UNIVERSITA' DEGLI STUDI DI GENOVA

FACOLTA' DI INGEGENRIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

TESI DI LAUREA

STRATEGIE DI OTTIMIZZAZIONE ENERGETICA TRAMITE INTRUSION PREVENTION SYSTEM

RELATORE:

Chiar.mo Prof. Mauro Migliardi

CORRELATORE:

Dott. Alessio Merlo

CANDIDATI:

Diego Raso

Elena Spadacini

Anno Accademico 2012-2013

Ringraziamenti

Indice

Capitolo 1

Introduzione

Figura 1.1: Ecosostenibilità

In questa tesi affronteremo due tematiche riguardanti le reti estremamente importanti e attuali: la sicurezza informatica e l'ecosostenibilità.

Fin dall'origine, il problema della sicurezza informatica è stato al centro di discussioni e ricerche. Negli anni si è notato un cambiamento nelle motivazioni degli attacchi informatici: se i primi virus erano mirati essenzialmente a distruggere i dati della vittima (per esempio formattarne l'hard disk), ora, con l'evoluzione dei worm, alla costruzione di botnet utilizzate per spam e attacchi di Distribuited Denial of Service indirizzati principalmente al mondo industriale. Le botnet, data la loro enorme e crescente diffusione, meritano un ulteriore approfondimento: se un computer è infettato da una bot, gli hackers sono in grado di controllarlo in remoto tramite Internet. Da quel momento in poi, il computer diventa inconsapevolmente uno "zombie" in balia degli attaccanti e si aggiunge agli altri infetti della botnet. Generalmente gli hackers possono venderla o utilizzarla per fini malevoli, nella maggior parte dei casi per lanciare un DDoS, comandando a migliaia di computer di accedere simultaneamente allo stesso sito Web, in modo tale che il server non sia in grado di gestire tutte le richieste che riceve. Ricapitolando, gli esperti del settore affermano che si è passati da motivazioni "distruttive" a "commerciali". Di conseguenza il numero di attacchi e i derivanti danni economici della vittima sono cresciuti notevolmente negli ultimi anni. Dati alla mano, [1], nel 2012 in Inghilterra ben l'87% delle piccole aziende (con aumento del 11% rispetto al 2011) e il 93% delle grandi ha avuto una falla nel sistema di sicurezza con danno, considerando l'incidente più grave,

mediamente compreso tra 35000 e 65000 sterline e tra 450000 e 850000 sterline rispettivamente.

	ISBS 2013 small businesses	ISBS 2013 large organisations
Business disruption	£30,000 - £50,000 *over 3-5 days*	£300,000 - £600,000 *over 3-6 days*
Time spent responding to incident	£2,000 - £5,000 *6-12 man-days*	£6,000 - £13,000 *25-45 man-days*
Lost business	£300 - £600	£10,000 - £15,000
Direct cash spent responding to incident	£500 - £1,500	£35,000 - £60,000
Regulatory fines and compensation payments	£0	£750 - £1,500
Lost assets (including lost intellectual property)	£150 - £300	£30,000 - £40,000
Damage to reputation	£1,500 - £8,000	£25,000 - £115,000
Total cost of worst incident on average	£35,000 - £65,000	£450,000 - £850,000
2012 comparative	*£15,000 - £30,000*	*£110,000 - £250,000*
2010 comparative	*£27,500 - £55,000*	*£280,000 - £690,000*

Figura 1.2: Media costo incidente informatico grave

Le compagnie, per limitare l'efficacia degli attacchi, utilizzano generalmente la seguente architettura di rete, considerata ormai uno standard: un firewall sulla frontiera della rete aziendale che decide, in base alle regole configurate dall'amministratore, quale traffico fare entrare e quale no e uno o più Intrusion Detection System (IDS) all'interno, per rilevare eventuali attacchi che, nonostante il firewall, siano riusciti ad entrare o, peggio, l'attività malevola di macchine all'interno della rete (host infettati o altro). Attualmente la maggior parte degli IDS opera in modalità in-line, realizzando un Intrusion Prevention System (IPS)[1], ossia un sistema non passivo ma in grado di rilevare, in tempo reale, gli attacchi che sfuggano alle maglie del firewall. Per poter analizzare tutto il traffico, l'IPS è posizionato in modo trasparente sul collegamento principale della rete.

L'altra importante tematica è l'ecosostenibilità, in particolare la green IT. Esistono diverse definizioni di green IT, o green computing, forse la più esplicativa è: "studio e messa in pratica di tecniche di progettazione più efficienti per la realizzazione di dispositivi IT seguendo il principio di una società ed economia più eco-sostenibile". In questa macro area, il nostro lavoro

[1] In passato vi era distinzione tra IDS, con comportamento passivo ossia rilevamento dell'attacco solo dopo che è avvenuto, e IPS, con un comportamento attivo ossia in grado di rilevarlo in real-time intraprendendo le giuste contromisure. Oggi giorno vi è un unico sistema capace di operare nelle due modalità ma è rimasta la distinzione in letteratura.

si inserisce nel cosiddetto "*Future Internet*", in quanto proponiamo un metodo che permetterà indirettamente di migliorare l'efficienza energetica. Lavorare per diminuire i consumi della rete è oggi giorno una priorità, sia per motivi ambientali, per ridurre gli sprechi e le emissioni di CO_2, sia economici, per limitare i costi dell'elettricità mantenendo un'adeguata qualità di servizio. Facendo riferimento ai dati 2010 di Telecom Italia [2], la cui tendenza è in linea con le altre grandi compagnie di telecomunicazioni, il gruppo ha consumato oltre 2 TWh di energia elettrica, che equivalgono a circa lo 0,67 % del consumo nazionale[2]. Confrontando i consumi del 2012 per Telecom Italia S.p.A con gli anni passati si può notare come questa voce è aumentata nel tempo, 1.06% rispetto al 2011 e 7.30% rispetto al 2010.

			% di incidenza delle Business Unit sui valori di Gruppo				
		Gruppo 2012	Domestic	Brasile	Argentina	Media	Olivetti
TOTALE ENERGIA ELETTRICA	kWh	2.753.536.413	71,04%	13,25%	14,15%	0,94%	0,62%

Figura 1.3: Consumi energetici gruppo Telecom Italia

ENERGIA ELETTRICA ACQUISTATA E PRODOTTA

			Variazione %	
		Telecom Italia S.p.A. 2012	2012 su 2011	2012 su 2010
ENERGIA ELETTRICA DA FONTI MISTE(*)	kWh	1.876.520.483	(1,06)%	(7,27)%
ENERGIA ELETTRICA DA FONTI RINNOVABILI	kWh	36.712.758	(0,86)%	(8,70)%
TOTALE ENERGIA ELETTRICA	kWh	**1.913.233.241**	**(1,06)%**	**(7,30)%**

(*) L'energia elettrica acquistata da fonti miste equivale a circa 1.793 GWh. L'energia elettrica autoprodotta da fonti miste equivale a circa 83 GWh ed è riferibile agli impianti di cogenerazione, con consumi associati pari a circa 21 milioni di m³ di gas metano. La produzione di energia elettrica da generatori funzionanti in continuità (non riportata in tabella) è stimata in circa 3 GWh.

Figura 1.4: Dettaglio consumi energetici Telecom Italia S.p.A

E' doveroso ricordare che, nonostante l'incremento di energia consumata, la compagnia ha investito in diverse iniziative green IT migliorando negli anni il suo eco-efficiency, ossia il rapporto tra i bit trasmessi sul canale e i consumi energetici (che tengono conto del contributo industriale, civile e autotrazione) [3].

[2] Il consumo nazionale per il 2010 è stimato in 305,5 TWh in base alle elaborazioni dell'Autorità per l'energia elettrica e il gas su dati GRTN/TERNA [1].

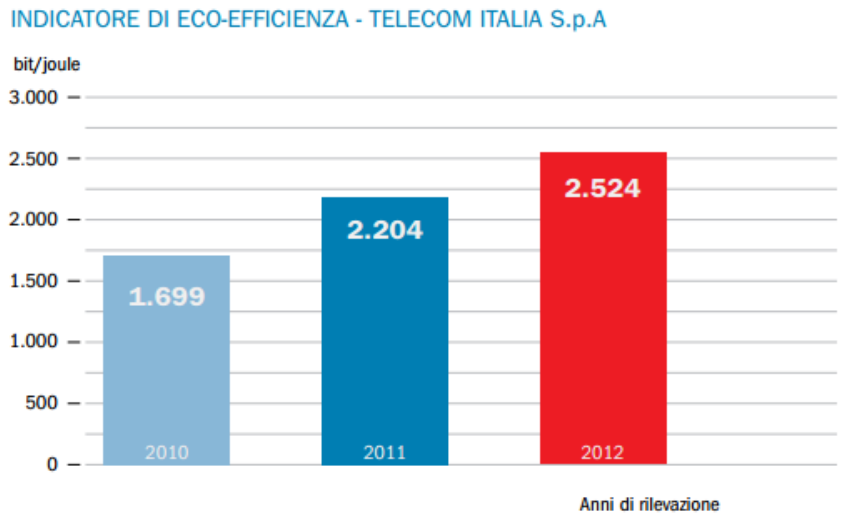

Figura 1.5: Indicatore di eco-efficienza Telecom Italia S.p.A

Se consideriamo il contesto europeo, sommando i consumi di tutti gli operatori nazionali, la situazione non cambia, infatti nel 2010 sono stati consumati ben 21 TWh e si stima per il 2020, senza l'utilizzo di tecnologie green, un valore di 36 TWh [4]. Per questo motivo l'Unione Europea ha investito molti fondi sul tema, per esempio nel progetto *ECONET* (low Energy COnsumption NETworks) [5]. Lo scopo del progetto è lo studio e l'utilizzo di tecnologie adattative per i dispositivi della rete cablata che permettano di risparmiare energia quando questi non vengono utilizzati, con obiettivo di diminuire i consumi del 50% sul breve-medio termine e l'80% sul lungo periodo.

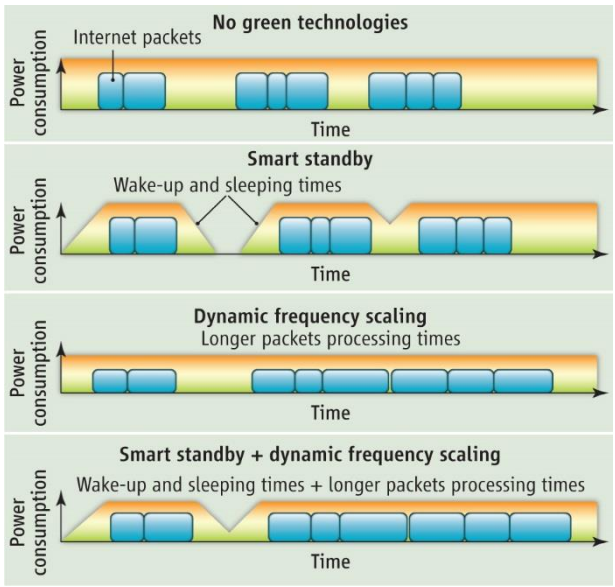

Figura 1.6: Tecniche router di nuova generazione

Brevemente, per raggiungere gli obiettivi prefissati, sono state ideate due tecniche generalmente combinate insieme: adattamento dinamico della potenza in base al carico della rete e ai vincoli Quality of Service con un opportuno trade-off tra profilo energetico e potenza di calcolo; smart standby per gestire efficientemente il passaggio da modalità active a sleep delle varie parti che compongono il router e viceversa.

In questo contesto si posiziona perfettamente il nostro studio orientato alla realizzazione di strategie di ottimizzazione energetica tramite Intrusion Prevention System, ossia un nuovo ed efficiente modello di analisi distribuita per anticipare la scoperta delle attività malevole già all'interno della rete e evitare consumi in trasmissione. Infatti, attualmente, un pacchetto inviato dalla sorgente alla destinazione è controllato da sistemi di protezione, come per esempio i già citati IPS, solo dopo aver terminato interamente il percorso. In altri termini un pacchetto facente parte di un attacco viene trasmesso nella rete di trasporto per essere, solo alla fine, analizzato e scartato. Assunto che l'operazione di verifica, a prescindere da dove venga effettuata, è inevitabile, abbiamo capito che anticiparla nella rete non avrebbe portato ad altro che vantaggi. Infatti, se un nodo, attraverso un IPS, rileva un pacchetto malevolo può scartarlo evitando così il consumo di risorse di tutti i router rimanenti lungo il cammino. Inoltre, con il nostro meccanismo di analisi distribuita è possibile scartare pacchetti riducendo il traffico per i nodi successivi, effetto che potrebbe portare a risparmi energetici ancora maggiori, rispetto a quelli attualmente osservati, grazie all'impiego dei router di nuova generazione, citati in precedenza.

Data la novità della nostra proposta, abbiamo dapprima studiato gli attuali mezzi a disposizione per la protezione da attacchi informatici così da poterli impiegare per i nostri fini. Essi saranno oggetto dei capitoli 2 e 3. Successivamente, cuore del nostro studio, abbiamo ideato ex novo un modello di analisi distribuita attraverso IPS, descritto nel capitolo 4, affrontando il problema con una prospettiva differente da precedenti lavori nel campo, per esempio [6]. Lo abbiamo implementato in un simulatore scritto in linguaggio Java, trattato nel capitolo 5, per valutarne le performance sotto diversi aspetti, capitolo 6. Infine, come riassunto nel capitolo 7, abbiamo appurato e dimostrato la veridicità delle nostre supposizioni con il pieno raggiungimento dell'obiettivo preposto.

Capitolo 2

Intrusion Detection System

Pertaining to techniques which attempt to detect intrusion into a computer or network by observation of actions, security logs, or audit data. Detection of break-ins or attempts either manually or via software expert systems that operate on logs or other information available on the network. [UCeRiverside, università della California]

Gli Intrusion Detection System sono sistemi, hardware o software, dedicatati alla rilevazione di attività indesiderate. Possono essere classificati in base a diversi aspetti:

1. Che informazioni monitora?
    a. Host-based (HIDS): si basa sulle informazioni del singolo pc;
    b. Network-based (NIDS): si basa sulla comunicazione della rete;
    c. Hybrid-based: combina i due precedenti;
2. Che metodo utilizza per determinare le intrusioni?
    a. Misuse-based: confronta le azioni osservate con i comportamenti malevoli noti, quindi in caso di riscontro rileva l'attacco;
    b. Anomaly-based: osserva deviazioni dal comportamento normale, appunto delle anomalie, rilevando l'attacco;
3. Come si comportano se rilevano un attacco?
    a. Passivo: inoltra il messaggio di allerta all'amministratore e sarà quest'ultimo a decidere come intervenire;
    b. Attivo: Oltre alla funzionalità del punto a, il sistema intraprende automaticamente delle azioni per impedire l'attacco o almeno rallentarlo, per esempio modificano lo stato e la configurazione del sistema. Generalmente nei NIDS si resetta quella connessione TCP oppure si aggiunge la regola di deny sul firewall/router per quel indirizzo IP o porta.

La risposta di un IDS può appartenere a una delle seguenti categorie:

1. Falso Positivo (FP): un evento innocuo classificato malevolo;
2. Vero Negativo (TN): un evento innocuo classificato correttamente;
3. Falso negativo (FN): un evento malevolo classificato innocuo;
4. Vero Positivo (TP): un evento malevolo classificato correttamente.

Sulla base di ciò può essere valutata la bontà dell'IDS, si rimanda alla sezione 2.3 per maggiori dettagli.

2.1 I diversi tipi di Intrusion Detection System

Tutti gli IDS monitorano informazioni per rilevare attacchi, ma i dati che hanno a disposizione sono di diversa natura a seconda di dove sono posizionati. Ovviamente, ciascuna delle due tipologie ha pro e contro: se con gli host-based si è in grado di determinare se il tentativo di attacco si è concretizzato e si possono dedurre attacchi locali, per esempio escalation dei privilegi e attacchi cifrati, con i network-based si rilevano attacchi distribuiti e si possono "facilmente controllare" più postazioni. Generalmente si utilizza il modello ibrido, combinando le due soluzioni in contemporanea per ottenere i vantaggi di entrambi.

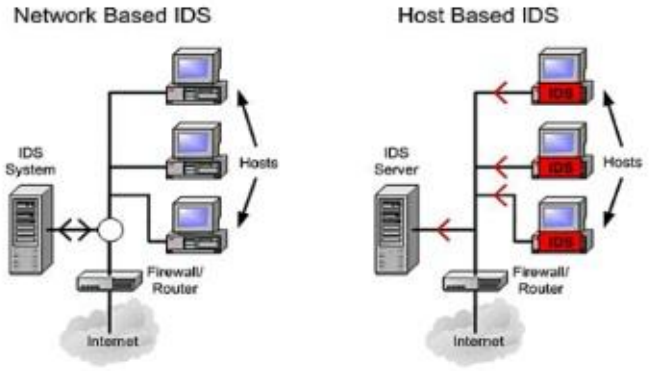

Figura 2.1: Architettura rete locale con IDS

2.1.1 Host Intrusion Detection System

Gli HIDS, come suggerisce il nome, lavorano sulle informazioni raccolte sul singolo pc. Ovviamente ne esistono di diversa complessità: il più semplice consulta i file di log delle varie applicazioni e quelli di sistema, controllando per esempio tentativi di login sospetti o accessi anomali a file; quelli un po' più complessi garantiscono un livello di sicurezza maggiore tenendo conto nelle loro analisi anche delle chiamate di sistema. Il principale pregio degli HIDS è che sono molto semplici e veloci ma, allo stesso tempo, potrebbero essere inefficaci in quanto un bravo attaccante è in grado di modificare i log ad hoc per nascondere le proprie tracce. Un esempio di questo tipo di sistema è OSSEC [50].

2.1.2 Network Intrusion Detection System

I NIDS monitorano il traffico della rete e quindi proteggono in contemporanea più host. Un NIDS è composto generalmente dalle componenti rappresentate in figura:

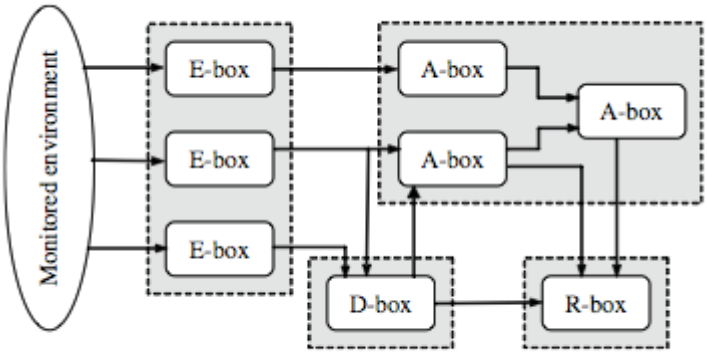

Figura 2.2: Componenti NIDS

1. Agenti o sensori (blocco E): Questi vengono posti in punti strategici della rete e osservano e analizzano cosa accade sul loro link. In caso di allerta inviano queste informazioni al management server;
2. Management server (blocco A): Colleziona le informazioni ricevute dai sensori e può effettuare ulteriori analisi per rilevare eventuali attacchi, per esempio un DDoS;
3. Database server (blocco D): Sono qui memorizzate tutte le informazioni necessarie al funzionamento del sistema;
4. Console (blocco R): Interfaccia per l'amministratore per monitorare lo stato della rete e degli host.

2.2 I diversi approcci di Intrusion Detection System

Come precedentemente scritto, gli IDS possono essere suddivisi in due grandi categorie, ciascuna dotata dei suoi pro e contro. Per questo motivo oggi giorno le due tecniche vengono combinate insieme per ottenere migliori risposte. In entrambi i casi, dato un vettore di input, ossia una serie di coppie attributi-valore, il sistema deve determinare una risposta.

2.2.1 Misuse-detection

L'idea alla base di questa tecnica è controllare se i dati osservati hanno riscontro con attacchi noti, se così è il sistema deduce non solo che vi è un attacco in corso, ma anche di che tipo e questo è fondamentale per il manager del sistema controllato. Gli attacchi noti sono generalmente codificati attraverso dei pattern, che prendono il nome di *signature*, e memorizzati in un database. Un pattern rappresenta, quindi, una classificazione dell'attacco in termini di

informazioni a priori che permettono di individuare e classificare ciò che è "malevolo". La tecnica di confronto più comune nell'ambito dei sistemi di tipo network è il pattern matching realizzato con le espressioni regolari[3] sia per la loro immediatezza che per le ottime performance di ricerca simultanea di numerose signature su un singolo flusso di dati. Altre tecniche, più complesse, includono l'utilizzo di sistemi basati su modelli formali e semi-formali che cercano di riconoscere l'insieme di transizioni eseguite per arrivare alla condizione di bersaglio.

I vantaggi dei misuse-detection sono diversi: in primo luogo non si hanno, o sono in numero ridotto, i falsi positivi evitando così falsi allarmi; in secondo luogo, se in modalità IPS, possono intraprendere le giuste azioni per impedire quel determinato attacco rilevato. Non sono ovviamente immuni ad aspetti negativi, principalmente sono in grado di riconoscere solo attacchi noti quindi completamente inefficaci contro i cosiddetti *zero-day-attack*. Inoltre il database deve essere costantemente aggiornato e aumenta continuamente le proprie dimensioni vista la grande fantasia degli attaccanti.

2.2.2 Anomaly-detection

Lo scopo di questo metodo, a differenza del precedente, è di imparare quale traffico è buono, ossia normale, andando poi a identificare come attacchi i comportamenti che si discostano da ciò, che sono appunto anomali.

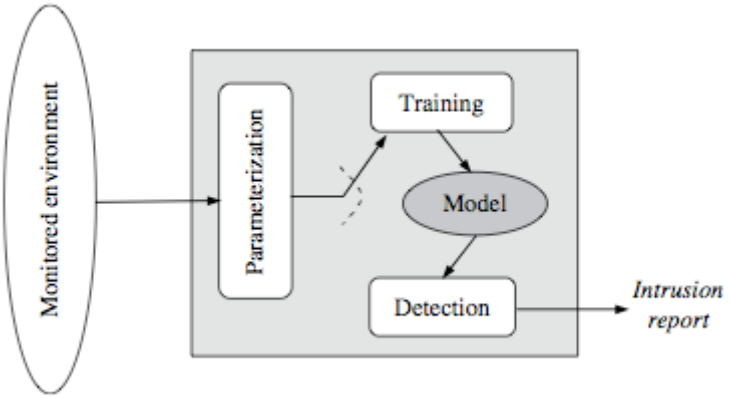

Figura 2.3: Schema di comportamento di un anomaly IDS

Si possono quindi distinguere due fasi:

1. Learning: Partendo dall'assunzione che tutto il traffico è "buono", si impara il normale comportamento della rete. I metodi con cui può essere eseguito questo addestramento

[3] Le espressioni regolari sono un modo per definire la ricerca di stringhe attraverso un modello di comparazione. Nello specifico si ricerca nel traffico osservato la stringa firma dell'attacco.

sono diversi, si rimanda per maggiori dettagli alle prossime sezioni, ma lo scopo comune è definire la linea tra buono-cattivo. Tale distinzione non è per nulla banale quindi è essenziale avere un *training set* grande e rappresentativo in modo tale da limitare il rate di falsi positivi;

2. Monitoring: Dato il modello costruito nella fase 1, si confrontano i dati a disposizione con esso. Ogni volta che una regola viene infranta, ossia si ha un comportamento anomalo, viene alzata la soglia di sospetto e quando essa raggiunge il valore limite viene lanciato l'allarme.

Il vantaggio principale di questa soluzione è la capacità di rilevare nuovi attacchi, cosa impossibile per i misuse. Altrettanto importante è l'unicità degli IDS che si "modellano" sul sistema o rete osservata deducendo regole ad hoc per quel contesto, di conseguenza un attaccante non può sapere a priori se la sua azione genererà o no un allarme poiché quello che viene riconosciuto come anomalo da un IDS non è detto che lo sia anche per un altro. Il difetto principale è l'altissima frequenza di falsi positivi che possono portare l'amministratore a ignorare gli allarmi, un po' come il famoso "Al lupo! Al lupo!" della celebre fiaba. La ricerca si sta occupando del problema cercando soluzioni per ovviarlo, in particolare i lavori si concentrano sulla fase di learning per minimizzare tale frequenza. Un altro difetto è che nell'apprendimento, in cui si assume che il traffico sia privo di anomalie, un attaccante può influenzare il sistema facendogli considerare come buono quello che è in realtà un attacco. Nei prossimi paragrafi analizzeremo più nel dettaglio come può essere realizzata questa fase così critica.

2.2.2.1 Metodi Statistici

Questa è la categoria più antica di algoritmi per la fase di apprendimento. L'idea alla base è costruire un modello probabilistico che dati alcuni parametri (per esempio la lunghezza del pacchetto, il tempo di inter-arrivo, ecc…) e i loro valori stocastici (per esempio la media, la varianza, ecc…) valuta se i dati osservati rispettano i profili imparati oppure no. Ovviamente quali e quanti parametri utilizzare dipende dalle singole realizzazioni. Facendo riferimento al modello descritto in [7], esso permette la creazione automatica di profili, ossia strutture che definiscono come il soggetto (l'utente che compie le azioni) interagisce con l'oggetto (ossia le risorse del sistema, per esempio file, programmi, ecc…). Questa interazione è rappresentata dall'*Audit Records*, più precisamente dalla sestupla: Subject, Action, Object, Exception-

Condition, Resource-Usage[4] e Time-stamp. Prima di descrivere i diversi modelli, è opportuno definire le metriche, ossia variabili aleatorie, utilizzate per l'analisi:

1. Event counter: numero di audit record che soddisfano una determinata proprietà in un certo intervallo temporale, per esempio il numero di login eseguiti in un'ora;
2. Interval timer: intervallo di tempo tra due eventi correlati, per esempio tra il login e il lougout;
3. Resource measure: l'ammontare di risorse consumate in un periodo, per esempio il tempo di CPU necessario per l'esecuzione di un programma.

L'autore propone cinque modelli:

1. Operational Model: Data una variabile aleatoria, si definisce a priori il suo valore limite, che viene confrontato con ogni osservazione, ossia il valore ottenuto dall'analisi degli audit record. Questo modello è indicato per situazioni di attacco ben noti, per esempio se si hanno più di 10 tentativi di login in pochi minuti è altamente probabile un'azione malevola.
2. Mean and standard deviation Model: Con questo modello si utilizza ciò che è stato osservato in passato, quindi non si deve avere una conoscenza a priori del sistema ma si adatta al contesto monitorato. Nel dettaglio, date le n osservazioni precedenti si calcola:

$$mean = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i \quad (1)$$

$$stdev = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n} - mean^2} \quad (2)$$

$$confidence\ Interval = mean \pm d \cdot stdev \ \ d \in R \quad (3)$$

   La nuova osservazione x_{n+1} è classificata anomala se non appartiene all'intervallo di confidenza, poiché la probabilità che x_{n+1} cada fuori da tale intervallo è al più $\frac{1}{d^2}$ (diseguaglianza di Chebyshev).
3. Markov process Model: Il comportamento del sistema è rappresentato con un processo markoviano in quanto la sua evoluzione, ossia le transizioni tra gli stati, non è deterministica ma probabilistica. Matematicamente si utilizzano le catene di Markov per calcolare $\underline{\pi}(n)$, il vettore che rappresenta la distribuzione di probabilità degli N stati all' iterazione n-esima.

[4]Lista di valori consumati per diverse risorse, per esempio numero di righe lette/stampate, occupazione in CPU, ecc…

Nello specifico, il metodo per classificare l'osservazione x_n è composto dalle seguenti fasi:

- Si determina lo stato s_i che rappresenta quel valore nel processo;
- Si recupera $\pi_i(n)$, ossia il valore della probabilità di s_i allo stadio n-esimo;
- Se $\pi_i(n)$ è troppo piccolo, cioè è altamente improbabile la transizione in quello stato allora l'osservazione è considerata anomala.

4. Time Series Model: Questo metodo combina una variabile di tipo interval timer con una delle altre due categorie per crearne la serie temporale, che può essere utilizzata per prevedere l'andamento futuro in base a quanto si è osservato in passato. Con questo modello si può misurare il trend del comportamento nel tempo individuando graduali ma significativi cambiamenti. E' quindi più precisa della tecnica 2 ma è più onerosa dal punto di vista computazionale.

2.2.2.2 I metodi Machine Learning e datamining

Il machine learning è una delle discipline fondamentali dell'intelligenza artificiale che si occupa di elaborare algoritmi auto-adattivi, cioè in grado di plasmare se stessi a partire da una fase di addestramento su un insieme dei dati del problema, detto training set. La linea di distinzione dai metodi statistici è estremamente labile, tanto che alcuni metodi, come per esempio i processi markoviani, vengono riportati in entrambe le categorie. Molti IDS utilizzano questa classe di algoritmi poiché le loro caratteristiche di dinamicità e adattamento ben si prestano a problemi in cui la conoscenza non è esplicita. Entrando nel dettaglio, per creare il modello è necessaria una fase di addestramento che può essere supervisionata, ossia i dati di training sono etichettati, cioè contengono la coppia input (i valori dei parametri) e output (la classe di appartenenza), oppure non-supervisionata cioè si ha a disposizione solo il vettore di input e il sistema genera automaticamente le classi di appartenenza, per esempio raggruppando insieme i dati simili. In [8] sono riportati i seguenti algoritmi:

- K-Nearest Neighbour: Utilizza l'apprendimento supervisionato ed è instance-based, ossia non generalizza le classi, ma classifica una nuova istanza confrontandola con i dati del training set in memoria, quindi nei fatti non costruisce un modello. Infatti, dato un nuovo vettore di input ricerca nel database le k istanze più vicine, per esempio in termini di distanza euclidea, e gli assegna la classe in maggioranza nel "vicinato". Il vantaggio di questo metodo è la flessibilità ai cambiamenti, ossia data una nuova osservazione classificata questa sostituisce il dato meno recente del training set mantenendo il contesto aggiornato, viceversa per le altre soluzioni tale operazione è molto più onerosa

perché il modello deve essere rielaborato. Lo svantaggio è sicuramente l'occupazione in memoria e il maggior tempo richiesto per la classificazione, se il training set ha dimensione *n* richiederà O(n).

- Support Vector Machine (SVM): Utilizza l'apprendimento supervisionato e realizza una classificazione binaria. I vettori di input vengono proiettati in un iperspazio e l'algoritmo determina l'iperpiano ottimo, $\underline{w} \cdot \underline{x} + b = 0$, che separa le regioni di classe diversa minimizzando l'errore empirico, ossia la differenza tra l'output corretto e quello predetto, e massimizzando la distanza geometrica, ossia la distanza euclidea tra due punti di classe diversa. Data una nuova osservazione $\underline{x}$ è classificata 0 se $\underline{w} \cdot \underline{x} + b < 0$, 1 altrimenti. L'algoritmo è applicabile solo a vettori linearmente separabili, ossia le classi possono essere separate da un piano, se così non è ci si può riportare a questa situazione aumentando la dimensione dello spazio, come illustrato in figura.

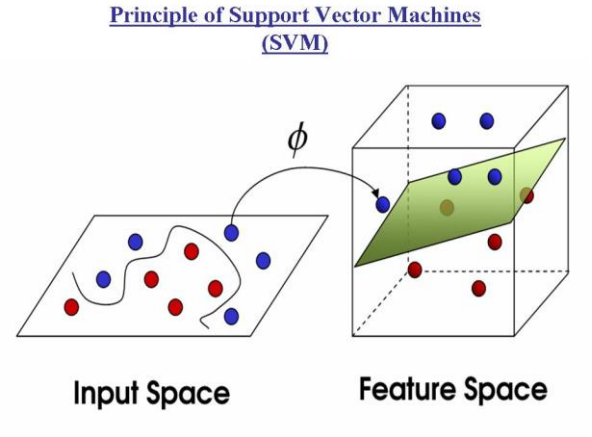

Figura 2.4: Principio di funzionamento SVM

Il punto di forza di questa tecnica è la possibilità di gestire vettori di grandi dimensioni, ossia utilizzare numerosi attributi per la classificazione senza impattare significativamente sulle prestazioni.

- Decision Tree: Utilizza l'apprendimento supervisionato per costruire l'albero delle decisioni. In tale struttura ogni nodo interno rappresenta una variabile, un arco verso un nodo figlio rappresenta un possibile valore per quella proprietà, infine i nodi foglia rappresentano il valore della variabile di output. Data una nuova osservazione, si esplora l'albero partendo dalla radice e scegliendo di volta in volta il figlio in cui l'input rispetta la proprietà fino ad arrivare al nodo foglia che ne definirà la classe di appartenenza.
- Rete Naive-Bayes: Utilizza l'apprendimento supervisionato per modellare le relazioni probabilistiche tra gli attributi di input e la classe di appartenenza. Infatti il modello

determina, nella fase di training, le probabilità condizionate $P(y_i|\underline{x})$ per ogni classe e data una nuova istanza di $\underline{x}$ la classifica con y_i per cui tale probabilità è massima.
Per calcolare le suddette probabilità, date N istanze, si utilizza il teorema di Bayes:

$$P(y_i|x_j) = \frac{P(x_j|y_i) \cdot P(x_j)}{P(y_i)} \quad (4)$$

Ove:

- $P(x_j) = \frac{n_j}{N}$ con n_j pari al numero di osservazioni di $x=x_j$;
- $P(y_i) = \frac{n_i}{N}$ con n_i pari al numero di osservazioni di $y=y_i$;
- $P(x_j|y_i) = \frac{n_{ji}}{n_i}$ con n_{ij} pari al numero di osservazioni di $x=x_j$ e $y=y_i$. Con questa formula gli eventi rari hanno probabilità 0, in quanto è difficile che siano presenti nel training set. Per limitare ciò possono essere usati dei metodi di smoothing, per esempio:

$$P(x_j|y_i) = \frac{n_{ji} + 1}{n_i + c} \quad c: numero\ di\ classi$$

- Rete neurale: Utilizza l'apprendimento supervisionato per costruire una rete neurale che trae ispirazione del cervello umano, ben noto per le sue capacità di apprendimento. Tale struttura è composta da una collezione di semplici unità di calcolo, dette neuroni, altamente interconnesse e il processo di elaborazione si basa sull'attivazione o inibizione di alcuni di essi. Tipicamente, il neurone artificiale ha una sola uscita e molti ingressi, ciascuno dei quali ha associato un peso. Ogni neurone è caratterizzato da una funzione di attivazione che attiva il neurone se supera la soglia θ. I pesi w_i sono inizialmente scelti casualmente in un intervallo di valori molto piccoli e vengono "adattati" nella fase di training minimizzando l'errore empirico.

$$funzioneAttivazione = f\left(\sum_{i=1}^{n} x_i \cdot w_i\right) \quad (5)$$

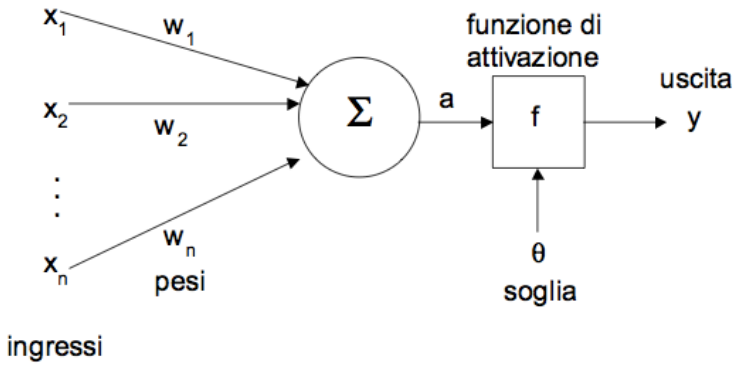

Figura 2.5: Principio di funzionamento reti neurali

Generalmente si utilizzano reti stratificate, ossia si individuano degli strati di neuroni tali che ogni neurone è connesso con tutti quelli dello strato successivo, ma non esistono connessioni tra i neuroni all'interno dello stesso strato, né tra neuroni di strati non adiacenti. Nel caso di un unico strato l'aggiornamento dei pesi può essere effettuato con la regola delta: sia t_j l'uscita stimata, y_j quella reale e η di rate di learning, si calcola:

$$\Delta w_i = \eta \cdot (y_j - t_j) \cdot x_i \; \eta \in [0,1] \quad (6)$$

$$w_i = w_i + \Delta w_i \quad (7)$$

Un tipo di rete neurale molto utilizzato negli IDS è il *Self-Organizing Map* poiché utilizza un apprendimento non supervisionato, ossia i pesi sono allenati per specializzare il neurone, cioè gli stessi input sollecitano lo stesso neurone di output. Uno dei vantaggi principali di questo insieme di algoritmi è che la fase di learning termina solo quando l'errore di stima è inferiore a una soglia impostata anche se ciò potrebbe implicare tempi di elaborazione molto lunghi, quindi è importante trovare un giusto compromesso tra accuratezza e performance.

- Algoritmi genetici: utilizzano l'apprendimento supervisionato e traggono ispirazione dalla teoria darwiniana. Come nella nota teoria data una popolazione, nel nostro contesto una famiglia di regole descriventi il traffico della rete, questa viene fatta evolvere applicando alle soluzioni "migliori" opportuni operatori genetici, realizzando cioè la selezione naturale. Più nello specifico, ogni soluzione è detta cromosoma ed è composta da diversi geni che ne costituiscono le varie parti, per esempio un gene potrebbe essere l'indirizzo IP sorgente. La popolazione evolve ad ogni iterazione dell'algoritmo, ossia:

- Si crea la nuova generazione, un insieme di cromosomi ottenuti scambiando geni tra cromosomi differenti (cross over) e/o scambiando la posizione dei geni su uno stesso cromosoma (mutazione);
- Si realizza la "selezione naturale" ossia si calcola il punteggio di ciascun cromosoma con un'opportuna fitness function. Nel nostro contesto il punteggio esprime il numero di occorrenze della regola nel training set (che si suppone essere privo di anomalie) quindi le soluzioni migliori saranno quelle con il punteggio più basso. Tutti i cromosomi con punteggio superiore a una determinata soglia vengono scartati.

L'algoritmo si arresta quando si soddisfa il criterio di terminazione che può essere una condizione sul numero di iterazioni e/o sulla bontà della migliore soluzione ottenuta.

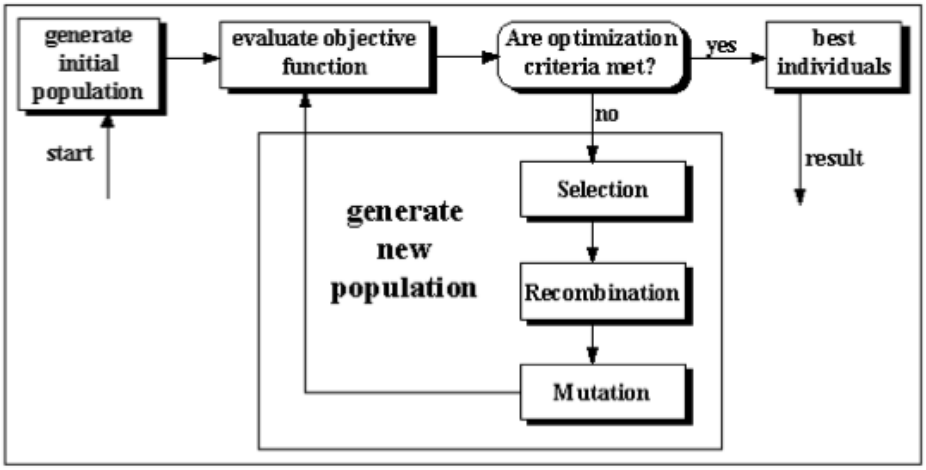

Figura 2.6: Principio di funzionamento algoritmo genetico

2.2.2.3 I metodi Immunitari

Il sistema immunitario umano è un ottimo esempio di intrusion prevention system, infatti ci protegge da patogeni (virus, batteri, funghi, ecc…) mai visti prima, il tutto in modo automatico e con ottime performance, ossia con un numero estremamente limitato di falsi positivi e negativi. Negli ultimi anni molti progetti e ricerche si sono focalizzati sull'emulazione dei meccanismi che regolano tale sistema, cronologicamente si possono distinguere tre macro approcci:

1) Classico: Si simula il comportamento del nostro sistema immunitario adattando algoritmi già esistenti;
2) Selezione negativa: Realizzano la suddetta teoria, ossia dato un patogeno le nostre cellule T generano il relativo antigene. Questo può essere self, ossia riconosciuto come non

dannoso per l'organismo, oppure not-self, ossia estraneo, generando una risposta immunitaria.

Tale processo si compone di tre fasi:

a. Si definisce cosa è self, ossia il normale comportamento della rete o del sistema;
b. Si generano i detectors, ossia gli anticorpi nell'analogia. Questa fase è estremamente critica poiché si deve trovare il giusto compromesso tra il tempo richiesto per l'elaborazione e l'accuratezza, espressa come rate di detection. Esistono diversi modelli, per esempio Kim e Bentley propongono di generarli casualmente e se si ha un riscontro con quanto definito nel punto a, vengono eliminati, altrimenti mantenuti. Sperimentalmente si è però dimostrato che per avere un livello di accuratezza accettabile, soprattutto in problemi di grandi dimensioni, questa fase richiede troppo tempo, quindi Ayara et al. hanno proposto un'estensione, ossia se si ha un riscontro con i dati self, il detector non viene eliminato ma mutato. Altre soluzioni generano gli anticorpi negando ciò che si osserva nella fase precedente.
c. Monitoring: Si rileva un'anomalia se i dati monitorati hanno un riscontro con uno dei detector.

Il vantaggio principale di questo gruppo di algoritmi è la possibilità di stimare la frequenza di identificazione che è relazionata al numero di anticorpi. Il difetto, invece, che considerare cattivo tutto ciò che non è self genera un elevato numero di falsi positivi, infatti nel corpo umano vi sono casi di patogeni not-self, per esempio i batteri dell'intestino, che non generano risposte immunitarie. Un sistema che utilizza quest'ultimo approccio è LYSIS.

3) Teoria del pericolo: Questa nuova teoria, che genera ancora controversie nel campo della biologia, non nega la selezione negativa ma ne sottolinea la semplicità. La teoria del pericolo lega la risposta immunitaria alla causa della morte delle cellule: una cellula termina la propria vita o per apoptosi, ossia un processo naturale di morte programmata, o per necrosi, ossia morte imprevista per cause non naturali. Quando una cellula termina la propria vita per apoptosi rilascia un segnale di basso pericolo e quindi generalmente ignorato dal sistema immunitario, viceversa per necrosi il segnale è di alto pericolo e perciò il sistema reagisce. Negli ultimi anni sono stati condotti molti studi orientati a emulare questo processo per realizzare IDS con bassa frequenza di falsi positivi poiché l'allarme sarà inviato solo a seguito di segnali di alto pericolo. Un sistema che realizza questo approccio è CARDINAL.

2.3 Altre caratteristiche di distinzione tra gli IDS

2.3.1 Stateful vs Stateless

Gli IDS possono essere distinti tra stateless e stateful in base all'approccio utilizzato per l'analisi degli eventi. I primi considerano ogni singolo evento "a se" poiché, come suggerisce il nome, non memorizzano alcuna informazione sui dati elaborati in precedenza; essi sono semplici da progettare e garantiscono alte velocità di elaborazione, ma non sono molto efficienti. Infatti, ad esempio, non possono rilevare attacchi caratterizzabili solo sulla base dell'osservazione di un flusso di eventi, ossia attacchi in cui un evento può avere effetti diversi a seconda della sua posizione nella sequenza. I secondi, al contrario, sono in grado di dedurli mantenendo informazioni sullo storico e sono più efficienti nel rilevare attacchi di tipo distribuito. D'altro canto sono più complessi da progettare e sono vulnerabili ad attacchi di tipo Denial of Service che "intasano" l'attività dell'IDS. Infatti, essendo costretti a mantenere le informazioni di tutti gli eventi osservati, un attaccante può riempirne il buffer impedendo la memorizzazione di nuovi dati e quindi un corretto funzionamento.

2.3.2 Centralizzata vs distribuita

Un ulteriore elemento di distinzione tra gli IDS, soprattutto per i network-based, è l'architettura: centralizzata o distribuita. Nel primo caso vi è un'unica macchina, posta nel punto in cui la rete interna si connette a Internet, che esegue le operazioni necessarie per la sicurezza; ciò implica che esiste un unico punto di fallimento. Questa tipologia è sicuramente più semplice da realizzare ma è destinata a diventare sempre più critica da applicare vista la continua crescita di complessità e dimensione delle reti. Al contrario, nella seconda architettura, vengono impiegati diversi componenti: i sensori, distribuiti in diversi punti nella rete, raccolgono i dati da elaborare e li inviano a un sistema centrale, detto Central Engine, che memorizza gli eventi, stabilisce se c'è un allarme, e lo comunica, nella forma prestabilita, all'utente, per esempio via e-mail o effetto sonoro. Ovviamente la comunicazione tra i suddetti componenti deve essere sicura, quindi è supportato un meccanismo di autenticazione con un formato noto. Generalmente si preferisce questa seconda soluzione poiché supera le limitazioni della prima essendo più efficace nel monitoraggio di reti di grandi dimensioni e non rappresentando un unico punto di fallimento.

2.4 Criteri di valutazione

Allo scopo di valutare le risposte fornite da un IDS, si possono calcolare diversi indici di prestazione:

- Sensibilità o frequenza di identificazione (detection rate): Esprime la percentuale di attacchi dedotti dal sistema calcolando il rapporto tra quelli correttamente individuati e il totale delle azioni malevole. Matematicamente

$$sensibilità = \frac{TP}{TP + FN} \cdot 100 \quad (8)$$

  Ovviamente l'IDS ideale avrà questo valore pari al 100%, ossia qualsiasi attacco sarà sicuramente classificato come malevolo.

- Specificità: Esprime la stessa informazione della sensibilità considerando come campione non le azione malevoli ma quelle innocue. Matematicamente

$$specificità = \frac{TN}{TN + FP} \cdot 100 \quad (9)$$

  Come sopra, l'IDS ideale avrà questo valore pari al 100%, ossia non saranno generati falsi allarmi. Infatti si definisce frequenza di falsi positivi

$$1 - specificità \quad (10)$$

- Accuratezza: Esprime la percentuale di risposte corrette calcolando il rapporto tra i dati classificati in modo esatto e il campione totale. Matematicamente:

$$accuratezza = \frac{TN + TP}{TN + FN + TP + FP} \cdot 100 \quad (11)$$

Per analizzare il comportamento di un IDS al variare dei propri parametri, per esempio la soglia di sospetto per l'invio dell'allarme, o per confrontare due o più sistemi, si utilizza la *Receiver Operating Characteristic curve* (ROC). Tale curva mostra graficamente la relazione tra la sensibilità e la frequenza di falsi positivi, rispettivamente sulle ordinate e sulle ascisse. Come già detto, tanto più la curva è vicina all'asse y nel suoi valori maggiori, vedi punto perfetto in figura, tanto più l'IDS si avvicina all'ideale.

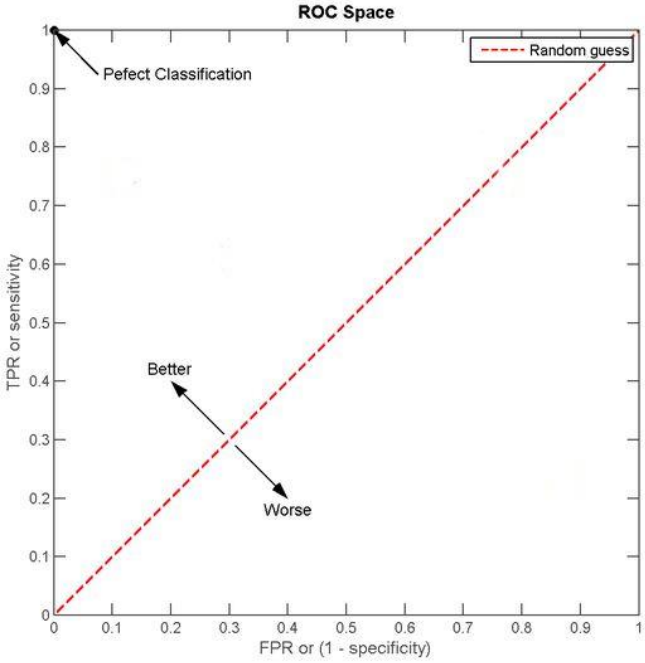

Figura 2.7: Curva ROC

2.5 Alcuni prodotti

Dopo aver analizzato le varie categorie e le relative proprietà degli IDS, proseguiamo con una panoramica dei prodotti più diffusi sul mercato. Per quanto riguarda gli IDS commerciali, indicati per reti complesse e di grandi dimensioni, le soluzioni più applicate nel mondo industriale sono quelle Cisco e IBM, seguite da Juniper Networks, McAfee e Sourcefire. Per il mondo open source esistono moltissime soluzioni, ma noi ci limiteremo ad analizzare nel dettaglio quattro sistemi tra i più noti: Snort, Bro, Suricata e Prelude.

2.5.1 Snort

Snort® is an open source network intrusion prevention and detection system (IDS/IPS) developed by Sourcefire. Combining the benefits of signature, protocol, and anomaly-based inspection, Snort is the most widely deployed IDS/IPS technology worldwide. With millions of downloads and nearly 400,000 registered users, Snort has become the de facto standard for IPS. [9]

Snort è un NIDS open source, distribuito con licenza GPL, creato da Martin Roesch nel 1998. In questi quindici anni il Sourcefire team ha continuato a migliorarlo e aggiornarlo, attualmente l'ultima versione, rilasciata nel novembre 2013, è la 2.9.5.6. Questo applicativo, compatibile con la stragrande maggioranza dei sistemi operativi, è estremamente leggero, performante ed esegue, in tempo reale, l'analisi del traffico e il logging dei pacchetti di reti IP.

Snort è basato su regole ed è attualmente così diffuso da essere considerato uno standard de facto. Entrando sempre più nel dettaglio, può lavorare in diverse modalità:

- Sniffer mode: Cattura il traffico della rete, utilizzando la libreria libpcap, e lo visualizza a video;
- Packet logger mode: Cattura il traffico della rete e lo salva su disco;
- Intrusioni detection mode: Cattura e analizza il traffico della rete generando avvisi in base alle regole definite;
- Inline mode: Cattura e analizza il traffico della rete bloccando i pacchetti in base alle regole definite, ossia si comporta come un IPS.

Snort utilizza un'architettura modulare composta da quattro componenti:

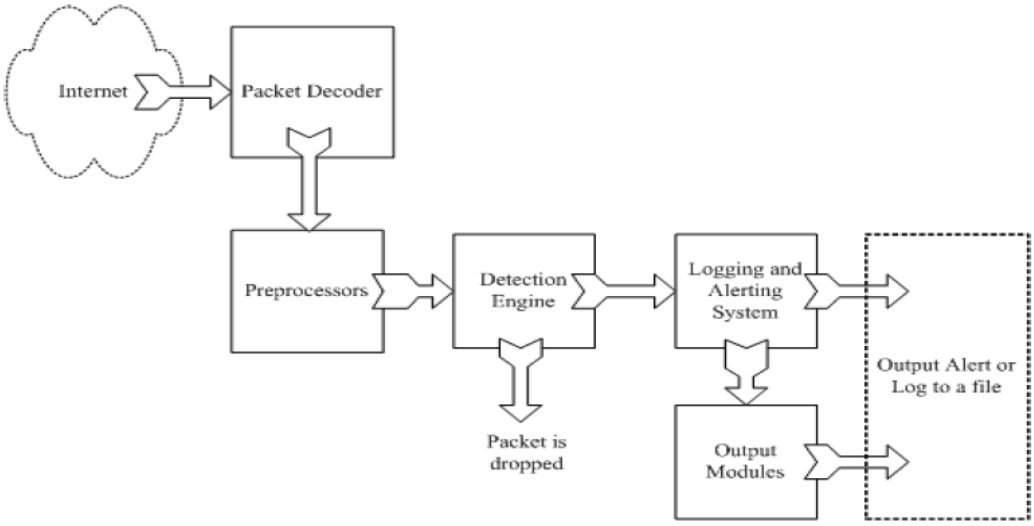

Figura 2.8: Architettura Snort

- Packet Decoder: Questo modulo si occupa della decodifica del pacchetto, più precisamente inizializza un'opportuna struttura "pacchetto" che ha come campi tutti gli elementi presenti negli header di livello due, tre e quattro. Nello specifico, si analizza nell'ordine:
  a. Livello datalink: Si determina il protocollo di livello due in uso e si invoca la relativa funzione di decodifica che inizializzerà i campi della suddetta struttura. Per esempio, se si rileva un frame Ethernet, verrà invocata la funzione *DecodeEthPkt()* che memorizzerà nella struttura il MACAddress sorgente, destinatario, ecc…
  b. Livello rete: Come sopra, ma decodifica e memorizza i campi del protocollo IP;
  c. Livello trasporto: Come sopra, ma decodifica e memorizza i campi del protocollo di livello 4 in uso. Snort supporta al momento la decodifica per TCP, UDP e ICMP.

2) Preprocessor: Questi plugin sono ideati per estendere le funzionalità di Snort e aumentarne l'efficacia. Per esempio *HTTPdecode* normalizza il contenuto del pacchetto in formato ASCII e viene utilizzato per evitare l'evasione di alcune regole. Supponiamo che il sistema reagisca a tentativi di accesso all' URI *"/wwwboard/passwd.txt"*, se il plugin non è abilitato e l'attaccante utilizza l'URI con caratteri esadecimali, ossia *"%2Fwwwboard%2F passwd.txt"*, Snort non reagisce; cosa non più vera se il preprocessor è abilitato perché prima di testare la regola l'URI verrà convertita nel formato standard. Altri plugin sono: *portScanning*, *frag3* e *stream5*, per la ricostruzione dei flussi a livello IP e TCP rispettivamente, e *SPADE*, oggetto del paragrafo 2.3.1.3, per il rilevamento di comportamenti anomali.
3) Detection Engine: Questo modulo è il "cuore" di Snort in quanto realizza il pattern matching del pacchetto con le regole. A seconda della modalità utilizzata l'analisi può terminare o alla prima regola soddisfatta o alla fine del rule-set, nel primo caso verrà eseguita solo l'azione della relativa regola, nel secondo tante azioni quante sono le regole verificate dal pacchetto. Ogni volta che una regola è verificata vengono invocati gli output plugins.
4) Output plugins: Ogni modulo realizza una diversa forma rappresentativa per i messaggi di alert e log, per esempio il formato dedicato alla memorizzazione su database, il formato XML per log su SNML (Simple Network Markup Language), ecc…

2.5.1.1 Le regole

Le regole sono l'elemento base di Snort e aggiungono un importante fattore di flessibilità e personalizzazione del sistema. Infatti scrivere una regola è relativamente "semplice" poiché è esprimibile con un linguaggio immediato ma allo stesso tempo potente. In questa sezione analizzeremo gli elementi basilari per la composizione di una regola, per maggiori dettagli si consiglia di consultare [9].

Per prima cosa ogni regola deve rispettare il seguente pattern:

azione protocollo ipsrc portsrc dir ipdst portdst (opzioni)

- azione: questo campo specifica l'azione da eseguire: alert, per visualizzare a video l'allerta, log per memorizzarla su disco, activate, per generare un alert e attivare una regola dynamic, e dynamic, inattiva fino a quando non è verificata la precondizione specificata con activate;
- protocollo: questo campo specifica il livello di analisi, ossia IP, TCP, UDP, ICMP;
- IPsrc e portsrc: indirizzo e porta da cui proviene il pacchetto;

- dir: indica la direzione del traffico, per esempio -> per indicare traffico dalla sorgente alla destinazione, <> per indicare traffico bidirezionale;
- IPdst e portdst: indirizzo e porta destinazione del pacchetto;
- opzioni: una lista di coppie *keywords:valore* separate da punto virgola che, come dice il nome, definisco le opzioni della regola;

Passando dal teorico al pratico, supponiamo di dover scrivere una semplice regola per dedurre un SIP INVITE flood che generi un allarme con messaggio "BLEEDING-EDGE VOIP INVITE Message Flood" se si rilevano più di 100 INVITE in 60 secondi sulla porta 5060 da qualsiasi mittente, ciò verrà tradotto in:

```
alert ip any any -> $HOME_NET 5060
(msg:"BLEEDING-EDGE VOIP INVITE Message Flood; content: "INVITE";
depth:6; threshold: type both, track by_src, count 100, second 60;
classtype: attempted-dos; sid:2003192;rev:1;)
```

2.5.1.2 Preprocessor SPADE

SPADE è acronimo di Statistical Packet Anomaly Detection Engine in quanto utilizza la probabilità congiunta per determinare eventuali anomalie nel traffico. Più precisamente dato un nuovo pacchetto, individuato con la quadrupla *IPsrc*, *IPdst, portsrc* e *portdst*, ne calcola la probabilità, in termini di frequenza relativa, che verrà utilizzata per determinarne il punteggio di anomalia. Se esso è maggiore della soglia fissata verrà generato l'allarme, altrimenti no. Ovviamente la scelta della soglia è un fattore estremamente critico, infatti influenza notevolmente la frequenza di falsi positivi. Per tale motivo esiste un opzione del plugin che permette di adattare periodicamente il valore della soglia basandosi su ciò che si è osservato nel passato.

Analizziamo alcuni esempi di alert riportati in [10]:

[**] [104:1:1] **spp_anomsensor: Anomaly threshold exceeded: 3.8919** [**] 08/22-22:37:00.419813 24.234.114.96:3246 -> VICTIM.HOST:80 TCP TTL:116 TOS:0x0 ID:25395 IpLen:20 DgmLen:48 DF ******S* Seq: 0xEBCF8EB7 Ack: 0x0 Win: 0x4000 TcpLen: 28 TCP Options (4) => MSS: 1460 NOP NOP SackOK

Questa allerta deriva dal tentativo di connessione al server web con indirizzo VICTIM.HOST che, però, non esiste. Il punteggio di anomalia, 3.8919, non è particolarmente elevato in quanto il sistema si è "assuefatto" al traffico sulla porta 80. Questo esempio mostra la più rilevante debolezza di tale approccio, ossia se si immette un'ingente quantità di traffico malevolo il sistema inizierà a considerarlo normale.

[**] [104:1:1] **spp_anomsensor: Anomaly threshold exceeded: 10.5464** [**] 08/22-22:22:46.577210 24.41.81.216:2065 -> VICTIM.HOST:27374 TCP TTL:108 TOS:0x0 ID:10314 IpLen:20 DgmLen:48 DF ******S* Seq: 0x63B97FE2 Ack: 0x0 Win: 0x4000 TcpLen: 28 TCP Options (4) => MSS: 1460 NOP NOP SackOK

In questo secondo e ultimo esempio, il punteggio di anomalia, 10.5464, è molto elevato in quanto il pacchetto è estremamente raro nella rete, infatti non vi è alcun servizio sulla porta 27374. Anzi, investigando ulteriormente si è notato che essa è generalmente associata a Sub Seven Trojan, una rinomata backdoor.

2.5.2 Bro

Bro, creato da Vern Paxson nel 1998, è un NIDS open source e Unix-based che monitora il traffico della rete in cerca di attività sospette. Dopo tanti anni di miglioramenti e nuove funzionalità, l'ultima versione rilasciata, nel novembre 2013, è la 2.2. Bro è estremamente efficiente e può essere utilizzato in reti con velocità nell'ordine dei Gbps senza perdere pacchetti; altrettanto, ad esempio, non si può dire di Snort.

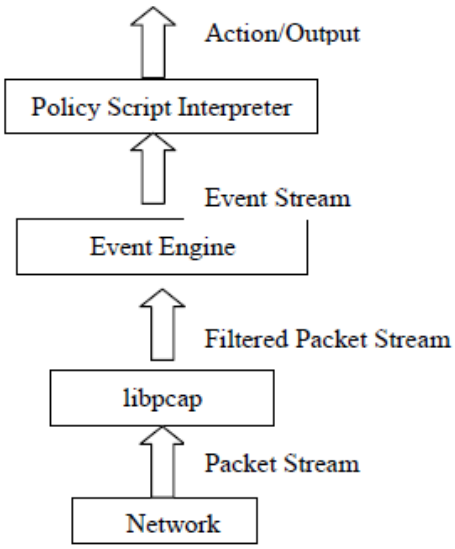

Figura 2.9: Diagramma funzionale Bro

Il processo di deduzione è composto da diverse fasi rappresentate in figura. Per prima cosa, utilizzando la libpcap, si decodifica il traffico di rete, scartando gli elementi considerati irrilevanti per l'analisi e inoltrando le restanti informazioni, memorizzate in un'opportuna struttura "pacchetto", all' *Event Engine*. Quest'ultimo, scritto in linguaggio C++, ha il compito di ricavarne la semantica a livello applicativo, ossia interpreta i dati ricevuti e li astrae in uno o più eventi di alto livello che descrivono l'attività individuata. Infine gli eventi sono inviati al *policy script interpreter*, che, gestendoli con politica FIFO, vi applica i *policy script* che si occupano di rilevare eventuali intrusioni. Uno dei punti di forza di questo sistema risiede proprio nella potenza di tali script che possono individuare intrusioni sia con pattern matching

con firme di attacchi noti che con rilevazioni di attività inusuali, ossia non conformi con il profilo dell'applicazione, come per esempio un numero di tentativi di login in un dato periodo superiore a una certa soglia. Quindi le "logiche" di analisi non solo sono più sofisticate ma possono anche essere personalizzate dall'utente in base al proprio contesto di utilizzo.

2.5.2.1 Policy script

I policy script sono scritti con un linguaggio proprietario di Bro che non è particolarmente immediato ma, per fortuna, esistono diversi tool che aiutano l'utente nello sviluppo. Il sistema è già fornito di una serie di script che analizzano diversi protocolli, come per esempio HTTP, SSH, FTP, DNS e altri ancora; questi script filtrano ed elaborano i risultati finali, per esempio per il logging, l'alert e il reporting. A differenza dei sistemi signature-based (per esempio Snort) Bro è principalmente un anomaly-based poiché confronta quanto osservato sulla rete con il profilo desiderato dell'applicazione. Esistono due diversi tipi di policy script: gli script "puri", che hanno maggiore priorità, e le firme.

Analizziamo un semplice esempio di script: generazione di un notice, ossia un messaggio di log all'utente, se l'host con indirizzo IP 1.1.1.1 effettua più di cinque connessioni TCP sulla porta 313. Il codice che lo realizza è il seguente:

```
Global attack_count = 0;
event connection_established(c: connection)
{
if ( c$id$orig_h == 1.1.1.1 &&
c$id$resp_p == 313/tcp &&
++attack_count == 5 )
      NOTICE([$note=Attack,
                  $conn=c,
                  $msg=fmt("Attack from %s to destination: %s", c$id$orig_h, c$id$resp_h)]);
}
```

Per quanto riguarda le regole, esse hanno una struttura simile a quella di Snort: una regola è univocamente individuata dal nome che segue la keyword *signature* e da una serie di attributi, le condizioni da verificare e l'azione da intraprendere, espressi tra parentesi graffe. Supponiamo di voler definire una regola con nome univoco testing che invii un messaggio "Found windows boot" se nell' http request viene rilevato il pattern *"boot.ini"*, il codice sarà il seguente:

```
Signature testsig {
   ip-proto == tcp
   dst-port == 80
   http-request /.*(boot.ini)/
   event "Found windows boot!"
}
```

Per sottolineare il potere espressivo del linguaggio, consideriamo come rilevare un cross-site scripting (xxs). Brevemente, il xxs è uno degli attacchi più utilizzato in internet che sfrutta un'inadeguata sanitizzazione[5] dell'input. Utilizzando tale mancanza, l'attaccante è in grado di eseguire il proprio codice maligno, inviando come parametro per la richiesta HTTP *<script>codice malevolo</script>*. Se utilizziamo un sistema basato su regole, possiamo rilevare tale minaccia inserendo una regola che invii un'allerta se rileva il carattere ASCII '<' che può essere facilmente evasa se l'attaccante utilizza una diversa codifica, per esempio l'esadecimale *'%3C'*. Generalmente un malevolo prova più codifiche sperando di trovare quella non controllata, ciò può portarlo al successo in sistemi basati su regole ma altrettanto non si può dire con uno script di Bro, che invii un allarme se il numero di caratteri non ASCII supera una certa soglia. Per maggiori dettagli consultare [11].

2.5.3 Confronto tra SNORT e Bro

Snort e Bro sono estremamente diffusi nel mercato degli IDS open source quindi è interessante confrontarli su diverse caratteristiche.

Parameter	Bro	Snort
Contextual signatures	yes	no
Flexible site customization	high	medium
High speed network capability	high	medium
Large user community	no	yes
Configuration GUI	no	yes
Analysis GUI	a few	a lot
Installation /deployment	difficult	Easy
Operating System compatibility	Unix	Any

Figura 2.10: Confronto tra Bro e Snort

In accordo con [12]:

- Velocità: Bro può essere utilizzato in reti di dimensioni maggiori rispetto a Snort. Infatti il primo, a differenza del secondo, riesce a gestire senza perdita di pacchetti o rallentamento di tutto il traffico velocità nell'ordine dei Gbps;
- Firme: Come già detto, le firme di Bro sono sicuramente più sofisticate di quelle di Snort;
- Flessibilità: Bro è estremamente flessibile, ossia può essere molto personalizzato ed esteso nelle funzioni con script in linguaggio Bro ad hoc. Anche Snort permette la

[5] La sanitizzazione implica la modifica del contenuto di un parametro per evitare una risposta non documentata dello script.

definizione di nuove regole, ma per estenderne le funzionalità è necessario scrivere complessi moduli, preprocessor e/o plugin di output, in C;

- Installazione: Snort è molto più semplice e rapido da installare rispetto a Bro;
- GUI: Snort, al contrario di Bro, ha diverse interfacce grafiche, per esempio *Snorby*, che permettono di elaborare e visualizzare diverse analisi in modo estremamente semplice. Invece per poter comprendere a pieno i risultati prodotti da Bro è indispensabile un ottima conoscenza dei sistemi Unix;
- Compatibilità con i sistemi operativi: Snort può essere eseguito praticamente su tutti i sistemi operativi, al contrario Bro solo sui sistemi Unix.

2.5.4 Prelude IDS

Prelude, ideato da Yoann Vandoorselaere nel 1998, è un sistema open source, distribuito con licenza GPL, rule-based relativamente diffuso. In tutti questi anni sono stati apportati diversi aggiornamenti, seppur con frequenza minore rispetto ai sistemi precedentemente descritti, l'ultima versione rilasciata, nel settembre 2013, è la 1.1.0. Sicuramente ciò che lo differenza dagli altri sistemi è l'utilizzo di un'architettura ibrida, ossia il sistema può gestire contemporaneamente sia HIDS che NIDS.

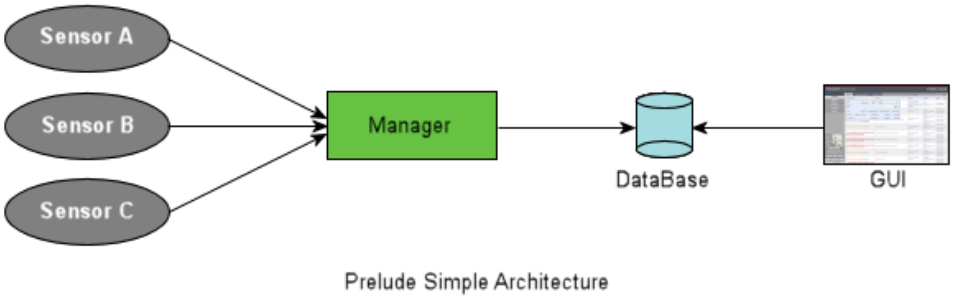

Figura 2.11: Architettura Prelude

Prelude utilizza una struttura modulare composta dai seguenti elementi:

- Sensore: Questo componente analizza i dati alla ricerca di anomalie e tentativi di intrusione, inviando al modulo Manager gli allarmi generati, conformi allo standard IDMEF, con connessione TLS. Prelude supporta diversi tipi di sensori come Snort, OSSEC e altri ancora.
- Manager: Questo componente è il cuore del sistema che applica le regole per compiere un'analisi più approfondita alle allerte ricevute dai sensori o, in configurazioni più complesse, da altri manager. In caso di intrusione, memorizza il relativo messaggio di errore, sempre in formato IDMEF, sul database.

- GUI: Questo componente è utilizzato dall'amministratore per visualizzazione in modo più user-friendly le allerte rilevate dal sistema. Sul sito ufficiale [13] si consiglia l'utilizzo di *Prewikka*.

2.5.4.1 IDMEF

Intrusion Detection Message Exchange Format garantisce l'interoperabilità tra i diversi sistemi e facilita la comunicazione con i moduli di gestione e correlazione. Brevemente, poiché ciò non è l'obiettivo della nostra tesi, lo standard RFC 4765 definisce il formato dei messaggi di allerta come messaggi XML o di tipo alert, per la rappresentazione degli allarmi, o heartbeat, per riferire lo stato del sensore al manager. La scelta di tale linguaggio aumenta notevolmente la leggibilità, sia per la semplicità della sintassi che per la sua ampia diffusione nel mondo IT, ma introduce un overhead maggiore sia in termini di occupazione di memoria che di utilizzo di CPU.

2.5.5 Suricata

The Suricata Engine is an Open Source Next Generation Intrusion Detection and Prevention Engine. This engine is not intended to just replace or emulate the existing tools in the industry, but will bring new ideas and technologies to the field. [16]

Brevemente, poiché sarà oggetto del prossimo capitolo, Suricata è un NIDS rule-based open source, rilasciato da Open Information Security Foundation nel dicembre 2009. Benché il software non abbia ancora raggiunto la sua completezza, non manca di integrare tecnologie e funzionalità degne di un next-gen IDPS engine. Infatti, Suricata è considerato da molti un sistema di nuova generazione poiché introduce diverse importanti novità: un'architettura multithreading, l'accelerazione grafica per l'analisi tramite la tecnologia nVidia CUDA® per il GPGPU, il trigger di script in linguaggio LUA, regole più sofisticate grazie alle Flow variables, l'uso di HTP per il parsing di dati http. Grazie a queste caratteristiche Suricata risulta essere estremamente performante e scalabile: essendo multi-thread il sistema bilancia il carico su ciascun processore, così da gestire reti con velocità nell'ordine dei 10 Gbps senza ne dover ridimensionare lo spazio delle regole ne perdere pacchetti.

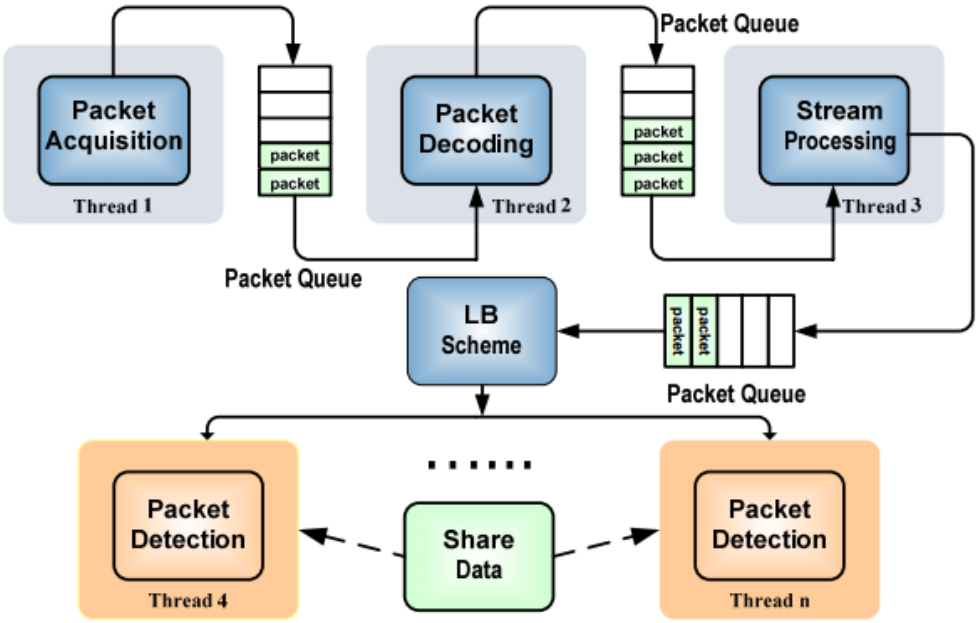

Figura 2.12: Diagramma funzionale Suricata

Il processo di analisi è composto da più passi, ciascuno realizzato da un thread che riceve i dati dal precedente e fornisce i propri output al successivo. Il primo modulo che entra in funzione è il *packet acquisition* che acquisisce i pacchetti dalla rete e li passa al *packet decoding*, che si occupa di decodificare il protocollo memorizzando le informazioni necessarie per l'analisi contenute nei campo dell'header. Nel terzo passo, realizzato dal *stream processing*, si riassemblano insieme i pacchetti che appartengono allo stesso flusso. Infine il *load balance* si occupa di assegnarli ai vari task di detection con lo scopo di sfruttare il più possibile il processore multi-core, ossia minimizzare l'idle time di ciascuno di essi. Infine il sistema è multipiattaforma, quindi compatibile con la maggior parte dei sistemi operativi.

Capitolo 3

Suricata

Figura 3.1: Logo Suricata

3.1 The Open Information Security Foundation

La Open Information Security Foundation (OISF) è una fondazione non-profit nata per costruire un IDS/IPS open source di nuova generazione. Il team è composto da illustri esponenti del campo della sicurezza informatica, per esempio Matt Jonkman, fondatore di *Emerging Threats*, o Victor Julien, creatore del firewall *Vuurmuur* e sviluppatore per il progetto Snort inline. In linea con la filosofia open source, richiede un ampia partecipazione della comunità sia attraverso la mailing list [15] che lo sviluppo di vero e proprio codice. Infine, per quanto riguarda i finanziamenti del progetto essi provengono direttamente dal *US Department of Homeland Security* (DHS) e dalle compagnie private che formano il OISF Consortium. Si rimanda per maggiori informazioni al sito ufficiale [16].

3.2 Caratteristiche del software

Suricata nasce nel Novembre 2009 con l'obiettivo di essere un sistema di detection innovativo, cioè dotato di tutte le caratteristiche indispensabili per un NIDS di nuova generazione. Per raggiungere tutte le funzionalità ideate in fase di progettazione, il sistema è in continuo aggiornamento, l'ultima versione *stabile*, la 1.4.6, è stata rilasciata il 24 Settembre 2013 mentre la *beta* 2.0beta1 nel Luglio dello stesso anno.

Suricata ha un'architettura multi threading, oggetto del paragrafo 3.3, che lo rende altamente scalabile. Questa caratteristica è probabilmente la più innovativa e rilevante, infatti è attualmente l'unico IDS che ha optato per questa soluzione sfruttando così a pieno i vantaggi dei processori multi-core in quanto, eseguendo l'istanza di Suricata, il sistema bilancerà il carico computazionale su ciascuno di essi. La scelta è sicuramente vincente poiché permette di gestire reti con velocità nell'ordine dei 10 Gbps senza dover sacrificare la "copertura" delle regole, ossia diminuire il numero di regole da testare. Per migliorare ulteriormente le performance, Suricata utilizza l'accelerazione grafica per l'analisi tramite la tecnologia nVidia CUDA® per il GPGPU. Non entrando troppo nello specifico, si utilizza tale tecnologia per il task di detection, che è il più impegnativo dell'intero processo, ossia ciascun Graphics Processing Unit esegue il pattern-matching con le regole.

Un altro punto di forza del sistema è l'efficiente meccanismo automatico di identificazione del protocollo che permette l'individuazione in tempi brevissimi, generalmente a stream appena iniziato. Ciò rende Suricata il miglior sistema nel rilevare malware Command and Control Channels, uno degli attacchi più frequenti, e permette di scrivere regole più precise e sofisticate rispetto agli altri sistemi rule-based, come per esempio Snort. In particolare, con Suricata è possibile specificare nella regola direttamente il protocollo e non la porta su cui si dovrebbe ricevere quel tipo di traffico, per esempio HTTP invece che porta 80, rendendo così inutili tentativi di evasione da parte dell'attaccante. Definito il protocollo si possono controllare diversi campi, per esempio l'URI della richiesta HTTP o l'identificatore del certificato SSL.

Un altro vantaggio rispetto ad altri sistemi è l'identificazione in real-time di un migliaio di tipi di file che passano sulla rete. Inoltre, sempre con le regole, è possibile estrarre determinati file dalla rete, cioè ricopiarli su disco aggiungendo meta file che riportano lo stato e il flusso al momento della cattura. Per esempio, se si desidera estrarre tutti i file pdf in ingresso e uscita, si può utilizzare la seguente regola:

```
alert http any any -> any any (msg:"FILE pdf detected"; filemagic:"PDF document";
filestore; sid:3; rev: 1;)
```

Un'altra importante funzionalità è il calcolo, sempre on the fly, del MD5 checksum del file, anche detto fingerprint, che può essere confrontato con una lista di valori derivanti da file che si desidera tenere fuori o dentro la rete.

```
#Esempio meta file
TIME: 01/27/2010- 17:41:11.579196
PCAP PKT NUM: 2847035
SRC IP: 68.142.93.214
DST IP: 10.7.185.57
PROTP:6
SRC PORT:80
DST PORT: 56207
FILENAME:
/msdownload/update/software/defu/2010/01/mpasfe_7af9217bac55e4a6f71c989
231e424a9e3d9055b.exe
MAGIC: PE32+ executable for MS Windows (GUI) Mono/.Net assembly
STATE: CLOSED
SIZE:5204
```

3.2.1 All features

Dopo la panoramica generale sulle funzionalità e potenzialità del sistema, analizziamole più nello specifico. Sul sito ufficiale [16] è possibile consultare un elenco puntuale e sintetico di tutte le caratteristiche del sistema:

1. Engine
    a. Network Intrusion Detection System (NIDS) engine;
    b. Network Security Monitoring (NSM) engine;
    c. Analisi off line dei files PCAP;
    d. Memorizzazione del traffico con il pcap logger;
    e. Processing automatico dei PCAP file;
2. Compatibilità con i sistemi operativi: Linux, FreeBSD, OpenBSD, Mac OS X, Windows.
3. Configurazione:
    a. File YAML di configurazione – leggibile, ben commentato e documentato;
    b. Supporto per l'inclusione di altri files.
4. TCP/IP engines
    a. Flow engine scalabile;
    b. Supporto completo per IPv6;
    c. Tunnel decoding: Teredo, IP-IP, IP6-IP4, IP4-IP6, GRE;
    d. TCP stream engine: rilevazione di sessioni, ricostruzione del flusso;
    e. IP defrag engine.

5. Protocol parsers
    a. Supporto per la decodifica di pacchetti: IPv4, IPv6, TCP, UDP, SCTP, ICMPv4, ICMPv6, GRE, Ethernet, PPP, PPPoE, Raw, SLL, VLAN;
    b. Decodifica per il livello applicativo: HTTP, SSL, TLS, SMB, SMB2, DCERPC, SMTP, FTP, SSH, DNS.
6. HTTP engine
    a. Parse stateful;
    b. Logging delle richieste HTTP, identificazione, estrazione e logging dei file;
    c. Keyword per confronti normalizzati di: uri e raw uri, headers e raw headers, cookie, user- agent, corpo delle richieste e delle risposte, metodi, status code e host.
7. Detection engine
    a. Parole chiave per identificare il protocollo;
    b. Supporto PCRE;
    c. Regole su profili, algoritmi molto rapidi di pattern matching con utilizzo di acceleratori grafici;
    d. File matching: file magic, dimensione, nome ed estensione, controllo MD5;
8. Output
    a. HTTP request logging;
    b. TLS handshake logging;
    c. Unified2 output compatibile con Barnyard2;
    d. Alert fast log;
    e. Alert debug log, per gli sviluppatori di regole;
    f. Memorizzazione del traffico con pcap logger, Pcap infor, per l'integrazione in wireshark via suriwire;
    g. Supporto per Prelude;
    h. Drop log, in stile netfilter, per i pacchetti scartati in modalità IPS;
    i. Syslog;
    j. Memorizzazione di statistiche (con aggiornamenti periodici), file;
    k. DNS request/reply logger.
9. Packet acquisition:
    a. Metodi di cattura ad elevate prestazioni: AF_PACKET, PF_RING;
    b. Metodi di cattura standard: Pcap;
    c. In modalità IPS: netfilter basato su Linux;

   d. Schede di cattura e device specializzati: Endace, Napatech;

10. Multi threading:
    a. Runmode predefiniti;
    b. Possibilità di configurazione per: numero di thread (da uno a una dozzina), CPU affinity (per vincolare l'esecuzione del thread sempre sullo stesso core)
    c. Utilizzo di operazioni atomiche e locking per ottimizzare le risorse.

3.3 Architettura

Come precedentemente annunciato, viene qui approfondita la struttura multi threading utilizzata da Suricata. Infatti ciò merita una particolare attenzione poiché non solo è l'unico IDS che la utilizza, ma soprattutto perché permette di raggiungere ottime performance utilizzando quello che è il futuro dei calcolatori, ossia i processori multi core. Altri sistemi, per sfruttare tali processori, propongono soluzioni alternative (per esempio Snort multi-istanza) ma, mantenendo la loro struttura singole core, non riescono a sfruttarne a pieno le potenzialità. Per considerazioni quantitative si rimanda alla sezione 3.5 in cui verranno comparati su diversi aspetti i tre sistemi open source di maggiore successo: Suricata, Snort e Bro.

3.3.1 I processori multi core

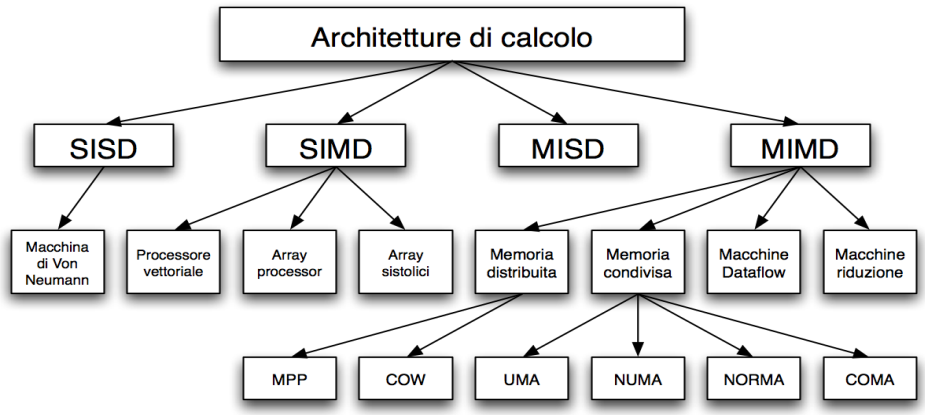

Figura 3.2: Famiglie processori multi core

La più semplice definizione di CPU multicore è "insieme d'unità processanti autonome in un unico chip che condividono risorse". Intel fu la prima, nel 2005, a commercializzare processori dual core (con i Pentium DSmithfield), la scelta fu obbligata in quanto non era più possibile aumentare la potenza di calcolo delle CPU single core incrementando unicamente la frequenza del clock. Più precisamente aumentare ulteriormente la frequenza implicava consumi troppo elevati, oltre i 100 W, quindi meccanismi di raffreddamento estremamente onerosi in rapporto all'incremento delle prestazioni. Il "salto evolutivo" poteva essere compiuto solo cambiando

mentalità, ossia puntando tutto sul parallelismo per permette di eseguire più operazioni in un unico ciclo di clock. Ovviamente tale paradigma è efficace solo se i programmi sono sviluppati e ottimizzati per un utilizzo multi-thread così da poter sfruttare appieno le caratteristiche di queste CPU. Di seguito si descriveranno brevemente, poiché non rappresentano il punto focale della tesi, le diverse architetture classificate da Flynn.

Architettura Multiple Instruction Single Data

Si hanno *n* processori, ciascuno con la propria unità di calcolo, che condividono la memoria. A ogni ciclo di clock il dato viene elaborato simultaneamente da tutti i processori, ciascuno in base all'istruzione ricevuta dalla propria unità di controllo. Ovviamente tale architettura è indicata se si eseguono istruzioni diverse sullo stesso dato.

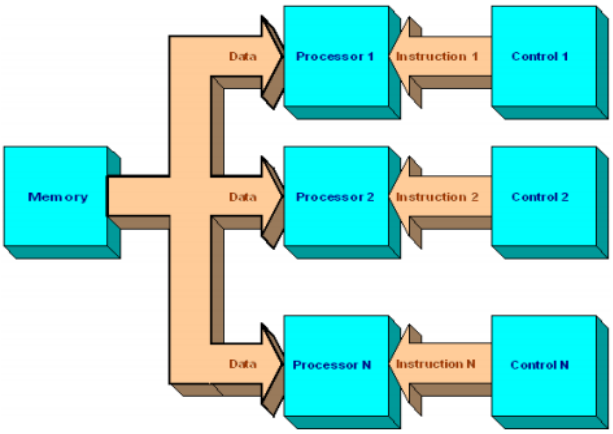

Figura 3.3: Architettura Multiple Instruction Single Data

Architettura Single Instruction Multiple Data

Si hanno *n* processori identici, ciascuno dotato della propria memoria locale, che lavorano sotto il controllo di un unico flusso di istruzioni. Ciò implica che a ogni ciclo di clock tutti i processori eseguono la stessa istruzione ma su dati diversi.

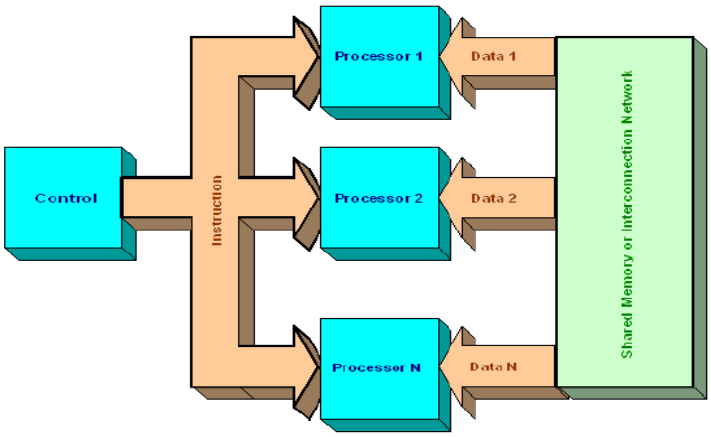

Figura 3.4: Architettura Single Instruction Multiple Data

Architettura Multiple Instruction Multiple Data

Questa è la classe più generale e potente nella classificazione di Flynn. Si hanno *n* processori, *n* flussi di istruzioni e *n* flussi di dati. Ogni processore possiede una memoria locale, quindi si possono eseguire potenzialmente programmi diversi su dati diversi.

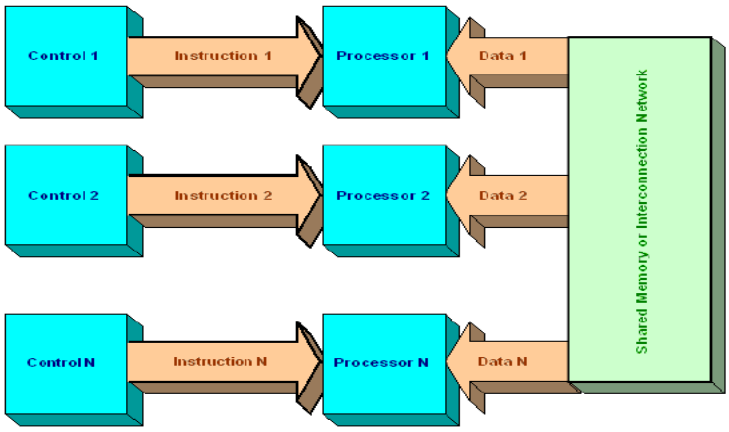

Figura 3.5: Architettura Multiple Instruciton Multiple Data

3.3.2 La pipeline

In Suricata il processo di analisi è realizzato come una pipeline di thread, anche detti moduli. Ciascun modulo è dedicato a un preciso compito, quindi presi i dati da una coda di input li inserirà, dopo averli elaborati, in una coda di output che potrebbe essere quella di input per un altro task. La pipeline è composta da quattro moduli:

3.3.2.1 Capture Module

Questo modulo si occupa di catturare i pacchetti dalla rete e memorizzarli in un buffer che sarà utilizzato come input dal modulo di decode.

3.3.2.2 Decode and stream application layer Module

Questo modulo preleva i pacchetti dal buffer e li converte in una struttura comprensibile a Suricata. Su di essa compie, nell'ordine: *stream-tracking*, per controllare che siano stati eseguiti tutti i passi per una corretta connessione di rete, *stream-assembly*, per ricostruire il flusso TCP che si riceve su più pacchetti e la ricostruzione del livello applicativo.

In Suricata un flusso, memorizzato in un'opportuna struttura dati, è definito dalla tupla *protocollo, sourceIP, destinationIP, sourcePort, destinationPort*. Tutti i pacchetti appartenenti a un flusso sono internamente collegati ad esso.

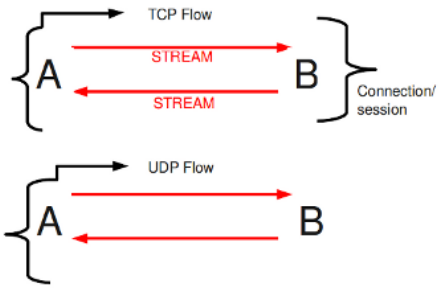

Figura 3.6: Flussi in Suricata

Suricata ha un significativo consumo di memoria in quanto memorizza tutti i dati relativi a un flusso per un tempo pari a *time-out*, impostabile sul file di configurazione. Inoltre a ogni nuova tupla, ossia a ogni nuova connessione, creerà, con un'operazione molto onerosa, il relativo flusso. Un attaccante potrebbe sfruttare questa organizzazione a suo favore realizzando un DoS su Suricata, infatti gli "basterebbe" inviare tanti pacchetti con valori diversi di protocollo, IPaddress e porte, per forzare la creazione e memorizzazione di molti flussi e saturarlo. Per sopperire al problema Suricata utilizza il valore *prealloc*, configurabile nel file .yaml, che determina il numero massimo di flussi attivi, essi sono pre istanziati e inizializzati solo in presenza di una nuova connessione, così da limitare il problema della creazione e l'eccessiva memorizzazione. Inoltre in caso di saturazione dei flussi, il sistema passa automaticamente alla modalità *emergency*, ossia lo stato di emergenza in cui si riduce il valore della variabile time-out così da chiudere più velocemente i flussi attivi lasciando spazio per quelli nuovi. Il modulo stream utilizza come input il flusso, memorizzandone i pacchetti fino a quando non sono riassemblati correttamente, per esempio dopo aver ricevuto l'ack, solo a questo punto possono essere inoltrati al modulo che procederà con l'analisi delle regole.

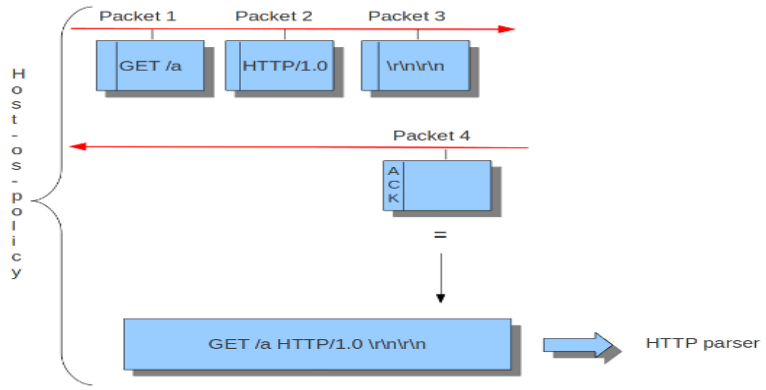

Figura 3.7: Ricostruzione stream in Suricata

3.3.2.3 Detect Module

Questo modulo realizza il pattern matching con le regole ed è il cuore del sistema. Essendo il task più oneroso del processo, richiede almeno il 70% del tempo totale di attraversamento del pacchetto, è oggetto di continue ricerche orientate a ideare soluzioni più efficienti.

Per migliorare le performance, l'analisi si compone di due fasi: il *pre-filter*, che filtra il traffico buono da quello cattivo, e il *Verification Unit*, che analizza più in profondità tutti i pacchetti sospetti. Tale suddivisione nasce da una semplice considerazione: non tutto il traffico è malevolo, anzi gli attacchi sono relativamente rari.

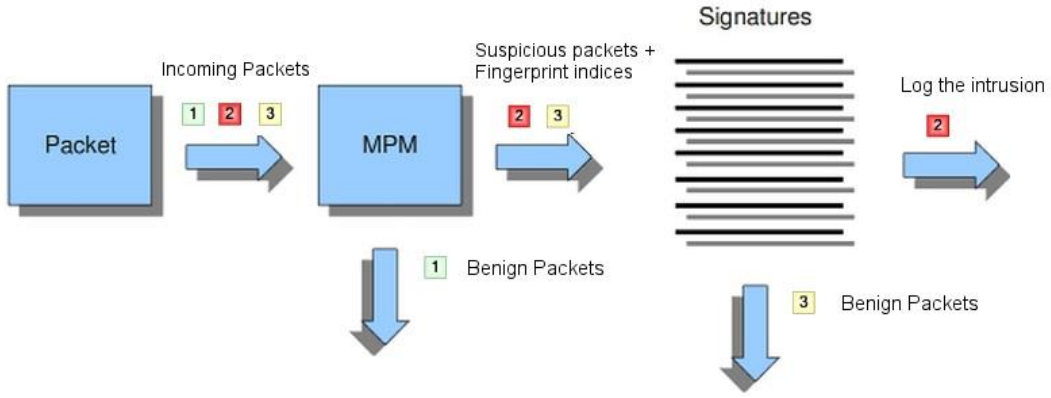

Figura 3.8: Modulo di detection in Suricata

Il pre-filtering è realizzato con l'algoritmo Multiple Pattern Matcher, infatti ogni regola è rappresentata da più *fingerprint*, ossia piccole parti del pattern complessivo. L'algoritmo garantisce il rilevamento di ogni pacchetto che verifica almeno una regola, quindi non vi sono falsi negativi ma con alta probabilità dei falsi positivi. Facendo riferimento alla figura, il pacchetto #1 non ha riscontri durante l'esecuzione del MPM ed è quindi sicuramente buono; viceversa i pacchetti #2 e #3 sono potenzialmente malevoli quindi necessitano di una analisi più approfondita. Questo passaggio iniziale permette di migliorare le performance, infatti avendo pattern di dimensione ridotta compiere il confronto con il pacchetto è molto semplice e rapido e si hanno due vantaggi: se il pacchetto è buono si evita il passaggio successivo e si diminuisce il numero di regole da testare nella validate unit. Infatti quest'ultima riceve il pacchetto e il fingerprint che ha avuto un riscontro andando ad analizzare nel dettaglio solo le regole che possono essere così rappresentate. Sempre nell'esempio, dopo una profonda analisi si è dedotto che il pacchetto #3 era un falso positivo, mentre il #2 faceva effettivamente parte di un'attività malevola.

3.3.2.4 Output Module

Questo modulo si occupa di processare tutte le allerte delle regole che hanno avuto un riscontro ed eventi che si sono realizzati. Queste informazioni possono essere espresse e memorizzate in moltissimi formati, dal file al database (la lista completa è consultabile al paragrafo 3.2.1 voce output). Ovviamente i log, in qualsiasi forma, occupano memoria quindi è fondamentale disabilitare tutti i formati che non vengono utilizzati.

Esistono diversi formati:

- Line based alerts log (fast.log): ogni alert è descritto da una riga, per questo è anche detto fast log;

```
10/05/10-10:08:59.667372  [**] [1:2009187:4] ET WEB_CLIENT
ACTIVEX iDefense  COMRaider ActiveX Control Arbitrary File
Deletion [**] [Classification: Web  Application Attack] [Priority: 3]
{TCP} x2.x2.232.144:80 -> 192.168.1.4:56068
```

- Log output per utilizzare Barnyard [51] (unified.log): il file di log è memorizzato in formato binario sul disco e viene successivamente elaborato da Barnyard. Quest'ultimo accede al file generato da Suricata, ne memorizza il contenuto su un database e lo cancella da disco.
- Line based log of HTTP requests (http.log): Sono qui memorizzati solo i dati relativi al traffico HTTP: Ogni riga contiene la richiesta, il nome dell'host, l'URI e lo user agent;

```
09/09/10-10:06:45:20.322315 www.nu.nl [**] / [**] Mozilla/5.0 (X11;
U; Linux i686; en-US; rv:1.9.2.9) Gecko/20100825 Ubuntu/9.10
(karmic)Firefox/3.6.9 [**] 192.168.1.4: 34138 -> 62.69.179.197:80
```

- Packet log (log.pcap): Con questa opzione è possibile tracciare e memorizzare tutti i pacchetti che attraversano il sensore, a prescindere che abbiano avuto o meno un riscontro con le regole, permettendo l'analisi del traffico in un secondo momento. Di norma non vengono salvati i pacchetti TCP che formano file di dimensioni superiori alla costante stream.reassembly.depth o flussi di pacchetti cifrati dopo lo scambio della chiave;
- Alert output to Prelude: Riprendendo quanto scritto nel paragrafo dedicato all'IDS Prelude, esso aggrega insieme, in un database, il traffico proveniente da più sensori. In caso di alert si devono inviare tre informazioni: la descrizione (il nome del sensore, la data, ecc…), l'header dei diversi protocolli (praticamente tutto IP, parte di UDP/TCP, ecc…) e il codice binario dell'intero pacchetto. Poiché queste ultime due parti sono

particolarmente onerose dal punto di vista della memoria, si può impostare Suricata in modo che non vengano inviate;

- Stats (stats.log): Su questo file di log sono scritti i valori di vari contatori interni all'engine. Si utilizza essenzialmente per ricavare statistiche o per conoscere dettagli del flusso di rete che attraversa il sistema.

Counter	TM Name	Value
flow_mgr.closed_pruned	FlowManagerThread	154033
flow_mgr.new_pruned	FlowManagerThread	67800
flow_mgr.est_pruned	FlowManagerThread	100921
flow.memuse	FlowManagerThread	6557568
flow.spare	FlowManagerThread	10002
flow.emerg_mode_entered	FlowManagerThread	0
flow.emerg_mode_over	FlowManagerThread	0
decoder.pkts	RxPcapem21	450001754
decoder.bytes	RxPcapem21	409520714250
decoder.ipv4	RxPcapem21	449584047
decoder.ipv6	RxPcapem21	9212
decoder.ethernet	RxPcapem21	450001754
decoder.raw	RxPcapem21	0
decoder.sll	RxPcapem21	0
decoder.tcp	RxPcapem21	448124337
decoder.udp	RxPcapem21	542040
decoder.sctp	RxPcapem21	0
decoder.icmpv4	RxPcapem21	82292
decoder.icmpv6	RxPcapem21	9164
decoder.ppp	RxPcapem21	0
decoder.pppoe	RxPcapem21	0
decoder.gre	RxPcapem21	0
decoder.vlan	RxPcapem21	0
decoder.avg_pkt_size	RxPcapem21	910
decoder.max_pkt_size	RxPcapem21	1514
defrag.ipv4.fragments	RxPcapem21	4
defrag.ipv4.reassembled	RxPcapem21	1
defrag.ipv4.timeouts	RxPcapem21	0
defrag.ipv6.fragments	RxPcapem21	0
defrag.ipv6.reassembled	RxPcapem21	0
defrag.ipv6.timeouts	RxPcapem21	0
tcp.sessions	Detect	41184
tcp.ssn_memcap_drop	Detect	0
tcp.pseudo	Detect	2087
tcp.invalid_checksum	Detect	8358
tcp.no_flow	Detect	0
tcp.reused_ssn	Detect	11
tcp.memuse	Detect	36175872
tcp.syn	Detect	85902
tcp.synack	Detect	83385
tcp.rst	Detect	84326
tcp.segment_memcap_drop	Detect	0
tcp.stream_depth_reached	Detect	109
tcp.reassembly_memuse	Detect	67755264
tcp.reassembly_gap	Detect	789
detect.alert	Detect	14721

Figura 3.9: Esempio stats.log di Suricata

- Syslog: Il funzionamento di questo tipo di log è del tutto analogo a quello di tipo fast, solo che l'alert è inviato al file di log del sistema;
- Drop.log: Memorizza i pacchetti scartati.

3.3.3 Esecuzione dei thread

Il multithreading è sicuramente l'elemento innovativo di Suricata, quindi tutte le opzioni che lo riguardano meritano un approfondimento. Si analizzerà nell'ordine: modalità di running, ossia l'organizzazione dei task, affinity e thread ratio. Per ottimizzare le performance del sistema è essenziale impostarli correttamente considerando il device che cattura il traffico, l'architettura del processore/i e la modalità operativa (IDS/IPS).

3.3.3.1 Running mode: auto

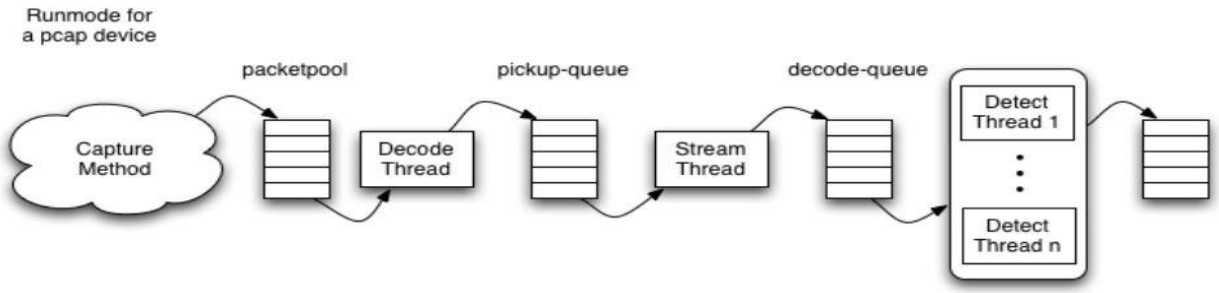

Figura 3.10: Running mode Auto

In questa modalità, come si può notare in figura, vengono creati più thread: uno per l'acquisizione dei pacchetti, uno per il decode, uno per lo stream e *n* per il detection, in quanto i primi due terminano la loro esecuzione in tempi molto più brevi del modulo di detect. Con questa soluzione si ha un overhead per la gestione delle code, ma è la più efficiente per alcuni metodi di cattura. Per esempio è indicata per i Pcap file, infatti in questo caso l'input del modulo di acquisizione è un unico file e quindi non possono esserci più thread che vi accedono. Inoltre, per quanto *n* sia un parametro configurabile nel file suricata.yaml, non è indicato inserire un numero troppo elevato, perché si perderebbero i vantaggi del parallelismo.

3.3.3.2 Runnig mode: workers

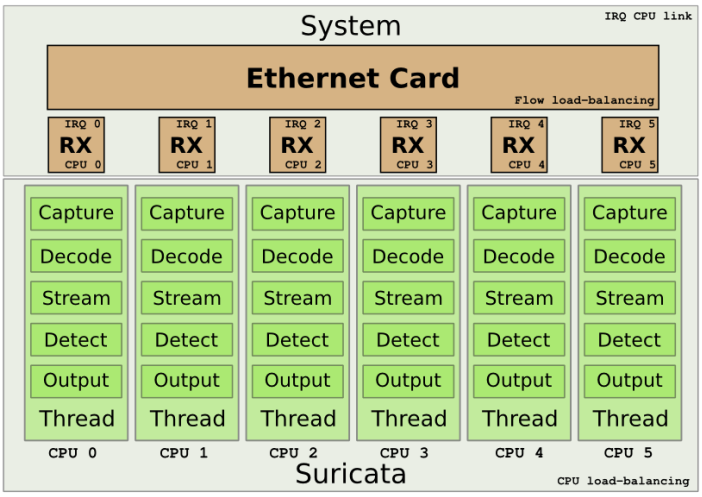

Figura 3.11: Running mode Workers

In questa modalità, come si può notare in figura, si crea un thread, che esegue l'intera pipeline, per ogni core. Questa organizzazione, elimina l'overhead delle code (che incide circa il 10% sul tempo totale) ma può essere utilizzato solo per metodi di cattura che permettono una suddivisione dei flussi, come per esempio le Network Interface Controllers ad elevate prestazioni. Queste schede utilizzano diverse tecnologie [17], le cui idee alla base sono:

- Creare all'interno della scheda di rete code indipendenti di pacchetti;
- Assegnare in modo univoco ogni pacchetto ricevuto alla coda corretta;
- Assegnare ad ogni core di sistema (o macchina virtuale) la coda di pacchetti corretta;
- Facilitare l'interazione fra il sistema, le code e i vari core.

3.3.3.3 Affinity e thread ratio

Suricata utilizza le ormai diffusissime CPU multi-core per analizzare simultaneamente più pacchetti. Sorgono spontanee due domande: dato un thread su quale core si eseguirà? Quanti thread di detection verranno utilizzati?

Per quanto riguarda la prima domanda, la risposta dipende dall'opzione *cpu_affinity*. Di default il sistema operativo cerca di bilanciare il carico computazionale, ossia di minimizzare il tempo di inattività di ciascun core. Di conseguenza un thread non verrà eseguito sempre dallo stesso core ma da quello disponibile in quel momento. Modificando tale opzione da *no* a *yes* è possibile "forzare" l'esecuzione di un thread sempre sullo stesso core, con l'evidente vantaggio di evitare la comunicazione intra-core. Inoltre Suricata permette di definire altre importanti proprietà del singolo thread come la priorità e su quali core eseguirlo. Vi sono più esempi in cui questa seconda configurazione permette di migliorare le performance, per esempio la modalità workers la utilizza in automatico. Per la modalità auto una configurazione "sensata" potrebbe essere quella in figura:

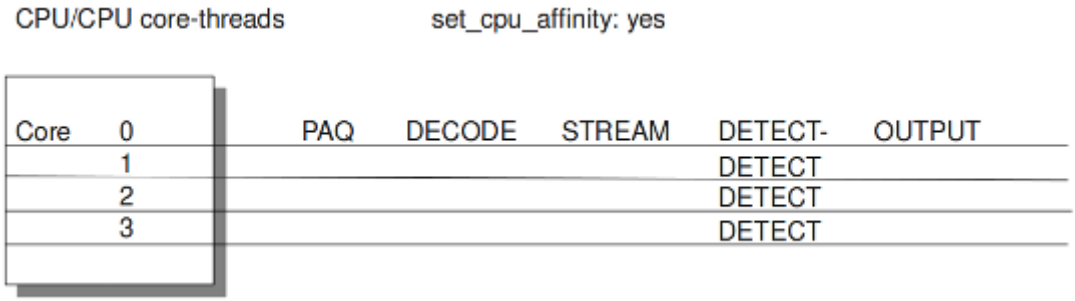

Figura 3.12: CPU affinity in Suricata

Quindi, in qualsiasi istante, il core #0 eseguirà un thread tra: acquisizione del pacchetto, decodifica, ricostruzione dello stream, analisi o output; i restanti processeranno il thread di detection. Considerando che il modulo di analisi è quello più oneroso, per bilanciare il carico computazionale di ciascun core, si può impostare al relativo thread sul core #0 la priorità minima. In questo modo sul core #0 verranno analizzati pochissimi pacchetti rispetto agli altri core e quindi potranno essere eseguiti gli altri moduli senza degradare le performance.

Per quanto riguarda la seconda domanda, ossia il numero di thread dedicati all'analisi, la risposta è data dall'opzione *detect-thread-ratio*. Nel dettaglio, il numero di thread dedicato all'analisi è il risultato della moltiplicazione tra tale valore e il numero di core/CPUs a disposizione. La scelta di questo numero è molto importante in quanto influenza notevolmente le performance del sistema, infatti, se troppo basso, si analizzeranno pochi pacchetti in contemporanea e non si sfrutterà la potenza di un'architettura multi-core, viceversa si avrà un significativo overhead causato dal frequente content switch e dall' accesso concorrente alle risorse condivise come, per esempio, la coda dei pacchetti.

3.4 Le regole

Suricata è volontariamente compatibile con il formato delle regole di Snort, spiegato in 2.5.5.1, infatti l'obiettivo della scelta è di facilitare il passaggio dall'attuale standard de facto a questo sistema, che si propone come futuro standard. Ciò significa che tutte le regole per Snort possono essere utilizzate da Suricata, ma non è vero il contrario. Infatti, come già accennato nel paragrafo 3.2, vi sono nuove parole chiave che non solo facilitano la scrittura delle regole, ma ne aumentano anche la capacità espressiva, si rimanda a 3.4.2 per un esempio. Per non essere ripetitivi riprendiamo solo alcuni concetti importanti: le azioni e le variabili.

L'azione, ossia come si deve comportare il sistema in caso di match di una regola, è forse l'opzione più importante. Riassumendo brevemente ciascuna azione:

1. Pass: Si interrompe l'analisi del pacchetto, cioè non si testano le regole successive;
2. Drop: Tale azione è utilizzabile solo in modalità IPS/inline e, in caso di riscontro, si interrompe l'analisi e il pacchetto non viene propagato al destinatario. Quest'ultimo non riceve alcun messaggio su ciò che sta accadendo quindi la connessione probabilmente andrà in time-out, evento certo per il TCP. L'amministratore può accorgersi della motivazione del time-out in quanto Suricata non solo scarta il pacchetto ma genera anche un messaggio di alert.
3. Reject: Come sopra, ma, per evitare il problema del time-out, viene inviato sia al mittente che al destinatario un pacchetto di reject: *Reset-packet*, se il pacchetto utilizza il protocollo TCP, *ICMP-error packet* altrimenti.
4. Alert: Il pacchetto verrà trattato come quelli "buoni", ossia terminata l'analisi sarà inviato sull'interfaccia di uscita ma verrà generato un messaggio di allerta visibile solo all'amministratore.

Ciascuna azione ha un livello di priorità, di default, in ordine decrescente: pass, drop, reject e alert. Ciò implica che, a prescindere dall'ordine in cui sono scritte sul file .rules, verranno testate

prima tutte le regole pass, poi le drop, e così via. Ovviamente è possibile modificare tale ordine cambiando le priorità di ciascuna azione nel file di configurazione.

Infine, così come in Snort, è possibile definire delle variabili il cui valore sarà sostituito solo in fase di caricamento. Possiamo distinguere tra IP-Address groups e port groups:

```
address-groups:
HOME_NET: “[192.168.0.0/16, 10.0.0.0/8, 172.16.0.0/12]”
EXTERNAL_NET: any
HTTP_SERVERS: “$HOME_NET”
SMTP_SERVERS: “$HOME_NET”
SQL_SERVERS: “$HOME_NET”
DNS_SERVERS: “HOME_NET”
TELNET_SERVERS: “HOME_NET”
AIM_SERVERS: any
port-groups:
HTTP_PORTS: “80”
SHELLCODE_PORTS: “!80”
ORACLE_PORTS: “1521”
SSH_PORTS: “22”
```

Quindi data la regola:

```
alert http $HOME_NET any -> any $HTTP_PORTS (msg:”SURICATA!”;
content.”suricata”; nocase; classtype:notsuspicious; rev:1;)
```

Essa genererà un messaggio di errore se si osserverà un pacchetto del protocollo HTTP avente nel body la parola suricata, a prescindere che sia minuscola o maiuscola, sourceIP del dominio 192.168.0.0/16, 10.0.0.0/8 o 172.16.0.0/12, sourcePort e destinationIP qualsiasi e destinationPort pari a 80.

3.4.1 L’organizzazione delle regole

Il software viene rilasciato già fornito di una serie di regole, scritte da esperti del ramo: Emerging Threats e Vulnerability Research (gli stessi di Snort).

I file *.rules*, sono collezioni di regole generalmente organizzate secondo classi di appartenenza nel settore della sicurezza informatica, per esempio *backdoor.rules*, *bad-traffic.rules*, *chat.rules*, *ddos.rules*, ecc… All’interno del singolo file possono esserci una o più regole; qualora si volessero disabilitare alcune di esse basterebbe anteporre il simbolo # che commenta ciò che segue. Le regole sono caricate solo all’avvio del sistema, di conseguenza se si cambiasse in runtime l’insieme delle regole, per esempio inserendone di nuove o abilitando/disabilitando quelle esistenti, le modifiche non sarebbero applicabili fino al successivo avvio del software.

Per ovviare a ciò, dalla versione 13dev, è possibile abilitare la funzionalità *Live Rules Swap* che forza il (ri)caricamento delle regole mentre Suricata è in esecuzione.

```
# When rulereload is enabled, sending a USR2 signal to the Suricata process
# will trigger a live rule reload. Experimental feature, use with care.
Rulereload:true
```

All'avvio si inizializza una struttura ad albero ideata per limitare la latenza del thread di detection nel recupero delle regole e il tempo di esecuzione del pattern matching. L'idea alla base è semplice: il sistema ha un numero estremamente elevato di regole, ma ciascuna di esse è dotata di una serie di attributi che un pacchetto non potrà mai verificare in contemporanea, per esempio se il pacchetto ricevuto utilizzasse il protocollo UDP sicuramente non sarebbero necessarie tutte le regole con chiave TCP.

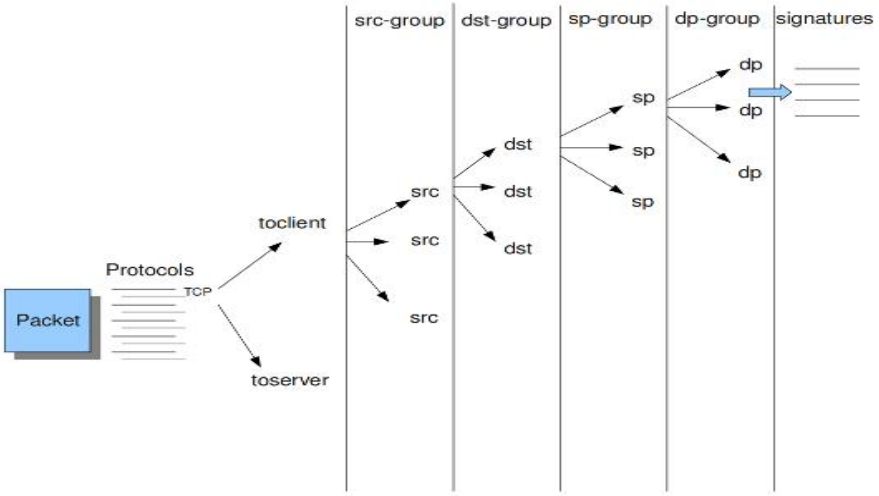

Figura 3.13: Albero delle regole di Suricata

Suricata permette quindi di raggruppare le regole in sottogruppi interni, indentificati da sei parametri: protocollo, direzione flusso, sourceIP, destinationIP, sourcePort e destinationPort. In questo modo, visitando l'albero, è possibile individuare rapidamente l'insieme di regole inerenti al pacchetto corrente e, data la dimensione ridotta del ruleset, l'elaborazione sarà più rapida, sia perché si avranno meno regole da testare, sia perché sarà meno probabile avere un falso positivo nella fase di pre-filtering. Questa tecnica può però risultare una lama a doppio taglio, infatti tanto più si partiziona, tanto maggiore sarà l'occupazione in memoria derivante da tutti i puntatori necessari per il recupero dell'informazione. La quantità di sottogruppi dovrà essere un ragionevole compromesso tra occupazione in memoria e performance.

Suricata aiuta l'utente in questa scelta, ossia nel file di configurazione è possibile indicare il comportamento desiderato ad alto livello: *high* se sono più importanti le performance della memoria, *low* per il caso duale e *medium* per cercare di bilanciare il più possibile i due aspetti.

```
Detect-engine:
-profile:medium          #The balance between performance and memory usage.
```

In alternativa è sempre possibile definire manualmente il numero di sottogruppi per ciascun parametro. Facendo riferimento all'immagine 3.13, si può ottenere ciò impostando nel file di configurazione:

```
-custum-values:
toclient_src_groups: 3
toclient_dst_groups: 3
toclient_sp_groups: 3
toclient_dp_groups: 3
```

3.4.2 Caso di studio

Supponiamo di voler ricevere un messaggio di allerta in caso di una richiesta HTTP con metodo GET e valore del parametro *"./download.php"*.

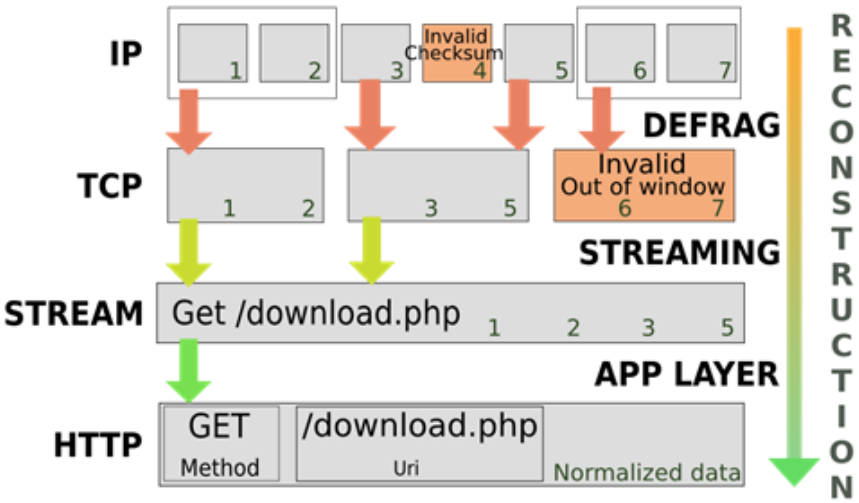

Figura 3.14: Esempio comportamento Suricata

Facendo riferimento alla situazione in figura, è evidente il funzionamento di Suricata per protocolli noti, ossia ricostruisce lo stream, normalizza i dati e controlla il valore dei campi specificati dalla regola.

Nello specifico:

1. Si osservano sette pacchetti che vengono riassemblati dal defrag engine. Il pacchetto #4 con il suo checksum errato genererà un messaggio di allerta in quanto avrà un riscontro con la regola presente nel *decoder-events.rules*

```
alert ip any any -> any any (msg:"SURICATA IPv4 invalid checksum";
ipv4 –csum:invalid; sid:2200073; rev:1;)
```

2. Inoltre, prima di passare i pacchetti a livello di stream, si controlla che non ve ne siano fuori dalla finestra TCP (cosa che accade per #6 e #7), con la regola definita nel *stream-events.rules*

```
alert tcp any any -> any any (msg:"SURICATA STREAM
ESTABLISHED packet out of window"; stream-
event:est_packet_out_of_window; sid:2210020; rev:1;)
```

3. A livello HTTP si normalizzano i dati dello stream (#1,#2, #3 e #5) così da evitare manipolazioni dell'attaccante per eludere la regola:

```
alert http any any -> any any (msg:"Download"; content:"GET";
http_method; content:"/download.php"; http_uri)
```

Per ottenere lo stesso risultato, su altri sistemi, si potevano utilizzare le seguenti regole, entrambe però "facilmente" aggirabili da un attaccante esperto.

```
alert tcp-pkt any any -> any 80 (msg:"HTTP dl"; content:"Get /download.php";
sid:1; rev: 1;)
```

Questa regola può essere usata come pre-matching poiché lancerà un allarme solo se la richiesta non è divisa in più pacchetti.

```
alert tcp any any -> any 80 (msg:"HTTP download";
flow:established,to_server; content:"Get /download.php";)
```

Anche questa può essere bypassata in quanto è case sensitive e basta manipolare opportunamente la richiesta, per esempio aggiungendo uno spazio tra la GET e l'URI per evitare il riscontro.

3.5 Confronto con Snort

Suricata ha diverse caratteristiche che lo contraddistinguono dagli altri sistemi di intrusion detection, primo tra tutti l'utilizzo del multi threading che permette di raggiungere velocità mai viste prima. Infatti, senza essere ripetitivi, questa scelta si sta dimostrando sempre più vincente ed è attualmente l'unico sistema open source che gestisce velocità dell'ordine dei 10 Gbps [18]. Per questo e altri motivi, Suricata si sta diffondendo sempre più, tanto da essere definito da alcuni come successore di Snort e futuro standard de facto. Quest'interesse crescente per il sistema ha portato a diversi studi che mettono a confronto sotto vari aspetti i due sistemi sopracitati.

Parametro	**SNORT**	**Suricata**
Regole	• VRT::SNORT rules • EmergingThreats rules	• VRT::SNORT rules • SO rules • EmergingThreats rules
Thread	Single thread	Multi thread
Installazione	Tramite pacchetto	Per lo più manuale
Log degli eventi	File, Unified2 per Barnyard2, database	
Documentazione	Molte risorse	Poche risorse
Analisi offline	Si	Si
File di configurazione	File .conf	File YAML
Linguaggio di programmazione	C	C
Sistema operativo	Multipiattaforma	Multipiattaforma
Supporto IPv6	Si	Si
Sviluppatore	Sourcefire, Inc.	OISF
In sviluppo da	1998	2009

Tabella 3.1: Confronto tra Snort e Suricata

Nella tesi [22] si è confrontata un'istanza di Suricata con quella di Snort, utilizzando un Intel Processor E5630 a 2.4 GHz quad cores 8 threads (Hyper-Threading enalble)[6], concludendo che Suricata consuma più CPU e memoria ma ha una scalabilità molto maggiore. Infatti Snort, essendo single thread, non trae alcun vantaggio dall'architettura multicore quindi utilizzerà al massimo interamente il core su cui viene eseguito, il 12.5% in figura 3.15. Viceversa Suricata, con la sua struttura multi thread, avrà un consumo crescente con l'aumento della velocità, in quanto i moduli di analisi lavoreranno in parallelo e quindi si avranno più core attivi in contemporanea. A questa maggiore utilizzazione corrisponde però una scalabilità molto superiore a Snort, infatti la percentuale di pacchetti scartati è inferiore rispetto a quest'ultimo e

[6] Tecnologia Intel per migliorare le prestazioni dei propri processori, in questo caso il sistema operativo ha a disposizione 4 core fisici ma 8 core logici.

il divario tra i due aumenta significativamente con il crescere della velocità, come illustrato in tabella 3.2.

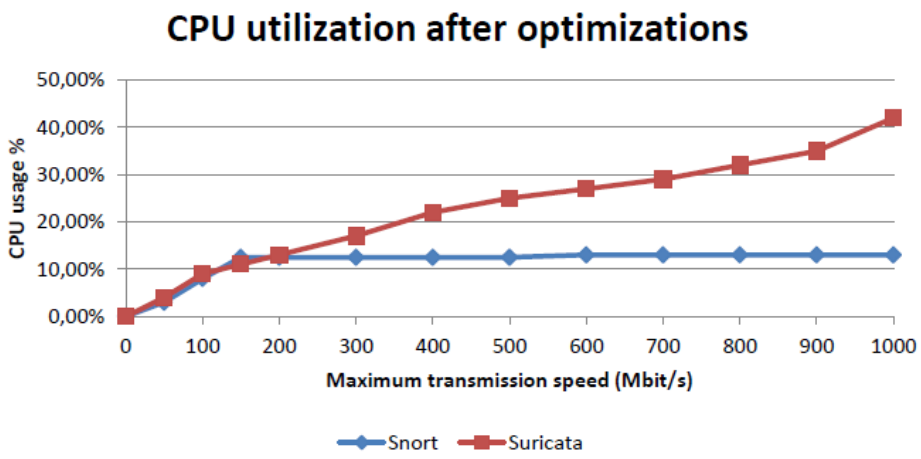

Figura 3.15: Consumo CPU Snort singola istanza e Suricata

	50M	100M	150M	200M	300M	400M	500M	600M	700M	800M	900M	1000M
Snort	0.00%	0.02%	0.65%	1.97%	5.49%	11.31%	18.59%	22.31%	29.56%	36.02%	42.27%	(58.36%)
Suricata	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.01%	0.01%	0.02%	0.02%	(1.84%)

Tabella 3.2: Percentuale pacchetti scartati da Snort e Suricata

Per limitare il problema di Snort sulle architetture multi core, ossia per sfruttare almeno in parte la maggiore potenza di calcolo a disposizione, gli ideatori del sistema suggeriscono di mandare in esecuzione più istanze dello stesso, una per ciascun core.

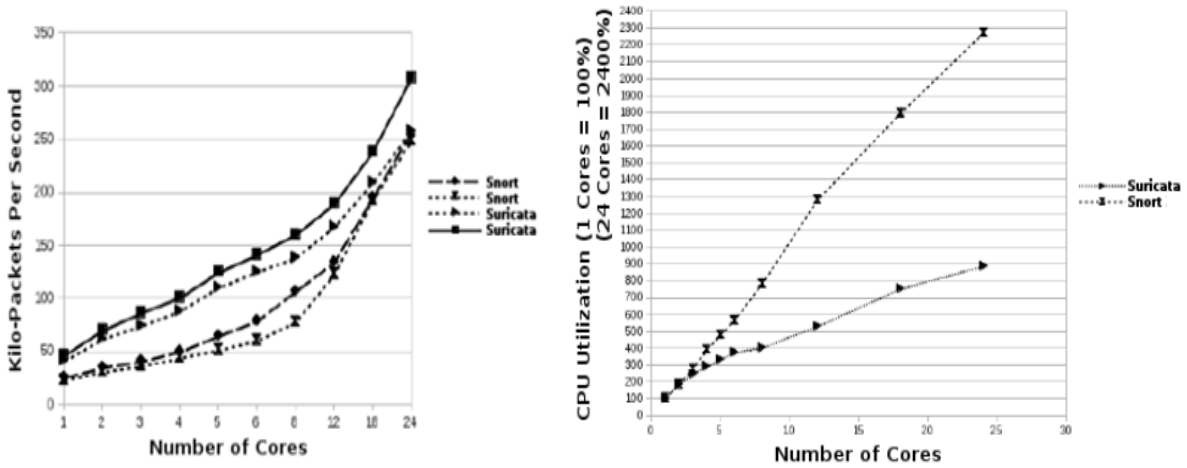

Figura 3.16: Consumo CPU e velocità di trasmissione di Snort multi istanza e Suricata

Facendo riferimento all'articolo [13], Suricata continua a risultare la scelta vincente, sia per le velocità di trasmissione gestite che per utilizzazione delle risorse.

Capitolo 4

Formalizzazione del modello di analisi distribuita

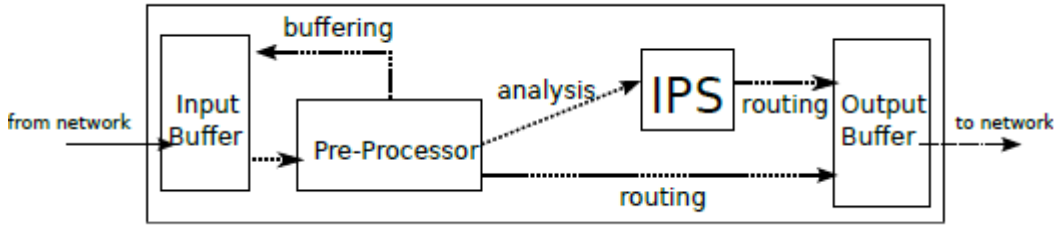

Figura 4.1: Architettura router con preprocessor

Come abbiamo illustrato nei capitoli precedenti esistono diversi Intrusion Prevention System che combinati ad altri sistemi di protezione, per esempio il firewall, realizzano le "difese" della rete locale, ossia impediscono che attacchi dall'esterno possano colpire gli host della rete interna. Questa soluzione ha però una validità "locale", ossia l'eventuale attività malevola viene rilevata solo alla soglia della sottorete o nodo di destinazione, quindi la rete di trasporto utilizzerà le proprie risorse per instradare un pacchetto che verrà scartato a destinazione. Da questa considerazione ha inizio il nostro studio: anticipare nella rete la scoperta di attività malevoli realizzando un meccanismo di analisi distribuita attraverso intrusion prevention system. Di conseguenza, a differenza di precedenti lavori, per esempio [6] e [21], non svilupperemo dei nuovi sistemi di IPS ma utilizzeremo quelli già disponibili combinandoli con le nostre strategie di ottimizzazione energetica per raggiungere i nostri obiettivi.

Il nostro fine principale è anticipare il più possibile la scoperta di pacchetti malevoli in ottica *green-aware*: l'idea è di evitare l'intera l'analisi nei soli nodi terminali ma anticiparne una parte, preferibilmente tutta, nella rete di trasporto così da diminuire i consumi energetici globali. Per realizzare tali propositi, è evidente che ogni nodo della rete può svolgere due diversi task: il routing e l'analisi del traffico con un qualsiasi IPS che consumano rispettivamente R e A. Alcune precisazioni sono doverose:

- Il routing deve essere sempre garantito, ciò implica che in caso di picchi di traffico si effettuerà l'instradamento tralasciando l'analisi dei pacchetti;
- L'energia del router non è "infinita", quindi si deve tener conto che ogni operazione ha un costo e può essere effettuata solo se possibile. Generalmente se il dispositivo ha dei pacchetti da elaborare, ma non ha risorse disponibili, questi sono "bufferizzati"[7].

[7] I pacchetti sono memorizzati ed elaborati appena possibile.

Infine è banale osservare, dato il vincolo espresso nel punto precedente, che l'energia a disposizione deve essere superiore o al minimo uguale al costo necessario all'instradamento del numero massimo di pacchetti gestibili dalla scheda;

- L'analisi può essere realizzata con un qualsiasi IPS ed è effettuata su una parte dei pacchetti che attraversano il nodo.

L'unico tassello mancante è un modulo che, applicando una determinata politica di gestione, smisti il traffico ricevuto dal nodo tra i due task. Senza scendere troppo in dettaglio, poiché sarà oggetto del prossimo paragrafo, ogni nodo analizzerà nell'intervallo T il massimo numero di pacchetti gestibili in quel contesto e i restanti, se vi sono ancora risorse disponibili, saranno inviati al nodo successivo. Concretamente, supponiamo che il pacchetto p sia parte di un attacco di A verso B e che il percorso A-B sia composto da n router, attualmente il consumo dell'invio del pacchetto p da A a B e successiva analisi è pari a $n \cdot R + A$; con il nostro modello, al contrario, supponendo che l'IPS del k-esimo nodo di tale cammino scarti p, considerandolo malevolo, si avrebbe un consumo pari a $k \cdot R + A$. In conclusione il nostro meccanismo di analisi distribuita soddisfa gli obiettivi che ci siamo preposti con risparmio energetico che cresce con la diminuzione del numero di nodi necessari per lo scarto delle attività malevole. Formalmente il nostro modello è tanto più conveniente tanto più:

$$k \cdot R + A \ll n \cdot R + A \quad (1)$$

4.1 I router

Internet è una rete a commutazione di messaggi composta principalmente da quattro dispositivi: hub, switch, router e gateway. Brevemente, poiché questo non è il tema principale della nostra tesi:

- Hub -> Privo di qualsiasi "intelligenza", quando riceve un segnale lo rigenera su tutte le porte di uscita, evitandone così l'attenuazione. Si può dire in termini molto semplicistici che è un ripetitore di segnale;
- Switch -> A differenza dell'hub, è dotato di "intelligenza", ossia quando riceve un segnale recupera l'informazione sull'indirizzo del destinatario e invia sulla opportuna interfaccia di uscita. Lo switch è quindi in grado di separare il traffico;
- Router -> A differenza dei precedenti, il router lavora a livello 3 del modello ISO-OSI collegando una rete di PC a un'altra che utilizza lo stesso protocollo. Quindi leggendo l'indirizzo IP di destinazione, invia il pacchetto sull'interfaccia di uscita che permette di seguire il percorso migliore;

- Gateway -> A differenza del router, gestisce il traffico dati tra reti non omogenee convertendo i messaggi da un protocollo ad un altro.

4.1.1 Il routing

Ogni router deve garantire tre funzionalità: la commutazione del traffico, il DNS (domain name system), l'aggiornamento e memorizzazione della tabella di routing. La tabella di routing indica, per ciascuna destinazione, la porta di uscita appartenente al percorso ottimo, ossia che minimizza la funzione di costo che può dipendere dal tempo e/o dalla distanza. Esse posso essere statiche, se configurate manualmente e "immutabili", o dinamiche, se si aggiornano automaticamente, con opportuni protocolli (RIP, OSPF e BGP), a seguito di un qualsiasi cambiamento nella topologia della rete. Oggi giorno moltissimi router forniscono servizi aggiuntivi, il firewall e il NAT per esempio, e il nostro modello, trattato nel prossimo paragrafo, fornisce un'ulteriore funzionalità che può portare a notevoli risparmi se applicato su tutti i dispositivi di una grande rete.

La propagazione del traffico, compito principale del router, merita un approfondimento.

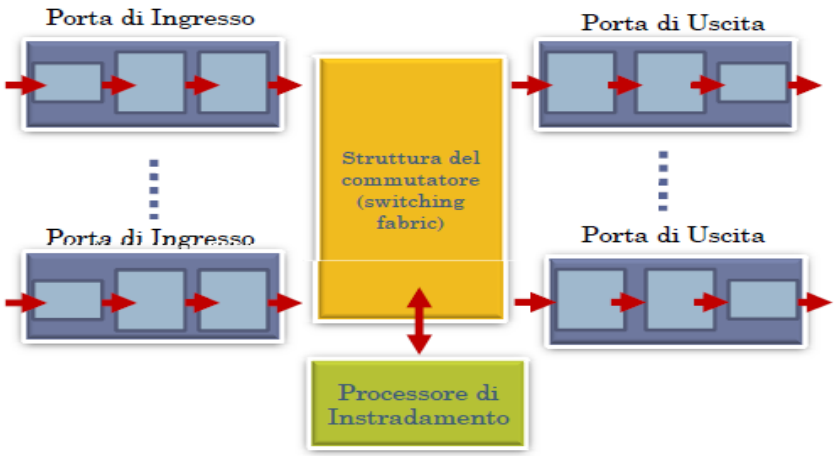

Figura 4.2: Architettura router

Brevemente, quando il router riceve un pacchetto su uno dei suoi link per prima cosa esegue l'operazione nota come *lookup*, ossia ricavato l'indirizzo di destinazione dall'header IP consulta la tabella di routing per determinare l'interfaccia di uscita. La ricerca segue la politica *longest pattern matching*, ossia il risultato sarà la porta di uscita relativa al prefisso di corrispondenza più lungo, in altri termini l'itinerario più specifico, fra tutti i prefissi che corrispondono all'indirizzo di destinazione. Una volta terminata la ricerca, ma prima di inserire il pacchetto sul buffer della relativa porta di uscita, si esegue lo *switching* che aggiorna l'header, per esempio decrementando il TTL o aggiornando il checksum.

Approfondendo passo per passo:

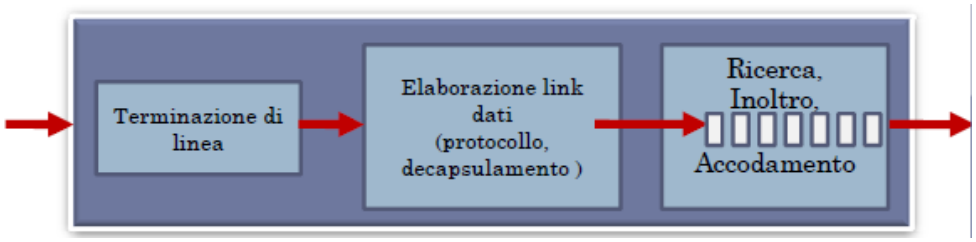

Figura 4.3: Diagramma funzionale interfaccia di ingresso

L'interfaccia di ingresso permette l'utilizzo del livello fisico e del link, ossia determina il segnale di linea della porta e ne interpreta le informazioni ricevute. Dopo questa semplice elaborazione il pacchetto viene inoltrato alla CPU che effettuerà la ricerca nella tabella di routing e lo trasferirà nell'opportuna interfaccia di uscita. Le schede di media fascia, dotate di una CPU locale e di una copia ombra della tabella di routing sulla memoria RAM, possono anticipare l'operazione di lookup. In questo caso il processore di instradamento potrà occuparsi unicamente del calcolo e mantenimento delle tabelle di routing ignorando la lookup di ogni pacchetto. I router di alta fascia, dotati di schede particolari che eseguono il processo di forwarding direttamente via hw, possono raggiungere prestazioni ancora più elevate. Ma quanto deve essere celere questa elaborazione? L'ottimo sarebbe una velocità lineare, ossia la ricerca deve essere terminata in un tempo inferiore all'inter-arrivo dei pacchetti, numeri alla mano, se l'interfaccia riceve 2.5 Gbps e i pacchetti sono lunghi 256 byte si devono eseguire circa un milione di ricerche al secondo.

Infine, terminata la ricerca, il commutatore si occupa di inserire il pacchetto nel buffer della porta di uscita. Aprendo una breve parantesi, il buffer può essere gestito con diverse politiche di scheduling e gestione dinamica della coda per il QoS, tale scelta è particolarmente rilevante se il commutatore ha una frequenza di invio pacchetti maggiore di quello della porta di uscita, ossia se i pacchetti si accodano necessariamente nel buffer. Per quanto riguarda la struttura del commutatore, esistono diverse soluzioni strettamente dipendenti dalla classe del router. In ordine crescente di efficienza:

- Commutazione attraverso la memoria

  Questo tipo di commutazione, presente nei router più semplici, avviene attraverso il controllo della CPU. Più nel dettaglio, quando una porta di ingresso riceve un pacchetto lo segnala, attraverso un interrupt, al processore di instradamento ed esso viene copiato dall'interfaccia di ingresso a quella di uscita attraverso la memoria. Se la velocità di

scrittura della memoria è B pacchetti al secondo, allora la velocità di elaborazione sarà B/2 pacchetti al secondo.

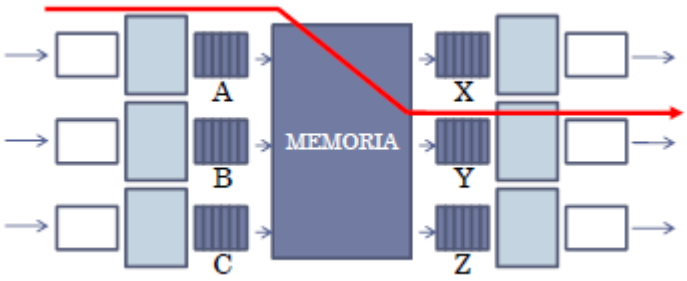

Figura 4.4: Commutazione pacchetto attraverso la memoria condivisa

- Commutazione per mezzo di un bus

  Il trasferimento del pacchetto non coinvolge il processore in quanto è la porta in ingresso a inviarlo direttamente all'interfaccia di uscita attraverso il bus condiviso. La velocità è ovviamente maggiore, rispetto alla soluzione precedente, ma comunque limitata alla velocità del bus.

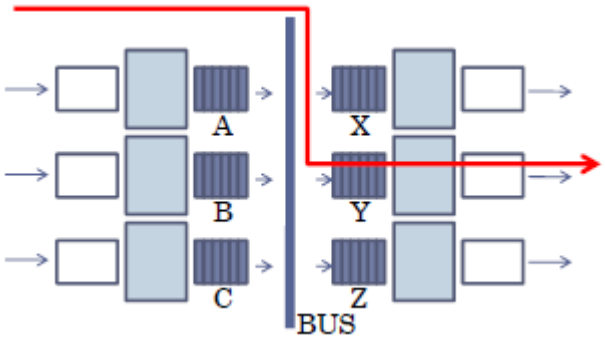

Figura 4.5: Commutazione pacchetto attraverso il bus

- Commutazione attraverso una rete intercollegata

  L'utilizzo del crossbar, una rete di interconnessione che consiste in 2N bus che connettono N porte di ingresso con N porte di uscita, è sicuramente la soluzione più efficiente. Infatti la probabilità che una porta in ingresso trovi il bus occupato, e quindi rimanga bloccata in attesa che si liberi, è significativamente più bassa.

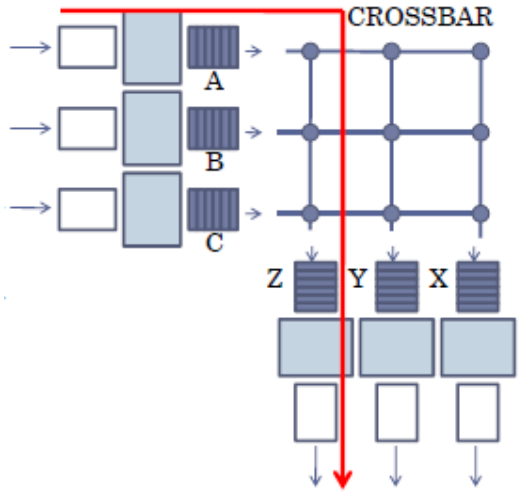

Figura 4.6: Commutazione pacchetto attraverso crossbar

Per concludere, da un punto di vista prestazionale, esistono potenzialmente tre colli di bottiglia:

- L'operazione di lookup: Una ricerca sequenziale è ormai inapplicabile, infatti data la dimensione delle tabelle l'operazione non potrebbe essere eseguita in tempi ragionevoli e il problema è destinato a diventare sempre più marcato a causa del continuo aumento del numero di utenti in Internet. Per limitare i tempi è possibile, per esempio, utilizzare una struttura ad albero in cui ciascun livello corrisponde a un bit dell'indirizzo. Quindi se l'indirizzo è composto da *n* bit, la ricerca è O(n).
- Packet buffering: Le code di pacchetti possono formarsi sia nelle porte di ingresso che di uscita. Esse non devono aumentare troppo in dimensione altrimenti lo spazio del buffer potrebbe esaurirsi portando alla perdita di pacchetti. Ipotizzando che le velocità delle *n* porte in ingresso e *n* di uscita siano le stesse, allora, per non avere accodamento, la velocità del processo deve essere *n* volte superiore alla velocità delle porte.
- Switching e dimensioni del pacchetto

4.2 Il Preprocessor

Il preprocessor, installato sul singolo router, è il modulo che realizza il sistema di analisi distribuita, infatti il suo scopo è quello di determinare quali pacchetti inoltrare all'instradamento e quali all'analisi. Più nello specifico, quando si riceve un pacchetto su un'interfaccia di rete, invece di invocare immediatamente la routine di lookup per risolvere la porta di uscita, si esegue il nostro preprocessor. Il funzionamento di questo componente è logicamente semplice: definito un periodo *T*, a ogni nuovo intervallo si determina, in funzione a quanto osservato, il numero massimo di pacchetti da inviare al processo di analisi, detto *maxPPSIPS*.

Per evitare che un pacchetto già classificato "buono"[8] sia nuovamente analizzato dai nodi successivi nel percorso ottimo verso la destinazione, si trasmette l'informazione del suo "stato di analisi": da analizzare o analizzato. Prendendo spunto dal precedente lavoro di un nostro collega, il TIFIPS [6], tale informazione è trasmessa nelle opzioni dell'header IP quindi è estremamente semplice da gestire.

Il preprocessor determina la politica da seguire in un quanto di tempo, ossia in un intervallo di durata T, utilizzando diverse variabili:

- maxPPSIPS, numero massimo di pacchetti che possono essere inoltrati all' IPS nel generico intervallo [kT,(k+1)T]. Tale valore non è statico ma aggiornato a ogni nuovo istante di "campionamento", applicando una delle formule descritte in 4.3;
- *#PktAnalyzed*, contatore dei pacchetti inviati all'IPS nel quanto, è banale osservare che

$$\#PktAnalyzed \leq maxPPSIPS \quad \forall\, [kT, (k+1)T] \quad (2)$$

- *#PktInT*, contatore dei pacchetti ricevuti nel quanto. Questo valore è utilizzato come feedback per "adattare" maxPPSIPS alle variazioni di traffico.
- Energia a disposizione per il routing e l'analisi nel quanto.

Definite le variabili, descriviamo il comportamento del modulo a seguito dei due eventi che lo attivano: l'arrivo di un pacchetto e lo scadere di T.

4.2.1 Arrivo di un pacchetto

Alla ricezione di un pacchetto sull'interfaccia, se vi sono risorse, ossia energia, il preprocessor lo elaborerà altrimenti sarà memorizzato e gestito appena possibile. Nel primo caso si recupera, dalle opzioni dell'header IP, l'informazione sullo stato: se il pacchetto è stato già analizzato lo si inoltra direttamente al processo di routing, altrimenti è un candidato all'analisi. Infatti non è detto che tutti i pacchetti ancora da analizzare siano controllati dal nodo corrente, tale scelta dipende dal carico dell'IPS e dalle risorse ancora a disposizione. Più precisamente la prima condizione, ossia #PktAnalyzed minore o uguale a maxPPSIPS, è necessaria per evitare di inviare all'IPS un numero di pacchetti superiore a quello teoricamente gestibile in *T*; la seconda, considerando che l'analisi del pacchetto è un'operazione molto più "onerosa" della lookup, evita un eccessivo consumo energetico. Se non si effettuassero tali controlli si andrebbe in contro ad un inevitabile sovraccarico del dispositivo con possibili conseguenze "catastrofiche", prima tra tutti il drop, ossia scarto, dei pacchetti. Riassumendo se il nodo riceve un pacchetto non ancora analizzato e le condizioni per l'analisi sono verificate esso verrà controllato, ed

[8] Si ricorda che i pacchetti classificati "cattivi", ossia parte di un'attività malevola, sono scartati.

eventualmente aggiornato nello stato[9] memorizzato nell' header IP, altrimenti sarà inoltrato direttamente al routing delegando il controllo a un nodo successivo nel percorso verso la destinazione.

4.2.2 Nuovo intervallo kT

Terminato un quanto si deve determinare la politica per il nuovo intervallo, essenzialmente il numero massimo di pacchetti da assegnare al task di analisi. Uno dei maggiori punti di forza del nostro modello è la dinamicità introdotta per tale scelta: maxPPSIPS non è costante, al contrario dipende da quanto si è osservato in precedenza ed è quindi strettamente legato alle effettive condizioni della rete. Infatti maxPPSIPS è calcolato con la formula:

$$maxPPSIPS = \left\lfloor (1 - load) \cdot \frac{T}{timePrcIPS} \cdot threshold \right\rfloor \quad (3)$$

Nel dettaglio la quantità load è definita dalla formula seguente:

$$load = \frac{\#PktEstimated}{maxPPS} \leq 1 \; \forall \; [kT, (k+1)T] \; (4)$$

Il numeratore, *#PktEstimated*, è il traffico stimato dal preprocessor, con una delle strategie trattate nel paragrafo 4.3, in ingresso alla scheda di rete per l'imminente intervallo. Al denominatore, invece, vi è la costante *maxPPS*, ossia il numero massimo di pacchetti supportato dall'interfaccia. Si evidenzia che la variabile load potrebbe essere calcolata come il rapporto tra il throughput stimato e quello massimo, ma ciò non incrementerebbe in modo significativo l'accuratezza. Infatti oggi giorno i pacchetti tendono ad avere tutti la stessa dimensione: essendo IP (a livello 3) e Ethernet (a livello 2) gli standard più utilizzati, la maggior parte delle applicazioni generano direttamente pacchetti di dimensione idonea. Si può quindi concludere che il risultato di load espresso in funzione del numero di pacchetti e quello in funzione dei throughput convergono allo stesso valore, anzi se tutti i pacchetti hanno dimensione pari alla MTU sono esattamente uguali:

$$load = \frac{MTU \cdot \#PktEstimated/_{T}}{MTU \cdot \#PktMax/_{T}} \; (5)$$

Nella formula (3) sono utilizzati anche *timePrcIPS* e *threshold:* il primo rappresenta il tempo medio di attraversamento di un pacchetto nell'IPS, il secondo, un numero compreso tra 0 e 1,

[9] L'aggiornamento dello stato da "da analizzare" a "analizzato" avviene solo per i pacchetti "buoni" in quanto quelli "cattivi", come già detto, vengono direttamente scartati.

la soglia di sicurezza che ammortizza tutte le eventuali approssimazioni e tutela dal caso pessimo, ossia tempi di analisi maggiori di quello medio.

Riassumendo, in ogni quanto di tempo, si "massimizzerà" il numero di pacchetti da analizzare, ma, allo stesso tempo, è garantito che in condizioni di "stress", ossia #PktEstimated circa pari a maxPPS, il numero di pacchetti destinato all'analisi sarà praticamente nullo, quindi si privilegerà il routing a discapito dell'analisi. Questo comportamento è del tutto desiderabile in quanto, come già sottolineato, l'instradamento deve essere sempre garantito e si desidera anticipare il più possibile il rilevamento di attività malevoli.

Per concludere è importante sottolineare che la nostra soluzione può essere applicata a un router di qualsiasi fascia. Infatti se il router dispone di un'unica CPU centralizzata, ossia è di bassa fascia, il termine maxPPS nella formula (3) sarà la somma dei relativi valori di ciascuna interfaccia. Più precisamente, date *n* schede di rete e definito maxPPS_i il massimo numero di pacchetti gestibili dall' i-esima interfaccia, si ha che:

$$maxPPS = \sum_{i=1}^{n} maxPPS_i \quad (6)$$

Invece, per router di media-alta fascia, il preprocessor sarebbe posizionato su ciascuna interfaccia. Quindi supponendo di avere *n* porte, se non si modifica la formula (3) è evidente che non sarebbero analizzati maxPPSIPS pacchetti ma n volte la suddetta quantità, quindi un numero molto maggiore rispetto a quello teoricamente gestibile dal sistema. Ovviare a questo problema è "semplice", assumendo che il traffico sia bilanciato sulle varie interfacce, ipotesi del tutto ragionevole in un nodo della rete di trasporto, il numero massimo di pacchetti ricevuti sulla porta i-esima destinati all' analisi, ossia maxPPSIPS_i, sarà:

$$maxPPSIPS_i \triangleq \frac{maxPPSIPS\ calcolato\ con\ (3)}{n} \quad (7)$$

Con tale accorgimento è evidente che:

$$\sum_{i=1}^{n} maxPPSIPS_i \leq maxPPSIPS\ calcolato\ con\ (3) \quad (8)$$

4.3 Politiche di gestione

Le strategie utilizzate per calcolare il numero massimo di pacchetti da destinare all'analisi in ogni quanto di tempo rappresentano il vero cuore del nostro modulo. Infatti permettono non solo di anticipare il più possibile la scoperta e scarto di pacchetti malevoli, con evidente

risparmio di energia per la trasmissione del pacchetto nel rimanente percorso per la destinazione, ma anche di adattarsi alle condizioni della rete con conseguente massimizzazione delle risorse disponibili senza violare il vincolo di energia massima. La dinamicità intrinseca nella formula (3) è estremamente importante in un contesto come le reti in cui il profilo di traffico è estremamente variabile e "imprevedibile": con una soluzione statica, ossia un numero costante di pacchetti destinati all'analisi nel quanto, si avrebbe un sistema inefficiente che non utilizzerebbe tutte le sue capacità in caso di traffico modesto e andrebbe in sovraccarico in caso di grandi quantitativi di dati; viceversa, con le nostre strategie di ottimizzazione energetica, in ogni quanto si massimizza il numero di pacchetti destinati all' analisi poiché maxPPSIPS è "inversamente proporzionale" al traffico reale o per lo meno previsto. E' evidente che, per ottenere il comportamento desiderato, le strategie devono elaborare buone stime su quello che sarà il traffico osservato nel quanto successivo poiché il nostro sistema lavora in *open loop* per tutto l'intervallo, in altri termini potranno essere effettuate delle manovre correttive solo alla fine del quanto stesso. Riassumendo, definito $\#PktObserved_k$ il numero di pacchetti ricevuti in [(k-1)T,kT], l'obiettivo del nostro studio è quello di determinare delle strategie che, dato $\#PktObserved_k$, stimi il numero di pacchetti che saranno ricevuti in [kT,(k+1)T], ossia il *#PktEstimated* introdotto nel paragrafo 4.2.2, in modo che:

$$\#PktEstimated \approx \#PktObserved_{k+1} \quad (9)$$

Per concludere, data la variabilità delle reti, si propongono tre strategie diverse sia per complessità che per efficacia in relazione al contesto di applicazione.

4.3.1 Costante

Questa strategia è la più semplice per complessità e occupazione di memoria. Infatti, con questa soluzione, si prevede il valore per il quanto successivo concentrandosi unicamente su ciò che si è osservato nel quanto attuale, tralasciando la storia pregressa. Formalmente:

$$\#PktEstimated = \#PktObserved_k \quad (10)$$

Data la sua semplicità, questa strategia è indicata per profili di traffico costanti o, al massimo, leggermente variabili da un valore asintotico. E' evidente che per traffico variabile, sia esso crescente o decrescente, questa strategia non può far altro che "inseguire" l'andamento reale rimanendo però sempre "in ritardo" di un quanto, come rappresentato in figura 4.6. L'errore di approssimazione diventa tanto più significativo tanto più il coefficiente di crescita/decrescita è elevato. Tale comportamento è illustrato nel grafico in figura: nei primi due quanti, in cui il traffico è costante, l'errore di stima è nullo, altrettanto non si può dire dal terzo al

quattordicesimo quanto in cui l'andamento stimato è "traslato" rispetto a quello reale con errori di stima che possono essere significativi.

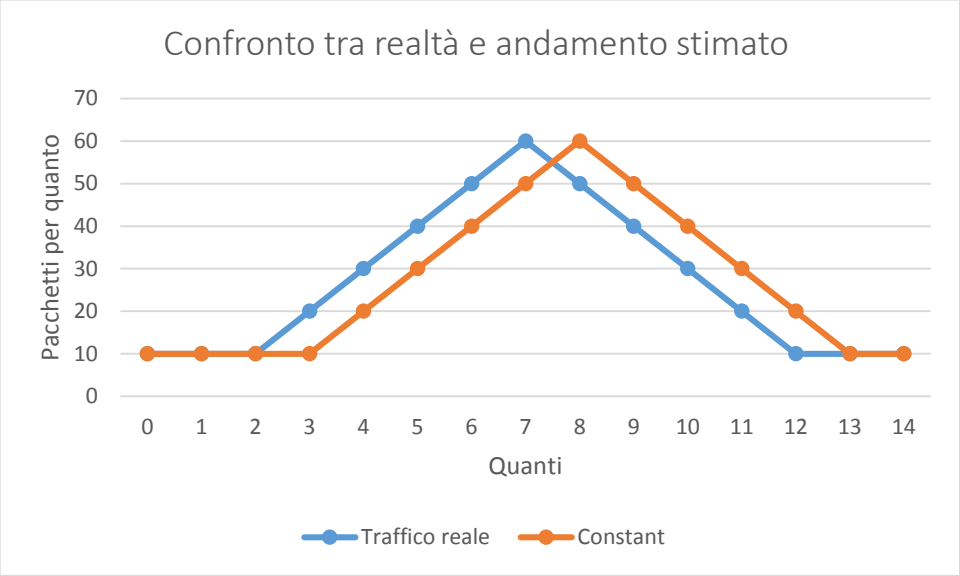

Figura 4.7: Confronto andamento reale e stimato dalla strategia costante

4.3.2 Lineare

Questa strategia, a differenza di quella costante, considera un breve storico utilizzando i valori osservati nei due quanti precedenti per stimare quello imminente. La tecnica scelta è l'interpolazione, ossia un metodo per individuare nuovi punti del piano cartesiano a partire da un insieme noto e finito di punti, nell' ipotesi che questi si possano riferire ad una funzione f(x). Nel nostro contesto la variabile x rappresenta gli istanti di campionamento, quindi un numero intero compreso da 0 a 2 inclusi, mentre la y il numero di pacchetti osservati nel relativo intervallo. Avendo a disposizione due soli valori, la strategia applica un'interpolazione lineare per ricavare la tendenza del traffico.

In generale dati i due punti (x_{k-1},y_{k-1}) e (x_k,y_k) la retta interpolante è:

$$\frac{y - y_{k-1}}{y_k - y_{k-1}} = \frac{x - x_{k-1}}{x_k - x_{k-1}} \quad (11)$$

Sostituendo i nostri valori nella formula (11), il numero di pacchetti stimato per il nuovo quanto è calcolato in funzione di $\#PktObserved_{k-1}$, definito come il numero di pacchetti ricevuti dal router nell'intervallo [(k-2)T,(k-1)T], e $\#PktObserved_k$ attraverso la relazione:

$$\#PktEstimated = 2 \cdot (\#PktObserved_k - \#PktObserved_{k-1}) + \#PktObserved_{k-1} \quad (12)$$

Con questa soluzione è possibile superare i limiti della strategia costante: in caso di crescita o decrescita del traffico, eccezion fatta per il quanto in cui vi è il cambio di tendenza[10], il valore predetto "segue" più fedelmente il profilo reale. Dati alla mano è evidente, riprendendo la stessa condizione di traffico del grafico del paragrafo 4.3.1, che la strategia lineare compie un errore di stima inferiore a quella costante.

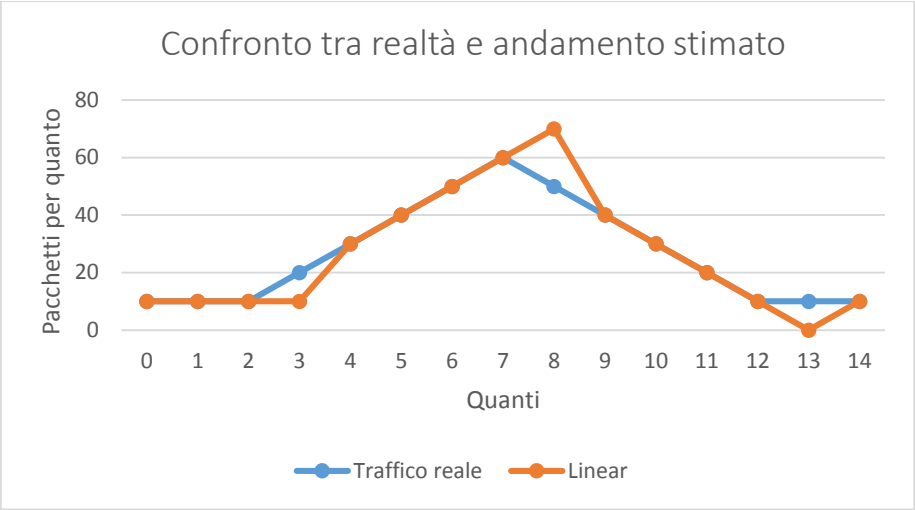

Figura 4.8: Primo confronto tra andamento reale e stimato dalla strategia lineare

Per concludere anche questa strategia ha un caso pessimo, inteso come condizioni di rete per cui l'errore di stima è eccessivamente elevato, che è rappresentato da un andamento "altalenante" con cambio di tendenza a ogni quanto.

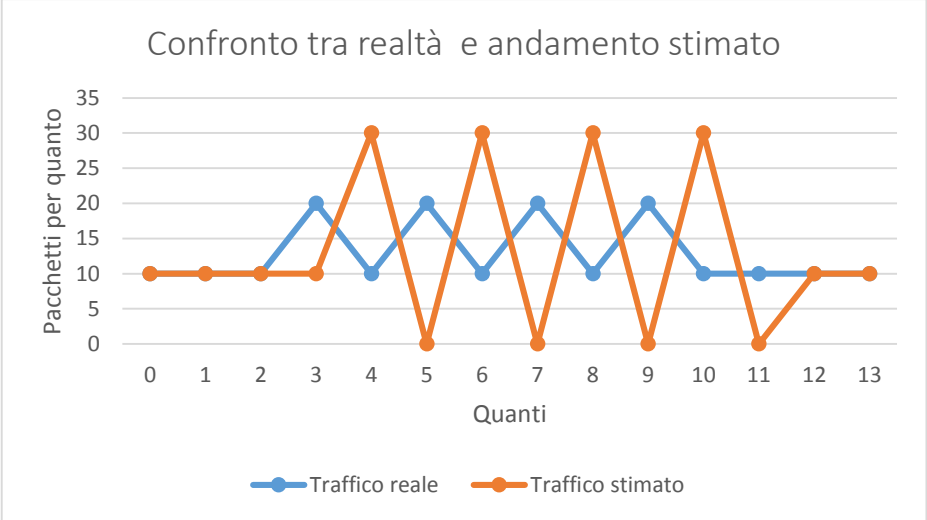

Figura 4.9: Secondo confronto tra andamento reale e stimato dalla strategia lineare

[10] Poiché approssimiamo con una retta, è evidente che in caso di punti crescenti la retta interpolante abbia un coefficiente angolare positivo in quanto la "previsione" non può essere altro che crescente (dualmente per il caso decrescente). Quindi nei cambi di pendenza, da crescita a decrescita e viceversa, il metodo va in contro a un inevitabile errore.

4.3.3 Minimi quadrati

La strategia ai minimi quadrati è la soluzione più onerosa, sia in termini computazionali che di occupazione di memoria. Infatti, con questo metodo di stima, si considerano *n* campioni passati per stimare il valore atteso per il quanto successivo. L'interpolazione tra i punti è sempre lineare, ma avendo un quantità di informazione maggiore, n contro i due della strategia lineare, il valore stimato segue in modo più fedele la curva rappresentate l'andamento del traffico dati nel tempo.

Per comprendere il calcolo di #PktEstimated è necessaria una breve spiegazione di alcune statistiche utilizzate dal metodo stesso: media, varianza e covarianza.

Si definisce media campionaria di una variabile aleatoria X calcolata su un insieme di N campioni x_i, la media delle N realizzazioni, ossia dei valori osservati, di X. Formalmente:

$$\bar{x} \triangleq \frac{1}{N} \cdot \sum_{i=1}^{N} x_i \quad (13)$$

Si definisce varianza campionaria di una variabile aleatoria X calcolata su un campione di N elementi, la media degli scostamenti quadratici dal valor medio delle realizzazioni x_i. Formalmente:

$$\sigma_x^2 = \frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \bar{x})^2 = \overline{x^2} - (\bar{x})^2 \quad (14)$$

E' importante ricordare che la varianza campionaria è un indice di quanto i dati sono dispersi intorno alla media campionaria.

Si definisce covarianza campionaria di due variabile aleatorie X e Y la media dei prodotti dei loro scostamenti dalla media. Formalmente:

$$\sigma_{x,y} = \frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \bar{x}) \cdot (y_i - \bar{y}) = \overline{x \cdot y} - \bar{x} \cdot \bar{y} \quad (15)$$

La covarianza è positiva se, mediamente, X e Y subiscono oscillazioni concordi (quando X supera il valor medio anche Y supera il valor medio), negativa se subiscono oscillazioni discordi (quando X supera il valor medio Y non lo supera, e viceversa), nulla se subiscono oscillazioni indipendenti (quando X supera il valor medio, Y a volte lo supera a volte no).

Come precedentemente detto, queste statistiche sono utilizzate dal metodo dei minimi quadrati il cui obiettivo è quello di determinare la funzione lineare, $f(x)=m \cdot x+q$, che meglio approssima

gli n punti (x_i,y_i) del piano cartesiano. Il generico dato di input (x_i,y_i) appartiene al grafico di f(x) se e solo se:

$$y_i = m \cdot x_i + q \quad (16)$$

Si definisce l'errore sul campione i-esimo, δ_i, come la distanza tra il campione (x_i,y_i) e il valore teorico (x_i,f(x_i)). Formalmente:

$$\delta_i = m \cdot x_i + q - y_i \quad (17)$$

Il metodo dei minimi quadrati determina i parametri m e q tali che sia minima la media degli errori al quadrato. Formalmente si determina il minimo della funzione:

$$f(m,q) = \frac{1}{n} \cdot \sum_{i=1}^{n} \delta_i^2 \quad (18)$$

Tralasciando tutti i passaggi intermedi, consultabili nella risorsa [20], si ottiene che:

$$y = \frac{\sigma_{x,y}}{\sigma_x^2} \cdot (x - \bar{x}) + \bar{y} \quad (19)$$

Per concludere, la strategia applica la formula (19) su *n* campioni assegnando a x_i il significato di i-esimo quanto e a y_i quello di numero di pacchetti ricevuti in esso; sono quindi valide le seguenti considerazioni:

$$x_i \in [0, n-1] \subset \mathcal{N} \quad (20)$$

$$y_i \in [0, maxPPS] \subset \mathcal{N} \quad (21)$$

Di conseguenza al termine del k-esimo intervallo il nodo utilizza il numero di pacchetti ricevuti dal quanto n-k a k, memorizzati in un'opportuna struttura dati, per stimare il numero di pacchetti atteso nel imminente intervallo, ossia in k+1.

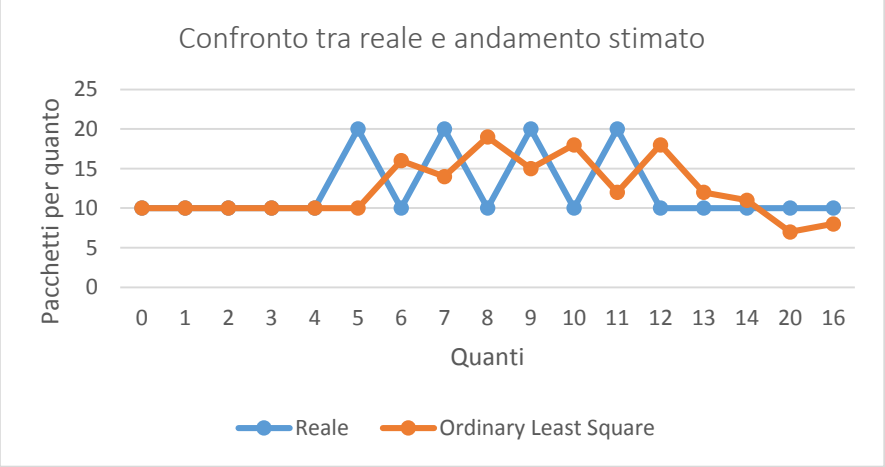

Figura 4.10: Primo confronto tra andamento reale e stimato dalla strategia OLS

Anche questa strategia non permette di seguire perfettamente un andamento fortemente oscillatorio. Confrontando con i risultati mostrati nel caso pessimo della strategia lineare, è evidente che entrambe le errano ma in maniera diversa.

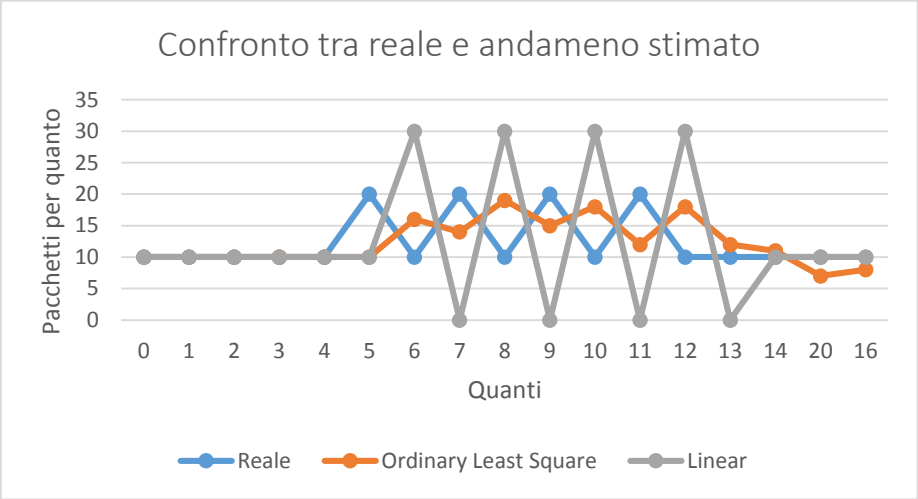

Figura 4.11: Confronto tra andamento reale e stimato dalle strategie lineare e OLS

Infine, si riporta il comportamento per situazioni di traffico variabile, in cui la strategia compie ancora una volta degli errori di stima:

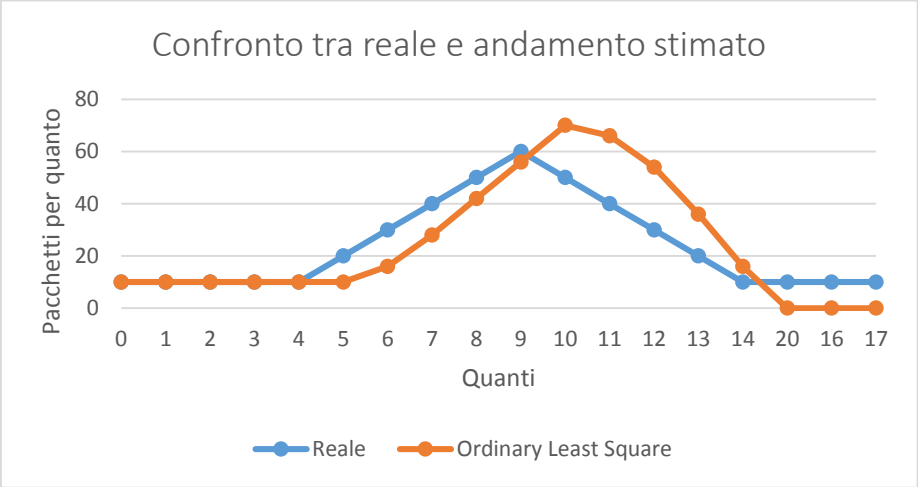

Figura 4.12: Secondo confronto tra andamento reale e stimato dalla strategia OLS

4.4 Considerazioni

Per concludere è necessario sottolineare alcuni aspetti caratterizzanti il nostro innovativo modello.

Il nostro modulo, eseguito su ciascun router che compone la rete, permette di anticipare l'analisi dei pacchetti, eseguita al momento solo dai nodi terminali, con risparmi energetici proporzionali alla percentuale di pacchetti malevoli e lunghezza dei percorsi.

Infatti eventuali pacchetti cattivi, una volta analizzati, sono scartati e quindi non consumeranno risorse per l'operazione di lookup e la trasmissione nei nodi ancora da attraversare lungo il cammino ottimo per la destinazione. Inoltre, grazie alle nostre strategie di ottimizzazione energetica, il numero di pacchetti analizzati dal nodo in ciascun quanto è ottimizzato; questo risultato è ottenibile grazie alla dinamicità intrinseca nella formula (3). Il risultato dell'espressione è strettamente legato alle condizioni di carico attuali, o meglio stimate, del router stesso quindi tanto è minore il numero di pacchetti ricevuti nell'intervallo e, di conseguenza, le risorse richieste dall'operazione di instradamento, tanto maggiore sarà il numero di pacchetti analizzati dall'IPS. Date queste considerazioni può accadere, se non vi sono pacchetti bufferizzati, che il throughput in ingresso al nodo sia maggiore a quello in uscita, in quanto il tempo di esecuzione del task di analisi è superiore a quello di routing. Ciò non rappresenta però un problema perché tale ritardo in trasmissione può essere sovrastimato così da garantire un ritardo massimo per pacchetto. Per concludere è necessario aprire una breve parentesi sulla percentuale di pacchetti controllati all'interno della rete, essa sarà minore o uguale al 100%. Infatti, tale valore è strettamente correlato al traffico, alla dimensione della rete e alle risorse dei nodi che la compongono, ossia dipende fortemente dal contesto. Abbiamo optato per una soluzione "best effort", a discapito di una deterministica[11], per garantire un ritardo massimo in trasmissione, essenziale per diverse applicazioni, assumendo che tutte le reti locali utilizzeranno comunque dei metodi di protezione da/verso l'esterno.

In ogni quanto di tempo il nostro preprocessor lavora in open loop, quindi non potranno essere effettuate delle correzioni fino alla terminazione dell'intervallo stesso. Ancora una volta, ciò non rappresenta un grave difetto poiché la scelta se inviare o meno un pacchetto all'analisi è vincolata sia da maxPPSIPS che dall'energia ancora a disposizione. Quindi in caso di stime del numero di pacchetti ricevuti nel quanto molto minore di quello reale, non andiamo in contro a eccessi di risorse grazie al vincolo sull'energia utilizzabile. In questo caso "pessimo" i pacchetti saranno bufferizzati fino allo scadere dell'intervallo ma, di nuovo, non è un problema significativo in quanto consideriamo T, cioè l'intervallo di tempo in cui il modulo opera in ciclo aperto, ragionevolmente piccolo. Al quanto successivo, i pacchetti memorizzati in precedenza sono inviati sul canale con priorità e, aggiornando le previsioni con il valore reale, il numero di pacchetti destinati all'analisi sarà ridotto e quindi si potrà evitare la bufferizzazione dei nuovi arrivati.

[11] Deterministica nel senso che garantisce il 100% di pacchetti controllati all'interno della rete a prescindere dalle condizioni e configurazione della stessa. Essa potrebbe però portare a ritardi massimi non noti a priori.

Il nostro modulo realizza un modello di analisi distribuita attraverso IPS nella rete di trasporto in cui l'analisi del traffico non è eseguita da un unico nodo ma da tutti i componenti della rete stessa. In fase di progettazione, vi erano due scelte alternative: il nodo che inizia l'analisi di un pacchetto la termina o un nodo ne esegue una porzione, facendo il possibile, delegando la parte rimanente al nodo successivo nel percorso. La nostra scelta è ricaduta sulla prima opzione per diversi motivi, primo tra tutti la filosofia *"keep it simple"*. Infatti il nostro modulo è semplice ma potente e permette di compiere delle scelte sul task da eseguire in tempi praticamente nulli, ovviamente rispetto al tempo richiesto dal routing e dall'analisi, delegando la verifica dei pacchetti a sistemi che sono stati progettati e ottimizzati a tale scopo. Con la seconda opzione, invece, si dovevano necessariamente modificare gli attuali IPS introducendo la funzionalità di "segmentazione" e ricostruzione dello "stato di analisi", ossia terminare l'analisi prima di aver eseguito tutti i controlli propagando l'informazione per permettere al nodo successivo di ricominciare esattamente da dove si era sospeso il precedente. Tale scelta è stata ulteriormente confermata dai dati raccolti attraverso prove empiriche sui tempi medi di analisi per pacchetto dell'IPS Suricata: essi sono talmente ridotti, nell'ordine dei microsecondi in un'architettura single core a 3.1 GHz, che un'eventuale segmentazione e trasmissione del pacchetto al nodo successivo porterebbe solo a un inutile overhead. Per concludere è doveroso sottolineare che, distribuendo l'analisi, la probabilità che i pacchetti per una determinata destinazione siano analizzati dallo stesso nodo è estremamente bassa quindi le allerte con threshold, cioè che devono superare una certa soglia, saranno inviate con frequenza minore rispetto al controllo centralizzato.

Infine merita un approfondimento la scelta delle opzioni delle header IP come mezzo per la trasmissione dell'informazione dello stato del pacchetto. L'utilizzo di un messaggio dedicato scambiato tra gli IPS sarebbe del tutto inefficiente, se non insensato, nel nostro modello in cui lo stato può assumere solo due valori. Inoltre non si avrebbe alcun vantaggio ma, anzi, si introdurrebbe un overhead sia in termini di traffico sulla rete che gestione dell'associazione tra pacchetto IP e relativo messaggio prodotto dall'IPS. Quindi, secondo gli argomenti presentati sopra risulta evidente l'efficacia di inserire tale informazione dentro il pacchetto stesso avvalendosi delle opzioni dell'header IP. Infatti oltre ad eliminare il problema del mapping tra pacchetto IP e il messaggio contenente l'informazione sullo stato, questa scelta è totalmente compatibile con tutti i dispositivi IP, ossia la quasi totalità dei router nella rete. Di conseguenza se il software del router è aggiornato per la gestione delle opzioni dell'header IP potrà far parte del nostro sistema distribuito, altrimenti provvederà al solo instradamento dei pacchetti. Questa è un'ottima soluzione, poiché permette ai gestori di rete di utilizzare il modulo su alcuni nodi

prefissati e di mantenere inalterata parte dell'infrastruttura in uso con un evidente risparmio economico rispetto a cambiare tutti gli apparati di rete. In prima istanza, avevamo pensato a un meccanismo di cifratura a chiave pubblica per garantire l'affidabilità e sicurezza dello stato, che potrebbe essere modificato e alterato da utenti malevoli. Nello specifico, sono possibili due attacchi:

- Aggiornamento dello stato di analisi al valore "da analizzare". Questo attacco ha l'obiettivo di consumare più risorse di quelle necessarie, poiché modificando lo stato, i pacchetti buoni già analizzati potrebbero essere nuovamente controllati con conseguente diminuzione del numero di pacchetti verificati all'interno della rete.

  Per realizzarlo, un attaccante dovrebbe impossessarsi di router della core network, condizione assunta irrealizzabile;
- Immissione di pacchetti cattivi con l'opzione IP falsificata a "analizzato". Questo attacco ha l'obiettivo di trasmettere pacchetti cattivi mascherandoli come buoni, quindi non verranno controllati nella rete, ma può essere facilmente risolto. La soluzione è banale: ogni router di frontiera che riceve pacchetti da una rete locale ne inizializza lo stato a "da analizzare". Si ottiene così uno scambio di informazioni sicuro poiché ogni router è consapevole che la modifica all'header IP è stata apportata da un nodo fidato.

Per quanto enunciato in precedenza, abbiamo optato per non utilizzare la crittografia poiché risulta essere onerosa, aggiunge overhead[12] e non è necessaria nel nostro contesto applicativo.

[12] Da un lato lo scambio delle chiavi tra i router, dall'altro la maggiore dimensione del pacchetto a parità di informazione trasmessa. Inoltre ogni nodo deve decifrare ciascun header in ingresso e cifrare prima di trasmettere, un operazione estremamente onerosa.

Capitolo 5

Implementazione del modello di analisi distribuita

Terminata, nel capitolo precedente, l'esposizione del modello di analisi distribuita tramite Intrusion Prevention System, si propone di seguito una sua possibile implementazione.

Nello specifico, abbiamo sviluppato un simulatore in linguaggio Java il cui compito, come dice il nome stesso, è quello di emulare una rete di trasporto. Per realizzare il modello distribuito, si esegue su ciascun nodo della rete il nostro modulo, cioè si assume che ciascun dispositivo sia in grado non solo di instradare i pacchetti ma anche di effettuarne l'analisi con un qualsiasi IPS. Dati i nostri obiettivi, la simulazione è improntata alla deduzione di tre fattori: il consumo energetico dell'analisi, della trasmissione[13] e il numero di pacchetti controllati all'interno della rete stessa. Infatti il nostro scopo è verificare che il modulo ideato permetta un risparmio energetico e, strettamente correlato, che minimizzi il numero di pacchetti in uscita ancora da analizzare.

Prima di descrivere le parti più significative del codice, oggetto del paragrafo 5.1, è essenziale introdurre i costi energetici per quanto, con la relativa semantica:

- *maxCost*: Massimo consumo sostenibile dal router. Formalmente, definito *AnalysisCost*, *RoutingCost* e *ActiveCost* rispettivamente il consumo nel quanto del router per il task di analisi, di instradamento e di funzionamento, allora:

  $$MaxCost \geq ActiveCost + AnalysisCost + RoutingCost \quad (1)$$

  Tale valore non può essere inferiore all'energia richiesta per il routing del massimo numero di pacchetti gestibili dalla scheda. Infatti, se così non fosse, il nodo, a prescindere dal nostro modulo, non garantirebbe la sua funzionalità di base, ossia l'instradamento. Definito il *lower bound*, il parametro può essere impostato a discrezione, generalmente un buon compromesso tra consumo energetico e percentuale di pacchetti analizzati. Come già sottolineato in passato, tanto maggiore è MaxCost, tanti più pacchetti potranno essere potenzialmente[14] analizzati;

[13] Il consumo energetico della trasmissione è la somma tra il costo di routing e quello di invio sul link.
[14] Potenzialmente in quanto la decisione se inoltrare o meno il pacchetto all'analisi dipende sia dal vincolo energetico che dal maxPPSIPS

- *routingPktCost*: Risorse necessarie al router per compiere l'operazione di instradamento di un pacchetto. Tale valore è il risultato della somma dei consumi dell'operazione di lookup e di trasmissione sul link;
- *analysisPktCost*: Stima delle risorse necessarie per compiere l'operazione di analisi ed eventuale operazione di lookup e trasmissione del pacchetto sul link. Il valore "reale" dipende dal tipo di pacchetto: se cattivo coincide con il tempo di analisi, viceversa con la somma tra il tempo di analisi e quello di instradamento. Di conseguenza, per determinare analysisPktCost, suggeriamo di considerare il valor medio dell'IPS in uso, in quanto se il pacchetto è buono il suo tempo di analisi sarà minore ma dovrà essere effettuato l'instradamento, viceversa il tempo di analisi sarà maggiore ma non si dovrà effettuare il routing;
- *requiredEnergy*: Consumo effettivo dell'analisi del pacchetto. Tale informazione è veicolata nel pacchetto stesso e dipende dalla sua classe di appartenenza. E' evidente che:

$$requiredEnergyGood \ll requiredEnergyBad \quad (2)$$

5.1 Definizione delle classi

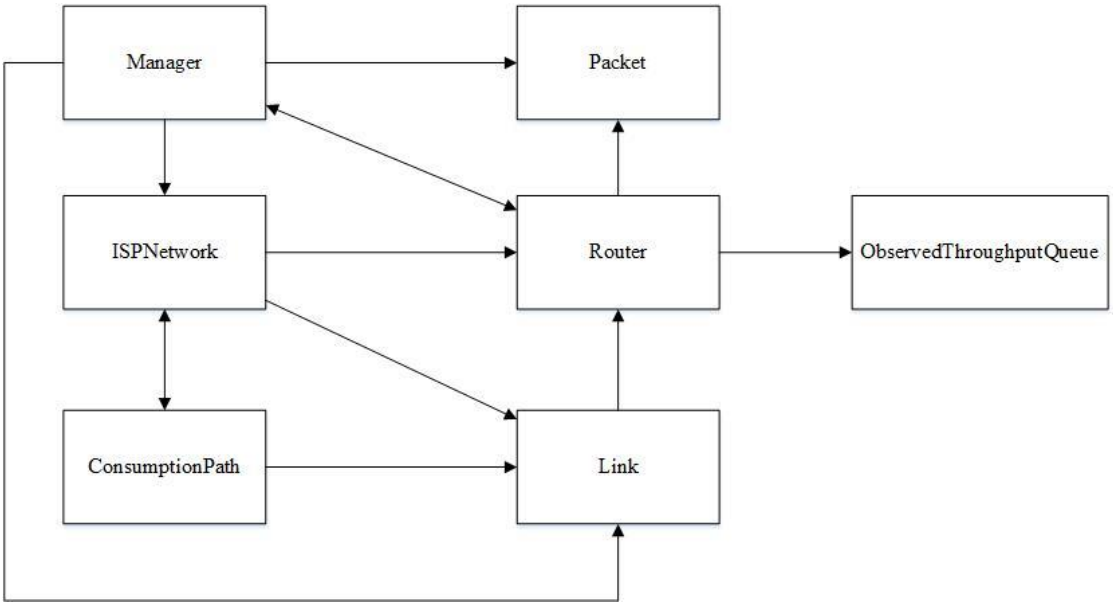

Figura 5.1: Schema del simulatore

Prima di procedere con l'esposizione delle singole classi è necessaria una breve introduzione sul funzionamento del simulatore nel suo insieme. Infatti, come già illustrato in precedenza, la nostra implementazione riproduce il funzionamento di una rete di trasporto. Non avendo a disposizione una macchina fisica per ciascun router, abbiamo implementato il singolo dispositivo con un thread la cui funzione caratteristica, la *run()*, esegue le operazioni del quanto:

preprocessor, analisi e instradamento. Per sincronizzare i diversi router abbiamo realizzato una classe, dal nome emblematico *Manager*, che si occupa di simulare il quanto e l'evoluzione della rete stessa. Senza scendere troppo in dettaglio, per maggiori informazioni si rimanda a 5.1.7, la prima funzionalità è realizzata con una sincronizzazione a barriera, per garantire che tutti i router terminino l'elaborazione del quanto prima di ricominciare con quella del nuovo intervallo, la seconda con la trasmissione dei pacchetti memorizzati nella coda di uscita di ciascun dispositivo. Inoltre, visto l'obiettivo del nostro studio, si aggregano i consumi energetici sostenuti dai singoli router così da ottenere, a fine simulazione, i valori globali. E' importante sottolineare che tali funzioni sono eseguite a "gioco fermo", ossia mentre Manager esegue le proprie istruzioni tutti i router sono sospesi in attesa di essere risvegliati. I parametri della simulazione, che determinano il contesto applicativo, sono diversi: la dimensione della rete[15], la topologia, il numero massimo di pacchetti ricevuti nel quanto e la percentuale di pacchetti malevoli. Per il momento non ammettiamo lo scarto di pacchetti né per mancanza di risorse né per throughput in ingresso superiore a quello gestibile dall'interfaccia di rete. Entrambi i casi sono gestiti utilizzando buffer di dimensione "illimitata": nel primo caso se si sono consumate tutte le risorse il pacchetto verrà memorizzato e gestito appena possibile, nel secondo si invieranno sul link al massimo maxPPS pacchetti, bufferizzando i restanti nella coda di uscita.

5.1.1 Packet

La classe Packet è semplice e composta da pochi campi:

```
//stato di analisi, di default "da analizzare"
boolean checked=false;
//Classe di appartenenza: true se buono, false altrimenti
boolean good;
//Energia richiesta dall'IPS per analizzarlo
int requiredEnergy;
//Nodo di partenza e di destinazione, informazioni necessarie per il routing
int source;
int destination;
//Memorizza il numero di volte in cui è stato bufferizzato
int delay;
//Istante di invio
int timeDelivery;
```

5.1.2 Router

La classe Router, che realizza il modulo oggetto della nostra tesi, è indubbiamente la più interessante. Per iniziare, analizziamo i principali campi di classe:

[15] Per dimensione si intende il numero di router che compongono la rete.

```
//Già illustrati in precedenza
int activeCost;
int maxCost;
int routingPktCost=0;
int analysisPktCost=0;

//Misura del consumo nel quanto attuale per routing e analisi, rispettivamente
int routingCost=0;
int analysisCost=0;

/*Numero massimo di pacchetti ricevibili/inviabili per ciascuna scheda nel
quanto*/
int maxPPS;
/*Capacità totale del router, ossia il prodotto tra il numero di interfacce e
maxPPS*/
int totalMaxPPS;

//Numero massimo di pacchetti da inviare all'analisi nel quanto
int maxPPSIPS;

//Numero di schede del router
int numberInterface=1;

//Contatore dei pacchetti inviati all'analisi fino a questo momento nel quanto
int analyzedThisTurn=0;
int ID;

/*Numero di pacchetti osservati nel quanto, utilizzato per il calcolo di
maxPPSIPS del nuovo intervallo.*/
int receivedPacket;

/*Indicatore della strategia da attuare (routing -> 0, costante -> 1, lineare ->
2 o minimi quadrati -> 3).*/
int strategy;

boolean isIdle=false;
boolean run=true;

//Campi per l'analisi della dimensione media del buffer.
float workingCount=0;
float totBufferSize=0;

Manager manager;

//Gestione dello storico e attuazione delle strategie
ObservedThroughputQueue strategyQueue;
//Buffer interfaccia di ingresso
ArrayList<Packet> waitingInputBuffer= new ArrayList<Packet>();
//Buffer pacchetti da analizzare nel quanto
ArrayList<Packet> inputBuffer= new ArrayList<Packet>();
//Buffer interfaccia di uscita
ArrayList<Packet> outputBufferToRoute= new ArrayList<Packet>();
//Pacchetti bufferizzati nel quanto
ArrayList<Packet> bufferizedQueue= new ArrayList<Packet>();
```

Da notare la variabile *activeCost* che è stata inserita per generalità e rappresenta il consumo energetico di "base" del router derivante dal solo fatto che è acceso. Tale costo è sostenuto dal dispositivo che non compie alcuna operazione di analisi e/o instradamento ma solo le proprie routine di gestione. Nella nostra simulazione, tale valore è volontariamente omesso, o meglio posto a zero, per un semplice motivo: esso è sempre presente, sia che si effettui il solo routing che il nostro modello. Di conseguenza dati i nostri risultati per ottenere il valore "effettivo" sarà sufficiente aggiungere, per ciascun router, il rispettivo costo di active moltiplicato per il numero di quanti emulati nelle nostre prove[16].

```
public void run()
{
   while(run)
   {
      if (!this.isIdle)
      {
             //Il preprocessor smista i pacchetti tra i task
             this.inputPacketSelection();
             //L'IPS compie le proprie analisi
             this.applyAnalysis();
             //Si attende l'inizio del nuovo quanto
             manager.endTurn();
       }
   }
}
```

La classe Router estende la classe Thread, il cui metodo *run()*, ossia l'entry point, contiene le istruzioni da eseguire, se il router ha dei pacchetti da elaborare (ossia non si trova nello stato di idle), in ciascun quanto.

I metodi *inputPacketSelection()* e *applyAnalysis()*, che realizzano il nostro modulo, meritano un' approfondita spiegazione. Brevemente, il primo realizza la logica di gestione decidendo, pacchetto per pacchetto, se inoltrarlo al routing, all'analisi o memorizzarlo in attesa di risorse disponibili; il secondo, come suggerisce il nome, il processo dell'IPS.

inputPacketSelection()

```
this.analyzedThisTurn=0;
double usableEnergy=this.maxCost-this.activeCost;
this.routingCost=0;
this.analysisCost=0;
float tmpCost=0;
ArrayList<Packet> packetBuffering=new ArrayList<Packet>();
Packet p;
```

[16] Nella nostra simulazione un router non può essere mai spento, ossia o in elaborazione o in stato di idle.

Per prima cosa, si "resetta" lo stato, ossia si azzerano tutti i valori necessari per il nuovo quanto sia per la logica del modulo, *tmpCost* e *analyzedThisTurn*, che per l'aggiornamento delle statistiche, *routingCost* e *analysisCost*. Infine si crea una lista, *packetBuffering*, che verrà utilizzata come buffer per eventuali pacchetti che il dispositivo non sarà in grado di elaborare a causa di un consumo totale delle risorse disponibili per l'intervallo.

Successivamente sono instradati i pacchetti bufferizzati nel quanto precedente, questa operazione è prioritaria sull'analisi poiché non è possibile ritardare "all'infinito" la trasmissione del pacchetto. Infatti la maggior parte delle attuali applicazioni vincola il ritardo massimo in ricezione quindi è indispensabile rispettarlo. Terminato ciò, il modulo può iniziare il suo vero e proprio compito prelevando il pacchetto in testa alla coda dei dati da elaborare, *waitingInputBuffer*, decretandone il task di destinazione. Prima di compiere una qualsiasi scelta, occorre verificare la disponibilità di risorse perché, se esaurite, l'informazione deve essere memorizzata per la futura elaborazione.

```
//Instrado gli eventuali pacchetti memorizzati nel quanto precedente.
for (int i=0; i<this.bufferizedQueue.size(); i++)
{
   if(tmpCost+this.routingPktCost<=usableEnergy)
   {
      this.routingCost+=this.routingPktCost;
      tmpCost+=this.routingPktCost;

      if(p.destination!=this.ID)
         this.outputBufferToRoute.add(p);
      else
         //Memorizzo i pacchetti giunti a destinazione per l'analisi dei ritardi
         manager.allPacketToDest.add(p);
   }
   else
   {
      packetBuffering.add(p);
      p.delay++;
   }
}

/*Metodo che richiama al proprio interno quello della classe
ObservedThroughputQueue che realizza la strategia di stima impostata.*/
calculatePacketIPS();

//Elaborazione del preprocessor.
for (int i=0; i<this.waitingInputBuffer.size(); i++)
{
   p=this.waitingInputBuffer.get(i);

   // controlliamo la disponibilità di energia almeno per il routing
   if(tmpCost+this.routingPktCost<=usableEnergy)
       politicaPreprocessor;
```

```
    else
    {
        packetBuffering.add(p);
        p.delay++;
    }
}

/*Si recupera l'informazione sul numero di pacchetti osservati nel quanto appena
concluso che sarà l'input per la strategia di stima del prossimo quanto.*/
this.receivedPacket=waitingInputBuffer.size();

//Si "svuota" la coda in ingresso
this.waitingInputBuffer=new ArrayList<Packet>();
this.bufferizedQueue= packetBuffering;
```

Nella condizione "energetica" si verifica se, data l'energia consumata fino a questo momento, vi è almeno la possibilità di compiere il routing del nuovo pacchetto in elaborazione. La motivazione di tale controllo è semplice: l'instradamento ha un costo inferiore all'analisi quindi, se non vi è possibilità per questa operazione, a maggior ragione non vi saranno risorse sufficienti per l'intrusion prevention system. Inoltre, con questa soluzione, si massimizza l'utilizzo delle risorse garantendo comunque l'elaborazione del pacchetto, poiché, se il modulo non avesse disponibilità per l'analisi esso verrà comunque instradato. Per concludere, riportiamo il codice indicato in precedenza con "politicaPreprocessor".

```
/*Controlliamo se ho energia per fare almeno il routing, se si entrerà in
funzione il preprocessor altrimenti bufferizziamo.*/
if(tmpCost+this.routingPktCost<=usableEnergy)
{
    /*Controllo, consultando le opzioni dell'header IP, se il pacchetto è già
    stato analizzato in uno dei nodi precedenti, se si effettua direttamente il
    routing, altrimenti è un candidato per l'analisi.*/

    if(p.checked)
    {
        //Che sia arrivato o no a destinazione effettuo il routing
        this.routingCost+=this.routingPktCost;
        tmpCost+=this.routingPktCost;
        if(p.destination!=this.ID)
            this.outputBufferToRoute.add(p);
        else
            manager.allPacketToDest.add(p);
    }

    else
    {
        /*Il pacchetto, candidato all'analisi sarà inoltrato a questo processo
        solo se:
        1)L'IPS può ancora analizzare dei pacchetti nel quanto
        2)Vi sono risorse a sufficienza per l'operazione
        altrimenti sarà ugualmente elaborato, ossia sarà instradato.*/
```

```
      if(this.analyzedThisTurn<this.maxPPSIPS &&
           tmpCost+this.analysisPktCost)<=usableEnergy )
      {
          tmpCost+=this.analysisPktCost;
          this.inputBuffer.add(p);
          this.analyzedThisTurn++;
      }
      else
      {
          this.routingCost+=this.routingPktCost;
          tmpCost+=this.routingPktCost;
          if(p.destination!=this.ID)
             this.outputBufferToRoute.add(p);
          else
             manager.allPacketToDest.add(p);
       }
   }
}
```

applyAnalysis()

Questo metodo simula il processo di analisi dell'intrusion prevention system. Poiché il nostro studio è orientato ai soli consumi energetici, si è volontariamente tralasciato il tempo reale dell'IPS per controllare un pacchetto. Tra l'altro un'analisi di performance, che considererebbe i tempi di esecuzione, non avrebbe qui alcun senso, poiché, come precedentemente scritto, la rete non è simulata su nodi reali e distinti ma su un'unica macchina.

```
private void applyAnalysis()
{
   /*Eseguo l'analisi per ogni pacchetto in inputBuffer, opportunamente riempito
   Da inputSelection()*/
   for(Packet p: inputBuffer)
   {
      //Se il pacchetto è buono deve essere instradato, se cattivo è scartato
      if(p.good)
      {
            //Aggiorno lo stato di analisi nell'headerIP a "Analizzato"
            p.checked=true;

            if (p.destination!=this.ID)
               this.outputBufferToRoute.add(p);
            else
                manager.allPacketToDest.add(p);

            this.routingCost+=routingPktCost;
      }
      //In entrambi i casi aggiorno il costo di analisi.
      this.analysisCost+=p.getEnergyCost();
   }
   inputBuffer=new ArrayList<Packet>();
}
```

Due istruzioni meritano una spiegazione aggiuntiva: l'aggiornamento del contatore di energia consumata dall'analisi e la reinizializzazione della coda *inputBuffer*. Inserire il primo nel metodo inputPacketSelection() sarebbe logicamente errato poiché, senza effettuare l'analisi, non sarebbe noto il tipo di pacchetto e, di conseguenza, l'energia effettivamente richiesta. Il secondo deriva dalla semplice considerazione che tutti i pacchetti destinati all'analisi, contenuti in inputBuffer, sono necessariamente elaborati poiché il preprocessor li assegna a questo task garantendo che non sia violato alcun vincolo.

5.1.3 Link

La classe Link rappresenta il collegamento fisico tra due router. I campi che la caratterizzano sono pochi e facilmente comprensibili:

```
//Estremi del link.
Router a,b;
int energyCost;
//Necessario per l'esplorazione del grafo rappresentante la rete.
int ID;
```

Per generalità abbiamo inserito il campo *energyCost* per definire l'energia consumata dalla trasmissione del pacchetto sul link. Nella nostra simulazione, assumendo che ciascuna interfaccia consumi le stesse risorse per l'invio sul canale, tale valore è pari a zero poiché "inglobato" direttamente nella variabile routingPktCost.

5.1.4 ObservedThroughputQueue

La classe ObservedThroughputQueue ha due compiti principali: applicare la strategia per la stima del numero di pacchetti previsti nel quanto da iniziare e la gestione dello storico.

Ciascuna strategia di stima riceve, come parametro, il numero di pacchetti in ingresso al dispositivo nel quanto appena terminato. Di conseguenza, la strategia costante non necessita di alcun storico, la lineare di un solo campione e quella ai minimi quadrati, con finestra di osservazione pari a *n*, degli n-1 valori osservati negli n-1 quanti precedenti. Abbiamo però optato per la gestione, a prescindere dalla strategia, di uno storico di n-1 campioni così da permettere, in lavori futuri, un cambiamento in real-time della stessa senza perdita di informazione, o meglio, garantendo dei valori reali per la stima. Tale scelta implementativa permette un calcolo accurato della previsione anche a fronte di cambiamenti di tattica, in particolare passando da costante a lineare o da quest'ultima ai minimi quadrati. Se così non fosse, la strategia potrebbe generare risultati precisi solo dopo una fase transitoria la cui durata dipende dal numero di campioni utilizzati. Infatti, se non si memorizzassero ad ogni quanto i

valori reali, l'unica soluzione, a seguito di un cambiamento, sarebbe quella di considerare i campioni passati mancanti pari a un valore di default. Per tutelarci da "overbooking"[17] e conseguente bufferizzazione dei pacchetti, si consiglia di impostarlo a maxPPS così da privilegiare, nella fase transitoria, il routing all'analisi.

Per la memorizzazione dello storico, onde evitare un'eccessiva occupazione in memoria, abbiamo implementato una coda circolare di dimensione *capacity*.

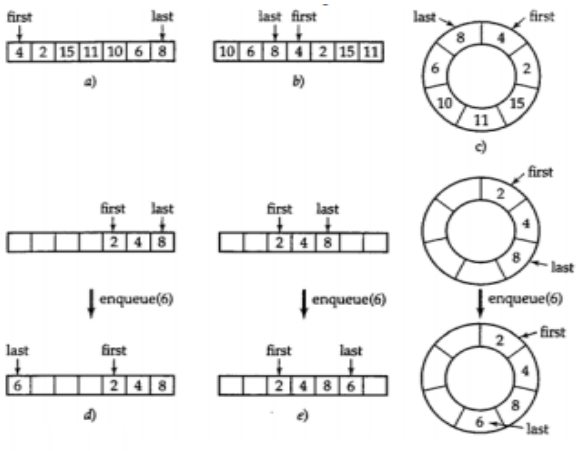

Figura 5.2: Coda circolare

Brevemente, gli elementi sono inseriti nell'array partendo dall'indice zero fino a *capacity-1* e, a seguito di un nuovo valore, esso sarà inserito nuovamente in posizione zero. Nel nostro contesto dato il nuovo valore, ricevuto come parametro del metodo, esso deve essere memorizzato sovrascrivendo quello più "vecchio". Per inserire correttamente la nuova informazione si utilizza il campo di classe *head* che indicizza contemporaneamente la cella in cui è memorizzato il valore più antico e quella per il nuovo inserimento. Per realizzare tale maccanismo si combina l'incremento del valore di head con l'operatore resto, % nel linguaggio Java, con dividendo capacity. Per concludere è evidente che scorrendo logicamente l'array in senso orario partendo da head, si ottiene la sequenza ordinata dei campioni, dal più antico al più recente. Con un esempio, dato capacity e head, rispettivamente 5 e 3:

23	42	12	78	90

La sequenza dei campioni dal più antico al più recente sarà:

1. Elemento in posizione head: 78;
2. Elemento in posizione (head+1)%capacity: 90;

[17] Abbiamo overbooking in caso di stime di throughput in ingresso molto minori di quelli effettivamente osservate, con conseguente sovraccarico dell'IPS.

3. Elemento in posizione (head+2)%capacity: 23;
4. Elemento in posizione (head+3)%capacity: 42;
5. Elemento in posizione (head+4)%capacity: 12;

Detto ciò, i campi di classe sono estremamente comprensibili:

```
//Ampiezza della finestra di osservazione
int capacity=1;
int head = 0;
int [] observedPacket;

//Necessari per la stima ai minimi quadrati e costanti, fissato capacity.
float mediaI=0;
float varI=0;
float mediaI2=0;
```

Stesso discorso per l'implementazione delle strategie descritte in dettaglio in 4.3

```
protected int constanThroughputStrategy(int observedThroughput)
{
   //Aggiornamento storico
   this.observedPacket[head]=observedThroughput;
   head=(head+1)%capacity;
   return observedThroughput;
}
```

```
protected int linearThroughputStrategy(int observedThroughput)
{
   int estimatedThroughput=0;
   int indexPreviousThroughput=(this.head+this.capacity-1)%this.capacity;
   int previousThroughput = this.observedPacket[indexPreviousThroughput];

   estimatedThroughput = (int)(2*observedThroughput-previousThroughput);

   /*controllo che il valore sia positivo, perché in caso di trend decrescente e
   traffico basso il risultato della stima potrebbe essere inferiore di zero,
   cosa impossibile nella nostra semantica*/
   if(estimatedThroughput<0)
       estimatedThroughput=0;
   //Aggiornamento dello storico
   this.observedPacket[head]=observedThroughput;
   head=(head+1)%capacity;
   return estimatedThroughput;
}
```

```
protected int ordinaryLeastSquaresThroughputStrategy(int observedThroughput)
{
   int estimatedThroughput=0;
   float output=0;
   float mediaO=0;
   float mediaIO=0;
   float covIO=0;
   int i=0;
```

```
    //Scandisco lo storico dal più antico al più recente.
    for(;i<this.capacity;i++)
    {
        output=this.observedPacket[(head+i)%this.capacity];
        mediaO+=output;
        mediaIO = mediaIO+(i*output);
    }

    mediaIO = (mediaIO+i*observedThroughput)/(i+1);
    mediaO = (mediaO+observedThroughput)/(i+1);
    covIO = mediaIO-this.mediaI*mediaO;
    estimatedThroughput=(int)(covIO*(i+1)/this.varI+mediaO -
                                    covIO*this.mediaI/this.varI);

    if( estimatedThroughput<0)
        estimatedThroughput=0;
    //Aggiornamento dello storico
    this.observedPacket[head]=observedThroughput;
    head=(head+1)%capacity;
    return estimatedThroughput;
}
```

Infine si sottolinea che si vincolo superiormente i valori delle stime a maxPPS,

5.1.5 ISPNetwork

La classe ISPNetework, dal nome eloquente, costruisce la rete da simulare memorizzandone gli elementi principali.

```
ArrayList<Router> routers = new ArrayList<Router>();
ArrayList<Link> links = new ArrayList<Link>();
ConsumptionPaths cp;
int linksnr=0;
int routersnr=0;
//Topologia della rete (0 -> star, 1 -> mesh parziale, 2 -> completamente
connessa)
int mode=0;
```

Al momento sono state implementate tre diverse topologie: simil-stella, mesh parziale e completamente connessa.

La prima topologia è una stella leggermente modificata, ossia una struttura a “croce” con una determinata profondità. Nello specifico, si definisce profondità il numero di nodi che compongono il generico livello, quindi il centro della stella avrà una quantità di figli pari al suddetto valore, gli altri, a eccezione delle foglie, un unico figlio. In figura 5.3 è riportata una topologia a stella con profondità quattro.

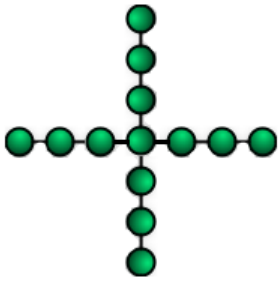

Figura 5.3: Topologia a stella

Il codice è relativamente semplice:

```
void buildStarTopolgy(int depth)
{
   ArrayList<Router> temp= new ArrayList<Router>();
   Router rChild;
   int i;
   //Seleziono il centro della stella
   Router r=routers.get(0);

   if(depth>=this.routersnr)
       depth=this.routersnr-1;

   /*Creazione del primo livello, ossia i collegamenti tra il centro della
   stella e i suoi depth figli*/
   for (i=1; i<=depth; i++)
   {
       rChild=routers.get(i);
       this.links.add(new Link(i-1,0, r, rChild));
       temp.add(rChild);
   }

   // Aggiungo a ciascuna foglia il relativo figlio e così via finché ho nodi
   while (i<this.routersnr)
   {
       r=temp.remove(0);
       rChild=routers.get(i);
       this.links.add(new Link(i-1,0, r, rChild));
       temp.add(rChild);
       i++;
   }
   this.linksnr=i-1;
}
```

Le reti mesh si dividono in due categorie, completamente connesse e parziali. A grandi linee, nelle prime ogni nodo è connesso a tutti gli altri dispositivi presenti in rete, nelle seconde non vi sono tutti i collegamenti teoricamente possibili.

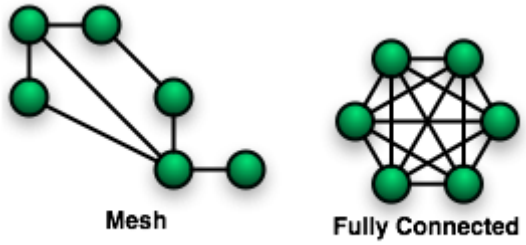

Figura 5.4: Topologie mesh

Si riporta, per completezza, i relativi metodi:

```
void buildPartialMeshTopology(int linkN)
{
   Random r=new Random(System.currentTimeMillis());

   ArrayList<Router> notConnected= new ArrayList<Router>(this.routers);
   ArrayList<Router> connected= new ArrayList<Router>();
   Router src, dst;
   int a,b;
   this.linksnr=0;

   /*Controlliamo che il numero di link sia minore di quello massimo, ossia il
   numero di archi presenti nel mesh completo. Se così non è chiamiamo
   direttamente la topologia mesh completa*/
   int maxLinks=this.routersnr*(this.routersnr-1)/2;
   if (linkN>maxLinks)
       buildCompleteMeshTolopology();

   else
   {
        /*Controllo per garantire che vi sia il minimo numero di archi,
        condizione necessaria per creare un grafo connesso*/
        if(linkN<this.routersnr-1)
             linkN=this.routersnr-1;

        /*Creo il grafo minimo connesso, quindi tutti i router avranno almeno un
        collegamento con un altro della rete. Per fare ciò si preleva
        casualmente un router dall'insieme di quelli non ancora connessi e
        l'altro dai connessi.*/

        a=Math.abs(r.nextInt(notConnected.size()));
        connected.add(notConnected.remove(a));
        for(this.linksnr=0;this.linksnr<this.routersnr-1;this.linksnr++)
        {
             a=Math.abs(r.nextInt(connected.size()));
             b=Math.abs(r.nextInt(notConnected.size()));
             dst=notConnected.remove(b);
             this.links.add(new Link(this.linksnr,0,connected.get(a),dst));
             connected.add(dst);
         }

        /*Aggiungo gli eventuali link ancora da creare generando casualmente uno
        dei due routers e in modo semi-casuale l'altro. Tale soluzione, a
        differenza della casuale, garantisce che non si cicli all'infinito. Caso
        possibile se il numero di link è prossimo a quello massimo.*/
```

```
        a=this.linksnr%this.routersnr;

        while (this.linksnr<linkN)
        {
             b=Math.abs(r.nextInt(this.routersnr));
             src=this.routers.get(a);
             for(int i=0;i<this.routersnr; i++ )
             {
                   dst=this.routers.get(b);

                   /*Se il link non esiste lo aggiungo, altrimenti mi sposto nel
                   router sucessivo.*/
                   if ((a!=b) && !checkLinkof(src,dst))
                   {
                        links.add(new Link(this.linksnr,0,src,dst));
                        this.linksnr++;
                        break;
                   }
                   b=(b+i)%this.routersnr;
             }
             a=(a+1)%this.routersnr;
        }
   }
}
```

```
void buildCompleteMeshTolopology()
{
   this.linksnr=0;
   Router src, dst;
   for (int i=0; i< this.routersnr; i++)
   {
       src=routers.get(i);
       for (int j=0; j<this.routersnr; j++)
       {
             if (i!=j)
             {
                   dst=routers.get(j);
                   this.links.add(new Link(this.linksnr,0, src, dest));
                   this.linksnr++;
             }
       }
   }
}
```

5.1.6 ConsumptionPaths

La classe ConsumptionPaths, utilizzata internamente da ISPNetwotk, costruisce il grafo di rete e la tabella di routing. Brevemente, un grafo è un insieme di elementi, detti nodi o vertici, collegati fra loro da archi o lati. Più formalmente, si definisce grafo una coppia ordinata di insiemi $G = (V, E)$, con V insieme dei nodi ed E insieme degli archi, cioè coppie di elementi di V.

$$E \subseteq V \, X \, V$$

Il grafo di rete è costruito utilizzando come insieme V i router ed E i link, entrambi opportunamente istanziati in fase di costruzione della topologia di rete. Per la creazione del grafo e successiva esplorazione, abbiamo utilizzato la libreria *jgrapht* che permette di compiere diverse operazioni, anche complesse, con poche linee di codice. In particolare abbiamo utilizzato l'oggetto *SimpleWeightedGraph<V,E>*[18]. Nello specifico, definito l'oggetto SimpleWeightedGraph<Router,DefaultEdge>, si costruisce il grafo di rete riempendo opportunamente i due insiemi.

```
public ConsumptionPaths(ISPNetwork net)
{
   Router r;
   Link l;
   this.routingTable=new Link [net.routersnr][net.routersnr];

   /*Aggiungiamo i vertici del grafo, ossia i routers. Nello stesso ciclo
   Inizializziamo anche i valori della tabella di routing che verrà riempita
   successivamente.*/
   for (int i=0; i<net.routersnr; i++)
   {
       r=net.routers.get(i);
       objectGraph.addVertex(r);
       for (int j=0; j<net.routersnr; j++)
             routingTable[i][j]=null;
   }

   /*Aggiungo gli archi del grafo, ossia i link espressi come router di partenza
   e destinazione*/
   for (int i=0; i<net.linksnr; i++)
   {
       l=net.links.get(i);
       objectGraph.addEdge(l.a, l.b);
   }
   //Utilizzato da ISPNetwork per controllare che il grafo sia connesso.*/
   ci=new ConnectivityInspector<Router, DefaultEdge>(objectGraph);
}
```

Una volta determinato il grafo, si procede con il calcolo della tabella di routing. Abbiamo optato per un'unica tabella di rete che memorizzi per ciascuna coppia di router il nodo successivo nel cammino ottimo dal primo al secondo. Queste informazioni sono inserite in una matrice quadrata di dimensione pari al numero di router della rete e sono consultate dalla classe Manager in fase di instradamento, con il metodo lookup che, dato il nodo e la destinazione del pacchetto, ritorna il prossimo router nel cammino ottimo. I percorsi minimi sono calcolati con l'algoritmo di Dijkstra utilizzando come metrica il numero di hop, ossia il numero di archi da attraversare seguendo il cammino dalla sorgente alla destinazione.

[18] A simple graph is an undirected graph for which at most one edge connects any two vertices, and loops are not permitted. [javadoc]

```
public void createRoutingTable(ISPNetwork net)
{
   List<DefaultEdge> l;
   Router source, dest = null;
   Link link=null;
   for(int i=0;i<routingTable.length;i++)
   {
      //Dato l'i-esimo router calcolo il percorso ottimo per tutti gli altri.
       for(int j=0;j<routingTable[i].length;j++)
       {
          if(i!=j)
          {
              source = net.routers.get(i);
              l=DijkstraShortestPath.findPathBetween(objectGraph,source,
                                                      net.routers.get(j));

              dest=objectGraph.getEdgeTarget(l.get(0));

              /*L'arco è composto da un source e target che non hanno alcuna
              valenza matematica, ossia rimangono statici. Per esempio l'arco
              (0,2) rimarrà tale sia se da 0 vado a 2 che da 2 a 0. Quindi si
              seleziona l'effettivo target per la nostra source*/
              if(dest.ID==i)
                    dest=objectGraph.getEdgeSource(l.get(0));

              link=net.getLinkof(source, dest);
              routingTable[i][j]= new Link(link.getId(),link.getEnergyCost(),
                                                  source, dest);
          }
       }
   }
}
```

5.1.7 Manager

Questa classe, come suggerisce il nome, è estremamente importante per la nostra simulazione. Infatti ha diversi compiti essenziali: instradare i pacchetti nella rete e sincronizzare i thread router per emulare l'evoluzione temporale, ossia il susseguirsi dei quanti. Inoltre gestisce le statistiche della rete nella sua interezza aggregando, quanto per quanto, i consumi energetici del nodo nell'intervallo appena terminato.

Per comprendere i metodi di Manager è necessaria una breve introduzione sulla sincronizzazione dei thread in parallelo. Infatti essi, condividendo la memoria, non sono del tutto indipendenti quindi non è garantito che le operazioni sulle risorse condivise siano eseguite in modo atomico. Non essendo noto a priori l'istante temporale in cui lo scheduler sospenderà il thread, quest'ultimo potrebbe essere interrotto da un context switch[19] prima che abbia terminato la "transazione" sulle risorse condivise, anche detta regione critica, rendendole quindi

[19] stato del sistema operativo durante il quale avviene il cambiamento del processo correntemente in esecuzione su una delle CPU. Prima di tutto è necessario salvare lo stato della computazione del processo, tra cui il *program counter* ed il contenuto dei *registri* generali, in modo che l'esecuzione potrà essere ripresa in seguito.

inconsistenti. Per ovviare a ciò si deve garantire che, per tutto il tempo in cui esegue la transazione, nessun altro thread possa accedere alla risorsa. In Java ogni oggetto (istanza di *Object*) ha associato un *mutual exclusion lock* ed è possibile garantirne il mutuo accesso inserendolo in blocchi di codice dichiarati *synchronized*. Inserendo la regione critica in un blocco synchronized sull'oggetto da elaborare, si garantisce che un solo thread alla volta possa eseguire quelle istruzioni. Spesso tali blocchi, come nel nostro caso, sono utilizzati per sospendere i thread fino al verificarsi di una determinata condizione. Per realizzare tale meccanismo vi sono due funzioni: *wait()* e *notify()*. Se un thread chiama la wait() su un oggetto di cui possiede il lock, esso, dopo averlo rilasciato, passa in stato di waiting e potrà essere risvegliato, ossia essere nuovamente schedulato, solo a seguito della notify() sullo stesso oggetto. Analizziamo, per iniziare, i campi di classe:

```
int badpackets=0;
int goodpackets=0;
int allAnalizedPacket=0;
/*Numero di routers che hanno terminato l'elaborazione per quel quanto. Quindi
appena si raggiunge il numero di routers della rete si potrà iniziare il nuovo
intervallo.*/
int routers=0;

/*Parametro utilizzato per determinare il numero massimo di pacchetti immessi su
un router di frontiera in un singolo quanto e il tipo di distribuzione. Utile in
ottica della nostra raccolta dati*/
int networkThroughput=0;
int distributionPktMode=0;

//Statistiche della rete
int fullCost=0;
int analysisCost=0;
int fullDeliveryCost=0;
int totalIdleCost=0;
/*Strutture per memorizzare i pacchetti da immettere, per ciascun router, nella
rete e per l'analisi dei ritardi.*/
ArrayList<ArrayList<Packet>>packetDistributor;
ArrayList<Packet> allPacketToDest;
```

Il primo metodo di Manager, invocato nel main del simulatore, è *buildingPacketPool* che immette i pacchetti nella rete permettendo l'inizio dell'evoluzione della stessa. I parametri passati sono di semplice comprensione: la destinazione, *packetDest*, il numero di pacchetti, *numpacket*, con la relativa percentuale di malevoli, *percentage*, e i costi di analisi per buoni e cattivi, *energygood* e *energybad* rispettivamente.

```
public void buildingPacketPool(int numpacket, int energygood, int energybad ,
float percentage, int packetDest)
{
   //Array contenente, per ciascun router, tutti i pacchetti in ingresso
   this.packetDistributor= new ArrayList<ArrayList<Packet>>();
   ArrayList<Packet> packetPool;
   for(int i=0;i<this.net.routersnr;i++)
   {
       packetPool = new ArrayList<Packet>();
       packetDistributor.add(i,packetPool);
   }

   /*Distribuzione dei pacchetti sui vari router. Con la modalità uniforme si
   assegna a ciascun router lo stesso numero di pacchetti, con la random, sia
   essa con distribuzione di probabilità uniforme o gaussiana, i pacchetti sono
   immessi in modo casuale.*/

   int actualbad=(int)((percentage*numpacket)/100);
   this.badpackets=badpackets+actualbad;
   this.goodpackets=goodpackets+(numpacket-actualbad);

   switch(this.distributionPktMode)
   {
       case 0 :
              {
                     uniformDistribution(energygood, energybad,packetDest);
                     break;
              }
       case 1 :
              {
                     randomDistribution(energygood, energybad,packetDest);
                     break;
              }
       default:
              {
                     randomDistribution(energygood, energybad,packetDest);
                     break;
              }
   }
}
```

Terminato il metodo, nella coda di ingresso di ciascun router vi saranno al massimo networkThroughput pacchetti pronti per l'elaborazione. I restanti, ancora memorizzati nella coda del router in *packetDistributor*, saranno distribuiti appena possibile. Per maggiori dettagli si rimanda al metodo *setNextTurn()*.

Descriviamo ora il codice che simula il quanto, nei fatti la sincronizzazione dei vari router che lavorano in parallelo. Come illustrato in 5.1.2, la generica iterazione del thread, che esprime le operazioni nel quanto, termina con il metodo *endTurn()* di Manager. Esso garantisce che ciascun nodo possa cominciare la nuova iterazione, ossia il nuovo quanto, con i dati aggiornati. In altri termini ciascun dispositivo attende la conclusione di tutti gli altri e l'instradamento dei pacchetti di tutte le interfacce di rete per poi ripartire con la nuova iterazione. Nei fatti,

permettendo l'analogia, è come se Manager realizzasse il "clock della rete" a cui tutti i dispositivi sono, per definizione, allineati.

```
public void endTurn()
{
   boolean isLast=false;

   synchronized(this)
   {
      //Marchiamo la fine dell'elaborazione del router.
      this.routers++;
      if (this.routers< this.net.routersnr)
      {
            try
            {
               //Vi sono ancora dei router all'opera quindi attendo.
               this.wait();
            }
            catch (InterruptedException e)
            {
                  e.printStackTrace();
            }
      }
      else
      {
          //L'ultimo router ha terminato, deve iniziare il nuovo quanto.
         isLast=true;
         routers=0;
      }

   }
   /*Se tutti i router hanno terminato l'elaborazione, procedo con le operazioni
   necessarie per il nuovo quanto*/
   if(isLast)
      this.setNextTurn();
}
```

L'ultimo router che chiama endTurn() invocherà il metodo *setNextTurn()* che, come suggerisce il nome, esegue le operazioni necessarie per l'aggiornamento delle statistiche e, se vi sono ancora pacchetti in circolazione, l'evoluzione della rete.

```
public void setNextTurn()
{
   //Numero di router idle, ossia senza pacchetti da elaborare, nel nuovo quanto
   int idle=0;
   Router temp;
   /*Per ciascun router si memorizzano i consumi, aggiornando le statistiche
   globali, si instradano i pacchetti presenti sulla coda di uscita e si
   immettono eventuali pacchetti ancora da distribuire.*/
   for (int i=0; i<this.net.routersnr; i++)
   {
      temp=this.net.routers.get(i);
      this.nextNodeRouting(temp);
      this.updateAnalysisStatistics(temp);
      this.distributePackets(temp);
   }
```

```
    /*Effettuate le operazioni di instradamento si controlla il numero di router
    idle per verificare il criterio di arresto.*/
    for (int i=0; i<this.net.routersnr; i++)
    {
        temp=this.net.routers.get(i);
        if (temp.waitinigInputBuffer.isEmpty() && temp.bufferizedQueue.isEmpty())
        {
            temp.isIdle=true;
            idle++;
        }
        else
        {
             //Aggiornamento per la dimensione media dei buffer
             temp.isIdle=false;
             temp.totBufferSize+=temp.bufferizedQueue.size();
             temp.workingCount++;
        }
    }

    /*Abbiamo terminato la simulazione, ossia tutti i router sono idle? Se sì,
    si scrivono i risultati su file e si fermano i thread, altrimenti si
    risvegliano i router per l'elaborazione del nuovo quanto./*
    if (idle==net.routersnr)
    {
         /*Calcoliamo le informazioni, oltre analysisCost e fullDeliveryCost, che
         saranno da stampate su file.*/
        int totalCost=this.analysisCost+this.fullDeliveryCost;
        float percentage = (float)(this.badpackets*100)/
                           (float)(this.goodpackets+this.badpackets);
        float checkedPercentage=(float)(this.allAnalizedPacket*100)/
                                (float)(this.goodpackets+this.badpackets);
        scrittura su file;

        /*Si arresta l'esecuzione dei router, impostandone il criterio di
         terminazione.*/
        for (int i=0; i<this.net.routersnr; i++)
              this.net.routers.get(i).run=false;
    }
    synchronized (this)
    {
        //Risvegliamo i router in attesa.
        this.notifyAll();
    }
}
```

Essendo la scrittura su file estremamente documentata, si è volontariamente omesso il codice per incrementare la leggibilità del metodo.

Per concludere analizziamo i tre metodi richiamati da ciascun router: *nextNodeRouting(Router)*, per instradare i pacchetti sull'interfaccia di uscita, *distributePackets(Router)*, per inserire pacchetti dall'esterno e *updateAnalysisStatistics(Router)*, per memorizzare il contributo sui consumi energetici globali.

```
private void nextNodeRouting(Router src)
{
   Packet p=null;
   Link l=null;

   /*Ogni nodo può inviare al massimo maxPPS pacchetti nel quanto, quindi,
   poiché non ammettiamo il drop, gli eventuali pacchetti in eccesso rimangano
   bufferizzati nella coda di uscita.*/
   ArrayList<Packet> temp=new ArrayList<Packet>();

   for (int i=0; i<src.outputBufferToRoute.size(); i++)
   {
       p = src.outputBufferToRoute.get(i);
      if(i<src.maxPPS)
      {
          //Ricerco nella tabella di routing il prossimo hop
          l = net.cp.lookUp(src.ID, p.destination);
          //Per sicurezza, se non esiste il link droppo
          if(l!=null)
          {
              this.fullDeliveryCost+=l.getEnergyCost();
              l.b.waitinigInputBuffer.add(p);
          }
      }
      else
      {
         temp.add(p);
      }
   }
   src.outputBufferToRoute=temp;
}
```

```
private void distributePackets(Router r)
{
   int packets=this.packetDistributor.get(r.ID).size();
   Packet p;
   /*Finchè vi sono pacchetti dall'esterno, li inseriamo nella coda di ingresso
   garantendo un throughput minore o uguale a networkThroughput.*/
   while(packets>0 && r.waitinigInputBuffer.size()<this.networkThroughput)
   {
       p=this.packetDistributor.get(r.ID).remove(0);
       r.waitinigInputBuffer.add(p);
       packets--;
   }
}
```

```
void updateAnalysisStatistics(Router temp)
{
   this.totalIdleCost+=temp.idleCost;
   this.analysisCost+=temp.analysisCost;
   this.fullDeliveryCost+=temp.routingCost;
   this.allAnalizedPacket+=temp.analyzedThisTurn;
}
```

5.2 Impostazione dei parametri

Prima di procedere con le simulazioni, è utile spendere alcune parole sulla scelta dei parametri più importanti.

Per iniziare analysisPktCost e routingPktCost, sono posti rispettivamente a 5.5 e 1. Per determinarli abbiamo utilizzato due pubblicazioni passate, [21] e [22]. Nel primo si confronta il throughput del solo routing con quello del routing più analisi, deducendo che nella prima configurazione si raggiunge un valore 8.76 volte maggiore rispetto alla seconda. Da ciò si può ricavare, con pochi semplici passaggi, che l'operazione di instradamento ha un costo pari a 1 mentre l'analisi di 7.76. Tali risultati fanno però riferimento a Snort quindi per riportaci a Suricata, il nostro IPS di studio, abbiamo utilizzato i dati raccolti in [22] che riportano un rate di 33731 pacchetti/secondo di Suricata contro 20090 pacchetti/secondo di Snort. Da questo rapporto sono quindi stati dedotti i valori:

$$routingPktCost = 1$$

$$analysisPktCost = avgAnalysisPktCost + routingPktCost = 4.62 + 1 = 5.62$$

Si sottolinea che il costo di analisi per pacchetto è ottenuto dalla somma del valor medio del consumo della sola analisi e del costo di routing, ossia è "sovrastimato" in quanto si considera che tutti i pacchetti analizzati, anche quelli malevoli, siano instradati. Tuttavia si può dedurre che i valori "reali" non saranno molto distanti poiché se un pacchetto è buono il tempo richiesto per il controllo sarà inferiore ma dovrà essere effettuato il routing, viceversa l'analisi sarà più onerosa ma non si avrà l'instradamento.

Definite queste quantità, abbiamo stabilito il valore di energyGood e energyBad. Tale distinzione è necessaria per una analisi più accurata, infatti i consumi derivanti dal controllo del pacchetto dipendono dalla sua classe di appartenenza. Nello specifico, come già illustrato nel capitolo 3, Suricata ha un meccanismo di pre-matching che permette di determinare in tempi brevi una parte dei pacchetti buoni; i restanti e i pacchetti malevoli subiscono un'ulteriore elaborazione di confronto con le regole presenti nel database delle firme. Abbiamo quindi assunto un rapporto 1 a 2 tra i due casi, quindi essendo il costo medio di analisi 4.62, si ottiene che:

$$energyGood = 3.08$$

$$energyBad = 6.16$$

Per semplicità di calcolo, considerando che i dati utilizzati sono stati dedotti da prove empiriche e quindi soggette ad errore, abbiamo impostato per le simulazioni i seguenti valori:

$$energyGood = 3$$

$$energyBad = 6$$

$$analysisPktCost = 4.5 + 1 = 5.5$$

Si ricorda ancora una volta che tutti i consumi non sono espressi in Watt ma in funzione del costo di routing di un pacchetto. Questa non è una limitazione, infatti, noto il consumo per effettuare l'instradamento di un pacchetto, basterà moltiplicare i nostri risultati per la suddetta quantità.

Inoltre abbiamo impostato i valori degli operandi della formula di maxPPSIPS, ottenuta da:

$$maxPPSIPS = \left\lfloor (1 - load) \cdot \frac{T}{timePrcIPS} \cdot threshold \right\rfloor \quad (1)$$

In particolare abbiamo optato per

$$threshold = 0.75$$

$$\frac{T}{timePrcIPS} = \frac{\sum_{i=1}^{n} maxPPS_i}{analysisPktCost - 1} \quad (2)$$

La soglia è stata determinata in modo empirico ma si rimanda a lavori futuri per eventuali scelte più accurate. Per quanto riguarda la formula (2) è indubbiamente necessario un approfondimento: avendo espresso tutti i parametri in funzione del routing di un pacchetto era inevitabile utilizzare lo stesso formalismo. Di conseguenza assunto che in T si effettua il routing del numeratore al secondo membro e che il tempo richiesto per l'analisi di un pacchetto è 4.5 volte superiore del routing, si ottiene la formula in questione.

Aprendo una breve parentesi sulla durata del quanto, esso deve essere sufficientemente piccolo per permettere una correzione dell'errore di stima il più possibile "immediata" per limitare i ritardi in trasmissione introdotti in caso di traffico previsto inferiore a quello reale. Tuttavia non può essere troppo ridotto poiché, per evitare di introdurre solo overhead, si deve garantire che il tempo di calcolo di maxPPSIPS sia trascurabile rispetto a T. Avendo misurato in modo empirico che il tempo richiesto dalle nostre strategie per prevedere il traffico è nell'ordine dei microsecondi, su un'architettura non particolarmente performante, una possibile scelta, che soddisfa i precedenti vincoli, potrebbe essere un quanto di durata un millisecondo.

Per concludere, merita un approfondimento la decisione di trascurare gli idleCost. Tale scelta non falsifica i nostri risultati in quanto è un consumo che deve essere sostenuto a prescindere dall'applicazione o meno del nostro modulo. In altri termini per ottenere dai nostri risultati i consumi effettivi basterà aggiungere il *totalIdleCost* di ciascun router. Definiamo totalIdleCost del router i-esimo il consumo sostenuto nei *k* istanti di attività:

$$totalIdleCost_i = k \cdot idleCost_i \quad (3)$$

Tuttavia è doveroso sottolineare che il consumo richiesto dal router per le sole attività di gestione, ossia l'idleCost, impatta sul costo totale in modo rilevante. Dati alla mano, [23], un router dual CPU Intel XEON Dual Core 3.0 GHz che non compie alcuna operazione consuma 207 W/h e sale a 248 W/h per un throghput di 1040 kpps. Per esplicitare ulteriormente la nostra soluzione, facendo riferimento ai dati sopracitati, il consumo totale non è 248 W/h ma la differenza tra questo valore e il costo di idle, ossia 41 W/h.

Capitolo 6

Risultati

Definito il simulatore, riportiamo in questo capitolo i risultati delle prove effettuate, improntate a confermare quanto supposto in fase di modellazione. Come abbiamo sottolineato più volte, il nostro modello di analisi distribuita attraverso IPS è indicato per le reti di trasporto. Infatti tanto più è lungo il percorso dalla sorgente alla destinazione, tanto maggiore è il risparmio energetico, soprattutto con una percentuale di pacchetti malevoli significativa.

Riferendoci ancora una volta a [21], utilizziamo per prove una topologia a stella che rappresenta una struttura "reale" di peering in cui le reti di Internet Service Provider diversi si congiungono in un Internet Exchange Point[20]. Abbiamo optato per una profondità, ossia il numero di rami, pari a sei.

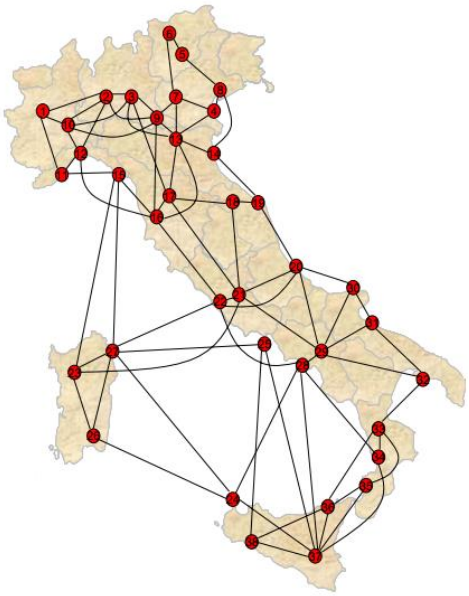

Figura 6.1: Rete di trasporto Telecom Italia

Di conseguenza, a prescindere dal numero di nodi, vi sono sei rami che si diramano da quello centrale. Avendo in totale sei nodi foglia, i pacchetti sono destinati a uno di essi e trasmessi dai restanti cinque. Ciò implica che, eccetto il nodo centrale, ciascun router ha due interfacce, da una riceve dall'altra trasmette. A differenza delle apparenze, la soluzione con un'unica destinazione non è limitativa se si impone, come nelle nostre simulazioni, maxPPS della strategia pari alla capacità massima dell'interfaccia in ricezione. In caso di più destinazioni, come illustrato nel capitolo 4, tale valore sarebbe la somma delle capacità di tutte le interfacce

[20]Il IXP rappresenta il punto di interconnessione tra traffici di ISP diversi, ossia il centro della nostra stella.

del router e si otterrebbero i nostri stessi risultati. Nelle simulazioni si quantificheranno, con i parametri descritti in 6.1, le performance delle strategie di ottimizzazione energetica ideate confrontandole con quelle dell'attuale comportamento delle reti, ossia il routing con analisi nei nodi terminali. Infine abbiamo optato, nella strategia ai minimi quadrati, per uno storico di cinque campioni. Si rimanda a lavori futuri lo studio del comportamento della soluzione di stima al variare del suddetto parametro.

Per concludere è doveroso ricordare ancora una volta che i costi riportati sono espressi in funzione dell'instradamento di un pacchetto.

6.1 Le prove

Di seguito si riporta la descrizione delle prove effettuate, ciascuna con i propri obiettivi. Come preannunciato nella sezione introduttiva, i risultati sono espressi in funzione di diversi parametri, di importanza dipendente dall'esperimento:

1. Costi globali. A prescindere da dove venga effettua, all'interno o all'esterno della rete, l'analisi di tutti i pacchetti è un'operazione inevitabile. Nello specifico, per controllare l'intero traffico è richiesto un consumo energetico pari a *totalAnalysisCost* che dovrà essere sempre sostenuto: nel routing tale costo è interamente pagato all'esterno, con la nostra soluzione una parte, tutto nel caso di 100% di pacchetti verificati, incide sul consumo interno della rete. I costi globali dell'i-esima soluzione sono calcolabili con la seguente formula:

$$globalCost_i = totalAnalysisCost + deliveryCost_i \quad (1)$$

2. Risparmio energetico percentuale sul solo instradamento. Tale valore è il risultato della formula:

$$\%risparmioSulRouting = \frac{globalCost_{Routing} - globalCost_{IPS}}{globalCost_{Routing}} \cdot 100 \quad (2)$$

   Seppur banale, è evidente che a valori percentuali negativi corrispondono maggiori consumi rispetto al routing, viceversa dei risparmi.

3. Costi di analisi e trasmissione all'interno della rete di trasporto. Si sottolinea che i consumi derivanti dal processo di analisi, *analysisCost*, si riferiscono ai pacchetti controllati all'interno della rete, quindi:

$$analysisCost \leq totalAnalysisCost \; \forall \; soluzione \quad (3)$$

4. Consumo percentuale dell'analisi e della trasmissione sul totale delle risorse impiegate all'interno della rete di trasporto, rispettivamente:

$$\%AnalysisCost = \frac{analysisCost}{totalCost} \cdot 100 \quad (4)$$

$$\%DeliveryCost = \frac{deliveryCost}{totalCost} \cdot 100 \quad (5)$$

5. Valore medio e massimo tra il numero medio di pacchetti bufferizzati in ciascun router. Questa misura, per svincolarsi dalle prestazioni del singolo nodo, è espressa in percentuale ossia rapportata alla massima capacità di gestione del router. Nel dettaglio, si calcola il valor medio percentuale per il generico router *i* che ha operato per *n* quanti. Formalmente, definito $bufferSize_{i,k}$ il numero di pacchetti memorizzati nell'intervallo *k* dal generico router i, si ha che:

$$avgBuffer_i = \frac{1}{n} \cdot \sum_{k=1}^{n} bufferSize_{i,k} \quad \forall\, router \quad (6)$$

$$\%avgBuffer_i = \frac{avgBuffer_i}{maxPPS_i} \cdot 100 \quad \forall\, router \quad (7)$$

   Infine si calcola il massimo e il valor medio tra tutte le percentuali. Per completezza si riporta in appendice A il valor percentuale medio di bufferizzazione di ciascun router da cui sono stati calcolati i parametri definiti poco sopra.

6. Percentuale di pacchetti controllati all'interno della rete. Si ricorda ancora una volta che il nostro modulo massimizza la suddetta quantità ma non garantisce una copertura totale. Infatti in condizioni di traffico intenso si privilegia ragionevolmente il routing all'analisi e quindi alcuni pacchetti possono arrivare a destinazione non ancora controllati. In quest'ultimo caso analysisCost sarà ovviamente inferiore del totalAnalysisCost, ossia una parte del consumo energetico richiesto per l'analisi sarà sostenuto all'esterno della rete.
7. Ritardo in trasmissione rispetto al routing. Per ciascun pacchetto si memorizza l'informazione relativa al numero totale di quanti in cui è stato bufferizzato, ossia il suo ritardo. Parliamo di ritardo rispetto al routing in quanto l'instradamento non bufferizza alcun pacchetto, altrettanto non si può affermare per il nostro modello. Si ricorda ancora una volta che ciascun router che esegue il nostro modulo può ritardare la trasmissione di un pacchetto in caso di stime errate al "ribasso", ossia se il traffico stimato per l'intervallo è inferiore a quello reale. Nonostante ciò, garantiamo una posticipazione nell'invio del pacchetto al successivo nodo nel cammino ottimo al massimo di un quanto, poiché se esso è stato memorizzato per insufficienza di risorse verrà instradato

con priorità nell'intervallo successivo. Come risultato delle prove secondo questo parametro riportiamo i valori medi e massimi di ciascuna strategia. Inoltre, in appendice B, le frequenze relative per ciascun valore di ritardo introdotto dalla nostra soluzione e l'evoluzione temporale degli stessi.

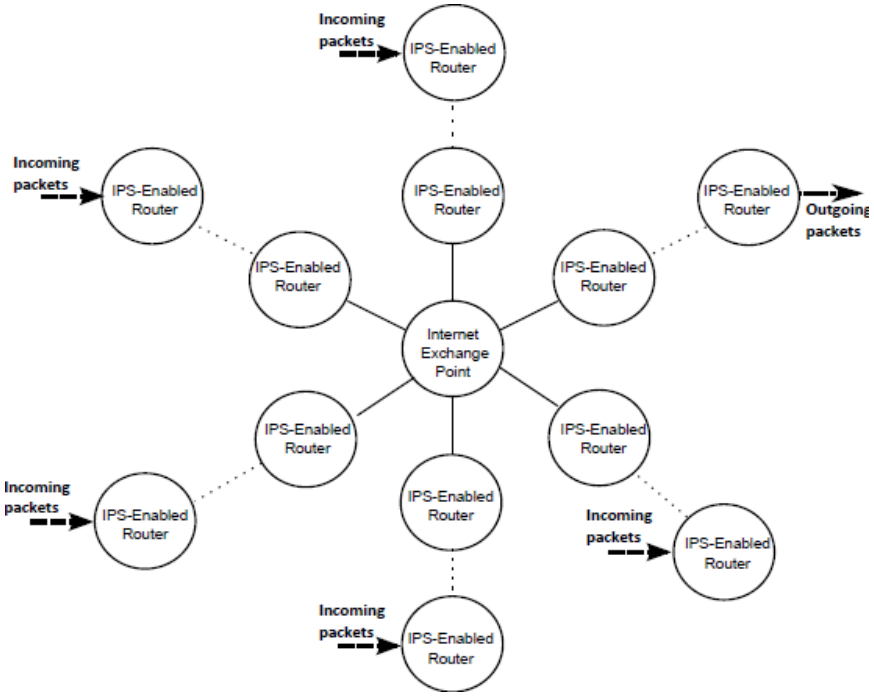

Figura 6.2: Topologia di rete con router utilizzanti il nostro modello

Le prove verranno effettuate su due reti aventi la stessa topologia ma dimensione differente. Infatti, come già detto in passato, il nostro modello è sensibile a diversi fattori tra cui la lunghezza dei percorsi medi, ossia aumenta significativamente le proprie performance con il crescere della dimensione della rete. Senza essere troppo ripetitivi è evidente che per cammini brevi, ossia composti da pochi nodi, anticipare lo scarto di un pacchetto malevolo permette di "risparmiare" un numero irrisorio di operazioni di routing e invio sul canale. Inoltre, avendo a disposizione poche unità di analisi, la probabilità di rilevare un'attività malevola sarà bassa anche in presenza di percentuali di pacchetti bad relativamente alte. Viceversa con una rete di grande dimensione, da una parte avendo più nodi, cioè più punti di analisi, è possibile controllare un maggior numero di pacchetti, dall'altra, essendo i percorsi più lunghi, accelerare la scoperta dei malevoli porta a un significativo risparmio di risorse della rete.

Per concludere, abbiamo impostato un totale di 100 mila pacchetti, suddiviso poi nelle foglie con distribuzione uniforme illustrata nel paragrafo 5.1.7. La scelta della distribuzione è obbligata in quanto i risultati delle prove devono essere confrontabili, ossia derivanti dalle stesse condizioni di rete. Di conseguenza, dei tre metodi di distribuzione implementati, la

uniforme è l'unica deterministica, cioè la sola che garantisce l'assegnamento, in ciascuna prova, dello stesso numero di pacchetti alle sorgenti, più precisamente 20 mila a ciascuna. Inoltre ciascuna scheda di rete dei router ha un numero massimo di pacchetti gestibili, ossia maxPPS, pari a 100. Il numero di pacchetti trasmessi per intervallo da ciascuna sorgente, che influenza implicitamente il traffico in ciascun ramo, è determinato dal parametro *networkThroughput*. Non ammettendo i drop della scheda fisica, tale valore è definito come percentuale di maxPPS garantendo che:

$$networkThroughput \leq maxPPS \quad (8)$$

Infine si sottolinea che tale scelta non rappresenta alcuna limitazione in quanto si mantiene un rapporto "reale" tra il traffico di rete e la capacità del nodo. Infatti aumentare il numero di pacchetti totale influenza unicamente la durata della simulazione.

6.1.1 Variazione percentuale bad

In questa prova, fissando il networkThroughput al 60% di maxPPS, si varia la percentuale di pacchetti malevoli. Per questo esperimento abbiamo determinato il seguente insieme di percentuali da valutare nei test:

$$\%bad = \{\, 0 \,;\, 0.1 \,;\, 0.6 \,;\, 1 \,;\, 2 \,;\, 5 \,;\, 10 \,;\, 20 \,;\, 30 \,;\, 40 \,;\, 50 \,;\, 60 \,;\, 70 \,;\, 80 \,;\, 90 \,;\, 100\}$$

Abbiamo optato per provare più valori per percentuali comprese tra 0 e 10 in quanto più probabili nella realtà. Inoltre, pur non essendo realistiche percentuali di traffico malevolo superiori al 50%, abbiamo scelto di verificare anche valori più elevati per illustrare in modo più evidente il legame tra la quantità di traffico malevolo e l'efficienza della nostra soluzione. Infatti l'obiettivo della simulazione è dimostrare che il risparmio energetico derivante dall'applicazione del nostro modulo di analisi distribuita attraverso IPS è tanto più evidente tanto più aumenta la sopracitata percentuale. In particolare il task di analisi è più oneroso rispetto a quello di instradamento, ma è un consumo accettabile in ottica di risparmio delle risorse di rete: anticipare lo scarto dei pacchetti malevoli ne evita la trasmissione nei nodi successivi per raggiungere la destinazione.

6.1.2 Variazione networkThroughput

In questa prova, fissata la percentuale di pacchetti malevoli al 6%, si varia il networkThroughput. Per simulare diverse condizioni di carico della rete, abbiamo determinato il seguente insieme di percentuali di maxPPS:

$$networkThroughput = \{\, 10\% \,,\, 20\% \,,\, 30\% \,,\, 40\% \,,\, 50\% \,,\, 60\% \,,\, 70\% \,,\, 80\% \,,\, 90\% \,,\, 100\% \,\}$$

L'obiettivo della simulazione è dimostrare che, data l'adattività delle nostre strategie, tanto minore è il carico della scheda, tanto maggiore sarà il numero di pacchetti analizzati e, di conseguenza, il risparmio energetico globale. Si sottolinea che tale tendenza non è legata in alcun modo alla percentuale di pacchetti malevoli in circolo nella rete, tale parametro influenzerà unicamente la percentuale di risparmio energetico. Riassumendo, a prescindere dalla quantità di traffico generata da un ipotetico attaccante, si avranno risparmi decrescenti all'aumentare del traffico in quanto tanto più la rete è carica tanto più si converge all'attuale comportamento delle reti.

6.1.3 Variazione del profilo di traffico

In questa prova si analizza il comportamento del nostro modello di analisi distribuita al variare delle condizioni di rete. Infatti, come illustrato nel capitolo 4, le nostre strategie di ottimizzazione energetica sono adatte per determinati contesti ma non per altri. Abbiamo quindi supposto tre diversi test: traffico in ingresso costante, variabile e altamente variabile. Per confrontare i risultati abbiamo inviato lo stesso numero di pacchetti per periodo, ossia a prescindere dal profilo di traffico si inviano 480 pacchetti su ciascuna foglia con periodo pari a otto quanti. Nello specifico, abbiamo utilizzato i seguenti "pattern":

- Costante: traffico fisso a 0.6·maxPPS.
- Variabile: Crescita lineare del 20% da 0.2·maxPPS a maxPPS e decrescita al valore iniziale. Di conseguenza l'intervallo di ripetizione è pari a 8 quanti.

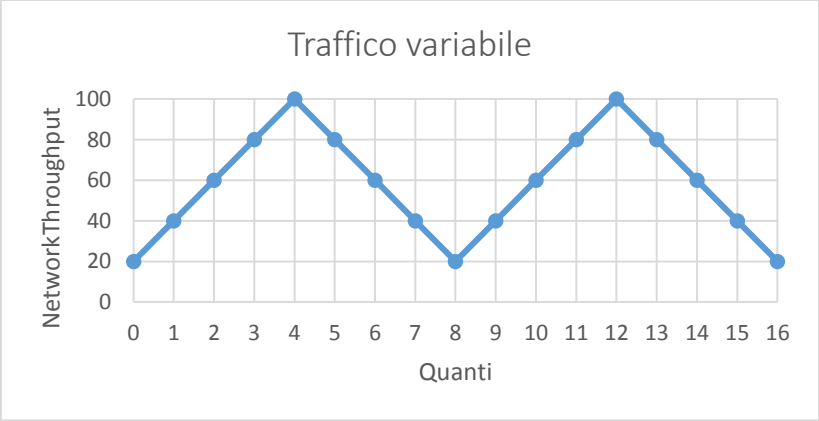

Figura 6.3: Profilo di traffico variabile

- Altamente variabile: Crescita da 0.4·maxPPS a 0.8·maxPPS e decrescita al valore iniziale. Di conseguenza l'intervallo di ripetizione è pari a 2 quanti.

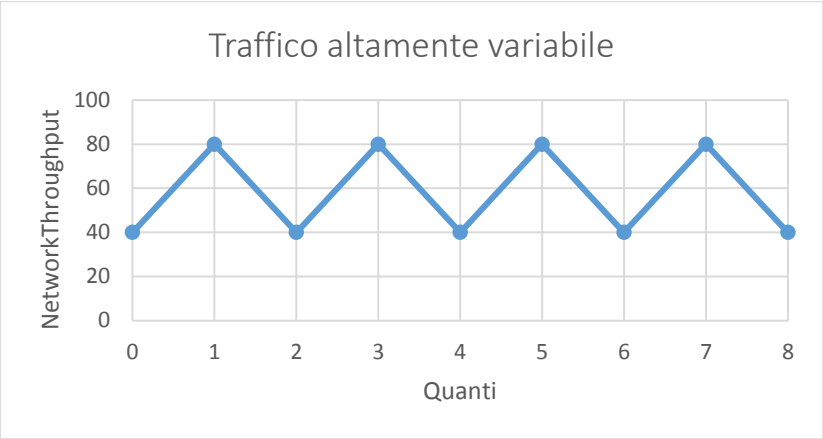

Figura 6.4: Profilo di traffico altamente variabile

L'obiettivo della simulazione è dimostrare che al variare del profilo di traffico le strategie compiono differenti errori di stima che si riflettono sul numero di pacchetti bufferizzati nell'intervallo e quindi sui ritardi in trasmissione. Nel dettaglio, in linea con quanto mostrato nei grafici del capitolo 4, supponiamo un buon comportamento per traffici costanti, viceversa per profili variabili. Per concludere è doveroso ricordare che i profili sopracitati sono rispettati nelle sorgenti ma non è garantito che siano mantenuti all'interno della rete. Per limitare ciò, i test sono effettuati con una percentuale di pacchetti malevoli pari allo 0% in modo da evitare lo scarto di pacchetti e quindi una variazione nel throughput. Rimane però possibile la modifica del profilo a causa delle inevitabili bufferizzazioni derivanti da stime errate.

6.2 Rete di media dimensione

Per simulare una rete di media dimensione, abbiamo considerato 150 router. Data la topologia a stella e l'invio di pacchetti dall'esterno solo sulle foglie, possiamo dedurre che il percorso massimo di un pacchetto è pari a 50 hop. L'aggettivo massimo è necessario in quanto per il routing sarà effettivamente pari a questo valore, viceversa sarà inferiore per i pacchetti malevoli analizzati, e di conseguenza scartati, lungo il cammino verso la destinazione.

6.2.1 Risultati prova 1

Di seguito sono elencati i costi totali, somma tra quelli di analisi e di trasmissione, sostenuti all'interno della rete di trasporto ricavati applicando le nostre strategie al variare della percentuale di pacchetti malevoli.

%bad	totalCost Routing	totalCost Constant	totalCost Lineary	totalCost OLS[21]
0	5080000	5380000	5378365	5377747
0.1	5080000	5375380	5374278	5372942
0.6	5080000	5353465	5352852	5351783
1	5080000	5336020	5334978	5335710
2	5080000	5291780	5292224	5292400
5	5080000	5159080	5159590	5159660
10	5080000	4937925	4938435	4938505
20	5080000	4495580	4496090	4496160
30	5080000	4053375	4053885	4053955
40	5080000	3610925	3611435	3611505
50	5080000	3168580	3169090	3169160
60	5080000	2719135	2719505	2719490
70	5080000	2269305	2269680	2269660
80	5080000	1819815	1820170	1820165
90	5080000	1370135	1370505	1370490
100	5080000	920555	921275	919880

Tabella 6.1: Consumi totali prova 1 - reti medie dimensioni

Come supposto già all'interno della rete, poco sopra al 5% di traffico malevolo, il nostro modello di analisi distribuita attraverso Intrusion Prevention System permette un risparmio energetico sull'attuale funzionamento dei router, il cui costo, riportato nella colonna routing, rimane ovviamente invariato nei test. Anche l'aumento dei consumi per percentuali basse è del tutto in linea con le nostre previsioni poiché l'analisi è un'operazione più onerosa dell'instradamento quindi la maggior parte dei pacchetti, essendo pochi quelli scartati in quanto malevoli, percorrerà ugualmente tutto il cammino fino a destinazione. Di conseguenza analizzandoli, per poi trasmetterli, aumenta il consumo energetico senza trarre nessun risparmio in trasmissione. Un'altra informazione interessante, riportata in tabella, sono le percentuali sul costo totale delle singole componenti di analisi e di trasmissione.

%bad	% energy analysis over Total Constant	% energy delivery over Total Constant	% energy analysis over Total Linear	% energy delivery over Total Linear	% energy analysis over Total OLS	% energy delivery over Total OLS
0	5.58	94.42	5.55	94.45	5.54	94.46
0.1	5.59	94.41	5.57	94.43	5.54	94.46
0.6	5.64	94.36	5.63	94.37	5.62	94.38
1	5.68	94.32	5.66	94.34	5.68	94.32
2	5.78	94.22	5.78	94.22	5.78	94.22
5	6.11	93.89	6.11	93.89	6.11	93.89
10	6.68	93.32	6.68	93.32	6.68	93.32
20	8.01	91.99	8.01	91.99	8.01	91.99
30	9.62	90.38	9.62	90.38	9.62	90.38

[21]Ordinary Least Square, ossia minimi quadrati.

40	11.63	88.37	11.63	88.37	11.63	88.37
50	14.20	85.80	14.20	85.80	14.20	85.80
60	17.65	82.35	17.65	82.35	17.65	82.35
70	22.47	77.53	22.47	77.53	22.47	77.53
80	29.67	70.33	29.67	70.33	29.67	70.33
90	41.60	58.40	41.59	58.41	41.59	58.41
100	65.18	34.82	65.13	34.87	65.23	34.77

Tabella 6.2: Contributo analisi e trasmissione prova 1 – reti medie dimensioni

Si può osservare che all'aumento della percentuale di pacchetti malevoli corrisponde una crescita del "peso" del costo di analisi su quello totale. Anche questo comportamento conferma le nostre aspettative: scartando più pacchetti, da una parte il costo di trasmissione diminuisce, dall'altra, ciascun router osserverà un minor traffico in ingresso permettendo al nostro modulo di assegnare più pacchetti all'analisi, anticipandone di conseguenza la scoperta. Infine si rimarca che il routing ha consumi inferiori al nostro modello all'interno della rete di trasporto per basse percentuali di traffico malevolo ma ciò si ripercuote in consumi all'esterno della stessa in quanto i pacchetti dovranno comunque essere analizzati. Possiamo quindi concludere, come illustrato in tabella, che globalmente la nostra soluzione non introduce consumi energetici aggiuntivi, ma contribuisce anzi ad abbatterli.

%bad	totalAnalysis Cost	globalCost Routing	% energy saved Constant	% energy saved Lineary	% energy saved OLS
0	300000	5380000	0,000	0.000	0.000
0.1	300300	5380300	0.091	0.091	0.091
0.6	301800	5381800	0.525	0.531	0.539
1	303000	5383000	0.873	0.877	0.878
2	306000	5386000	1.749	1.740	1.738
5	315000	5395000	4.373	4.363	4.362
10	330000	5410000	8.726	8.716	8.715
20	360000	5440000	17.360	17.351	17.350
30	390000	5470000	25.898	25.889	25.887
40	420000	5500000	34.347	34.337	34.336
50	450000	5530000	42.702	42.693	42.691
60	480000	5560000	51.095	51.088	51.088
70	510000	5590000	59.404	59.397	59.398
80	540000	5620000	67.619	67.613	67.613
90	570000	5650000	75.750	75.743	75.743
100	600000	5680000	83.793	83.780	83.805

Tabella 6.3: Risparmio energetico prova 1 – reti medie dimensioni

6.2.2 Risultati prova 2

Nella tabella 6.4 sono elencati i costi di analisi e di trasmissione sostenuti all'interno della rete, ricavati applicando le nostre strategie di ottimizzazione energetica al variare del traffico in ingresso. Come supposto, data l'adattività della nostra formula, tanto minore è il numero di

pacchetti ricevuti sull'interfaccia di ingresso di un router, tanti più saranno assegnati, dal nostro modulo, al processo di analisi. In altri termini, se la scheda non è particolarmente carica la maggior parte dei pacchetti ricevuti verrà analizzata, anticipando così la scoperta dei malevoli con conseguente risparmio in risorse di rete. Viceversa, ossia in condizioni di traffico intenso, si privilegerà l'instradamento al controllo quindi i pacchetti cattivi effettueranno un tratto più lungo, se non tutto, del cammino da sorgente a destinazione.

% sul Throughput massimo	Routing	analysis Cost Constant	delivery Cost Constant	analysis Cost Linear	delivery Cost Linear	analysis Cost OLS	delivery Cost OLS
10	5080000	318000	4775225	318000	4775225	318000	4775225
20	5080000	318000	4777045	318000	4777040	318000	4777050
30	5080000	318000	4780055	318000	4780045	318000	4780000
40	5080000	318000	4784265	318000	4784285	318000	4784065
50	5080000	318000	4789475	318000	4789800	318000	4789100
60	5080000	318000	4796975	318000	4797485	318000	4797555
70	5080000	318000	4808670	317955	4808885	317889	4813797
80	5080000	303753	4847973	288087	4830744	296121	4868458
90	5080000	97263	4986527	104970	4877566	93630	4995802
100	5080000	17625	5006327	9117	5041972	16446	5009627

Tabella 6.4: Contributo analisi e trasmissione prova 2 – reti medie dimensioni

A supporto di tale affermazione basti notare che all'aumento del carico della rete segue una crescita dei costi di trasmissione poiché i pacchetti malevoli sono scartati più avanti nel percorso verso la destinazione. Un'altra informazione importante è la percentuale di pacchetti controllati all'interno della rete che spiega, tra l'altro, i costi di analisi invariati fino a circa il 70% di maxPPS. Infatti fino a tale traffico in ingresso, il nostro modello di analisi distribuita attraverso IPS è in grado di controllare tutti i pacchetti all'interno della rete stessa.

% sul Throughput massimo	%checked Constant	%checked Linear	%checked OLS
10	100	100	100
20	100	100	100
30	100	100	100
40	100	100	100
50	100	100	100
60	100	100	100
70	100	99.985	99.971
80	96.656	90.061	93.644
90	30.058	29.758	29.102
100	3.919	2.026	3.655

Tabella 6.5: Percentuale pacchetti controllati nella rete prova 2 – reti medie dimensioni

Per completezza si riportano i risparmi globali ottenuti in queste condizioni considerando che il totalAnalysisCost è pari a 318000 per tutti i test[22].

% sul Throughput massimo	% energysaved Constant	% energysaved Linear	% energysaved OLS
10	5.646	5.646	5.646
20	5.612	5.612	5.612
30	5.557	5.557	5.558
40	5.479	5.478	5.482
50	5.382	5.376	5.389
60	5.243	5.234	5.232
70	5.026	5.023	4.932
80	4.298	4.618	3.919
90	1.732	3.750	1.560
100	1.365	0.704	1.304

Tabella 6.6: Risparmio energetico prova 2 – reti medie dimensioni

6.2.3 Risultati prova 3

Di seguito si elenca, per ciascuna strategia, il valor medio e massimo tra i valor medi della dimensione dei buffer di ciascun router ottenuti per condizione di traffico costante. Inoltre si riporta la percentuale di pacchetti controllati, infatti un numero ridotto di pacchetti bufferizzati potrebbe comportare una percentuale di verificati bassa quindi più analisi all'esterno.

	Routing	Constant	Linear	OLS
Max %avgBuffer	0	0.171	0.772	0.494
Media %avgBuffer	0	0.014	0.121	0.034
% checked	0	100	99.455	99.249

Tabella 6.7: Dimensione massima e media dei buffer e percentuale controllati test 1 prova 3 – reti medie dimensioni

Si può notare che le tre strategie memorizzano per l'elaborazione del quanto successivo una percentuale estremamente bassa di pacchetti. Infatti, in caso di traffico costante le tre soluzioni stimano circa lo stesso valore quindi anche l'errore di previsione, che determina la dimensione del buffer, sarà lo stesso. Infine si evince che la strategia lineare, più sensibile alle piccole fluttuazioni, memorizza un numero di pacchetti leggermente maggiore rispetto alle altre.

Come supposto i risultati relativi alla dimensione dei buffer in condizioni di traffico variabile sono diversi.

	Routing	Constant	Linear	OLS
Max %avgBuffer	0	4.156	3.269	38.098
Media %avgBuffer	0	0.229	0.214	12.396
% checked	0	92.333	60.117	90.067

Tabella 6.8: Dimensione massima e media dei buffer e percentuale controllati test 2 prova 3 – reti medie dimensioni

[22] Il costo totale di analisi è lo stesso per tutti i test in quanto la percentuale di pacchetti malevoli è fissa al 6%.

Si può notare che la strategia Ordinary Least Square ha una percentuale significativamente più alta delle altre. Tale comportamento è in linea con le considerazioni descritte nel capitolo 4, infatti lo storico caratterizzante la strategia la penalizza in fase di stima.

Infine si può osservare che in condizioni di traffico altamente variabile, come da ipotesi, la strategia lineare compie grandi errori di stima che si ripercuotono in un elevato numero di pacchetti bufferizzati. Viceversa la OLS riesce a inseguire in modo più accurato il pattern.

	Routing	Constant	Linear	OLS
Max %avgBuffer	0	10.468	31.710	8.000
Media %avgBuffer	0	0.354	6.950	0.303
% checked	0	100	99.981	99.363

Tabella 6.9: Dimensione massima e media dei buffer e percentuale controllati test 3 prova 3 – reti medie dimensioni

Per concludere si riportano i dati relativi al valor massimo e medio del numero di quanti di ritardo introdotto dalle nostre soluzioni, che sono strettamente correlati alla dimensione dei buffer.

Ritardo medio	Routing	Constant	Linear	OLS
Costante	0	0.007	0.061	0.017
Variabile	0	0.115	0.107	6.279
Altamente variabile	0	0.177	3.502	0.151

Tabella 6.10: Ritardo medio prova 3 – reti medie dimensioni

Max ritardo	Routing	Constant	Linear	OLS
Costante	0	2	3	3
Variabile	0	2	1	8
Altamente variabile	0	2	4	3

Tabella 6.11: Ritardo massimo prova 3 – reti medie dimensioni

Si rimanda all'appendice B per le frequenze relative e per l'andamento temporale dei ritardi.

6.3 Rete di grande dimensione

Per simulare una rete di grande dimensione, abbiamo considerato 300 router. Data la topologia a stella e l'invio di pacchetti dall'esterno solo sulle foglie, possiamo dedurre che il percorso massimo di un pacchetto è pari a 100 hop. L'aggettivo massimo è necessario in quanto per il routing sarà effettivamente pari a questo valore, viceversa sarà inferiore per i pacchetti malevoli analizzati, e di conseguenza scartati, lungo il cammino verso la destinazione.

6.3.1 Risultati prova 1

Di seguito sono elencati i costi totali ricavati all'interno della rete di trasporto applicando le nostre strategie al variare della percentuale di pacchetti malevoli. Dalla tabella si deduce che il

nostro modello permette un risparmio energetico sull'attuale funzionamento dei router a partire da una percentuale di bad compresa tra il 2 e il 5%. Le motivazioni sono già state espresse in 6.2.1.

%bad	Total Cost Routing	Total Cost Constant	Total Cost Lineary	Total Cost OLS
0	10080000	10380000	10378899	10378413
0.1	10080000	10370380	10369812	10367942
0.6	10080000	10323465	10323089	10322158
1	10080000	10286020	10285314	10285710
2	10080000	10191780	10192224	10192400
5	10080000	9909080	9909590	9909660
10	10080000	9437925	9438435	9438505
20	10080000	8495580	8496090	8496160
30	10080000	7553375	7553885	7553955
40	10080000	6610925	6611435	6611505
50	10080000	5668580	5669090	5669160
60	10080000	4719135	4719505	4719490
70	10080000	3769305	3769680	3769660
80	10080000	2819815	2820170	2820165
90	10080000	1870135	1870505	1870490
100	10080000	920555	921275	919880

Tabella 6.12: Consumi totali prova 1 – reti grandi dimensioni

Riportare i risultati ottenuti in termini di risparmio percentuale globale sul solo instradamento sottolinea ulteriormente i guadagni ottenibili grazie al nostro modello.

%bad	totalAnalysis Cost	globalCost Routing	% energy saved Constant	% energy saved Lineary	% energy saved OLS
0	300000	10380000	0.000	0.000	0.000
0.1	300300	10380300	0.096	0.096	0.095
0.6	301800	10381800	0.562	0.564	0.568
1	303000	10383000	0.934	0.936	0.937
2	306000	10386000	1.870	1.865	1.864
5	315000	10395000	4.675	4.670	4.669
10	330000	10410000	9.338	9.333	9.332
20	360000	10440000	18.625	18.620	18.619
30	390000	10470000	27.857	27.852	27.851
40	420000	10500000	37.039	37.034	37.033
50	450000	10530000	46.167	46.162	46.162
60	480000	10560000	55.311	55.308	55.308
70	510000	10590000	64.407	64.403	64.404
80	540000	10620000	73.448	73.445	73.445
90	570000	10650000	82.440	82.437	82.437
100	600000	10680000	91.381	91.374	91.387

Tabella 6.13: Risparmio energetico prova 1 – reti grandi dimensioni

Un'altra informazione interessante, riportata in tabella, sono le percentuali sul costo totale delle singole componenti di analisi e di trasmissione. Come precedentemente illustrato in 6.2.1, si può osservare che all'aumento della percentuale di pacchetti malevoli corrisponde una crescita del "peso" del costo di analisi su quello totale. Per concludere si può notare che in questa prova, ovviamente, le percentuali di consumo derivanti dal processo di analisi sulle risorse totali sono inferiori rispetto alla stessa in rete di medie dimensioni. Infatti, per ciascun test, il costo di analisi sarà lo stesso, o poco diverso se la percentuale dei pacchetti controllati nella rete è inferiore, ma altrettanto non si può dire della trasmissione il cui contributo sarà maggiore, dato che pacchetti buoni effettuano tutto il percorso da sorgente a destinazione.

%bad	% energy analysis over Total Constant	% energy delivery over Total Constant	% energy analysis over Total Linear	% energy delivery over Total Linear	% energy analysis over Total OLS	% energy delivery over Total OLS
0	2.890	97.110	2.880	97.120	2.875	97.125
0.1	2.896	97.104	2.890	97.110	2.873	97.127
0.6	2.923	97.077	2.922	97.078	2.918	97.082
1	2.946	97.054	2.941	97.059	2.946	97.054
2	3.002	96.998	3.002	96.998	3.002	96.998
5	3.179	96.821	3.179	96.821	3.179	96.821
10	3.497	96.503	3.496	96.504	3.496	96.504
20	4.237	95.763	4.237	95.763	4.237	95.763
30	5.163	94.837	5.163	94.837	5.163	94.837
40	6.353	93.647	6.353	93.647	6.353	93.647
50	7.938	92.062	7.938	92.062	7.938	92.062
60	10.171	89.829	10.171	89.829	10.171	89.829
70	13.530	86.470	13.529	86.471	13.529	86.471
80	19.150	80.850	19.148	80.852	19.148	80.852
90	30.479	69.521	30.473	69.527	30.473	69.527
100	65.178	34.822	65.127	34.873	65.226	34.774

Tabella 6.14: Contributo analisi e trasmissione prova 1 – reti grandi dimensioni

6.3.2 Risultati prova 2

Di seguito sono elencati i consumi relativi all' analisi e alla trasmissione ricavati all'interno della rete applicando le nostre strategie al variare del traffico in ingresso. Anche questa prova conferma quanto dedotto in 6.1.2: i consumi totali decrescono con il diminuire del throughput della rete poiché tanto minore è il traffico tanti più pacchetti possono essere analizzati.

% sul Throughput massimo	Routing	analysis Cost Constant	delivery Cost Constant	analysis Cost Linear	delivery Cost Linear	analysis Cost OLS	delivery Cost OLS
10	10080000	318000	9475225	318000	9475225	318000	9475225
20	10080000	318000	9477045	318000	9477040	318000	9477050
30	10080000	318000	9480055	318000	9480045	318000	9480000
40	10080000	318000	9484265	318000	9484285	318000	9484065
50	10080000	318000	9489475	318000	9489800	318000	9489100
60	10080000	318000	9496975	318000	9497485	318000	9497555
70	10080000	318000	9508670	318000	9508885	318000	9513600
80	10080000	317997	9539115	300738	9532544	315012	9550425
90	10080000	183207	9750254	143637	9565355	174726	9762137
100	10080000	35199	9785416	18609	9925088	34776	9790277

Tabella 6.15: Contributo analisi e trasmissione prova 2 – reti grandi dimensioni

A supporto di tale conclusione si riporta la percentuale di pacchetti analizzati all'interno della rete da cui si può osservare che, fino a quasi l'80% del throughput massimo, il nostro modello di analisi distribuita attraverso IPS è in grado di controllare tutti i pacchetti all'interno della rete stessa.

% sul Throughput massimo	%checked Constant	%checked Linear	%checked OLS
10	100	100	100
20	100	100	100
30	100	100	100
40	100	100	100
50	100	100	100
60	100	100	100
70	100	100	100
80	99.999	94.278	99.004
90	56.833	41.879	54.097
100	7.825	4.135	7.729

Tabella 6.16: Percentuale pacchetti controllati nella rete prova 2 – reti grandi dimensioni

Come desideravamo, tale soglia è ancora più alta di quella ricavata nella prova descritta nel paragrafo 6.1.2. Infatti in entrambe le prove, fissato il test, il modulo calcolerà lo stesso maxPPSIPS, assegnando lo stesso numero di pacchetti all'analisi. Ne consegue che, avendo qui percorsi più lunghi sarà possibile sfruttare più risorse garantendo quindi una percentuale maggiore di pacchetti analizzati.

Si riportano, in analogia alla simulazione per la rete di media dimensione, il risparmio globale percentuale sul solo instradamento ricordando che, per il 6% di pacchetti malevoli, totalAnalysisCost è sempre pari a 318000.

% sul Throughput massimo	% energy saved out of routing Constant	% energy saved out of routing Linear	% energy saved out of routing OLS
10	5.816	5.816	5.816
20	5.799	5.799	5.799
30	5.770	5.770	5.770
40	5.729	5.729	5.731
50	5.679	5.676	5.683
60	5.607	5.602	5.602
70	5.495	5.493	5.447
80	5.202	5.265	5.093
90	3.171	4.949	3.057
100	2.833	1.490	2.786

Tabella 6.17: Risparmio energetico prova 2 - reti grandi dimensioni

6.3.3 Risultati prova 3

Di seguito si elenca, per ciascuna strategia, il massimo e la media dei valor medi della dimensione dei buffer di ciascun router, ottenuti per condizione di traffico costante, variabile e altamente variabile. Si può osservare che i risultati massimi ottenuti sono identici a quelli rilevati nella stessa prova su rete di media dimensione, inferiori i valor medi.

	Routing	Constant	Linear	OLS
Max %avgBuffer	0	0.171	0.772	0.494
Media %avgBuffer	0	0.007	0.061	0.017
% checked	0	100	99.633	99.471

Tabella 6.18: Dimensione massima e media dei buffer e percentuale controllati test 1 prova 3 – reti grandi dimensioni

	Routing	Constant	Linear	OLS
Max %avgBuffer	0	4.156	3.269	38.098
Media %avgBuffer	0	0.114	0.107	6.198
% checked	0	99.994	61.965	99.340

Tabella 6.19: Dimensione massima e media dei buffer e percentuale controllati test 2 prova 3 – reti grandi dimensioni

	Routing	Constant	Linear	OLS
Max %avgBuffer	0	10.468	31.710	8
Media %avgBuffer	0	0.177	3.475	0.151
% checked	0	100	99.981	99.510

Tabella 6.20: Dimensione massima e media dei buffer e percentuale controllati test 3 prova 3 – reti grandi dimensioni

Tale comportamento deriva dall'implementazione del nostro modello: per costruzione tanto più un router è prossimo alla sorgente tanto più analizza tutti i maxPPSIPS pacchetti destinati al processo dell'Intrusion Prevention System. Infatti, a prescindere dalla lunghezza del cammino, la probabilità di trovare maxPPSIPS pacchetti non ancora controllati è, ovviamente, più alta nei primi nodi del percorso che negli ultimi. Inoltre un errore di stima al "ribasso", ossia un throughput previsto inferiore a quello effettivo e conseguente eccessivo maxPPSIPS, può portare ad un overbooking delle risorse con memorizzazione nel buffer. Combinando insieme

queste considerazioni, si può dedurre che il buffering dei pacchetti, in caso di stime errate, è più probabile nei primi nodi del cammino in cui si hanno molti pacchetti ancora da controllare.

Router	%avg Buffer Constant 150	%avg Buffer Constant 300	%avg Buffer Linear 150	%avg Buffer Linear 300	%avg Buffer OLS 150	%avg Buffer OLS 300
sorgente	0	0	0.075	0.075	0.009	0.009
next	0.171	0.171	0.177	0.177	0.419	0.419
next	0.111	0.111	0.323	0.323	0.494	0.494
next	0.090	0.090	0.314	0.314	0.063	0.063
next	0.039	0.039	0.772	0.772	0.015	0.015
next	0.009	0.009	0.653	0.653	0.015	0.015
next	0	0	0.296	0.296	0.012	0.012
next	0	0	0.662	0.662	0	0
next	0	0	0.246	0.246	0	0
next	0	0	0.096	0.096	0	0
next	0	0	0.021	0.021	0	0
next	0	0	0	0	0	0

Tabella 6.21: Dimensione buffer primi nodi del cammino prova 3

In conclusione i valori massimi riscontrati in entrambe le simulazioni fanno riferimento ai nodi più vicini alle foglie sorgenti che, stimando i router nelle due reti nello stesso modo, sono uguali. In tabella 6.21 è riportata la dimensione media dei buffer di un ramo partendo dalla sorgente verso la destinazione in condizioni di traffico costante che conferma quanto illustrato in precedenza. Con gli altri profili, seppur con valori diversi, si ottiene lo stesso andamento.

Per concludere, si riportano i risultati relativi ai ritardi per cui valgono le motivazioni espresse nei paragrafi passati.

Ritardo medio	Routing	Constant	Linear	OLS
Costante	0	0.007	0.061	0.017
Variabile	0	0.115	0.107	6.279
Altamente variabile	0	0.177	3.502	0.151

Tabella 6.22: Ritardo medio prova 3 – reti grandi dimensioni

Ritardo massimo	Routing	Constant	Linear	OLS
Costante	0	2	3	3
Variabile	0	2	1	8
Altamente variabile	0	2	4	3

Tabella 6.23: Ritardo massimo prova 3 – reti grandi dimensioni

Si rimanda all'appendice B per le frequenze relative e l'andamento temporale dei ritardi per ciascuna strategia.

6.4 Sintesi

Le prove effettuate hanno confermato tutte le nostre supposizioni.

In primo luogo è evidente che il nostro modello di analisi distribuita attraverso Intrusion Prevention System migliora le performance con il crescere della dimensione della rete. Infatti avendo più nodi a disposizione per l'analisi si potrà controllare un numero maggiore di pacchetti, aumentando così la percentuale di quelli verificati, ed eventualmente scartati, all'interno della rete stessa. Inoltre anticipare la scoperta di pacchetti malevoli comporta un guadagno maggiore per percorsi lunghi in quanto le risorse per l'analisi nella rete permettono un risparmio su quelle in trasmissione, con evidente vantaggio rispetto ad instradare tutti i pacchetti per verificarli solo a destinazione. Detto ciò, si può comprendere il comportamento visualizzato nel grafico 6.4 riguardante il risparmio energetico percentuale globale nelle condizioni definite nella prima prova. Infatti nelle due simulazioni si otterranno circa gli stessi valori per percentuali di traffico malevolo basse, ossia in caso di un elevato numero di pacchetti che compie l'intero cammino, ma si differenziano sempre più con l'aumentare di attività da parte di attaccanti. La motivazione è racchiusa nel funzionamento del nostro modello: massimizzando la capacità di analisi di ciascun router, si ottiene che un pacchetto controllato nella rete di media dimensione al nodo *k*-esimo sarà verificato dallo stesso anche in quella di grande dimensione[23] ma essendo, nel primo caso, il percorso per la destinazione più breve anche il risparmio energetico sarà inferiore. Riassumendo il nostro modello è globalmente più conveniente dell'attuale meccanismo di routing in quanto anticipare anche solo una parte di analisi, che dovrà comunque essere effettuata, può portare a risparmi che crescono con l'aumentare della percentuale di pacchetti malevoli.

[23] Come illustrato in 6.3.3 i router si comportano nello stesso modo. Quindi se un pacchetto è stato controllato nella rete di medie dimensioni sarà analizzato sicuramente anche nella grande, ma è il vero il contrario.

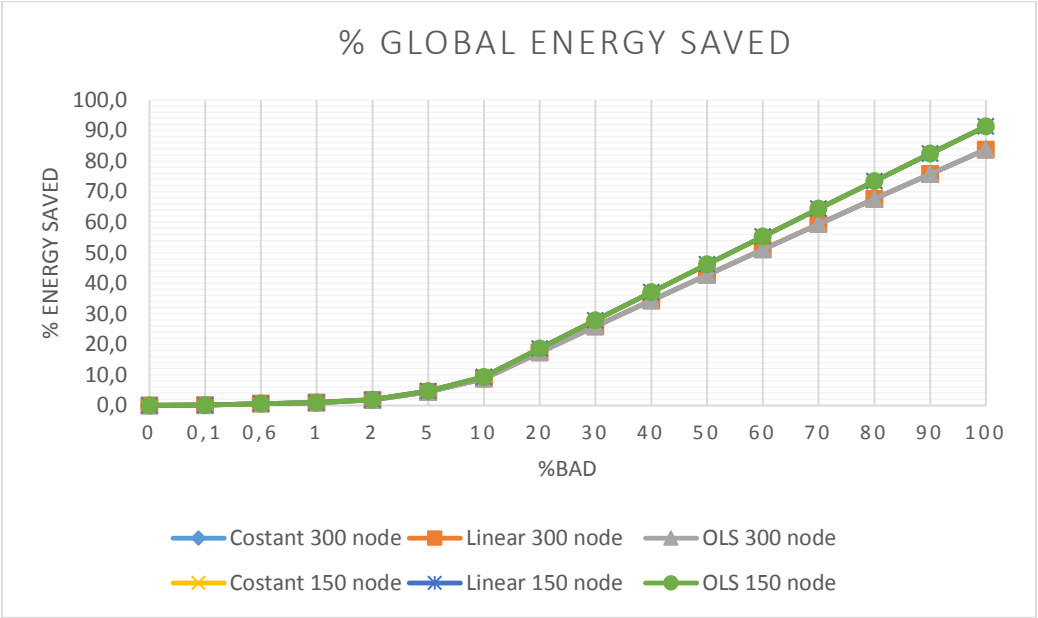

Figura 6.5: Risparmio energetico prova 1

Considerando invece i costi solo all'interno della rete, la nostra soluzione risulta comunque conveniente per percentuali di pacchetti malevoli maggiori del 5% nelle grandi reti e 10% nelle medie.

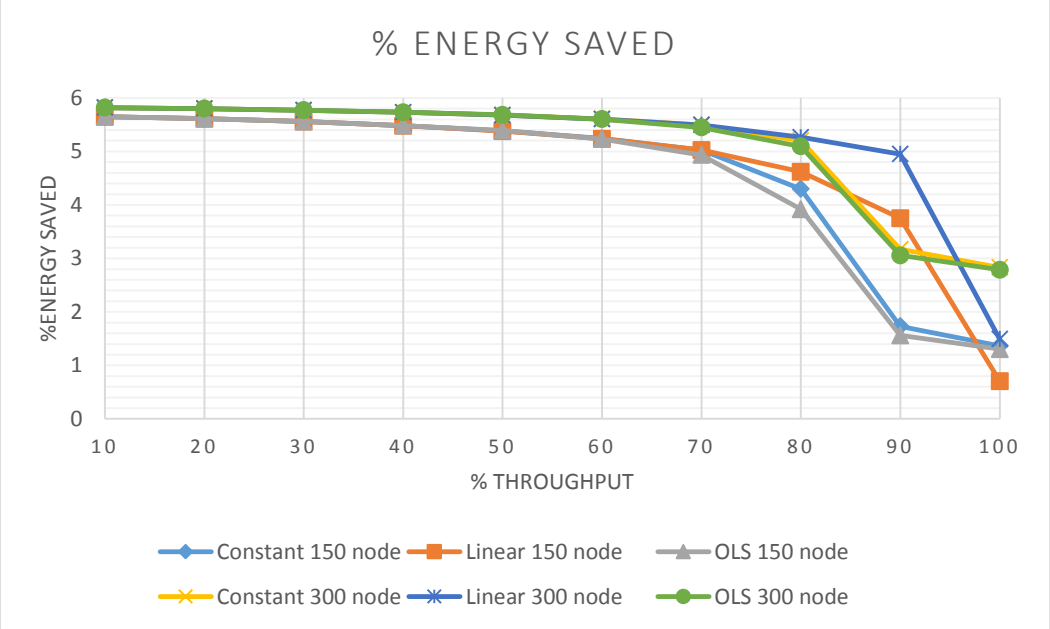

Figura 6.6: Risparmio energetico prova 2

Con la seconda prova, i cui risparmi sul routing sono rappresentati nel grafico 6.5, è possibile sottolineare forse il maggior punto di forza del nostro metodo, ossia la sua adattività alle condizioni di rete che, da una parte, massimizza il numero di pacchetti analizzati per intervallo, dall'altra, rispetta il "dogma" che l'instradamento deve essere sempre effettuato.

Infatti tanto minore è il traffico in ingresso, tanti più pacchetti saranno analizzati dal singolo nodo con conseguente risparmio delle risorse di rete. In altri termini, se l'interfaccia di ingresso non è particolarmente sollecitata, il modulo destinerà diversi pacchetti all'analisi, viceversa altrimenti. Ma non solo, controllando i pacchetti il prima possibile, si ottiene un significativo "effetto a catena": lo scarto dei pacchetti implica una diminuzione del traffico in ingresso al nodo successivo che avrà, di conseguenza, una maggiore disponibilità di analisi. Dal grafico si evince un comportamento "anomalo", seppur in positivo, della strategia lineare per un networkThroughput pari al 90% di quello massimo. Abbiamo quindi investigato le ragioni di tale comportamento osservando che, a differenza delle altre, si ha una maggiore bufferizzazione dei pacchetti che le permettono di analizzare circa lo stesso numero di pacchetti ma su un campione[24] con una maggiore percentuale di malevoli, quindi ottiene maggiori risparmi in trasmissione.

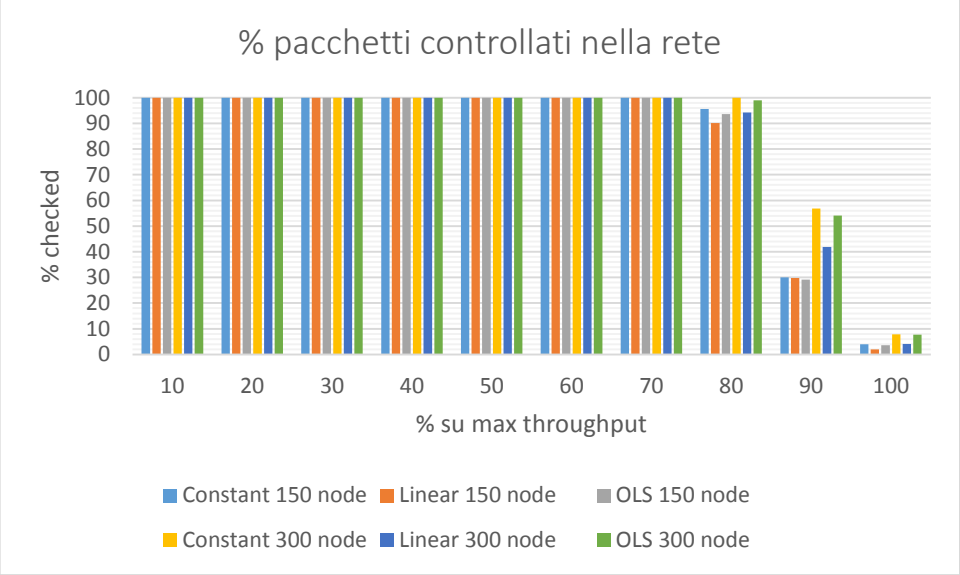

Figura 6.7: Percentuale pacchetti controllati prova 2

Inoltre il nostro modello permette di controllare, ed eventualmente scartare, una piccola parte dei pacchetti anche in caso di traffico intenso con conseguente risparmio energetico e alleggerimento del carico della rete. Nel caso "pessimo", ossia throughput pari al massimo gestibile dalle schede dei router, si ha ugualmente una percentuale di controllati maggiore di zero, più precisamente circa il 3% per 150 nodi e il 6% per 300, e un risparmio energetico rispettivamente del 1% e del 2% nei confronti del routing. Tale comportamento deriva da errori di stima a seguito di cambiamento del traffico osservato, che permettono, seppur introducendo un ritardo, di analizzare alcuni pacchetti anche se non vi sono le condizioni.

[24] Per campione si intende i pacchetti analizzati dall'IPS.

Fortunatamente una parte dei pacchetti controllati è classificata bad e di conseguenza scartata, con risparmio nelle risorse di rete e alleggerimento del carico dei nodi successivi, amplificando il già citato effetto a catena. Purtroppo tali errori sono inevitabili in quanto diversi nodi passano da un traffico in ingresso nullo a quasi massimo, il valore effettivo dipende dalle impredicibili bufferizzazioni. Numeri alla mano, per le reti composte da 150 nodi la dimensione media dei buffer è di circa il 12% della capacità massima che comporta un ritardo medio di 4 quanti; per le grandi i valori sono circa il 10% e 6 rispettivamente. Il maggiore ritardo introdotto è perfettamente giustificato dalla percentuale di pacchetti controllati, infatti nel caso di percorsi più lunghi più nodi compiono errori di stima e quindi possono incorrere in memorizzazione.

Infine si sottolinea che le nostre strategie di ottimizzazione energetica per determinare il numero di pacchetti da destinare all'analisi si differenziano, al variare dei profili di traffico, sulla bontà di stima quindi, indirettamente, sulla bufferizzazione e ritardo in trasmissione rispetto al comportamento "classico" delle reti. Queste considerazioni aiutano a valutare, soprattutto in ottica di sviluppi futuri, in quali circostanze possa essere vantaggioso utilizzare una strategia invece che un'altra. Inoltre al momento abbiamo trascurato la dimensione dei buffer, o meglio li abbiamo assunti di dimensione illimitata, quindi è garantita la ricezione a destinazione di tutti i pacchetti inviati. Viceversa, nella realtà, questo è sicuramente un fattore critico poiché, essendo di dimensione massima prefissata, memorizzare nell'intervallo un numero di pacchetti superiore a tale limite causa la perdita di alcuni di essi. Anche i ritardi in trasmissione potrebbero essere problematici in un caso reale poiché oggi giorno molte applicazioni hanno vincoli sui tempi di ricezione. Seppur il nostro modello limita il tempo massimo di trasmissione a $2 \cdot n$ quanti, con n pari al numero di nodi che compongono il percorso, questo valore potrebbe risultare ancora troppo elevato. Esso potrebbe essere ulteriormente limitato modificando l'attuale comportamento del modulo, si rimanda al prossimo capitolo per maggiori dettagli.

Capitolo 7

Conclusioni e sviluppi futuri

In questa tesi abbiamo presentato delle strategie di ottimizzazione energetica, ossia un modello di analisi distribuito attraverso Intrusion Prevention System. Il nostro lavoro ha come tema principale la *green-awareness* quindi l'obiettivo era ideare un modello per ridurre i consumi energetici delle reti di trasporto. La nostra idea è semplice ma potente: anticipare la scoperta delle attività malevole già all'interno della rete per evitare consumi in trasmissione. Infatti, attualmente, un pacchetto inviato dalla sorgente alla destinazione è controllato da sistemi di protezione, come per esempio gli IPS, solo dopo aver terminato il percorso, in altri termini un pacchetto facente parte di un attacco viene trasmesso nella rete di trasporto per essere, solo alla fine, analizzato e scartato. Assunto che l'operazione di verifica, a prescindere da dove venga effettuata, è inevitabile, abbiamo capito che anticiparla nella rete non avrebbe portato ad altro che vantaggi. Infatti se un nodo, attraverso un IPS, rileva un pacchetto malevolo può scartarlo evitando così il consumo di risorse di tutti i router rimanenti lungo il cammino. L'ideale sarebbe verificare tutti i pacchetti appena trasmessi, questa soluzione non è però realizzabile poiché l'operazione di analisi è onerosa e quindi si creerebbero dei colli di bottiglia con dei ritardi inaccettabili per le odierne qualità di servizio. Per questo motivo, abbiamo optato per una soluzione distribuita con politica "best effort", ossia ciascun nodo controlla il numero massimo di pacchetti che può gestire in quelle condizioni di rete delegando quelli in eccesso al nodo successivo nella speranza che abbia energia a sufficienza per analizzarli. Di conseguenza se il pacchetto *p* malevolo con sorgente A e destinazione B è analizzato dal k-esimo nodo del percorso lungo *n*, definito R e A rispettivamente il costo del routing e analisi del pacchetto, con la nostra soluzione si ottiene un consumo pari a:

$$energyDistributedAnalysis = k \cdot R + A \;\; con\, k \in [1, n] \;\; (1)$$

Viceversa con l'attuale funzionamento delle reti il consumo energetico è pari a:

$$energyRouting = n \cdot R + A \;\; (2)$$

Per realizzare il meccanismo di analisi distribuita abbiamo ideato un modulo, da installare su ciascun router, che determina quali pacchetti inoltrare al processo di analisi e quali all'instradamento, si rimanda al capitolo 4 per tutti i dettagli.

Il primo problema che abbiamo dovuto affrontare è stato evitare che un pacchetto analizzato subisse un nuovo controllo lungo il cammino ottimo verso la destinazione. Per risolverlo

trasmettiamo nel campo opzioni dell'header IP l'informazione sullo stato di analisi che può assumere i valori: "analizzato" e "da analizzare". Infatti se un pacchetto è assegnato all'IPS verrà rinviato sul canale, ovviamente solo se classificato buono, dopo aver concluso interamente l'operazione di verifica. L'alternativa, sviluppata in precedenti lavori [6], era distribuire l'analisi partizionandola, cioè ogni nodo compie una parte del processo di controllo e invia al successivo che ricomincerà da dove si era sospeso il primo. Abbiamo abolito tale meccanismo sia per l'eccessiva complessità, poiché prevedeva di modificare il codice degli IPS, ma soprattutto per inefficienza, con gli attuali tempi di elaborazione di sistemi come Snort e Suricata si aggiungerebbe solo dell'overhead. Utilizzare le opzioni IP permette una totale compatibilità con tutti i dispositivi IP, ossia la quasi totalità dei router nella rete. Di conseguenza se il software del router è aggiornato per la gestione delle opzioni dell'header IP potrà far parte del nostro sistema distribuito, altrimenti provvederà al solo instradamento dei pacchetti. Questa è un'ottima soluzione, poiché permette ai gestori di rete di utilizzare l'IPS per l'analisi distribuita su alcuni nodi prefissati e di mantenere inalterata l'attuale infrastruttura in uso con un evidente risparmio monetario rispetto a cambiare tutti gli apparati di rete.

Superato questo primo aspetto ci siamo concentrati sull'idealizzazione del già citato modulo da installare sul singolo router. Ricapitolando brevemente (per maggiori dettagli consultare il capitolo 4) per ciascun pacchetto in ingresso all'interfaccia di rete, esso determina in base alle nostre strategie di ottimizzazione energetica e alle risorse ancora disponibili, il task di elaborazione, ossia instradamento o analisi. Nello specifico, stabilito il periodo *T*, all'inizio di ciascun intervallo si calcola il numero di pacchetti da analizzare, la quantità maxPPSIPS, prevedendo il traffico in ingresso, *#PktEstimated*, con tre possibili meccanismi di stima: costante, lineare e minimi quadrati.

$$maxPPSIPS = \left\lfloor (1 - load) \cdot \frac{T}{timePrcIPS} \cdot threshold \right\rfloor \quad (3)$$

$$load = \frac{\#PktEstimated}{maxPPS} \leq 1 \; \forall \; [kT, (k+1)T] \quad (4)$$

La formula permette di adattare il numero di pacchetti destinati all'analisi alle condizioni di rete grazie ad una struttura closed loop, ciò è sicuramente il punto di forza del nostro modello che permette di minimizzare i tempi di rilevamento di attività malevole senza ritardare eccessivamente la trasmissione del pacchetto, privilegiando in condizioni di traffico intenso il routing all'analisi. Infatti il secondo operando della formula 3 rappresenta il numero massimo di pacchetti gestibili dall'IPS scelto nel periodo T, quindi a ogni quanto si massimizza il numero di pacchetti da inviare al suddetto processo bilanciando il carico in base al traffico in ingresso.

In altri termini, se si ricevono pochi pacchetti sulla scheda di rete maxPPSIPS è poco inferiore al massimo gestibile, viceversa sarà praticamente nullo. Nonostante ciò è possibile un overbooking delle risorse in caso di traffico stimato inferiore a quello reale. Infatti in questa casistica si assegneranno troppi pacchetti all'analisi che consumerà quindi tutte le risorse a disposizione con inevitabile bufferizzazione, e quindi posticipazione al successivo intervallo, di alcuni pacchetti. Tale difetto è però mitigato da due fattori: la priorità nell'intervallo successivo di eventuali pacchetti memorizzati, che sono instradati appena possibile, e la durata sufficientemente ridotta dell'intervallo T, che permette di aggiornare in tempi brevi le stime effettuando manovre correttive.

Definito e implementato il modello, abbiamo verificato quanto supposto con diverse simulazioni, descritte nel capitolo 6, ciascuna mirata a dimostrare determinati comportamenti. Per prima cosa abbiamo appurato che l'obiettivo principale della tesi, ossia il risparmio energetico è stato pienamente raggiunto con risultati tanto più soddisfacenti tanto maggiore è la percentuale di pacchetti malevoli nella rete. Formalmente data una percentuale *i* di pacchetti malevoli, definito $globalCostRouting_i$ e $globalCostIPS_i$ rispettivamente il consumo dell'attuale comportamento delle reti e della nostra soluzione si ottiene che:

$$globalCostIPS_i \leq globalCostRouting_i \ \forall\, i \in [0{,}100] \quad (5)$$

Inoltre tale risparmio è tanto maggiore tanti più nodi compongono la rete poiché minimizzando i tempi di scarto dei pacchetti malevoli, ciò accadrà circa nello stesso punto del cammino a prescindere dalla lunghezza dei percorsi. In altri termini se l'analisi del pacchetto p malevolo è effettuata dal k-esimo nodo è evidente che se si hanno cammini di lunghezza *n* e *m*, con il primo maggiore del secondo, si ha che:

$$globalCostIPS_{n,p} = globalCostIPS_{m,p} = k \cdot R + A \quad (6)$$

$$globalCostRouting_{n,p} = n \cdot R + A \quad (7)$$

$$globalCostRouting_{m,p} = m \cdot R + A \quad (8)$$

Le prove effettuate hanno anche sottolineato l'adattività alle condizioni di rete che, da una parte, massimizza il numero di pacchetti analizzati per intervallo, dall'altra, rispetta il "dogma" che l'instradamento deve essere sempre effettuato. Infatti variando il throughput in ingresso si è osservato che tanto più questo decresceva tanto maggiore era la percentuale di pacchetti analizzata nella rete e minori i consumi energetici. Ma non solo, controllando i pacchetti il prima possibile, si ottiene un significativo "effetto a catena": lo scarto dei pacchetti implica una diminuzione del traffico in ingresso al nodo successivo che avrà, di conseguenza, una maggiore

disponibilità di analisi. Inoltre il nostro modello permette di controllare, ed eventualmente scartare, una piccola parte dei pacchetti anche in caso di traffico intenso con conseguente risparmio energetico e alleggerimento del carico della rete.

7.1 Sviluppi futuri

Con la nostra tesi abbiamo proposto un modello per l'analisi distribuita tramite Intrusion Prevention System che permette già dei notevoli risparmi energetici. Con il nostro studio abbiamo ideato e sviluppato il cuore del sistema che può essere ulteriormente raffinato per "limare" alcuni aspetti come la dimensione dei buffer e il ritardo in trasmissione.

Nelle nostre simulazioni abbiamo verificato separatamente il comportamento delle tre strategie di stima per l'ottimizzazione energetica ideate confrontandole con l'attuale comportamento delle reti, ossia in ciascun test tutti i router utilizzavano staticamente la stessa tecnica di stima. I risultati hanno evidenziato le differenze tra le soluzioni proposte, in particolare al variare del profilo di traffico le strategie si distinguono per ritardi in trasmissione e percentuali di pacchetti bufferizzati da ciascun nodo. Da queste considerazioni si può trarre spunto per un meccanismo adattativo della strategia di stima, in dettaglio, si potrebbe sviluppare una politica che permetta al router di modificarla in run-time, cosa al momento non prevista. L'idea alla base è di utilizzare la strategia costante, la meno onerosa dal punto di vista computazionale, modificandola solo in caso di necessità, ossia per il tempo in cui il traffico è variabile. In prima istanza, rimandando ad analisi più approfondite, si potrebbe legare tale cambiamento alla dimensione del buffer: se superiore a una certa soglia per un determinato numero di quanti, parametri da individuare con studi mirati, si potrebbe modificare il meccanismo di stima passando a uno più indicato alle condizioni di traffico osservate, per esempio da costante a lineare o da quest'ultimo ai minimi quadrati. Si dovrebbe progettare anche la decisione inversa, ossia il criterio che stabilisca, ovviamente condizioni di rete permettendo, quando ritornare al metodo di stima costante.

Un altro aspetto critico è rappresentato dai ritardi introdotti in trasmissione dall'applicazione del nostro modulo in caso di stime al "ribasso" che producono overbooking delle risorse e conseguente bufferizzazione dei pacchetti in eccesso. Ricordando che il nostro modello già limita il tempo massimo di trasmissione a $2 \cdot n$ quanti, con n pari al numero di nodi che compongono il percorso, questo valore potrebbe essere ancora troppo elevato per alcune applicazioni. Di conseguenza si rimanda a lavori futuri una modifica dell'attuale comportamento del nostro modulo per limitare ulteriormente i ritardi in trasmissione. Un possibile spunto potrebbe essere quello di inserire, sempre nel campo opzioni dell'header IP,

l'informazione sul numero di quanti di ritardo accumulati fino a quel momento dal pacchetto. La scelta, su ciascun router, per determinare il processo a cui inoltrare il pacchetto ricevuto sarà vincolata sia dallo stato che dal sopracitato parametro: il pacchetto è candidato all'analisi se il suo stato è "da analizzare" e il ritardo inferiore a una determinata soglia, da stabilire con studi futuri. Con questa variante supponiamo che sia possibile limitare il ritardo massimo senza diminuire sensibilmente la percentuale di pacchetti controllati all'interno della rete, ovviamente questa affermazione deve essere motivata e verificata con simulazioni.

Infine il nostro modello lavora a livello di pacchetto, quindi potrebbe essere interessante verificare eventuali cambiamenti nelle performance elevandosi a livello di flusso. Ciò potrebbe essere interessante poiché, come spiegato nel capitolo 3, alcuni IPS come Suricata compiono le proprie elaborazioni sui flussi definiti dalla tupla (protocollo, sourceIP, destinationIP, sourcePort, destinationPort).

Appendice A

Reti medie dimensioni

Traffico costante

Router	Constant	Linear	OLS
0	0.000	0.000	0.000
1	0.000	0.000	0.000
2	0.000	0.000	0.000
3	0.000	0.000	0.000
4	0.000	0.000	0.000
5	0.000	0.000	0.000
6	0.000	0.000	0.000
7	0.000	0.000	0.000
8	0.000	0.000	0.000
9	0.000	0.000	0.000
10	0.000	0.000	0.000
11	0.000	0.000	0.000
12	0.000	0.000	0.000
13	0.000	0.000	0.000
14	0.000	0.000	0.000
15	0.000	0.000	0.000
16	0.000	0.000	0.000
17	0.000	0.000	0.000
18	0.000	0.000	0.000
19	0.000	0.000	0.000
20	0.000	0.000	0.000
21	0.000	0.000	0.000
22	0.000	0.000	0.000
23	0.000	0.000	0.000
24	0.000	0.000	0.000
25	0.000	0.000	0.000
26	0.000	0.000	0.000
27	0.000	0.000	0.000
28	0.000	0.000	0.000
29	0.000	0.000	0.000
30	0.000	0.000	0.000
31	0.000	0.000	0.000
32	0.000	0.000	0.000
33	0.000	0.000	0.000
34	0.000	0.000	0.000
35	0.000	0.000	0.000
36	0.000	0.000	0.000
37	0.000	0.000	0.000
38	0.000	0.000	0.000
39	0.000	0.000	0.000
40	0.000	0.000	0.000
41	0.000	0.000	0.000
42	0.000	0.000	0.000
43	0.000	0.000	0.000
44	0.000	0.000	0.000
45	0.000	0.000	0.000
46	0.000	0.000	0.000
47	0.000	0.000	0.000
48	0.000	0.000	0.000

49	0.000	0.000	0.000
50	0.000	0.000	0.000
51	0.000	0.000	0.000
52	0.000	0.000	0.000
53	0.000	0.000	0.000
54	0.000	0.000	0.000
55	0.000	0.000	0.000
56	0.000	0.000	0.000
57	0.000	0.000	0.000
58	0.000	0.000	0.000
59	0.000	0.000	0.000
60	0.000	0.000	0.000
61	0.000	0.000	0.000
62	0.000	0.000	0.000
63	0.000	0.000	0.000
64	0.000	0.000	0.000
65	0.000	0.000	0.000
66	0.000	0.000	0.000
67	0.000	0.000	0.000
68	0.000	0.000	0.000
69	0.000	0.000	0.000
70	0.000	0.000	0.000
71	0.000	0.000	0.000
72	0.000	0.000	0.000
73	0.000	0.000	0.000
74	0.000	0.000	0.000
75	0.000	0.000	0.000
76	0.000	0.000	0.000
77	0.000	0.000	0.000
78	0.000	0.000	0.000
79	0.000	0.000	0.000
80	0.000	0.000	0.000
81	0.000	0.000	0.000
82	0.000	0.000	0.000
83	0.000	0.000	0.000
84	0.000	0.021	0.000
85	0.000	0.021	0.000
86	0.000	0.021	0.000
87	0.000	0.021	0.000
88	0.000	0.021	0.000
89	0.000	0.000	0.000
90	0.000	0.096	0.000
91	0.000	0.096	0.000
92	0.000	0.096	0.000
93	0.000	0.096	0.000
94	0.000	0.096	0.000
95	0.000	0.000	0.000
96	0.000	0.246	0.000
97	0.000	0.246	0.000
98	0.000	0.246	0.000
99	0.000	0.246	0.000
100	0.000	0.246	0.000
101	0.000	0.000	0.000
102	0.000	0.662	0.000
103	0.000	0.662	0.000
104	0.000	0.662	0.000
105	0.000	0.662	0.000
106	0.000	0.662	0.000
107	0.000	0.000	0.000

108	0.000	0.296	0.012
109	0.000	0.296	0.012
110	0.000	0.296	0.012
111	0.000	0.296	0.012
112	0.000	0.296	0.012
113	0.000	0.000	0.000
114	0.009	0.653	0.015
115	0.009	0.653	0.015
116	0.009	0.653	0.015
117	0.009	0.653	0.015
118	0.009	0.653	0.015
119	0.000	0.000	0.000
120	0.039	0.772	0.015
121	0.039	0.772	0.015
122	0.039	0.772	0.015
123	0.039	0.772	0.015
124	0.039	0.772	0.015
125	0.000	0.000	0.000
126	0.090	0.314	0.063
127	0.090	0.314	0.063
128	0.090	0.314	0.063
129	0.090	0.314	0.063
130	0.090	0.314	0.063
131	0.000	0.000	0.000
132	0.111	0.323	0.494
133	0.111	0.323	0.494
134	0.111	0.323	0.494
135	0.111	0.323	0.494
136	0.111	0.323	0.494
137	0.000	0.000	0.000
138	0.171	0.177	0.419
139	0.171	0.177	0.419
140	0.171	0.177	0.419
141	0.171	0.177	0.419
142	0.171	0.177	0.419
143	0.000	0.000	0.000
144	0.000	0.075	0.009
145	0.000	0.075	0.009
146	0.000	0.075	0.009
147	0.000	0.075	0.009
148	0.000	0.075	0.009
149	0.000	0.000	0.000

Tabella A.1: Dimensione buffer traffico costante – reti medie dimensioni

Traffico variabile

Router	Constant	Linear	OLS
0	0.000	0.000	0.000
1	0.000	0.000	0.000
2	0.000	0.000	0.000
3	0.000	0.000	0.000
4	0.000	0.000	0.000
5	0.000	0.000	0.000
6	0.000	0.000	0.000
7	0.000	0.000	0.000
8	0.000	0.000	0.000
9	0.000	0.000	0.000
10	0.000	0.000	0.000
11	0.000	0.000	0.000

12	0.000	0.000	0.000
13	0.000	0.000	0.000
14	0.000	0.000	0.000
15	0.000	0.000	0.000
16	0.000	0.000	0.000
17	0.000	0.000	0.000
18	0.000	0.000	0.000
19	0.000	0.000	0.000
20	0.000	0.000	0.000
21	0.000	0.000	0.000
22	0.000	0.000	0.000
23	0.000	0.000	0.000
24	0.000	0.000	0.000
25	0.000	0.000	0.000
26	0.000	0.000	0.000
27	0.000	0.000	0.000
28	0.000	0.000	0.000
29	0.000	0.000	0.000
30	0.000	0.000	0.000
31	0.000	0.000	0.000
32	0.000	0.000	0.000
33	0.000	0.000	0.000
34	0.000	0.000	0.000
35	0.000	0.000	0.000
36	0.000	0.000	0.000
37	0.000	0.000	0.000
38	0.000	0.000	0.000
39	0.000	0.000	0.000
40	0.000	0.000	0.000
41	0.000	0.000	0.000
42	0.000	0.000	0.000
43	0.000	0.000	0.000
44	0.000	0.000	0.000
45	0.000	0.000	0.000
46	0.000	0.000	0.000
47	0.000	0.000	0.000
48	0.000	0.000	0.000
49	0.000	0.000	0.000
50	0.000	0.000	0.000
51	0.000	0.000	0.000
52	0.000	0.000	0.000
53	0.000	0.000	0.000
54	0.000	0.000	0.000
55	0.000	0.000	0.000
56	0.000	0.000	0.000
57	0.000	0.000	0.000
58	0.000	0.000	0.000
59	0.000	0.000	0.000
60	0.000	0.000	0.000
61	0.000	0.000	0.000
62	0.000	0.000	0.000
63	0.000	0.000	0.000
64	0.000	0.000	0.000
65	0.000	0.000	0.000
66	0.000	0.000	0.000
67	0.000	0.000	0.000
68	0.000	0.000	0.000
69	0.000	0.000	0.000
70	0.000	0.000	0.000

71	0.000	0.000	0.000
72	0.000	0.000	0.000
73	0.000	0.000	0.000
74	0.000	0.000	0.000
75	0.000	0.000	0.000
76	0.000	0.000	0.000
77	0.000	0.000	0.000
78	0.000	0.000	21.544
79	0.000	0.000	21.544
80	0.000	0.000	21.544
81	0.000	0.000	21.544
82	0.000	0.000	21.544
83	0.000	0.000	0.000
84	0.000	0.000	24.271
85	0.000	0.000	24.271
86	0.000	0.000	24.271
87	0.000	0.000	24.271
88	0.000	0.000	24.271
89	0.000	0.000	0.000
90	0.000	0.000	28.546
91	0.000	0.000	28.546
92	0.000	0.000	28.546
93	0.000	0.000	28.546
94	0.000	0.000	28.546
95	0.000	0.000	0.000
96	0.000	0.000	26.855
97	0.000	0.000	26.855
98	0.000	0.000	26.855
99	0.000	0.000	26.855
100	0.000	0.000	26.855
101	0.000	0.000	0.000
102	0.000	0.000	28.726
103	0.000	0.000	28.726
104	0.000	0.000	28.726
105	0.000	0.000	28.726
106	0.000	0.000	28.726
107	0.000	0.000	0.000
108	0.000	0.000	28.428
109	0.000	0.000	28.428
110	0.000	0.000	28.428
111	0.000	0.000	28.428
112	0.000	0.000	28.428
113	0.000	0.000	0.000
114	0.000	0.000	30.504
115	0.000	0.000	30.504
116	0.000	0.000	30.504
117	0.000	0.000	30.504
118	0.000	0.000	30.504
119	0.000	0.000	0.000
120	0.000	0.000	38.098
121	0.000	0.000	38.098
122	0.000	0.000	38.098
123	0.000	0.000	38.098
124	0.000	0.000	38.098
125	0.000	0.000	0.000
126	0.000	0.000	37.760
127	0.000	0.000	37.760
128	0.000	0.000	37.760
129	0.000	0.000	37.760

130	0.000	0.000	37.760
131	0.000	0.000	0.000
132	0.614	1.635	36.917
133	0.614	1.635	36.917
134	0.614	1.635	36.917
135	0.614	1.635	36.917
136	0.614	1.635	36.917
137	0.000	0.000	0.000
138	2.099	3.269	36.949
139	2.099	3.269	36.949
140	2.099	3.269	36.949
141	2.099	3.269	36.949
142	2.099	3.269	36.949
143	0.000	0.000	0.000
144	4.156	1.514	33.278
145	4.156	1.514	33.278
146	4.156	1.514	33.278
147	4.156	1.514	33.278
148	4.156	1.514	33.278
149	0.000	0.000	0.000

Tabella A.2: Dimensione buffer traffico variabile – reti medie dimensioni

Traffico altamente variabile

Router	Constant	Linear	OLS
0	0.000	0.000	0.000
1	0.000	0.000	0.000
2	0.000	0.000	0.000
3	0.000	0.000	0.000
4	0.000	0.000	0.000
5	0.000	0.000	0.000
6	0.000	0.000	0.000
7	0.000	0.000	0.000
8	0.000	0.000	0.000
9	0.000	0.000	0.000
10	0.000	0.000	0.000
11	0.000	0.000	0.000
12	0.000	0.000	0.000
13	0.000	0.000	0.000
14	0.000	0.000	0.000
15	0.000	0.000	0.000
16	0.000	0.000	0.000
17	0.000	0.000	0.000
18	0.000	0.000	0.000
19	0.000	0.000	0.000
20	0.000	0.000	0.000
21	0.000	0.000	0.000
22	0.000	0.000	0.000
23	0.000	0.000	0.000
24	0.000	0.000	0.000
25	0.000	0.000	0.000
26	0.000	0.000	0.000
27	0.000	0.000	0.000
28	0.000	0.000	0.000
29	0.000	0.000	0.000
30	0.000	0.000	0.000
31	0.000	0.000	0.000
32	0.000	0.000	0.000
33	0.000	0.000	0.000

34	0.000	0.000	0.000
35	0.000	0.000	0.000
36	0.000	0.000	0.000
37	0.000	0.000	0.000
38	0.000	0.000	0.000
39	0.000	0.000	0.000
40	0.000	0.000	0.000
41	0.000	0.000	0.000
42	0.000	0.000	0.000
43	0.000	0.000	0.000
44	0.000	0.000	0.000
45	0.000	0.000	0.000
46	0.000	0.000	0.000
47	0.000	0.000	0.000
48	0.000	0.000	0.000
49	0.000	0.000	0.000
50	0.000	0.000	0.000
51	0.000	0.000	0.000
52	0.000	0.000	0.000
53	0.000	0.000	0.000
54	0.000	0.000	0.000
55	0.000	0.000	0.000
56	0.000	0.000	0.000
57	0.000	0.000	0.000
58	0.000	0.000	0.000
59	0.000	0.000	0.000
60	0.000	0.000	0.000
61	0.000	0.000	0.000
62	0.000	0.000	0.000
63	0.000	0.000	0.000
64	0.000	0.000	0.000
65	0.000	0.000	0.000
66	0.000	0.000	0.000
67	0.000	0.000	0.000
68	0.000	0.000	0.000
69	0.000	0.000	0.000
70	0.000	0.000	0.000
71	0.000	0.000	0.000
72	0.000	0.000	0.000
73	0.000	0.000	0.000
74	0.000	0.000	0.000
75	0.000	0.000	0.000
76	0.000	0.000	0.000
77	0.000	0.000	0.000
78	0.000	0.000	0.000
79	0.000	0.000	0.000
80	0.000	0.000	0.000
81	0.000	0.000	0.000
82	0.000	0.000	0.000
83	0.000	0.000	0.000
84	0.000	0.000	0.000
85	0.000	0.000	0.000
86	0.000	0.000	0.000
87	0.000	0.000	0.000
88	0.000	0.000	0.000
89	0.000	0.000	0.000
90	0.000	0.000	0.000
91	0.000	0.000	0.000
92	0.000	0.000	0.000

93	0.000	0.000	0.000
94	0.000	0.000	0.000
95	0.000	0.000	0.000
96	0.000	0.000	0.000
97	0.000	0.000	0.000
98	0.000	0.000	0.000
99	0.000	0.000	0.000
100	0.000	0.000	0.000
101	0.000	0.000	0.000
102	0.000	9.555	0.000
103	0.000	9.555	0.000
104	0.000	9.555	0.000
105	0.000	9.555	0.000
106	0.000	9.555	0.000
107	0.000	0.000	0.000
108	0.000	22.626	0.006
109	0.000	22.626	0.006
110	0.000	22.626	0.006
111	0.000	22.626	0.006
112	0.000	22.626	0.006
113	0.000	0.000	0.000
114	0.000	24.668	0.018
115	0.000	24.668	0.018
116	0.000	24.668	0.018
117	0.000	24.668	0.018
118	0.000	24.668	0.018
119	0.000	0.000	0.000
120	0.012	31.113	0.039
121	0.012	31.113	0.039
122	0.012	31.113	0.039
123	0.012	31.113	0.039
124	0.012	31.113	0.039
125	0.000	0.000	0.000
126	0.027	31.369	0.045
127	0.027	31.369	0.045
128	0.027	31.369	0.045
129	0.027	31.369	0.045
130	0.027	31.369	0.045
131	0.000	0.000	0.000
132	0.054	31.582	0.509
133	0.054	31.582	0.509
134	0.054	31.582	0.509
135	0.054	31.582	0.509
136	0.054	31.582	0.509
137	0.000	0.000	0.000
138	0.072	31.710	0.467
139	0.072	31.710	0.467
140	0.072	31.710	0.467
141	0.072	31.710	0.467
142	0.072	31.710	0.467
143	0.000	0.000	0.000
144	10.468	25.880	8.000
145	10.468	25.880	8.000
146	10.468	25.880	8.000
147	10.468	25.880	8.000
148	10.468	25.880	8.000
149	0.000	0.000	0.000

Tabella A.3: Dimensione buffer traffico altamente variabile – reti medie dimensioni

Reti grandi dimensioni

Traffico costante

Router	Constant	Linear	OLS
0	0.000	0.000	0.000
1	0.000	0.000	0.000
2	0.000	0.000	0.000
3	0.000	0.000	0.000
4	0.000	0.000	0.000
5	0.000	0.000	0.000
6	0.000	0.000	0.000
7	0.000	0.000	0.000
8	0.000	0.000	0.000
9	0.000	0.000	0.000
10	0.000	0.000	0.000
11	0.000	0.000	0.000
12	0.000	0.000	0.000
13	0.000	0.000	0.000
14	0.000	0.000	0.000
15	0.000	0.000	0.000
16	0.000	0.000	0.000
17	0.000	0.000	0.000
18	0.000	0.000	0.000
19	0.000	0.000	0.000
20	0.000	0.000	0.000
21	0.000	0.000	0.000
22	0.000	0.000	0.000
23	0.000	0.000	0.000
24	0.000	0.000	0.000
25	0.000	0.000	0.000
26	0.000	0.000	0.000
27	0.000	0.000	0.000
28	0.000	0.000	0.000
29	0.000	0.000	0.000
30	0.000	0.000	0.000
31	0.000	0.000	0.000
32	0.000	0.000	0.000
33	0.000	0.000	0.000
34	0.000	0.000	0.000
35	0.000	0.000	0.000
36	0.000	0.000	0.000
37	0.000	0.000	0.000
38	0.000	0.000	0.000
39	0.000	0.000	0.000
40	0.000	0.000	0.000
41	0.000	0.000	0.000
42	0.000	0.000	0.000
43	0.000	0.000	0.000
44	0.000	0.000	0.000
45	0.000	0.000	0.000
46	0.000	0.000	0.000
47	0.000	0.000	0.000
48	0.000	0.000	0.000
49	0.000	0.000	0.000
50	0.000	0.000	0.000
51	0.000	0.000	0.000

52	0.000	0.000	0.000
53	0.000	0.000	0.000
54	0.000	0.000	0.000
55	0.000	0.000	0.000
56	0.000	0.000	0.000
57	0.000	0.000	0.000
58	0.000	0.000	0.000
59	0.000	0.000	0.000
60	0.000	0.000	0.000
61	0.000	0.000	0.000
62	0.000	0.000	0.000
63	0.000	0.000	0.000
64	0.000	0.000	0.000
65	0.000	0.000	0.000
66	0.000	0.000	0.000
67	0.000	0.000	0.000
68	0.000	0.000	0.000
69	0.000	0.000	0.000
70	0.000	0.000	0.000
71	0.000	0.000	0.000
72	0.000	0.000	0.000
73	0.000	0.000	0.000
74	0.000	0.000	0.000
75	0.000	0.000	0.000
76	0.000	0.000	0.000
77	0.000	0.000	0.000
78	0.000	0.000	0.000
79	0.000	0.000	0.000
80	0.000	0.000	0.000
81	0.000	0.000	0.000
82	0.000	0.000	0.000
83	0.000	0.000	0.000
84	0.000	0.000	0.000
85	0.000	0.000	0.000
86	0.000	0.000	0.000
87	0.000	0.000	0.000
88	0.000	0.000	0.000
89	0.000	0.000	0.000
90	0.000	0.000	0.000
91	0.000	0.000	0.000
92	0.000	0.000	0.000
93	0.000	0.000	0.000
94	0.000	0.000	0.000
95	0.000	0.000	0.000
96	0.000	0.000	0.000
97	0.000	0.000	0.000
98	0.000	0.000	0.000
99	0.000	0.000	0.000
100	0.000	0.000	0.000
101	0.000	0.000	0.000
102	0.000	0.000	0.000
103	0.000	0.000	0.000
104	0.000	0.000	0.000
105	0.000	0.000	0.000
106	0.000	0.000	0.000
107	0.000	0.000	0.000
108	0.000	0.000	0.000
109	0.000	0.000	0.000
110	0.000	0.000	0.000

111	0.000	0.000	0.000
112	0.000	0.000	0.000
113	0.000	0.000	0.000
114	0.000	0.000	0.000
115	0.000	0.000	0.000
116	0.000	0.000	0.000
117	0.000	0.000	0.000
118	0.000	0.000	0.000
119	0.000	0.000	0.000
120	0.000	0.000	0.000
121	0.000	0.000	0.000
122	0.000	0.000	0.000
123	0.000	0.000	0.000
124	0.000	0.000	0.000
125	0.000	0.000	0.000
126	0.000	0.000	0.000
127	0.000	0.000	0.000
128	0.000	0.000	0.000
129	0.000	0.000	0.000
130	0.000	0.000	0.000
131	0.000	0.000	0.000
132	0.000	0.000	0.000
133	0.000	0.000	0.000
134	0.000	0.000	0.000
135	0.000	0.000	0.000
136	0.000	0.000	0.000
137	0.000	0.000	0.000
138	0.000	0.000	0.000
139	0.000	0.000	0.000
140	0.000	0.000	0.000
141	0.000	0.000	0.000
142	0.000	0.000	0.000
143	0.000	0.000	0.000
144	0.000	0.000	0.000
145	0.000	0.000	0.000
146	0.000	0.000	0.000
147	0.000	0.000	0.000
148	0.000	0.000	0.000
149	0.000	0.000	0.000
150	0.000	0.000	0.000
151	0.000	0.000	0.000
152	0.000	0.000	0.000
153	0.000	0.000	0.000
154	0.000	0.000	0.000
155	0.000	0.000	0.000
156	0.000	0.000	0.000
157	0.000	0.000	0.000
158	0.000	0.000	0.000
159	0.000	0.000	0.000
160	0.000	0.000	0.000
161	0.000	0.000	0.000
162	0.000	0.000	0.000
163	0.000	0.000	0.000
164	0.000	0.000	0.000
165	0.000	0.000	0.000
166	0.000	0.000	0.000
167	0.000	0.000	0.000
168	0.000	0.000	0.000
169	0.000	0.000	0.000

170	0.000	0.000	0.000
171	0.000	0.000	0.000
172	0.000	0.000	0.000
173	0.000	0.000	0.000
174	0.000	0.000	0.000
175	0.000	0.000	0.000
176	0.000	0.000	0.000
177	0.000	0.000	0.000
178	0.000	0.000	0.000
179	0.000	0.000	0.000
180	0.000	0.000	0.000
181	0.000	0.000	0.000
182	0.000	0.000	0.000
183	0.000	0.000	0.000
184	0.000	0.000	0.000
185	0.000	0.000	0.000
186	0.000	0.000	0.000
187	0.000	0.000	0.000
188	0.000	0.000	0.000
189	0.000	0.000	0.000
190	0.000	0.000	0.000
191	0.000	0.000	0.000
192	0.000	0.000	0.000
193	0.000	0.000	0.000
194	0.000	0.000	0.000
195	0.000	0.000	0.000
196	0.000	0.000	0.000
197	0.000	0.000	0.000
198	0.000	0.000	0.000
199	0.000	0.000	0.000
200	0.000	0.000	0.000
201	0.000	0.000	0.000
202	0.000	0.000	0.000
203	0.000	0.000	0.000
204	0.000	0.000	0.000
205	0.000	0.000	0.000
206	0.000	0.000	0.000
207	0.000	0.000	0.000
208	0.000	0.000	0.000
209	0.000	0.000	0.000
210	0.000	0.000	0.000
211	0.000	0.000	0.000
212	0.000	0.000	0.000
213	0.000	0.000	0.000
214	0.000	0.000	0.000
215	0.000	0.000	0.000
216	0.000	0.000	0.000
217	0.000	0.000	0.000
218	0.000	0.000	0.000
219	0.000	0.000	0.000
220	0.000	0.000	0.000
221	0.000	0.000	0.000
222	0.000	0.000	0.000
223	0.000	0.000	0.000
224	0.000	0.000	0.000
225	0.000	0.000	0.000
226	0.000	0.000	0.000
227	0.000	0.000	0.000
228	0.000	0.000	0.000

229	0.000	0.000	0.000
230	0.000	0.000	0.000
231	0.000	0.000	0.000
232	0.000	0.000	0.000
233	0.000	0.000	0.000
234	0.000	0.021	0.000
235	0.000	0.021	0.000
236	0.000	0.021	0.000
237	0.000	0.021	0.000
238	0.000	0.021	0.000
239	0.000	0.000	0.000
240	0.000	0.096	0.000
241	0.000	0.096	0.000
242	0.000	0.096	0.000
243	0.000	0.096	0.000
244	0.000	0.096	0.000
245	0.000	0.000	0.000
246	0.000	0.246	0.000
247	0.000	0.246	0.000
248	0.000	0.246	0.000
249	0.000	0.246	0.000
250	0.000	0.246	0.000
251	0.000	0.000	0.000
252	0.000	0.662	0.000
253	0.000	0.662	0.000
254	0.000	0.662	0.000
255	0.000	0.662	0.000
256	0.000	0.662	0.000
257	0.000	0.000	0.000
258	0.000	0.296	0.012
259	0.000	0.296	0.012
260	0.000	0.296	0.012
261	0.000	0.296	0.012
262	0.000	0.296	0.012
263	0.000	0.000	0.000
264	0.009	0.653	0.015
265	0.009	0.653	0.015
266	0.009	0.653	0.015
267	0.009	0.653	0.015
268	0.009	0.653	0.015
269	0.000	0.000	0.000
270	0.039	0.772	0.015
271	0.039	0.772	0.015
272	0.039	0.772	0.015
273	0.039	0.772	0.015
274	0.039	0.772	0.015
275	0.000	0.000	0.000
276	0.090	0.314	0.063
277	0.090	0.314	0.063
278	0.090	0.314	0.063
279	0.090	0.314	0.063
280	0.090	0.314	0.063
281	0.000	0.000	0.000
282	0.111	0.323	0.494
283	0.111	0.323	0.494
284	0.111	0.323	0.494
285	0.111	0.323	0.494
286	0.111	0.323	0.494
287	0.000	0.000	0.000

288	0.171	0.177	0.419
289	0.171	0.177	0.419
290	0.171	0.177	0.419
291	0.171	0.177	0.419
292	0.171	0.177	0.419
293	0.000	0.000	0.000
294	0.000	0.075	0.009
295	0.000	0.075	0.009
296	0.000	0.075	0.009
297	0.000	0.075	0.009
298	0.000	0.075	0.009
299	0.000	0.000	0.000

Tabella A.4: Dimensione buffer traffico costante – reti grandi dimensioni

Traffico variabile

Router	Constant	Linear	OLS
0	0.000	0.000	0.000
1	0.000	0.000	0.000
2	0.000	0.000	0.000
3	0.000	0.000	0.000
4	0.000	0.000	0.000
5	0.000	0.000	0.000
6	0.000	0.000	0.000
7	0.000	0.000	0.000
8	0.000	0.000	0.000
9	0.000	0.000	0.000
10	0.000	0.000	0.000
11	0.000	0.000	0.000
12	0.000	0.000	0.000
13	0.000	0.000	0.000
14	0.000	0.000	0.000
15	0.000	0.000	0.000
16	0.000	0.000	0.000
17	0.000	0.000	0.000
18	0.000	0.000	0.000
19	0.000	0.000	0.000
20	0.000	0.000	0.000
21	0.000	0.000	0.000
22	0.000	0.000	0.000
23	0.000	0.000	0.000
24	0.000	0.000	0.000
25	0.000	0.000	0.000
26	0.000	0.000	0.000
27	0.000	0.000	0.000
28	0.000	0.000	0.000
29	0.000	0.000	0.000
30	0.000	0.000	0.000
31	0.000	0.000	0.000
32	0.000	0.000	0.000
33	0.000	0.000	0.000
34	0.000	0.000	0.000
35	0.000	0.000	0.000
36	0.000	0.000	0.000
37	0.000	0.000	0.000
38	0.000	0.000	0.000
39	0.000	0.000	0.000
40	0.000	0.000	0.000
41	0.000	0.000	0.000

42	0.000	0.000	0.000
43	0.000	0.000	0.000
44	0.000	0.000	0.000
45	0.000	0.000	0.000
46	0.000	0.000	0.000
47	0.000	0.000	0.000
48	0.000	0.000	0.000
49	0.000	0.000	0.000
50	0.000	0.000	0.000
51	0.000	0.000	0.000
52	0.000	0.000	0.000
53	0.000	0.000	0.000
54	0.000	0.000	0.000
55	0.000	0.000	0.000
56	0.000	0.000	0.000
57	0.000	0.000	0.000
58	0.000	0.000	0.000
59	0.000	0.000	0.000
60	0.000	0.000	0.000
61	0.000	0.000	0.000
62	0.000	0.000	0.000
63	0.000	0.000	0.000
64	0.000	0.000	0.000
65	0.000	0.000	0.000
66	0.000	0.000	0.000
67	0.000	0.000	0.000
68	0.000	0.000	0.000
69	0.000	0.000	0.000
70	0.000	0.000	0.000
71	0.000	0.000	0.000
72	0.000	0.000	0.000
73	0.000	0.000	0.000
74	0.000	0.000	0.000
75	0.000	0.000	0.000
76	0.000	0.000	0.000
77	0.000	0.000	0.000
78	0.000	0.000	0.000
79	0.000	0.000	0.000
80	0.000	0.000	0.000
81	0.000	0.000	0.000
82	0.000	0.000	0.000
83	0.000	0.000	0.000
84	0.000	0.000	0.000
85	0.000	0.000	0.000
86	0.000	0.000	0.000
87	0.000	0.000	0.000
88	0.000	0.000	0.000
89	0.000	0.000	0.000
90	0.000	0.000	0.000
91	0.000	0.000	0.000
92	0.000	0.000	0.000
93	0.000	0.000	0.000
94	0.000	0.000	0.000
95	0.000	0.000	0.000
96	0.000	0.000	0.000
97	0.000	0.000	0.000
98	0.000	0.000	0.000
99	0.000	0.000	0.000
100	0.000	0.000	0.000

101	0.000	0.000	0.000
102	0.000	0.000	0.000
103	0.000	0.000	0.000
104	0.000	0.000	0.000
105	0.000	0.000	0.000
106	0.000	0.000	0.000
107	0.000	0.000	0.000
108	0.000	0.000	0.000
109	0.000	0.000	0.000
110	0.000	0.000	0.000
111	0.000	0.000	0.000
112	0.000	0.000	0.000
113	0.000	0.000	0.000
114	0.000	0.000	0.000
115	0.000	0.000	0.000
116	0.000	0.000	0.000
117	0.000	0.000	0.000
118	0.000	0.000	0.000
119	0.000	0.000	0.000
120	0.000	0.000	0.000
121	0.000	0.000	0.000
122	0.000	0.000	0.000
123	0.000	0.000	0.000
124	0.000	0.000	0.000
125	0.000	0.000	0.000
126	0.000	0.000	0.000
127	0.000	0.000	0.000
128	0.000	0.000	0.000
129	0.000	0.000	0.000
130	0.000	0.000	0.000
131	0.000	0.000	0.000
132	0.000	0.000	0.000
133	0.000	0.000	0.000
134	0.000	0.000	0.000
135	0.000	0.000	0.000
136	0.000	0.000	0.000
137	0.000	0.000	0.000
138	0.000	0.000	0.000
139	0.000	0.000	0.000
140	0.000	0.000	0.000
141	0.000	0.000	0.000
142	0.000	0.000	0.000
143	0.000	0.000	0.000
144	0.000	0.000	0.000
145	0.000	0.000	0.000
146	0.000	0.000	0.000
147	0.000	0.000	0.000
148	0.000	0.000	0.000
149	0.000	0.000	0.000
150	0.000	0.000	0.000
151	0.000	0.000	0.000
152	0.000	0.000	0.000
153	0.000	0.000	0.000
154	0.000	0.000	0.000
155	0.000	0.000	0.000
156	0.000	0.000	0.000
157	0.000	0.000	0.000
158	0.000	0.000	0.000
159	0.000	0.000	0.000

160	0.000	0.000	0.000
161	0.000	0.000	0.000
162	0.000	0.000	0.000
163	0.000	0.000	0.000
164	0.000	0.000	0.000
165	0.000	0.000	0.000
166	0.000	0.000	0.000
167	0.000	0.000	0.000
168	0.000	0.000	0.000
169	0.000	0.000	0.000
170	0.000	0.000	0.000
171	0.000	0.000	0.000
172	0.000	0.000	0.000
173	0.000	0.000	0.000
174	0.000	0.000	0.000
175	0.000	0.000	0.000
176	0.000	0.000	0.000
177	0.000	0.000	0.000
178	0.000	0.000	0.000
179	0.000	0.000	0.000
180	0.000	0.000	0.000
181	0.000	0.000	0.000
182	0.000	0.000	0.000
183	0.000	0.000	0.000
184	0.000	0.000	0.000
185	0.000	0.000	0.000
186	0.000	0.000	0.000
187	0.000	0.000	0.000
188	0.000	0.000	0.000
189	0.000	0.000	0.000
190	0.000	0.000	0.000
191	0.000	0.000	0.000
192	0.000	0.000	0.000
193	0.000	0.000	0.000
194	0.000	0.000	0.000
195	0.000	0.000	0.000
196	0.000	0.000	0.000
197	0.000	0.000	0.000
198	0.000	0.000	0.000
199	0.000	0.000	0.000
200	0.000	0.000	0.000
201	0.000	0.000	0.000
202	0.000	0.000	0.000
203	0.000	0.000	0.000
204	0.000	0.000	0.000
205	0.000	0.000	0.000
206	0.000	0.000	0.000
207	0.000	0.000	0.000
208	0.000	0.000	0.000
209	0.000	0.000	0.000
210	0.000	0.000	0.000
211	0.000	0.000	0.000
212	0.000	0.000	0.000
213	0.000	0.000	0.000
214	0.000	0.000	0.000
215	0.000	0.000	0.000
216	0.000	0.000	0.000
217	0.000	0.000	0.000
218	0.000	0.000	0.000

219	0.000	0.000	0.000
220	0.000	0.000	0.000
221	0.000	0.000	0.000
222	0.000	0.000	0.000
223	0.000	0.000	0.000
224	0.000	0.000	0.000
225	0.000	0.000	0.000
226	0.000	0.000	0.000
227	0.000	0.000	0.000
228	0.000	0.000	21.544
229	0.000	0.000	21.544
230	0.000	0.000	21.544
231	0.000	0.000	21.544
232	0.000	0.000	21.544
233	0.000	0.000	0.000
234	0.000	0.000	24.271
235	0.000	0.000	24.271
236	0.000	0.000	24.271
237	0.000	0.000	24.271
238	0.000	0.000	24.271
239	0.000	0.000	0.000
240	0.000	0.000	28.546
241	0.000	0.000	28.546
242	0.000	0.000	28.546
243	0.000	0.000	28.546
244	0.000	0.000	28.546
245	0.000	0.000	0.000
246	0.000	0.000	26.855
247	0.000	0.000	26.855
248	0.000	0.000	26.855
249	0.000	0.000	26.855
250	0.000	0.000	26.855
251	0.000	0.000	0.000
252	0.000	0.000	28.726
253	0.000	0.000	28.726
254	0.000	0.000	28.726
255	0.000	0.000	28.726
256	0.000	0.000	28.726
257	0.000	0.000	0.000
258	0.000	0.000	28.428
259	0.000	0.000	28.428
260	0.000	0.000	28.428
261	0.000	0.000	28.428
262	0.000	0.000	28.428
263	0.000	0.000	0.000
264	0.000	0.000	30.504
265	0.000	0.000	30.504
266	0.000	0.000	30.504
267	0.000	0.000	30.504
268	0.000	0.000	30.504
269	0.000	0.000	0.000
270	0.000	0.000	38.098
271	0.000	0.000	38.098
272	0.000	0.000	38.098
273	0.000	0.000	38.098
274	0.000	0.000	38.098
275	0.000	0.000	0.000
276	0.000	0.000	37.760
277	0.000	0.000	37.760

278	0.000	0.000	37.760
279	0.000	0.000	37.760
280	0.000	0.000	37.760
281	0.000	0.000	0.000
282	0.614	1.635	36.917
283	0.614	1.635	36.917
284	0.614	1.635	36.917
285	0.614	1.635	36.917
286	0.614	1.635	36.917
287	0.000	0.000	0.000
288	2.099	3.269	36.949
289	2.099	3.269	36.949
290	2.099	3.269	36.949
291	2.099	3.269	36.949
292	2.099	3.269	36.949
293	0.000	0.000	0.000
294	4.156	1.514	33.278
295	4.156	1.514	33.278
296	4.156	1.514	33.278
297	4.156	1.514	33.278
298	4.156	1.514	33.278
299	0.000	0.000	0.000

Tabella A.5: Dimensione buffer traffico variabile– reti grandi dimensioni

Traffico altamente variabile

Router	Constant	Linear	OLS
0	0.000	0.000	0.000
1	0.000	0.000	0.000
2	0.000	0.000	0.000
3	0.000	0.000	0.000
4	0.000	0.000	0.000
5	0.000	0.000	0.000
6	0.000	0.000	0.000
7	0.000	0.000	0.000
8	0.000	0.000	0.000
9	0.000	0.000	0.000
10	0.000	0.000	0.000
11	0.000	0.000	0.000
12	0.000	0.000	0.000
13	0.000	0.000	0.000
14	0.000	0.000	0.000
15	0.000	0.000	0.000
16	0.000	0.000	0.000
17	0.000	0.000	0.000
18	0.000	0.000	0.000
19	0.000	0.000	0.000
20	0.000	0.000	0.000
21	0.000	0.000	0.000
22	0.000	0.000	0.000
23	0.000	0.000	0.000
24	0.000	0.000	0.000
25	0.000	0.000	0.000
26	0.000	0.000	0.000
27	0.000	0.000	0.000
28	0.000	0.000	0.000
29	0.000	0.000	0.000
30	0.000	0.000	0.000
31	0.000	0.000	0.000

32	0.000	0.000	0.000
33	0.000	0.000	0.000
34	0.000	0.000	0.000
35	0.000	0.000	0.000
36	0.000	0.000	0.000
37	0.000	0.000	0.000
38	0.000	0.000	0.000
39	0.000	0.000	0.000
40	0.000	0.000	0.000
41	0.000	0.000	0.000
42	0.000	0.000	0.000
43	0.000	0.000	0.000
44	0.000	0.000	0.000
45	0.000	0.000	0.000
46	0.000	0.000	0.000
47	0.000	0.000	0.000
48	0.000	0.000	0.000
49	0.000	0.000	0.000
50	0.000	0.000	0.000
51	0.000	0.000	0.000
52	0.000	0.000	0.000
53	0.000	0.000	0.000
54	0.000	0.000	0.000
55	0.000	0.000	0.000
56	0.000	0.000	0.000
57	0.000	0.000	0.000
58	0.000	0.000	0.000
59	0.000	0.000	0.000
60	0.000	0.000	0.000
61	0.000	0.000	0.000
62	0.000	0.000	0.000
63	0.000	0.000	0.000
64	0.000	0.000	0.000
65	0.000	0.000	0.000
66	0.000	0.000	0.000
67	0.000	0.000	0.000
68	0.000	0.000	0.000
69	0.000	0.000	0.000
70	0.000	0.000	0.000
71	0.000	0.000	0.000
72	0.000	0.000	0.000
73	0.000	0.000	0.000
74	0.000	0.000	0.000
75	0.000	0.000	0.000
76	0.000	0.000	0.000
77	0.000	0.000	0.000
78	0.000	0.000	0.000
79	0.000	0.000	0.000
80	0.000	0.000	0.000
81	0.000	0.000	0.000
82	0.000	0.000	0.000
83	0.000	0.000	0.000
84	0.000	0.000	0.000
85	0.000	0.000	0.000
86	0.000	0.000	0.000
87	0.000	0.000	0.000
88	0.000	0.000	0.000
89	0.000	0.000	0.000
90	0.000	0.000	0.000

91	0.000	0.000	0.000
92	0.000	0.000	0.000
93	0.000	0.000	0.000
94	0.000	0.000	0.000
95	0.000	0.000	0.000
96	0.000	0.000	0.000
97	0.000	0.000	0.000
98	0.000	0.000	0.000
99	0.000	0.000	0.000
100	0.000	0.000	0.000
101	0.000	0.000	0.000
102	0.000	0.000	0.000
103	0.000	0.000	0.000
104	0.000	0.000	0.000
105	0.000	0.000	0.000
106	0.000	0.000	0.000
107	0.000	0.000	0.000
108	0.000	0.000	0.000
109	0.000	0.000	0.000
110	0.000	0.000	0.000
111	0.000	0.000	0.000
112	0.000	0.000	0.000
113	0.000	0.000	0.000
114	0.000	0.000	0.000
115	0.000	0.000	0.000
116	0.000	0.000	0.000
117	0.000	0.000	0.000
118	0.000	0.000	0.000
119	0.000	0.000	0.000
120	0.000	0.000	0.000
121	0.000	0.000	0.000
122	0.000	0.000	0.000
123	0.000	0.000	0.000
124	0.000	0.000	0.000
125	0.000	0.000	0.000
126	0.000	0.000	0.000
127	0.000	0.000	0.000
128	0.000	0.000	0.000
129	0.000	0.000	0.000
130	0.000	0.000	0.000
131	0.000	0.000	0.000
132	0.000	0.000	0.000
133	0.000	0.000	0.000
134	0.000	0.000	0.000
135	0.000	0.000	0.000
136	0.000	0.000	0.000
137	0.000	0.000	0.000
138	0.000	0.000	0.000
139	0.000	0.000	0.000
140	0.000	0.000	0.000
141	0.000	0.000	0.000
142	0.000	0.000	0.000
143	0.000	0.000	0.000
144	0.000	0.000	0.000
145	0.000	0.000	0.000
146	0.000	0.000	0.000
147	0.000	0.000	0.000
148	0.000	0.000	0.000
149	0.000	0.000	0.000

150	0.000	0.000	0.000
151	0.000	0.000	0.000
152	0.000	0.000	0.000
153	0.000	0.000	0.000
154	0.000	0.000	0.000
155	0.000	0.000	0.000
156	0.000	0.000	0.000
157	0.000	0.000	0.000
158	0.000	0.000	0.000
159	0.000	0.000	0.000
160	0.000	0.000	0.000
161	0.000	0.000	0.000
162	0.000	0.000	0.000
163	0.000	0.000	0.000
164	0.000	0.000	0.000
165	0.000	0.000	0.000
166	0.000	0.000	0.000
167	0.000	0.000	0.000
168	0.000	0.000	0.000
169	0.000	0.000	0.000
170	0.000	0.000	0.000
171	0.000	0.000	0.000
172	0.000	0.000	0.000
173	0.000	0.000	0.000
174	0.000	0.000	0.000
175	0.000	0.000	0.000
176	0.000	0.000	0.000
177	0.000	0.000	0.000
178	0.000	0.000	0.000
179	0.000	0.000	0.000
180	0.000	0.000	0.000
181	0.000	0.000	0.000
182	0.000	0.000	0.000
183	0.000	0.000	0.000
184	0.000	0.000	0.000
185	0.000	0.000	0.000
186	0.000	0.000	0.000
187	0.000	0.000	0.000
188	0.000	0.000	0.000
189	0.000	0.000	0.000
190	0.000	0.000	0.000
191	0.000	0.000	0.000
192	0.000	0.000	0.000
193	0.000	0.000	0.000
194	0.000	0.000	0.000
195	0.000	0.000	0.000
196	0.000	0.000	0.000
197	0.000	0.000	0.000
198	0.000	0.000	0.000
199	0.000	0.000	0.000
200	0.000	0.000	0.000
201	0.000	0.000	0.000
202	0.000	0.000	0.000
203	0.000	0.000	0.000
204	0.000	0.000	0.000
205	0.000	0.000	0.000
206	0.000	0.000	0.000
207	0.000	0.000	0.000
208	0.000	0.000	0.000

209	0.000	0.000	0.000
210	0.000	0.000	0.000
211	0.000	0.000	0.000
212	0.000	0.000	0.000
213	0.000	0.000	0.000
214	0.000	0.000	0.000
215	0.000	0.000	0.000
216	0.000	0.000	0.000
217	0.000	0.000	0.000
218	0.000	0.000	0.000
219	0.000	0.000	0.000
220	0.000	0.000	0.000
221	0.000	0.000	0.000
222	0.000	0.000	0.000
223	0.000	0.000	0.000
224	0.000	0.000	0.000
225	0.000	0.000	0.000
226	0.000	0.000	0.000
227	0.000	0.000	0.000
228	0.000	0.000	0.000
229	0.000	0.000	0.000
230	0.000	0.000	0.000
231	0.000	0.000	0.000
232	0.000	0.000	0.000
233	0.000	0.000	0.000
234	0.000	0.000	0.000
235	0.000	0.000	0.000
236	0.000	0.000	0.000
237	0.000	0.000	0.000
238	0.000	0.000	0.000
239	0.000	0.000	0.000
240	0.000	0.000	0.000
241	0.000	0.000	0.000
242	0.000	0.000	0.000
243	0.000	0.000	0.000
244	0.000	0.000	0.000
245	0.000	0.000	0.000
246	0.000	0.000	0.000
247	0.000	0.000	0.000
248	0.000	0.000	0.000
249	0.000	0.000	0.000
250	0.000	0.000	0.000
251	0.000	0.000	0.000
252	0.000	9.555	0.000
253	0.000	9.555	0.000
254	0.000	9.555	0.000
255	0.000	9.555	0.000
256	0.000	9.555	0.000
257	0.000	0.000	0.000
258	0.000	22.626	0.006
259	0.000	22.626	0.006
260	0.000	22.626	0.006
261	0.000	22.626	0.006
262	0.000	22.626	0.006
263	0.000	0.000	0.000
264	0.000	24.668	0.018
265	0.000	24.668	0.018
266	0.000	24.668	0.018
267	0.000	24.668	0.018

268	0.000	24.668	0.018
269	0.000	0.000	0.000
270	0.012	31.113	0.039
271	0.012	31.113	0.039
272	0.012	31.113	0.039
273	0.012	31.113	0.039
274	0.012	31.113	0.039
275	0.000	0.000	0.000
276	0.027	31.369	0.045
277	0.027	31.369	0.045
278	0.027	31.369	0.045
279	0.027	31.369	0.045
280	0.027	31.369	0.045
281	0.000	0.000	0.000
282	0.054	31.582	0.509
283	0.054	31.582	0.509
284	0.054	31.582	0.509
285	0.054	31.582	0.509
286	0.054	31.582	0.509
287	0.000	0.000	0.000
288	0.072	31.710	0.467
289	0.072	31.710	0.467
290	0.072	31.710	0.467
291	0.072	31.710	0.467
292	0.072	31.710	0.467
293	0.000	0.000	0.000
294	10.468	25.880	8.000
295	10.468	25.880	8.000
296	10.468	25.880	8.000
297	10.468	25.880	8.000
298	10.468	25.880	8.000
299	0.000	0.000	0.000

Tabella A.6: Dimensione buffer traffico altamente variabile – reti grandi dimensioni

Appendice B

Definito *#Occorrenze*$_i$ il numero di pacchetti ricevuti con ritardo pari a *i*, si calcola la frequenza relativa come:

$$frequenzaRelativa_i = \frac{\#Occorrenze_i}{100000} \quad (1)$$

Definiti *#pacchetti*$_{ij}$ come il numero di pacchetti partiti nell'istante *i* e arrivati a destinazione con ritardo *j*, m_i il numero totale di pacchetti inviati in i e *n* il ritardo massimo introdotto, si calcola la media dei ritardi per ogni istante i come:

$$mediaRitardi_i = \frac{1}{m_i}\sum_{j=0}^{n} \#pacchetti_{ij} \cdot j \quad (2)$$

Si è osservato che i valori non sono influenzati dalla dimensione della rete.

Frequenze relative dei ritardi

Traffico costante

Ritardo	Constant	Linear	OLS
0	0.99335	0.95430	0.98575
1	0.00630	0.03075	0.01155
2	0.00035	0.01490	0.00250
3	0.00000	0.00005	0.00020

Tabella B.1: Frequenze relative ritardi traffico costante

Traffico variabile

Ritardo	Constant	Linear	OLS
0	0.88960	0.89290	0.00755
1	0.10630	0.10710	0.01935
2	0.00410	0.00000	0.01795
3	0.00000	0.00000	0.02195
4	0.00000	0.00000	0.01730
5	0.00000	0.00000	0.02485
6	0.00000	0.00000	0.33125
7	0.00000	0.00000	0.50180
8	0.00000	0.00000	0.05800

Tabella B.2: Frequenze relative ritardi traffico variabile

Traffico altamente variabile

Ritardo	Constant	Linear	OLS
0	0.82355	0.00320	0.85370
1	0.17585	0.00725	0.14150
2	0.00060	0.00910	0.00465
3	0.00000	0.44575	0.00015
4	0.00000	0.53470	0.00000

Tabella B.3: Frequenze relative ritardi traffico altamente variabile

Appendice B

Andamento temporale della media dei ritardi

Traffico costante

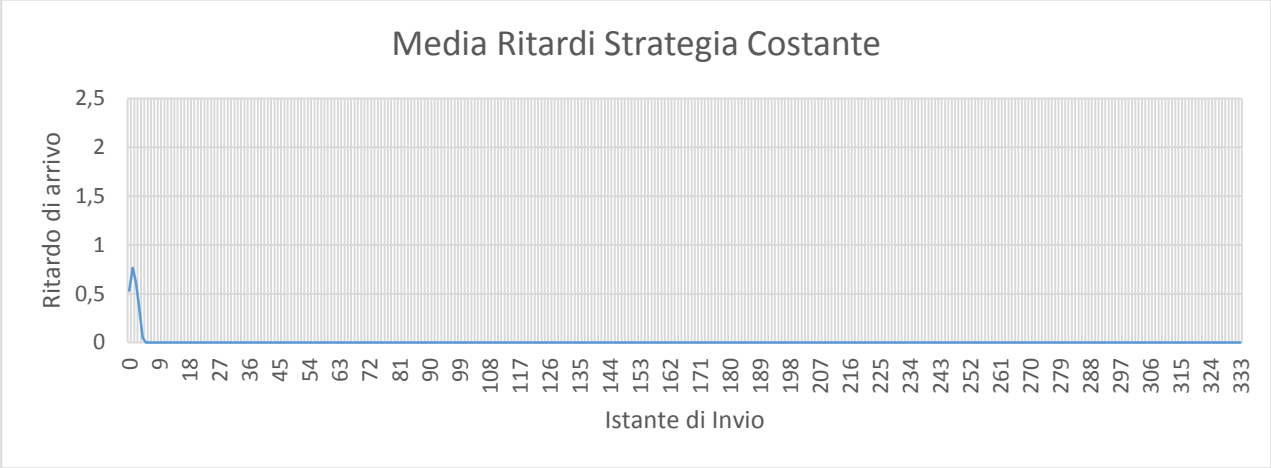

Figura B.1: Andamento temporale ritardi traffico costante strategia costante

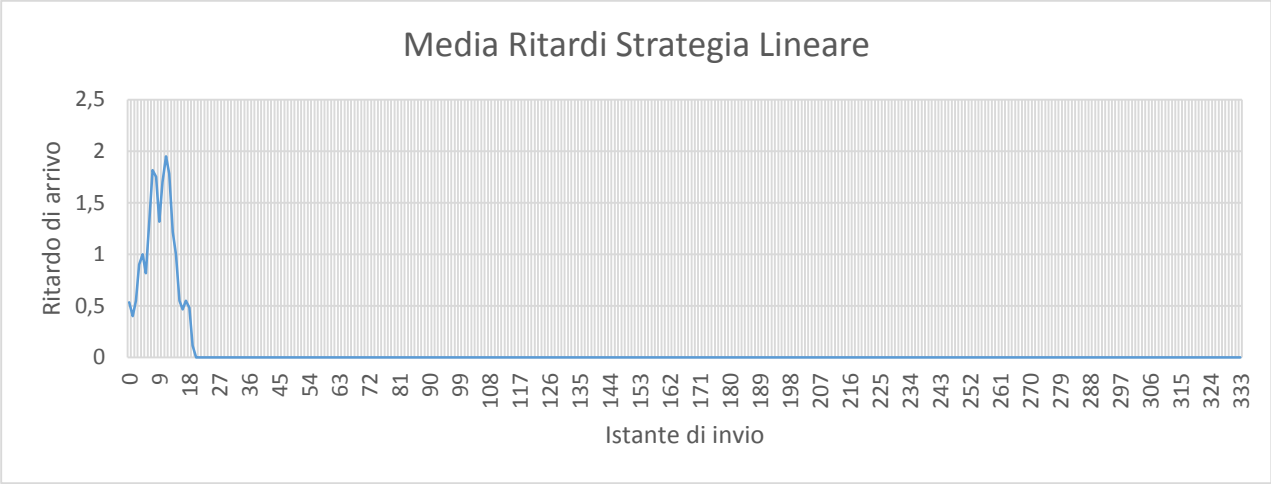

Figura B.2: Andamento temporale ritardi traffico costante strategia lineare

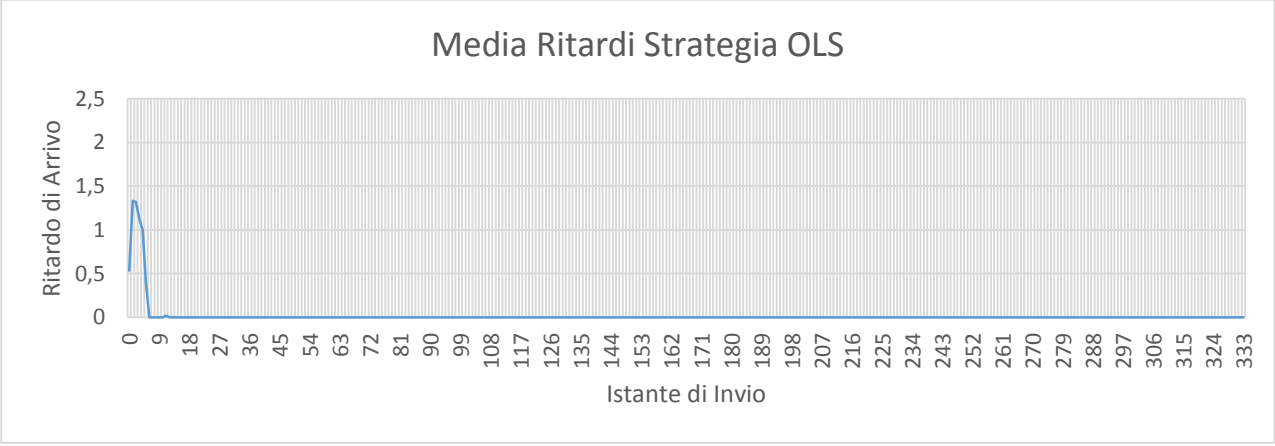

Figura B.3: Andamento temporale ritardi traffico costante strategia OLS

Appendice B

Traffico variabile

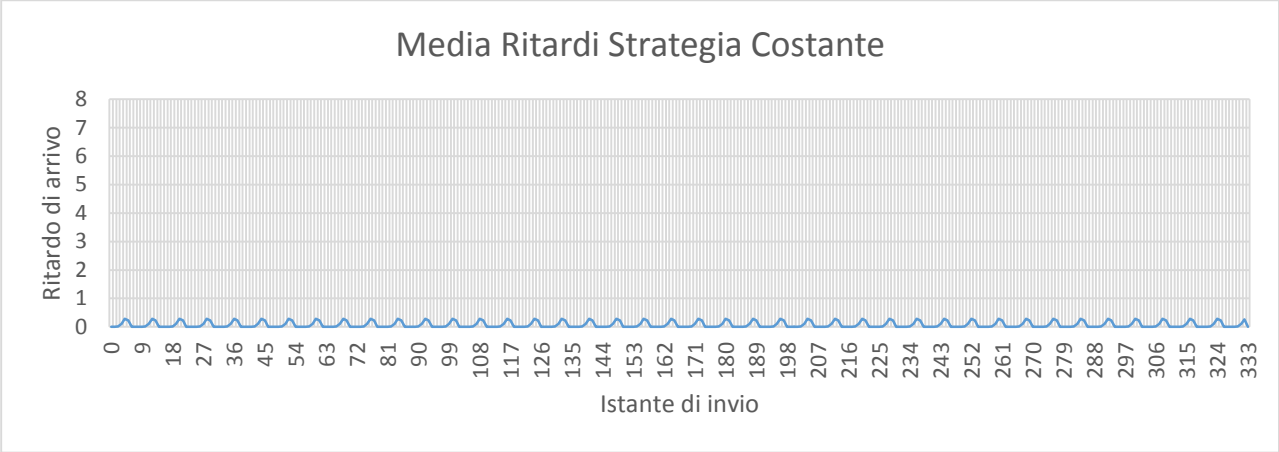

Figura B.4: Andamento temporale ritardi traffico variabile strategia costante

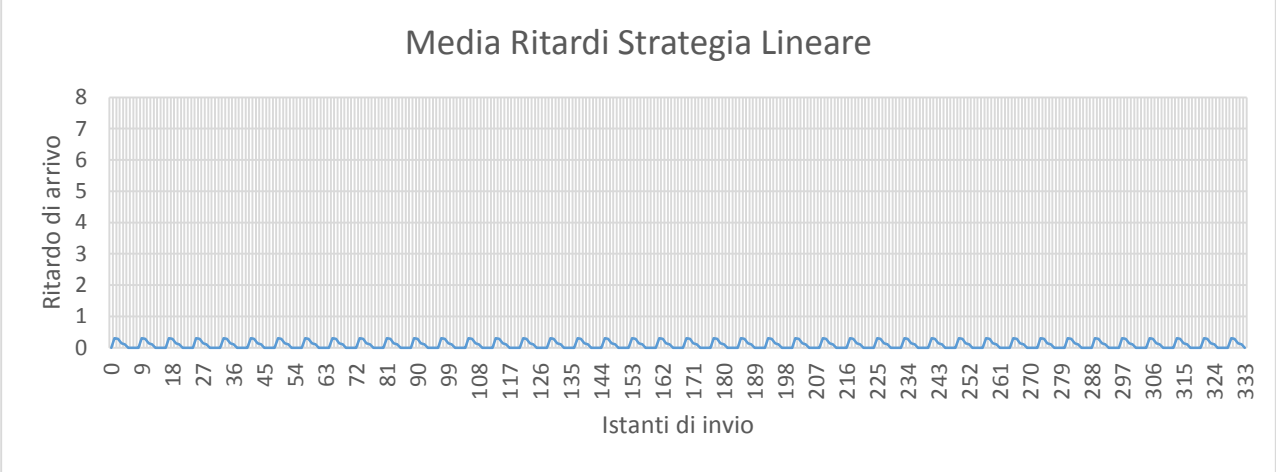

Figura B.5: Andamento temporale ritardi traffico variabile strategia lineare

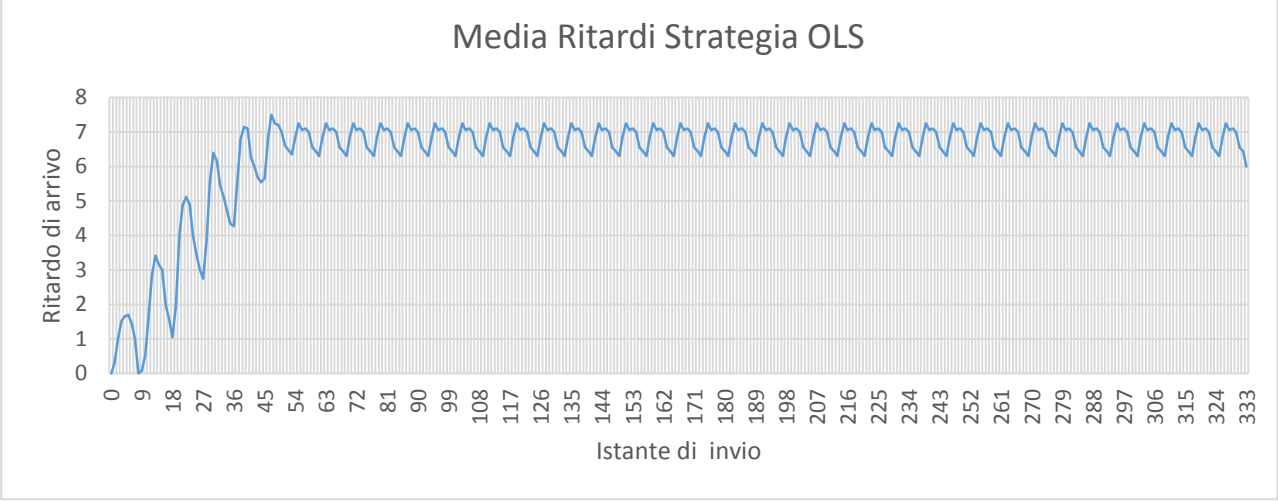

Figura B.6: Andamento temporale ritardi traffico variabile strategia OLS

Appendice B

Traffico altamente variabile

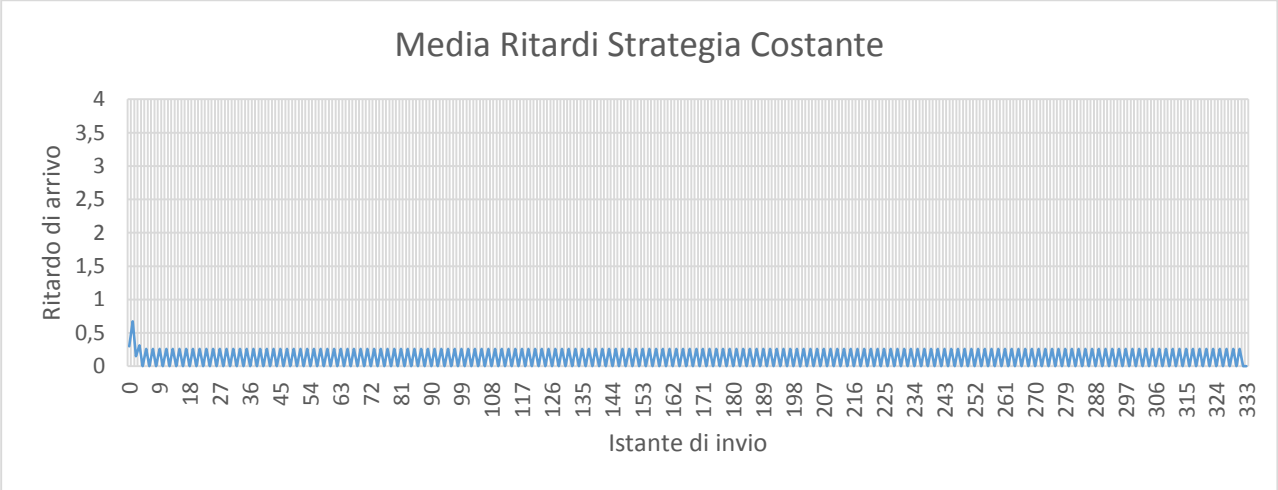

Figura B.7: Andamento temporale ritardi traffico altamente variabile strategia costante

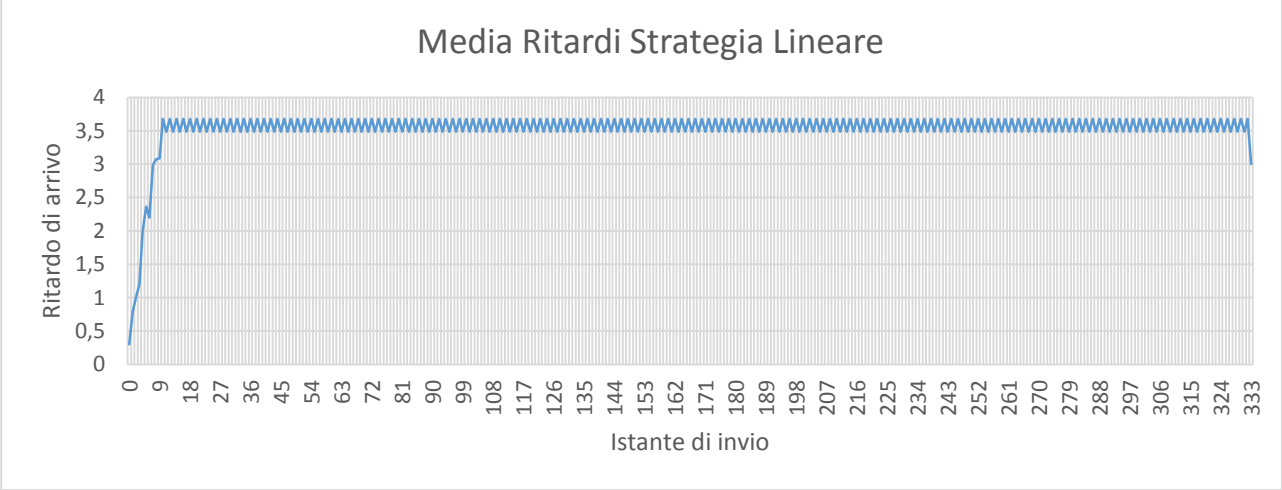

Figura B.8: Andamento temporale ritardi traffico altamente variabile strategia lineare

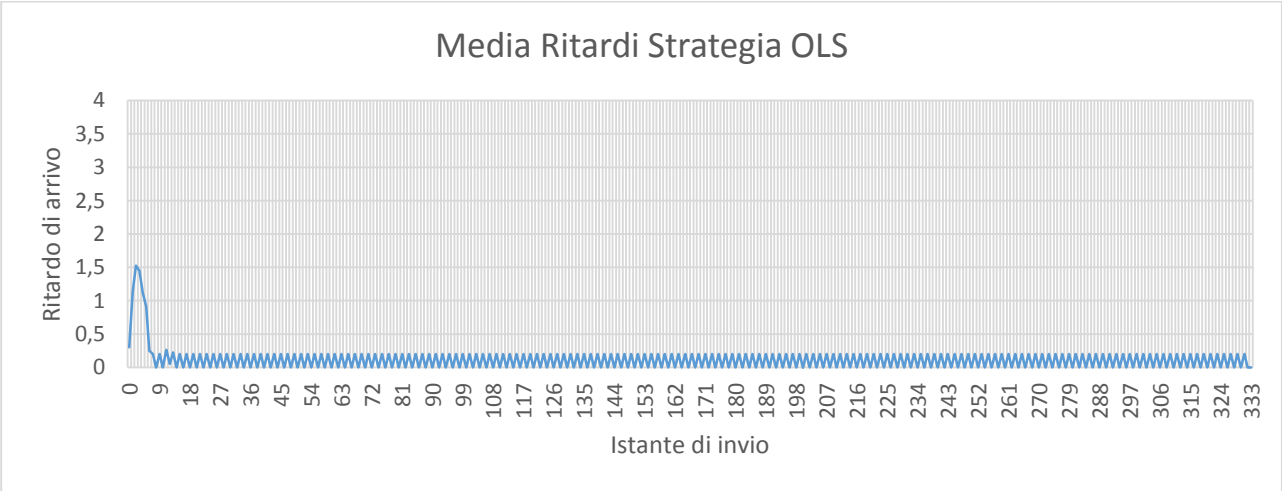

Figura B.9: Andamento temporale ritardi traffico altamente variabile strategia OLS

Lista Immagini

Lista Tabelle

Bibliografia

[01] "Price Water House Coopers Information Security Breaches Survey 2013", 2013,

https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/200455/bis-13-p184-2013-information-security-breaches-survey-technical-report.pdf

[02] "Report Efficienza energetica Telecom", 2010,

http://energia24club.it/01NET/Card_Library/TelecomItalia_efficienza_energetica.pdf

[03] "Report Telecom 2012", 2012,

http://www.telecomitalia.com/content/dam/telecomitalia/it/archivio/documenti/Sostenibilita/Report_di_sostenibilita/2012/BilancioSostenibilita2012-ITA.pdf

[04] **Diego Reforgiato Recupero**, "*Toward a green economy"*, 2013,

http://www.sciencemag.org/content/339/6127/1533.full

[05] *Progetto ECONET*, *www.econet-project.eu*

[06] **Gianluca Stringhini**, "Studio di un protocollo per l'intrusion prevention distribuita", 2008

[07] **Dorothy E. Denning**, "An intrusion-detection Model", 1987

[08] **Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin, Wei-Yang Lin**, "Intrusion detection by machine learning: A review", 2009

[09] Progetto Snort, *http://www.snort.org/*

[10] **Jamil Farshchi**, "Intrusion Detection FAQ: Statistical based approach to Intrusion Detection", *http://www.sans.org/security-resources/idfaq/statistic_ids.php*

[11] **Ganesh Kumar Varadarajan**, **Manuel Humberto Santander Peláez**, "Web Application Attack Analysis Using Bro IDS", 2012, *http://www.sans.org/reading-room/whitepapers/detection/web-application-attack-analysis-bro-ids-34042*

[12] **Pritika Mehra**, "A brief study and comparison of Snort and Bro Open Source Network Intrusion Detection Systems", 2012,

http://www.ijarcce.com/upload/august/4-A%20brief%20study%20and%20comparison%20of.pdf

[13] Progetto Prelude IDS, *https://www.prelude-ids.org/*

[14] **Joshua S. Whitea, Thomas T. Fitzsimmonsb, Jeanna N. Matthewsc,** "Quantitative Analysis of Intrusion Detection Systems: Snort and Suricata", 2013,
http://people.clarkson.edu/~jmatthew/publications/SPIE_SnortSuricata_2013.pdf

[15] Suricata mail list, *https://lists.openinfosecfoundation.org/mailman/listinfo*

[16] Sito ufficiale suricata, *http://suricata-ids.org/*

[17] **Marco Mezzalama, Gianluca Oglietti**, "Acquisizione del traffico IP da una rete ad elevato throughput utilizzando un sistema Open Source: tecnologie e problematiche", 2011,

http://security.polito.it/doc/public/torsec_aica2011_sniffer.pdf

[18] **Eric Leblond**, "Suricata to 10 Gbps and beyond", 2012, *https://home.regit.org/2012/07/suricata-to-10gbps-and-beyond/*

[19] **Mauno Pihelgas**, "A comparative analysis of open-source intrusion detection system", 2012

[20] **Università degli studi di Pisa**, "Lezione sui minimi quadrati",

http://www.dm.unipi.it/syl/upload_doc/201.Lez13.pdf

[21] **Mauro Migliardi e Alessio Merlo**, "Improving energy efficiency in Distributed Intrusion Detection Systems".

[22] **Eugene Albin**, "A comparative analysis of the snort and suricata intrusion-detection systems", 2011, *http://www.hsdl.org/?view&did=691228*

[23] **Raffaele Bolla, Roberto Bruschi, Andrea Ranieri**, "Green support for PC-based software router: Performance evaluation and modeling", 2009,
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.2791&rep=rep1&type=pdf

Consultati

[24] F. R. Yu, X. Zhang, V. C. M. Leung (Eds), "Architectural Design of An Energy-efficient Router", 2012, *https://www.econet-project.eu/Publications/PublicationsList*

[25]**Václav Mikolášek**, "Intrusion Detection Systems – State of the Art Report", 2008,
http://www.vmars.tuwien.ac.at/documents/extern/2391/report.pdf

[26] **Stephanie Forrest, Stephen Hofmeyr, and Anil Somayaji**, "Computer immunology", 1997

[27] **Robert Koch**, "Towards Next-Generation Intrusion Detection", 2011,

http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05954707

[28] Real time web monitor, *http://www.akamai.com/html/technology/dataviz1.html*

[29] **Akamai**, "The state of the internet", 2013, *http://www.akamai.com/dl/akamai/akamai-soti-q313.pdf?WT.mc_id=soti_Q313*

[30] Suricata,
https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricata_Developers_Guide

[31] Suricata, *https://doxygen.openinfosecfoundation.org/*

[32] Suricata, *https://home.regit.org/tag/suricata/*

[33] Suricata, *http://blog.inliniac.net/*

[34] Snort, *http://www.snortattack.org*

[35] Snort, *http://web.quipo.it/scrawl/snort_article.html*

[36] Snort, *http://www.sans.org/security-resources/idfaq/statistic_ids.php*

[37] Snort, *http://webpages.cs.luc.edu/~pld/courses/447/sum08/class6/biles.spade.pdf*

[38] Snort, *http://www.di.unisa.it/~ads/corso-security/www/CORSO-0001/snort/content.htm*

[39] Attacchi: *http://cnc.ucr.edu/security/glossary.html*

[40] Attacchi: *http://www.ippari.unict.it/wikippari/wiki/IDS_rilevamento_intrusioni*

[41] Attacchi: *http://wireline.telecomitalia.it/TIPortale/docs/innovazione/012005/lamastra.pdf*

[42] Rete trasporto TI, *http://policom.dei.polimi.it/education/comunicazioni/Reti%20di%20trasporto%20TI%20PoliMI%2019%20ottobre%202009.pdf*

[43] **Christian Callegari**, "Statistical approaches for network anomaly detection", 2009

[44] **Gerardo Lamastra, Luca Viale**, "Tecnologie innovative per il rilevamento e il contrasto degli attacchi informatici"

[45] **Jungwon Kim, Peter Bentley, Uwe Aickelin, Julie Greensmith, Gianni Tedesco, Jamie Twycross**, "Immune system approaches to intrusion detection – a review", 2007

[46] **Beatrice Lazzerini**, "Introduzione alle reti neurali"

[47] **Wei Li,** "Using Genetic Algorithm for Network Intrusion Detection"

[48] Java: *http://docs.oracle.com/javase*

[49] **Francesco Licandro**, "Architettura di un router"

[50] OSSEC, *http://www.ossec.net/*

[51] Barnyard, *http://barnyard.sourceforge.net/*