UNIVERSITÀ DI PISA
FACOLTÀ DI INGEGNERIA

Corso di Laurea in
Ingegneria delle Telecomunicazioni

Tesi di Laurea Specialistica

Tecniche di Network Anomaly Detection Multidimensionale

Relatori:
Prof. Michele Pagano
Prof. Stefano Giordano
Ing. Christian Callegari
Ing. Teresa Pepe

Candidato:
Alessandro Casella

Anno Accademico 2011/2012

Sommario

La continua diffusione delle tecnologie che permettono di trasmettere informazioni attraverso la rete Internet a velocità sempre maggiori e la costante crescita dell'impiego della rete stessa per fornire servizi rendono necessario lo sviluppo di tecniche sempre più veloci ed efficaci per la protezione da possibili attacchi informatici.

In questa tesi focalizziamo l'attenzione sugli Intrusion Detection System (IDS), quei sistemi utilizzati per identificare attività dannose non autorizzate che hanno superato eventuali misure di prevenzione. In particolare vengono trattati sistemi che eseguono un'analisi statistica, con lo scopo di rilevare anomalie nell'andamento di parametri relativi al traffico di rete.

L'obiettivo è quello di sviluppare nuovi IDS "multidimensionali", che monitorano contemporaneamente più serie temporali di dati con l'impiego di un unico algoritmo di change detection, per individuare intrusioni di tipo diverso.

SOMMARIO

Dalle prove sperimentali eseguite, sono state ottenute prestazioni simili a quelle proprie di sistemi monodimensionali applicati separatamente a ciascuna sequenza analizzata: il nuovo approccio dimostra quindi vantaggi in termini di costo computazionale.

Ringraziamenti

Innanzitutto voglio ringraziare i miei genitori e i miei nonni, persone che mi hanno permesso di affrontare il mio percorso di studio con estrema tranquillità, fornendomi in ogni momento l'appoggio necessario.

Un ringraziamento va a mio fratello e a tutti i miei amici, capaci di regalarmi momenti di divertimento e distrazione da periodi di studio talvolta faticosi e stressanti.

Ringrazio i professori Michele Pagano e Stefano Giordano, che mi hanno seguito nella realizzazione della tesi.

Doverosa e sentita è la gratitudine a chi mi ha fornito, sempre con estrema disponibilità, le conoscenze e gli strumenti necessari per portare a termine questo elaborato, Christian Callegari e Teresa Pepe, e a tutti i ragazzi del laboratorio di reti di telecomunicazioni, compagni delle mie giornate di lavoro.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Con il continuo sviluppo di tecnologie che permettono di trasmettere informazioni attraverso la rete Internet a velocità sempre più elevate e con il costante aumento dell'impiego della rete stessa per fornire servizi di qualsiasi tipo (un esempio banale potrebbe essere la crescente diffusione di negozi online), si sta rendendo necessario lo sviluppo di tecniche sempre più veloci ed efficaci per la protezione da possibili attacchi informatici. In questo caso ci riferiamo agli Intrusion Detection System (IDS), ossia quei sistemi di allarme che vanno alla ricerca delle intrusioni che già hanno oltrepassato eventuali misure preventive adottate, quali firewall e cifratura dei dati.

Data l'evoluzione delle metodologie di attacco, è importante utilizzare metodi in grado di individuare attività anomale di qualsiasi natura, sia note che non; ciò viene fatto attraverso IDS di tipo anomaly based (il cui significato verrà spiegato meglio in seguito), che analizzano l'andamento di alcune caratteristiche del traffico di rete, con lo scopo di scoprire devia-

zioni significative rispetto ad un modello di comportamento normale.

Poiché, come vedremo nel capitolo 1, esistono vari tipi di azioni malevole, è preferibile monitorare più descrittori di traffico, in modo da rilevare quante più intrusioni possibile. Una possibile strada è quindi quella di applicare singolarmente uno stesso IDS ogni volta ad una serie diversa di dati: definiamo questo metodo "monodimensionale".

Lo scopo del lavoro svolto nel corso di questa tesi è invece quello di verificare se è possibile adottare un approccio "multidimensionale", ossia andare a ricercare anomalie in sequenze temporali di natura diversa, tramite un singolo algoritmo che le analizzi tutte contemporaneamente. Il vantaggio sarebbe quello di dover eseguire l'anomaly detection una sola volta, riducendo così il costo computazionale e il tempo necessario per individuare eventuali attacchi.

La struttura di questo elaborato è la seguente: il capitolo 1 ha lo scopo di fare una panoramica sulle tipologie di attacchi informatici e illustrare una classificazione degli IDS, in modo da individuare il ruolo dei sistemi che vedremo nel resto della trattazione.

Nel capitolo 2 viene spiegato il concetto di "mondimensionalità", considerato nell'ambito dell'anomaly detection; vedremo esempi pratici di IDS monodimensionali, dove (in alcuni) si fa uso di una particolare struttura dati, denominata

Count-Min Sketch, che è stata impiegata anche nei sistemi multidimensionali realizzati.

Il capitolo 3 è dedicato alla descrizione dei test statistici di change detection presenti in letteratura, oppure, come nel caso del CuSum, creati estendendo l'algoritmo monodimensionale corrispondente, che sono stati utilizzati per sviluppare IDS multidimensionali basati su sketch.

Nel capitolo 4 vengono presentati i risultati sperimentali ottenuti testando i metodi di detection realizzati sul dataset costituito da tracce di traffico appartenenti all'archivio MAWI, che vengono illustrate nel capitolo stesso, insieme ai dettagli implementativi dei vari algoritmi.

Infine, dopo le considerazioni conclusive, si trovano due appendici: una descrive la procedura con cui le tracce di traffico utilizzate per la valutazione delle prestazioni sono state convertite nel formato adatto ad essere letto dai nostri IDS, l'altra riporta il codice sorgente dell'IDS basato sul CuSum multidimensionale.

Capitolo 1

Nozioni generali sugli IDS

In questo capitolo vengono presentati gli Intrusion Detection System (indicati con l’acronimo IDS), ossia i metodi impiegati per individuare la presenza di attacchi a sistemi informatici. Innanzitutto viene fatta una panoramica sui tipi di intrusioni, per poter comprendere meglio l’obiettivo dell’utilizzo degli IDS; vedremo poi una classificazione di tali sistemi, in modo da individuare il ruolo degli algoritmi di anomaly detection trattati nel resto di questo elaborato.

1.1 Attacchi e IDS

Un attacco informatico può essere visto come una qualsiasi attività o accesso non autorizzato in grado di creare problemi agli utenti di un sistema informatico, come ad esempio impedire la disponibilità di un servizio offerto attraverso la rete Internet oppure scoprire e corrompere dati sensibili. Gli autori di tali azioni illecite, chiamati *intruder* o *attacker*, possono essere classificati come segue [8]:

- *masquerader*: è un soggetto che, non essendo autorizzato, si finge un utente legittimo per poter accedere al sistema di interesse.
- *misfeasor*: utente legittimo che compie azioni che, secondo i propri privilegi, non sarebbero permesse.
- *clandestine user*: intruso non autorizzato che, ottenuto il controllo dei sistemi di monitoraggio, cancella le tracce della sua presenza.

Le attività maligne eseguite da tali soggetti consistono in vari tipi di azioni, con diversi obiettivi [8]:

- *packet sniffing*: intercettazione passiva del traffico che transita sulla rete, ad esempio per scoprire password o altre informazioni.
- *snooping* e *downloading*: scaricare materiale da una macchina.

- modifiche (non autorizzate) a dati e registri.
- *spoofing*: impersonare altri utenti (ad esempio cambiando l'indirizzo IP sorgente nell'intestazione di un pacchetto IP).
- *jamming* e *flooding*: inondare la macchina vittima con un numero di pacchetti tale da esaurirne le risorse e creare un malfunzionamento.
- esecuzione di codice maligno su una macchina vittima.
- risalire a password o chiavi.

Queste varietà di attacchi possono essere raggruppate in quattro classi [19]:

- *probe* o *scan*: attacco volto a scoprire informazioni sulla vittima (ad esempio quali sono le porte aperte) in modo da sfruttarle per attacchi successivi.
- *Denial of Service* (DoS): attacco cha ha lo scopo di impedire la corretta erogazione di uno o più servizi offerti dalla rete, ostacolando così gli utenti che richiedono tali servizi; tipicamente le modalità con cui viene realizzato sono quelle di esaurire le risorse del sistema (inviando un numero eccessivo di richieste, in modo che non possano essere processate quelle degli utenti leggitimi) e saturare la rete.

- *Remote to Local* (R2L): accesso non autorizzato da una macchina remota (eseguito da un *attacker* di tipo *masquerader*).
- *User to Root* (U2R): utilizzo non autorizzato dei privilegi di super-user da parte di un utente che non li possiede (eseguito da un *attacker* di tipo *misfeasor*).

Per proteggere una possibile vittima da queste attività potenzialmente dannose esistono protocolli di sicurezza e dispositivi come i firewall; tuttavia le soluzioni preventive impiegate da questi possono fallire, quindi servono metodi di riconoscimento di tali intrusioni. Gli Intrusion Detection System sono proprio quei sistemi che monitorano reti e host con l'obiettivo di scoprire eventuali attacchi, in modo da poter adottare le contromisure necessarie ad impedirne il successo.

1.2 Tipologie di IDS

Gli IDS possono essere classificati differenziandoli in base a quattro parametri, che tengono conto dell'ambito di appilcazione, dalla tecnica impiegata per la ricerca delle intrusioni, della modalità con cui vengono considerate le relazioni tra eventi successivi e dell'architettura degli elementi che li compongono.

Secondo quanto appena detto, le distinzioni sono le seguenti [8]:

- Host based IDS vs. Network based IDS
  - *Host based IDS* (HIDS): ha l'obiettivo di rivelare attacchi rivolti ad uno specifico host, di conseguenza dipende dall'architettura e dal sistema operativo del sistema monitorato; processa informazioni di alto livello (ad esempio le *system call* e i comandi eseguiti su un computer); è efficace nel rilevare intrusioni che avvengono all'interno di una rete (l'attacker si trova sulla stessa rete locale della vittima).
  - *Network based IDS* (NIDS): ha l'obiettivo di rivelare attacchi rivolti ad una rete nel suo complesso, quindi è indipendente dall'architettura che caratterizza gli host che ne fanno parte (può infatti essere diversa da uno all'altro); i dati analizzati (di più basso livello rispetto ad un HIDS) sono ricavati dai pacchetti scambiati con la rete monitorata, così da essere efficace nell'individuare intrusioni provenienti dall'esterno della LAN.
- Misuse based IDS vs. Anomaly based IDS
  - *Misuse based IDS*: basa la propria ricerca su un database relativo ad attacchi noti, andando a vedere se i dati analizzati presentano le caratteristiche di un'attività malevola conosciuta; sebbene sia preciso nell'identificazione di tali intrusioni, non è in

grado di scoprire attacchi di cui non si conoscono i caratteri distintivi.

- *Anomaly based IDS*: eseguendo un confronto con un modello di traffico normale (costruito nella fase di training durante la quale è stato osservato un comportamento di cui siamo sicuri essere privo di attacchi), classifica l'attività monitorata come normale o anomala; è quindi capace di individuare anche attacchi sconociuti, ma può generare falsi allarmi.

- Stateless IDS vs. Stateful IDS
  - *Stateless IDS*: tratta ogni evento in modo indipendente dagli altri, è quindi di sempilce progettazione e presenta una elevata velocità di processing, ma in alcuni casi si rivela poco efficiente.
  - *Stateful IDS*: mantenedo informazioni relative ai fatti passati, l'effetto di un certo evento dipende da ciò che è accaduto fino alla sua analisi; è un sistema più complesso da mettere in pratica, ma nello stesso tempo dimostra una maggiore efficacia in presenza di attacchi distribuiti nel tempo.

- Centalized IDS vs. Distributed IDS
  - *Centalized IDS*: semplice da realizzare in quanto costituito da un'unica macchina che esegue tutte

le operazioni (monitoraggio, raccolta e analisi dei dati e generazione degli allarmi); ha lo svantaggio di una limitata robustezza poichè presenta un unico punto da oltrepassare per far andare a buon fine un attacco.

– *Distributed IDS*: le funzionalità sono divise tra diversi elementi; sensori distribuiti nella rete generano gli eventi, i quali vengono monitorati da una console e registrati da un motore centrale che si occupa di generare eventuali allarmi.

I particolari algoritmi di detection trattati nel resto di questo elaborato sono di tipo network based (le informazioni processate vengono ricavate leggendo vari campi dei pacchetti che transitano sulla rete) e anomaly based; vediamo quindi alcuni aspetti di questa categoria.

Un anomaly based IDS si propone di eseguire l'*anomaly detection* con un approccio statistico: secondo quanto riportato nella descrizione del primo sistema di questo tipo (IDES, ideato da Denning [11]), durante una fase iniziale di apprendimento si osserva l'andamento normale (privo di anomalie) del sistema monitorato; caratterizzando tale comportamento in termini di metriche statistiche e modelli, vengono costruiti i profili con i quali confrontare le osservazioni successive. Una metrica è una variabile aleatoria che rappresenta una misura quantitativa accumulata in un certo periodo di tempo; durante

la fase di monitoraggio del sistema vengono raccolte osservazioni di tale variabile aleatoria, le quali vengono valutate come anomale o meno eseguendo test che impiegano i profili.

In un IDS di tipo network based si vuole caratterizzare il comportamento della rete, quindi le metriche rappresentano i descrittori di taffico, ossia parametri ricavabili dall'analisi dei pacchetti: lunghezza, tempo di interarrivo e numero di pacchetti (totali o solamente di un determinato tipo), numero di porte TCP o UDP diverse utilizzate, e altri.

Capitolo 2

Anomaly detection monodimensionale

Dopo la panoramica generale sugli IDS presentata nel capitolo 1, vediamo adesso come vengono tipicamente impiegati nell'anomaly detection i descrittori di traffico citati al termine della sezione 1.2.

In questo capitolo vengono illustrati alcuni approcci adottati in anomaly based IDS, che per la modalità di utilizzo delle metriche possiamo definire "mondimensionali" (vedremo meglio in seguito il significato di tale affermazione).

2.1 Il concetto di monodimensionalità

Abbiamo visto in precedenza che gli IDS appartenenti alla categoria di nostro interesse (network based e anomaly based) cercano di rilevare anomalie nel traffico di rete monitorando parametri che sono in grado di descriverne l'andamento (le metriche o descrittori di traffico). Come è stato già detto, vi sono moltissime possibilità di selezione di tali metriche, e la modellizzazione del comportamento della rete sarà tanto più accurata quanto maggiore è il numero di quelle che vengono considerate; tuttavia, per ridurre la complessità computazionale, è opportuno sceglierne il minor numero possibile che garantisca un tasso di errore (inteso come una mancata rivelazione di intrusioni) accettabile.

I sistemi che considerano un solo descrittore di traffico per volta, ossia applicano singolarmente per ogni metrica l'algoritmo di anomaly detection in essi implementato, vengono definiti "monodimensionali". È chiaro che attacchi di tipo diverso possano provocare anomalie in parametri differenti; quindi, se si vogliono scoprire le diverse intrusioni, si ha la necessità di ripetere l'esecuzione dello stesso algoritmo per quelle metriche che ne sono influenzate.

Gli IDS che adottano questo approccio rappresentano sia un punto di partenza per lo sviluppo di nuovi metodi di anomaly detection multidimensionale (che operano contemporaneamente su più di una metrica per volta), sia un metro di giudi-

zio per valutare le prestazioni di questi ultimi. Vediamo quindi alcuni esempi pratici di anomaly detection monodimensionale.

2.2 IDS basati su istogrammi

Algoritmi di anomaly detection presentati in [13] e in [19] fanno uso di istogrammi per valutare la distribuzione di alcune *feature* del traffico, in modo da identificare le anomalie con eventuali cambiamenti significativi rispetto a distribuzioni di riferimento (costruite a partire dai dati di training).

Una *feature* è una caratteristica del traffico, più concretamente uno dei campi presenti nelle intestazioni dei pacchetti (sia del livello IP che del livello di trasporto); un istogramma rappresenta il numero di flussi, pacchetti o byte (osservati durante un intervallo di tempo prestabilito, detto *time bin*) in funzione dei possibili valori di una o combinazioni di più feature [13] (un esempio potrebbe essere il numero di flussi osservati nel traffico analizzato in funzione dell'indirizzo IP destinatario).

Negli IDS che impiegano questa rappresentazione dei dati si possono individuare i seguenti passi principali:

- Selezione delle feature: le feature vengono scelte in base ai possibili attacchi che si pensa possano essere eseguiti, infatti feature differenti sono utili per la rivelazione di anomalie di tipo diverso.

- Costruzione degli istogrammi di riferimento: durante la fase di training si osserva del traffico che si conosce essere privo di anomalie e, per tutte le feature scelte, si costruiscono gli istogrammi di riferimento con i quali verranno confrontati quelli relativi alle attività da classificare.

- Costruzione degli istogrammi di test: osservando il traffico da monitorare, in ogni time bin, vengono costruiti gli istogrammi relativi alle feature interessate.

- Classificazione: sempre in ogni time bin, ciascun istogramma viene confrontato con l'istogramma di riferimento riguardante la sua stessa feature; tale confronto viene eseguito calcolando la distanza, secondo opportune funzioni (quali la distanza euclidea, la distanza di Manhattan, la distanza di Mahalanobis, la divergenza di Kullback-Leibler e la divergenza di Jensen-Shannon [19]), tra i due vettori che rappresentano i valori dei due istogrammi in esame (quello di riferimento e quello di test). Se la distanza calcolata supera una determinata soglia, l'attività monitorata nel time bin corrente viene classificata come anomala, altrimenti viene ritenuta normale.

È possibile osservare che algoritmi di questo tipo tengono sotto controllo un solo parametro relativo al traffico osservato (il numero di flussi, pacchetti o byte), andando a vedere come esso si distribuisce lungo i valori della feature scelta. Per

poter individuare intrusioni di tipo diverso, il descrittore del traffico resta lo stesso, mentre viene cambiata la natura della feature; inoltre il calcolo della distanza tra gli istogrammi di test (uno per ogni feature) e i corrispondenti istogrammi di riferimento viene ripetuto separatamente per ognuno di essi. Questo è il motivo per cui tali tecniche di anomaly detection sono classificate come monodimensionali.

2.3 Il Count-Min Sketch

Il *Count-Min Sketch* (CMS) è una struttura introdotta per la prima volta da Cormode e Muthukrishnan in [10] con lo scopo di "riassumere" stream di dati, in modo da semplificare il calcolo di funzioni e quantità statistiche relative ad essi; nel caso degli IDS permette di organizzare i valori dei descrittori di traffico osservati durante un time bin in funzione di aggregati dei valori di una feature (che da ora in poi verrà indicata con la parola "chiave").

Lo stream di dati in input al sistema può infatti essere visto come una serie temporale di elementi s_i (dove i rappresenta l'indice temporale) costituiti da una coppia chiave - valore (o peso) [15]: $s_i = (k_i, v_i)$.

Il count-min sketch è più concretamente una matrice di D righe e W colonne; ad una generica riga di indice d (compreso tra 0 e $D - 1$) è associata una funzione hash h_d (funzioni

diverse per ciascuna riga, scelte in modo da distribuire in modo uniforme le chiavi k_i nella tabella e ridurre le collisioni [15]) con possibili valori di uscita nell'intervallo $(0, 1, \ldots, W-1)$.

Durante la fase di anomaly detection, tutti i valori dello sketch vengono inizializzati a 0 all'inizio di ciascun time bin; ad ogni nuovo elemento (k_i, v_i) che si osserva nello stream di dati, per tutte le righe dello sketch si calcola l'uscita della funzione hash corrispondente applicata alla chiave k_i: $h_d(k_i)$ per $d = 0, \ldots, D-1$. In ciascuna riga, i valori così calcolati rappresentano l'indice della colonna dello sketch il cui elemento deve essere incrementato sommando il peso v_i.

Riassumendo, se indichiamo con $C[d][w]$ la cella dello sketch posta nella riga d e colonna w, l'operazione di incremento eseguita per ogni dato (k_i, v_i) in ingresso è la seguente:

$$C[d][h_d(k_i)] = C[d][h_d(k_i)] + v_i \qquad \forall d = 0, \ldots, D-1 \quad (2.1)$$

In questo modo, al termine del time bin, il valore della generica cella $C[d][w]$ corrisponderà alla somma dei pesi v osservati durante tale intervallo temporale, associati all'aggregato delle chiavi k che, messe in ingresso alla funzione hash d-esima h_d, danno come risultato w. Facciamo un esempio pratico per comprendere meglio: lo stream di dati è ottenuto leggendo l'intestazione dei pacchetti che transitano in un determinato punto della rete, la chiave k corrisponde all'indirizzo IP destinatario, mentre il peso v è rappresentato dal numero di byte.

In questo caso particolare, $C[d][w]$ coincide con il numero totale di byte transitati durante il time bin appena trascorso, destinati all'insieme di host che hanno un indirizzo IP il cui hash (calcolato con la funzione h_d) è w.

2.4 IDS basati sullo sketch

Una volta che la struttura dello sketch è stata riempita come abbiamo visto nella sezione 2.3 (al termine del time bin), è possibile applicarvi un qualunque algoritmo di anomaly detection.

In [9] vengono descritti due di questi metodi; quello più semplice consiste nel calcolare il modulo della differenza tra ciascun elemento dello sketch corrente e il corrispondente elemento dello sketch di riferimento, ossia quello relativo all'ultimo intervallo temporale non anomalo:

$$d_{ij} = \left| C[i][j] - C^{ref}[i][j] \right| \qquad \begin{array}{l} \forall i = 0, \ldots, D-1 \\ \forall j = 0, \ldots, W-1 \end{array} \tag{2.2}$$

se queste distanze superano una certa soglia per almeno un determinato numero H di righe dello sketch, il time bin viene considerato anomalo e si passa alla fase di identificazione delle chiavi responsabili dell'anomalia. Il secondo metodo presentato in [9] è più complesso e, senza entrare nei dettagli, consiste

nel confrontare (con l'ausilio della Jensen-Shannon Divergence) due distribuzioni empiriche, calcolate una a partire dallo sketch reale costruito nel time bin corrente, e l'altra utilizzando la predizione dello stesso sketch ottenuta elaborando la storia passata.

Altri esempi di impiego del count-min sketch in IDS si trovano in [19] e in [15], nei quali l'algoritmo di change-detection applicato è il CuSum, che verrà descritto nella sezione 2.5.

2.5 CuSum monodimensionale applicato a IDS basati sullo sketch

L'algoritmo CuSum (acronimo che sta per Cumulative Sum) è uno dei metodi di sequential change point detection: si propone infatti di scoprire, con il minor ritardo possibile, cambiamenti nella distribuzione di una sequenza di dati monitorata y_k [15].

È bene precisare che esistono varie formulazioni di tale algoritmo, che possono essere distinte in parametriche e non parametriche.

La versione parametrica del CuSum suppone di conoscere le densità di probabilità dei dati prima e dopo l'eventuale cambiamento, indicate rispettivamente con $f_{\Theta_1}(y_k)$ e $f_{\Theta_2}(y_k)$. La statistica di decisione g_k viene inizializzata a 0 e aggiornata per ogni intervallo temporale k, come indicato nell'espressione 2.3:

$$\begin{cases} g_0 = 0 \\ g_k = \max\left\{0, g_{k-1} + \log \frac{f_{\Theta_1}(y_k)}{f_{\Theta_2}(y_k)}\right\} \qquad per\ k \geq 1 \end{cases} \tag{2.3}$$

La quantità $\log \frac{f_{\Theta_1}(y_k)}{f_{\Theta_2}(y_k)}$ rappresenta il logaritmo del rapporto di verosimiglianza (spesso indicato con l'acronimo LLH-ratio che sta per log-likelihood ratio), che assume mediamente valori negativi prima del cambiamento e positivi dopo: quando infatti la d.d.p. dei dati coincide con la funzione f_{Θ_1} si ha (in media) $f_{\Theta_1}(y_k) > f_{\Theta_2}(y_k)$, viceversa quando la d.d.p. è f_{Θ_2} vale la relazione $f_{\Theta_1}(y_k) < f_{\Theta_2}(y_k)$. Di conseguenza, la variabile di decisione g_k si mantiene vicino a 0 prima del cambiamento, e cresce dopo un'eventuale variazione della distribuzione: si decide quindi di rivelare un'anomalia quando $g_k > h$ (dove h è un valore di soglia prestabilito). La figura 2.1 riporta un esempio degli andamenti temporali dei dati y_k e della statistica di decisione g_k calcolata nel modo appena descritto.

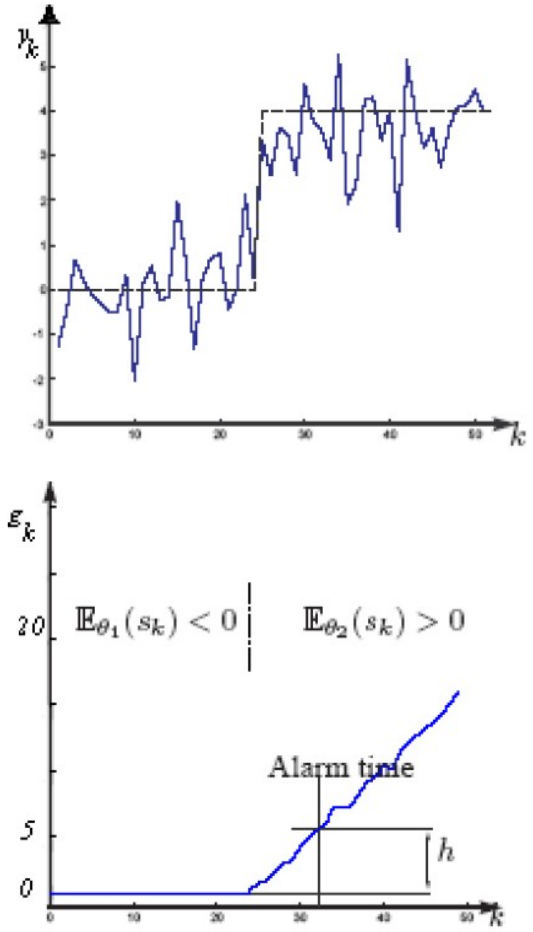

Figura 2.1: Andamento del CuSum parametrico

Purtroppo l'ipotesi di conoscere la forma delle densità di probabilità dei dati prima e dopo un eventuale cambiamento non è vera quando abbiamo a che fare con il monitoraggio del traffico di rete. Per questo motivo, nel caso che interessa gli IDS, dobbiamo far ricorso alle formule CuSum non parametriche, le quali sono appunto indipendenti da tale conoscenza. In generale, tutte le varianti CuSum di questo tipo possono essere espresse sostituendo il logaritmo del rapporto di verosimiglianza con una generica funzione $L(y_k)$ nell'equazione 2.3, che quindi diventa:

$$\begin{cases} g_0 = 0 \\ g_k = \max\left\{0, g_{k-1} + L(y_k)\right\} \qquad per\ k \geq 1 \end{cases} \tag{2.4}$$

La funzione $L(y_k)$ deve essere tale che, come per il LLH-ratio, il suo valor medio $\mathbb{E}\{L(y_k)\}$ risulti negativo prima del cambiamento e positivo dopo [15]. Le possibilità di scelta in questo senso sono numerose, come è stato sperimentato in [19]; noi ci limiteremo a vederne una, quella descritta in [15] e applicata nel corso di questo elaborato per la realizzazione di un IDS monodimensionale con cui confrontare gli algoritmi multidimensionali. L'espressione in esame è la (2.5):

$$L(y_k) = y_k - y_{k-1} - (\hat{\mu}_k + c \cdot \hat{\sigma}_k) \tag{2.5}$$

dove c è un parametro e $\hat{\mu}_k$ e $\hat{\sigma}_k$ rappresentano rispettivamente

la stima della media e della deviazione standard dei dati, ottenute utilizzando l'algoritmo EWMA (Exponentially Weighted Moving Average) esplicitato nelle equazioni 2.6 e 2.7 (α è un parametro che può assumere valori minori di 1, tipicamente 0.9; le stime non sono comunque influenzate significativamente da tale parametro purché venga scelto vicino a 1, come dichiarato in [15]):

$$\hat{\mu}_k = \alpha \cdot \hat{\mu}_{k-1} + (1-\alpha) \cdot y_k \tag{2.6}$$

$$\hat{\sigma}_k^2 = \alpha \cdot \hat{\sigma}_{k-1}^2 + (1-\alpha) \cdot (y_k - \hat{\mu}_k)^2 \tag{2.7}$$

la deviazione standard $\hat{\sigma}_k$ si ottiene semplicemente dalla radice quadrata della varianza stimata $\hat{\sigma}_k^2$. La statistica di decisione assume quindi la forma descritta dalla (2.8):

$$\begin{cases} g_0 = 0 \\ g_k = \max\left\{0, g_{k-1} + y_k - y_{k-1} - \left(\hat{\mu}_k + c \cdot \sqrt{\hat{\sigma}_k^2}\,\right)\right\} & per\ k \geq 1 \end{cases} \tag{2.8}$$

Per quanto riguarda il particolare impiego di questo algoritmo congiuntamente alla struttura dello sketch, ognuna delle sue parti viene applicata separatamente a ciascuna delle celle di tale struttura dati. Alla generica cella dello sketch di posto (i, j) sono quindi associate le seguenti quantità (riferite al k-esimo time bin):

$y_k[i][j]$	valore della cella di posto (i, j) dello sketch nel time bin corrente
$y_{k-1}[i][j]$	valore della cella di posto (i, j) dello sketch nel time bin precedente
$\hat{\mu}_k[i][j]$	stima del valor medio delle caselle di posto (i, j) fino al time bin corrente
$\hat{\sigma}_k^2[i][j]$	stima della varianza delle caselle di posto (i, j) fino al time bin corrente
$g_k[i][j]$	statistica di decisione per la cella di posto (i, j)

Quando la statistica di decisione $g_k[i][j]$ supera un determinato valore di soglia h in una delle colonne (la j-esima in questo caso), viene registrato un allarme per la riga i-esima; se si ha almeno uno di questi allarmi in ognuna delle righe dello sketch, il time bin viene considerato anomalo e si procede quindi all'identificazione delle chiavi responsabili di tale anomalia (procedura che vedremo in seguito) [15].

Il metodo appena descritto è stato impiegato in [15] per monitorare il numero di pacchetti di SYN (quantità corrispondente quindi al peso v che costituisce il valore di incremento dello sketch) associati ai flussi identificati da tre tipi di chiavi, per rilevare anomalie di natura diversa: la prima chiave è costituita dall'indirizzo IP destinatario e dalla porta destinataria ($DIP|DP$), con lo scopo di individuare le vittime di attachi DoS/DDoS di tipo SYN flooding; la seconda viene ottenuta combinando entrambe gli indirizzi IP (sorgente e destinatario:

$SIP|DIP$), per scoprire attacchi di tipo *PortScan* (intrusione volta a testare la vulnerabilità di un host vittima); la terza è composta dall'indirizzo IP sorgente e dalla porta destinataria ($SIP|DP$), al fine di identificare le sorgenti degli attacchi DoS/DDoS oppure attività di *NetScan* (quelle tipiche della fase di diffusione di *worm* verso nuove vittime).

In [19] si applica questa formulazione del CuSum a sei sketch distinti costruiti utilizzando sei diverse chiavi ($SIP|DP$, $DIP|DP$, $SIP|SP$, $SP|DIP$, $SIP|SP|DIP$, $SIP|DP|DIP$), monitorando per ognuno la metrica corrispondente al numero di flussi osservati per ogni chiave.

Come vedremo in seguito, il test del CuSum monodimensionale illustrato in questa sezione è stato invece eseguito, nel corso di questo elaborato, su otto sketch che condividono tutti la stessa chiave (l'indirizzo IP destinatario) e differiscono per il tipo di metrica, per implementare un IDS di riferimento con cui confrontare le prestazioni dei sistemi multidimensionali realizzati.

Capitolo 3

Anomaly detection multidimensionale

La parte innovativa di questa tesi consiste nello sviluppo di Intrusion Detection System multidimensionali, che hanno l'obiettivo di rivelare intrusioni di natura diversa, monitorando contemporaneamente più serie temporali di descrittori di traffico.

Nel capitolo corrente verranno descritti i test statistici presenti in letteratura che permettono di scoprire cambiamenti nella distribuzione di stream di dati vettoriali; questi sono gli algoritmi che sono stati adattati per applicarli nel campo della network anomaly detection, in modo da creare IDS basati sull'impiego dello sketch.

3.1 Test statistici multidimensionali

Ciò che distingue un IDS multidimensionale da un IDS monodimensionale è il fatto di seguire l'andamento simultaneo di più sequenze temporali di dati (che differiscono per tipo da una all'altra), cercando di scoprire in esse eventuali deviazioni dal comportamento normale, con l'impiego di un test la cui esecuzione deve essere unica (ossia tiene conto contemporaneamente dei valori osservati in ciascuna serie).

Serie temporali diverse (di valori scalari) possono essere viste come un'unica serie, nella quale i dati sono di tipo vettoriale, cioè appartengono ad uno spazio multidimensionale. I test di nostro interesse quindi devono fornire una misura statistica della distanza tra vettori: in un anomaly based IDS infatti si vuole quantificare la deviazione tra il vettore che rappresenta i dati correnti e l'insieme di quelli che costituiscono il modello di comportamento non anomalo.

Vediamo quali sono i test di questo tipo.

3.1.1 e-distance

In [18] viene presentato un test per verificare l'uguaglianza in distribuzione tra due insiemi di dati multidimensionali, basandosi sul calcolo della distanza euclidea tra gli elementi contenuti in essi.

Indichiamo i due insiemi da confrontare con $A_1 = \{x_1, \ldots, x_{n_1}\}$ e $A_2 = \{y_1, \ldots, y_{n_2}\}$, composti rispettivamente da n_1 e n_2 vettori indipendenti appartenenti allo spazio $\mathbb{R}^L$; la statistica che permette di valutare la differenza tra le distribuzioni a cui appartengono i due gruppi di elementi è quella indicata con il nome di *e-distance*, definita come:

$$e(A_1, A_2) = \frac{n_1 n_2}{n_1 + n_2}\left(\frac{2}{n_1 n_2}\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\|x_i - y_j\| + \right. \\ - \frac{1}{n_1^2}\sum_{i=1}^{n_1}\sum_{j=1}^{n_1}\|x_i - x_j\| + \\ \left. - \frac{1}{n_2^2}\sum_{i=1}^{n_2}\sum_{j=1}^{n_2}\|y_i - y_j\|\right) \tag{3.1}$$

Il simbolo $\|.\|$ indica la norma euclidea (che per un generico vettore $z = [z_0, \ldots, z_{L-1}]^T$ si ottiene come $\|z\| = \sqrt{\sum_{l=0}^{L-1} z_l^2}$), quindi la *e-distance* (equazione 3.1) può essere interpretata come una sorta di differenza tra le medie delle distanze euclidee calcolate tra i vari elementi degli insiemi. In [18] ne è stata dimostrata la non negatività.

Il criterio di decisione riguardo all'uguaglianza in distribuzione dei due insiemi consiste sempre nel confrontare la statistica calcolata $e(A_1, A_2)$ con un valore di soglia: se questo viene superato si assume che le distribuzioni siano diverse.

Il test in questione può essere utilizzato per rilevare va-

riazioni all'interno di serie temporali $\{x_i\}$, se assumiamo che l'insieme $A_1 = A_{ref}$ sia composto dai campioni di riferimento $x_i^{(ref)}$ (gli ultimi n_1 campioni non anomali osservati) e l'insieme di test $A_2 = A_{te}$ sia costituito da un solo elemento $x^{(te)}$ ($n_2 = 1$), ossia il campione osservato nell'intervallo temporale corrente, che deve essere confrontato con l'insieme di riferimento. Con queste assunzioni l'espressione 3.1 diventa:

$$\begin{aligned} e(A_{ref}, A_{te}) = & \frac{n_1}{n_1+1}\left(\frac{2}{n_1}\sum_{i=1}^{n_1}\|x_i^{(ref)} - x^{(te)}\| + \right. \\ & \left. - \frac{1}{n_1^2}\sum_{i=1}^{n_1}\sum_{j=1}^{n_1}\|x_i^{(ref)} - x_j^{(ref)}\|\right) \end{aligned} \tag{3.2}$$

Nel caso particolare in cui venga utilizzato un unico campione di riferimento $x^{(ref)}$ ($n_1 = n_2 = 1$), la *e-distance* coincide esattamente con la distanza euclidea tra i due singoli elementi confrontati:

$$e(A_{ref}, A_{te}) = \|x^{(ref)} - x^{(te)}\| \tag{3.3}$$

Una volta calcolata (indifferentemente secondo l'espressione 3.2 o la (3.3)), la quantità $e(A_{ref}, A_{te})$ viene confrontata con la soglia; se risulta maggiore di essa, il campione corrente viene ritenuto anomalo, altrimenti l'insieme di riferimento viene aggiornato con un meccanismo a finestra: l'elemento appena testato entra a far parte di A_{ref}, da cui si scarta il campione

cronologicamente più vecchio.

3.1.2 Mahalanobis distance

Un limite della *e-distance* descritta nella sezione 3.1.1 consiste nel dare lo stesso peso ad eventuali variazioni che si possono avere su componenti diverse dei vettori. Questo è dovuto all'impiego della distanza euclidea, che per come è definita tratta allo stesso modo tutte le componenti. La conseguenza è che il test della *e-distance* funziona bene quando i vettori sono costituiti da componenti dello stesso tipo, ossia hanno una variabilità confrontabile tra loro, altrimenti è possibile che non vengano rivelate variazioni anomale sulle componenti che, nel caso di comportamento normale, hanno una minore variabilità. Consideriamo ad esempio che in situazioni normali i valori di una componente possano essere anche molto distanti tra loro, mentre per un'altra componente si mantengono pressoché costanti: una variazione della stessa entità dovrebbe essere trattata in modo diverso, rivelando o meno un'anomalia, a seconda che avvenga sulla prima o sulla seconda componente. Se quindi la soglia viene fissata in modo da non far apparire anomale certe variazioni sulla componente meno variabile, non verranno scoperti scostamenti riscontrabili sull'altra componente, che in realtà sono anomali; viceversa, scegliendo un valore di soglia più piccolo, si avrebbe un incremento dei falsi positivi riguar-

danti false anomalie riscontrate quando la variazione avviene sulla prima componente.

Una soluzione al problema appena descritto è la sostituzione (nell'equazione 3.2) della distanza euclidea con la distanza di Mahalanobis, che, nel caso di componenti incorrelate tra loro, normalizza le differenze componente per componente con la rispettiva deviazione standard.

Le differenze tra la distanza euclidea e la distanza di Mahalanobis sono evidenti confrontando le espressioni 3.4 e 3.5:

$$d_E(x,y) = \|x-y\| = \sqrt{\sum_{l=0}^{L-1}(x_l - y_l)^2} \tag{3.4}$$

$$d_M(x,y) = \sqrt{\sum_{l=0}^{L-1}\frac{1}{\sigma_l^2}(x_l - y_l)^2} \tag{3.5}$$

$d_E(x,y)$ e $d_M(x,y)$ indicano rispettivamente la distanza euclidea e la distanza di Mahalanobis tra i vettori $x = [x_0, \ldots, x_{L-1}]^T$ e $y = [y_0, \ldots, y_{L-1}]^T$, ognuno dei quali è costituito da L componenti; σ_l^2 rappresenta la varianza della l-esima componente.

La normalizzazione componente per componente fa sì che una deviazione su una componente con varianza elevata ha un peso minore rispetto ad una variazione di uguale valore assoluto che avviene su una componente meno variabile.

Per completezza viene riportata di seguito l'espressione ge-

nerale della distanza di Mahalanobis, valida anche per vettori le cui componenti non sono a due a due incorrelate; C indica la matrice di covarianza dei vettori.

$$d_M(x, y) = \sqrt{(x - y)^T C^{-1} (x - y)} \tag{3.6}$$

La relazione che comunque è stata utilizzata negli algoritmi di anomaly detection implementati è la (3.5).

3.1.3 MB-GT

Un altro algoritmo di change detection che fa uso della distanza euclidea è quello presentato in [14], chiamato *Memory Based Graph Theoretic*, in breve MB-GT. Considerando la solita sequenza $\{x_i\}$ di vettori L-dimensionali, l'obiettivo è quello di scoprire se c'è stato un cambiamento all'interno della finestra temporale $\Gamma^t = \{x_{t-N+1}, \ldots, x_t\}$ comprendente gli ultimi N campioni osservati. Per fare ciò la finestra Γ^t viene divisa, più volte in modo diverso, in due sottofinestre e ogni volta viene calcolata la distanza media tra le due.

Riassumiamo in formule quanto detto; per semplicità cambiamo gli indici temporali all'interno della finestra, in modo che x_N sia l'ultimo campione osservato:

$$\Gamma^t = \{x_1, \ldots, x_N\}$$

questa viene divisa in due sottofinestre $\gamma^t_{i,j-1}$ e $\gamma^t_{j,N}$, al variare

degli indici temporali i e j, che indicano l'inizio di ognuna delle due, tra 1 e N:

$$\begin{aligned} \gamma^t_{i,j-1} &= \{x_i, \ldots, x_{j-1}\} \\ \gamma^t_{j,N} &= \{x_j, \ldots, x_N\} \qquad \textit{per } 1 \leq i < j \leq N \end{aligned}$$

per ogni possibile scelta della coppia di indici (i, j) viene calcolata quindi la quantità

$$C_{i,j} = \frac{\sum_{k=i}^{j-1} \sum_{l=j}^{N} \|x_k - x_l\|}{(j-i)(N-j+1)} \tag{3.7}$$

Infine, il parametro che quantifica la probabilità della presenza di un cambiamento di distribuzione nella finestra complessiva (la statistica da confrontare con la soglia per decidere se rivelare un'anomalia) è la cosidetta cifra di merito γ_t^{MB-GT}:

$$\begin{aligned} \gamma_t^{MB-GT} &= \max_{1 \leq i < j \leq N} C_{i,j} \\ &= \max_{1 \leq i < j \leq N} \frac{\sum_{k=i}^{j-1} \sum_{l=j}^{N} \|x_k - x_l\|}{(j-i)(N-j+1)} \end{aligned} \tag{3.8}$$

L'algoritmo descritto, così come viene presentato in [14], presuppone che nello stream di dati possa verificarsi un unico cambiamento di distribuzione definitivo (prima di esso i dati seguono una determinata distribuzione, dopo ne seguono sempre un'altra). Per un'applicazione di network anomaly

detection bisogna prescindere da questa ipotesi, in quanto le attività anomale che si vogliono individuare non modificano definitivamente il comportamento normale della rete (quello legittimo con il quale eseguire il confronto). È quindi necessario introdurre una modifica nella costruzione della finestra Γ^t: essa non sarà più costituita dagli ultimi N campioni osservati, ma i primi $N-1$ elementi nella finestra saranno gli ultimi $N-1$ campioni non anomali della sequenza monitorata e l'ultimo elemento sarà il campione osservato nel time bin corrente. Se non viene rivelato un cambiamento, nell'intervallo temporale successivo tutta la finestra scorre di un time bin, altrimenti i primi $N-1$ elementi restano fissi e viene sostituito l'ultimo.

3.1.4 Density-test

Un approccio diverso rispetto ai test visti fino ad ora è quello che caratterizza il *density test* descritto in [16]: questo algoritmo si propone infatti di valutare la presenza di cambiamenti da un insieme di dati di riferimento ad un insieme di dati di test (sempre multidimensionali di dimensione L), passando attraverso la stima della funzione di densità di probabilità propria del primo insieme.

I passi principali di tale test sono:

1. Si divide l'insieme di riferimento A_{ref} in due parti (ciascuna con lo stesso numero di elementi):

$A_{ref} = A^{(1)}_{ref} \cup A^{(2)}_{ref}$.

2. Si stima la d.d.p. dei dati di riferimento tramite il metodo di *kernel density estimator* (KDE), utilizzando $A^{(1)}_{ref}$; indichiamo la funzione ottenuta con $f^{(1)}_{ref}(\cdot)$.

3. Si calcola la statistica $\delta(A_{ref}, A_{te})$ basata sulla differenza tra il logaritmo della funzione $f^{(1)}_{ref}$ calcolata per i valori degli elementi dell'insieme di test A_{te} e il logaritmo della stessa funzione valutata nei punti appartenenti a $A^{(2)}_{ref}$.

Vediamo in dettaglio le operazioni elencate.
La densità di probabilità dei dati di riferimento viene stimata mediante KDE:

$$f^{(1)}_{ref}(s) = \sum_{x_i \in A^{(1)}_{ref}} \frac{1}{n_{ref_1}} \, G(\Sigma_{x_i}, s - x_i) \tag{3.9}$$

Nella (3.9) n_{ref_1} indica il numero di elementi presenti in $A^{(1)}_{ref}$; $G(\Sigma_{x_i}, s - x_i)$ è una kernel function gaussiana, centrata in x_i e di *bandwidth* Σ_{x_i} (matrici $L \times L$ ottenute a partire dai dati $x_i \in A^{(1)}_{ref}$ con una procedura iterativa), della forma:

$$G(\Sigma_x, s-x) = \frac{1}{(2\pi)^{L/2} \left|\det \Sigma_x\right|^{1/2}} \, e^{-\frac{1}{2}(s-x)^T \Sigma_x^{-1} (s-x)} \tag{3.10}$$

La statistica $\delta(A_{ref}, A_{te})$ si ottiene dalla differenza tra il logaritmo della funzione di verosimiglianza (LLH) di A_{te} e il

logaritmo della funzione di verosimiglianza di $A^{(2)}_{ref}$ (scalato per il rapporto tra il numero di elementi dell'insieme di test n_{te} e il numero di elementi della seconda parte dell'insieme di riferimento n_{ref_2}), calcolati considerando la densità di probabilità stimata:

$$\begin{aligned}\delta(A_{ref}, A_{te}) &= LLH(f^{(1)}_{ref}, A_{te}) - \frac{n_{te}}{n_{ref_2}} LLH(f^{(1)}_{ref}, A^{(2)}_{ref}) \\ &= \log\left(\prod_{y\in A_{te}} f^{(1)}_{ref}(y)\right) - \frac{n_{te}}{n_{ref_2}} \log\left(\prod_{z\in A^{(2)}_{ref}} f^{(1)}_{ref}(z)\right) \\ &= \sum_{y\in A_{te}} \log\left(f^{(1)}_{ref}(y)\right) - \frac{n_{te}}{n_{ref_2}} \sum_{z\in A^{(2)}_{ref}} \log\left(f^{(1)}_{ref}(z)\right)\end{aligned} \tag{3.11}$$

Dall'espressione 3.11 si osserva che la quantità δ assumerà valori piccoli negativi se l'insieme di test differisce molto in distribuzione dall'insieme di riferimento, altrimenti si manterrà nell'intorno di 0. La strategia di decisione che ne segue, impiegando una soglia positiva, è la seguente:

$$\begin{cases} se\ -\delta > soglia\ \Rightarrow\ anomalia\ rivelata \\ se\ -\delta \leq soglia\ \Rightarrow\ comportamento\ normale \end{cases} \tag{3.12}$$

3.1.5 Log-likelihood ratio

Quello descritto nella sezione 3.1.4 può essere visto come un metodo basato sul rapporto di verosimiglianza che passa attraverso la stima di densità di probabilità.

In [17] si trova un algoritmo che permette di calcolare direttamente il rapporto di verosimiglianza, senza stimare le distribuzioni dei dati. Esso considera n_{ref} campioni di riferimento e n_{te} campioni di test presi da una serie temporale $\{x_i\}$ (il test sarà applicato nel nostro caso per $n_{te} = 1$), e ha l'obiettivo di decidere tra due ipotesi:

- H_0: i campioni di riferimento e quelli di test seguono la stessa distribuzione $p_{ref}(x)$;

- H_1: i campioni di riferimento seguono la distribuzione $p_{ref}(x)$, mentre quelli di test seguono la distribuzione $p_{te}(x)$ (diversa dalla prima).

La statistica di descisione è rappresentata dal logaritmo del rapporto di verosimiglianza calcolato sui campioni di test

nelle due ipotesi, ed è indicata con S:

$$\begin{aligned} S &= \log \frac{p\big(x^{(te)}|H_1\big)}{p\big(x^{(te)}|H_0\big)} \\ &= \log \left(\prod_{i=1}^{n_{te}} \frac{p_{te}\big(x_i^{(te)}\big)}{p_{ref}\big(x_i^{(te)}\big)} \right) \\ &= \sum_{i=1}^{n_{te}} \log \left(\frac{p_{te}\big(x_i^{(te)}\big)}{p_{ref}\big(x_i^{(te)}\big)} \right) \end{aligned} \tag{3.13}$$

Al valore S si applica la solita strategia di decisione a soglia:

$$\begin{cases} se\ S > soglia \ \Rightarrow\ anomalia\ rivelata \\ se\ S \leq soglia \ \Rightarrow\ comportamento\ normale \end{cases}$$

Resta da vedere come stimare, direttamente a partire dai dati, il rapporto $w(x) = \frac{p_{te}\big(x\big)}{p_{ref}\big(x\big)}$, senza passare attraverso le due d.d.p. coinvolte. Questa operazione viene eseguita adottando un modello (illustrato nell'espressione 3.14) che impiega kernel function gaussiane (la cui forma è descritta dalla (3.15)), centrate nei campioni di test e pesate con opportuni

coefficienti α.

$$\hat{w}(x) = \sum_{i=1}^{n_{te}} \alpha_i K_\sigma\big(x, x_i^{(te)}\big) \tag{3.14}$$

$$K_\sigma(x, y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right) \tag{3.15}$$

Senza entrare nei dettagli, il parametro σ viene scelto all'interno di un insieme di possibili valori, in modo da massimizzare una particolare funzione dei campioni (di riferimento e di test).

I coefficienti $\{\alpha_i\}_{i=1}^{n_{te}}$ vengono calcolati online mediante una procedura ricorsiva che, per ogni nuovo dato osservato, ne aggiorna i valori a partire da quelli precedenti. Il calcolo iniziale del vettore $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_{n_{te}}]^T$ viene eseguito secondo le seguenti operazioni:

1. Costruire la matrice $\boldsymbol{K}$ di elementi $K_{i,l} = K_\sigma\big(x_i^{(te)}, x_l^{(te)}\big)$ (per $i = 1, \ldots, n_{te}$ e $l = 1, \ldots, n_{te}$).

2. Costruire il vettore $\boldsymbol{b}$ di elementi $b_l = \frac{1}{n_{ref}} \sum_{i=1}^{n_{ref}} K_\sigma\big(x_i^{(ref)}, x_l^{(te)}\big)$ (per $l = 1, \ldots, n_{te}$).

3. Inizializzare $\boldsymbol{\alpha}$ (con elementi $\alpha_i > 0$) e scegliere un parametro ϵ ($0 < \epsilon \ll 1$).

4. Fino ad un numero massimo di iterazioni ripetere:

(a) $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \epsilon \boldsymbol{K}^T(\mathbf{1}./\boldsymbol{K}\boldsymbol{\alpha})$
dove $\mathbf{1}$ è un vettore di elementi unitari e "./" indica la divisione elemento per elemento;

(b) $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + (1 - \boldsymbol{b}^T\boldsymbol{\alpha})\boldsymbol{b}/(\boldsymbol{b}^T\boldsymbol{b})$

(c) $\boldsymbol{\alpha} \leftarrow max(\mathbf{0}, \boldsymbol{\alpha})$

(d) $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}/(\boldsymbol{b}^T\boldsymbol{\alpha})$

Ad ogni osservazione di un nuovo campione di test $x_{n_{te}}^{(te)}$, il vettore $\boldsymbol{\alpha}$ viene aggiornato eseguendo:

1. $c = \sum_{l=1}^{n_{te}} \alpha_l K_\sigma\big(x_{n_{te}}^{(te)}, x_l^{(te-old)}\big)$, dove $x_l^{(te-old)}$ sono i campioni di test che si avevano nel time bin precedente.

2. $\alpha_l \leftarrow (1 - \eta\lambda)\alpha_{l+1} \qquad per\ l = 1, \ldots, n_{te} - 1$
$\alpha_{n_{te}} \leftarrow \eta/c$
dove η e λ sono due parametri positivi tali che $0 \leq \eta\lambda < 1$.

3. Costruire il vettore $\boldsymbol{b}$ come nel calcolo iniziale di $\boldsymbol{\alpha}$ e:

   (a) $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + (1 - \boldsymbol{b}^T\boldsymbol{\alpha})\boldsymbol{b}/(\boldsymbol{b}^T\boldsymbol{b})$

   (b) $\boldsymbol{\alpha} \leftarrow max(\mathbf{0}, \boldsymbol{\alpha})$

   (c) $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}/(\boldsymbol{b}^T\boldsymbol{\alpha})$

La stima di $w(x)$ eseguita in questo modo risulta accurata se si ha a disposizione un numero sufficiente di campioni di test, poichè le kernel function sono centrate in tali punti; se

quindi si vuole testare un elemento della serie temporale per volta (come accade per gli IDS), si devono apportare delle modifiche all'algoritmo presentato in [17]. Tali cambiamenti consistono appunto nel centrare le funzioni $K_\sigma(\cdot)$ nei valori degli elementi di riferimento (modificando di conseguenza le operazioni eseguite nel calcolo dei coefficienti $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_{n_{ref}}]^T$). Si ottiene quindi la quantità

$$\hat{w'}(x) = \sum_{i=1}^{n_{ref}} \alpha_i K_\sigma\big(x, x_i^{(ref)}\big) \tag{3.16}$$

che rappresenta ora la stima del rapporto $w'(x) = \frac{p_{ref}\big(x\big)}{p_{te}\big(x\big)}$.

La statistica S è in questo modo legata alla funzione $w'(\cdot)$ secondo la relazione:

$$\begin{aligned} S &= \sum_{i=1}^{n_{te}} \log\left(\frac{p_{te}\big(x_i^{(te)}\big)}{p_{ref}\big(x_i^{(te)}\big)}\right) \\ &= \sum_{i=1}^{n_{te}} \log\left(\frac{1}{w'\big(x_i^{(te)}\big)}\right) \\ &= -\sum_{i=1}^{n_{te}} \log\Big(w'\big(x_i^{(te)}\big)\Big) \end{aligned} \tag{3.17}$$

Sostituendo quindi $w'(\cdot)$ con la sua stima $\hat{w'}(\cdot)$ è possibile calcolare il valore da confrontare con la soglia per decidere riguardo alla rivelazione di un'eventuale anomalia.

3.1.6 CuSum multidimensionale

L'ultimo algoritmo presentato è stato ottenuto estendendo all'ambito dei dati multidimensionali il CuSum visto nella sezione 2.5. Per una migliore comprensione, di seguito sono elencate le quantità coinvolte (è stata modificata la notazione rispetto a quella impiegata nella descrizione del metodo monodimensionale, dove l'indice temporale era posto come pedice).

- Indice temporale del time bin corrente: k.

- Quantità vettoriali (di dimensione L, indifferente se vettori riga o colonna)

  - campione della serie temporale monitorata, osservato nel k-esimo time bin:

    $$\boldsymbol{y}(k) = \begin{bmatrix} y_0(k) \\ y_1(k) \\ \vdots \\ y_{L-1}(k) \end{bmatrix}$$

  - vettore media (ciascuna componente rappresenta la stima della media della componente corrispondente nella serie $\boldsymbol{y}(k)$):

$$\hat{\boldsymbol{\mu}}(k) = \begin{bmatrix} \hat{\mu}_0(k) \\ \hat{\mu}_1(k) \\ \vdots \\ \hat{\mu}_{L-1}(k) \end{bmatrix}$$

– vettore varianza (ciascuna componente rappresenta la stima della varianza della componente corrispondente nella serie $\boldsymbol{y}(k)$):

$$\hat{\boldsymbol{\sigma}}^2(k) = \begin{bmatrix} \hat{\sigma}_0^2(k) \\ \hat{\sigma}_1^2(k) \\ \vdots \\ \hat{\sigma}_{L-1}^2(k) \end{bmatrix}$$

- Quantità scalari

  – L-function: $L(\boldsymbol{y}(k))$

  – statistica CuSum: $g(k)$.

Per ogni campione osservato nella sequenza temporale le operazione eseguite sono:

1. Aggiornamento di ciascuna componente dei vettori media e varianza tramite l'algoritmo EWMA

(per $l = 0, 1, \ldots, L-1$):

$$\hat{\mu}_l(k) = \alpha \cdot \hat{\mu}_l(k-1) + (1-\alpha) \cdot y_l(k) \tag{3.18}$$

$$\hat{\sigma}_l^2(k) = \alpha \cdot \hat{\sigma}_l^2(k-1) + (1-\alpha) \cdot (y_l(k) - \hat{\mu}_l(k))^2 \tag{3.19}$$

2. Calcolo CuSum

$$\begin{aligned} L(\boldsymbol{y}(k)) =& \|\boldsymbol{y}(k) - \boldsymbol{y}(k-1)\| + \\ & - \left(\|\hat{\boldsymbol{\mu}}(k)\| + c \cdot \sqrt{\|\hat{\boldsymbol{\sigma}}^2(k)\|} \right) \end{aligned} \tag{3.20}$$

$$g(k) = \max\{0, g(k-1) + L(\boldsymbol{y}(k))\} \tag{3.21}$$

dove $\|.\|$ indica la norma (o distanza) euclidea.

La statistica da confrontare con il valore di soglia prestabilito per decidere se il campione osservato è anomalo assume quindi una forma analoga a quella che si aveva nel caso monodimensionale (come è possibile osservare comparando l'espressione 2.8 con la seguente):

$$\begin{cases} g(0) = 0 \qquad \text{(inizializzazione)} \\ g(k) = \max\Big\{ 0, g(k-1) + \|\boldsymbol{y}(k) - \boldsymbol{y}(k-1)\| + \\ \qquad - \left(\|\hat{\boldsymbol{\mu}}(k)\| + c \cdot \sqrt{\|\hat{\boldsymbol{\sigma}}^2(k)\|} \right) \Big\} \qquad \textit{per } k \geq 1 \end{cases} \tag{3.22}$$

3.2 IDS multidimensionali basati sullo sketch

Come nel caso monodimensionale, i test presentati nella sezione 3.1 possono essere applicati alla struttura dati dello sketch per implementare un IDS. Dato che adesso abbiamo a che fare con uno stream di dati vettoriali, lo sketch dovrà essere adattato a tale tipologia di valori.

La sequenza monitorata può essere ancora modellata con coppie chiave - peso $s(i) = (k(i), \boldsymbol{v}(i))$, dove ora il peso è una quantità multidimensionale (di dimensione L):
$\boldsymbol{v}(i) = [v_0(i), \ldots, v_{L-1}(i)]^T$.
Di conseguenza anche le celle dello sketch rappresentano vettori L-dimensionali $\boldsymbol{C}[d][w] = \left[C_0[d][w], \ldots, C_{L-1}[d][w]\right]^T$, in cui ciascuna componente (inizializzata a 0 all'inizio di ciascun time bin) viene incrementata con la corrispondente componente del peso $\boldsymbol{v}(i)$ osservato (nelle colonne dello sketch il cui indice corrisponde al risultato della funzione hash, associata alla riga, applicata alla chiave):

$$C_l[d][h_d(k(i))] = C_l[d][h_d(k(i))] + v_l(i) \qquad \begin{array}{l} \forall\ d = 0, \ldots, D-1 \\ \forall\ l = 0, \ldots, L-1 \end{array} \tag{3.23}$$

Costuendo lo sketch con questa modalità per ogni interval-

lo temporale di riferimento (time bin), si ottiene in ciascuna delle sue celle una sequenza multidimensionale a cui applicare (separatamente per ognuna di esse) uno degli algoritmi di anomaly detection visti in precedenza.

Capitolo 4

Risultati sperimentali

In quest'ultimo capitolo vengono presentati i risultati ottenuti testando gli IDS basati sugli algoritmi illustrati nel capitolo 3. Le prove sperimentali sono state realizzate impiegando come dati di input alcune delle tracce di traffico reperibili nell'archivio MAWILab; oltre alla descrizione di tale dataset, vedremo qualche dettaglio in più sugli algoritmi implementati e infine verranno discusse le prestazioni ottenute.

4.1 Dataset impiegato: le tracce MAWI

Il dataset utilizzato per eseguire i test utili per valutare le prestazioni degli algoritmi di anomaly detection è costituito dalle tracce di traffico che si trovano nell'archivio MAWILab (acronimo che sta per Measurement and Analysis of the Wide Internet), pubblicamente disponibile on-line in [1]. Ciascuna traccia è relativa al traffico osservato per 15 minuti (dalle 14:00 alle 14:15, ora del Giappone) in uno specifico giorno su un link transoceanico che collega Giappone e Stati Uniti; tale collegamento aveva originariamente una banda di 18 Mbps, per passare a 100 Mbps a luglio 2006 e infine a 150 Mbps a giugno 2007.

I dati presenti nell'archivio vengono aggiornati quotidianamente, a partire dall'anno 2001 (ad oggi sono disponibili tracce di traffico catalogate fino ad aprile 2011), e sono registrati in file di tipo pcap che contengono informazioni relative ai pacchetti transitati sul link monitorato (principalmente si riferiscono agli header del livello IP e del livello di trasporto). Prima di essere impiegate come ingresso dei metodi di anomaly detection implementati, tali tracce devono essere convertite per mezzo di tre tool open source (Softflowd disponibile in [6], Nfdump e Nfcapd disponibili in [3]), per ottenere file con il formato desiderato (per maggiori dettagli è possibile consultare l'appendice A di questo elaborato).

La caratteristica importante delle tracce MAWI è che cia-

scun flusso è stato catalogato in funzione della probabilità di rappresentare un'anomalia. Forniscono quindi un aiuto per valutare le prestazioni (in termini di falsi positivi e falsi negativi) di un qualsiasi algoritmo di anomaly detection, attraverso un confronto tra le anomalie rivelate da quest'ultimo e quelle registrate nella traccia; bisogna comunque tenere presente la natura probabilistica della classificazione: l'elenco delle anomalie indicate nell'archivio è infatti il risultato dell'impiego congiunto di quattro detector (basati rispettivamente su trasformata di Hough, distribuzione Gamma, Kullback-Leibler divergence e Principal Component Analysis (PCA)), quindi non abbiamo la certezza che siano realmente causate da un attacco. In particolare, dall'analisi eseguita dagli autori del progetto MAWI, il traffico risulta suddiviso in quattro raggruppamenti [12]:

- *anomalous*: traffico quasi certamente anomalo;
- *suspicious*: traffico che molto probabilmente è anomalo, ma non è stato chiaramente identificato come tale con i metodi impiegati;
- *notice*: traffico non anomalo (non deve essere considerato come un'attacco) che tuttavia è stato segnalato da almeno uno dei quattro detector (per questo non è stato classificato come *benign*);
- *benign*: traffico normale, non rivelato da nessuno dei detector.

Le anomalie registrate sono state a loro volta suddivise in tre categorie:

- *attack*: anomalie riconducibili ad attachi noti;
- *special*: anomalie associate ad attività che coinvolgono porte well-known;
- *unknown*: anomalie di tipo sconosciuto, che non rientrano nelle due categorie precedenti.

Il traffico etichettato come *anomalous*, *suspicious* o *notice* è elencato in file di tipo xml (due per ogni traccia, uno per le anomalie di tipo *anomalous* e *suspicious*, uno per quelle di tipo *notice*), dove per ogni anomalia individuata vengono riportati [1]:

- T: tipo di anomalia (*anomalous*, *suspicious* o *notice*).
- D_r, D_a: distanze da due punti di riferimento rispettivamente per il traffico regolare e anomalo, calcolate elaborando con un algoritmo opportuno (SCANN) le uscite dei quattro detector; sulla base di tali valori viene deciso il tipo di anomalia: *anomalous* se $D_a \leq D_r$, *suspicious* se $D_a > D_r$ e $D_a/D_r - 1 \leq 0.5$, *notice* se $D_a/D_r - 1 > 0.5$.
- C: categoria dell'anomalia, rappresentata da un numero (*attack* per i valori $1 \leq C \leq 500$, *special* per $500 < C \leq 900$ e *unknown* per $C > 900$).

- $\boldsymbol{V}$: vettore costituito da 12 elementi binari i cui valori indicano quali detector hanno rivelato l'anomalia; i quattro detector sono stati infatti applicati ciascuno con tre modalità (in ordine di sensibilità crescente: conservative, optimal, sensitive); ogni elemento del vettore $\boldsymbol{V}$ è associato ad un detector con una determinata modalità, nel seguente ordine: Hough (sensitive, optimal, conservative), Gamma (sensitive, optimal, conservative), KL (sensitive, optimal, conservative), PCA (sensitive, optimal, conservative).

- Filtri che identificano l'anomalia tramite feature del traffico, che possono essere: indirizzo IP sorgente, indirizzo IP destinatario, porta sorgente, porta destinataria, protocollo di trasporto.

- Timestamp che individuano inizio e fine del traffico anomalo.

Un riassunto della classificazione delle anomalie è presente nella tabella 4.1, dove sono indicate la tipologia (*label*), la categoria più specifica dell'attacco con il corrispondente valore del campo C presente nei file xml e una breve descrizione [12] [1].

Tabella 4.1: Classificazione delle anomalie nelle tracce MAWI

Label	Categoria (C)	Dettagli
Attack	Sasser (1)	Traffico TCP sulle porte 1023, 5554 o 9898
Attack	NetBIOS (2)	Traffico sulla porta TCP 139 o sulla porta UDP 137
Attack	RPC (3)	Traffico TCP sulla porta 135
Attack	SMB (4)	Traffico TCP sulla porta 445
Attack	Ping flooding (20)	Traffico ICMP elevato
Attack	Port scan e altri attacchi (10-SYN, 11-RST, 12-FIN, 51-FTP, 52-SSH, 53-HTTP, 54-HTTPS)	Traffico con più di 7 pacchetti in cui almeno la metà hanno i flag SYN, RST o FIN settati, oppure traffico http, ftp, ssh, dns avente il flag SYN settato per almeno il 30% dei pacchetti
Special	Http (503-HTTP, 504-HTTPS)	Traffico TCP sulle porte 80 o 8080 con meno del 30% dei flag SYN settati

Tabella 4.1: continua nella prossima pagina

Tabella 4.1: continua dalla pagina precedente

Label	Categoria (C)	Dettagli
Special	ftp, ssh, dns (501-FTP, 502-SSH)	Traffico TCP sulle porte 20, 21 o 22 oppure TCP & UDP sulla porta 53 con meno del 30% dei flag SYN settati
Unknown	Unknown (901)	Traffico che non corrisponde ai precedenti casi

Tabella 4.1: Classificazione delle anomalie nelle tracce MAWI

4.2 Dettagli implementativi degli algoritmi

In questa sezione vediamo nel dettaglio come gli algoritmi di anomaly detection sono stati sviluppati per eseguire l'analisi delle informazioni presenti nelle tracce di traffico del dataset considerato.

4.2.1 Aspetti comuni

Innanzitutto è necessario capire come le tracce, ciascuna relativa a 15 minuti di traffico in uno specifico giorno, vengono trattate dagli IDS implementati: ognuna di esse contiene i dati che vanno in ingresso all'algoritmo in un determinato time bin; considerandole una di seguito all'altra, si ottiene quindi la serie temporale da analizzare.

Altro aspetto importante è vedere quali sono le metriche scelte che vanno a costituire il vettore dei dati analizzati in ogni intervallo temporale. Prendendo in considerazione le anomalie che possono essere riscontrate in una generica traccia (quelle elencate nella tabella 4.1 e altri attacchi, come ad esempio quelli di tipo UDP flooding), i descrittori di traffico considerati sono:

1. Numero di byte.
2. Numero di porte destinatarie TCP o UDP diverse utilizzate.
3. Numero di indirizzi IP sorgente diversi.
4. Numero di pacchetti TCP con flag SYN settato.
5. Numero di pacchetti TCP con flag RST settato.
6. Numero di pacchetti TCP con flag FIN settato.
7. Numero di pacchetti ICMP.
8. Numero di pacchetti UDP.

Le metriche appena elencate vengono impiegate per costruire il vettore di incremento $\boldsymbol{v}_i$ con il quale, per ogni entry presente nella traccia di ingresso (ciascuna riga di tale file corrisponde al traffico osservato per un determinato flusso, identificato dalla quintupla src IP, dst IP, src port, dst port, protocol), vengono aggiornati i valori dello sketch. Per tutti gli algoritmi implementati (quelli presentati nel capitolo 3, eccetto quello basato sulla distanza di Mahalanobis), prima di eseguire l'operazione di incremento delle celle dello sketch descritta dall'equazione 3.23, i valori di tali metriche vengono normalizzati per tenere in considerazione la natura diversa di ognuno (quindi gli intervalli in cui possono variare); le costanti

di normalizzazione utilizzate sono distinte per ciascun elemento del vettore e si ottengono dagli sketch relativi ai time bin di riferimento (quelli costruiti, inizialmente senza normalizzazione e poi normalizzati, analizzando le tracce di training ripulite dalle anomalie), calcolandone la media dei valori delle celle oppure il massimo (a seconda dell'algoritmo considerato). Tale operazione non viene invece eseguita nel test che fa uso della distanza di Mahalanobis, poiché è già presente una normalizzazione che impiega le varianze delle componenti del vettore (come illustrato nell'equazione 3.5); in questo caso le varianze vengono stimate a partire dai valori degli elementi degli sketch di rifermento.
Come descritto nella sezione 3.2, per la costruzione dello sketch è necessaria una chiave k: questa è identificata dall'indirizzo IP destinatario, in modo da poter individuare attacchi rivolti a particolari destinazioni.

La struttura generale degli IDS realizzati risulta quindi essere la seguente:

1. Fase di training: costruzione di n_{ref} (parametro che può essere scelto) sketch di riferimento, leggendo le tracce iniziali che sono state ripulite dalle anomalie (grazie ai filtri presenti nei file xml dell'archivio MAWI).

2. Fase di test: analisi in successione delle tracce che seguono quelle di riferimento; per ognuna di esse (ciascuna rappresentante un time bin), vengono eseguiti

(a) costruzione dello sketch di test;

(b) calcolo della statistica di decisione (in base al particolare algoritmo) per ciascuna cella dello sketch e confronto con un valore di soglia;

(c) decisione complessiva riguardo al time bin;

(d) identificazione delle chiavi responsabili dell'anomalia (se rilevata).

Restano quindi da vedere con quale criterio l'intervallo temporale analizzato viene dichiarato anomalo o meno e come vengono individuate le chiavi coinvolte in una eventuale anomalia. Dopo il calcolo delle statistiche di decisione eseguito per ogni elemento dello sketch, in una tabella analoga (con le stesse dimensioni $D \times W$) si memorizzano quali di esse hanno superato la soglia; il time bin viene considerato anomalo se in tutte le righe si è verificato almeno uno di questi allarmi "locali". Avendo registrato, in fase di costruzione dello sketch, i valori delle chiavi che sono stati mappati (secondo le fuzioni hash) nelle diverse celle, è a questo punto possibile identificare quelle che hanno provocato un'anomalia in tutte le righe dello sketch: sono proprio queste le chiavi che vengono ritenute responsabili; i loro valori sono quindi registrati su opportuni file di uscita (separatamente per ciascun time bin).

4.2.2 Implementazione del CuSum

Particolare attenzione deve essere dedicata agli IDS basati sul CuSum, sia nella versione monodimensionale che in quella multidimensionale. Per entrambe infatti sono state apportate alcune modifiche rispetto a quanto visto nella descrizione degli algoritmi nei capitoli 2 e 3, così da adattarli nel migliore dei modi alla struttura del sistema realizzato.

Per prima cosa si osserva che nelle espressioni 2.8 e 3.22 che descrivono i due algoritmi CuSum compaiono i campioni osservati nel time bin precedente, che adesso indichiamo con $\boldsymbol{y}(k-1)$; poiché l'obiettivo è quello di individuare gli scostamenti dal comportamento legittimo modellato con i dati acquisiti nella fase di training, la quantità $\boldsymbol{y}(k-1)$ viene sostituita con $\boldsymbol{y}_{ref}(k-1)$, che rappresenta l'ultimo dei campioni precedenti non anomali; ciò significa che se il campione osservato nel time bin k-esimo è anomalo, nel time bin successivo non verrà modificato il valore di $\boldsymbol{y}_{ref}(k-1)$.

Per lo stesso motivo citato in precedenza, anche i valori di media e varianza ($\hat{\boldsymbol{\mu}}(k)$ e $\hat{\boldsymbol{\sigma}}^2(k)$) devono tener conto solo dei dati non anomali osservati fino all'istante corrente, quindi prima viene calcolata la statistica di decisione e solo se questa non supera la soglia si procede all'aggiornamento delle due stime secondo le espressioni 3.18 e 3.19 (nel caso monodimensionale sono le (2.6) e (2.7)).

Inoltre, affinché un'anomalia che si verifica in un time bin

non influenzi l'analisi nei successivi intervalli temporali, se la statistica di decisione $g(k)$ supera la soglia, viene reinizializzata a 0 (una volta terminate le operazioni da eseguire nel time bin corrente); nel time bin seguente si avrà quindi $g(k-1) = 0$.

Un ulteriore dettaglio è rappresentato dai valori con cui vengono inizializzate le varie quantità nel primo intervallo temporale (il time bin 0, corrispondente alla prima traccia di riferimento):

- statistica di decisione: $g(0) = 0$;
- stima della media: $\hat{\boldsymbol{\mu}}(0) = \boldsymbol{y}(0)$;
- stima della varianza: $\hat{\boldsymbol{\sigma}}^2(0) = \mathbf{0}$.

Si ricorda infine che tutte le operazione descritte vengono eseguite separatamente per ogni cella dello sketch; nel time bin k-esimo si hanno quindi $D \times W$ campioni $\boldsymbol{y}[d][w](k)$ (i valori delle celle dello sketch), stime di media e varianza $\hat{\boldsymbol{\mu}}[d][w](k)$ e $\hat{\boldsymbol{\sigma}}^2[d][w](k)$, statistiche di decisione $g[d][w](k)$. Nella descrizione precedente gli indici d, w sono stati omessi per semplicità di notazione.

Quanto detto in questa sezione vale sia per il CuSum multidimensionale, sia per quello monodimensionale; quest'ultimo è stato implementato per avere dei risultati di riferimento con cui confrontare quelli ottenuti con gli algoritmi multidimensionali, quindi è stato eseguito separatamente per tutte le otto

metriche elencate nella sezione 4.2.1 (impiegando in tutti i casi l'indirizzo IP destinatario come chiave). Da ognuno degli otto test si ottiene, per ciascun time bin, un file con l'elenco delle chiavi responsabili delle anomalie rilevate; per eseguire quindi un confronto con gli IDS multidimensionali è stato quindi necessario unirli in un unico file contenente le chiavi responsabili delle anomalie per almeno una delle metriche.

4.3 Prestazioni ottenute

Le prestazioni degli IDS testati sono state calcolate, per ogni traccia di traffico analizzata, eseguendo un confronto tra i flussi identificati dalle chiavi ritenute responsabili delle anomalie, elencate nei file di uscita dell'algoritmo di detection, e quelli associati alle anomalie realmente presenti nella traccia, identificati dai filtri presenti nei file xml dell'archivio MAWI relativi al traffico anormale.

In particolare sono stati considerati come falsi positivi i flussi che nell'archivio MAWI non sono etichettati né come "anomalous" né come "suspicious", ma che vengono indicati come anomali dall'algoritmo di detection; vengono ritenuti falsi negativi i flussi con etichetta "anomalous" non rivelati dal nostro IDS (non c'è infatti certezza che quelli con etichetta "suspicious" non siano riconducibili ad attività normali). Il rate di falsi positivi (o probabilità di falso allarme, indicata

con P_{FA}) viene calcolato dividendo il numero di falsi positivi per il numero di flussi che non sono né "anomalous" né "suspicious"; il rate di falsi negativi P_{FN} si ottiene dal rapporto tra il numero di falsi negativi e il numero di flussi realmente "anomalous".

Il metodo più comune per visualizzare le prestazioni di un qualsiasi sistema di detection è quello di graficare la cosiddetta curva ROC (Receiver Operating Characteristics), che rappresenta la probabilità di corretta rivelazione (o detection, calcolata come $P_D = 1 - P_{FN}$), in funzione della probabilità di falso allarme (quantità entrambe comprese tra 0 e 1). Il sistema in esame è quindi tanto migliore quanto più tale curva è caratterizzata da elevati valori di P_D (vicini a 1) corrispondenti a valori piccoli di P_{FA} (vicini a 0).

4.3.1 Algoritmi a confronto

Con la modalità appena descritta sono quindi state ottenute le curve ROC riportate nelle figure seguenti (dalla 4.1 alla 4.8), che mettono a confronto le prestazioni dei vari algoritmi. Questi sono stati eseguiti impiegando $n_{ref} = 4$ tracce di riferimento, tranne che per il CuSum monodimensionale e multidimensionale, per i quali tale parametro è stato settato a 1 (valori diversi non hanno comunque portato a cambiamenti significativi); inoltre la curva relativa alle prestazioni dell'algoritmo *e-distance* viene riportata solo per la traccia del 5 gennaio

2007, in quanto per le altre ha mostrato risultati praticamente identici al test basato sulla distanza di Mahalanobis (i due algoritmi, una volta eseguita la normalizzazione, sono infatti molto simili). Le tracce MAWI utilizzate, comprese quelle di riferimento, sono quelle relative ai giorni che vanno dall'1 all'11 gennaio 2007.

La figura 4.1 riporta le curve ROC ottenute dai diversi sistemi per la traccia del 5 gennaio 2007; si osserva che in questo caso specifico gli algoritmi del *density-test* e del *log-likelihood ratio* hanno prestazioni leggermente migliori degli altri, ma in ogni caso non si hanno valori della probabilità di corretta rivelazione maggiori di 0.6 in corrispondenza di rate di falsi positivi minori di 0.2.

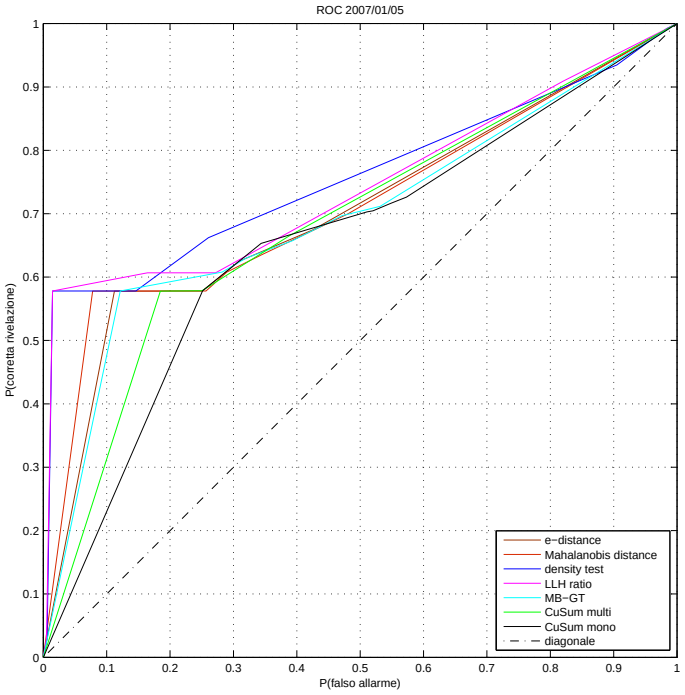

Figura 4.1: Confronto ROC relative alla traccia del 2007/01/05

La traccia del 6 gennaio è quella per la quale si osservano i risultati migliori (figura 4.2): tutti i sistemi testati presentano una probabiltà di corretta rivelazione di poco superiore a 0.7 relativa a probabilità di falso allarme di circa 0.2.

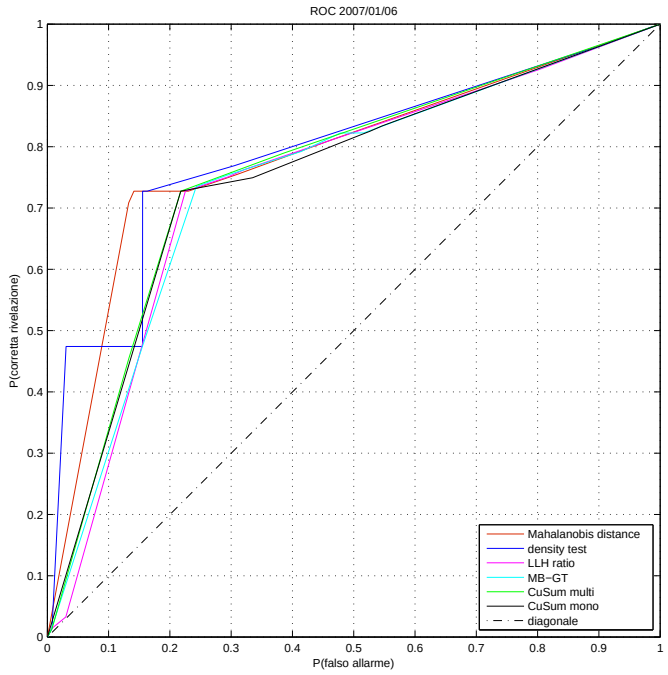

Figura 4.2: Confronto ROC relative alla traccia del 2007/01/06

Nella figure 4.3 e 4.6 osserviamo che per le tracce del 7 e del 10 gennaio l'algoritmo del *log-likelihood ratio* si comporta molto peggio degli altri, ottenendo più falsi positivi rispetto alle corrette rivelazioni.

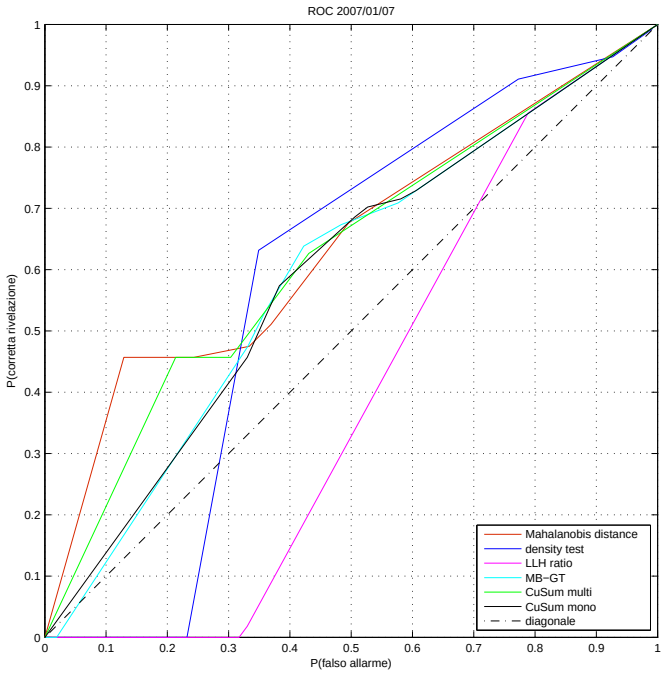

Figura 4.3: Confronto ROC relative alla traccia del 2007/01/07

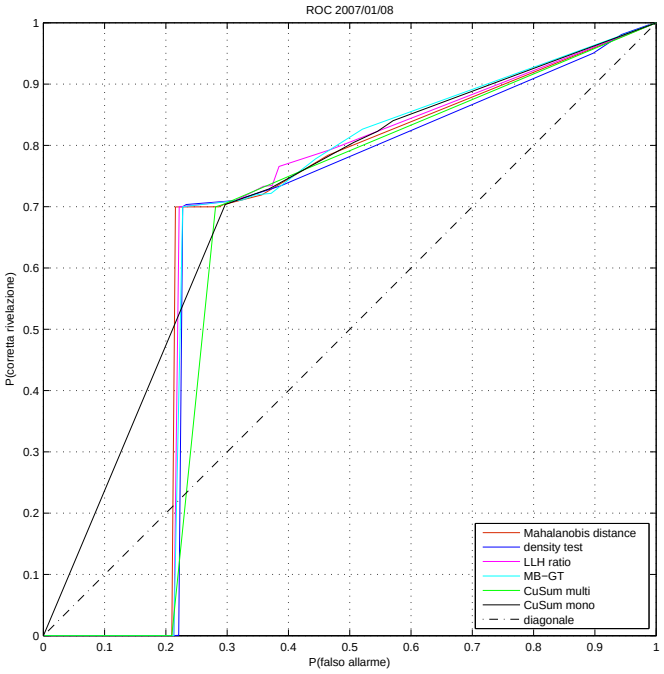

Figura 4.4: Confronto ROC relative alla traccia del 2007/01/08

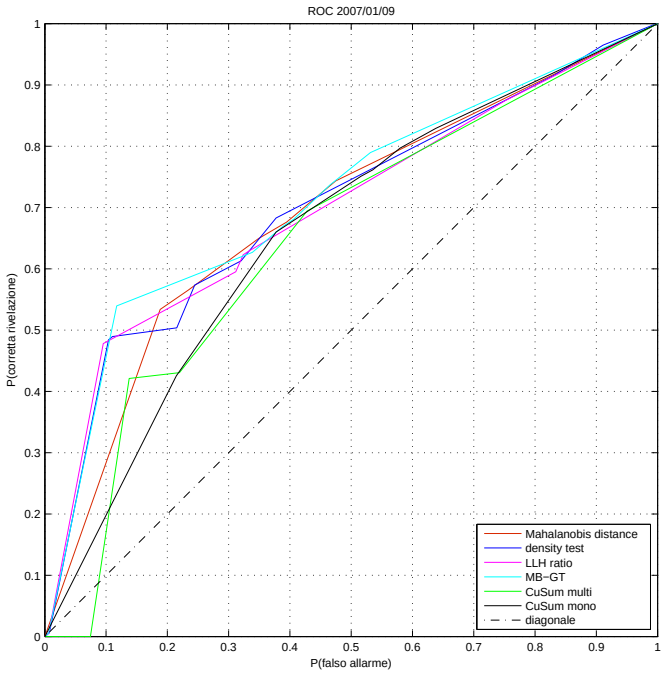

Figura 4.5: Confronto ROC relative alla traccia del 2007/01/09

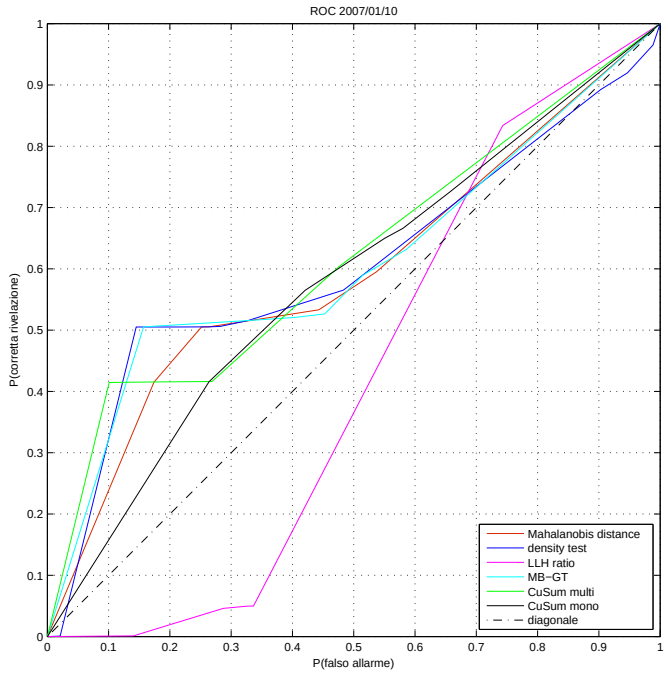

Figura 4.6: Confronto ROC relative alla traccia del 2007/01/10

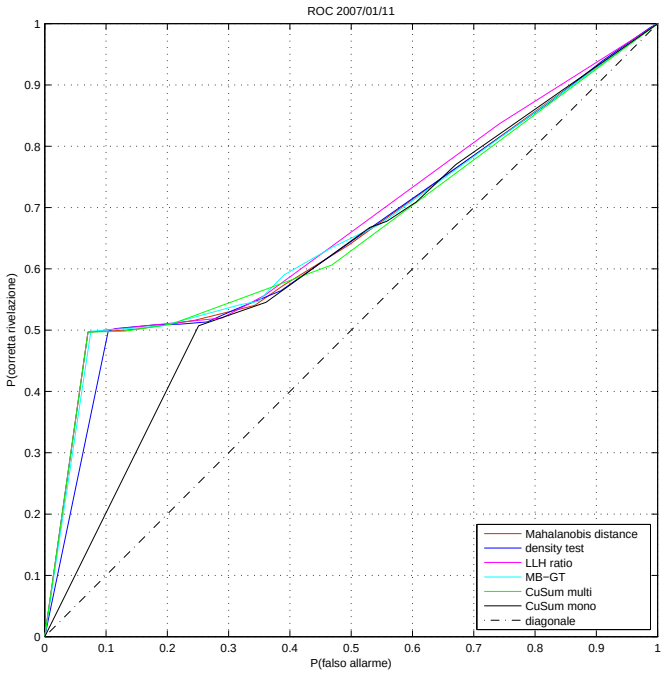

Figura 4.7: Confronto ROC relative alla traccia del 2007/01/11

In figura 4.8 sono riportate le curve ROC degli IDS testati, calcolate complessivamente su tutte le tracce che vanno dal 5 all'11 gennaio 2007.

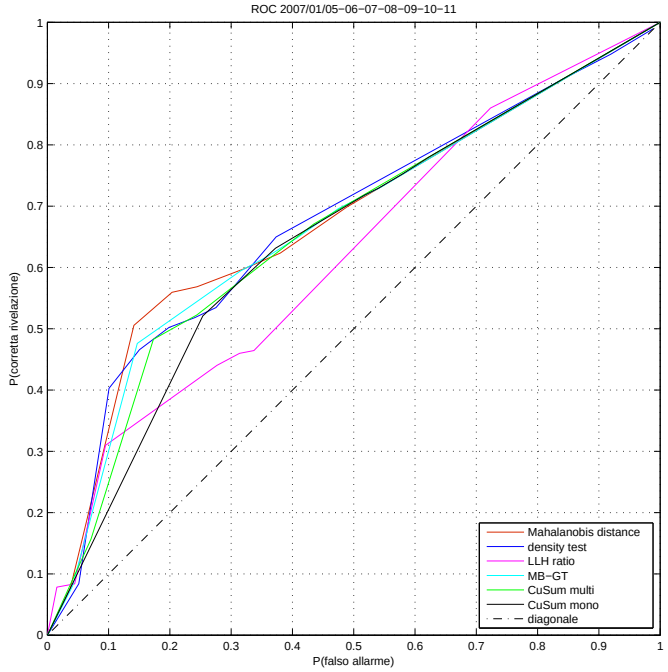

Figura 4.8: Confronto ROC complessive relative alle tracce dal 2007/01/05 al 2007/01/11

Dai grafici precedenti su può osservare che gli algoritmi multidimensionali implementati hanno prestazioni simili tra loro, ad eccezione dell'IDS basato sulla stima diretta del LLH-ratio (curva ROC complessiva di colore rosa nella figura 4.8), che si comporta leggermente peggio degli altri. Purtroppo i risultati visti fino ad ora non si possono considerare soddisfacenti: si ha infatti una probabilità di corretta rivelazione troppo bassa (inferiore a 0.6 nelle ROC complessive in figura 4.8, e nel migliore dei casi poco sopra a 0.7 per la traccia del 6 gennaio in fig. 4.2) per i valori della probabilità di falso allarme di intresse (quelli minori di 0.2). Il problema non è comunque causato dall'approccio multidimensionale dei vari IDS, come dimostrato dal fatto che si ottengono risultati simili anche dall'esecuzione del CuSum monodimensionale sulle singole metriche (le curve ROC nere nelle figure di questa sezione).

4.3.2 Altri metodi di calcolo delle prestazioni

Dato che, come abbiamo detto, non sono stati ottenuti buoni risultati con le modalità viste in precedenza, le prestazioni dei vari IDS sono state rivalutate considerando diversamente le anomalie. Ricordiamo che il metodo di calcolo delle prestazioni originario assume come falsi positivi i flussi che non sono etichettati né come "anomalous" né come "suspicious", ma che vengono indicati come anomali dall'algoritmo di de-

tection; invece considera falsi negativi i flussi con etichetta "anomalous" non rivelati. Nei file dell'archivio MAWI contenti l'elenco delle attività anormali presenti nelle tracce sono presenti comunque ulteriori informazioni (come descritto nella sezione 4.1), che permettono di eseguire un'analisi in base sia alla categoria dell'anomalia (*attack*, *special* o *unknown*), sia a quanto evidente è tale alterazione (a seconda di quanti dei quattro detector utilizzati dal progetto MAWI sono stati in grado di individuarla). Sono quindi stati sviluppati nuovi metodi di valutazione delle prestazioni che lasciano invariato il calcolo dei falsi positivi, modificando invece il conteggio dei falsi negativi, così come illustrato nella descrizione seguente.

1. "fn 2 detector": considera falsi negativi le mancate rivelazioni di flussi etichettati come *anomalous* individuati da almeno due dei quattro detector usati dal progetto MAWI.

2. "fn 3 detector": considera falsi negativi le mancate rivelazioni di flussi etichettati come *anomalous* individuati da almeno tre dei quattro detector usati dal progetto MAWI; come si nota dalla fig. 4.9, i risultati ottenuti con questo e il precedente metodo non si discostano molto da quelli che si hanno con il metodo originario.

3. "fn 4 detector": considera falsi negativi le mancate rivelazioni di flussi etichettati come *anomalous* individuati

da tutti i quattro detector usati dal progetto MAWI; in questo caso le prestazioni migliorano rispetto ai precedenti.

4. "fn attack": considera falsi negativi le mancate rivelazioni di flussi etichettati come *anomalous* appartenenti alla categoria *attack*, ossia quella degli attacchi noti.

5. "fn attack special": considera falsi negativi le mancate rivelazioni di flussi etichettati come *anomalous* appartenenti alla categoria *attack* o alla categoria *special* (attacchi che utilizzano porte well-known); così come considerando la sola categoria *attack*, anche in questo caso i risultati ottenuti sono molto scadenti, probabilmente a causa del numero di flussi appartenenti alle due categorie considerate, relativamente piccolo rispetto a quello delle anomalie totali.

6. "fn unknown": considera falsi negativi le mancate rivelazioni di flussi etichettati come *anomalous* appartenenti alla categoria *unknown* (attività anomale sconosciute); a differenza dei due casi precedenti, questo metodo presenta risultati migliori rispetto a quello originario.

7. "fn unknown 4 detector": considera falsi negativi le mancate rivelazioni di flussi etichettati come *anomalous* appartenenti alla categoria *unknown*, individuati da tutti i

quattro detector usati dal progetto MAWI; come ci si può aspettare si ha un miglioramento rispetto ai due metodi "fn unknown" e "fn 4 detector".

8. "fn 4 conservative": considera falsi negativi le mancate rivelazioni di flussi etichettati come *anomalous* individuati da tutti i quattro detector impiegati nel progetto MAWI in modalità *conservative* (si ricorda che nella ricerca degli attacchi ognuno di essi è stato usato con tre gradi di sensibilità crescente: *conservative*, *optimal*, *sensitive*); questo approccio permette di ottenere buoni risultati per la maggior parte delle tracce analizzate (fig. 4.9), ma in alcuni casi si dimostra inefficace (come nell'analisi della traccia del giorno 11, da cui è stata ottenuta la curva ROC mostrata in figura 4.10), probabilmente perché troppo conservativo.

Le considerazioni fatte per ciascuno di questi metodi sono valide per tutti gli IDS testati; la figura 4.9 conferma quanto affermato nel caso del CuSum multidimensionale, riportando le curve ROC che considerano complessivamente le tracce relative ai giorni dal 2 all'11 gennaio 2007. Le ROC degli altri algoritmi sono raffigurate nelle figure 4.11, 4.12, 4.13 e 4.14.

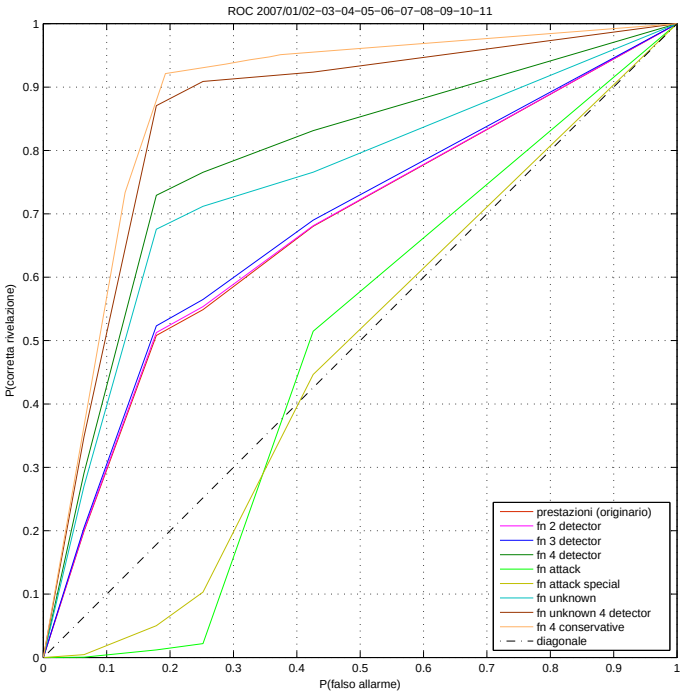

Figura 4.9: Confronto dei metodi di calcolo delle prestazioni; ROC CuSum multidimensionale

La figura 4.9 evidenzia un netto miglioramento delle prestazioni del CuSum multidimensionale nei casi in cui vengono considerate nei falsi negativi le mancate rivelazioni delle anomalie di tipo sconosciuto oppure quelle individuate da tutti i

quattro detector impiegati nel progetto MAWI: in corrispondenza di probabilità di falso allarme appena inferiori a 0.2, si passa infatti da avere un tasso di corretta rivelazione di circa 0.5 ottenuto con il metodo originario di calcolo delle prestazioni, ad un valore prossimo a 0.7; inoltre se mettiamo insieme i due metodi, considerando quindi solo i flussi anomali di tipo sconosciuto individuati da tutti i quattro detector, si raggiungono valori superiori a 0.85. Le due curve che mostrano i risultati peggiori sono quelle relative agli attacchi noti e a quelli che coinvolgono porte well-known, mentre considerando le anomalie individuate da almeno due o almeno tre dei quattro detector non si osservano variazioni significative rispetto al metodo originario.

Dalla figura 4.9 si osservano prestazioni migliori anche con il metodo "fn 4 conservative"; tuttavia, come dimostra la figura 4.10, per la traccia del giorno 11 otteniamo risultati inaccettabili, quindi tale metodo non viene preso in considerazione per il calcolo delle prestazioni degli altri IDS.

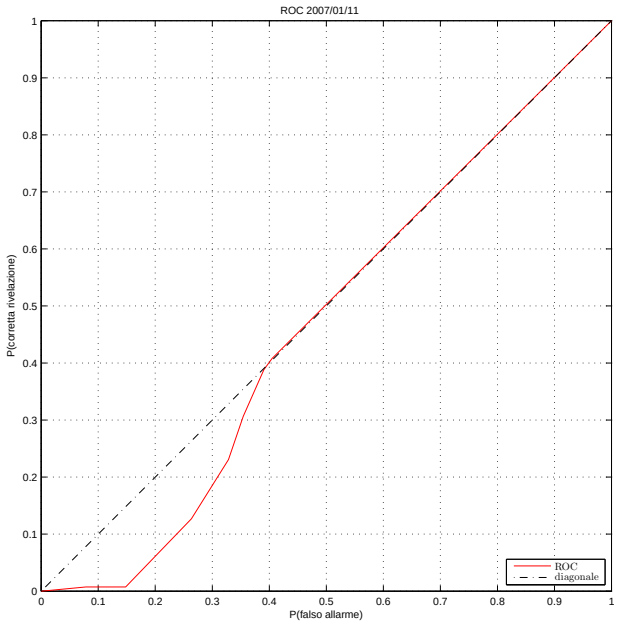

Figura 4.10: Metodo "fn 4 conservative"; traccia 2007/01/11; ROC CuSum multidimensionale

Le figure 4.11, 4.12, 4.13 e 4.14 mettono a confronto le prestazioni ottenute con i vari metodi di calcolo per i restanti IDS; anche per questi osserviamo gli stessi miglioramenti visti per il CuSum multidimensionale nel caso di attacchi sconosciuti e l'inefficacia per gli attacchi noti.

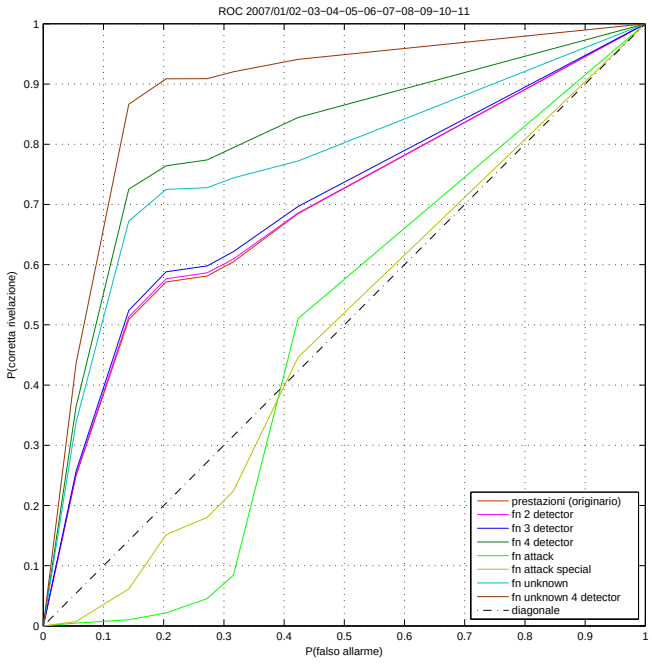

Figura 4.11: Confronto dei metodi di calcolo delle prestazioni; ROC Mahalanobis distance

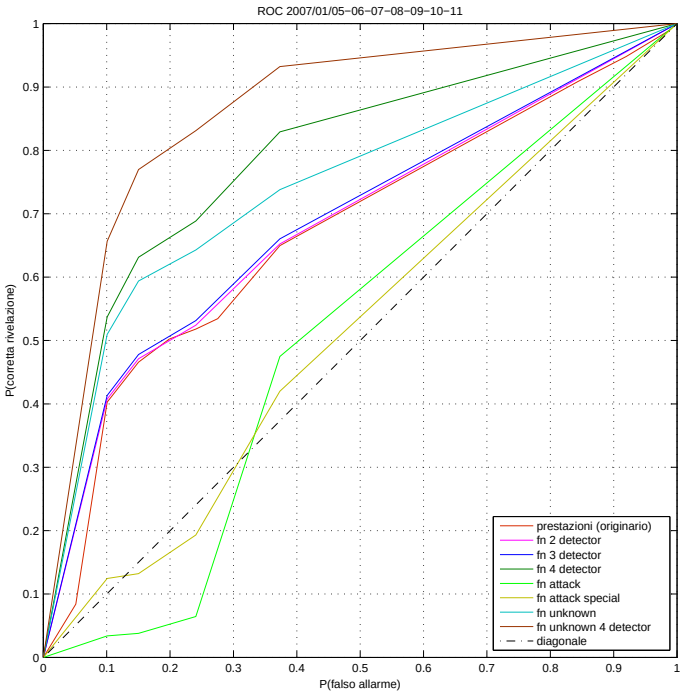

Figura 4.12: Confronto dei metodi di calcolo delle prestazioni; ROC density test

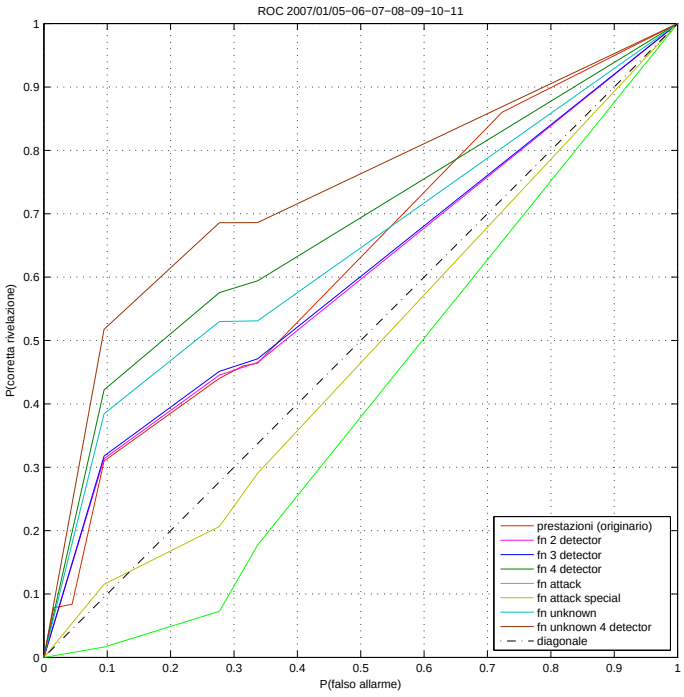

Figura 4.13: Confronto dei metodi di calcolo delle prestazioni; ROC LLH ratio

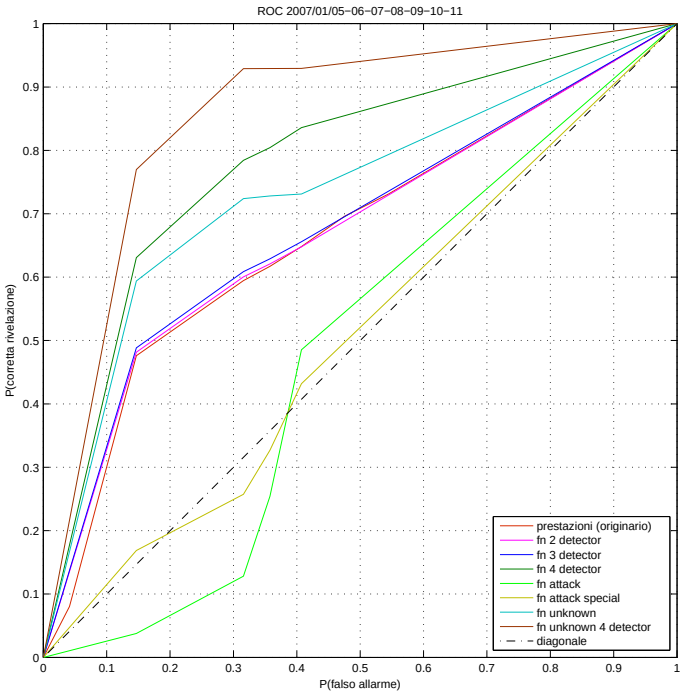

Figura 4.14: Confronto dei metodi di calcolo delle prestazioni; ROC MB-GT

Con riferimento alla figura 4.9, si osserva che il metodo di detection basato sul CuSum multidimensionale e, come già detto, anche gli altri IDS multidimensionali, non si dimostrano in grado di rivelare correttamente gli attacchi noti e quelli che coninvolgono porte well-known (appartenenti rispettivamente alle categorie *attack* e *special*), presenti nelle tracce analizzate; tuttavia individuano meglio le anomalie di tipo sconosciuto (quelle della categoria *unknown*). Come abbiamo visto nel capitolo 1, gli IDS più efficienti per la rilevazione di attachi noti sono quelli di tipo misuse based, che però non sono in grado di individuare attività anomale nuove; i sistemi testati in questo elaborato (che ricordiamo essere di tipo anomaly based) possono quindi essere considerati in una prospettiva di impiego congiunto con IDS misuse based già esistenti, al fine di individuare intrusioni sia conosciute che ignote.

Un ulteriore aspetto incoraggiante è rappresentato dal fatto che le prestazioni sono migliori se consideriamo le anomalie individuate da tutti i quattro detector utilizzati dal progetto MAWI, ossia le attività fortemente anomale. Le anomalie che portano ad un peggioramento sono quindi quelle che non sono state individate da almeno uno dei quattro detector: ciò significa che, probabilmente, se venissero uniti i risultati di un certo numero degli IDS sviluppati, si potrebbero migliorare le prestazioni complessive.

4.3.3 Confronto tra CuSum monodimensionale e CuSum multidimensionale

In questa sezione viene illustrato un confronto più dettagliato tra i risultati ottenuti dalle due versioni del CuSum considerate. I grafici seguenti riportano le curve ROC del CuSum multidimensionale, del CuSum mondimensionale eseguito separatamente su ciascuna metrica e del CuSum monodimensionale i cui risultati riferiti ai singoli descrittori di traffico sono stati uniti; i test con cui sono state ottenute sono stati eseguiti impiegando 1 traccia di riferimento ($n_{ref} = 1$) e settando i parametri propri degli algoritmi (vedi le espressioni 2.6, 2.8, 3.18 e 3.22) con i seguenti valori: $\alpha = 0.9$, $c = 0.5$. Le prestazioni sono state calcolate complessivamente per le tracce dei giorni dal 5 all'11 gennaio, con il metodo originario (quello descritto all'inizio della sezione 4.3) e con quelli che hanno dimostrato un miglioramento rispetto ad esso ("fn 4 detector", "fn unknown" e "fn unknown 4 detector").

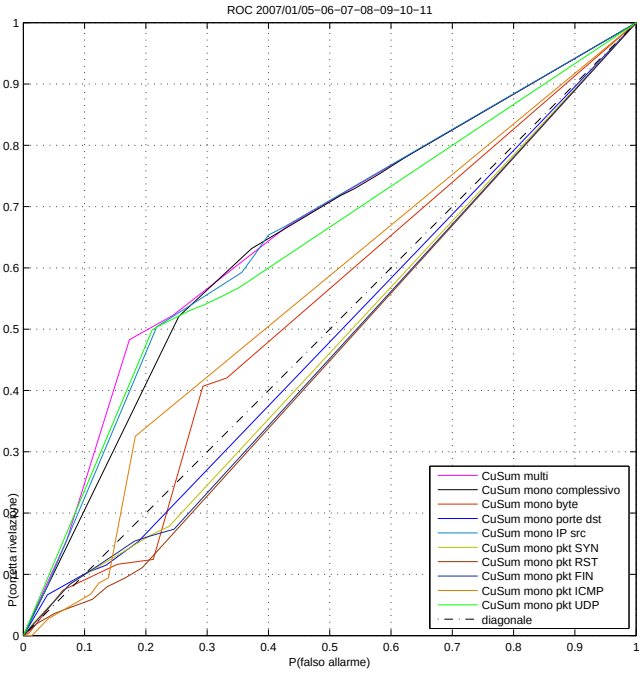

Figura 4.15: Confronto CuSum multi/mono-dimensionale; metodo originario

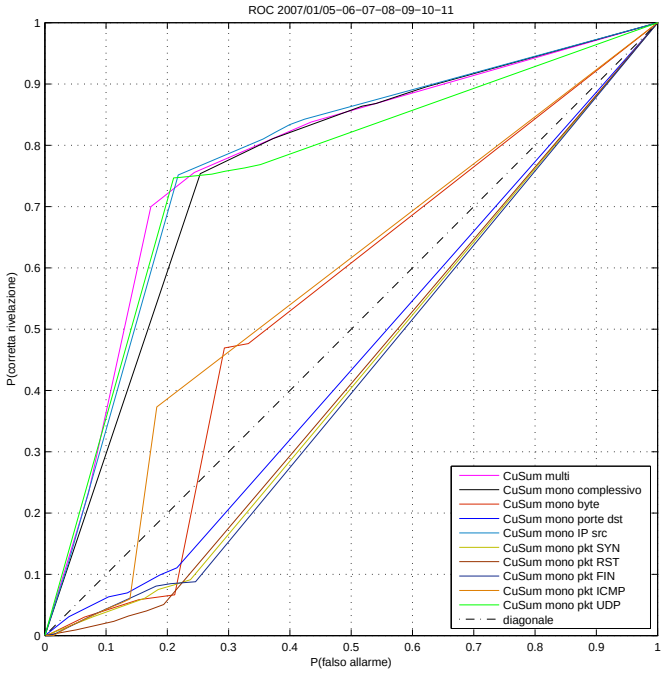

Figura 4.16: Confronto CuSum multi/mono-dimensionale; metodo "fn 4 detector"

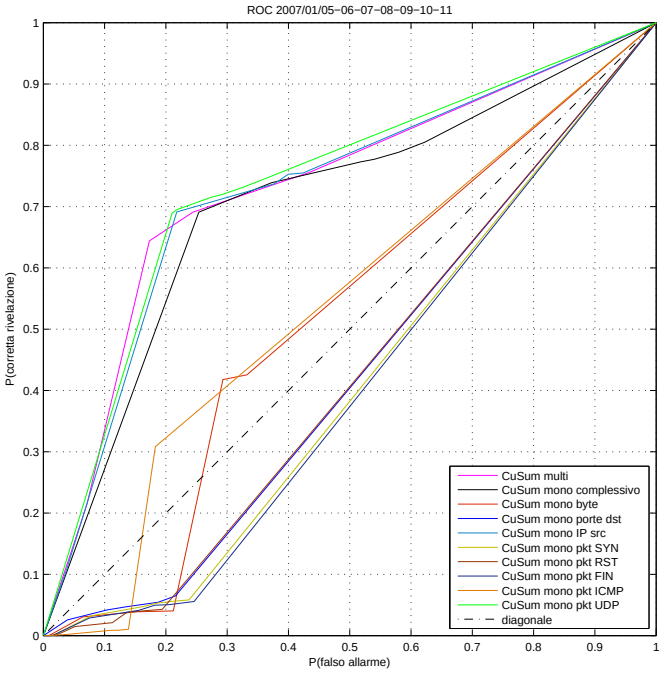

Figura 4.17: Confronto CuSum multi/mono-dimensionale; metodo "fn unknown"

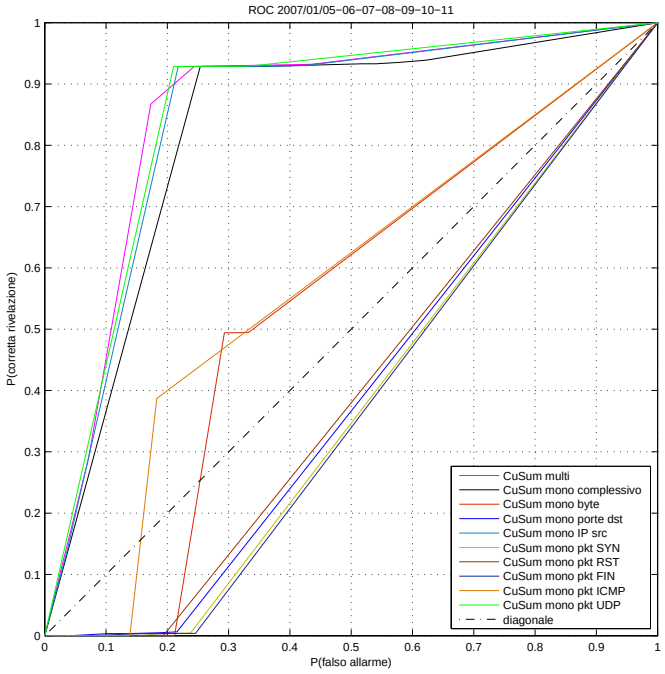

Figura 4.18: Confronto CuSum multi/mono-dimensionale; metodo "fn unknown 4 detector"

Ciò che si osserva nelle figure 4.15, 4.16, 4.17 e 4.18 è il fatto che la versione multidimensionale del CuSum ha prestazioni paragonabili a quelle che si ottengono eseguendo quella monodimensionale su ciascuna metrica e unendo i diversi risultati; inoltre si comporta meglio di ciascuna singola esecuzione.

Il vantaggio è quindi quello di ridurre il costo computazionale della ricerca delle intrusioni: monitorare nel complesso una serie temporale vettoriale è infatti meno pesante, in termini di tempo oppure di risorse (nel caso di un'analisi monodimensionale eseguita in parallelo), rispetto ad analizzare separatamente l'andamento dei diversi descrittori di traffico, ripetendo le stesse operazioni per ciascuno. Per confermare ciò confrontiamo il tempo di esecuzione dei due algoritmi sulla traccia del 2 gennaio 2007. Valutiamo per prima cosa l'esecuzione più veloce, utilizzando una soglia maggiore del valore massimo della statistica di decisione, in modo da non considerare, non rilevando anomalie, il tempo necessario all'individuazione delle chiavi responsabili: il CuSum multidimensionale ha impiegato 6823 secondi (1 ora e 54 minuti), mentre per eseguire la versione monodimensionale in sequenza su ogni metrica sono serviti complessivamente 12576 secondi (circa 3 ore e 30 minuti), con un incremento quindi dell'84%. Applicando invece in parallelo il CuSum monodimensionale sulle diverse metriche, con le stesse risorse computazionali del caso precedente assegnate a ciascuna esecuzione, sono trascorsi 5960 se-

condi: rispetto a questo valore, il CuSum multidimensionale ha impiegato un tempo maggiore solo del 14% necessitando di minori risorse. Dividendo per il tempo minore sperimentato, quello dell'esecuzione parallela del CuSum monodimensionale, otteniamo tempi normalizzati di 1.14 per il CuSum multidimensionale, 2.11 per il CuSum monodimensionale eseguito in serie e ovviamente 1.00 per quello eseguito in parallelo. Impiegando invece una soglia pari a 0 (il valore minimo assunto dalla statistica di decisione), ci troviamo invece nel caso in cui la fase di identificazione delle chiavi responsabili dei flussi anomali richiede il tempo massimo: la versione multidimensionale impiega 9480 secondi, quella monodimensionale 17080 secondi in serie, mentre in parallelo 8021 secondi; dividendo per quest'ultimo valore si hanno tempi normalizzati rispettivamente di 1.18, 2.13, e 1.00. Il tempo necessario ad unire i risultati delle esecuzioni del CuSum monodimensionale per le diverse metriche è trascurabile (17 secondi). Le tabelle 4.2 e 4.3 riassumono quanto detto. Risultati analoghi sono stati ottenuti anche per altre tracce.

Come abbiamo visto all'inizio di questa sezione, altri IDS basati su algoritmi multidimensionali hanno un comportamento simile a quello del CuSum, quindi gli stessi vantaggi sono riscontrabili anche per questi.

Versione CuSum	Tempo (sec)	Tempo normalizzato
multidimensionale	6823	1.14
monodimensionale in serie	12576	2.11
monodimensionale in parallelo	5960	1.00

Tabella 4.2: Tempi di esecuzione degli algoritmi CuSum sulla traccia del 2007/01/02;
nessuna anomalia rilevata

Versione CuSum	Tempo (sec)	Tempo normalizzato
multidimensionale	9480	1.18
monodimensionale in serie	17080	2.13
monodimensionale in parallelo	8021	1.00

Tabella 4.3: Tempi di esecuzione degli algoritmi CuSum sulla traccia del 2007/01/02;
numero massimo di anomalie (soglia = 0)

Conclusioni

Il lavoro svolto nel corso di questo elaborato ha cercato di verificare se è possbile eseguire una ricerca delle anomalie all'interno del traffico di una rete monitorata con un approccio multidimensionale; l'obiettivo è infatti considerare contemporaneamente più descrittori di traffico, in modo da individuare intrusioni di natura diversa, e allo stesso tempo ridurre il costo computazionale rispetto ad una stessa analisi eseguita con IDS monodimensionali classici.

I nuovi sistemi sviluppati hanno integrato differenti test statistici di change detection per valori multidimensionali con la tecnica di organizzazione dei dati del count min sketch, impiegata già in IDS monodimensionali esistenti, la cui struttura di base è quindi stata mantenuta.

Impiegando come dataset un sottoinsieme delle tracce di traffico appartenenti all'archivio MAWILab, per le quali sono disponibili informazioni riguardo alle attività anomale presenti, sono stati ottenuti risultati sperimentali che hanno permes-

so di valutare le prestazioni dei metodi di detection implementati. Non possiamo dire in assoluto che tali algoritmi si sono dimostrati molto efficienti; da un'analisi più dettagliata è stato comunque possibile osservare un miglioramento significativo nel caso della ricerca di anomalie di tipo sconosciuto, che, nell'ottica di un utilizzo congiunto con misuse based IDS, possiamo considerare l'obiettivo primario degli anomaly based IDS (la tipologia di quelli sviluppati nell'elaborato).

Infine, nei test eseguiti, il confronto con i risultati ottenuti applicando il metodo del CuSum monodimensionale impiegato come riferimento non ha evidenziato particolari differenze, dimostrando quindi che quello multidimensionale è un approccio che presenta possibili vantaggi in termini di tempi necessari per l'analisi o occupazione di risorse.

Appendice A

Conversione delle tracce MAWI

In questa appendice viene descritta la procedura adottata per convertire le tracce di traffico presenti nell'archivio MAWI nel formato adatto ad essere letto dai sistemi di detection implementati. Le tracce sono infatti disponibili sotto forma di file di tipo pcap, nei quali quindi le informazioni sono memorizzate pacchetto per pacchetto; i nostri algoritmi si aspettano invece in ingresso dati relativi ai flussi.

Per ottenere i file di input nel formato desiderato sono stati usati tre tool software reperibili liberamente in rete: Softflowd (disponibile in [6]), Nfdump e Nfcapd (disponibili in [3] e descritti in [4]).

Questi tre programmi operano insieme nel seguente modo:

- Softflowd, leggendo file pcap oppure mettendosi in ascolto su un'interfaccia di rete, ricostruisce i flussi di traffico e li esporta tramite il protocollo Netflow, in modo che possano essere raccolti dal tool Nfcapd.
- Nfcapd legge i dati provenienti da Softflowd e li registra in un file binario che può essere interpretato da Nfdump.
- Nfdump traduce il file proveniente da Nfcapd in un formato intelligibile sullo standard output (quindi da un terminale Linux), registrando statistiche flusso per flusso (indirizzi IP e porte sorgenti e destinatari, numero di byte, flag TCP, ecc...).

La descrizione dettagliata della loro sintassi di esecuzione può essere trovata in [7], [2] e [5]; noi ci limitiamo a vedere i comandi utilizzati per il nostro scopo.

Facciamo l'esempio di voler convertire la traccia MAWI relativa al 2 gennaio 2007, rappresentata dal file pcap "200701021400.dump", in un file "2007_01_02.dat" che contenga per ogni riga le informazioni associate ai flussi identificati dalla quintupla (indirizzo IP src, indirizzo IP dst, porta src, porta dst, protocollo). Operando sempre nella directory dove si vuole elaborare la traccia, dobbiamo

1. Aprire un'istanza della shell (ossia un terminale) e lanciare Nfcapd in modo che si metta in ascolto, con il co-

mando
nfcapd -t 120s -b localhost -l netflow/
dove le opzioni hanno il seguente significato:

- -t 120s: memorizza i dati in ingresso in file disponibili ogni 120s, con un nome della forma nfcapd.<YYYYMMDDhhmm> che indica l'istante di inizio (es: nfcapd.201205100915 contiene il traffico arrivato in ingresso al tool il 10 maggio 2012 dalle 9:15 per la durata dei secondi specificati nell'opzione);
- -b localhost: colleziona i file di uscita localmente;
- -l netflow/: nome della directory dove mettere i file di uscita.

2. Aprire un'altra istanza della shell dove lanciare
softflowd -v 9 -n localhost:9995 -m 1500
-r 200701021400.dump

   - -v 9: esporta flussi NetFlow v.9;
   - -n localhost:9995: esporta i flussi sull'host locale sulla porta 9995 (quella dove sta in ascolto Nfcapd);
   - -m 1500: inizia a esportare i flussi quando il loro numero supera 1500;
   - -r 200701021400.dump: file pcap da convertire.

3. Quando Softflowd ha terminato, interrompere Nfcapd manualmente (Ctrl+C), eventualmente rinominare se si vuole il file creato (es con nfcapd.2007_01_02) ed eseguire nfdump -a -N
   -o "fmt:%pr,%sap,%dap,%flg,%pkt,%byt,%fl"
   -r nfcapd.2007_01_02 inet | tee 2007_01_02.dat

   - -a: aggrega i flussi per quintupla;
   - -N: stampa il protocollo come numero e i byte senza unità di misura;
   - -o "fmt:%pr,%sap,%dap,%flg,%pkt,%byt,%fl": formato di ogni riga del file di uscita, in cui si hanno protocollo, indirizzo IP e porta src (es. 192.168.0.1:80), indirizzo IP e porta dst (nel caso di ICMP la porta dst ha il formato type.code, es: 8.0), flag TCP (nell'ordine URG, ACK, PUSH, RST, SYN, FIN oppure rappresentati da un numero esadecimale considerando, se settati, anche i flag CWR e ECN), numero di pacchetti, numero di byte, numero di flussi;
   - -r nfcapd.2007_01_02: file di ingresso da convertire;
   - inet: filtro per registrare solo il traffico IPv4; ulteriori filtri, costruiti a partire da quelli presenti nei file xml dell'archivio MAWI che identificano il traffico anomalo, possono essere inseriti per ripulire la traccia dalle anomalie;

- tee 2007_01_02.dat: comando (scritto dopo la pipe "|") per salvare l'output sul file 2007_01_02.dat.

Appendice B

Codice CuSum multidimensionale

Di seguito viene riportato il codice sorgente scritto in linguaggio C dell'IDS che impiega l'algoritmo CuSum multidimensionale.

```
/**
 * sketch_CUSUM.c

 * il codice lavora sulle tracce prese
 dall'archivio MAWILab (ciascuna corrisponde
 ad un time bin di 15min):
  - costruisce gli sketch di riferimento
  dalle prime N1 tracce ripulite dalle
  anomalie;
```

```
 - costruisce per ogni traccia successiva
 (comprendente flussi anomali) lo sketch di
 test;
 - confronta lo sketch di test con quelli di
 riferimento secondo il CUSUM;
 - se la distanza supera una certa soglia
 (il cui valore e' passato da riga di
 comando) indica la presenza di un'anomalia.

* prevede la possibilta' di passare piu'
valori diversi della soglia: esegue un
confronto per ciascuno.

* file di uscita:
- "vertical_anomalous_time_bin.txt"
file dei time bin anomali
- "dst_IP_anomali_<data>.txt"
file degli indirizzi IP destinatari
responsabili delle anomalie
dove <data> indica la data della traccia di
test analizzata
(es. dst_IP_anomali_2007_01_06.txt)

* parametri da riga di comando:
- NUM_FILE   numero di tracce totali (N1 di
       riferimento + NUM_FILE-N1 da
       testare)
- N1     numero di time bin di riferimento
```

```
 - percorso_ris percorso relativo dove
 inserire i risultati
 - th      valori delle soglie (anche piu'
           di 1, devono essere gli ultimi
           parametri passati)

 * sintassi di esecuzione:
 ./sketch_CUSUM NUM_FILE N1 percorso_ris
 soglia_1 soglia_2 ... soglia_n

 * esempio di esecuzione:
 ./sketch_CUSUM 11 3 ./risultati/rif_3_tracce
 0 1

 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <regex.h>
#include <unistd.h>
#include "funzioni_vettori.h"
#include "liste.h"
#include <float.h>

/* dimensioni dello sketch */
#define D  16
```

CODICE CUSUM MULTIDIMENSIONALE

```
#define W 512

/* numero di righe della matrice degli
attacchi per cui si puo' stabilire che il
time-bin sia anomalo */
#define h_anom 15

/* numero di metriche usate */
#define L 8

#define K 4

#define max(A, B) ((A) > (B) ? (A) : (B))
#define modulodif(a, b) ((a)>(b)?(a)-(b):(b)-(a))

#define DEBUG 1

/* definisco l'array di coefficienti per le
hash */
/* numero delle h per la divisione della
chiave in parti piu' piccole */
#define M 3
unsigned int ***hash;
unsigned int **T0,**T1,**T2;
double **attacks;
#define K1 65536
#define K2 131071
#define P 131071 /* numero primo */
```

```
/* --- fine definizioni --- */

/* Generazione tabelle di hash */

unsigned int make_hash
(unsigned long long* esp, int d, int n)
{
  int i;
  unsigned int h;
  unsigned long long a;
  a = 1LL*hash[d][n][0];
  for (i = 1; i < K; i++)
  {
    a = (a + (1LL*hash[d][n][i]*esp[i]) % P)
      % P;
  }
  h = (unsigned int) (a % P);
  return h;
}

void genera_tabelle ()
{
  unsigned int key;
  int i, n;
  unsigned long long *esp;
  esp = (unsigned long long*) malloc
    (sizeof(unsigned long long)*K);
```

CODICE CUSUM MULTIDIMENSIONALE

```
  for (i = 0; i < D; i++)
  {
    for (key = 0; key < K1; key++)
    {
      esp[0] = 1LL;
      for (n = 1; n < K; n++)
      {
        esp[n] = (esp[n-1]*key);
      }
      T0[i][key] = make_hash(esp, i, 0);
      T1[i][key] = make_hash(esp, i, 1);
      T2[i][key] = make_hash(esp, i, 2);
    }
    for (key = K1; key < K2; key++)
    {
      esp[0] = 1LL;
      for (n = 1; n < K; n++)
      {
        esp[n] = (esp[n-1]*key);
        /* il numero piu' grosso che viene
        fuori sta su 51 bit con K = 4 */
      }
      T2[i][key] = make_hash(esp, i, 2);
    }
  }
  free(esp);
  return;
}
```

```
/**
 * traduce un vettore di 4 elementi
 contenente un indirizzo IP in un numero
 decimale
 *
 * @param ip  vettore di 4 elementi
        contenente un indirizzo IP da
        tradurre
 *
 * @return  numero decimale corrispondente
      all'indirizzo IP
 */
unsigned int ip2dec (unsigned int* ip)
{
  unsigned int result = 0;
  int i;
  for (i = 0; i < 4; i++)
  {
    result = ( result << 8 ) | ip[i];
  }
  return result;
}

/**
 * traduce un numero decimale in un vettore
 di 4 elementi contenente un indirizzo IP
 *
```

```
 * @param key   numero decimale da tradurre
 * @param ip    vettore di 4 elementi da
          riempire con l'indirizzo IP
 */
void dec2ip (unsigned int key,
            unsigned int* ip)
{
  ip[3] = key % 256;
  ip[2] = (key>>8) % 256;
  ip[1] = (key>>16) % 256;
  ip[0] = key>>24;
  return;
}

/**
 * aggiorna lo sketch
 *
 * @param sketch  sketch da aggiornare
 * @param Mkey    chiave di cui fare l'hash
 * @param metriche vettore dei valori con cui
                    aggiornare lo sketch
 */
void aggiorna_sketch (double ***sketch,
      unsigned int Mkey, double *metriche)
{
  unsigned int x0,x1,x2,h;
  int d, l;
  for (d = 0; d < D; d++)
```

```
  {
    x0 = Mkey % 16;
    x1 = Mkey >> 16;
    x2 = x0+x1;
    h = (unsigned int)
      (T0[d][x0]^T1[d][x1]^T2[d][x2])%W;
    for (l = 0; l < L; l++)
    {
      sketch[d][h][l] += metriche[l];
    }
  }
  return;
}


int main (int argc, char* argv[])
{
  unsigned int x, Mkey;
  /* vettore delle metriche */
  double metriche[L];
  int i, j, k, d, w, cont;
  /* indice della metrica nelle caselle dello
  sketch */
  int l;
  /* costanti di normalizzazione per la
  costruzione degli sketch */
  double cost_norm[L];
  /* numero di valori degli sketch di
```

```
riferimento usato per calcolare le costanti
di normalizzazione */
unsigned int num_val_rif;
/* numero totale di tracce */
unsigned int NUM_FILE;
/* numero di sketch di riferimento */
unsigned int N1;
/* indice del primo time bin da cui si
inizia a testare */
unsigned int inizio_test;
/* insieme degli sketch di riferimento */
double ****sketch_rif;
/* sketch calcolato nel time bin
corrente */
double ***sketch;
/* sketch calcolato nel time bin
precedente */
double ***sketch_prec;
/* valor medio degli sketch */
double ***media;
/* varianza degli sketch */
double ***varianza;
/* parametro per il calcolo del valor medio
e varianza con l'algoritmo EWMA */
double alpha = 0.9;
/* valore cusum  */
double **cusum;
/* parametro per il calcolo del cusum */
```

```
double c = 0.5;
double distanza;
/* fattore per cui moltiplicare il cusum
per calcolare la distanza */
double fattore_dist = 100;
unsigned int x0, x1, x2, h;
/* nome del file di ingresso aperto ogni
volta (compreso il percorso assoluto) */
char file[128];
/* estensione del nome dei file (es:.dat)*/
char estensione[8];
/* percorso assoluto dei file */
char percorso[128];
/* array con i nomi dei file di ingresso
(solo il nome del file, senza il percorso e
l'estensione) */
char **nome_file_in;
/* soglia */
long long int th;
/* valore restituito dalla fork per creare
processi figli */
pid_t pid_fork;
/* file di ingresso contenente le tracce da
leggere */
FILE *f_in;
/* file di uscita */
FILE *f1, *f2;
/* liste usate per l'identificazione degli
```

```
indirizzi IP destinatari anomali */
/* matrice di D*W liste degli indirizzi IP
mappati in un certo valore hash:
hash_IP[d][w] lista degli IP il cui hash
con la funzione hash d-esima e' w */
lista_IP *hash_IP[D][W];
/* array di D liste degli indirizzi IP
responsabili delle anomalie: anom_IP[d]
lista degli IP responsabili delle
anomalie nella riga d-esima dello sketch */
lista_IP *anom_IP[D];
/* puntatori di appoggio per scorrere le
liste hash_IP e anom_IP */
lista_IP *pApp_IP, *pApp_IP_2;

/* variabili usate per leggere i file delle
tracce e costruire gli sketch */
/* campo protocol */
int protocol;
/* indirizzo IP sorgente */
unsigned int src_IP[4];
/* porta sorgente */
unsigned int src_port;
/* indirizzo IP destinatario */
unsigned int dst_IP[4];
/* porta destinataria */
unsigned int dst_port;
/* type e code nel caso protocollo ICMP */
```

```
unsigned int icmp_type, icmp_code;
/* flag TCP */
int cwr = 0, ece = 0, urg = 0, ack = 0,
  psh = 0, rst = 0, syn = 0, fin = 0;
/* flag TCP espressi in esadecimale */
unsigned int flag_0x;
/* numero di flag TCP */
const unsigned int num_flag = 6;
/* numero di pacchetti */
unsigned int pkt = 0;
/* numero di byte */
unsigned int byte = 0;
/* numero di flussi */
unsigned int flow = 0;
/* valore dei campi packet, byte o flow
scalato per l'unita' di misura */
float dimensione;
/*unita' di misura (fattore moltiplicativo)
del numero di pkt, byte o flussi */
unsigned int unita_misura;
/* numero di campi letti per riga */
unsigned int campi_letti = 0;
/* campo letto dalla riga */
char *campo;
/* buffer in cui inserire la riga letta dal
file */
char *riga;
/* dimensione della riga in byte */
```

```
unsigned int dim_riga = 128;
/* copia della riga usata nella funzione
strsep */
char *riga_temp;
/*separatori dei campi presenti nel file di
ingresso: virgola */
const char *sep_campi = ",";
/* formato delle righe del file */
const char *formato_riga =
"[0-9]+[\t ]*,[\t ]*(([0-9]{1,3}[.]){3}"
"[0-9]{1,3}:[0-9]{1,5}[.]*[0-9]*[\t ]*,"
"[\t ]*){2}(([UAPRSF.]{6})|(0x[0-9a-fA-F]"
"{2}))[\t ]*,[\t ]*([0-9.]+[ a-zA-Z]*"
"[\t ]*,[\t ]*){2}[0-9]+";
/* espressione regolare usata per
controllare il formato della riga */
regex_t re;
/* puntatore ad una lista contenente per
ogni IP dst la lista degli IP src usati e
lista delle porte dst usate */
lista_SIP_DIP_DP *pun_SIP_DIP_DP;
/* puntatore di appoggio per scorrere la
lista SIP_DIP_DP */
lista_SIP_DIP_DP *pApp_SIP_DIP_DP;
/* puntatore di appoggio per scorrere la
lista degli indirizzi IP sorgente */
lista_IP *pApp_DIP;
/* puntatore di appoggio per scorrere la
```

```
lista delle porte destinatarie */
lista_porte *pApp_DP;
double dist_min = DBL_MAX;
double dist_max = DBL_MIN;
/* buffer per scrivere sullo stdout con la
funzione write */
char buffer[256];

/* controllo il numero di parametri passati
da riga di comando */
if (argc < 5)
{
  fprintf(stderr,"Errore: numero parametri"
    "da riga di comando insufficiente\n");
  fprintf(stderr,"Parametri:\tNUM_FILE\tN1"
    "\tpercorso_ris\tsoglie\n");
  return 1;
}

/* compilo l'espressione regolare del
formato della riga dei file delle tracce */
if (regcomp(&re, formato_riga,
  REG_EXTENDED|REG_ICASE|REG_NOSUB) != 0)
{
  fprintf(stderr, "Errore lettura file: "
  "compilazione espressione regolare "
  "formato riga\n");
  return 1;
```

```
}

/* numero totale di tracce */
NUM_FILE = (unsigned int) strtol(argv[1],
(char**)NULL, 10);
/* numero di sketch di riferimento */
N1 = (unsigned int) strtol(argv[2],
(char**)NULL, 10);
if (NUM_FILE < N1+1)
{
  fprintf(stderr, "Errore numero tracce: "
  "NUM_FILE < N1+1\n");
  return 1;
}
/* stampo a video i parametri passati da
riga di comando */
sprintf(buffer, "Numero tracce di "
"riferimento\tN1 = %u\n", N1);
write(1, buffer, strlen(buffer));
sprintf(buffer, "Numero tracce totali\t"
"NUM_FILE = %u\n", NUM_FILE);
write(1, buffer, strlen(buffer));
sprintf(buffer, "Percorso dei "
"risultati\t%s\n", argv[3]);
write(1, buffer, strlen(buffer));

/* stampo a video i parametri utilizzati
dall'algoritmo cusum */
```

```
sprintf(buffer, "Parametri impiegati "
"dall'algoritmo:\talpha = %.2f\tc = %.2f\t"
"fattore_distanza = %.2f\n", alpha, c,
fattore_dist);
write(1, buffer, strlen(buffer));

/* genera coefficienti per le k-universal
hash */
srand(871);
hash = (unsigned int***) malloc
        (sizeof(unsigned int**)*D);
for (i = 0; i < D; i++)
{
  hash[i] = (unsigned int**) malloc
            (sizeof(unsigned int*)*M);
  for (k = 0; k < M; k++)
  {
    hash[i][k] = (unsigned int*) malloc
                 (sizeof(unsigned int)*K);
    for (j = 0; j < K; j++)
    {
      hash[i][k][j] = rand();
    }
  }
}

T0 = (unsigned int**) malloc
      (sizeof(unsigned int*)*D);
```

CODICE CUSUM MULTIDIMENSIONALE

```
T1 = (unsigned int**) malloc
      (sizeof(unsigned int*)*D);
T2 = (unsigned int**) malloc
      (sizeof(unsigned int*)*D);
for (i = 0; i < D; i++)
{
  T0[i] = (unsigned int*) malloc
          (sizeof(unsigned int)*K1);
  T1[i] = (unsigned int*) malloc
          (sizeof(unsigned int)*K1);
  T2[i] = (unsigned int*) malloc
          (sizeof(unsigned int)*K2);
}

/* genero le tabelle e dealloco i
coefficienti */
genera_tabelle();

for (i = 0; i < D; i++)
{
  for (k = 0; k < M; k++)
  {
    free(hash[i][k]);
  }
  free(hash[i]);
}
free(hash);
```

CODICE CUSUM MULTIDIMENSIONALE

```
/* inizializzazione dell'insieme degli
sketch di riferimento */
sketch_rif = (double****) malloc
              (sizeof(double***)*N1);
for (x = 0; x < N1; x++)
{
  sketch_rif[x] = (double***) malloc
                  (sizeof(double**)*D);
  for (i = 0; i < D; i++)
  {
    sketch_rif[x][i] = (double**) malloc
                       (sizeof(double*)*W);
    for (j = 0; j < W; j++)
    {
      sketch_rif[x][i][j] = (double*)malloc
                            (sizeof(double)*L);
      for (l = 0; l < L; l++)
      {
        sketch_rif[x][i][j][l] = 0;
      }
    }
  }
}

/* nomi dei file di ingresso
(senza il percorso) */
nome_file_in = (char**) malloc
               (sizeof(char*)*NUM_FILE);
```

CODICE CUSUM MULTIDIMENSIONALE

```
for (x = 0; x < N1; x++)
{
  nome_file_in[x] = (char*) malloc
                    (sizeof(char)*128);
  /* file con le tracce di riferimento
  (prive di anomalie) */
  if (x < 9)
  {
    sprintf(nome_file_in[x],
    "2007_01_0%d_senza_anom_port", x+1);
  }
  else
  {
    sprintf(nome_file_in[x],
    "2007_01_%d_senza_anom_port", x+1);
  }
}
for (x = N1; x < NUM_FILE; x++)
{
  nome_file_in[x] = (char*) malloc
                    (sizeof(char)*128);
  /* file con le tracce di test */
  if (x < 9)
  {
    sprintf(nome_file_in[x],
    "2007_01_0%d", x+1);
  }
  else
```

```
  {
    sprintf(nome_file_in[x],
    "2007_01_%d", x+1);
  }
}
/* estensione dei file */
strcpy(estensione, ".dat");
/* directory contenente i file di ingresso
(percorso relativo) */
strcpy(percorso, "../../../Tracce_MAWI/"
"tracce/tracce_input_change_detection/");

/* allocazione buffer per leggere le righe
del file di ingresso */
riga = (char*) malloc
        (sizeof(char)*dim_riga);

/* costruzione dell'insieme degli sketch di
riferimento; ipotesi: primi N1 time bin non
anomali */
for (x = 0; x < N1; x++)
{
  /* apro il file di ingresso in lettura */
  strcpy(file, percorso);
  strcat(file, nome_file_in[x]);
  strcat(file, estensione);
  f_in = fopen(file, "r");
  if (f_in == NULL)
```

```
{
  fprintf(stderr,
  "Errore apertura file %s\n", file);
  return 1;
}
sprintf(buffer, "Analisi traccia di"
    "riferimento %s (time bin %u) ...\n",
    file, x);
write(1, buffer, strlen(buffer));
/* inizializzo la lista SIP_DIP_DP */
pun_SIP_DIP_DP = NULL;
/* leggo il file una riga per volta */
while (fgets(riga,dim_riga, f_in)!=NULL)
{
  if (regexec(&re, riga, (size_t) 0,
              NULL, 0) == 0)
  {
    /* se la riga letta ha il formato
    corretto leggo i vari campi */
    campi_letti = 0;
    /* inizializzo a 0 dst_port: nel
    caso di ICMP non viene assegnato
    nessun valore a tale variabile */
    dst_port = 0;
    riga_temp = riga;
    for (;;)
    {
      /* leggo un campo della riga */
```

```
campo = strsep(&riga_temp,
               sep_campi);
if (campo == NULL)
{
  /* riga terminata */
  break;
}
/* elimino gli spazi e le
tabulazioni in testa a campo
(spostando il puntatore al primo
carattere diverso da spazio o \t)*/
while ((campo[0] == ' ') ||
       (campo[0] == '\t'))
{
  campo++;
}
if (campo[0] == '\0')
{
  /* piu' separatori consecutivi */
  continue;
}
campi_letti++;
unita_misura = 1;
for (i = 0; campo[i] != '\0'; i++)
{
  if ((campo[i] == ' ') ||
      (campo[i] == '\t'))
  {
```

```
  campo[i] = '\0';
  if ((campi_letti != 5) &&
      (campi_letti != 6) &&
      (campi_letti != 7))
  {
    /* elimino gli spazi in coda
    se campo non corrisponde al
    numero di pkt,byte o flussi*/
    break;
  }
}
if ((campi_letti == 5) ||
    (campi_letti == 6) ||
    (campi_letti == 7))
{
  /* leggo l'unita' di misura
  (se presente) del numero di
  pkt, byte o flussi */
  if (campo[i] == 'K')
  {
    unita_misura = 1000;
    break;
  }
  else if (campo[i] == 'M')
  {
    unita_misura = 1000000;
    break;
  }
```

```
      else if (campo[i] == 'G')
      {
        unita_misura = 1000000000;
        break;
      }
    }
  }
  switch (campi_letti)
  {
    case 1:
      /* campo Protocol */
      sscanf(campo, "%u", &protocol);
      break;
    case 2:
      /* campo SrcIPAddr:Port */
      sscanf (campo,"%u.%u.%u.%u:%u",
              &src_IP[0], &src_IP[1],
              &src_IP[2], &src_IP[3],
              &src_port);
      break;
    case 3:
      /* campo DstIPAddr:Port */
      if (protocol == 1)
      {
        /* ICMP: DstPort corrisponde
        a type.code */
        sscanf (campo,
            "%u.%u.%u.%u:%u.%u",
```

```
            &dst_IP[0], &dst_IP[1],
            &dst_IP[2], &dst_IP[3],
            &icmp_type, &icmp_code);
      }
      else
      {
        sscanf (campo,
              "%u.%u.%u.%u:%u",
              &dst_IP[0], &dst_IP[1],
              &dst_IP[2], &dst_IP[3],
              &dst_port);
      }
      break;
    case 4:
      /* campo flag TCP */
      cwr = ece = urg = ack = 0;
      psh = rst = syn = fin = 0;
      if (campo[0] == '0')
      {
        /* flag espressi come numero
        esadecimale */
        sscanf (campo, "%x",
                &flag_0x);
        cwr = (flag_0x & 128) >> 7;
        ece = (flag_0x & 64) >> 6;
        urg = (flag_0x & 32) >> 5;
        ack = (flag_0x & 16) >> 4;
        psh = (flag_0x & 8) >> 3;
```

```
    rst = (flag_0x & 4) >> 2;
    syn = (flag_0x & 2) >> 1;
    fin = (flag_0x & 1);
  }
  else
  {
    /* flag espressi come
    sequenza di caratteri */
    for(i = 0; i < num_flag; i++)
    {
      switch (campo[i])
      {
        case 'U':
        case 'u':
          urg = 1;
          break;
        case 'A':
        case 'a':
          ack = 1;
          break;
        case 'P':
        case 'p':
          psh = 1;
          break;
        case 'R':
        case 'r':
          rst = 1;
          break;
```

```
            case 'S':
            case 's':
              syn = 1;
              break;
            case 'F':
            case 'f':
              fin = 1;
              break;
          }
        }
      }
      break;
    case 5:
      /* campo Packets */
      sscanf(campo, "%f",
            &dimensione);
      pkt = (unsigned int)
        (dimensione * unita_misura);
      break;
    case 6:
      /* campo Bytes */
      sscanf(campo, "%f",
            &dimensione);
      byte = (unsigned int)
        (dimensione * unita_misura);
      break;
    case 7:
      /* campo Flows */
```

```
        sscanf(campo, "%f",
              &dimensione);
        flow = (unsigned int)
          (dimensione * unita_misura);
        break;
    }
}
/* metrica 0: numero di byte */
metriche[0] = byte;
/* metrica 1: numero di porte (TCP o
UDP) destinatarie diverse usate
dall'indirizzo IP dst; in particolare
metriche[1] = 0 se la porta dst e'
gia' stata usata
metriche[1] = 1 se la porta dst non
e' gia' stata usata
tramite l'incremento eseguito nella
costruzione sketch verra' contato il
numero di porte dst diverse */
/* metrica 2: numero di indirizzi IP
sorgente diversi usati dall'indirizzo
IP dst; in particolare
metriche[2] = 0 se l'indirizzo IP src
e' gia' stata usato
metriche[2] = 1 se l'indirizzo IP src
non e' gia' stato usato
tramite l'incremento eseguito nella
costruzione sketch verra' contato il
```

```
        numero di IP src diversi */
        metriche[1] = metriche[2] = 0;
        for (pApp_SIP_DIP_DP=pun_SIP_DIP_DP;
    pApp_SIP_DIP_DP != NULL;
    pApp_SIP_DIP_DP = pApp_SIP_DIP_DP->pNext)
        {
          /* scorro la lista degli indirizzi
          IP destinatari finche' non trovo
          quello analizzato */
          /* attenzione: i membri della lista
          pApp_SIP_DIP_DP->src_IP[0],...,
          pApp_SIP_DIP_DP->src_IP[3] vengono
          utilizzati come IP dst */
          if ((dst_IP[0] ==
              pApp_SIP_DIP_DP->src_IP[0]) &&
              (dst_IP[1] ==
              pApp_SIP_DIP_DP->src_IP[1]) &&
              (dst_IP[2] ==
              pApp_SIP_DIP_DP->src_IP[2]) &&
              (dst_IP[3] ==
              pApp_SIP_DIP_DP->src_IP[3]))
          {
            /* esco dal ciclo:
            pApp_SIP_DIP_DP punta
            all'elemento con dst_IP cercato*/
            break;
          }
        }
```

```
if (pApp_SIP_DIP_DP == NULL)
{
  /* indirizzo IP destinatario non
  presente nella lista:lo inserisco*/
  if ((protocol == 6) ||
  (protocol == 17))
  {
    /* se il protocollo e' TCP o UDP
    inserisco la porta dst nella
    lista (ultimo argomento = 1) */
    inserisci_sip_dip_dp
    (&pun_SIP_DIP_DP, dst_IP[0],
    dst_IP[1], dst_IP[2], dst_IP[3],
    src_IP[0], src_IP[1], src_IP[2],
    src_IP[3], dst_port, 1);
    /* setto a 1 l'indicatore del
    numero di porta destinataria
    diverso (da quelli gia' usati) */
    metriche[1] = 1;
  }
  else
  {
    /* protocollo non TCP o UDP: non
    inserisco la porta dst nella
    lista (ultimo argomento = 0) */
    inserisci_sip_dip_dp
    (&pun_SIP_DIP_DP, dst_IP[0],
    dst_IP[1], dst_IP[2], dst_IP[3],
```

```
      src_IP[0], src_IP[1], src_IP[2],
      src_IP[3], dst_port, 0);
    }
    /* setto a 1 l'indicatore
    dell'indirizzo IP sorgente diverso
    (da quelli gia' usati) */
    metriche[2] = 1;
  }
  else
  {
    /* indirizzo IP destinatario
    presente nella lista:
    pApp_SIP_DIP_DP punta all'elemento
    che lo contiene */
    if ((protocol == 6) ||
        (protocol == 17))
    {
      /* protocollo TCP o UDP */
      for (pApp_DP =
          pApp_SIP_DIP_DP->lista_DP;
          pApp_DP != NULL;
          pApp_DP = pApp_DP->pNext)
      {
        /* scorro la lista delle porte
        destinatarie (per il
        particolare indirizzo IP
        destinatario) finche' non trovo
        quello analizzata */
```

```
      if (dst_port == pApp_DP->port)
      {
        /* esco dal ciclo */
        break;
      }
    }
    if (pApp_DP == NULL)
    {
      /* porta destinataria non
      presente nella lista: la
      inserisco */
      inserisci_porta
      (&(pApp_SIP_DIP_DP->lista_DP),
      dst_port);
      /* setto a 1 l'indicatore del
      numero di porta destinataria
      diverso(da quelli gia' usati)*/
      metriche[1] = 1;
    }
  }
  for (pApp_DIP =
      pApp_SIP_DIP_DP->lista_DIP;
      pApp_DIP != NULL;
      pApp_DIP = pApp_DIP->pNext)
  {
    /* scorro la lista degli
    indirizzi IP sorgente
    (per il particolare indirizzo IP
```

```
      destinatario) finche' non trovo
      quello analizzato */
      if((src_IP[0]==pApp_DIP->ip[0])&&
        (src_IP[1]==pApp_DIP->ip[1]) &&
        (src_IP[2]==pApp_DIP->ip[2]) &&
        (src_IP[3]==pApp_DIP->ip[3]))
      {
        /* esco dal ciclo */
        break;
      }
    }
    if (pApp_DIP == NULL)
    {
      /* indirizzo IP sorgente non
      presente nella lista:
      lo inserisco */
      inserisci_ip
      (&(pApp_SIP_DIP_DP->lista_DIP),
      src_IP[0], src_IP[1], src_IP[2],
      src_IP[3]);
      /* setto a 1 l'indicatore
      dell'indirizzo IP sorgente
      diverso (da quelli gia' usati) */
      metriche[2] = 1;
    }
  }
  /* metrica 3: numero pkt TCP SYN */
  /* metrica 4: numero pkt TCP RST */
```

```
/* metrica 5: numero pkt TCP FIN */
/* metrica 6: numero pkt ICMP */
/* metrica 7: numero pkt UDP */
metriche[3] = metriche[4] = 0;
metriche[5] = metriche[6] = 0;
metriche[7] = 0;
switch (protocol)
{
  case 6:
    if (syn == 1)
    {
      metriche[3] = pkt;
    }
    if (rst == 1)
    {
      metriche[4] = pkt;
    }
    if (fin == 1)
    {
      metriche[5] = pkt;
    }
    break;
  case 1:
    metriche[6] = pkt;
    break;
  case 17:
    metriche[7] = pkt;
    break;
```

```
        }
        /* chiave: indirizzo IP dst */
        Mkey = ip2dec(dst_IP);
        aggiorna_sketch (sketch_rif[x], Mkey,
                         metriche);
      }
    }
    /* libero la memoria allocata per la
    lista SIP_DIP_DP */
    libera_sip_dip_dp (&pun_SIP_DIP_DP);
    fclose(f_in);
}

/*inizializzo sketch attuale e precedente*/
sketch = (double***) malloc
          (sizeof(double**)*D);
sketch_prec = (double***) malloc
          (sizeof(double**)*D);
/* inizializzazione della media */
media = (double***) malloc
          (sizeof(double**)*D);
/* inizializzazione della varianza */
varianza = (double***) malloc
            (sizeof(double**)*D);
/* inizializzazione del cusum */
cusum = (double**) malloc
          (sizeof(double*)*D);
for (i = 0; i < D; i++)
```

```
{
  sketch[i] = (double**) malloc
              (sizeof(double*)*W);
  sketch_prec[i] = (double**) malloc
                   (sizeof(double*)*W);
  media[i] = (double**) malloc
             (sizeof(double*)*W);
  varianza[i] = (double**) malloc
                (sizeof(double*)*W);
  cusum[i] = (double*) malloc
             (sizeof(double)*W);
  for (j = 0; j < W; j++)
  {
    sketch[i][j] = (double*) malloc
                   (sizeof(double)*L);
    sketch_prec[i][j] = (double*) malloc
                        (sizeof(double)*L);
    media[i][j] = (double*) malloc
                  (sizeof(double)*L);
    varianza[i][j] = (double*) malloc
                     (sizeof(double)*L);
    cusum[i][j] = 0;
  }
}

/* numero valori sketch di riferimento */
num_val_rif = N1*D*W;
/* calcolo costanti di normalizzazione e
```

```
normalizzazione sketch di riferimento */
for (l = 0; l < L; l++)
{
  /* inizializzazione */
  cost_norm[l] = 0;
  /* calcolo: media dei valori degli sketch
  di riferimento (se diversa da 0) */
  for (x = 0; x < N1; x++)
  {
    for (i = 0; i < D; i++)
    {
      for (j = 0; j < W; j++)
      {
        cost_norm[l] +=
                    sketch_rif[x][i][j][l];
      }
    }
  }
  if (cost_norm[l] == 0)
  {
    cost_norm[l] = 1;
  }
  else
  {
    cost_norm[l]= cost_norm[l]/num_val_rif;
  }
  /*normalizzo gli sketch di riferimento */
  for (i = 0; i < D; i++)
```

```
  {
    for (j = 0; j < W; j++)
    {
      /* primo sketch di riferimento:
      normalizzazione e inizializzazione
      valori usati nel calcolo del cusum */
      sketch_rif[0][i][j][l]/=cost_norm[l];
      sketch_prec[i][j][l] =
                    sketch_rif[0][i][j][l];
      media[i][j][l] =
                    sketch_rif[0][i][j][l];
      varianza[i][j][l] = 0;
      /* sketch di riferimento successivi:
      normalizzazione */
      for (x = 1; x < N1; x++)
      {
        sketch_rif[x][i][j][l]/=cost_norm[l];
      }
    }
  }
}
sprintf(buffer, "Costruzione degli sketch "
              "di riferimento eseguita\n");
write(1, buffer, strlen(buffer));

/* soglia nel padre: primo valore da riga
di comando (stringa convertita in intero)*/
th = (long long int) strtol(argv[4],
```

```
      (char**)NULL, 10);
/* valore restituito dalla fork: lo
inizializzo con un valore > 0 */
pid_fork = 1;
/* creo un processo figlio per ogni altra
soglia passata da riga di comando */
for (i = 5; i < argc; i++)
{
  if (pid_fork != 0)
  {
    /* sono nel padre: creo un figlio */
    pid_fork = fork();
  }
  if (pid_fork == 0)
  {
    /* sono nel processo figlio: assegno
    alla soglia il valore i-esimo passato
    da riga di comando */
    th = (long long int) strtol(argv[i],
          (char**)NULL, 10);
    /* esco dal ciclo */
    break;
  }
}

/* cusum sull'insieme di riferimento */
sprintf(buffer, "Soglia %Ld; calcolo cusum"
" sull'insieme di riferimento ...\n", th);
```

```
write(1, buffer, strlen(buffer));
for (x = 1; x < N1; x++)
{
  if (DEBUG)
  {
    /* inizializzazione distanza minima e
    massima */
    dist_min = DBL_MAX;
    dist_max = DBL_MIN;
  }
  for (i = 0; i < D; i++)
  {
    for (j = 0; j < W; j++)
    {
      /* calcolo del cusum */
      cusum[i][j] +=
      dist_euclid (sketch_rif[x][i][j],
          sketch_prec[i][j], L) -
          norma(media[i][j], L) -
          c*sqrt(norma(varianza[i][j], L));
      if (cusum[i][j] < 0)
      {
        cusum[i][j] = 0;
      }
      /* calcolo la distanza */
      distanza = fattore_dist*cusum[i][j];
      /* confronto distanza con soglia */
      if (distanza > th)
```

```
{
  /* se la distanza supera la soglia
  reinizializzo il cusum a 0 */
  cusum[i][j] = 0;
}
else
{
  /* se la distanza non supera la
  soglia aggiorno sketch_prec,
  media e varianza */
  for (l = 0; l < L; l++)
  {
    sketch_prec[i][j][l] =
              sketch_rif[x][i][j][l];
    media[i][j][l] =
    alpha*media[i][j][l] +
    (1-alpha)*sketch_rif[x][i][j][l];
    varianza[i][j][l] =
    alpha*varianza[i][j][l] +
    (1-alpha)*(sketch_rif[x][i][j][l]
    -media[i][j][l])*
    (sketch_rif[x][i][j][l]-
    media[i][j][l]);
  }
}
if (DEBUG)
{
  if (distanza < dist_min)
```

```
          {
            dist_min = distanza;
          }
          if (distanza > dist_max)
          {
            dist_max = distanza;
          }
        }
      }
    }
    if (DEBUG)
    {
      sprintf(buffer, "Traccia %s (time bin "
        "%u)\tSoglia %Ld\tDistanza minima = "
        "%.6f\n", file, x, th, dist_min);
      write(1, buffer, strlen(buffer));
      sprintf(buffer, "Traccia %s (time bin "
       "%u)\tSoglia %Ld\tDistanza massima = "
       "%.6f\n", file, x, th, dist_max);
      write(1, buffer, strlen(buffer));
    }
  }
  sprintf(buffer, "Soglia %Ld; calcolo cusum"
    "sull'insieme di riferimento eseguito\n",
    th);
  write(1, buffer, strlen(buffer));
  /*libero memoria allocata per sketch_rif */
  for (x = 0; x < N1; x++)
```

```
{
  for (i = 0; i < D; i++)
  {
    for (j = 0; j < W; j++)
    {
      free(sketch_rif[x][i][j]);
    }
    free(sketch_rif[x][i]);
  }
  free(sketch_rif[x]);
}
free(sketch_rif);

/* FILE DEI TIME BIN ANOMALI */
sprintf(file, "%s/soglia_%Ld/"
        "vertical_anomalous_time_bin.txt",
        argv[3], th);
f1 = fopen(file, "w");
if (f1 == NULL)
{
  fprintf(stderr, "Errore apertura file %s"
  " in scrittura\n", file);
  return 1;
}

inizio_test = N1; /* indice del primo time
          bin da cui si inizia a testare */
for (x = inizio_test; x < NUM_FILE; x++)
```

```
{
  /* INIZIALIZZAZIONE LISTE */
  /* matrice degli attacchi */
  attacks = (double**) malloc
            (sizeof(double*)*D);
  for (d = 0; d < D; d++)
  {
    attacks[d] = (double*) malloc
                 (sizeof(double)*W);
    for (w = 0; w < W; w++)
    {
      attacks[d][w] = 0;
      /* matrice degli aggregati */
      hash_IP[d][w] = NULL;
      /*inizializzo a 0 lo sketch attuale*/
      for (l = 0; l < L; l++)
      {
        sketch[d][w][l] = 0;
      }
    }
    /* vettore degli IP anomali */
    anom_IP[d] = NULL;
  }

  /* FASE DI DETECTION */
  /* si costruisce lo skecth nel time-bin
  attuale e si aggiorna il cusum */
```

```
/* apro il file di ingresso in lettura */
strcpy(file, percorso);
strcat(file, nome_file_in[x]);
strcat(file, estensione);
f_in = fopen(file, "r");
if (f_in == NULL)
{
  fprintf(stderr, "Errore apertura file "
          "%s\n", file);
  return 1;
}
sprintf(buffer, "Inizio test traccia %s "
             "(time bin %u) soglia %Ld\n",
             file, x, th);
write(1, buffer, strlen(buffer));
sprintf(buffer, "Costruzione sketch "
"traccia %s(time bin %u)soglia %Ld...\n",
file, x, th);
write(1, buffer, strlen(buffer));
/* inizializzo la lista SIP_DIP_DP */
pun_SIP_DIP_DP = NULL;
/* leggo il file una riga per volta */
while (fgets(riga,dim_riga, f_in)!=NULL)
{
  if (regexec(&re, riga, (size_t) 0,
              NULL, 0) == 0)
  {
    /* se la riga letta ha il formato
```

```
corretto leggo i vari campi */
campi_letti = 0;
/* inizializzo a 0 dst_port: nel
caso di ICMP non viene assegnato
nessun valore a tale variabile */
dst_port = 0;
riga_temp = riga;
for (;;)
{
  /* leggo un campo della riga */
  campo = strsep(&riga_temp,
                 sep_campi);
  if (campo == NULL)
  {
    /* riga terminata */
    break;
  }
  /* elimino gli spazi e le
  tabulazioni in testa a campo
  (spostando il puntatore al primo
  carattere diverso da spazio o \t)*/
  while ((campo[0] == ' ') ||
         (campo[0] == '\t'))
  {
    campo++;
  }
  if (campo[0] == '\0')
  {
```

```
  /* piu' separatori consecutivi */
  continue;
}
campi_letti++;
unita_misura = 1;
for (i = 0; campo[i] != '\0'; i++)
{
  if ((campo[i] == ' ') ||
      (campo[i] == '\t'))
  {
    campo[i] = '\0';
    if ((campi_letti != 5) &&
        (campi_letti != 6) &&
        (campi_letti != 7))
    {
      /* elimino gli spazi in coda
      se campo non corrisponde al
      numero di pkt,byte o flussi*/
      break;
    }
  }
  if ((campi_letti == 5) ||
      (campi_letti == 6) ||
      (campi_letti == 7))
  {
    /* leggo l'unita' di misura
    (se presente) del numero di
    pkt, byte o flussi */
```

```
        if (campo[i] == 'K')
        {
          unita_misura = 1000;
          break;
        }
        else if (campo[i] == 'M')
        {
          unita_misura = 1000000;
          break;
        }
        else if (campo[i] == 'G')
        {
          unita_misura = 1000000000;
          break;
        }
      }
    }
    switch (campi_letti)
    {
      case 1:
        /* campo Protocol */
        sscanf(campo, "%u", &protocol);
        break;
      case 2:
        /* campo SrcIPAddr:Port */
        sscanf (campo,"%u.%u.%u.%u:%u",
                &src_IP[0], &src_IP[1],
                &src_IP[2], &src_IP[3],
```

```
            &src_port);
    break;
  case 3:
    /* campo DstIPAddr:Port */
    if (protocol == 1)
    {
      /* ICMP: DstPort corrisponde
      a type.code */
      sscanf (campo,
          "%u.%u.%u.%u:%u.%u",
          &dst_IP[0], &dst_IP[1],
          &dst_IP[2], &dst_IP[3],
          &icmp_type, &icmp_code);
    }
    else
    {
      sscanf (campo,
            "%u.%u.%u.%u:%u",
            &dst_IP[0], &dst_IP[1],
            &dst_IP[2], &dst_IP[3],
            &dst_port);
    }
    break;
  case 4:
    /* campo flag TCP */
    cwr = ece = urg = ack = 0;
    psh = rst = syn = fin = 0;
    if (campo[0] == '0')
```

```
{
  /* flag espressi come numero
  esadecimale */
  sscanf (campo, "%x",
          &flag_0x);
  cwr = (flag_0x & 128) >> 7;
  ece = (flag_0x & 64) >> 6;
  urg = (flag_0x & 32) >> 5;
  ack = (flag_0x & 16) >> 4;
  psh = (flag_0x & 8) >> 3;
  rst = (flag_0x & 4) >> 2;
  syn = (flag_0x & 2) >> 1;
  fin = (flag_0x & 1);
}
else
{
  /* flag espressi come
  sequenza di caratteri */
  for(i = 0; i < num_flag; i++)
  {
    switch (campo[i])
    {
      case 'U':
      case 'u':
        urg = 1;
        break;
      case 'A':
      case 'a':
```

```
                ack = 1;
                break;
              case 'P':
              case 'p':
                psh = 1;
                break;
              case 'R':
              case 'r':
                rst = 1;
                break;
              case 'S':
              case 's':
                syn = 1;
                break;
              case 'F':
              case 'f':
                fin = 1;
                break;
            }
          }
        }
        break;
      case 5:
        /* campo Packets */
        sscanf(campo, "%f",
              &dimensione);
        pkt = (unsigned int)
          (dimensione * unita_misura);
```

```
        break;
      case 6:
        /* campo Bytes */
        sscanf(campo, "%f",
              &dimensione);
        byte = (unsigned int)
          (dimensione * unita_misura);
        break;
      case 7:
        /* campo Flows */
        sscanf(campo, "%f",
              &dimensione);
        flow = (unsigned int)
          (dimensione * unita_misura);
        break;
      }
}
/* metrica 0: numero di byte */
metriche[0] = byte/cost_norm[0];
/* metrica 1: numero di porte (TCP o
UDP) destinatarie diverse usate
dall'indirizzo IP dst; in particolare
metriche[1] = 0 se la porta dst e'
gia' stata usata
metriche[1] = 1 se la porta dst non
e' gia' stata usata
tramite l'incremento eseguito nella
costruzione sketch verra' contato il
```

```
        numero di porte dst diverse */
        /* metrica 2: numero di indirizzi IP
        sorgente diversi usati dall'indirizzo
        IP dst; in particolare
        metriche[2] = 0 se l'indirizzo IP src
        e' gia' stata usato
        metriche[2] = 1 se l'indirizzo IP src
        non e' gia' stato usato
        tramite l'incremento eseguito nella
        costruzione sketch verra' contato il
        numero di IP src diversi */
        metriche[1] = metriche[2] = 0;
        for (pApp_SIP_DIP_DP=pun_SIP_DIP_DP;
    pApp_SIP_DIP_DP != NULL;
    pApp_SIP_DIP_DP = pApp_SIP_DIP_DP->pNext)
        {
          /* scorro la lista degli indirizzi
          IP destinatari finche' non trovo
          quello analizzato */
          /* attenzione: i membri della lista
          pApp_SIP_DIP_DP->src_IP[0],...,
          pApp_SIP_DIP_DP->src_IP[3] vengono
          utilizzati come IP dst */
          if ((dst_IP[0] ==
              pApp_SIP_DIP_DP->src_IP[0]) &&
              (dst_IP[1] ==
              pApp_SIP_DIP_DP->src_IP[1]) &&
              (dst_IP[2] ==
```

```
        pApp_SIP_DIP_DP->src_IP[2]) &&
        (dst_IP[3] ==
        pApp_SIP_DIP_DP->src_IP[3]))
    {
      /* esco dal ciclo:
      pApp_SIP_DIP_DP punta
      all'elemento con dst_IP cercato*/
      break;
    }
  }
  if (pApp_SIP_DIP_DP == NULL)
  {
    /* indirizzo IP destinatario non
    presente nella lista:lo inserisco*/
    if ((protocol == 6) ||
    (protocol == 17))
    {
      /* se il protocollo e' TCP o UDP
      inserisco la porta dst nella
      lista (ultimo argomento = 1) */
      inserisci_sip_dip_dp
      (&pun_SIP_DIP_DP, dst_IP[0],
      dst_IP[1], dst_IP[2], dst_IP[3],
      src_IP[0], src_IP[1], src_IP[2],
      src_IP[3], dst_port, 1);
      /* setto a 1 l'indicatore del
      numero di porta destinataria
      diverso (da quelli gia' usati) */
```

```
      metriche[1] = 1/cost_norm[1];
    }
    else
    {
      /* protocollo non TCP o UDP: non
      inserisco la porta dst nella
      lista (ultimo argomento = 0) */
      inserisci_sip_dip_dp
      (&pun_SIP_DIP_DP, dst_IP[0],
      dst_IP[1], dst_IP[2], dst_IP[3],
      src_IP[0], src_IP[1], src_IP[2],
      src_IP[3], dst_port, 0);
    }
    /* setto a 1 l'indicatore
    dell'indirizzo IP sorgente diverso
    (da quelli gia' usati) */
    metriche[2] = 1/cost_norm[2];
  }
  else
  {
    /* indirizzo IP destinatario
    presente nella lista:
    pApp_SIP_DIP_DP punta all'elemento
    che lo contiene */
    if ((protocol == 6) ||
        (protocol == 17))
    {
      /* protocollo TCP o UDP */
```

```
for (pApp_DP =
    pApp_SIP_DIP_DP->lista_DP;
    pApp_DP != NULL;
    pApp_DP = pApp_DP->pNext)
{
  /* scorro la lista delle porte
  destinatarie (per il
  particolare indirizzo IP
  destinatario) finche' non trovo
  quello analizzata */
  if (dst_port == pApp_DP->port)
  {
    /* esco dal ciclo */
    break;
  }
}
if (pApp_DP == NULL)
{
  /* porta destinataria non
  presente nella lista: la
  inserisco */
  inserisci_porta
  (&(pApp_SIP_DIP_DP->lista_DP),
  dst_port);
  /* setto a 1 l'indicatore del
  numero di porta destinataria
  diverso(da quelli gia' usati)*/
  metriche[1] = 1/cost_norm[1];
```

```
  }
}
for (pApp_DIP =
    pApp_SIP_DIP_DP->lista_DIP;
    pApp_DIP != NULL;
    pApp_DIP = pApp_DIP->pNext)
{
  /* scorro la lista degli
  indirizzi IP sorgente
  (per il particolare indirizzo IP
  destinatario) finche' non trovo
  quello analizzato */
  if((src_IP[0]==pApp_DIP->ip[0])&&
    (src_IP[1]==pApp_DIP->ip[1]) &&
    (src_IP[2]==pApp_DIP->ip[2]) &&
    (src_IP[3]==pApp_DIP->ip[3]))
  {
    /* esco dal ciclo */
    break;
  }
}
if (pApp_DIP == NULL)
{
  /* indirizzo IP sorgente non
  presente nella lista:
  lo inserisco */
  inserisci_ip
  (&(pApp_SIP_DIP_DP->lista_DIP),
```

```
      src_IP[0], src_IP[1], src_IP[2],
      src_IP[3]);
      /* setto a 1 l'indicatore
      dell'indirizzo IP sorgente
      diverso (da quelli gia' usati) */
      metriche[2] = 1;
    }
}
/* metrica 3: numero pkt TCP SYN */
/* metrica 4: numero pkt TCP RST */
/* metrica 5: numero pkt TCP FIN */
/* metrica 6: numero pkt ICMP */
/* metrica 7: numero pkt UDP */
metriche[3] = metriche[4] = 0;
metriche[5] = metriche[6] = 0;
metriche[7] = 0;
switch (protocol)
{
  case 6:
    if (syn == 1)
    {
      metriche[3] = pkt/cost_norm[3];
    }
    if (rst == 1)
    {
      metriche[4] = pkt/cost_norm[4];
    }
    if (fin == 1)
```

```
      {
        metriche[5] = pkt/cost_norm[5];
      }
      break;
    case 1:
      metriche[6] = pkt/cost_norm[6];
      break;
    case 17:
      metriche[7] = pkt/cost_norm[7];
      break;
}
/*chiave: indirizzo IP destinatario*/
Mkey = ip2dec(dst_IP);
for (d = 0; d < D; d++)
{
  x0 = Mkey % 16;
  x1 = Mkey >> 16;
  x2 = x0+x1;
  h = (unsigned int)
   (T0[d][x0]^T1[d][x1]^T2[d][x2])%W;
  for (l = 0; l < L; l++)
  {
    /* aggiorno lo sketch */
    sketch[d][h][l] += metriche[l];
  }
  /* inserisco in hash_IP[d][h] la
  lista degli indirizzi IP dst il cui
  hash (funzione d-esima) e' h */
```

```
        for (pApp_IP = hash_IP[d][h];
            pApp_IP != NULL;
            pApp_IP = pApp_IP->pNext)
        {
          /* controllo se l'indirizzo IP e'
          gia' nella lista */
          if ((dst_IP[0]==pApp_IP->ip[0])&&
              (dst_IP[1]==pApp_IP->ip[1])&&
              (dst_IP[2]==pApp_IP->ip[2])&&
              (dst_IP[3]==pApp_IP->ip[3]))
          {
            /* indirizzo IP gia' nella
            lista: esco dal ciclo */
            break;
          }
        }
        if (pApp_IP == NULL)
        {
          /* inserisco l'indirizzo IP se
          non e' gia' nella lista */
          inserisci_ip(&hash_IP[d][h],
                       dst_IP[0], dst_IP[1],
                       dst_IP[2], dst_IP[3]);
        }
      }
    }
  }
/* libero memoria allocata per la lista*/
```

```
libera_sip_dip_dp (&pun_SIP_DIP_DP);
fclose(f_in);
sprintf(buffer, "Costruzione sketch "
  "traccia %s (time bin %u) soglia %Ld "
  "eseguita\n", file, x, th);
write(1, buffer, strlen(buffer));

if (DEBUG)
{
  /* inizializzo distanza min e max */
  dist_min = DBL_MAX;
  dist_max = DBL_MIN;
}
/* calcolo cusum con lo sketch attuale */
sprintf(buffer,"Calcolo distanze traccia"
" %s (time bin %u) soglia %Ld ...\n",
file, x, th);
write(1, buffer, strlen(buffer));
for (i = 0; i < D; i++)
{
  for (j = 0; j < W; j++)
  {
    /* calcolo del cusum */
    cusum[i][j]+=dist_euclid(sketch[i][j],
        sketch_prec[i][j], L) -
        norma(media[i][j], L) -
        c*sqrt(norma(varianza[i][j], L));
    if (cusum[i][j] < 0)
```

```
{
  cusum[i][j] = 0;
}
/* calcolo della distanza */
distanza = fattore_dist*cusum[i][j];
if (distanza > th)
{
  /* anomaly detection */
  attacks[i][j] = 1;
  /* se la distanza supera la soglia
  reinizializzo il cusum a 0 */
  cusum[i][j] = 0;
}
else
{
  /* se la distanza non supera la
  soglia aggiorno sketch_prec,
  media e varianza */
  for (l = 0; l < L; l++)
  {
    sketch_prec[i][j][l] =
                    sketch[i][j][l];
    media[i][j][l] =
          alpha*media[i][j][l] +
          (1-alpha)*sketch[i][j][l];
    varianza[i][j][l] =
    alpha*varianza[i][j][l] +
    (1-alpha)*(sketch[i][j][l]-
```

```
          media[i][j][l])*
          (sketch[i][j][l]-media[i][j][l]);
        }
      }
      if (DEBUG)
      {
        if (distanza < dist_min)
        {
          dist_min = distanza;
        }
        if (distanza > dist_max)
        {
          dist_max = distanza;
        }
      }
    }
  }
  sprintf(buffer,"Calcolo distanze traccia"
  "%s (time bin %u) soglia %Ld eseguito\n",
  file, x, th);
  write(1, buffer, strlen(buffer));
  if (DEBUG)
  {
    sprintf(buffer, "Traccia %s (time bin "
      "%u)\tSoglia %Ld\tDistanza minima = "
      "%.6f\n", file, x, th, dist_min);
    write(1, buffer, strlen(buffer));
    sprintf(buffer, "Traccia %s (time bin "
```

```
  "%u)\tSoglia %Ld\tDistanza massima = "
  "%.6f\n", file, x, th, dist_max);
  write(1, buffer, strlen(buffer));
}

/* IDENTIFICAZIONE DELLE ANOMALIE */
sprintf(buffer,"Identificazione anomalie"
  " traccia %s (time bin %u) soglia %Ld"
  " ...\n", file, x, th);
write(1, buffer, strlen(buffer));
cont = 0;
for (d = 0; d < D; d++)
{
  for (w = 0; w < W; w++)
  {
    if (attacks[d][w] == 1)
    {
      /* incremento count se nella riga
      d-esima c'e' almeno un valore
      "attacks" = 1 */
      cont++;
      break;
    }
  }
  if (cont > h_anom)
  {
    /* consideriamo il time bin anomalo
    se lo sketch ha piu' di h_anom righe
```

```
      con almeno una cella anomala */
      fprintf(f1, "Anomalia rivelata nel "
      "timebin %d (traccia %s)\n",x,file);
      fflush(f1);
      break;
    }
  }

  /* se il time bin e' anomalo, metto in
  anom_IP[d] la lista degli IP dst per i
  quali e' stata rivelata anomalia
  nella riga d-esima dello sketch */
  if (cont > h_anom)
  {
    for (d = 0; d < D; d++)
    {
      for (w = 0; w < W; w++)
      {
        if (attacks[d][w] == 1)
        {
          for (pApp_IP = hash_IP[d][w];
               pApp_IP != NULL;
               pApp_IP = pApp_IP->pNext)
          {
            inserisci_ip(&anom_IP[d],
            pApp_IP->ip[0], pApp_IP->ip[1],
            pApp_IP->ip[2], pApp_IP->ip[3]);
          }
```

```
      }
    }
  }
}

/* libero la memoria allocata per la
lista hash_IP e per la matrice attacks */
for (d = 0; d < D; d++)
{
  for (w = 0; w < W; w++)
  {
    libera_ip(&hash_IP[d][w]);
  }
  free(attacks[d]);
}
free(attacks);

sprintf(file, "%s/soglia_%Ld/"
        "dst_IP_anomali_%s.txt", argv[3],
        th, nome_file_in[x]);
f2 = fopen(file, "w");
if (f2 == NULL)
{
  fprintf(stderr, "Errore apertura file "
          "%s in scrittura\n", file);
  return 1;
}
for (pApp_IP=anom_IP[0];pApp_IP != NULL;
```

```
      pApp_IP = pApp_IP->pNext)
  {
    /* scorro la lista degli IP anomali
    relativi alla riga 0 dello sketch e
    cerco ogni IP nelle liste relative a
    ciascuna delle righe successive */
    for (d = 1; d < D; d++)
    {
      for (pApp_IP_2 = anom_IP[d];
          pApp_IP_2 != NULL;
          pApp_IP_2 = pApp_IP_2->pNext)
      {
        if ((pApp_IP_2->ip[0] ==
              pApp_IP->ip[0]) &&
            (pApp_IP_2->ip[1] ==
              pApp_IP->ip[1]) &&
            (pApp_IP_2->ip[2] ==
              pApp_IP->ip[2]) &&
            (pApp_IP_2->ip[3] ==
              pApp_IP->ip[3]))
        {
          break;
        }
      }
      if (pApp_IP_2 == NULL)
      {
        /* indirizzo IP non trovato nella
        lista anom_IP[d] */
```

```
        break;
      }
    }
    if (pApp_IP_2 != NULL)
    {
      /* indirizzo IP presente in tutte le
      liste anom_IP relative a tutte le
      righe dello sketch */
      /* stampo sul file la lista degli
      IP dst che hanno provocato anomalia
      in tutte le righe dello sketch */
      fprintf(f2, "%u.%u.%u.%u\n",
          pApp_IP->ip[0], pApp_IP->ip[1],
          pApp_IP->ip[2], pApp_IP->ip[3]);
    }
  }
  for (d = 0; d < D; d++)
  {
    libera_ip(&anom_IP[d]);
  }
  fclose(f2);
  sprintf(buffer,"Identificazione anomalie"
    " traccia %s (time bin %u) soglia %Ld "
    "eseguita\n", file, x, th);
  write(1, buffer, strlen(buffer));
  sprintf(buffer, "Fine test traccia %s "
  "(time bin %u) soglia %Ld\n", file,x,th);
  write(1, buffer, strlen(buffer));
```

```
}
fclose(f1);

/* DEALLOCAZIONE VARIABILI ALLOCATE */
for (i = 0; i < D; i++)
{
  free(T0[i]);
  free(T1[i]);
  free(T2[i]);
}
free(T0); free(T1); free(T2);

for (i = 0; i < D; i++)
{
  for (j = 0; j < W; j++)
  {
    free(sketch[i][j]);
    free(sketch_prec[i][j]);
    free(media[i][j]);
    free(varianza[i][j]);
  }
  free(sketch[i]);
  free(sketch_prec[i]);
  free(media[i]);
  free(varianza[i]);
  free(cusum[i]);
}
free(sketch);
```

```
  free(sketch_prec);
  free(media);
  free(varianza);
  free(cusum);

  /* libero memoria allocata dalla regcomp */
  regfree(&re);

  free(riga);

  for (x = 0; x < NUM_FILE; x++)
  {
    free(nome_file_in[x]);
  }
  free(nome_file_in);

  return 0;
}
```

Bibliografia

[1] MAWILab.
http://www.fukuda-lab.org/mawilab/.

[2] Nfcapd, sintassi di esecuzione.
http://www.linuxcertif.com/man/1/nfcapd/.

[3] Nfdump.
http://http://sourceforge.net/projects/nfdump/.

[4] Nfdump, documentazione.
http://nfdump.sourceforge.net/.

[5] Nfdump, sintassi di esecuzione.
http://www.linuxcertif.com/man/1/nfdump/.

[6] Softflowd.
https://code.google.com/p/softflowd.

[7] Softflowd, sintassi di esecuzione.
http://www.linuxcertif.com/man/1/nfdump/.

[8] Christian Callegari. Intrusion Detection System. Slide delle lezioni del corso di Sicurezza nelle Reti, Università di Pisa, Ingegneria delle Telecomunicazioni. 2007.

[9] Christian Callegari, Stefano Giordano, Michele Pagano, and Teresa Pepe. Forecasting the Distribution of Network Traffic for Anomlay Detection. 2011.

[10] Graham Cormode and S. Muthukrishnan. An Improved Data Stream Summary: The Count-Min Sketch and its Applications. *Journal of Algorithms*, 55(2):58–75, 2005.

[11] Dorothy E. Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, February 1987.

[12] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking. *ACM CoNEXT*, 2010.

[13] Andreas Kind, Marc Ph. Stoecklin, and Xenofontas Dimitropoulos. Histogram-Based Traffic Anomaly Detection. *IEEE Transactions on Network and Service Management*, 6(2), June 2009.

[14] Daniel Nikovski and Ankur Jain. Fast Adaptive Algorithms for Abrupt Change Detection. *Machine Learning Journal*, 2009.

[15] Osman Salem, Sandrine Vaton, and Annie Gravey. A scalable, efficient and informative approach for anomaly-based Intrusion Detection Systems: theory and practice. *International Journal of Network Management*, 2010.

[16] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Statistical Change Detection for Multi-Dimensional Data. 2007.

[17] Masashi Sugiyama and Yoshinobu Kawahara. Change-Point Detection in Time-Series Data by Direct Density-Ratio Estimation. 2009.

[18] Gabor J. Szekely and Maria L. Rizzo. Testing for Equal Distributions in High Dimension. 2004.

[19] Michele Del Vigna. Sviluppo di algoritmi statistici per anomaly-based Intrusion Detection Systems. Tesi di laurea, Università di Pisa, Ingegneria delle Telecomunicazioni. 2011.