ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Scienze dell'Informazione

PROGETTAZIONE E IMPLEMENTAZIONE DI UN SISTEMA DINAMICO PER LA GESTIONE DEL PACKET FILTERING

Elaborata nel corso di: **Reti di calcolatori**

Tesi di laurea di:
LOPERFIDO FRANCESCO

Relatore:
GABRIELE D'ANGELO

Correlatore:
CIRO BARBONE

ANNO ACCADEMICO 2009-2010
SESSIONE II

A Mamma e Papà

“L’immaginazione è più importante della conoscenza”

Albert Einstein

Indice

Indice delle immagini

Introduzione

La gestione delle reti e in particolare la misurazione del traffico, è stata considerata un'attività necessaria fin dai primi tempi della nascita delle reti. Gli amministratori hanno sempre dovuto tener traccia dei flussi di dati per diverse ragioni, tra cui l'identificazione di traffico anomalo.

Come in altre realtà, anche in quella del Polo Scientifico Didattico di Cesena c'è la necessità di monitorare il traffico generato dalla rete, per assicurare un corretto utilizzo dei servizi offerti e l'integrità dei dati associati a ogni singolo utente.

Pur avendo un livello di sicurezza che prevede l'autenticazione e autorizzazione di un utente e che permette di tenere traccia di tutte le operazioni effettuate, questo non esclude la rete dall'essere soggetta a incidenti informatici, che possono derivare da tentativi di accesso agli host tramite innalzamento illecito di privilegi o dai classici programmi malevoli come virus, trojan e worm. Il rimedio adottato per identificare eventuali minacce prevede l'utilizzo di un dispositivo IDS (*Intrusion Detection System*) con il compito di analizzare il traffico e confrontarlo con una serie d'impronte che fanno riferimento a scenari d'intrusioni conosciute.

A seguito dell'installazione e della configurazione di questo servizio di *detection*, durante la fase di analisi del funzionamento, è però emerso che, nonostante le capacità di elaborazione dell'hardware fossero notevoli, le risorse non erano sufficienti a garantire un corretto funzionamento del servizio sull'intero traffico che attraversa la rete.

Da qui è nata l'idea di un progetto con lo scopo di risolvere questo problema. Avendo analizzato con attenzione la problematica e avendo escluso l'acquisto di un dispositivo hardware adatto all'analisi delle reti ad alta velocità a causa dell'eccessiva spesa, l'unica soluzione possibile richiedeva un intervento software.

Il criterio adottato prevede la creazione di un'applicazione con lo scopo di eseguire un'analisi preventiva, in modo da alleggerire la mole di dati da sottoporre all'IDS nella fase di scansione vera e propria del traffico.

Sfruttando le statistiche calcolate su dei dati forniti direttamente dagli apparati di rete, si vuole riuscire a identificare del traffico che utilizza dei protocolli noti ed è per tanto giudicabile non pericoloso con una buona probabilità. I dati ricevuti dagli switch corrispondono a dei pacchetti campionati sull'intero traffico. In particolare, le informazioni su cui sono elaborate le statistiche si limitano a sorgente, destinazione, porta e protocollo di ogni singolo pacchetto.

Lavorando su questi campi si vuole tenere traccia degli indirizzi IP che generano del traffico verso una sola destinazione, attraverso un unico protocollo, su una singola porta e i cui relativi campioni superano una determinata soglia definita dall'amministratore.

La riduzione vera e propria del carico di lavoro, avviene tramite l'inserimento di alcune regole di firewall che permettono il filtraggio dei pacchetti generati dalle sorgenti identificate in precedenza. Così facendo si vuole garantire che per un intervallo il traffico associato a tali indirizzi non sia rilevato dalla sonda IDS.

L'intero sistema è stato progettato in modo da poter ottenere una gestione quanto più dinamica possibile. Per questo motivo le principali operazioni corrispondono a dei moduli che comunicando tra di loro riescono a coordinarsi senza richiedere l'intervento dell'amministratore.

Struttura della tesi

- **Capitolo 1:** viene dato un quadro generale dell'infrastruttura di rete del Polo Scientifico Didattico di Cesena e vengono illustrati i meccanismi di funzionamento dei vari servizi riguardanti l'assegnazione degli indirizzi IP, la gestione dei file LOG e la procedura di autenticazione e autorizzazione degli utenti.
- **Capitolo 2:** confrontata con altre tecniche per il monitoraggio delle reti ad alta velocità, come RMON II e NetFlow, in questo capitolo, viene illustrata nel dettaglio la tecnologia sFlow utilizzata per fornire alla

nostra applicazione i dati in input necessari per eseguire una prima classificazione del traffico.

- **Capitolo 3:** descrive la fase progettuale durante la quale sono stati fissati gli obiettivi, offre una panoramica dell'applicazione, spiega il meccanismo di decodifica dei dati ricevuti in input e illustra le varie funzionalità nel dettaglio.
- **Capitolo 4:** riguarda la fase di realizzazione, vengono commentate dettagliatamente le varie parti del codice e motivate le scelte implementative.
- **Capitolo 5:** riporta i risultati ottenuti durante il calcolo delle prestazioni attraverso il commento di alcuni schemi.
- **Capitolo 6:** presenta le conclusioni relative al lavoro svolto e i possibili sviluppi futuri.

1 Rete del Polo di Cesena

In questo capitolo verrà analizzata l'infrastruttura di rete del Polo Scientifico-Didattico di Cesena [LUP10] nella quale è stato realizzato il progetto relativo a questa tesi e la cui architettura è stata schematizzata nella *figura 1.1*.

Una volta definita la topologia della struttura di rete, verrà descritto nel dettaglio il meccanismo di assegnazione degli indirizzi IP e dopo aver analizzato ogni singola macchina e il suo utilizzo, si concluderà con l'affrontare le varie fasi che compongono il processo di autenticazione e autorizzazione.

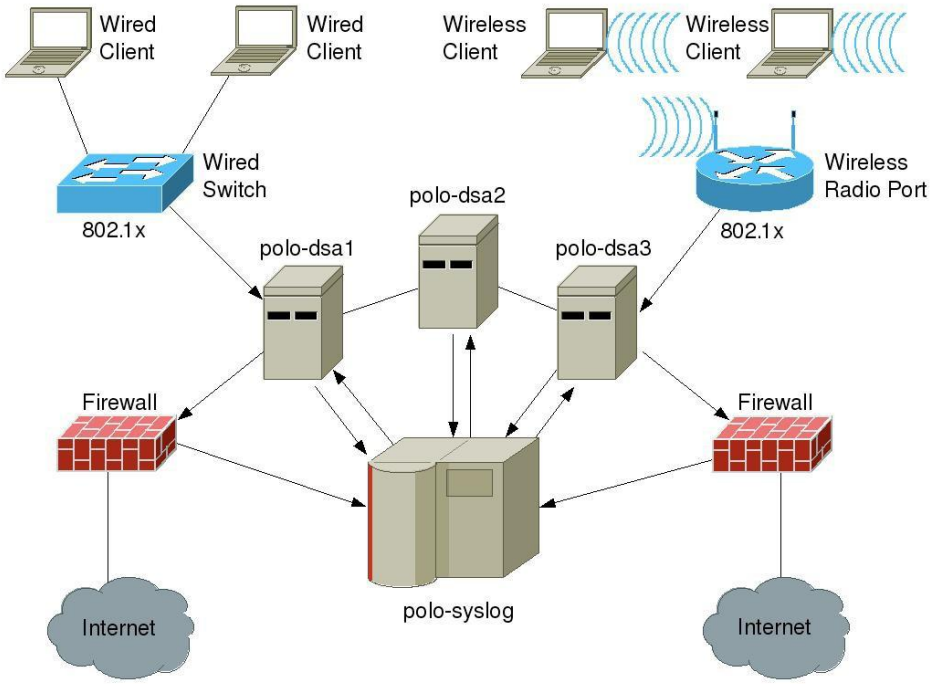

Figura 1.1 Architettura di rete del Polo di Cesena

1.1 Topologia

La struttura di rete del Polo è costituita da un anello in fibra *monomodale*[1] che collega tutte le strutture. All'apparato di *MAN*[2] sono collegati tutti gli *switch*[3] che forniscono connettività secondo uno schema a *stella*[4].

Attualmente sono installati e gestiti circa 100 apparati che forniscono connettività in ogni struttura del Polo di Cesena. Tutte le connessioni tra apparati sono in *Gigabit Ethernet*[5] e, dove richiesto, vengono fornite connessioni a 100 *Mbit/s*[6] o anche a 1000 Mbit/s nel caso di Server o Personal Computer che svolgono funzioni particolari.

1.2 Assegnazione indirizzi

L'assegnazione degli indirizzi IP[7] avviene in maniera dinamica, ossia la concessione di un IP a un dispositivo non è permanente ma temporanea. Questo meccanismo è stato adottato in modo da poter far fronte alla richiesta di un numero di utenti in continuo aumento.

Tale assegnazione è gestita da un protocollo di rete chiamato DHCP, acronimo di *Dynamic Host Configuration Protocol*, che valuta l'assegnazione in conformità a diversi parametri, come indirizzo fisico della macchina, lease-time (inteso come durata di connessione dell'indirizzo), sotto-rete eccetera.

Per una questione di sicurezza è stato necessario legare quest'assegnazione dinamica a un sistema di autenticazione che verrà affrontato in seguito.

1.3 Server di log

Il server di log, che prende il nome di **polo-syslog**, ha il compito di memorizzare in un registro tutte le attività svolte, al fine di garantire una possibile consultazione futura.

[1] La fibra monomodale presenta un core di diametro ridotto che riduce la dispersione modale permettendo una propagazione quasi rettilinea (monomodo) del raggio luminoso.
[2] Acronimo di Metropolitan Area Network la MAN è una tipologia di rete di telecomunicazioni con un'estensione limitata a perimetro metropolitano.
[3] Lo switch è un dispositivo di rete che inoltra selettivamente i frame ricevuti verso un preciso destinatario.
[4] La stella è una topologia di rete il cui numero di canali è uguale al numero di nodi meno uno e al centro si trova un dispositivo di rete.
[5] Gigabit Ethernet indica un protocollo di trasmissione ethernet operante a 1000 Mbit/s.
[6] Mbit/s è l'unità di misura che indica la capacità di trasmissione di un milione di bit al secondo.
[7] Acronimo di Internet Protocol è il protocollo su cui si basa la rete Internet.

Questo server, il cui sistema operativo è GNU/Linux, comunicando con tutte le altre macchine, esegue la registrazione di tutti i log riguardanti gli accessi alla rete. Tale servizio prende il nome di syslog-ng.

Syslog-ng è un'implementazione open source per sistemi Unix e Unix-like del protocollo Syslog. Questo servizio dà la possibilità di effettuare una configurazione in base alle proprie esigenze rendendone l'uso molto più accessibile. Syslog-ng interagisce grazie al *demone*[8] syslogd, cioè un *thread*[9] in esecuzione il quale assume un comportamento differente in base al tipo di richiesta ricevuta. Una caratteristica di questo demone è che, a differenza di quanto avviene in base alla configurazione di default di Syslog che prevede la memorizzazione degli eventi in un file di testo, permette l'inserimento dei log direttamente nel database MySQL.

Oltre a syslog-ng, sempre sulla stessa macchina, è presente un servizio open source chiamato ***arpwatch*** che monitora la rete alla ricerca di attività ARP.

Il protocollo ARP [ALL06], o *Address Resolution Protocol*, viene utilizzato all'interno di una rete per individuare l'indirizzo hardware di una macchina. Durante la richiesta di autenticazione la macchina client invia il suo indirizzo *MAC*[10] e, se va a buon fine, quest'ultimo viene associato a un indirizzo IP. La coppia IP – MAC viene così memorizzata da arpwatch insieme al timestamp (ovvero data e ora al momento della comunicazione) in modo che in futuro se questa dovesse cambiare, lo stesso arpwatch possa accorgersene ed emettere un avviso. Di norma arpwatch documenta l'accaduto inviando un e-mail all'amministratore del sistema, ma nel caso specifico è stato configurato in modo da inviare tali informazioni al server di log, il quale automaticamente provvede a caricarle nel database MySQL.

Questo servizio torna utile nel caso in cui una macchina già autenticata sulla rete tenti di cambiare il proprio indirizzo MAC o IP al fine, ad esempio, di camuffare la propria identità sulla rete.

[8] Nei sistemi operativi multitasking il demone indica un programma eseguito in background senza che sia sotto il controllo dell'utente.
[9] Il thread è la suddivisione di un processo in due o più filoni che vengono eseguiti con correntemente.
[10] Acronimo di Media Access Control è un codice a 48 bit assegnato in modo univoco a ogni scheda di rete ethernet.

1.4 Server di autenticazione

Come illustrato nella *figura 1.1* le macchine adibite all'autenticazione degli utenti, sono **polo-dsa1**, **polo-dsa2** e **polo-dsa3**. Tutte e tre le macchine implementano servizi analoghi e sono direttamente connesse con gli apparati che fanno da tramite con le macchine client. Questi dispositivi intermedi sono degli switch, nel caso in cui l'accesso avviene via cavo, mentre corrispondono a delle radio porte, gestite dal *WESM* [11], per l'accesso *wireless*[12].

Microsoft Windows Server 2003, il sistema operativo utilizzato su queste macchine, permette l'utilizzo di un servizio che prende il nome di IAS, acronimo di *Internet Authentication Service*, che è un elemento utile per la gestione del metodo AAA (*Authentication, Authorization and Accounting*). Secondo tale metodo, l'autorizzazione di accesso è preceduta da una verifica dell'autenticità la quale, se ha esito positivo, permette il caricamento del profilo interessato.

IAS nasce come implementazione del protocollo RADIUS (*Remote Authentication Dial In User Service*), il quale fornisce la gestione e il controllo degli utenti che cercano di connettersi a una rete protetta e presenta un'architettura di tipo client/server che opera a livello applicativo sfruttando il protocollo *UDP*[13]. Oltre all'autenticazione tramite server RADIUS, che tratteremo nei prossimi paragrafi, IAS supporta autenticazioni per macchine che possiedono un sistema operativo Microsoft Windows. In questo caso le informazioni per la verifica della validità delle credenziali vengono prese dall'*Active Directory*[14]. La novità di questa tecnologia è data dal fatto che partendo da una radice, nel nostro caso **dir.unibo.it**, è possibile creare diversi domini, ognuno dei quali possiede il proprio DC (*Domain Controller*[15]) che, appartenendo all'albero dell'AD (Active Directory), può recuperare facilmente le informazioni su tutti gli utenti compresi in uno o più domini dell'intera foresta dell'AD. Conservandone una replica di queste informazioni, il DC ha così la possibilità di verificare la validità delle credenziali inviate da un utente per autenticarsi.

[11] Acronimo di *Wireless Edge Service Module* è un modulo utilizzato per la gestione delle Radio Port hp.
[12] Il termine wireless indica una comunicazione tra dispositivi elettronici che non fa uso di cavi. Dall'inglese "senza fili".
[13] Acronimo di *User Datagram Protocol*, UDP è uno dei principali protocolli della suite di protocolli Internet. Abbinato al protocollo IP, gestisce il trasporto di pacchetti di rete. Essendo di tipo connectionless e non gestendo il riordinamento dei pacchetti né la ritrasmissione di quelli persi, è considerato di minore affidabilità rispetto al protocollo TCP, ma risulta comunque essere uno dei protocolli di trasporto più rapido.

L'utilizzo di questa tecnologia è stata resa possibile grazie al fatto che i server polo-dsa svolgono autonomamente la funzione di DC.

A ogni operazione lo IAS memorizza un log che viene inviato in copia al server di log centralizzato polo-syslog.

1.5 Firewall

Il firewall è un componente passivo di difesa perimetrale. Apparato di rete hardware o software, filtra tutti i pacchetti entranti e uscenti, da e verso una rete o un computer, applicando regole che contribuiscono alla sicurezza della stessa [WIK01].

Nel caso specifico dell'infrastruttura del Polo di Cesena, i firewall sono due, uno *master* e uno *slave*[16]. Entrambi sono ridondanti e in alta affidabilità, assicurando così un certo grado di assoluta continuità operazionale durante un periodo di tempo misurato.

Il carico di lavoro non è distribuito, ossia il firewall slave è in stand-by ed entra in esecuzione solo nel caso in cui il firewall master cessi di funzionare a causa di un guasto. Le sessioni aperte che viaggiano lungo il firewall, come ad esempio sessioni di posta o web, sono clonate sul firewall slave in modo che, chiamato in causa, quest'ultimo entri in funzione raccogliendo le informazioni dalla tabella delle sessioni duplicata.

Importante è sapere che i firewall in questione svolgono anche la funzione di DHCP, quindi oltre a controllare i pacchetti in transito, assegnano gli indirizzi IP alle macchine client.

Anche in questo caso il registro degli eventi viene duplicato sul server polo-syslog.

[14] Active Directory è un insieme di servizi di rete adottati dai sistemi operativi Microsoft a partire da Windows 2000 Server, che si basa sui concetti di dominio e di directory. L'insieme dei servizi di rete concessi dall'Active Directory può essere sfruttato dall'utente nel momento in cui esegue il *login* su una qualsiasi macchina appartenente al dominio. Il dominio dell'Active Directory si sviluppa a partire da un nodo radice dal quale si diramano tutti i nodi sottostanti che costituiscono i cosiddetti "siti" periferici ai quali sono connessi tutti i PC appartenenti al dominio.

[15] Nei sistemi Windows server, un domain controller (DC) è un server che risponde alle richieste di autenticazione nell'ambito di un dominio Active Directory.

[16] Master e Slave indicando rispettivamente il dispositivo principale, sul quale viene fatto maggiore affidamento, e quello secondario, che ricopre un ruolo di secondo ordine o subentra al primo in caso di guasti.

1.6 Proceduta di autenticazione e autorizzazione

1.6.1 Collegamento wired

L'autenticazione avviene tramite protocollo 802.1x. *"Tale protocollo è uno standard IEEE basato sul controllo delle porte di accesso alla rete LAN e MAN […] Questo standard provvede ad autenticare ed autorizzare dispositivi collegati alle porte di accesso (Switch e access point) stabilendo un collegamento punto a punto […] e si basa lui stesso su di un altro protocollo per l'autenticazione, chiamato EAP"* [JNT07].

Acronimo di *Extensible Authentication Protocol*, EAP [BJH04] è un protocollo di autenticazione utilizzato dagli AP (*Access Point*) nelle connessioni PPP (*Point to Point Protocol*). Il meccanismo di questo protocollo rimanda l'operazione di autenticazione a un server centralizzato, come RADIUS [RWS00]. Un tentativo di autenticazione genera un tunnel fra client e server attraverso il quale l'AP invia le credenziali inserite dal client che vengono elaborate dal RADIUS (*figura 1.2).*

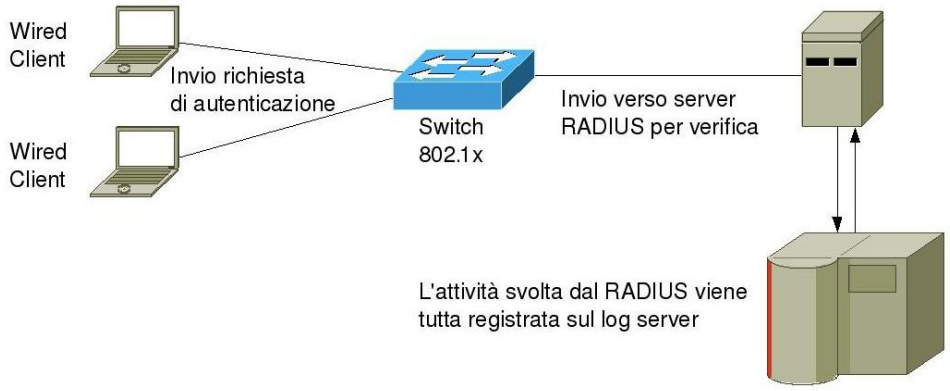

Figura 1.2 Fase di autenticazione wired

Se l'autenticazione ha esito positivo avviene l'autorizzazione che consiste nello spostamento del client dal tunnel temporaneo verso la vera e propria rete (*figura 1.3).*

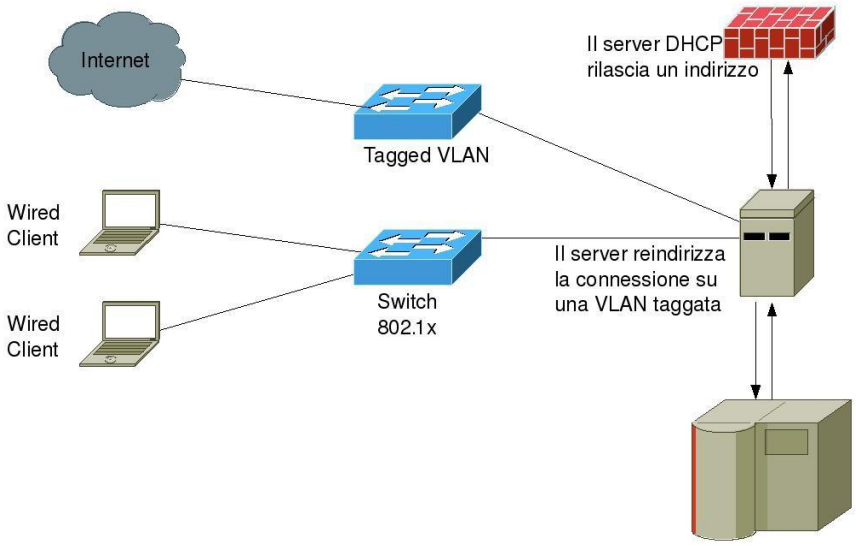

Figura 1.3 Fase di autorizzazione wired

Nel caso specifico della rete del Polo di Cesena, nella fase di autorizzazione il client viene indirizzato verso una VLAN.

Acronimo di *Virtual Local Network*, una VLAN 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 non comunicanti tra loro [WIK02].

Nella VLAN confluiscono tutti i client connessi alla rete che sono stati autenticati con successo. Questo avviene grazie al cosiddetto VLAN Trunking, ossia la possibilità di collegare fra loro due o più switch unendo le VLAN presenti in essi.

A ciascuna VLAN è associato un VLAN ID che permette di identificare a quale di queste appartiene il pacchetto inviato tra i due switch. Per rendere possibile tale identificazione, prima del pacchetto di rete viene incapsulato un preambolo di 2 Byte chiamato VLAN Tag, dove la porta su cui viaggia questo pacchetto viene definita Tagged o Trunk Port.

Nel momento in cui il client esce dal tunnel temporaneo creato dall'AP ed entra nella VLAN, il server DHCP riceve la richiesta di assegnazione indirizzo IP e sulla base di una tabella di IP assegnati dinamicamente, viene assegnato un indirizzo IP disponibile al client, che da quel momento può navigare in Internet passando attraverso il Firewall.

1.6.2 Collegamento wireless

Per i client che si collegano in wireless, il meccanismo di autenticazione avviene sempre attraverso il protocollo 802.1x ma è differente la modalità di connessione rispetto al collegamento via cavo che comunica con la rete direttamente attraverso uno switch.

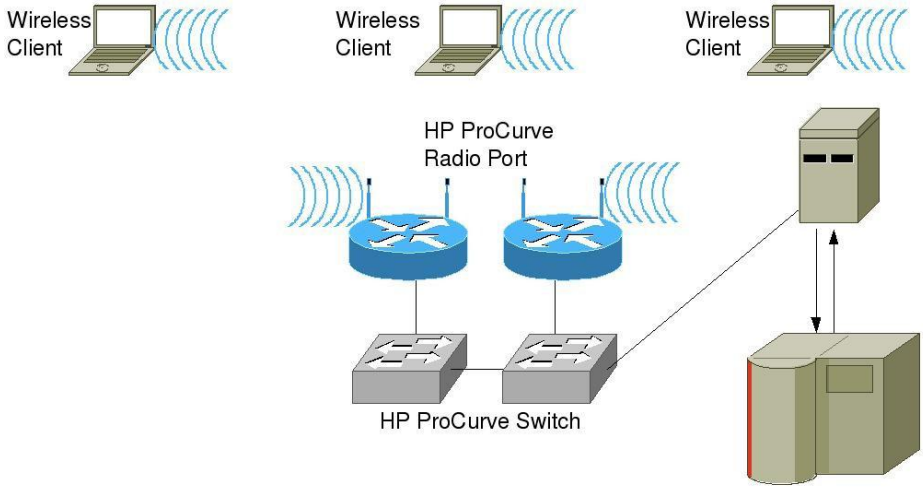

Figura 1.4 Fase di autenticazione wireless

In tutte le strutture del Polo Scientifico-Didattico di Cesena sono presenti più HP ProCurve Radio Port, ossia degli Access Point tutti logicamente collegati e gestiti dai WESM.

Come mostrato in *figura 1.4* in modalità wireless il client prima stabilisce una connessione con la Radio Port e solo successivamente, attraverso lo Switch al quale la Radio Port è collegata fisicamente, avviene l'autenticazione come già descritto in precedenza per la connessione wired.

Gli switch HP ProCurve cui le Radio Port sono connesse sono caratterizzati dal fatto di essere modulari. Permettono di aggiungere dei moduli hardware allo switch stesso per estenderne le funzionalità. Nel caso specifico gli switch possiedono un modulo definito WESM, acronimo di *Wireless Edge Service Module*, che ha reso possibile la centralizzazione di più Radio Port.

Quando un client tenta di collegarsi in wireless in realtà, stabilisce un collegamento con questo modulo WESM che, oltre a offrire diverse altre funzioni come accesso da remoto via SSH[17], analisi del traffico tramite IDS[18] e

Hotspot[19], funge da collettore di connessioni. Per rendere l'idea, l'invio delle credenziali al server RADIUS è un compito svolto dal modulo WESM.

Questi due switch ProCurve con moduli WESM supportano il load balancing, in altre parole il bilanciamento del carico di lavoro e sono utilizzati entrambi contemporaneamente raccogliendo le connessioni dei client in modo da uniformare il carico e aumentare le prestazioni.

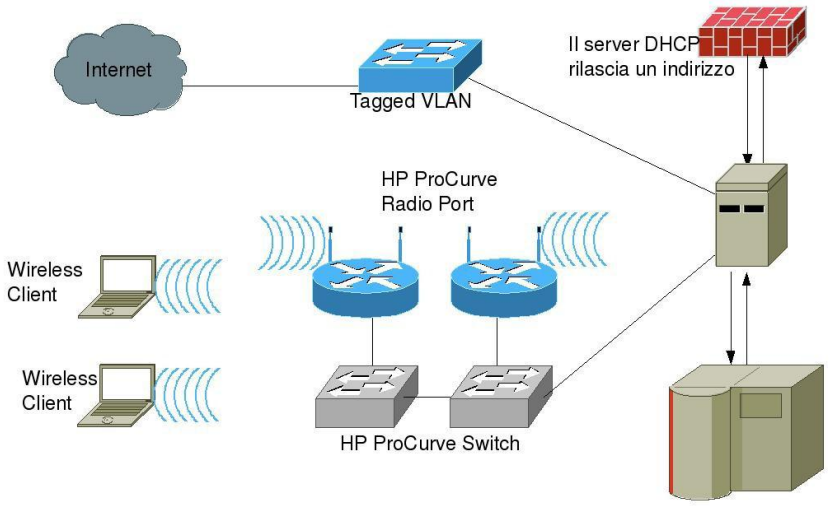

Figura 1.5 Fase di autorizzazione wireless

Anche la fase di autorizzazione, come mostrato in *figura 1.5*, avviene in maniera analoga alla connessione via cavo. L'unica differenza è dovuta al differente mezzo trasmissivo che si utilizza per la navigazione. Nelle connessioni wired ogni client utilizza un solo cavo come mezzo di collegamento al quale corrisponderà una connessione univoca e priva d'interferenze. Uno switch pertanto potrà avere al massimo tante connessioni quante sono le interfacce per il collegamento dei cavi.

Nelle connessioni wireless invece, ogni ProCurve Radio Port gestisce più connessioni contemporaneamente poiché il mezzo di trasporto è costituito dall'etere.

[17] Acronimo di *Secure SHell* è un protocollo che permette di stabilire una sessione remota cifrata ad interfaccia a riga di comando con un altro host.
[18] Acronimo di *Intrusion Detection System* è un dispositivo software o hardware utilizzato per identificare accessi non autorizzati ai computer o alle reti locali.
[19] Hotspot è un termine con cui ci si riferisce a un'intera area dove è possibile accedere su Internet in modalità senza fili attraverso l'uso di un router collegato a un provider di servizi Internet.

La tecnologia di base di una connessione wireless prevede, infatti, l'utilizzo di onde radio a bassa potenza. Tali onde vengono utilizzate per coprire ambienti eterogenei dove le diverse postazioni sono collocate, permettendo la connessione di più utenti alla stessa Radio Port.

Dato che il mezzo di trasporto utilizzato per la connessione è comune a tutti i computer connessi senza fili, si possono verificare interferenze nella connessione o errori nell'instradamento di pacchetti. Per questo motivo il traffico di ogni singola macchina viene isolato e per rendere possibile ciò viene creato un tunnel via software tra la Radio Port e l'AP in maniera univoca.

Il traffico indirizzato a uno specifico client passerà quindi solo ed esclusivamente attraverso quel tunnel virtuale.

2 Sistemi di monitoraggio

Le reti moderne sono sempre più veloci e complesse. Gestione e monitoraggio tradizionali non permettono di ottenere un'analisi dettagliata del traffico catturato che invece è in parte possibile con tecnologie più recenti come RMON II [WAL97], NetFlow [CLA04] e **sFlow** [SFL03].

In questo capitolo verrà trattata esclusivamente la tecnologia sFlow e solo infine verrà accennato il funzionamento delle altre.

2.1 Interrogazione dei dispositivi

Prima di affrontare quest'argomento è importante avere un'idea di come le informazioni relative al traffico che attraversano i dispositivi di rete vengano recuperate e utilizzate per il monitoraggio e l'analisi.

In una rete sono presenti solitamente router, switch, gruppi di continuità, eccetera. SNMP (*Simple Network Management Protocol*) è il protocollo standard per controllare da remoto questi dispositivi. Una console invia dei comandi SNMP per monitorare il loro stato, per configurarli e ricevere segnalazioni. Ogni apparato richiede che un comando abbia una sintassi particolare e spesso differente da un dispositivo all'altro. L'insieme dei vari parametri viene raccolto all'interno di un archivio virtuale definito MIB (*Management Information Base*).

Tutte le varie tecnologie accennate in precedenza utilizzano SNMP.

2.2 sFlow

Sviluppato da *InMon*[20], sFlow [SFL03][LIU09] è uno standard per il monitoraggio di switch e router ad alta velocità. La sua prima implementazione è stata rilasciata nel 2001.

sFlow è una tecnologia di campionamento che raccoglie delle statistiche dai dispositivi di rete ed essendo implementata su hardware, è applicabile a reti ad alta velocità.

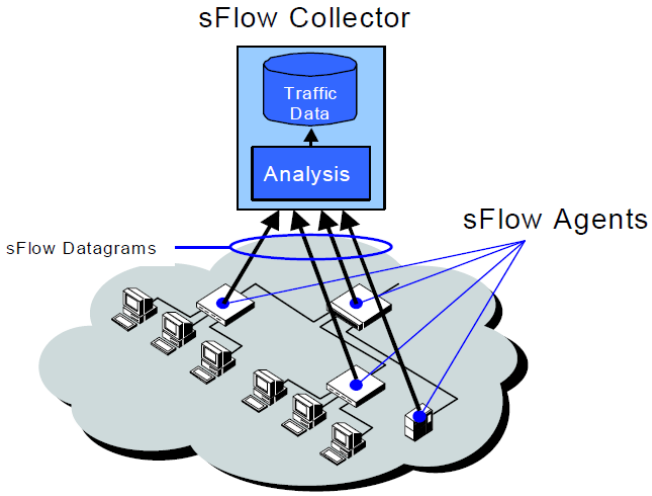

Figura 2.1 Schema generale di un sistema sFlow

Un sistema sFlow (*figura 2.1)* è costituito da **sFlow Agent** e uno **sFlow Collector**. Lo sFlow Collector è un server centrale che raccoglie e analizza gli **sFlow Datagram** inviati da tutti gli sFlow Agent. Uno sFlow Agent è l'implementazione su hardware di un meccanismo di campionamento (*figura 2.2)*, che comprende due modalità:

- **Counter Sampling**: un intervallo di *polling*[21] definisce quando il packet counter di una specifica interfaccia deve essere inviato allo sFlow Collector.
- **Packet Flow Sampling**: basato sulla frequenza di campionamento, 1 su N pacchetti viene catturato e inviato al server che funge da collettore.

[20] InMon è un'azienda che sviluppa sistemi per il monitoraggio del traffico di reti ad alta velocità e focalizza l'attenzione sull'uso di tecniche di campionamento statistico.
[21] Richiesta periodica di disponibilità di un componente al fine di effettuare una successiva eventuale interazione con esso

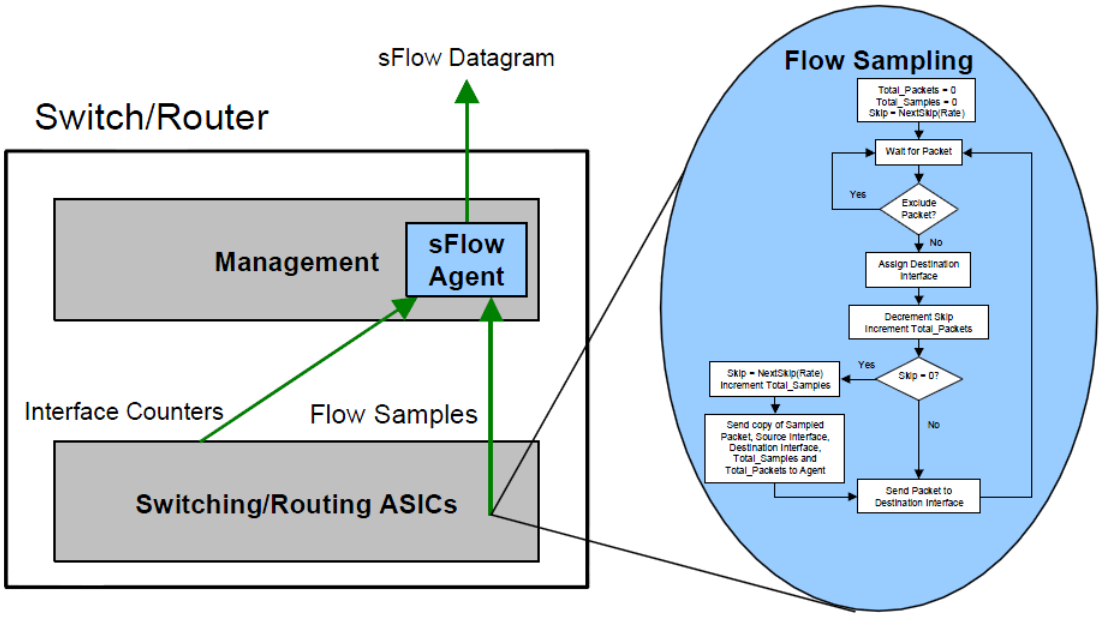

Figura 2.2 Schema generale del meccanismo di campionamento

2.2.1 Counter Sampling

Il sistema sFlow counter collector è costituito da diverse componenti software, come illustrato in *figura 2.3.*

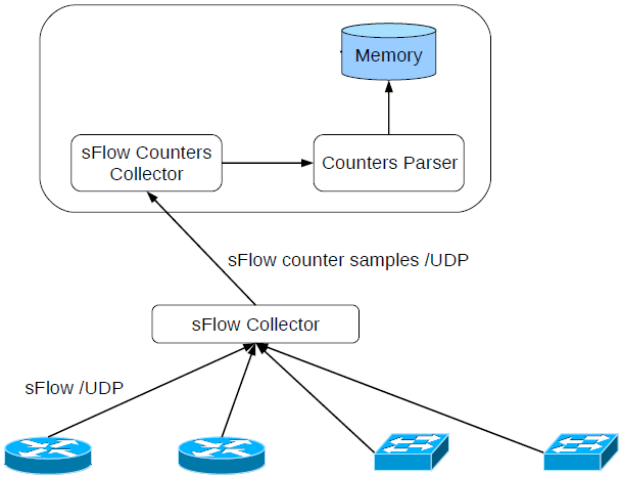

Figura 2.3 Architettura di uno sFlow Counter Collector

Lo sFlow Collector ha il compito di ricevere i pacchetti sFlow (nel formato sFlow Datagram) dai dispositivi di rete, decodificarli e trasmettere i **counter sample** (*figura 2.4)* allo sFlow counter collector.

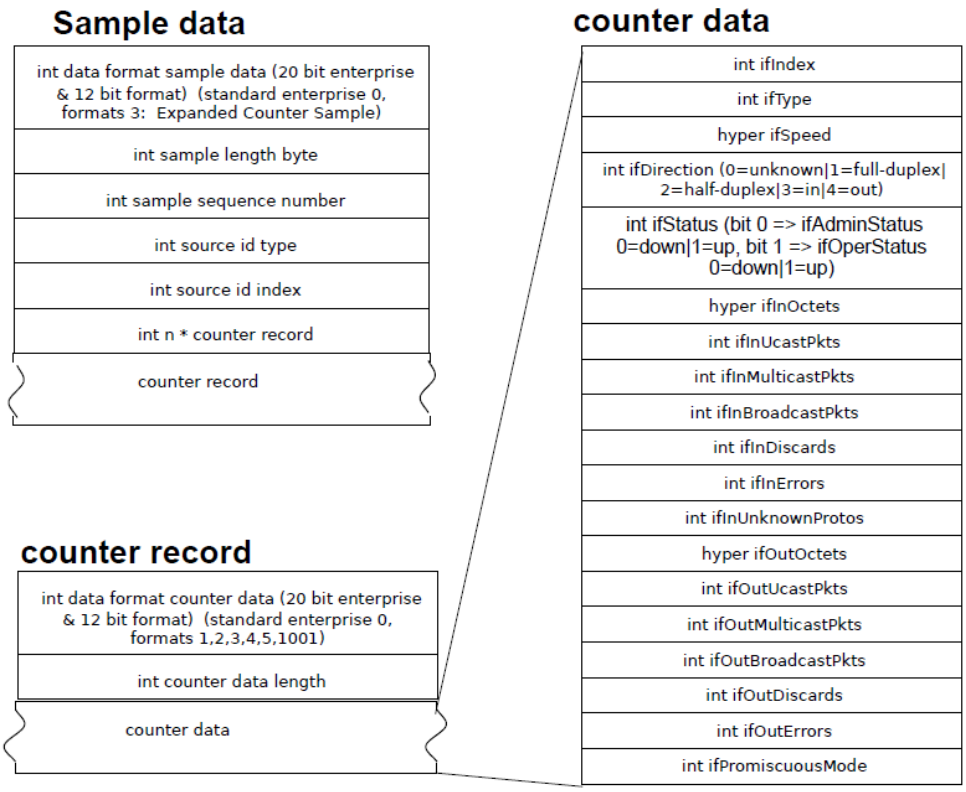

Figura 2.4 Formato di un Counter Sample

I counter sample relativi a tutti i contatori vengono recuperati tramite SNMP, operazione effettuata su ASIC (*Application Specific Integrated Circuit*) piuttosto che dalla CPU dello switch.

Quando lo Sflow counter collector riceve i counter sample, li decifra ulteriormente e trasmette al counter parser i counter, che verranno successivamente memorizzati in base all'interfaccia di appartenenza.

2.2.2 Packet Sampling

Le tradizionali tecnologie per la gestione della rete, come SNMP e RMON, non consentono di analizzare in profondità il contenuto del traffico monitorato. Per eseguire un'analisi dettagliata è necessario acquisire e scomporre singolarmente ogni pacchetto, operazione che risulta poco pratica e impossibile nel caso di reti ad alta velocità.

Una soluzione è di campionare il flusso del traffico ed esaminare solo un piccolo sottoinsieme di tutti i pacchetti attraverso un metodo statistico.

La preoccupazione principale e fondamentale del campionamento riguarda la precisione con il quale viene effettuato. Poiché il traffico fluttua in maniera imprevedibile e in modo dinamico, un inaccurato campionamento può portare a decisioni sbagliate da parte degli operatori di rete.

Studi statistici [PAP02] hanno dimostrato che, come mostra il grafico in *figura 2.5*, l'accuratezza aumenta proporzionalmente al numero di pacchetti campionati.

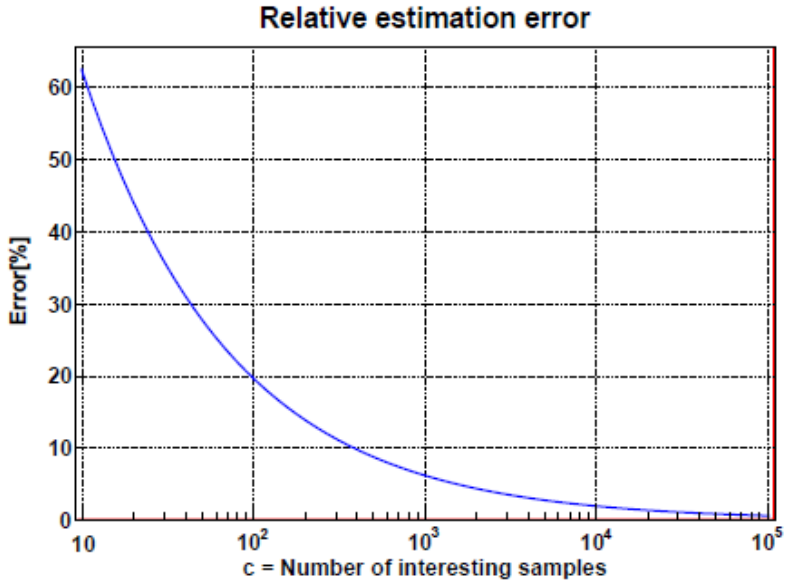

Figura 2.5 Errore di stima relativo al campionamento

Supponendo per esempio che il traffico HTTP venga trasmesso a una velocità di un pacchetto al secondo, dopo un'ora saranno stati generati 3600 pacchetti. Se la frequenza di campionamento è fissata a 0,25%, i campioni relativi dovrebbero essere circa 9 (3600*0.25/100). Indipendentemente dal traffico totale che attraversa la rete, tramite l'equazione $196*\sqrt{(1/c)}$ [JPB92], dove c indica i pacchetti campionati appartenenti a una determinata classe, si può definire la percentuale di errore. Nel nostro caso c indica i pacchetti appartenenti al flusso HTTP e l'errore è pari al 65%, che corrisponde alla possibilità di ottenere un numero di campioni che variano tra i 1200 e i 6000 pacchetti.

Per migliorare l'accuratezza di questa stima si possono utilizzare due metodi. Il primo prevede l'aumento della frequenza di campionamento, ad esempio portandola all'1% si abbasserebbe la percentuale di errore al 33%. Il secondo meccanismo, invece, consiste nell'ampliare l'intervallo di tempo in cui

viene campionato il traffico. Ad esempio calcolando le statistiche sulla base delle informazioni recuperate in un mese il margine di errore verrebbe ridotto al 2,4%.

2.2.2.1 Packet Flow Sampling

Il Packet Sampling implementato da sFlow genera un campione (**flow sample**) ogni N pacchetti osservati. Uno sFlow Datagram contiene almeno uno di questi campioni che, come mostrato in *figura 2.6*, racchiudono le informazioni sulla porta d'ingresso, porta di uscita e un *record* [22] di flusso che, in base al tipo di traffico, può contenere informazioni differenti sul flusso, come ad esempio l'header del pacchetto, il frame Ethernet, IPv4, IPv6 e altro.

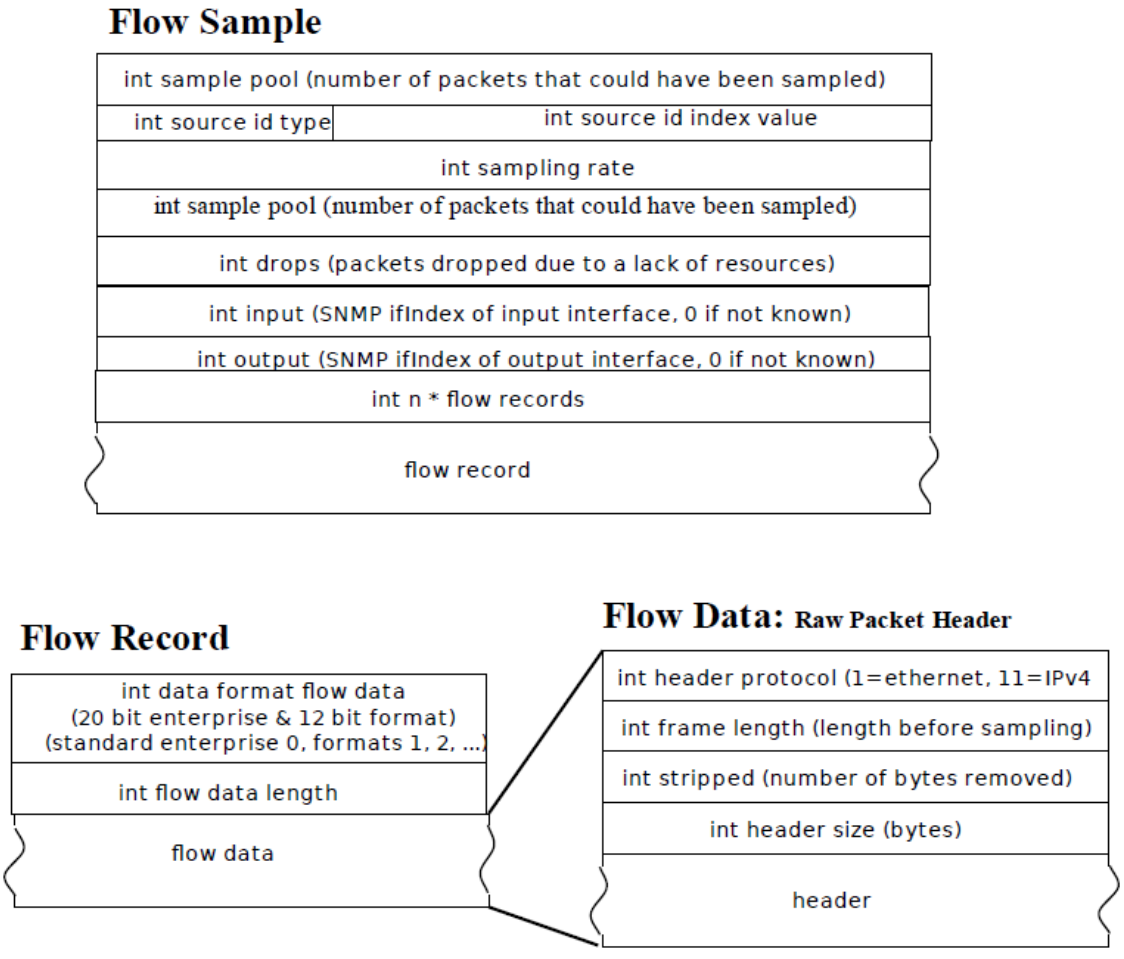

Figura 2.6 Formato di un Flow Sample

[22] In informatica il record è un oggetto di un database che contiene un insieme di campi o elementi. Viene spesso utilizzato per indicare semplicemente un insieme di informazioni relative a un argomento comune.

2.2.3 sFlow Datagram

Il formato dello sFlow Datagram [LAP04] utilizza lo standard *XDR*[23]. XDR è molto più compatto dell'*ANS.1*[24] e rende molto più semplice la codifica da parte dello sFlow Agent e la decodifica allo sFlow Collector.

Le impostazioni di default prevedono l'invio dei campioni, sottoforma di pacchetti UDP, sulla porta 6343. La mancanza di affidabilità nel meccanismo di trasporto UDP non influenza significativamente l'accuratezza del calcolo delle statistiche ottenute con lo sFlow Agent. Infatti, la durata tipica degli intervalli di polling, rende trascurabile la possibilità di perdita di una lunga sequenza di sFlow Datagram, tale da non permettere la ricezione di tutti i campioni associati a un counter.

Il *payload* [25] UDP contiene lo sFlow Datagram (*figura 2.7)* il quale fornisce le seguenti informazioni:

- Versione sFlow;
- Indirizzo IP del dispositivo originario;
- Numero di sequenza;
- Numero di campioni contenuti;
- Flow sample e Counter sample.

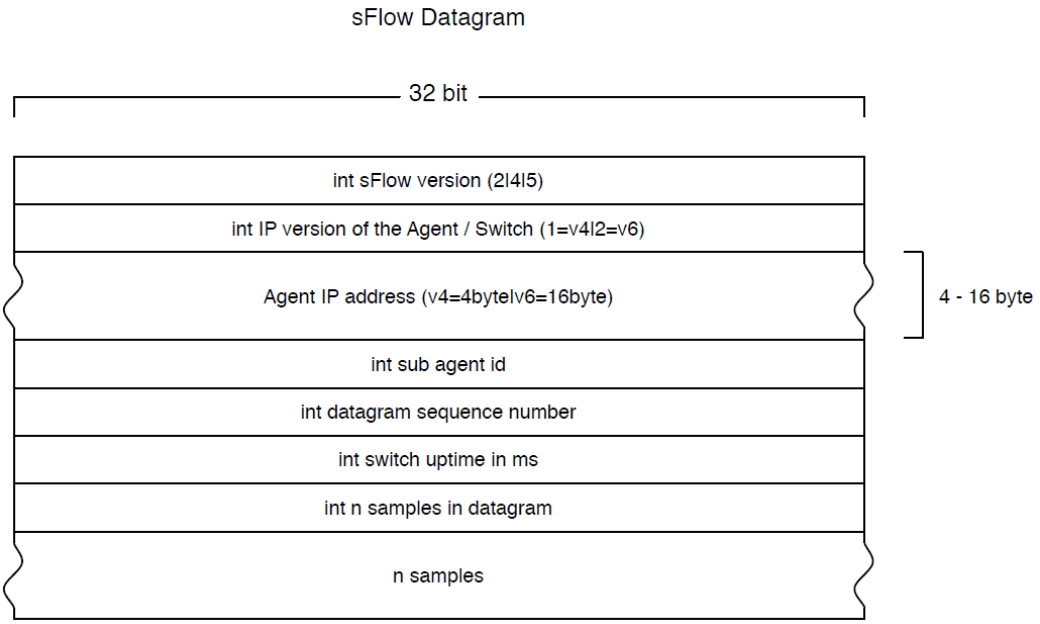

Figura 2.7 Formato di uno sFlow Datagram

[23] Acronimo di *External Data Rapresentation* è uno standard per la descrizione e la codifica dei dati che risulta essere utile per il trasferimento dei dati tra architetture diverse.

2.2.4 Altri approcci

Come accennato all'inizio del capito, esistono altri approcci per il monitoraggio della rete. I più conosciuti sono RMON e NetFlow.

2.2.4.1 Rmon

RMON è uno standard basato su SNMP che permette la gestione del traffico di rete. Nato inizialmente come semplice *sniffer*[25] remoto, con la versione successiva (RMON II) è stata estesa la possibilità di gestione del traffico a livello protocollo, con conseguente aumento delle risorse necessarie al suo agente per il recupero delle informazioni. Proprio quest'ultimo aspetto è una limitazione che non ha permesso di raggiungere risultati eccellenti, e spesso ha portato al solo utilizzo delle funzionalità base di questa tecnologia, ovvero all'implementazione solo di una parte delle funzioni disponibili. Per questo motivo spesso si parla di RMON 4 groups, il quale indica che dei 9 gruppi resi disponibili dallo standard, solo 4 vengono utilizzati (Statistics, History, Alarms, Events).

2.2.4.2 NetFlow

NetFlow è un protocollo della Cisco. Un *probe*[26], solitamente ospitato all'interno di un dispositivo di rete come router o switch, analizza il traffico e crea un record per ogni flusso. Quest'ultimo viene esportato a un collector che li analizza.

A differenza della tecnologia sFlow, NetFlow non utilizza meccanismi di campionamento ma analizza ogni singolo pacchetto.

[25] Sniffing si definisce l'attività di intercettazione passiva dei dati che transitano in una rete telematica.
[26] Tradotto con il termine "sonda", in informatica indica un dispositivo che svolge la funzione di analisi.
[24] Acronimo di *Abstract Sntax Notation One* è uno standard che descrive la struttura per la rappresentazione, codifica, trasmissione e decodifica dei dati.
[25] Tradotto con il termine "carico utile", indica generalmente ciò che viene trasportato da un mezzo o servizio di trasporto.

2.2.5 Confronto con altre tecnologie

La tabella presente nella *figura 2.8* mette a confronto le caratteristiche dello sFlow con le soluzioni basate su RMON e Cisco NetFlow.

RMON (4 groups) indica le quattro funzioni di base offerte da RMON (Statistiche, Cronologia, Allarmi, Eventi) che spesso vengono implementate nelle interfacce degli switch.

RMON II si riferisce all'intera implementazione della tecnologia RMON II, la quale tipicamente è presente nelle sonde hardware.

NetFlow fa riferimento alla tecnologia parte integrante del Cisco ISO Software.

	RMON (4 groups)	RMON II	NetFlow®	sFlow®
Packet Capture	N	Y	N	P
Interface Counters	P	P	N	Y
Protocols				
Packet headers	N	P	N	Y
Ethernet/802.3	N	Y	N	Y
IP/ICMP/UDP/TCP	N	Y	Y	Y
IPX	N	Y	N	Y
Appletalk	N	Y	N	Y
Layer 2				
Input/output interface	N	N	Y	Y
Input/output priority	N	N	N	Y
Input/output VLAN	N	N	N	Y
Layer 3				
Source subnet/prefix	N	N	Y	Y
Destination subnet/prefix	N	N	Y	Y
Next hop	N	N	Y	Y
BGP 4				
Source AS	N	N	P	Y
Source Peer AS	N	N	P	Y
Destination AS	N	N	P	Y
Destination Peer AS	N	N	P	Y
Communities	N	N	N	Y
AS Path	N	N	N	Y
Real-time data collection	Y	Y	P	Y
Configuration				
Configurable without SNMP	N	N	Y	Y
Configurable via SNMP	Y	Y	N	Y
Low Cost	Y	N	N	Y
Scalable (switch interfaces/collector)	P	N	N	Y
Wire-speed	Y	P	P	Y

N	Feature not supported
P	Feature partially supported. Either the feature is incomplete or can only be enabled by disabling other features.
Y	Fully supported

Figura 2.8 Tecnologie a confronto sulla base delle informazioni recuperate

È evidente come grazie al meccanismo di campionamento la tecnologia sFlow riesca a ottenere maggiori informazioni le quali, anche se non si riferiscono all'intero traffico, sono sufficienti per rendere più efficaci sistemi di monitoraggio e analisi. NetFlow è la tecnologia che più si avvicina allo sFlow poiché è in grado recuperare informazioni su tutti i livelli a differenza di RMON II che si limita alle informazioni relative ai protocolli. Ovviamente la scelta da parte di NetFlow di analizzare ogni singolo pacchetto non permette di ottenere informazioni più dettagliate a causa di elevate risorse computazionali che risulterebbero necessarie.

3 Progetto

L'analisi del traffico della rete è un'attività necessaria per diverse ragioni, che vanno dalla ricerca dei cosiddetti colli di bottiglia, alla scoperta di pattern di traffico anomalo. Quest'ultimo aspetto ricopre un ruolo di primaria importanza nella rete del Polo di Cesena. Infatti, il numero in continuo aumento di utenti collegati alla rete, comporta un rischio sempre maggiore di macchine infette, causa di potenziali incidenti informatici. La necessità di poter garantire un servizio in grado di identificare problemi di vulnerabilità e prevenire attacchi alla rete è così sfociata nell'utilizzo di un IDS open source, OSSIM [LUP10].

A causa dell'enorme quantità di dati che attraversano la rete, il detector non garantiva un'adeguata analisi dell'intero traffico che gli veniva sottoposto. È nata così l'esigenza di trovare una soluzione a questo problema.

In questo capitolo verranno illustrate le fasi di progettazione della nostra applicazione. Partendo da quelli che sono gli obiettivi e dai requisiti necessari, si passerà all'analisi dettagliata delle varie funzionalità dopo aver definito uno schema generale.

3.1 Obiettivi

L'idea di questo progetto è legata alla necessità di ridurre il carico di lavoro dell'IDS. L'approccio utilizzato è esclusivamente software, poiché il *detector*[27] è già installato su hardware con ottime capacità di calcolo.

[27] Strumento che sorveglia la rete e opera attivamente attraverso l'esecuzione di azioni successive alla lettura di un determinato pacchetto. Gli IDS sono i detector in commercio più usati.

Oltre a mettere l'IDS nelle condizioni ottimali per garantire un corretto funzionamento, si è cercato di rendere l'applicativo adattabile alle risorse hardware disponibili, che non sempre possono essere di alto livello come nel nostro caso, e soprattutto si è cercato di fare in modo che il sistema svolga il suo compito nel modo più dinamico e automatizzato possibile.

3.2 Requisiti e panoramica del sistema

Per raggiungere gli obiettivi fissati è stato ovviamente necessario sviluppare una serie di servizi, che verranno di seguito brevemente illustrati attraverso l'analisi di uno schema generale (*figura 3.1)*, in modo da offrire una chiara panoramica dell'intero sistema e riuscire a comprendere meglio nel dettaglio le varie funzionalità descritte nei prossimi paragrafi.

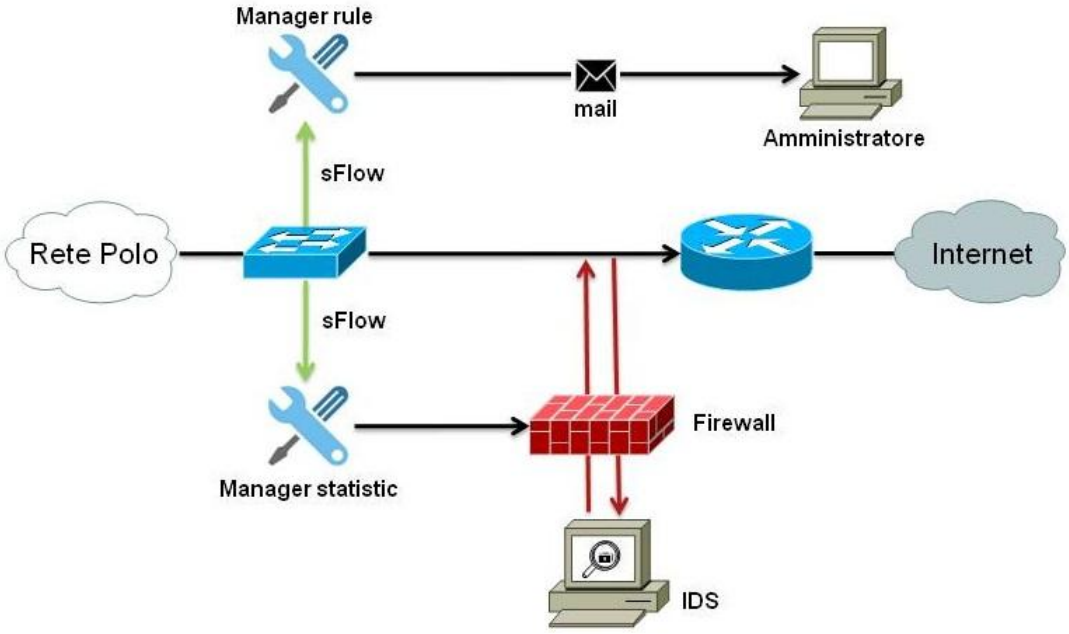

Figura 3.1 Schema generale del sistema

È stata scelta una struttura modulare in modo da poter avere la possibilità in futuro di introdurre facilmente nuove funzionalità e rendere l'intera applicazione molto flessibile alle esigenze dell'amministratore.

Come già detto, avendo individuato nell'eccessivo traffico da analizzare il collo di bottiglia che impediva all'IDS di eseguire una corretta identificazione di eventuali minacce alla rete, l'applicazione creata punta principalmente a ridurre il carico di

lavoro intervenendo direttamente sulla mole di dati da sottoporre alla scansione del detector. Elaborando le informazioni ricevute dagli apparati di rete che sfruttano la tecnologia sFlow, una parte dell'applicativo, simulando il comportamento di una *whitelist*[28], invia dei comandi al firewall presente sulla stessa macchina che ospita l'IDS, in modo da evitare di sottoporre alle interrogazioni di quest'ultimo il traffico ritenuto innocuo.

Un secondo servizio invece, sempre lavorando su dati relativi al traffico forniti dagli switch, ha il compito di emettere un avviso nel caso in cui si verificano dei tentativi di connessione utilizzando porte o protocolli non consentiti. Più precisamente l'eventuale azione che ne consegue consiste nell'informare l'amministratore attraverso una mail.

3.3 Informazioni fornite dalla tecnologia sFlow

Come accennato, le varie azioni che possono scaturire dal nostro sistema sono una conseguenza dei risultati ottenuti sulla base dei dati che vengono processati. Queste informazioni d'input vengono generate attraverso l'utilizzo della tecnologia sFlow.

Essendo il nostro scopo principale quello di eseguire una prima classificazione del traffico, se dovessimo lavorare analizzando ogni singolo pacchetto ci troveremmo nella stessa e identica situazione dell'IDS, senza aver fatto nessun significativo passo avanti. Per questo motivo è stata sfruttata la tecnologia sFlow presente sugli switch HP ProCurve installati nella rete del Polo di Cesena.

Questa tecnica si adatta perfettamente a quelle che sono le nostre esigenze, infatti, ci permette di lavorare su un numero minore di dati corrispondenti a delle statistiche calcolate non sull'intero traffico che attraversa gli apparati di rete ma, come descritto nel *capitolo 2*, su dei campioni. Infatti, l'utilità di questa tecnica è legata al fatto che, nonostante i dati forniti vengono ottenuti attraverso meccanismi di campionamento, l'accuratezza di tali informazioni dipende esclusivamente dal numero di campioni recuperati, che noi stessi possiamo definire.

[28] Consiste in un elenco di entità che godono di un particolare privilegio. Nel nostro caso corrispondono all'elenco di indirizzi IP che possono essere sottratti al controllo perché il loro traffico non è ritenuto pericoloso.

3.3.1 Decodifica delle informazioni

sFlow Datagram costituisce il formato con cui gli switch forniscono i dati alla nostra applicazione.

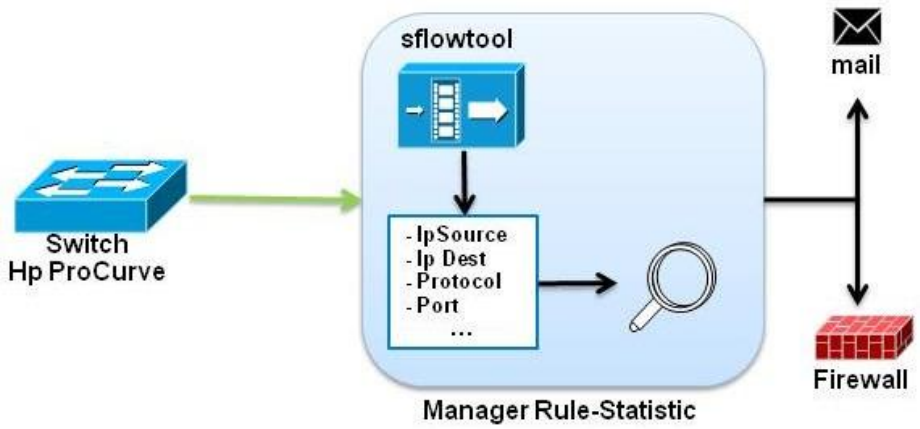

Figura 3.2 Fase di decodifica di uno sFlow Datagram

La decodifica di questi pacchetti (*figura 3.2)* per il recupero delle informazioni avviene tramite uno strumento software creato dall'InMon corporation (*sflowtool).* Quando viene eseguito, il tool si mette in ascolto e attende la ricezione da parte degli apparati di rete dei rispettivi datagrammi (UDP) che dovrà processare per estrarre le informazioni. La formattazione dei dati ricavati consiste in una semplice riga di testo (*figura 3.3.)*

Ip Agent	Input Port	Output Port	...	In Vlan	Out Vlan	Ip Source	Ip Dest	Protocol	...	Port Source	Port Dest	...	Size Sample	...
Source			Layer 2			Layer 3/4							Byte	

Figura 3.3 Formattazione dell'output prodotto da "sflowtool"

Rimandando al capitolo precedente per avere maggiori dettagli sulle informazioni trasmette dagli switch attraverso gli sFlow Datagram, quelle che meritano la nostra attenzione si limitano ai seguenti campi:

- **IP Agent**: indica l'indirizzo IP dell'apparato di rete che ha inviato lo sFlow Datagram.
- **Input-Output Port**: indicano le porte dello switch su cui è stato ricevuto (input) il pacchetto campionato e successivamente inoltrato (output).

- **In - out VLAN**: corrispondono ai VLAN-Id di eventuali Virtual LAN cui è associato il rispettivo pacchetto campionato. Entrambi i campi presenteranno un valore identico visto che il passaggio a una VLAN differente può avvenire solo quando gli stessi switch implementano delle regole di routing, cosa che nel nostro caso non avviene.
- **Source-Destination IP**: semplicemente indicano l'indirizzo IP (Source) della macchina che ha generato il flusso di traffico cui appartiene il pacchetto campionato, e l'indirizzo IP (Destination) della macchina destinato a riceverlo.
- **Protocol**: assume il valore numerico del rispettivo tipo di protocollo utilizzato per la comunicazione (Esempio: 6 = TCP).
- **Source-Destination Port**: corrispondono rispettivamente alla porta (source) attraverso la quale il client ha inviato il pacchetto, e la porta (destination) sulla quale il server è in ascolto per riceverlo.
- **Size Sample**: indica la grandezza del pacchetto campionato.

3.4 Dettagli delle funzionalità

Avendo un quadro generale del meccanismo di funzionamento del sistema, di quali servizi offre e quali sono i dati su cui lavora, è possibile descrivere con maggiori dettagli quelle che sono le varie funzionalità della nostra applicazione.

3.4.1 Segnalazione via e-mail

La segnalazione di allarmi attraverso l'invio di una mail all'amministratore di rete, può essere uno strumento efficace per far fronte a incidenti informatici. Ovviamente questa tecnica necessita della presenza fisica dell'amministratore di rete, quindi ha il limite di non poter garantire sempre un intervento immediato nel caso in cui si verifichi una minaccia. Per questo motivo abbiamo focalizzato maggiormente la nostra attenzione su aspetti più dinamici e abbiamo implementato questo metodo solo nella fase di analisi che

controlla esclusivamente porte e protocolli utilizzati, da macchine non autorizzate, per comunicare.

Questo servizio non è stato attivato per un'esigenza legata alla gestione della rete del Polo ma semplicemente per intervenire personalmente nel caso in cui si verificano degli eventi noti, causa di eventuali incidenti informatici. Un esempio può essere il tentativo di un attacco alla rete che sfrutta delle *backdoor*[29] per raggirare i sistemi di sicurezza attivi. Essendo oramai di dominio pubblico la conoscenza di quali porte vengono utilizzate solitamente per eseguire queste azioni è sufficiente tenerle sotto controllo ed emettere un avviso nel caso in cui accada l'evento.

Il livello di efficienza atteso da quest'attività non è elevato visto che l'intera applicazione ha uno scopo ben diverso e inoltre come detto prima non vi è la garanzia che tutto il traffico osservato sia sottoposto ad analisi. Questo metodo garantisce solo un livello di sicurezza leggermente maggiore.

[29] Paragonabili a porte di servizio, consentono di superare in parte o in tutto le procedure di sicurezza attivate in un sistema informatico.

3.4.2 Identificazione traffico innocuo

Il ruolo di maggiore importanza per raggiungere l'obiettivo principale, consiste nel riconoscimento del traffico che con buona certezza può essere ritenuto non pericoloso per la sicurezza della rete.

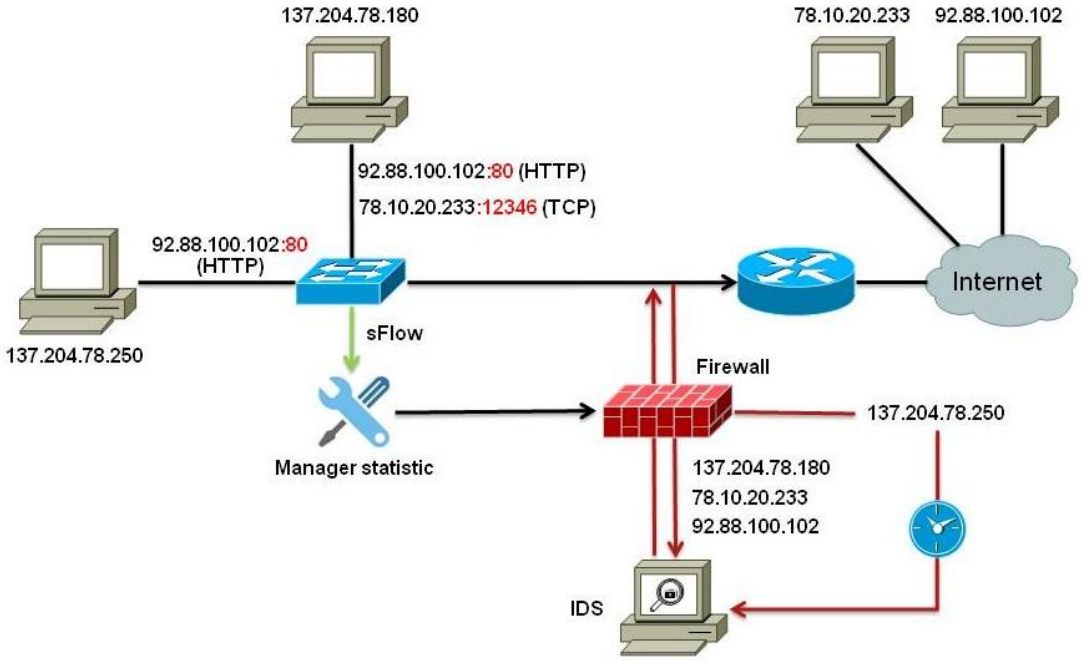

Figura 3.4 Fase d'identificazione del traffico innocuo ed esclusione dal controllo

Calcolando quante *sessioni*[30] vengono generate da uno specifico IP e basandosi sulla conoscenza delle porte e dei protocolli che solitamente non sono coinvolti in incidenti informatici, si omette dalla scansione dell'IDS il traffico generato da tale indirizzo IP per un preciso intervallo di tempo. Nel dettaglio quello che avviene è un'esclusione temporanea dal controllo di tutti i pacchetti che presentano come sorgente un indirizzo IP che ha una sola sessione attiva, attraverso un singolo protocollo e su un'unica porta, non ritenuti pericolosi dall'amministratore (*figura 3.4).*

[30] Connessione tra due entità utilizzata per trasferire i dati in entrambi i sensi per tutta la durata del collegamento.

Quest'approccio risulta ancora più affidabile dato che, campionando secondo metodi statistici, gli switch inviano un numero maggiore di pacchetti contenenti informazioni relative a sorgenti che generano un elevato traffico. Questo equivale a permettere al sistema di avere una quantità di dati maggiore per classificare con più attendibilità il traffico relativo alla rispettiva sorgente e, allo stesso tempo, nel caso in cui il flusso di dati in questione viene riconosciuto come innocuo, consente di omettere dal controllo del detector una notevole mole di dati.

3.4.3 Gestione del firewall IDS

Il servizio che si occupa della gestione del firewall, installato sullo stesso server su cui è presente l'IDS, in realtà non consiste in una vera e propria funzionalità di questo applicativo. Il suo compito, come appena affermato, si limita semplicemente a bloccare il passaggio di traffico ritenuto non pericoloso durante la fase d'identificazione, e riabilitarlo dopo un intervallo di tempo.

La scelta di creare un modulo a sé stante oltre a essere stata effettuata per rimanere fedeli alla struttura modulare e avendo così la possibilità di inserire nuove funzionalità in futuro senza doversi preoccupare nuovamente di come gestire il filtraggio del traffico, è dovuta soprattutto alla gestione dinamica degli indirizzi IP che la rete del Polo di Cesena adotta. Infatti, è possibile che l'indirizzo IP sospeso momentaneamente dal controllo, in futuro venga assegnato a una macchina differente che, come tutte, necessità di essere controllata perché possibile fonte di pericolo per la sicurezza del sistema.

3.5 Modifiche in corso d'opera

L'approccio utilizzato per far fronte a questa circostanza non è stato da subito quello in precedenza descritto. In un primo momento il criterio adottato prevedeva di attuare una politica inversa a quella che poi effettivamente è stata implementata. I dati da elaborare erano comunque forniti attraverso la tecnica sFlow che attuava i suoi processi di campionamento, ma proprio andando ad approfondire quest'ultimo aspetto di tale tecnologia si è notato la scarsa efficienza e il largo margine di errore in cui si rischiava di cadere.

3.5.1 Approccio originario

L'idea iniziale prevedeva un intervento drastico. Per ridurre notevolmente la percentuale di attività dell'IDS, si era pensato di impostare il firewall in modo da bloccare interamente il traffico, lasciando totalmente inattivo il detector nella fase di avvio. Il compito della nostra applicazione consisteva quindi nell'identificare i flussi di dati che potevano rappresentare una minaccia e, conoscendo l'indirizzo IP sorgente, sbloccarlo e permettere l'analisi del rispettivo traffico.

Con le conoscenze attuali è evidente che questo metodo non è appropriato per quelli che sono i nostri scopi. È, infatti, difficile se non impossibile classificare del traffico come possibile minaccia senza utilizzare i dati di payload di ogni singolo pacchetto, azione che non è resa possibile sulla base delle informazioni estratte tramite sFlow. Inoltre, dopo aver approfondito come avviene il campionamento dei pacchetti che transitano sui vari switch e aver appreso che non tutti i flussi di traffico generati dalle varie sorgenti hanno la garanzia di essere campionati, è sorto il problema di come gestire proprio questo traffico di cui non viene fornito nessun campione e di conseguenza non si ha nessuna informazione.

La prima possibile soluzione a questo problema è poi stata quella definitiva. La decisione, infatti, è stata quella di lasciare il compito all'IDS di scovare eventuali minacce, incarico per il quale non esiste applicazione migliore, e assegnare al nostro applicativo la competenza di eliminare dal controllo solo il traffico che con un'alta probabilità può essere giudicato non pericoloso.

4 Implementazione

In questo capitolo spiegheremo com'è stata sviluppata la nostra applicazione partendo da quelle che sono le scelte implementative, per poi analizzare nel dettaglio le singole funzioni.

4.1 Scelte implementative

L'intera applicazione è stata sviluppata in linguaggio C, utilizzando come ambiente di sviluppo Xcode (versione 3.2.4), che consiste in un pacchetto per lo sviluppo software su sistemi Mac OS X, il quale comprende anche il compilatore multi-target GCC.

4.2 Schema generale

L'applicazione è composta da tre moduli, ognuno dei quali svolge uno specifico compito:

- **ManagerSflowStat**: si occupa di decodificare i dati ricevuti attraverso gli sFlow Datagram, per poi processarli e calcolare le statistiche utili per intervenire sul filtraggio del traffico potenzialmente innocuo.
- **ManagerSflowRule**: estratte le informazioni dagli sFlow Datagram che riceve, esegue una verifica sui campi corrispondenti alla porta e al protocollo per garantire che non siano violate delle regole e nel caso in cui questo avviene, emette un avviso.
- **ManagerFirewall**: lavorando a livello di rete (Layer 3 ISO/OSI), gestisce le regole di filtraggio modificando la tabella del firewall sulla

base degli indirizzi IP che gli vengono inviati dal modulo ManagerSflowStat.

4.3 Strutture dati comuni

Esclusa la componente per la gestione del firewall, in entrambi quelle restanti, la memorizzazione dei dati ricevuti e decifrati avviene su delle strutture dati comuni ma non condivise fra i diversi moduli.

4.3.1 Matrice *mat*

Consiste in una matrice di puntatori a caratteri utilizzata per memorizzare l'output restituito dal tool che si occupa dell'interpretazione dei dati inviati dagli apparati di rete.

```
char *mat[MAX_RIGHE];

for (w=0; w<MAX_RIGHE; w++)
      mat[w]=(char*)calloc(MAX_COLONNE,sizeof(char));
```

Implementata per velocizzare le successive fasi di calcolo, le sue dimensioni sono definite attraverso dei parametri presenti in un file di configurazione. Si è deciso di allocare staticamente la memoria semplicemente perché il numero di righe, corrispondente al numero di pacchetti campionati, non varia durante l'esecuzione del programma, così come il numero di caratteri di ogni singola riga non supera mai una determinata soglia.

4.3.2 Struttura *elemento*

Costituisce l'elemento base di una lista in cui vengono memorizzati i soli campi di nostro interesse di ogni singolo pacchetto campionato.

```
struct elemento{
      char *ipS;
      char *ipD;
      int protocol;
      int port;
      int occ;
      int ctrl;
      struct elemento *ptr;
};
```

Il significato delle variabili della struttura è auto esplicativo dato che consistono in delle informazioni relative al traffico di rete. Fanno eccezione le due variabili *occ* e *ctrl* le quali vengono utilizzate solo durante il calcolo delle statistiche. La prima indica il numero di occorrenze di pacchetti campionati presenti nella matrice *mat* e appartenenti allo stesso flusso di dati, mentre la seconda è utilizzata come variabile di controllo che indica se l'indirizzo IP sorgente corrispondente deve essere escluso o no dalla scansione dell'IDS.

Come vedremo nei prossimi paragrafi, una lista non è la soluzione migliore a livello di complessità computazionale, ma, avendo maggiore interesse a individuare il meccanismo più adatto al nostro scopo e di conseguenza dover effettuare diverse modifiche al codice, è stata scelta perché è più semplice e veloce da implementare a differenza di altre strutture dati, come liste ordinate o hash table, che sono sicuramente più efficienti.

4.4 Modulo per il controllo delle regole

Esegue un controllo puntuale sul valore dei campi corrispondenti a porta e protocollo di un singolo pacchetto campionato senza tenere nessuna traccia dei riscontri precedenti. Il *main* è composto da due funzioni che si occupano di gestire le regole e dall'inizializzazione ed esecuzione di due *thread* associati rispettivamente alle funzioni di caricamento e controllo dei dati.

```
int main(int argc,char* argv[])
{
      …

      loadRules(rules);
      splitRules(rules,campoRules);

      pthread_mutex_init(&shared.MUTEX, NULL);
      pthread_mutex_init(&shared.MUTEX_STAT, NULL);
      pthread_mutex_init(&shared.MUTEX_LOG, NULL);
      pthread_cond_init(&shared.STAT, 0);
      pthread_cond_init(&shared.LOG, 0);

            pthread_t scrittura;
            pthread_t lettura;

            err=pthread_create(&scrittura,NULL,sflow,(void*)argv[1]);

            …
```

```
	err = pthread_create(&lettura,NULL,controllo,NULL);

	…

	pthread_join(scrittura, NULL);
	pthread_join(lettura, NULL);

	return 0;
}
```

4.4.1 Gestione delle regole

Le regole consistono in una coppia di valori corrispondente a porta e protocollo che l'amministratore di rete ha la necessità di porre sotto controllo. Per garantire una gestione semplice e dinamica queste regole vengono memorizzate in un file di testo che viene letto all'avvio del servizio.

La funzione *loadRules* svolge proprio questo compito, apre un file di testo e carica nella matrice *rules*, passata dal main per riferimento, le varie regole presenti.

```
void loadRules(char **rules)
{
	FILE *fd;
	int i = 0;
	char buf[DIM_BUFFER];
	char *res;

	fd=fopen("fileRules.txt", "r");

	…

	while(1)
	{
		res=fgets(buf, sizeof(buf), fd);
		if( res==NULL )
			break;
		strcpy(rules[i],buf);
		i++;
	}
	fclose(fd);
}
```

La formattazione delle regole è legata all'output fornito dallo strumento di decodifica dei dati forniti dagli swtich. Essendo quest'output disposto su una riga di testo, con i vari campi intervallati da una virgola, le regole sono composte da una serie di asterischi, tanti quanti sono i campi dell'output, e solo

nelle posizioni corrispondenti alla porta e al protocollo vengono inseriti i valori da monitorare.

La funzione *splitRules* ha il compito di scomporre i campi e memorizzarli in una seconda matrice dove ogni singola riga corrisponderà a uno specifico campo.

```
void splitRules(char **riga,char **campo){

      char *p;
      int i=0,
                z=0;

      for(z=0;z<20;z++)
      {
            i=(20*z);
            p = strtok(riga[z],",");
            while (p != NULL)
            {
                  strcpy(campo[i],p);
                  i++;
                  p = strtok (NULL, ",");
            }
      }
}
```

4.4.2 Caricamento dati

Al primo thread avviato è associata la funzione *sFlow* con il compito di leggere e memorizzare nella matrice *mat*, condivisa all'interno del modulo stesso, le informazioni ricevute relative ai vari pacchetti campionati.

La funzione *popen* riceve come parametro il comando con il quale viene lanciato sflowtool che decifra i dati contenuti negli sFlow Datagram e restituisce lo standard output nella *pipe* collegata allo stream restituito come valore di ritorno. Più precisamente questa funzione crea una pipe, invoca il programma passato come parametro attraverso la shell e restituisce il puntatore allo stream associato alla pipe creata. La lettura dei dati avviene fino al riempimento della matrice, dopo il quale viene risvegliato il secondo thread che ha così la possibilità di ottenere il controllo in mutua esclusione della struttura dati condivisa e svolgere la sua mansione.

```
void *sflow(void *arg){

      FILE *in;
      extern FILE *popen();
      char buff[DIM_BUFFER];
      int i = 0,
          w = 0;

      …

      while(1)
      {
            if(fgets(buff,sizeof(buff),in)!=NULL)
            {
                  pthread_mutex_lock(&shared.MUTEX);

                  strcpy(mat[i],buff);

                  if (i<(MAX_RIGHE-1))
                        i++;
                  else
                  {
                        i=0;

                        pthread_cond_broadcast(&shared.STAT);
                        pthread_mutex_unlock(&shared.MUTEX);
                        pthread_mutex_lock(&shared.MUTEX_LOG);
                        phread_cond_wait
                        (&shared.LOG,&shared.MUTEX_LOG);
                        pthread_mutex_unlock(&shared.MUTEX_LOG);
                  }
                        pthread_mutex_unlock(&shared.MUTEX);

                        if(LOG == 1)
                              writeLog("sflow.log",buff);
                  }
            }
            pclose(in);
}
```

4.4.3 Controllo regole

Associata al secondo thread, la funzione *controllo*, per ogni riga della matrice esegue la scansione di una lista per verificare se è già presente un elemento che fa riferimento allo stesso flusso di dati a cui appartiene il pacchetto associato alla riga che si sta analizzando e, in caso di esito positivo, passa a quella successiva. Se l'elemento non viene trovato, ne viene inserito uno nuovo in coda alla lista e viene effettuato un controllo, che consiste in un confronto dei campi interessati con i corrispondenti di ogni regola caricata in

precedenza. Se i valori corrispondono le informazioni che si desiderano ricevere per e-mail vengono inviate come parametri alla funzione *send_mail*.

```
void *controllo()
{
      int i=0,y=0,z=0,w=0,x=0,q=0;
      int COND;
      struct elemento *List,*Cursore,*tmp1,*tmp3,*Coda;

      …

      while(1)
      {
            pthread_mutex_lock(&shared.MUTEX_STAT);
            pthread_cond_wait
            (&shared.STAT, &shared.MUTEX_STAT);
            pthread_mutex_unlock(&shared.MUTEX_STAT);
            pthread_mutex_lock(&shared.MUTEX);

            splitPacketSflow(mat,campoPacket);

            i = 0;
            z = 0;
            y = 0;

            List=
            (struct elemento*)malloc(sizeof(struct elemento));
            List->ipS=
            (char*)calloc(strlen(campoPacket[9]),sizeof(char));
            List->ipD=
            (char*)calloc(strlen(campoPacket[10]),sizeof(char));
            strcpy(List->ipS,campoPacket[9]);
            strcpy(List->ipD,campoPacket[10]);
            List->protocol = atoi(campoPacket[11]);
            List->port = atoi(campoPacket[14]);
            List->ptr = NULL;

            Coda = List;

            for(q = 0; q < N_RULES; q++)
            {
                  if(Coda>protocol==atoi(campoRules[11+(20*q)])
                     && Coda->port==atoi(campoRules[14+(20*q)]))
                  {
                        send_mail(Coda->ipS,Coda->ipD,
            campoPacket[11],campoPacket[14]);
      }
}


for(y=1;y<MAX_RIGHE;y++)
{
      Cursore = List;
      COND = 1;
```

```
        z=9+(20*y);
        i=11+(20*y);
        x=14+(20*y);

        while(Cursore != NULL)
        {
              if(strcmp(Cursore->ipS,campoPacket[z])==0)
              {
                    if(strcmp(Cursore->ipD,campoPacket[z+1])==0)
                    {
                       if(Cursore->port == atoi(campoPacket[x])
                       && Cursore->protocol==atoi(campoPacket[i]))
                       {
                          COND = 0;
                          break;
                        }
                    }
              }

              tmp3 = Cursore;
              Cursore = Cursore->ptr;
        }

        if(COND == 1)
        {
              tmp3->ptr =
              (struct elemento *)malloc(sizeof(struct elemento));
              Coda = tmp3->ptr;
              Coda->ipS =
              (char*)calloc(strlen(campoPacket[z]),sizeof(char));
              Coda->ipD =
              (char*)calloc(strlen(campoPacket[z+1]),sizeof(char))
              strcpy(Coda->ipS,campoPacket[z]);
              strcpy(Coda->ipD,campoPacket[z+1]);
              Coda->protocol = atoi(campoPacket[i]);
              Coda->port = atoi(campoPacket[x]);
              Coda->ptr = NULL;

              for(q = 0; q < N_RULES; q++)
              {
                    if(Coda->protocol==atoi(campoRules[11+(20*q)])
                       && Coda->port==atoi(campoRules[14+(20*q)]))
                    {
                          send_mail(Coda->ipS,Coda->ipD,
                          campoPacket[i],campoPacket[x]);
                    }
              }
        }
        pthread_mutex_unlock(&shared.MUTEX);
        pthread_cond_broadcast(&shared.LOG);
  }

  while(List != NULL)
  {
        tmp1 = List;
        List = List->ptr;
```

```
            free(tmp1);
      }

}
```

4.4.3.1 Gestione pacchetti campionati

Per effettuare il controllo attraverso un semplice confronto, devono essere suddivisi, così come avviene per le regole, anche i campi dei dati contenuti nella matrice *mat.* La funzione *splitPacketSflow* si occupa, infatti, di scindere i vari campi e assegnarli alla matrice *campoPacket*.

```
void splitPacketSflow(char **riga,char **campo){

      char *p;
      int i,z;
      for(z=0;z<MAX_RIGHE;z++)
      {
            i=(20*z);
            p = strtok(riga[z],",");
            while (p != NULL)
            {
                  strcpy(campo[i],p);
                  i++;
                  p = strtok (NULL, ",\0");
            }
      }
}
```

4.4.3.2 Invio mail

Questa funzione ha il compito di inviare una mail all'amministratore di rete e inoltre si occupa della formattazione del messaggio che si vuole spedire, nel quale inserisce le informazioni ricevute come parametri.

```
void send_mail(char *ips,char *ipd,char *protocol,char *port)
{
      char comando[4096];
      char comando1[]="echo '";
      char comando2[]=
      "' | mail -a'From:Detector' -s'Violazione regola' amministratore@dominio.it";

      strcpy(comando,comando1);
      strcat(comando,ips);
      strcat(comando,"\t");
      strcat(comando,ipd);
      strcat(comando,"\t");
      strcat(comando,protocol);
      strcat(comando,"\t");
      strcat(comando,port);
```

```
	strcat(comando,comando2);

	system(comando);
}
```

4.5 Modulo per il calcolo delle statistiche

Sulla base delle informazioni strettamente legate al flusso, vengono calcolate le statistiche su un intervallo di pacchetti campionati definito dall'amministratore attraverso le dimensioni della matrice *mat*. Il *main* si occupa di lanciare l'esecuzione di due thread le cui rispettive funzioni svolgono la fase di caricamento dei dati e calcolo delle statistiche.

```
int main(int argc,char* argv[])
{
	…

	pthread_mutex_init(&shared.MUTEX, NULL);
	pthread_mutex_init(&shared.MUTEX_STAT, NULL);
	pthread_mutex_init(&shared.MUTEX_LOG, NULL);
	pthread_cond_init(&shared.STAT, 0);
	pthread_cond_init(&shared.LOG, 0);

	pthread_t scrittura;
	pthread_t lettura;

	err= pthread_create(&scrittura,NULL,sflow,(void*)argv[1]);

	…

	err = pthread_create(&lettura,NULL,controllo,NULL);

	pthread_join(scrittura, NULL);
	pthread_join(lettura, NULL);

	return 0;
}
```

L'operazione di memorizzazione dei dati nella matrice e in seguito la suddivisione dei singoli campi, avviene in maniera identica a come descritto precedentemente per il modulo *ManagerSflowRule*.

4.5.1 Calcolo delle statistiche

Per ogni riga della matrice contenente i dati sui pacchetti campionati, viene effettuata la scansione di una lista che contiene un elemento per ogni

indirizzo IP sorgente. Nel caso in cui durante la scansione viene riscontrata la presenza di un elemento con un indirizzo IP sorgente uguale a quello in esame, si procede confrontando IP destinazione e successivamente porta e protocollo. Se tutti i controlli hanno un esito positivo, significa che il pacchetto in questione fa parte di un flusso dati di cui è già giunto un altro campione in precedenza. Per questo motivo viene incrementata la variabile *occ*, che tiene traccia del numero di ricorrenze di pacchetti campioni che fanno riferimento a uno stesso traffico di dati, e impostata a zero la variabile *COND* per evitare di aggiungere in coda alla lista un nuovo elemento con un indirizzo IP sorgente già presente. Anche nel caso in cui le verifiche sui vari campi non vadano a buon fine, ma esiste già un elemento con uno specifico indirizzo IP sorgente, non viene inserito un nuovo elemento nella lista. In quest'ultimo caso però viene assegnato alla variabile *ctrl* il valore uno, che indica il coinvolgimento del corrispondente indirizzo IP sorgente in diverse sessioni, intese come differente destinazione, porta o protocollo, e di conseguenza l'esclusione dall'elenco delle sorgenti generatrici di traffico giudicabile innocuo.

L'invio dell'IP da non sottoporre al controllo dell'IDS al modulo che si occupa della gestione del firewall, avviene proprio sulla base del valore della variabile *ctrl*. Prima di svuotare la lista viene controllato per ogni singolo elemento il valore di tale variabile e nel caso in cui corrisponde a zero, l'indirizzo IP associato viene inviato al modulo *ManagerFirewall* attraverso la funzione *sendToFirewall*.

```
void *controllo()
{
      int i = 0,
          y = 0,
          z = 0,
          x = 0,
          w = 0,
          COND = 0;

      struct elemento *List,*Cursore,*tmp1,*tmp3,*Coda;

      …

while(1){
      pthread_mutex_lock(&shared.MUTEX_STAT);
      pthread_cond_wait(&shared.STAT, &shared.MUTEX_STAT);
      pthread_mutex_unlock(&shared.MUTEX_STAT);
```

```
pthread_mutex_lock(&shared.MUTEX);

splitPacketSflow(mat,campoPacket);

i = 0;
x = 0;
y = 0;
z = 0;

List =
(struct elemento *)malloc(sizeof(struct elemento));
List->ipS=
(char*)calloc(strlen(campoPacket[9]),sizeof(char));
List->ipD=
(char*)calloc(strlen(campoPacket[10]),sizeof(char));
strcpy(List->ipS,campoPacket[9]);
strcpy(List->ipD,campoPacket[10]);
List->occ = 0;
List->protocol = atoi(campoPacket[11]);
List->port = atoi(campoPacket[14]);
List->ctrl = 0;
List->ptr = NULL;

Coda = List;
for(y=1;y<MAX_RIGHE;y++){

      Cursore = List;
      COND = 1;
      z=9+(20*y);
      i=11+(20*y);
      x=14+(20*y);

      while(Cursore != NULL){
            if(strcmp(Cursore->ipS,campoPacket[z])==0)
            {
              if(strcmp(Cursore->ipD,campoPacket[z+1])==0)
              {
               if(Cursore->port == atoi(campoPacket[x]) &&
                  Cursore->protocol==atoi(campoPacket[i]))
               {
                   COND = 0;
                   Cursore->occ++;
                   break;
               }
               else
               {
                   COND = 0;
                   Cursore->ctrl = 1;
                   break;
               }
              }

            else
            {
                  COND = 0;
                  Cursore->ctrl = 1;
```

```
                        break;
                    }
                }
                tmp3 = Cursore;
                Cursore = Cursore->ptr;
            }

            if(COND == 1)
            {
                tmp3->ptr=
                (struct elemento *)malloc(sizeof(struct elemento));
                Coda = tmp3->ptr;
                Coda->ipS=
                (char*)calloc(strlen(campoPacket[z]),sizeof(char));
                Coda->ipD=
                (char*)calloc(strlen(campoPacket[z+1]),sizeof(char))
                strcpy(Coda->ipS,campoPacket[z]);
                strcpy(Coda->ipD,campoPacket[z+1]);
                Coda->occ = 0;
                Coda->protocol = atoi(campoPacket[i]);
                Coda->port = atoi(campoPacket[x]);
                Coda->ctrl = 0;
                Coda->ptr = NULL;
            }
        }
        sendToFirewall("empty");

        while(List != NULL){
            tmp1 = List;
            if(List->ctrl == 0 && List->occ > MAX_OCC){
                sendToFirewall(List->ipS);
            }
                List = List->ptr;
                free(tmp1);
            }
            pthread_mutex_unlock(&shared.MUTEX);
            pthread_cond_broadcast(&shared.LOG);
        }
}
```

4.5.1.1 Informazioni per il filtraggio del traffico

Nel caso in cui viene ritenuto opportuno, esclude dall'analisi il traffico generato da un particolare indirizzo IP, questo viene inviato al modulo che si occuperà di inserirlo nella tabella di firewall per il filtraggio del traffico attraverso la funzione *sendToFirewall* la quale crea una connessione con il server in ascolto e spedisce un messaggio contenente l'indirizzo IP.

```
void sendToFirewall(char *ip){

    int DescrittoreSocket;
```

```
	DescrittoreSocket =
	CreaSocket(IP_SERVER_FIREWALL,PORT_MANAGER_TO_FIREWALL);
	SpedisciMessaggio(DescrittoreSocket,ip);
}
```

4.6 Modulo per la gestione del firewall

ManagerFirewall implementa un server che rimane in ascolto e attende la ricezione di un messaggio in base al quale deciderà quale azione eseguire. Per essere più precisi, ricevendo la stringa “empty” chiamerà la funzione *unblock_all*, mentre una sequenza corrispondente a un indirizzo IP verrà inviata come parametro alla funzione *block*.

```
int main(int argc,const char *argv[]) {

	char buffer[200];
	int DescrittoreSocket,NuovoSocket;
	int exitCond = 0;
	int Quanti = 0;
	int i = 0,
	    z = 0;

	DescrittoreSocket=CreaSocket(1745);

	while (!exitCond)
	{
		if ((NuovoSocket=accept(DescrittoreSocket,0,0))!=-1)
		{
			…

			buffer[Quanti]='\0';
			if(strcmp(buffer,"empty")==0)
			{
				unblock_all();
			}
			else
			{
				block(buffer);
			}

			    ChiudiSocket(NuovoSocket);
			 }
		}

	ChiudiSocket(DescrittoreSocket);

	return 0;

}
```

4.6.1 Comandi ebtables

Dato che il modulo è installato sullo stesso server su cui è presente l'IDS, i comandi per modificare le regole di firewall vengono eseguiti a riga di comando grazia alla funzione *system*, che li riceve come parametri.

4.6.1.1 Eliminazione di tutte le regole

Nel modulo *ManagerSflowStat*, calcolate le statistiche, prima di controllare quali indirizzi IP devono essere bloccati, viene inviata al modulo *ManagerFirewall* la stringa "empty" , che riconduce alla funzione *unblock_all* la quale ha il compito di svuotare interamente la tabella del firewall. Oltre al fatto di evitare l'inserimento di una stessa regola più volte, questo criterio è utile soprattutto per garantire che un determinato IP venga riabilitato al controllo e non rimanga perennemente bloccato.

```
void unblock_all(){

      int ctrl;
      …
      strcpy(comando,"ebtables -F");
      ctrl = system(comando);

}
```

4.6.1.2 Inserimento nuova regola

La ricezione di un indirizzo IP comporta l'inserimento di una nuova regola per il filtraggio del traffico. La funzione *block* si occupa di formattare in modo corretto il comando, inserendo l'indirizzo IP ricevuto.

```
void block(char *ip){

      char comando[50];
      int ctrl;

      …

      strcpy(comando,"ebtables -A INPUT -i eth1.220 -p 0x800 --ip-src ");
      strcat(comando,ip);
      strcat(comando," -j DROP");

      ctrl = system(comando);

      …

}
```

4.7 Socket

Come canale di comunicazione fra i due moduli, *ManagerSflowStat* e *ManagerFirewall*, è stata creata una socket che permette appunto la comunicazione tra i due processi mediante il protocollo TCP.

Affinché vi possa essere uno scambio di dati è necessario come prima cosa creare questo canale di comunicazione. Questo compito è svolto dalle due funzioni *CreaSocketServer* e *CreaSocketClient*, che vengono chiamate rispettivamente dal gestore del firewall, che funge da server in ascolto, e dal processo che si occupa del calcolo delle statistiche, che invia i dati. Questa fase di trasmissione delle informazioni avviene grazie alla funzione *SpedisciMessaggio*, che conoscendo il descrittore della socket interessata non fa altro che effettuare una scrittura su questa.

```
int CreaSocketServer(int Porta)
{
      int sock,errore;
      struct sockaddr_in temp;

      sock=socket(AF_INET,SOCK_STREAM,0);

      temp.sin_family=AF_INET;
      temp.sin_addr.s_addr=INADDR_ANY;
      temp.sin_port=htons(Porta);

      errore=fcntl(sock,F_SETFL,O_NONBLOCK);
      errore=bind(sock,(struct sockaddr*) &temp,sizeof(temp));
      errore=listen(sock,100);

      return sock;
}

int CreaSocketClient(char* Destinazione, int Porta)
{
        struct sockaddr_in temp;
        struct hostent *h;
        int sock;
        int errore;

        temp.sin_family=AF_INET;
        temp.sin_port=htons(Porta);
        h=gethostbyname(Destinazione);
        if (h==0)
        {
          printf("Gethostbyname fallito\n");
          exit(1);
        }
        bcopy(h->h_addr,&temp.sin_addr,h->h_length);
```

```
        sock=socket(AF_INET,SOCK_STREAM,0);

        errore=connect(sock, (struct sockaddr*) &temp, sizeof(temp));
        return sock;
}


void SpedisciMessaggio(int sock, char* Messaggio)
{
      if (write(sock,Messaggio,(strlen(Messaggio)))<0)
      {
          printf("Impossibile mandare il messaggio.\n");
          ChiudiSocket(sock);
          exit(1);
      }

      return;
}


void ChiudiSocket(int sock)
{
      close(sock);
      return;
}
```

4.8 File di LOG

Per tutte le operazioni implementate nei vari moduli, è stata prevista la possibilità di creare un file di log per tenere traccia dei vari input ricevuti dalle funzioni e le modifiche effettuate.

```
void writeLog(char *name, char *buff){

      FILE *f;
      time_t ora,
             t;

      f=fopen(name,"a");
      t = time(&ora);
      fputs(asctime(localtime(&t)),f);
      fputs(buff,f);
      fclose(f);

}
```

Sulla base del valore assegnato alla variabile LOG, passato come parametro all'avvio del rispettivo servizio, le informazioni vengono o no salvate sul file.

5 Valutazione delle prestazioni

In questo capitolo verranno forniti i risultati ottenuti analizzando solo il traffico wireless. Il traffico wired è stato omesso semplicemente per ridurre i dati con cui lavorare, in modo da poter gestire con più facilità i risultati ed esporli con maggiore chiarezza. Ad accreditare questa scelta c'è che la maggior parte delle minacce deriva proprio dalla rete wireless alla quale gli utenti possono accedere liberamente. Nella rete wired, invece, si può aver accesso solo attraverso le macchine di laboratorio, le quali presentano dei dispositivi di sicurezza funzionanti correttamente, come antivirus, e sono impostate delle regole di firewall molto più restrittive.

La macchina su cui è installato il servizio è composta da 4 AMD Opteron(tm) Processor 850, 8 GigaByte di RAM e 250 GigaByte di Hard Disk.

5.1 Strumenti per il calcolo e formattazione risultati

Come strumento per il calcolo delle prestazioni è stato utilizzato il comando *EBTABLES* che permette di visualizzare l'intera lista delle regole di firewall applicate, con accanto a ogni singola direttiva il numero di pacchetti interessati, che nel nostro caso equivale al numero di pacchetti eliminati dal controllo del detector.

IP	Azione	Pacchetti	Byte
74.120.9.64	DROP	24393	32802
209.222.18.54	DROP	30721	1231180
174.140.129.63	DROP	15176	819048
137.204.0.0/16	ACCEPT	75231	13959675

Figura 5.1 Esempio dei dati ottenuti dal comando "ebtables –L --Lc"

La tabella in *figura 5.1* riporta i risultati relativi all'analisi su un intervallo di 600 campioni (*MAX_RIGHE*) estratti dagli sFlow Datagram, dove il traffico escluso è associato a degli indirizzi IP che presentano una singola sessione attiva e i rispettivi campioni ricorrono nell'intervallo un numero di volte superiore a 5 (*MAX_OCC*). Equivale a dire che durante il periodo in cui sono stati caricati e processati nuovi dati in memoria per ottenere le rispettive statistiche, il traffico analizzato è stato pari a circa 13 megabyte rispetto a quello reale di circa 16 Megabyte, ottenendo così una riduzione di circa il 19% dei dati destinati alla scansione dell'IDS (*figura 5.2*).

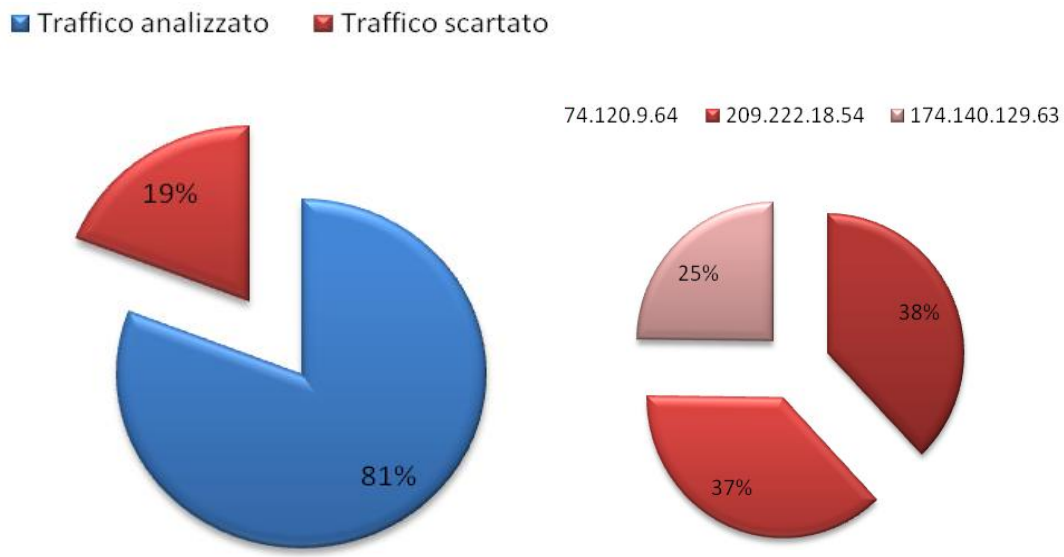

Figura 5.2 Rappresentazione dei dati presenti nella tabella in figura 5.1

Analizzando i file di log relativi al traffico escluso, emerge che si tratta in tutti e tre i casi di un flusso di dati che utilizza il protocollo TCP sulla porta 80.

5.2 Stima delle prestazioni

Stabilito con quale criterio avviene la valutazione delle performance, verrà riportata di seguito una media dei risultati ottenuti dalle verifiche effettuate sulla base dei valori assegnati ai parametri di configurazione e, inoltre, verranno fornite una seria d'informazioni statistiche relative al flusso di dati scartato. Più precisamente verranno motivate le scelte relative ai valori assegnati ai parametri e verrà commentato il comportamento dell'applicazione nei vari casi.

È importante tenere presente che i pacchetti, associati al protocollo http, verranno considerati come i più sicuri da poter escludere, questo non perché tale

protocollo è esente da rischi ma semplicemente perché due indirizzi IP impegnati in una singola sessione che utilizza il protocollo TCP sulla porta 80, nella maggior parte dei casi non rappresentano una fonte di pericolo.

5.2.1 Elevata esclusione di traffico

La *figura 5.3* rende evidente come impostando un intervallo piccolo (*MAX_RIGHE = 200*) di pacchetti campionati su cui calcolare le statistiche e una bassa ricorrenza (*MAX_OCC >= 2*) di pacchetti campionati appartenenti a uno stesso flusso di dati, permette di escludere in media il 46% di traffico dall'analisi dell'IDS, raggiungendo in alcuni casi picchi del 68% e non scendendo mai sotto il 20% di dati omessi dalla scansione del detector.

Sempre dalla *figura 5.3* si può notare come vengano aggiornate frequentemente le regole di firewall e di conseguenza come siano brevi gli intervalli di tempo in cui vengano rispettivamente applicate.

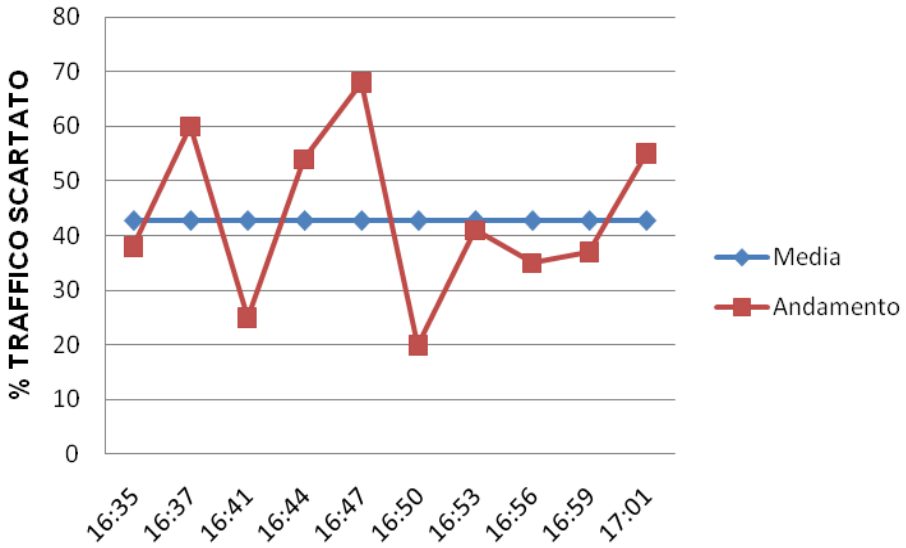

Figura 5.3 Percentuale di traffico esclusa in corrispondenza dei vari intervalli di dati

È importante notare (*figura 5.4*) come, conseguentemente alla non elevata soglia di ricorrenze, nel traffico scartato la percentuale maggiore, utilizza protocolli di livello applicativo non riconducibili a nessuno di quelli già noti. In particolare al 41% che impiega il protocollo UDP, si aggiunge un 52% di traffico associato al TCP, ma di cui non si conosce il protocollo a livello

applicativo. Questo equivale a ridurre il livello di sicurezza proprio perché si rischia di scartare del traffico di cui non si conoscono informazioni tali da poter garantire con buona probabilità che non costituisce un pericolo.

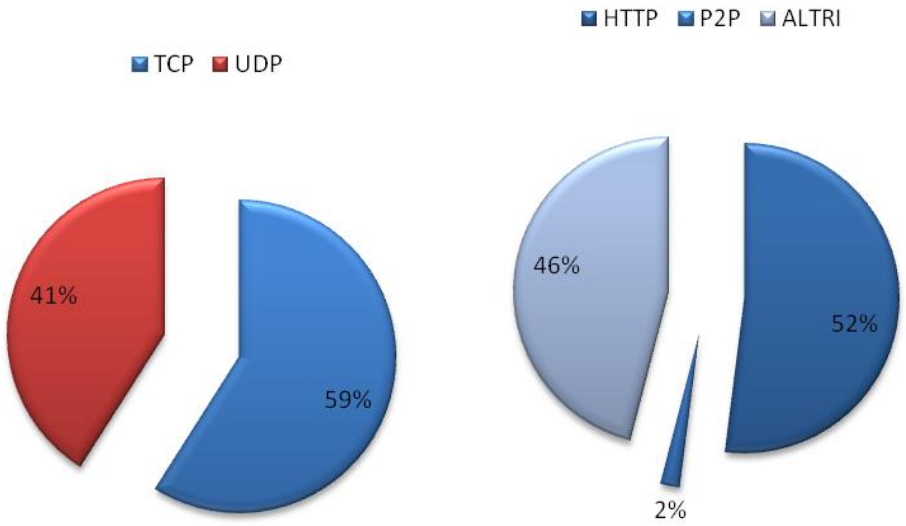

Figura 5.4 Percentuale dei protocolli utilizzati dal traffico escluso

Data l'elevata percentuale di scarto dei dati, c'è la probabilità che nei protocolli associati al traffico escluso ne rientrino alcuni che semplicemente per le loro caratteristiche non dovrebbero essere selezionati. È il caso del P2P che è basato sulla creazione di numerose sessioni e, quindi, non si presta a quelli che sono i criteri d'identificazione dell'applicazione.

5.2.2 Garanzia di una maggiore sicurezza

Nella *figura 5.5* si può notare come, aumentando il numero di pacchetti campionati (*MAX_RIGHE = 1200*) su cui effettuare i controlli e incrementando il numero minimo (*MAX_OCC >= 7*) di ricorrenze di campioni corrispondenti a uno stesso flusso, la percentuale di traffico scartato, si abbassa vertiginosamente, mediamente il 5,8%, e aumenta il tempo trascorso per l'inserimento di nuove regole per il filtraggio, il triplo di quello impiegato nella verifica in precedenza descritta.

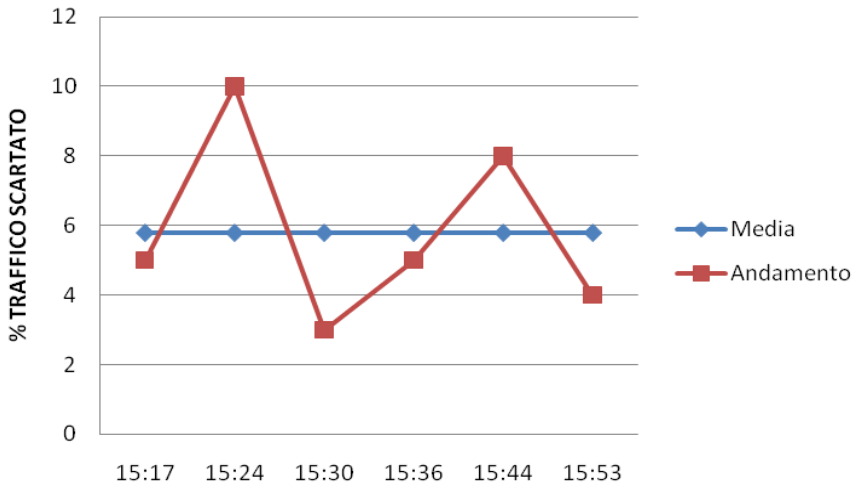

Figura 5.5 Percentuale di traffico esclusa in corrispondenza dei vari intervalli di dati

Anche questa volta è importante riportare le informazioni riguardanti il traffico escluso.

In questo caso, a fronte di una riduzione della percentuale corrispondente al traffico omesso, c'è una maggiore accuratezza per quanto riguarda il tipo di protocollo applicativo che viene scartato. La *figura 5.6* evidenzia infatti come la maggior parte dei pacchetti esclusi dalla scansione, utilizza lo http.

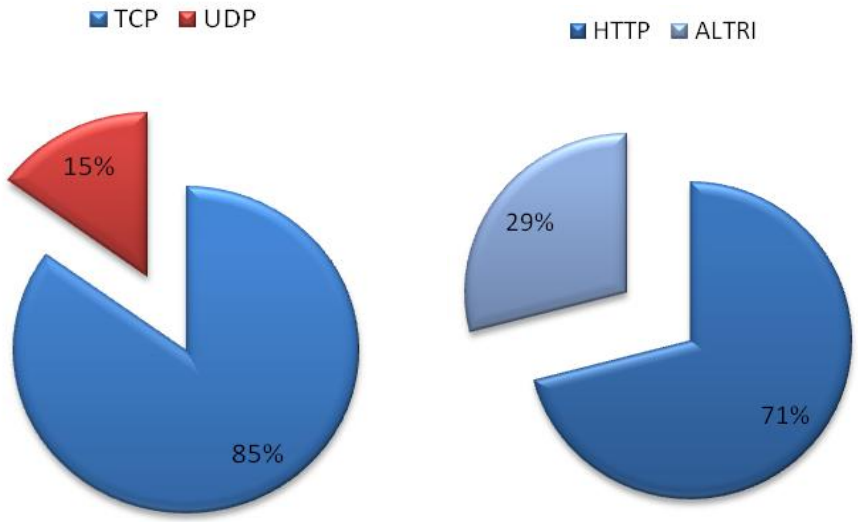

Figura 5.6 Percentuale dei protocolli utilizzati dal traffico escluso

6 Conclusioni e sviluppi futuri

6.1 Conclusioni

In questa tesi, dopo aver illustrato la necessità di monitorare il traffico di una rete e aver rilevato l'aspetto legato alla sicurezza, intesa come gestione degli incidenti informatici, è stata illustrata la progettazione e l'implementazione di un sistema dinamico per la gestione del *packet filtering*. Questo progetto è legato alla necessità di risolvere un problema riscontrato nella fase d'identificazione del traffico sulla rete del Polo Scientifico Didattico di Cesena e che impediva il corretto funzionamento del dispositivo con il compito di analizzare il flusso di dati, in altre parole l'IDS.

La soluzione adottata prevede la riduzione della mole di dati da sottoporre alla scansione del detector, tramite un'analisi preventiva basata sulle informazioni fornite dagli apparati di rete attraverso la tecnologia sFlow e che permette di effettuare una prima classificazione del traffico.

Proprio questa tecnica si è rivelata la chiave al nostro problema, infatti, essendo quest'ultimo dovuto all'elevato numero d'informazioni che l'IDS doveva analizzare, senza l'utilizzo di questa tecnologia, che prevede il campionamento del traffico di una rete ad alta velocità, il problema sarebbe persistito anche nella fase di analisi preventiva. Il fatto di riuscire a ottenere dei dati campionati relativi al flusso dei dati, ha reso possibile l'eliminazione del traffico giudicabile con alta probabilità non pericoloso, prima che venga sottoposto alla scansione vera e propria.

Si può ritenere raggiunto anche uno dei primi obiettivi fissati in fase di progettazione, in altre parole la dinamicità dell'intero sistema. Una volta avviato il servizio, infatti, tutte le varie fasi, partendo dal caricamento delle informazioni,

passando dall'elaborazione dei dati, terminando con le azioni che ne scaturiscono, sono coordinate attraverso dei thread in mutua esclusione e non prevedono in nessun caso l'intervento dell'amministratore.

Inoltre la presenza di un file di configurazione che permette di modificare dei parametri a livello di codice, rende il sistema adattabile alle proprie esigenze. Infatti, come emerso durante il calcolo delle prestazioni, riducendo il numero di campioni da analizzare e abbassando il numero minimo di occorrenze di pacchetti campionati appartenenti a un determinato flusso di dati, si può arrivare a casi limite in cui viene escluso fino al 70% del traffico. Indubbiamente qualsiasi configurazione venga adottata, il livello di sicurezza tende ad abbassarsi, poiché non viene monitorato l'intero traffico. Tenendo presente però che l'applicazione è stata creata proprio per risolvere il problema di un'analisi non corretta, tutto questo potrebbe rappresentare comunque un compresso accettabile di fronte a una scelta fra il dover affrontare una spesa ingente per attrezzature in grado di gestire reti ad alta velocità e il dover rinunciare totalmente alla scansione identificativa.

Anche se l'esito generale può risultare soddisfacente, ciò non toglie che ci siano alcuni aspetti, elencati nel prossimo paragrafo, da poter migliorare.

6.2 Sviluppi futuri

L'applicativo sviluppo al momento può essere utilizzato esclusivamente su delle reti, dove sono presenti degli apparati che implementano la tecnologia sFlow, limitazione dovuta al fatto che gli switch utilizzati per la rete del Polo di Cesena presentano questa tecnologia. Sarebbe utile rendere il sistema applicabile anche dove sono presenti dispositivi di rete che utilizzano tecnologie simili, come NetFlow o RMON II.

Un'altra modifica possibile, che non garantirebbe a priori un miglioramento delle prestazioni, ma che sarebbe interessante valutare, riguarda la fase di calcolo delle statistiche. Attualmente le informazioni vengono processate non appena vengono ricevuti un numero di campioni ben definito, che corrispondono alle dimensioni della matrice destinata a contenere i rispettivi dati. Un'alternativa potrebbe essere quella di alternare la fase di caricamento e quella di calcolo sulla base del tempo trascorso, indipendentemente da quanti pacchetti campionati vengono recapitati all'applicazione.

Bibliografia

[WIK01] Free documentation, *Firewall*, http://it.wikipedia.org/wiki/Firewall, 2010

[JNT07] JNT Association, *IEE 802.1x*, http://www.ja.net/documents/publications/factsheets/064-ieee.802.1x.pdf, 2007

[BJH04] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, Ed., RFC3748, *Extensible Authentication Protocol (EAP)*, 2004.

[RWS00] C. Rigney, S. Willens, A. Rubens, W. Simpson, RFC2865, *Remote Authentication Dial In User Service (RADIUS)*, 2000

[WIK02] Free documentation, *VLAN*, http://it.wikipedia.org/wiki/VLAN, 2010

[PAP02] P. Phaal and S. Panchen, *Packet Sampling Basics*, http://www.sflow.org/packetSamplingBasics/index.htm, 2002

[LUP10] M. Lupini, *Individuazione e prevenzione di incidenti informatici attraverso l'uso di ossim*, 2010.

[LAP04] M. Lavine and P. Phaal, *sFlow Version 5*, July 2004 http://www.sflow.org/sflow_version_5.txt

[SFL03] sFlow®, *Traffic Monitoring using sFlow*, http://www.sflow.org/sFlowOverview.pdf, *2003*

[LIU09] Guoming Liu, *Management, Optimization and Evolution of the LHCb Online Network*, http://cdsweb.cern.ch/record/1254304/files/CERN-THESIS-2010-040.pdf, 2009

[WAL97] S. Waldbusser, *Remote Network Monitoring Management Information Base Version 2*, http://tools.ietf.org/html/rfc2021, 1997

[CLA04] B.Claise, *Cisco System NetFlow Services Export Version 9*, http://www.ietf.org/rfc/rfc3954.txt, 2004

[JPB92] Jonathan Jedwab, Peter Phaal and Bob Pinna, *Traffico estimation for the largest sources on a network, using packet sampling with limited storage*, http://www.hpl.hp.com/techreports/92/HPL-92-35.pdf

[ALL06] M. Allen, *ARP and RARP Address Translation*, http://www.comptechdoc.org/independent/networking/guide/netarp.html, 0.6.3

[DSF03] B. De Schuymer and Nick Fedchick, *ebtables/iptables interaction on a Linuc-based bridge*, http://ebtables.sourceforge.net/br_fw_ia/br_fw_ia.html, 2003