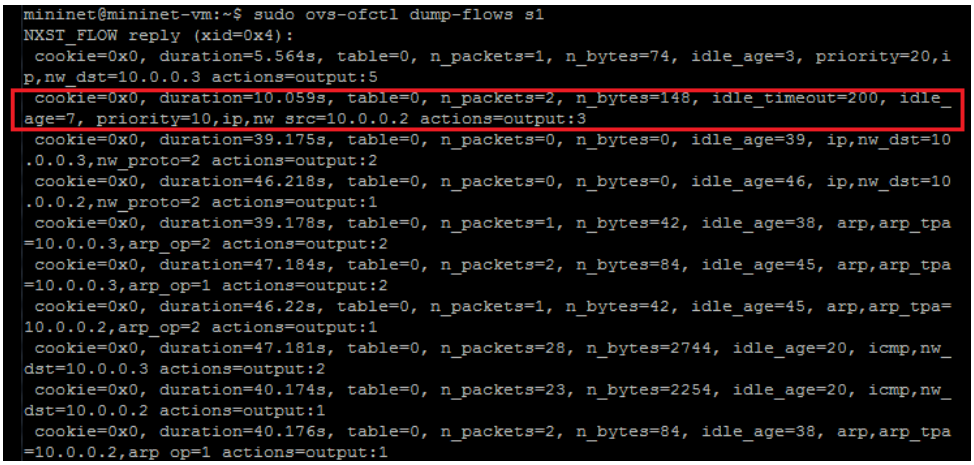


Figura 5.3: Lista regole installate nello switch *s1* (1).

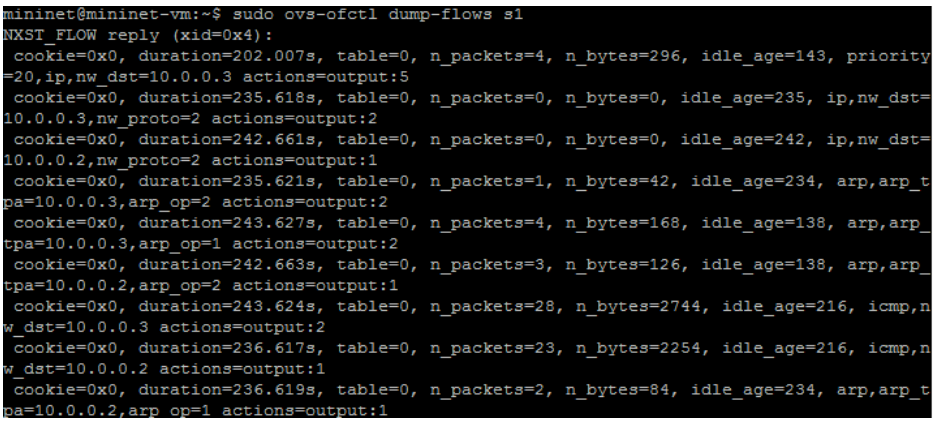


Figura 5.4: Lista regole installate nello switch *s1* (2).

interno, delle regole che aveva installato precedentemente per poi poterle aggiornare? Da quello che si è osservato, il controller può solo richiedere statistiche di traffico agli switch per cui non sembra sia presente una vera e propria lista di regole. Per questo si è optato per l'utilizzo di un file di testo. In questo caso risulta comodo utilizzare questo file perché, essendo sia il controller che il DPI sulla stessa macchina fisica, possono facilmente accedere a un file locale. Se fossero stati su macchine diverse connesse nella rete, si sarebbe dovuto implementare una funzionalità che facesse in modo di connettere queste due macchine tra loro per poter scambiarsi informazioni.

Capitolo 6

Conclusioni

Questa tesi aveva come obiettivo la sperimentazione della capacità del paradigma SDN di fornire certe funzionalità per la sicurezza. Dopo aver studiato questo paradigma e dopo aver indicato alcuni aspetti della sicurezza che si sono studiati in letteratura, si è passati alla pratica, imparando ad utilizzare gli strumenti base per creare e gestire queste particolari reti. Tutto ciò è stato fatto per riuscire a creare un sistema di sicurezza di base.

Per lo sviluppo di tale sistema, si sono usati strumenti efficaci di emulazione che hanno permesso di comprendere i problemi più rilevanti. Mininet è uno di questi strumenti. È molto efficace in quanto permette di emulare topologie di rete di cui fanno parte host, switch, controller e collegamenti virtuali, consentendo di sperimentare possibili implementazioni di applicazioni OpenFlow oriented e instradamento personalizzato, altamente flessibili. Offre un ottimo strumento per cercare di implementare reti SDN. Questa flessibilità è un cavallo di battaglia di questo approccio, ma ciò significa che, avendo l'amministratore il potere di programmare tutta la rete, egli deve pianificare attentamente tutta la gestione del traffico. Tutto questo richiede una programmazione molto attenta del controller, che è il cuore dell'architettura.

Una delle maggiori difficoltà è stata la gestione delle azioni da implementare, la quale, da risolvere, si è dimostrata più complessa del previsto. Poiché il piano di controllo è all'interno del controller, se non dovesse funzionare o fosse stato programmato in maniera errata, la rete non svolgerebbe il suo ruolo dal momento che è il controller a stabilire le regole. Questo significa che colui che programma il controller, deve sapere esattamente come gestire tutti i pacchetti che riceve dagli apparati ad esso connessi. Tale gestione però, non è immediata.

Nel caso di studio si è vista una topologia di rete costituita da un unico switch e da un solo controller; già in questo caso si sono avute delle difficoltà nel contemplare tutte le tipologie di traffico che possono presentarsi e tutti i vincoli che vengono imposti dal protocollo OpenFlow. Ma se si prova ad allargare gli orizzonti, considerando uno scenario più complesso dove sono presenti un controller e più switch, ci si accorge subito che si deve pianificare il traffico non solo di uno (come nel caso di studio), ma di tutti gli switch presenti. Se volessimo complicare ancora di più la questione, si potrebbero avere più controller (caso reale), anziché uno, ed essere poi in grado di coordinarli tutti insieme.

In linea di massima i risultati sono stati raggiunti, quindi si può affermare che l'approccio SDN può rappresentare un forte potenziale di innovazione nelle reti odierne e che, come approfondito dallo sviluppo della tesi, si possono applicare molte tipologie di sicurezza atte alla salvaguardia degli utenti all'interno delle stesse.

Appendice A

Cenni alla programmazione in Python

In questa appendice si vedranno alcuni semplici cenni alla programmazione in Python. Questa non fornisce tutte le conoscenze necessarie affinché si riesca a programmare un controller ma pone le basi e mostra il funzionamento di questo linguaggio.

Python[19][20] è un linguaggio di programmazione dinamico ad alto livello, orientato agli oggetti, utilizzabile per molti tipi di sviluppo software; in particolare per scrivere degli script. È possibile reperirlo in rete, direttamente dal seguente link: `http://www.python.it/download/`. Per la versione Windows verrà scaricato un installer sul quale basta cliccarci due volte per far partire l'installazione. Per la versione di Unix, invece, è presente il codice sorgente da compilare. Di seguito, le spiegazioni sono descritte immaginando di essere in un ambiente Unix-like.

A.1 Hello, World!

Come primo esempio come non poter proporre il famoso "Hello, World". Per implementarlo basta aprire un nuovo documento vuoto e digitare il seguente codice:

```
print ''Hello, World!''
```

e salvare il file con estensione *.py*: per esempio *hello.py*.

A.1.1 Eseguire da linea di comando

Una volta creato il file, per eseguirlo si deve aprire il terminale e spostarsi (con il comando *cd*) nella cartella dove risiede. Una volta fatto ciò, si digita:

python hello.py

per mostrare a video l'output che ci si aspetta. Se invece si specifica come prima linea

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

allora è possibile eseguire il programma omettendo il "python" iniziale. Infine, se come prima istruzione inseriamo

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

e se lo script viene reso eseguibile, allora il sistema operativo utilizzerà quella prima linea per sapere quale interprete utilizzare per l'esecuzione e per l'analisi del resto del file.

A.1.2 Eseguire in modalità interattiva

È anche possibile entrare nella modalità interattiva. Per fare ciò, si deve aprire il terminale e digitare *python*. Così facendo, verrà avviata questa modalità nella quale basta scrivere direttamente la linea di codice contenuta nel file *hello.py* e digitare *Enter* per poter vedere subito l'output. Per uscire da questa modalità si deve digitare *Ctrl+D* o *exit()* o *quit()*. Se aggiungiamo al file un'altra riga con un altro *print*, il testo di questo verrà visualizzato al di sotto della già esistente.

A.2 Espressioni

Per poter visualizzare il risultato di alcune semplici espressioni basta digitare, per esempio:

```
print "Il risultato di 4 + 5: ", 4+5
```

e si vedrà a video:

Il risultato di 4+5: 9

Si possono creare anche espressioni più complesse ed è anche possibile anteporre l'espressione al testo. Python da risposte differenti in base alla presenza o meno di numeri decimali. Se, ad esempio, uno dei due operandi (o entrambi) presenta uno o più numeri decimali, allora il risultato sarà anch'esso un numero decimale. L'ordine gerarchico con il quale vengono eseguite le operazioni è il seguente:

() Espressioni tra parentesi;

** Elevamento a potenza;

*,/,% Moltiplicazione, divisione e resto;

+,- Addizione e sottrazione.

A.3 Commenti e Import

Per inserire un commento basta anteporlo al simbolo #. Il testo che segue, sarà visualizzato solo a fronte dell'apertura del file Python con un editor di testo.

Come nel linguaggio C, dove si utilizzano gli *include*, così in Python le istruzioni di *import* (che vengono indicate all'inizio del codice) servono per caricare un modulo (all'interno del quale sono presenti classi e funzioni) all'interno del programma, in modo tale da rendere possibile l'utilizzo di maggiori funzionalità. Queste funzionalità spaziano dalla gestione delle stringhe all'utilizzo dei thread fino alla capacità di usare i comandi della console di Linux.

A.4 Variabili

Le variabili in Python non hanno bisogno di essere etichettate fin da subito. Il tipo di variabile viene attribuito nel momento in cui la si inizializza. Quindi se ad una variabile si attribuisce il valore 2 allora significa che sarà una variabile intera mentre se le si attribuisce il valore "4 + 5", il suo tipo sarebbe diverso, ovvero una stringa. Per conoscere il tipo di una variabile si può utilizzare la funzione *type(nome_ variabile)*.

Per le stringhe sono presenti diverse funzioni, utilizzabili facilmente; ad esempio:

Concatenazione *"Hello, "+"World!"* == "Hello, World!"

Moltiplicazione *"u"*5* == "uuuuu"

Split Questa funzione può essere applicata se si importa il modulo *string*. La sintassi è la seguente: *s = string.split(str, :)*. Considera la stringa *str* ed ogni volta che trova l'occorrenza del secondo parametro passato alla funzione (nell'esempio il carattere dei due punti), salva in una lista di stringhe (nel nostro caso *s*) i caratteri che ha analizzato fino quell'occorrenza.

A.5 Condizioni

Per prendere le decisioni in base ad una o più condizioni, è possibile utilizzare il costrutto *if...else* nella seguente modalità:

```
if condizione:
    istruzioni
elif condizione:
    istruzioni
else:
    istruzioni
```

Dove la condizione può essere una concatenazione di più istruzioni legate dagli operatori booleani *and* oppure *or*. Il costrutto *elif* è l'abbreviazione di "else .. if" in modo da permettere al programmatore di utilizzare un solo comando per generare un'altra condizione.

A.6 Cicli

Come in ogni linguaggio di programmazione, anche in python sono presenti i cicli condizionati. A differenza di altri linguaggi che usano le parentesi graffe per limitare il codice da eseguire, in questo caso si utilizza solo l'indentazione del testo; quindi si deve stare attenti a come si dispongono le istruzioni.

A.6.1 Ciclo while

La struttura del ciclo while è la seguente:

```
while condizione:
    istruzioni indentate correttamente
```

dove finché la condizione specificata risulta vera, le istruzioni presenti "all'interno" del while vengono eseguite ciclicamente. Se si vuole creare un ciclo infinito basta che si utilizzi il seguente comando: *while 1:*.

A.6.2 Ciclo for

Oltre al ciclo precedente viene spesso usato anche il ciclo *for* con la seguente struttura:

```
for item in range_di_valori:
    istruzioni
```

Anche qui si deve fare attenzione alle istruzioni che siano indentate correttamente. Se si vuole che il campo *range_ di_ valori* faccia ad esempio un ciclo di 10 giri (partendo da 0 e arrivando a 9) si utilizza una comoda funzione di Python chiamata *range(inizio, fine)*. Nel caso di questo esempio, si dovrà utilizzare *range(0, 10)*.

Oltre a semplici valori come quelli prodotti dalla funzione *range(...)* appena descritta, è possibile scorrere, per ogni giro, un elemento di una lista composta da elementi di origine diversa; di seguito viene mostrato un esempio:

```
list = ['cane', 4, 8, 'gatto', 6]
for item in list:
    istruzioni
```

A.7 Gestione dei file

Una funzionalità molto importante è la capacità di creare, aprire e modificare file.

A.7.1 Apertura

Per aprire un file si deve utilizzare la seguente sintassi:

```
var = open("path", "mod")
```

dove *path* indica il pathname assoluto del file all'interno del computer, mentre *mod* indica la modalità con la quale si vuole aprire il file. Più precisamente:

r il file viene aperto in modalità di sola lettura quindi non è possibile modificarlo in alcun modo. Se il file non esiste, viene generato un errore.

r+ il file viene aperto ed è possibile scrivere al suo interno.

w file viene aperto in modalità di scrittura; se il file non è presente, esso viene creato mentre se è presente nel pathname indicato, allora verrà sostituito con il nuovo che si sta per creare.

a il file viene aperto in modalità "append"; ciò significa che, se il file è presente esso viene aperto e viene letto poi, se si vogliono scrivere delle informazioni, verranno aggiunte in fondo al file. Se il file non è presente nella cartella indicata, verrà generato un errore.

a+ simile al precedente solo che se il file non è presente, esso viene creato e aperto nella modalità "append".

A.7.2 Lettura

Se si è aperto il file con una delle modalità dove è possibile leggerlo, sono presenti alcune funzioni che risultano comode per l'uso (nell'esempio supponiamo che la variabile con cui abbiamo aperto il file si chiami *var*):

s = var.read(): nella variabile *s* è presente il contenuto dell'intero file;

s = var.read(N): vengono letti N bytes e salvati della variabile *s*;

s = var.readline(): viene letta una riga e salvata nell'apposita variabile;

s = var.readlines(): vengono lette tutte le righe e vengono salvate nella variabile *s* che diventa una lista di stringhe (quindi in *s[0]* si ha il contenuto della prima riga del file).

A.7.3 Scrittura

Esistono due comandi con i quali è possibile scrivere sul file (sempre usando la variabile *var* per aprire il file):

var.write(str): scrive sul file i valori passati come parametri e ritorna il numero di bytes scritti;

var.writeline(list): scrive la lista list nelle varie righe del file.

A.7.4 Chiusura

Infine, per chiudere correttamente un file aperto, si utilizza:

```
var.close()
```

A.8 Funzioni

Come ormai in tutti i linguaggi di programmazione recenti, anche in Python è possibile definire delle funzioni, utilizzate per non ripetere sempre le stesse operazioni, dove l'unica cosa che cambia sono i valori delle variabili utilizzate. Per definire una funzione si utilizza la parola chiave *def* seguita dal suo nome e dalla lista di variabili da utilizzare al suo interno. Se ad esempio si vuole definire una funzione chiamata *f1* che ha due parametri *a* e *b*, quello che si deve scrivere è:

```
def f1( a, b):
    set di istruzioni indentate
    [return]
```

ed è possibile inserire nelle istruzioni indentate tutte le operazioni verrebbero ripetute nel codice. Per richiamare la funzione all'interno del flusso principale del programma basta inserire il nome della funzione indicando, tra parentesi tonde, i valori o le variabili che si vogliono usare come parametri. è buona regola inserire il "return" alla fine di ogni funzione. Nel caso in cui si vuole che la funzione ritorni un certo valore è obbligatorio specificare tale valore subito dopo, altrimenti, se non si vuole far ritornare nessun valore, basta che si scriva semplicemente "return".

È bene specificare che le variabili all'interno delle funzioni hanno durata pari alla durata della funzione, dopodiché "spariscono". Se ad esempio una variabile locale (ovvero variabile interna alla funzione) ha lo stesso nome di una variabile globale (riconosciuta dall'intero programma), la prima nasconde la seconda all'interno della funzione.

A.9 Thread

Così come in Java, anche in Python è possibile gestire i thread ovvero eseguire concorrentemente diversi flussi sequenziali di istruzioni all'interno di un pro-

gramma. Ognuno di essi ha un set di istruzioni che, una volta avviati, eseguono e terminano, facendo ritornare i propri risultati al programma principale. I thread vengono utilizzati per migliorare le performance di un programma.

Per creare un thread si deve, per prima cosa, importare il modulo *threading*. Successivamente lo si crea e lo si fa partire nel seguente modo:

```
t = threading.Thread(target= name_target)
t.start()
```

dove *name_ target* è il nome della funzione che definisce il comportamento del thread. Quando la funzione giunge al *return* allora il thread termina.

Oltre a questa modalità è possibile utilizzarne un'altra per generare thread; consiste nella creazione di una classe apposita (ad esempio *MyThread*) che definisce tutto il comportamento completo del thread tramite l'uso di apposite funzioni. La classe ha la seguente struttura:

```
class MyThread (threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)

    def other_functions(self, a, b, c):
        istruzioni
    def run(self):
        istruzioni
```

e conseguentemente, per creare un thread che appartiene alla classe *MyThread* basta sostituire il nome della classe in questo modo:

```
t = MyThread()
t.start()
```

invocando il metodo *start*, si manda in esecuzione la funzione *run* della classe, mandandolo in esecuzione.

Se si vuole che il programma principale aspetti la fine dell'esecuzione di un thread (perché ad esempio gli serve l'output che genera) deve chiamare il metodo *join* che svolge questo compito. Infine, se si vuole dare la possibilità ad un thread di leggere (ed eventualmente modificare) una variabile presente nel flusso principale del programma, la si deve dichiarare globale, con l'uso della clausola *global*.

A.9.1 Svantaggi

Oltre a tutti i vantaggi che portano i thread, vi sono anche alcuni svantaggi che se non vengono gestiti in modo efficace possono creare dei problemi. Questi sono:

Deadlock situazione nella quale due thread si bloccano a vicenda in attesa che uno esegua una certa operazione che serve all'altro e viceversa.

Condivisione gestire l'accesso alle risorse condivise: un thread legge un valore sbagliato di una variabile condivisa perché un altro thread non ha aspettato il suo turno per poter eseguire le sue azioni. Per cercare di risolvere a questo problema si utilizzano i semafori.

A.10 Comandi della command line di Linux

In Python è anche possibile utilizzare i comandi che di solito si scriverebbero da linea di comando (ad esempio *cd*, *chmod*, *ls -l*) direttamente dall'interno dei programmi. Ciò significa che è possibile creare e rimuovere file, avviare o fermare dei processi e così via.

Per poter usufruire di questa funzionalità si deve importare il relativo modulo: *import os*. Dopodiché è possibile usufruire delle seguenti funzioni:

os.system("comando command line") esegue il comando specificato nella stringa;

os.path.isfile("pathname file") ritorna *True* se il file indicato nella stringa passata come parametro esiste o *False* se non esiste;

os.chdir("pathname directory") equivale al commando *cd* della command line di Linux; permette di cambiare la directory corrente con quella specificata nella stringa.

A.11 Gestione errori

Per gestire i vari errori (eccezioni) che possono incombere durante l'esecuzione del programma, ad esempio l'apertura in lettura di un file inesistente o la divisione per zero, si utilizza un particolare costrutto che agevola la gestione del problema. Il costrutto è il seguente:

```
try:
    istruzioni "sensibili" che possono causare eccezioni
except:
    operazioni da eseguire se viene lanciata un'eccezione gestibile
else:
    operazioni da eseguire se non sono state lanciate eccezioni
finally:
    operazioni da eseguire comunque
```

Un'istruzione contenuta nel modulo *traceback* che si utilizza nel *except* è

print traceback.format_ exc

. Essa permette di stampare a video le ultime chiamate che il programma ha eseguito prima di terminare nell'eccezione. Questo rende l'individuazione dell'errore da parte del programmatore più facile da recuperare.

Appendice B

Codice Python

B.1 Codice per la creazione della topologia usata nel caso di studio

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class MyTopology(Topo):
        def __init__(self):
                #Initialize topology
                Topo.__init__(self)

                #Adding switch
                switch = self.addSwitch('s1')

                #Adding host
                h1 = self.addHost('h1')

                h2 = self.addHost('h2')
```

```
                    h3 = self.addHost('DPI')

                    #Adding link
                    self.addLink(h1, switch)
                    self.addLink(h2, switch)
                    self.addLink(h3, switch)
topos = {'mytopology': ( lambda: MyTopology() )}
```

B.2 Codice che utilizza il tool ndpiReader

```
from timeit import default_timer
import os
import json
from pprint import pprint
import traceback
import string


def function2(ip):
  i = 0
  flag = 1
  while i < len(ip):
    if ip[i] == ':':
      flag = 0
    i = i + 1
  if flag == 1:
    return 1;
  else:
    return 0;

duration = 0
start = default_timer()
os.chdir("/home/mininet/prove/nDPI/example")
while duration < 60:
  print "-----------------------------------------"
```

```
os.system("sudo ./ndpiReader -i DPI-eth0 -s 6 -v2 -j /
    home/mininet/prove/file.json")
try:
    json_data=open('/home/mininet/prove/file.json')
    data = json.load(json_data)
    i = 0
    try:
        out_file = open("/home/mininet/prove/pacchettiDPI.
            txt","w")
        os.chmod("/home/mininet/prove/pacchettiDPI.txt",
            0777)
    except:
        out_file.close()
        print traceback.format_exc()

    while i < len(data["known.flows"]):
        if function2(data["known.flows"][i]["host_a.name"
            ]) == 1 and function2(data["known.flows"][i]["
            host_b.name"]) == 1 and data["known.flows"][i][
            "detected.protocol.name"] != "Unknown":

            print "host A = ",data["known.flows"][i]["host_a
                .name"],"host B = ",data["known.flows"][i]["
                host_b.name"],"protocol = ",data["known.flows
                "][i]["protocol"]
            print data["known.flows"][i]["detected.protocol.
                name"]
            print data["known.flows"][i]["host_a.port"]
            print data["known.flows"][i]["host_n.port"]
            try:
                s = data["known.flows"][i]["protocol"]
                out_file.write(s)
                s = ":"
                out_file.write(s)

                s = data["known.flows"][i]["detected.protocol.
                    name"]
```

```
            out_file.write(s)
            s = ":"
            out_file.write(s)

            s = data["known.flows"][i]["host_a.name"]
            out_file.write(s)
            s = ":"
            out_file.write(s)

            s = data["known.flows"][i]["host_a.port"]
            out_file.write(str(s))
            s = ":"
            out_file.write(s)

            s = data["known.flows"][i]["host_b.name"]
            out_file.write(s)
            s = ":"
            out_file.write(s)

            s = data["known.flows"][i]["host_n.port"]
            out_file.write(str(s))
            s = "\n"
            out_file.write(s)

        except:
            out_file.close()
            print traceback.format_exc()

      i = i + 1
    out_file.close()

    json_data.close()
  except:
    print "Nessuna comunicazione in questo arco di tempo
        ."
    try:
```

```
        out_file = open("/home/mininet/prove/pacchettiDPI.
            txt","w")
        os.chmod("/home/mininet/prove/pacchettiDPI.txt",
            0777)
        s = "None"
        out_file.write(s)
        out_file.close()
      except:
        out_file.close()
        print traceback.format_exc()

  duration = default_timer() - start
```

B.3 Codice del controller utilizzato nel caso di studio

```
#!/usr/bin/python

#Packages
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
import pox.lib.packet as pkt
import threading
import time
import os.path
import traceback
import os

import string

#Allow logging messages to the console
log = core.getLogger()

global msg
```

```
IDLE_TIMEOUT = 400 # in seconds
global dst_ip_array
dst_ip_array = [0] * 4
global src_ip_array
src_ip_array = [0] * 4
global dl_type_array
dl_type_array = [0] * 4
global switch

class MyThread (threading.Thread):
  def __init__(self):
    threading.Thread.__init__(self)
    self._stop = threading.Event()

  def stop(self):
    self._stop.set()

  def inject_action(self, h1, h2, action, port_a, port_b
    ):
    port_1 = int(port_a)
    port_2 = int(port_b)

    if port_1 == 22:
      dst = h1
    elif port_2 == 22:
      dst = h2
    dst1 = string.split(dst, '.')

    if int(dst1[3]) == 2:
      #Host h1
      if action == "DROP":
        msg = of.ofp_flow_mod( action=of.
          ofp_action_output(port=5), priority=20, match
          =of.ofp_match( dl_type=pkt.ethernet.IP_TYPE,
          nw_dst=dst_ip_array[int(dst1[3])]) )
        for currentConnection in core.openflow.
          connections.keys(): #for each switch
```

```
            connected to the Controller
            core.openflow.connections[currentConnection].
                send(msg) #Send message to the Switch

    elif int(dst1[3]) == 3:
      #Host h2
      if action == "DROP":
        msg = of.ofp_flow_mod( action=of.
            ofp_action_output(port=5), priority=20, match
            =of.ofp_match( dl_type=pkt.ethernet.IP_TYPE,
            nw_dst=dst_ip_array[int(dst1[3])]) )
        for currentConnection in core.openflow.
            connections.keys(): #for each switch
            connected to the Controller
            core.openflow.connections[currentConnection].
                send(msg) #Send message to the Switch

    print "\n \n--> Azione iniettata <-- \n \n"
    return


  def run(self):
    while 1:
      time.sleep(6)
      """thread worker function"""
      try:
        in_file = open("/home/mininet/prove/pacchettiDPI
            .txt","r")
      except:
        in_file = open("/home/mininet/prove/pacchettiDPI
            .txt","w")
        os.chmod("/home/mininet/prove/pacchettiDPI.txt",
            0777)
        s = "None"
        in_file.write(s)
        in_file.close()
```

```
if os.path.isfile("/home/mininet/prove/
   pacchettiDPI.txt"):
  try:
    in_file = open("/home/mininet/prove/
       pacchettiDPI.txt","r")
  except:
    print traceback.format_exc()
  lista = in_file.readlines()
  if lista[0] != "None":
    for i in range(0,len(lista)):
      s = string.split(lista[i], ':') #s[0]:
         protocol -- s[1]:protocol_name -- s[2]:
         host_a -- s[3]:host_a port -- s[4]:host_b
          -- s[5]:host_b port

      if s[1] == "SSH":
        if os.path.isfile("/home/mininet/prove/
           action.txt") == False:
          action_file = open("/home/mininet/prove/
             action.txt","w")
          os.chmod("/home/mininet/prove/action.txt
             ", 0777)
          action_file.close()
        if os.stat("/home/mininet/prove/action.txt
           ")[6] == 0: #file vuoto
          try:
            action_file = open("/home/mininet/
               prove/action.txt","w")
            os.chmod("/home/mininet/prove/action.
               txt", 0777)
            action_file.writelines("DROP "+s[1]+"
               "+s[2]+" "+s[3]+" "+s[4]+" "+s[5])
            action_file.close()
            self.inject_action(s[2], s[4], "DROP",
                s[3], s[5])
          except:
            action_file.close()
```

```
            print traceback.format_exc()
        else: #file non vuoto
            flagg = 0
            k = 0
            try:
                action_file = open("/home/mininet/
                    prove/action.txt","r")
                linea = action_file.readlines()

                while k < len(linea) and flagg != 1:
                    ss = string.split(linea[k], " ")
                    if ss[1] == "SSH":
                        if ss[2] == s[2] and ss[4] == s
                            [4]: #se gli host combaciano
                            if ss[3] == s[3] or ss[5] == s
                                [5]:
                                flagg = 1
                    k = k + 1

                if flagg != 1: #la coppia non e'
                    presente
                    action_file.close()
                    try:
                        action_file = open("/home/mininet/
                            prove/action.txt",'a')
                        action_file.writelines("DROP "+s
                            [1]+" "+s[2]+" "+s[3]+" "+s[4]+
                            " "+s[5])
                        action_file.close()
                        self.inject_action(s[2], s[4], "
                            DROP", s[3], s[5]) #inietta
                            action
                    except:
                        action_file.close()
                        print traceback.format_exc()
            except:
                action_file.close()
```

```
                print traceback.format_exc()



class MyController(object):
  def __init__(self, connection):
    self.connection = connection
    #Listen for PacketIn event
    connection.addListeners(self)
    t = MyThread()
    t.start()

  #Handler for PacketIn event
  def _handle_PacketIn(self, event):
    #Convert an HEX to DEC
          def convertToInt(hexval):
                  dpid_split = string.split(hexval, '-')
                  concat_hex = ''
                  for elem in dpid_split:
                          concat_hex += elem
                  intval = int(concat_hex, 16)
                  return intval

    packet = event.parsed
    src_dpid = dpid_to_str(event.dpid) #Read the dpid of
        the switch that fired the PacketIn event

    int_dpid = convertToInt(src_dpid)
    log.debug("---------------------------------------")
    log.debug("Receiving packet from switch s%d (DPID=%s
       )", int_dpid, src_dpid) #dpid = datapath id

    #Creating the match object
    match = of.ofp_match.from_packet(packet)
    if match.dl_type == pkt.ethernet.ARP_TYPE or match.
       dl_type == pkt.ethernet.IP_TYPE: # if it is an
       ARP or an IP packet...
```

```
        dst_ip = match.nw_dst #dst_ip is an object of type
            IPAddr
        dst_ip_str = dst_ip.toStr() #Convert the IP
           destination address to a string
        dst_ip_split = string.split(dst_ip_str, '.') #
           Isolate all decimal

        src_ip = match.nw_src #src_ip is an object of type
            IPAddr
        src_ip_str = src_ip.toStr() #Convert the source IP
            address to a string
        src_ip_split = string.split(src_ip_str, '.') #
           Isolate all decimal

        dst_ip_array[int(dst_ip_split[3])] = dst_ip
        src_ip_array[int(src_ip_split[3])] = src_ip
        switch = core.openflow.connections.keys()

        log.debug("nw_proto %d",match.nw_proto)
        log.debug("data_link %x",match.dl_type)

        if match.nw_proto == 6 or match.nw_proto == 17 or
           match.tp_src == 22 or match.tp_dst == 22:
          log.debug("port src %d",match.tp_src)
          log.debug("port dst %d",match.tp_dst)

          #Setting rule based on network destination
             address...
          if dst_ip_split[3] == '2':
            #Host h1 destinazione
            log.debug("IP DESTINATION ADDRESS: %s", dst_ip
               )
            log.debug("SOURCE IP ADDRESS: %s", src_ip)

            dl_type_array[int(dst_ip_split[3])] = match.
               dl_type
```

```
        msg = of.ofp_flow_mod( action=of.
            ofp_action_output(port=3), priority=10,
            match=of.ofp_match( dl_type=match.dl_type,
            nw_dst=dst_ip) )
        msg.idle_timeout = IDLE_TIMEOUT

        #connection.send(msg) #Send message to the
            Switch
        for currentConnection in core.openflow.
            connections.keys(): #for each switch
            connected to the Controller
          core.openflow.connections[currentConnection
              ].send(msg) #Send message to the Switch

        msg = of.ofp_flow_mod( action=of.
            ofp_action_output(port=3), priority=10,
            match=of.ofp_match( dl_type=pkt.ethernet.
            IP_TYPE, nw_dst=dst_ip) )
        msg.idle_timeout = IDLE_TIMEOUT

        for currentConnection in core.openflow.
            connections.keys(): #for each switch
            connected to the Controller
          core.openflow.connections[currentConnection
              ].send(msg) #Send message to the Switch


      elif src_ip_split[3] == '2':
        #Host h1 sorgente
        log.debug("IP DESTINATION ADDRESS: %s", dst_ip
            )
        log.debug("SOURCE IP ADDRESS: %s", src_ip)

        msg = of.ofp_flow_mod( action=of.
            ofp_action_output(port=3), priority=10,
            match=of.ofp_match( dl_type=match.dl_type,
            nw_src=src_ip) )
```

```
            msg.idle_timeout = IDLE_TIMEOUT
            for currentConnection in core.openflow.
               connections.keys(): #for each switch
               connected to the Controller
              core.openflow.connections[currentConnection
                 ].send(msg) #Send message to the Switch

            msg = of.ofp_flow_mod( action=of.
               ofp_action_output(port=3), priority=10,
               match=of.ofp_match( dl_type=pkt.ethernet.
               IP_TYPE, nw_src=src_ip) )
            msg.idle_timeout = IDLE_TIMEOUT

            for currentConnection in core.openflow.
               connections.keys(): #for each switch
               connected to the Controller
              core.openflow.connections[currentConnection
                 ].send(msg) #Send message to the Switch

        else:
#-------------------------------------
          if dst_ip_split[3] == '2' and match.nw_proto ==
             1:
            #Host h1
            log.debug("IP DESTINATION ADDRESS: %s", dst_ip
               )
            log.debug("SOURCE IP ADDRESS: %s", src_ip)

            msg = of.ofp_flow_mod( action=of.
               ofp_action_output(port=1), match=of.
               ofp_match( dl_type=match.dl_type, nw_dst=
               dst_ip, nw_proto=1) )
            #Send message to the Switch
            for currentConnection in core.openflow.
               connections.keys(): #for each switch
               connected to the Controller
```

```
            core.openflow.connections[currentConnection
                ].send(msg) #Send message to the Switch

        msg = of.ofp_flow_mod( action=of.
            ofp_action_output(port=1), match=of.
            ofp_match( dl_type=pkt.ethernet.IP_TYPE,
            nw_dst=dst_ip, nw_proto=1) )
        #Send message to the Switch
        for currentConnection in core.openflow.
            connections.keys(): #for each switch
            connected to the Controller
            core.openflow.connections[currentConnection
                ].send(msg) #Send message to the Switch
    elif dst_ip_split[3] == '2' and match.nw_proto
        == 2:

        msg = of.ofp_flow_mod( action=of.
            ofp_action_output(port=1), match=of.
            ofp_match( dl_type=match.dl_type, nw_dst=
            dst_ip, nw_proto=2) )
        #Send message to the Switch
        for currentConnection in core.openflow.
            connections.keys(): #for each switch
            connected to the Controller
            core.openflow.connections[currentConnection
                ].send(msg) #Send message to the Switch

        msg = of.ofp_flow_mod( action=of.
            ofp_action_output(port=1), match=of.
            ofp_match( dl_type=pkt.ethernet.IP_TYPE,
            nw_dst=dst_ip, nw_proto=2) )
        for currentConnection in core.openflow.
            connections.keys(): #for each switch
            connected to the Controller
            core.openflow.connections[currentConnection
                ].send(msg) #Send message to the Switch
```

```
        elif dst_ip_split[3] == '2':

          msg = of.ofp_flow_mod( action=of.
              ofp_action_output(port=3), match=of.
              ofp_match( dl_type=match.dl_type, nw_dst=
              dst_ip) )
          #Send message to the Switch
          for currentConnection in core.openflow.
              connections.keys(): #for each switch
              connected to the Controller
            core.openflow.connections[currentConnection
                ].send(msg) #Send message to the Switch

          msg = of.ofp_flow_mod( action=of.
              ofp_action_output(port=3), match=of.
              ofp_match( dl_type=pkt.ethernet.IP_TYPE,
              nw_dst=dst_ip) )


          for currentConnection in core.openflow.
              connections.keys(): #for each switch
              connected to the Controller
            core.openflow.connections[currentConnection
                ].send(msg) #Send message to the Switch

#-------------------------------
        elif dst_ip_split[3] == '1':
          #Host DPI
          log.debug("IP DESTINATION ADDRESS: %s", dst_ip
              )
          log.debug("SOURCE IP ADDRESS: %s", src_ip)

          msg = of.ofp_flow_mod( action=of.
              ofp_action_output(port=3), match=of.
              ofp_match( dl_type=match.dl_type, nw_dst=
              dst_ip) )
          #Send message to the Switch
```

```
            for currentConnection in core.openflow.
               connections.keys(): #for each switch
               connected to the Controller
              core.openflow.connections[currentConnection
                 ].send(msg) #Send message to the Switch

            msg = of.ofp_flow_mod( action=of.
               ofp_action_output(port=3), match=of.
               ofp_match( dl_type=pkt.ethernet.IP_TYPE,
               nw_dst=dst_ip) )
            #Send message to the Switch
            for currentConnection in core.openflow.
               connections.keys(): #for each switch
               connected to the Controller
              core.openflow.connections[currentConnection
                 ].send(msg) #Send message to the Switch
#-------------------------------------
          elif dst_ip_split[3] == '3' and match.nw_proto
             == 1:
            #Host h2
            log.debug("IP DESTINATION ADDRESS: %s", dst_ip
               )
            log.debug("SOURCE IP ADDRESS: %s", src_ip)

            msg = of.ofp_flow_mod( action=of.
               ofp_action_output(port=2), match=of.
               ofp_match( dl_type=match.dl_type, nw_dst=
               dst_ip, nw_proto=1) )
            log.debug("Porta destinazione 2")
            #Send message to the Switch
            for currentConnection in core.openflow.
               connections.keys(): #for each switch
               connected to the Controller
              core.openflow.connections[currentConnection
                 ].send(msg) #Send message to the Switch
```

```
            msg = of.ofp_flow_mod( action=of.
                ofp_action_output(port=2), match=of.
                ofp_match( dl_type=pkt.ethernet.IP_TYPE,
                nw_dst=dst_ip, nw_proto=1) )
            #Send message to the Switch
            for currentConnection in core.openflow.
                connections.keys(): #for each switch
                connected to the Controller
              core.openflow.connections[currentConnection
                ].send(msg) #Send message to the Switch
        elif dst_ip_split[3] == '3' and match.nw_proto
          == 2:

            msg = of.ofp_flow_mod( action=of.
                ofp_action_output(port=2), match=of.
                ofp_match( dl_type=match.dl_type, nw_dst=
                dst_ip, nw_proto=2) )
            #Send message to the Switch
            for currentConnection in core.openflow.
                connections.keys(): #for each switch
                connected to the Controller
              core.openflow.connections[currentConnection
                ].send(msg) #Send message to the Switch

            msg = of.ofp_flow_mod( action=of.
                ofp_action_output(port=2), match=of.
                ofp_match( dl_type=pkt.ethernet.IP_TYPE,
                nw_dst=dst_ip, nw_proto=2) )


            for currentConnection in core.openflow.
                connections.keys(): #for each switch
                connected to the Controller
              core.openflow.connections[currentConnection
                ].send(msg) #Send message to the Switch

        elif dst_ip_split[3] == '3':
```

```
            msg = of.ofp_flow_mod( action=of.
                ofp_action_output(port=3), match=of.
                ofp_match( dl_type=match.dl_type, nw_dst=
                dst_ip) )
            #Send message to the Switch
            for currentConnection in core.openflow.
                connections.keys(): #for each switch
                connected to the Controller
              core.openflow.connections[currentConnection
                ].send(msg) #Send message to the Switch

            msg = of.ofp_flow_mod( action=of.
                ofp_action_output(port=3), match=of.
                ofp_match( dl_type=pkt.ethernet.IP_TYPE,
                nw_dst=dst_ip) )
            for currentConnection in core.openflow.
                connections.keys(): #for each switch
                connected to the Controller
              core.openflow.connections[currentConnection
                ].send(msg) #Send message to the Switch

          else:
            log.debug("IP DESTINATION (%s) or SOURCE (%s)
                ADDRESS DOES NOT EXISIST! ", dst_ip, src_ip
                )


# ********** MAIN CLASS **********
class my_Controller(object):
  def __init__(self):
    core.openflow.addListeners(self)

  #Define EVENT HANDLER
  def _handle_ConnectionUp(self, event):
    log.debug("[CONTROLLER] Switch %s has come up.",
        dpid_to_str(event.dpid))
```

```
    MyController(event.connection)

def launch():
  core.registerNew(my_Controller)
  log.info("External controller is running!")
```

Appendice C

Ulteriori problemi riguardanti la sicurezza in SDN

Viene di seguito mostrata una breve presentazione che racchiude tutte le tipologie di sicurezza in SDN che si sono viste. Oltre a quelle approfondite nella tesi ve ne sono altre interessanti che vengono brevemente illustrate.

Sicurezza in Software Define Network

Student: Giulio Crestani

Mail: giulio.crestani@studio.unibo.it
Electronics, Informatics and Telecommunications Engineering - University of Bologna

9 ottobre 2014

Sommario

FlowTags
Introduzione 1

Middlebox: dispositivo di rete che trasforma, filtra e ispeziona il traffico di rete per fini diversi dal forwarding. Esempi di middlebox sono:

- Firewall filtrano traffici non desiderati o pericolosi;
- NAT modificano gli indirizzi sorgente e destinazione;
- IDS (*Intrusion Detection System*) che controllano il traffico e raccolte di dati per eseguire analisi di sicurezza.

FlowTags
Introduzione 2

Gli IDS, a differenza dei firewall, non filtrano i pacchetti in tempo reale, ma sono capaci di eseguire ispezioni più approfondite. I firewall, invece, hanno il compito di accettare o meno ogni pacchetto che arriva dalla rete.

I middlebox sono componenti critici dell'infrastruttura di rete per la garanzia di sicurezza e di performance. Questo perchè le modifiche dinamiche dipendenti dal traffico attraverso di essi, causano una maggiore difficoltà di corretta applicazione delle policy sulla rete.

FlowTags
Introduzione 3

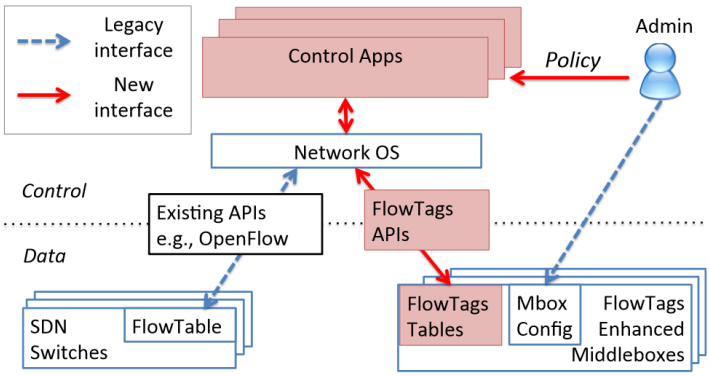
Si ritiene necessario avere una capacità di tracciamento del flusso (*flow tracking*) per assicurare un'applicazione consistente delle policy in presenza di tali modifiche dinamiche del traffico. Per questo si propone **FlowTags**, un'estensione dell'architettura SDN dove i middlebox aggiungono *tag* ai pacchetti in uscita per fornire il contesto necessario.

FlowTags
Architettura 1

L'idea chiave di FlowTags è quella di etichettare i pacchetti con il necessario contesto di middlebox. Così facendo, le azioni di elaborazione dei pacchetti di un middlebox FlowTags comporteranno l'aggiunta del relativo tag all'interno del header del pacchetto.
Il controller SDN configura le azioni sugli switch e sui middlebox per usare questi tag (aggiunti da altri middlebox) come parte delle loro operazioni sul piano di dati al fine di applicare correttamente le politiche a livello di rete.

FlowTags
Architettura 2

SDN oggi fornisce un'interfaccia tra il controller e gli switch per controllare il comportamento di inoltro. FlowTags estende questa architettura lungo tre dimensioni fondamentali evidenziati in figura:



FlowTags
Architettura 3: le tre dimensioni fondamentali

Potenziamento FlowTags dei middlebox i middlebox tengono conto dei tag dei pacchetti in ingresso mentre li processano e possono anche aggiungere nuovi tag basati sul contesto. Gli switch usano i tag per pilotare i pacchetti;

Nuove API FlowTags tra controller e middlebox con FlowTags;

Nuove applicazioni di controllo che consentono di configurare il comportamento di tagging dei middlebox e switch.

Un chiaro svantaggio di questa architettura è il fatto che lavora con solo criteri predefiniti e non è in grado di gestire azioni dinamiche.

DDoS detection

Introduzione

Il meccanismo di attacco DDoS si basa sullo sfruttamento dell'enorme risorsa asimmetrica tra Internet (composto da migliaia di host) e un server vittima. Questo viene limitato a trattare con un numero eccezionalmente elevato di false richieste. Di conseguenza, le richieste degli utenti legittimi non sono nemmeno elaborate perchè le risorse di sistema sono eseguite fino ad esaurimento e la vittima viene respinta da Internet.

- Quali sfide bisognerebbe affrontare per rilevare con successo un attacco DDoS?

La prima difficoltà nasce dai campi di intestazione dei pacchetti che vengono modificati per assomigliare a quelli normali. Come risultato, è un compito molto difficile distinguere tra i pacchetti legittimi di traffico normale e quelli inutili inviati da host compromessi.
Il secondo problema è l'enorme numero di pacchetti da analizzare.

DDoS detection

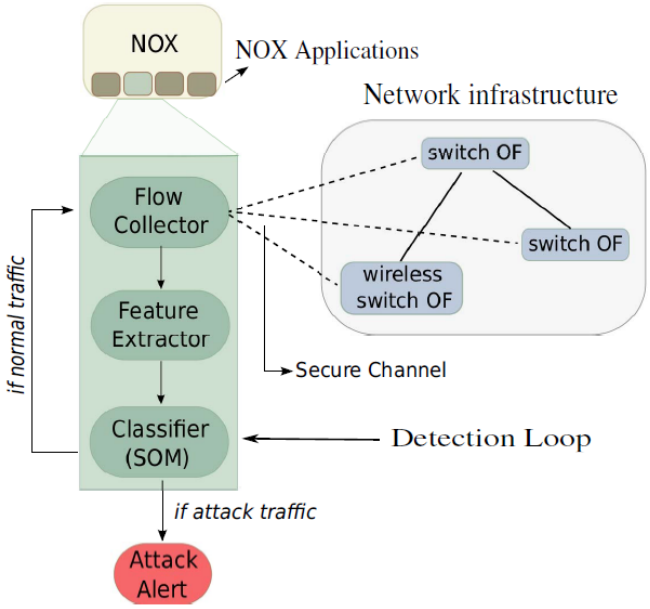
Possibile soluzione

Si propone un metodo leggero per il rilevamento degli attacchi DDoS basato sulle caratteristiche dei flussi di traffico per affrontare le sfide di cui sopra.
Questo metodo viene implementato su una rete dove switch OpenFlow mantengono le flow table con le statistiche di tutti i flussi attivi.
Tutte le informazioni necessarie sono accedute in modo efficiente dal controller e poi elaborate da un meccanismo intelligente di rilevamento degli attacchi. Per quanto è nostra conoscenza, questo metodo è in diretto contrasto con approcci esistenti, che richiedono pesanti elaborazioni per estrarre informazioni specifiche necessarie per l'analisi del traffico.

DDoS detection
SOM

Il meccanismo intelligente impiegato da questo metodo si basa su SOM (*Self Organizing Maps*), una rete neurale artificiale[1] con le caratteristiche del flusso di traffico.
Si usa SOM per classificare il traffico di rete come normale o anormale, cioè un potenziale attacco, tenendo statistiche sui flussi come parametri per il calcolo SOM.

[1] Modelli matematici che rappresentano l'interconnessione tra elementi definiti neuroni artificiali, ossia costrutti matematici. Questi modelli possono essere utilizzati per risolvere problemi ingegneristici di intelligenza artificiale come quelli che si pongono in diversi ambiti tecnologici.



DDoS detection

Descrizione immagine

Flow Collector responsabile della richiesta periodica di entry da tutte le flow table degli switch. Tali richieste, così come le loro relative risposte, sono trasmesse attraverso un canale sicuro, cioè un canale isolato dall'host collegato fino allo switch;

Feature Extractor riceve i flussi raccolti, estrae le caratteristiche che sono importanti per il rilevamento degli attacchi DDoS, e li raccoglie in 6-tuple[2] per essere passate al classificatore;

Classifier analizza se una delle 6-tuple corrisponde a un attacco DDoS o al traffico legittimo. La classificazione può essere effettuata con qualsiasi metodo statistico o di apprendimento. In questo caso si utilizza SOM come metodo di classificazione.

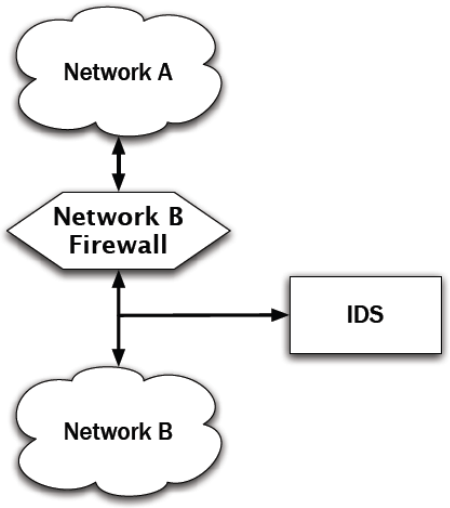
[2] Considera 6 voci della flow table per ogni pacchetto che analizza

OpenSAFE

Introduzione

La sicurezza avviene in due fasi: protezione attiva (firewall, per esempio) e monitoraggio della rete.
In questo lavoro, si cerca di monitorare la rete. Tipicamente, questo controllo è fornito sia tramite middlebox posti direttamente sul percorso di rete o tramite ispezione di copie del traffico in punti interessanti della rete. Di seguito viene presentata una tipica configurazione della rete utilizzata oggigiorno per il monitoring.

OpenSAFE
Introduzione 2



OpenSAFE
Introduzione 3

La gestione di più dispositivi su un intervallo è un compito difficile. Tutto il traffico deve essere trasmesso ad ogni dispositivo nello stesso intervallo, o particolare cura deve essere presa per dirigere sottoinsiemi di traffico a diversi devices.
Questo rende molto più difficile per gli amministratori l'aggiunta di nuovi dispositivi di controllo, che possono essere fastidiosi quando sono acquisiti apparecchi più specializzati o quando il carico diventa troppo grande per l'hardware del corrente dispositivo di monitoraggio.

OpenSAFE

Open Security Auditing and Flow Examination

Per risolvere questo problema si usa OpenSAFE (*Open Security Auditing and Flow Examination*), un sistema unificato per il monitoraggio della rete.

Sfruttando la tecnologia di rete aperta come OpenFlow, OpenSAFE può dirigere il traffico di rete in modi arbitrari, può gestire qualsiasi numero di ingressi di rete e gestire il traffico in modo tale che possa essere utilizzato da molti servizi mentre filtra pacchetti a velocità di linea.

OpenSAFE

Componenti principali

OpenSAFE è costituito da tre componenti principali:

1. un insieme di astrazioni di progettazione per pensare al flusso del traffico di rete;
2. ALARMS (A Language for Arbitrary Route Management for Security), un linguaggio di policy per specificare e gestire facilmente percorsi;
3. un componente OpenFlow che implementa la politica.

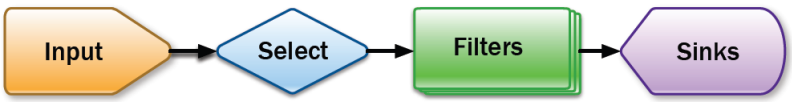
OpenSAFE
Paths

Usiamo la nozione di paths come l'astrazione di base per descrivere la selezione del traffico e del percorso che questo dovrebbe prendere.

Fondamentalmente, vogliamo sostenere la costruzione di percorsi che consentono al traffico desiderato di accedere al sistema ed essere indirizzato a uno o più sistemi di monitoraggio della rete.

OpenSAFE
Paths 2

In OpenSAFE, l'articolazione dei paths avviene gradualmente lungo il sentiero desiderato del percorso. Come mostrato in figura, i percorsi sono composti di più parti: ingressi, selezionatori, filtri e sink.

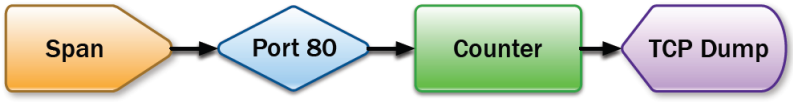


OpenSAFE
Paths 3

Ad alto livello, ogni percorso inizia con un ingresso, applica un criterio di selezione opzionale, seleziona il traffico corrispondente attraverso zero o più filtri, e termina in una o più sink. Gli ingressi possono solo produrre traffico, il sink può ricevere solo il traffico e i filtri devono fare entrambe le cose. Un semplice esempio di traduzione di un percorso di path può essere il seguente:

OpenSAFE
Paths 4

Che viene successivamente tradotto nel modo seguente:



Questa immagine mostra il traffico in ingresso su una porta span (input) che viene selezionato per la porta 80 (selezionatore), instradato attraverso un contatore (filtro) ed infine inviato ad un tcp dump (sink).

OpenSAFE
ALARMS

Per consentire agli amministratori di rete di gestire e aggiornare facilmente la propria infrastruttura di controllo, si introduce ALARMS, un linguaggio per la gestione del percorso arbitrario per la sicurezza del traffico. ALARMS utilizza le astrazioni appena descritte per creare una semplice sintassi del linguaggio di policy per descrivere i percorsi. Questi ultimi vengono definiti tra componenti noti, e ciascun componente può essere soggetto a una regola di distribuzione che, in caso di più componenti, viene parallelizzata.

CloudWatcher
Introduzione 1

Poichè una rete cloud è piuttosto complicata e difficile da riconfigurare, dovremmo indagare attentamente sui luoghi appropriati per l'installazione di dispositivi di sicurezza. In caso contrario, potremmo avere bisogno di ridefinire o spostare dispositivi di sicurezza di frequente, e non è un lavoro facile. In secondo luogo, dobbiamo implementare dispositivi di sicurezza di rete considerando la dinamicità del cloud computing.

CloudWatcher

Introduzione 2 - Esempio

Consideriamo il caso dell'installazione di un NIDS[3] su un collegamento tra l'host A e l'host B, lasciando che il sistema di rilevamento del traffico di rete prodotto da una macchina virtuale sia in esecuzione sull'host A.
Se le macchine virtuali sull'host A si spostassero/migrassero su un altro host C, avremo bisogno di trasferire il sistema di rilevamento come collegamento tra host A e l'host C. Questo tipo di migrazione delle macchine virtuali è abbastanza frequente nel cloud computing.

[3] *Network Intrusion Detection System*: strumenti che analizzano il traffico di una LAN al fine di individuare anomalie nei flussi.

CloudWatcher

Funzionamento 1

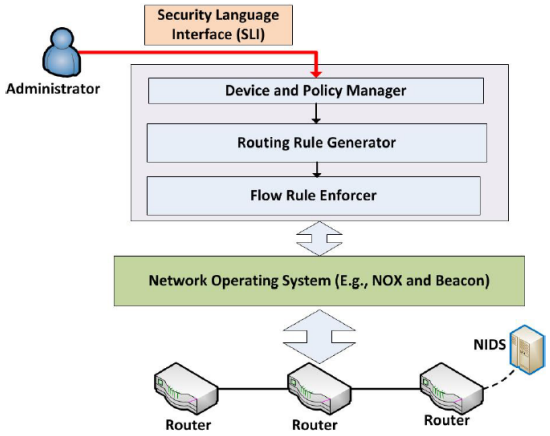
Per affrontare questi problemi, si propone un nuovo framework chiamato CloudWatcher che offre i seguenti vantaggi:

- controlla i flussi di rete per garantire che tutti i pacchetti necessari siano controllati da alcuni dispositivi di sicurezza;
- offre un semplice linguaggio di script per policy per aiutare le persone a utilizzare i servizi forniti facilmente.

CloudWatcher
Funzionamento 2

Rispetto alla configurazione di veri dispositivi fisici, il controllo dei percorsi dei flussi di rete per passare attraverso alcuni nodi è molto più facile da realizzare. Inoltre, alcune tecnologie recenti come *software-defined networking (SDN)* forniscono un modo di controllare flussi di rete come meglio si desidera. Con l'aiuto di queste tecnologie, CLOUDWATCHER cambia i percorsi di routing per i flussi di rete, e fa si che i flussi trasmettano attraverso i nodi della rete in cui i dispositivi di sicurezza risiedono.

CloudWatcher
Funzionamento 3



Resonance
Introduzione

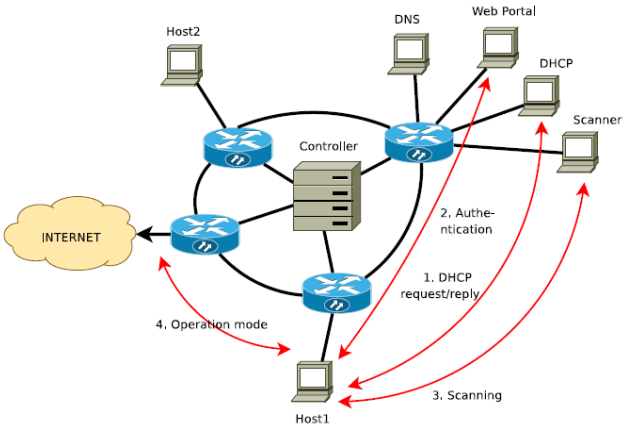
Oggigiorno l'autenticazione e la sicurezza di un host all'interno di una rete aziendale sono impegnativi e gli operatori di rete si basano su un mix di tecniche reattive, ad hoc; implementano politiche usando middlebox, IDS e un insieme di complesse configurazioni di rete.
Invece di fare affidamento su middlebox o sugli host, una rete aziendale deve offrire meccanismi che controllino direttamente il traffico di rete in base alle politiche di sicurezza ed in base alla risposta dei monitor di rete distribuiti.

Resonance
Proprietà 1

1. Fornisce meccanismi per attuare direttamente e dinamicamente le politiche di sicurezza nella rete, nei dispositivi e negli switch, lasciando poche responsabilità sia agli host o strati superiori della rete;
2. Gestisce il livello di rete con funzioni di base necessarie per implementare politiche di sicurezza così come un'interfaccia di controllo che consente ai sistemi di monitoraggio di controllare il traffico in base a criteri predefiniti;
3. Controlla il traffico utilizzando politiche che vengono installate dal controller negli switch programmabili. Estendendo questo paradigma, creiamo un quadro di controllo di accesso dinamico che integra il controller con sottosistemi di controllo. Questa integrazione consente ad un operatore di specificare come la rete dovrebbe controllare il traffico dell'azienda prendendo spunto da un cambiamento delle condizioni di rete.

Resonance
Proprietà 2

Ad esempio, Resonance può mettere automaticamente in quarantena gli host o sottoinsiemi di traffico quando un'anomalia o una falla nella sicurezza viene rilevata.



Resonance
Proprietà 3

Il controller attua le politiche di controllo degli accessi installando le entry appropriate nella flow table all'interno degli stessi switch; utilizza l'indirizzo MAC corrispondente all'interfaccia di un host per mappare il traffico che fluisce.

Ident++

Introduzione 1

La sicurezza di rete sta gravitando verso un controllo più centralizzato. Questa forte centralizzazione pone un pesante fardello per l'amministratore che deve gestire le politiche di sicurezza complesse ed essere in grado di adattarsi alle richieste degli utenti. Per far fronte a ciò, l'amministratore deve delegare il controllo all'host finale e agli utenti, una capacità che manca nelle reti odierne.
La delegazione rende gli amministratori meno un collo di bottiglia quando le policy hanno bisogno di essere cambiate e permette all'amministrazione della rete di seguire le linee organizzative.

Ident++

Introduzione 2

Per abilitare la delega, si propone ident++, un semplice protocollo per richiedere informazioni aggiuntive da parte degli host e delle reti circa il percorso di un flusso. ident++ consente agli end-host di partecipare a procedimenti di sicurezza della rete, fornendo le informazioni che l'amministratore potrebbe non avere o regole da far rispettare a loro nome.

Ident++
Funzionamento 1

È un protocollo progettato per aiutare gli amministratori di rete delegando gli aspetti della politica di sicurezza di rete ad altri decisori: end-host, utenti, applicazioni, sviluppatori di applicazioni, o di terze parti (di fiducia). ident++ è semplice; si ispira al *Identification Protocol*[4], ma è più ricco e più flessibile.

[4] RFC 1413: *The Identification Protocol provides a means to determine the identity of a user of a particular TCP connection. Given a TCP port number pair, it returns a character string which identifies the owner of that connection on the server's system.*

Ident++
Funzionamento 2

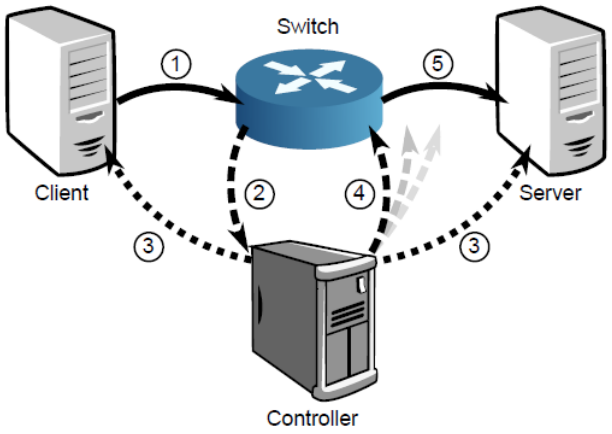
Un demone ident++ in esecuzione su mittenti e destinatari risponde alle domande di coloro che prendono le decisioni circa i flussi. La risposta è una lista di coppie chiave-valore che possono essere utilizzate per fornire informazioni su cui, colui che decide, può basare la sua decisione.
I decisori ident++ abilitati sul percorso possono anche rispondere a domande per conto degli end-host o possono modificare le risposte per aggiungere ulteriori informazioni.

Ident++
Delegazione

La delegazione in ident++ è doppia:

- Incoraggia end-host e gli utenti a classificare il traffico e
- permette loro di specificare le regole che devono essere eseguite sulla rete tramite l'utilizzo di coppie chiave-valore.

Ident++
Schema di interazione 1



Ident++

Schema di interazione 2

1. Il client inizia il flusso inviando un pacchetto;
2. Lo switch inoltra il pacchetto al controller;
3. Il controller richiede ulteriori informazioni usando ident++ sia al mittente (client) che al ricevente (server) del pacchetto;
4. Se il controller approva, installa le entries lungo il percorso per il flusso;
5. Il pacchetto può procedere verso la destinazione.

LIDS

Introduzione 1

I dispositivi mobili sono sempre più utilizzati in applicazioni delicate che vanno dal biomonitoraggio e controllo negli ospedali ai materiali robotici di movimento e trasporto all'interno delle fabbriche.
Questi dispositivi hanno problemi di sicurezza: molti sono costruiti con metodi standard, a basso consumo, dispositivi con risorse limitate i quali mancano le funzionalità di un computer vero e proprio.
Inoltre, l'interruzione del servizio, anche parziale, da parte di un'entità esterna può causare gravi danni, che vanno dalla perdita finanziaria per i danni materiali alla perdita effettiva della vita.

LIDS

Introduzione 2

Viene presentato un sistema basato sulla rete di rilevamento delle intrusioni (IDS) per i dispositivi mobili che comunica con una rete di infrastrutture in loco. Può essere utilizzato insieme a qualsiasi tipo di sicurezza esistente sul dispositivo, senza modifiche ai dispositivi stessi.

Tradizionalmente, le implementazioni IDS hanno sofferto di un'incapacità di far fronte alla mobilità degli end-host, così come un insieme limitato di azioni (ad esempio, la registrazione dell'anomalia o avviso di un amministratore di sistema), che possono essere prese in risposta al rilevamento di anomalie. Entrambe queste difficoltà hanno limitato l'adozione di IDS al di fuori degli ambienti tradizionali.

LIDS

Funzionamento 1

Learning IDS (L-IDS) utilizza il protocollo OpenFlow e il paradigma SDN per affrontare entrambi questi problemi: la mobilità degli end-host può essere gestita in modo trasparente dalla rete e sono disponibili una varietà di risposte rapide e riconfigurazioni di rete per gestire l'attacco in tempo reale (ad esempio: caduta di flusso, riautenticazione, reindirizzamento honeypot[5], e l'isolamento del sistema), piuttosto che limitarsi a reagire dopo il fatto.

[5] *Un tipo di trappola utilizzata dai professionisti della sicurezza per rilevare gli attacchi degli hacker o prelevare campioni di malware.*

LIDS

Funzionamento 2

La maggior parte della logica IDS risiede nel controller OpenFlow, ma parte del processo di rilevamento delle anomalie e tutto il processo di misurazione del traffico risiede nella rete stessa, sugli switch OpenFlow.
Il controller in L-IDS ha tre ruoli:

1. Rilevamento di alcune anomalie sulla base dell'evoluzione del traffico nel corso del tempo;
2. L'adeguamento dello stato delle regole di routing e delle anomalie per la mobilità dell'end host;
3. La riconfigurazione della rete in risposta alle intrusioni.

NetFPGA

Introduzione 1

Ogni anno worm e virus rappresentano miliardi di dollari in danni economici.

Diversi attacchi di rete infettano e diffondono attraverso il payload dei pacchetti e attraverso traffici di rete progettati su misura. Un modo efficace per individuare e prevenire attacchi di rete è attraverso il DPI (*Deep Packet Inspection*).

Questo prende in esame non solo le intestazioni, ma anche i payload dei pacchetti. Di conseguenza, un sistema di sicurezza che incorpora un deep packet filter offre una migliore protezione dagli attacchi rispetto ai firewall tradizionali.

NetFPGA
Introduzione 2

Tuttavia, la scansione del payload di ogni pacchetto di ogni byte richiede elevati requisiti di calcolo soprattutto se ci sono più pattern che devono essere abbinati.

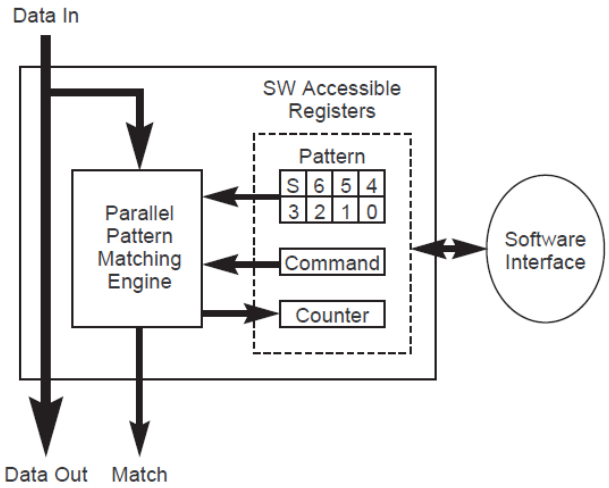
Utilizzando approcci software basati, sul rilevamento di una serie di pattern di stringhe in un pacchetto di rete su 1Gbps è un compito difficile anche sugli ultimi multiprocessori general purpose. Negli ultimi anni, un certo numero di ricercatori ha studiato nuovi modi per implementare e accelerare il pattern matching su Field Programmable Gate Array (FPGA).

NetFPGA
Descrizione

La scheda di interfaccia di rete NetFPGA è una piattaforma che può essere usata per sviluppare e testare algoritmi su FPGA basati su deep packet inspection. Oltre a questo hardware, si ha bisogno di un ambiente di test all'interno del quale si testano esperimenti realistici che provano, appunto, il progetto hardware. Questo ambiente si chiama DETER (The Cyber DEfense Technology Experimental Research) ed è progettato per eseguire esperimenti e simulazioni nell'ambito della sicurezza di rete.

NetFPGA

Descrizione



NICE

Introduzione

Il modello di programmazione centralizzato, in cui un singolo controller gestisce la rete, sembra ridurre la probabilità di errori. Tuttavia, il sistema è intrinsecamente distribuito e asincrono, con eventi che accadono a diversi switch e host dove, come conseguenza, nascono inevitabili ritardi che incidono sulla comunicazione con il controller.

NICE

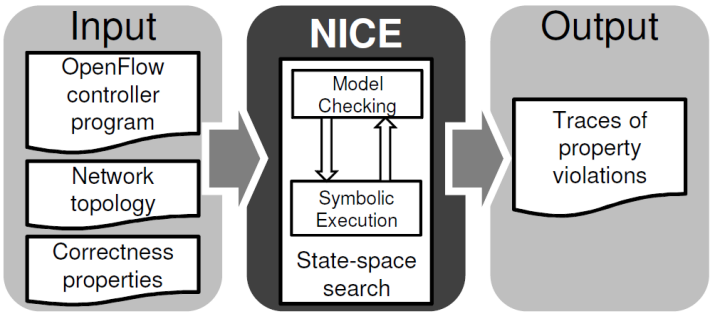
Funzionamento 1

NICE (*No bugs In Controller Execution*) è uno strumento che applica model checking (*tecnica realizzata mediante la verifica del modello, spesso derivato dal modello hardware o software, che soddisfa una specifica formale*) per esplorare lo spazio degli stati di tutto il sistema controller, gli switch e gli host.

La scalabilità è la sfida principale, data la diversità dei pacchetti di dati, lo stato del sistema di grandi dimensioni e i numerosi eventi possibili. Per risolvere questo problema, viene proposto un nuovo modo per aumentare il controllo del modello: attraverso l'esecuzione simbolica di event handler.

NICE

Immagine riassuntiva



NICE
Funzionamento 2

Verifica i programmi non modificati eseguiti sul controller generando automaticamente i flussi dei pacchetti, curati sotto molti punti di vista degli eventi possibili.
Per utilizzare NICE, il programmatore fornisce il programma del controller, e la specifica di una topologia di switch e host da utilizzare. Può inoltre *istruire* NICE per controllare le proprietà generiche di correttezza come per esempio nessun ciclo di inoltro o buchi neri, ed eventualmente scrivere parti supplementari, proprietà di correttezza specifiche dell'applicazione.

FlowChecker
Introduzione 1

OpenFlow è un'architettura innovativa che fornisce una piattaforma programmabile aperta per il controllo dell'accesso alla rete.

Separando i piani dati e di controllo, gli utenti possono utilizzare i controller centralizzati OpenFlow per installare filtri negli switch OpenFlow e controllare i dati globali processati sulla rete.

Il controller esegue nuovi protocolli o algoritmi che possono inserire, modificare o rimuovere filtri negli switch al fine di applicare le politiche o le proprietà a livello di rete.

FlowChecker
Funzionamento

Il sistema presentato, denominato FlowChecker, può essere utilizzato dagli amministratori/utenti per:

1. verificare la coerenza dei diversi switch e controller tra le varie infrastrutture OpenFlow;
2. verificare la correttezza della sintesi della configurazione generata da nuovi protocolli implementati;
3. eseguire il debug dei problemi di raggiungibilità e di sicurezza.

FlowChecker può anche essere usato per studiare l'impatto dei nuovi protocolli o algoritmi sulla rete semplicemente cambiando lo stato delle flow table e poi analizzando l'effetto.

FlowChecker
FlowVisor 1

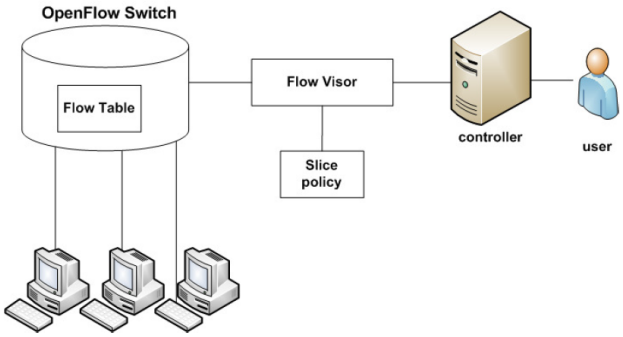
Si introduce FlowVisor come strato di virtualizzazione di rete. Esso divide la rete in diverse sezioni, con ogni fetta associata al proprio insieme di flussi d'esecuzione su switch. FlowVisor:

- supporta più controller sulla stessa rete OpenFlow;
- si assicura che un controller ospite possa solo osservare e controllare gli switch che si suppone facciano parte del suo campo;
- suddivide la banda di collegamento assegnando un rate minimo per l'insieme di flussi che costituiscono una fetta;
- suddivide la flow table di ogni switch in modo tale da tenere traccia di quali entry appartengono a ciascun controller ospite.

FlowChecker

FlowVisor 2

FlowVisor funziona come un proxy OpenFlow tra switch e controller, come mostrato in figura:



Tutti i messaggi tra gli switch e il controller sono mandati attraverso il FlowVisor. Controller multipli possono essere ospitati da FlowVisor con ogni controller che richiede la sua fetta di rete.

FLOVER

Introduzione 1

OpenFlow fornisce meccanismi integrati per amministratori di applicazioni al fine di controllare, in modo statico o in fase di esecuzione, che le policy di sicurezza della rete siano conformi ai modelli.

Si sostiene che il supporto all'inserimento automatico di policy per la sicurezza, vale a dire, la possibilità di convalidare il fatto che gli aggiornamenti delle regole di flusso prodotti dinamicamente conservino tutte le proprietà di sicurezza specificate nel criterio di sicurezza di rete, sia di vitale importanza per l'ampia adozione di OpenFlow.

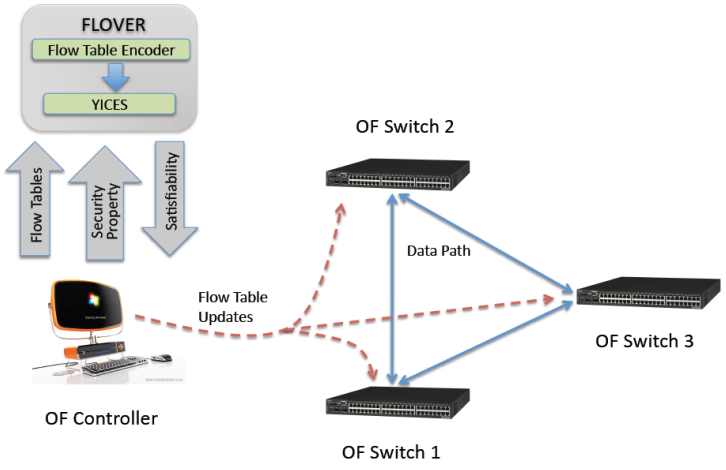
In letteratura si propone un metodo automatico per verificare che una determinata proprietà non bypassi un insieme di regole di flusso generate da un controller OpenFlow.
Non-bypassability è una caratteristica di sicurezza di base applicata dalla maggior parte dei firewall e switch.

FLOVER
Funzionamento 1

FLOVER è implementato come un'applicazione OpenFlow che gira su un controller OpenFlow. Ogni volta che un controller fornisce regole di flusso appena create a uno switch, FLOVER verifica l'insieme delle regole di flusso appena aggiornate con un insieme di specifiche proprietà non bypassabili.

In particolare, se FLOVER riceve una richiesta di una regola di flusso da uno switch, per prima cosa crea un comando aggiornato o aggiunge un nuovo comando composto da almeno una regola che sia come quella che gli ha mandato la richiesta, e verifica che le proprietà non bypassabili predefinite siano presenti.

FLOVER
Funzionamento 2



FLOVER

Yices

Yices è un risolutore dotato di una modellazione di file di input in logica del primo ordine. Se un determinato modello è soddisfacente, cioè, esiste almeno un'istanza che soddisfa tutti i vincoli del modello, Yices crea un esempio appagante. In caso contrario, Yices riporta il modello per essere inappagabile. Sfruttiamo la solidità di Yices e la sua capacità di trovare in modo efficiente esempi soddisfacenti, per verificare set di regole di flusso. [Model Checking Invariant Security Properties in OpenFlow - Son, Shin, Yegneswaran, Porras, Gu]

FSL

Introduzione

Una vasta gamma di imprese moderne si basano sulle loro reti informatiche per le operazioni di tutti i giorni e si affidano ai loro amministratori di rete per configurare i componenti di sicurezza in modo che la rete non sia violata da individui o malware.

Mentre le politiche di sicurezza di rete tradizionali sono state applicate in base alla configurazione manuale dei singoli componenti di rete come router, firewall, NAT e VLAN, emergenti progetti di rete di imprese supportano le politiche globali dichiarate nel corso di astrazioni di alto livello. Si è promossa l'evoluzione di meccanismi di sicurezza di rete più semplici e potenti per la progettazione, l'implementazione e sperimentazione di un linguaggio di policy sulla sicurezza di rete basato sul flusso.

FSL

Funzionamento 1

Si propone Flow-based Security Language (FSL) che è progettato per fornire concetti significativi di alto livello agli amministratori di rete, mentre allo stesso tempo supporta implementazioni efficienti delle policy risultanti delle reti moderne.

FSL

Funzionamento 2

Il progetto specifico di FSL si basa su tre problemi pratici:

- **Vincoli generali**: molti criteri del controllo dell'accesso considerano solo le azioni binarie di permesso o negazione; tuttavia, la moderna sicurezza delle reti richiede un approccio più generale. Gli operatori vorrebbero imporre vincoli di connettività sulla rate, sul percorso (imponendogli di evitare o includere certi nodi), e sulla direzionalità (come ad esempio limitando i client per flussi solo in uscita) della comunicazione.

FSL
Funzionamento 3

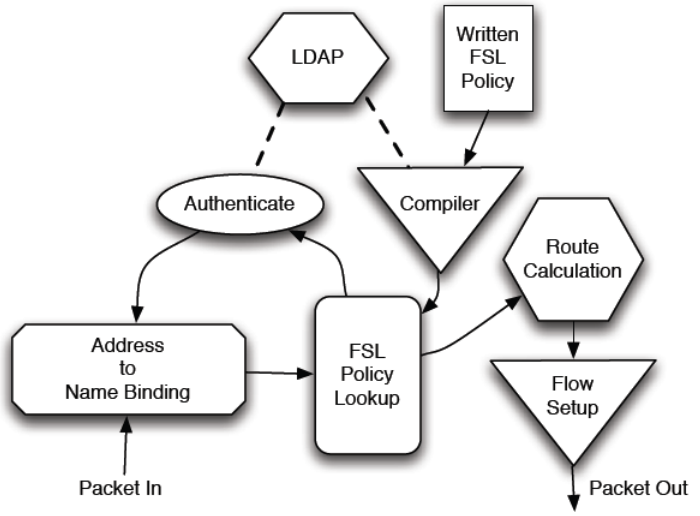
- **Autorità distribuita ed incrementale**: ci sono molti creatori di policy in grandi imprese, e così il linguaggio delle policy deve far fronte ad eventuali conflitti tra le loro policy. In più gli operatori possono aver bisogno di immettere rapidamente nuove policy in un sistema senza garantire che le nuove entry siano consistenti con tutte le politiche già presenti. Perciò, FSL deve gestire due tipi di conflitti tra policy: quelli tra gli amministratori e quelli tra le regole di un amministratore.

FSL
Funzionamento 4

- **Speed**: le decisioni sulle policy devono essere eseguite su ogni pacchetto, e sui link di rete che possono facilmente raggiungere milioni di decisioni da imporre al secondo.

FSL

Funzionamento 5



Bibliografia

[1] http://searchsdn.techtarget.com/definition/SDN-controller-software-defined-networking-controller.

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

[3] Thomas D. Madeau and Ken Gray, *SDN: Software Defined Networks*, O'Reilly, 1ª edizione (2013).

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

[5] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker and Jonathan Turner, "OpenFlow: Enabling Innovation in Campus Networks", *ACM SIGCOMM Computer Communication Review*, Vol. 38, Issue. 2, pp. 69-74, March 2008.

[6] Nick McKeown, "How SDN will Shape Networking" *Open Networking Summit 2011.*

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

[8] Introduzione a Mininet e lavorare con Mininet, http://mininet.org/walkthrough/

[9] Chiara Contoli, "SDN in practice: OpenFlow protocol and Mininet emulator" *Department of Electrical, Electronic and Information Engineering "Guglielmo Marconi" - DEI*, May 2014

[10] Spanning Tree Protocol: `http://net.infocom.uniroma1.it/corsi/reticalc/lucidi/2009-10/pdf/STP.pdf`

[11] Spanning Tree Protocol: `http://it.wikipedia.org/wiki/Spanning_tree_(networking)`

[12] Sandra Scott-Hayward, Gemma O'Callaghan and Sakir Sezer, "SDN Security: A Survey" *IEEE SDN for Future Networks and Services*, pp. 56-62, 2013.

[13] Seyed Kaveh Fayazbakhsh, Vyas Sekar, Minlan Yu and Jeffrey C. Mogul, "FlowTags: Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions" *Proceedings of the second workshop on Hot topics in software defined networks* ACM, 2013.

[14] API Northbound e Southbound: `http://whatis.techtarget.com/definition/northbound-interface-southbound-interface`

[15] Jeffrey R. Ballard, Ian Rae and Aditya Akella, "Extensible and Scalable Network Monitoring Using OpenSAFE" *Proc.INM/WREN* 2010

[16] Seungwon Shin and Guofei Gu, "CloudWatcher: Network Security Monitoring Using OpenFlow in Dynamic Cloud Networks (or: How to Provide Security Monitoring as a Service in Clouds?)" *20th IEEE International Conference on Network Protocols (ICNP)*, IEEE 2012, pp. 1-6.

[17] Ankur Nayak, Alex Reimers, Nick Feamster and Russ Clark, "Resonance: Dynamic Access Control for Enterprise Networks" *Proceedings of the 1st ACM workshop on Research on enterprise networking*, ACM, 2009, pp. 11-18.

[18] Comandi ovs-ofctl: `http://openvswitch.org/cgi-bin/ovsman.cgi?page=utilities/ovs-ofctl.8`

[19] Tutorial python: `http://www.python.it/doc/Easytut/easytut-it/easytut-it.html`

[20] Documentazione ufficiale python: `http://www.python.it/doc/`

[21] Guida introduttiva nDPI: `http://www.ntop.org/wp-content/uploads/2013/12/nDPI_QuickStartGuide.pdf`

[22] nDPI: `http://www.ntop.org/products/ndpi/`