UNIVERSITA' DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di laurea in Ingegneria Informatica

STUDIO DI FLUSSI DI TRAFFICO IN RETE TRAMITE QUANTILI DELLA TRASFORMATA WAVELET.

Relatore: Ch.mo Prof. Claudio Narduzzi

Correlatore: Ing. Giada Giorgi

Laureando: Stefano Dindo

1 Luglio 2008

ANNO ACCADEMICO 2007 - 2008

Sommario

Lo scopo di questa tesi è la realizzazione di un tool per il monitoraggio del traffico di rete la cui analisi viene svolta utilizzando la Discrete Wavelet Transform ed un'analisi statistica basata sui quantili. La tesi inizia con accenni teorici sulle wavelet per poi illustrare mediante diagrammi di flusso gli algoritmi impiegati per l'implementazione del tool.
La tesi si conclude con l'analisi di attività di misura effettuate utilizzando il tool realizzato. La prima traccia di traffico che viene analizzata è riferita ad un flusso di traffico controllato cioè è stata realizzzata monitorando una rete in cui tutte le attività di accesso ad internet erano tenute sotto controllo. La seconda traccia invece mostra l'andamento del flusso di traffico di una intera giornata della rete realativa ad un piano del nostro dipartimento.

Indice

Elenco delle figure

Capitolo 1

Introduzione

In questi ultimi anni internet, grazie alla sua diffusione, ha rivoluzionato il modo di comunicare delle persone, quindi è divenuta di fondamentale importanza la gestione delle reti da parte delle aziende, delle istituzioni e delle comunità. Infatti, la rete internet porta con se molte insidie quali worm, virus, spam e molte altre che compromettono il corretto funzionamento dell'infrastruttura di rete. Quindi per garantire il corretto funzionamento delle reti si sono sviluppate varie tecniche di misura e monitoraggio del flusso di traffico.

Ci sono quattro ragioni principali del perchè le misure di traffico sono utili:

- **Reti non affidabili:** spesso un malfunzionamento di un singolo elemento della rete potrebbe disturbare le operazioni dell'intera infrastruttura oppure ridurne le performance. Esempi di questo scenario possono essere la dimensione del pacchetto non conforme, indirizzo di rete non corretto oppure attacchi alla rete.

- **Attività di debugging dei protocolli o applicazioni:** spesso gli sviluppatori vogliono testare le migliorie introdotte nelle nuove versioni di protocolli. Infatti, le misure di traffico di rete forniscono una valutazione sulla correttezza dei nuovi protocolli o applicazioni e se necessario mostrano la compatibilità con i precedenti formati.

- **Caratterizzazione del carico di rete:** questo rappresenta il motivo principale dell'impiego di sistemi di monitoraggio in quanto permettono di caratterizzare il carico di traffico di rete, estraendo informazioni sul flusso e

la tipologia di traffico; e di identificare la capacità della rete in termini di banda.

- **Valutazione delle performance:** i tool di monitoraggio consento di valutare quanto la rete sia performante per l'impiego per cui è stata sviluppata.

In ambito scientifico si sono sviluppate varie metodologie di misura che possono essere classificate come:

- **Strumenti Hardware per il monitoraggio:** questo approccio di misura sfrutta speciali strumenti sviluppati appositamente per particolari attività di monitoraggio. Solitamente questi strumenti sono molto costosi, dipendono dal numero di interfacce di rete, dal tipo di connessioni, dalla capacità di memorizzazione e dal tipo di protocolli che possono analizzare.

- **Tool software per il monitoraggio:** utilizzano moduli a livello kernel che apportano delle modifiche alle interfacce di rete dei personal computer dando loro la capacità di acquisire pacchetti. Uno dei più usati è il TCPDUMP che è un tool a livello utente per la cattura dei pacchetti TCP/IP resa possibile dal Berkeley Packet Filter. L'approccio software è una soluzione meno costosa rispetto all'approccio hardware ma offre prestazioni inferiori rispetto a strumenti dedicati per l'analisi del traffico.

- **Approccio di misura Passivo:** prevede di osservare e memorizzare il traffico di pacchetti che transitano nella rete senza generare traffico su di essa.

- **Approccio di misura Attivo:** prevede di generare pacchetti dallo strumento di misura per valutare le caratteristiche della rete. Un esempio di questo approccio è l'utility *ping* per valutare la latenza della rete verso una particolare destinazione di internet.

- **Analisi di traffico Offline:** utilizza strumenti che prevedono di acquisire e memorizzare il flusso di traffico per analizzarlo in un secondo momento.

Lo scopo di questa tesi è quello di realizzare un tool software capace di analizzare il traffico di rete online, fornendo così uno strumento di misura in grado di fornire delle valutazioni sullo stato della rete in tempo reale in modo da avviare delle contromisure, in caso di problemi, nel minor tempo possibile.

Capitolo 2

Teoria delle Wavelet

2.1 Introduzione alle wavelet

Le Wavelet sono funzioni matematiche che permettono di scomporre i dati nelle diverse componenti in frequenza e di studiare ciascuna componente con una risoluzione adatta alla scala.

Queste funzioni presentano dei vantaggi rispetto ai tradizionali metodi di Fourier nell'analizzare situazioni fisiche in cui il segnale presenta discontinuità e spike. Poichè i seni e coseni utilizzati nella trasformata di Fourier non consentono di approssimare adeguatamente segnali discontinui, si ricorre alle wavelet che ben si adattano a tali approssimazioni in quanto, nell'analisi tramite wavelet, la scala che utilizziamo per elaborare il segnale gioca un ruolo particolare. Se analizziamo un segnale in una 'finestra' grande, noteremo le sue caratteristiche generali, se lo analizziamo in una 'finestra' piccola noteremo le sue caratteristiche con maggiore attenzione alle piccole variazioni. L'analisi wavelet permette di vedere entrambe.

La procedura di analisi tramite Wavelet è quella di utlizzare una funzione wavelet prototipo, detta *Mother Wavelet*.

L'analisi temporale è compiuta utilizzando una versione contratta e ad alta frequenza della wavelet prototipo, mentre l'analisi in bassa frequenza è compiuta con una versione contratta di essa. Poichè il segnale originale può essere rappresentato in termini della espansione wavelet (usando i coefficienti in una combinazione lineare delle funzioni wavelet), si possono utilizzare i coefficienti come rappresentazione del segnale originario. Poichè molti di tali coefficienti possono essere nulli, la wavelet è molto utilizzata per la compressione dei dati.

La wavelet è largamente impiegata in settori quali: astronomia, acustica, elaborazione di segnali e immagini , ottica, previsione dei terremoti e molti altri.

2.2 Breve storia delle wavelet

Le Wavelet hanno generato negli ultimi anni un forte interesse sia nel campo teorico che in quello applicativo. Il termine wavelet o ondelette è stato coniato verso la fine degli anni '70 da alcuni ricercatori francesi, quali Morlet, Arens, Fourgeau, Giard e Grossman.

L'esistenza delle wavelet come funzioni è conosciuta fin dagli inizi del secolo e molte delle idee presenti nella wavelet derivano ad esempio dallo studio degli stati coerenti nella meccanica quantistica e dagli operatori di Calderon-Zygmund nella matematica. Come anticipato precedentemente, nella matematica astratta, è stato mostrato che le tecniche basate sulle serie di Fourier e trasformate di Fourier non sono del tutto adeguate per molti problemi, mentre le tecniche cosidette di Littlewood-Paley spesso le sostituiscono nel modo migliore. Queste tecniche furono inizialmente sviluppate negli anni '30 per comprendere le proprietà di sensibilità della serie di Fourier. Negli anni successivi queste si svilupparono in potenti strumenti utili per lo studio delle equazioni differenziali e integrali. Infatti, si scoprì che esse rientrano nella teoria di Calderon-Zygmund, relativa all'analisi armonica. Uno degli approcci standard, valido anche per problemi di vario tipo che si scostano dalla teoria citata, è quello di dividere un fenomeno complicato in molte singole parti, ognuna delle quali viene studiata separatamente.

Negli anni '70, per stabilire se una funzione avesse una decomposizione in somma di funzioni più semplici negli spazi di Handy, viene usata la formula di Calderon la quale non è altro che un esempio di trasformata wavelet continua.

Nei primi anni '80, Stromberg scoprì le wavelet ortogonali, cercando di comprendere ulteriormente gli spazi di Handy. Indipendentemente da questi sviluppi nell'analisi armonica, Alex Grossman e Jean Morlet studiarono la trasformata wavelet nella sua forma continua.

In seguito, attorno al 1985, diversi gruppi, in particolare quello di Yves Mayer, capirono che le teorie di Calderon e quelle relative alla meccanica quantistica avevano una discreta analogia e poterono fornire una visione unificata dei risultati ottenuti nell'analisi armonica. Nello stesso tempo si cominciò a comprendere che queste tecniche potevano essere validi sostituti delle serie di Fourier nelle applicazioni numeriche.

P. Lemarié e Y. Mayer costruirono nuove espressioni ortogonali delle wavelet. Con la nozione di analisi in multirisoluzione, introdotta da S. Mallat e Mayer, fu sviluppato un ambito sistemistico per comprendere queste espansioni, e si fornì inoltre un legame con il filtraggio in quadratura. Ben presto Ingrid Daubechies arrivò ad una costruzione di wavelet non nulle solo in un intervallo finito e con una regolarità arbitrariamente elevata , ma fissa. Con ciò si arriva agli ultimi sviluppi della teoria delle wavelet, a cui stanno contribuendo tuttora numerosi autori.

2.3 Dalla Trasformata di Fourier alla Trasformata Wavelet Discreta

2.3.1 Analisi Tempo-Frequenza

La maggior parte dei segnali reali fisici è di tipo non stazionario, cioè con caratteristiche variabili nel tempo. Un esempio che rappresenta molto bene un segnale non stazionario è il seguente:

$$x(t) = \sum_{i=0}^{N} A_i e^{-a_i(t-t_i)} \cos(2\pi f_i t + \varphi_i) u(t-t_i) \tag{2.1}$$

in quanto è una combinazione di sinusoidi con diversi parametri (ampiezza A_i, tempi di inizio t_i, frequenze di dominio f_i, fasi iniziali φ_i e coefficienti di smorzamento a_i). Se su questo segnale si effettua la trasformata di Fourier, si ottiene una funzione X(f) il cui modulo fornisce alcune informazioni sulla presenza delle componenti armoniche f_i e delle rispettive ampiezze A_i. Uno dei problemi della trasformata di Fourier è che le informazioni relative agli altri parametri sono inglobati nella fase di X(f); quindi le informazioni, soprattutto nel caso in cui il segnale abbia molte componenti, sono confuse al punto da risultare illeggibili. In altre parole, la trasformata di Fourier evidenzia la presenza delle componenti armoniche ma non permette di ricavare facilmente informazioni su quando e come tali frequenze siano effettivamente presenti.

Il motivo di tutto ciò è intrinseco nella definizione della trasformata:

$$X(f) = \int_{-\infty}^{-\infty} x(t) e^{-j2\pi ft} dt = \left\langle e^{-j2\pi ft}, x(t) \right\rangle \tag{2.2}$$

da cui si nota che X(f) rappresenta il prodotto scalare fra il segnale e una funzione sinusoidale complessa di durata temporale infinita, la quale è perfettamente locale

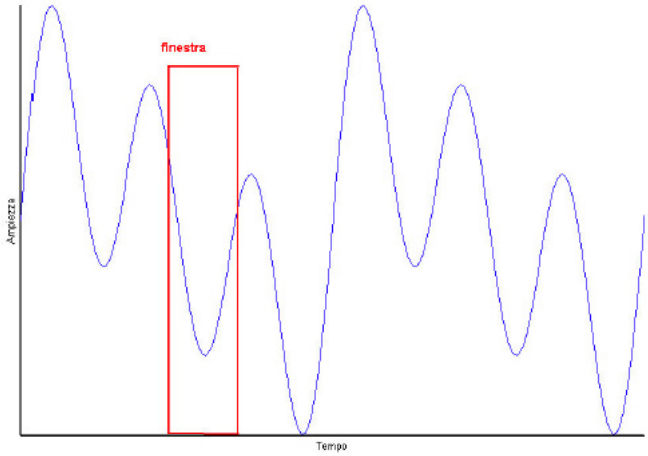


Figura 2.1: Segnale con Finestra

in frequenza ma assolutamente globale nel tempo. In altri termini, questa trasformazione è adatta a segnali stazionari perchè permette di evidenziare nettamente fenomeni che avvengono nel dominio della frequenza ma non discrimina eventi che avvengono nel dominio del tempo. Per i segnali non stazionari occorre quindi inserire nella trasformazione una dipendeza dal tempo; il modo più immediato consiste nel rendere locale la (2.1) non operando su tutto il supporto del sengale x(t) ma su porzioni di esso, ottenute moltiplicandolo per una finestra che trasla nel tempo (vedi Figura 2.1). Nasce così la *trasformata di Fourier a breve termine* o Short time Fourier Transform (STFT), ideata da Dennis Bobar nel 1946:

$$STFT_x(\tau, f) = \int_{-\infty}^{+\infty} x(t)g(t-\tau)e^{-j2\pi ft}dt = \left\langle g(t-\tau)e^{j2\pi ft}, x(t)\right\rangle \qquad (2.3)$$

Con questa nuova trasformazione si evidenzia lo spettro del segnale all'interno della finestra temporale definita dalla funzione $g(t-\tau)$ ottenendo quindi un'informazione sul suo contenuto armonico in un intorno dell'istante τ. Questo approccio, pur essendo molto semplice, presenta un difetto non trascurabile: moltiplicare nel dominio del tempo il segnale x(t) con la funzione finestra g(t) equivale a effettuare la convoluzione dei loro spettri X(f) e G(f) nel dominio della frequenza; la STFT fornisce quindi lo spettro del segnale alterato dalla presenza della finestra. Il vero problema è che tale difetto non è eliminabile; se si vuole ridurre l'effetto di G(f) occorre ridurre la banda Δf, ma ciò implica un aumento dell'ampiezza temporale di Δt e quindi una diminuzione della precisione nel dominio del tempo, nel senso che due eventi separati da meno di Δt non sono più discriminabili; viceversa, se

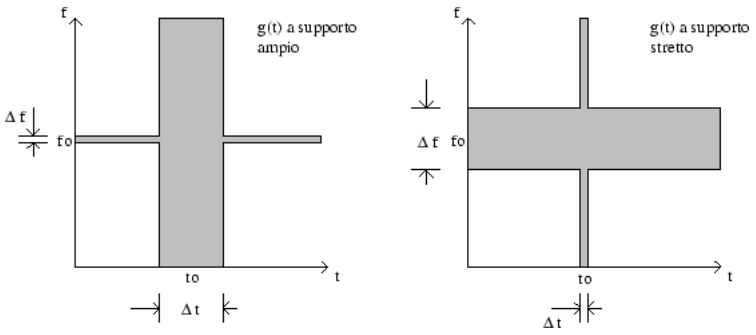


Figura 2.2: STFT di una sinusoide con un impulso

si riduce la finestra Δt necessariamente aumenta la banda Δf e quindi si ha una diminuzione della precisione in frequenza. La figura riassume il concetto appena esposto, evidenziando la STFT applicata a un segnale composto dalla somma di una sinuisoide e di un impulso. Alla luce delle precedenti considerazioni, la conclusione è immediata:

- fissata la funzone finestra g(t) vengono automaticamente definite le indeterminazioni Δt e Δf rispettivamente nel dominio del tempo e della frequenza;
- due eventi separati per meno di Δt o Δf non sono più discriminati dalla STFT nel rispettivo dominio;
- le due indeterminazioni sono inversamente proporzionali, quindi la risoluzione temporale può essere migliorata solo a scapito della risoluzione in frequenza, e viceversa;
- le due indeterminazioni sono fisse e questo implica che le risoluzioni relative $\frac{\Delta t}{t}$ e $\frac{\Delta f}{f}$ risultano variabili.

L'ultima osservazione è quella fondamentale: in molte applicazioni non è possibile permettersi di localizzare fenomeni lenti e fenomeni veloci con la stessa indeterminazione, nel senso che identificare una frequenza di 100MHz con precisione Δf = 1kHz è molto diverso dell'identificare 100kHz con la stessa precisione.

2.3.2 Da STFT a CWT

Nel paragrafo precedente si è notato che la STFT è una tecnica di analisi a risoluzione fissa e si è appurato come questo possa costituire una limitazione. Si è anche osservato come la causa sia intrinseca nella presenza della funzione finestra g(t) a

Figura 2.3: La wavelet db 10

supporto e a banda limitata, che viene modulata con $e^{j2\pi ft}$ e moltiplicata scalarmente con il segnale x(t). Per ottenere un'analisi a risoluzione variabile occorre fare in modo che le risoluzioni relative a $\frac{\Delta t}{t}$ e $\frac{\Delta f}{f}$ risultino costanti e questo richiede che all'aumentare della frequenza f aumenti in modo proporzionale alla banda Δf. A tale riguardo viene in aiuto una proprietà fondamentale della trasformata di Fourier: comprimendo nel tempo una funzione si ottiene una espansione in frequenza del suo spettro, e viceversa:

$$F\left\{x\left(\frac{t}{a}\right)\right\} = |a|X(af) \tag{2.4}$$

Da questa considerazione nasce l'idea di sostituire l'operazione di *modulazione* con l'operazione di *scalamento*, ovvero anzichè moltiplicare il segnale per la finestra g(t) ad ampiezza temporale costante ed F-trasformare, si esegue direttamente il prodotto scalare con scalamenti e traslazioni di un unico prototipo. Quello che si ottiene prende il nome di *trasformata wavelet continua* (CWT):

$$CWT_x(a,b) = \int_{-\infty}^{+\infty} x(t)\varphi_{ab}(t)dt = \langle \varphi_{ab}(t), x(t) \rangle \tag{2.5}$$

con

$$\varphi_{ab} = \frac{1}{\sqrt{|a|}}\varphi\left(\frac{t-b}{a}\right) \tag{2.6}$$

il prodotto $\varphi(t)$ prende il nome di wavelet madre; a è il parametro di scalamento, b è il parametro di traslazione. La denominazione wavelet deriva dal fatto che, graficamente , il prototipo è una funzione che oscilla e si smorza come una piccola onda. Un esempio classico di wavelet è riportato in figura 2.3.

Si osservi subito un particolare fondamentale; valutando la formula di Fourier

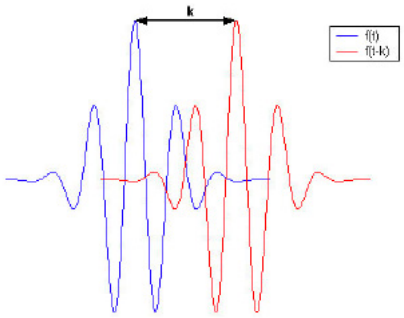


Figura 2.4: Esempio di traslazione di una funzione

della wavelet, a meno del parametro di traslazione b, si ottiene:

$$F\left\{\frac{1}{\sqrt{|a|}}\left(\frac{t-b}{a}\right)\right\} = \psi(af) \tag{2.7}$$

essendo

$$\psi(f) = F\{\varphi(t)\} \tag{2.8}$$

Se $\varphi(t)$ ha banda Δf centrata in f_0 allora $\varphi(\frac{t}{a})$ ha banda $\left(\frac{\Delta f}{a}\right)$ centrata in $\frac{f_0}{a}$: la banda è in termini realtivi costante.

Il concetto di traslazione (o shifting) è invece espresso nella figura 2.4. Come interpretare l'informazione fornita da una rappresentazione tempo-scala? Occorre svincolarsi dal concetto di frequenza, poichè confrontando la definizione di trasformata di Fourier con la definizione di trasformata wavelet continua, è immediato notare che il prodotto scalare del segnale x(t) viene ora eseguito con una funzione non periodica e limitata nel tempo: il concetto di frequenza dell'armonica $e^{j2\pi ft}$ viene sostituito con il concetto di scala della wavelet $\varphi_{ab}(t)$. L'analogia è tuttavia immediata: valori piccoli di a significano wavelet compresse nel tempo, quindi contenenti armoniche ad alta frequenza; effettuare il prodotto scalare con esse implica ottenere proiezioni che portano informazioni sui dettagli del segnale, cioè su fenomeni rapidamente variabili; d'altra parte, valori grandi di a comportano wavelets lentamente variabili, con banda stretta, che invece colgono il comportamento del segnale a lungo termine. Queste fondamentali proprietà sono note come *localizzazione temporale* (più la wavelet è concentrata nel tempo, migliore è la risoluzione in questo dominio) e come *localizzazione spettrale* (più la wavelet è concentrata in frequenza, migliore è la risoluzione in questo dominio). Tra le altre proprietà della

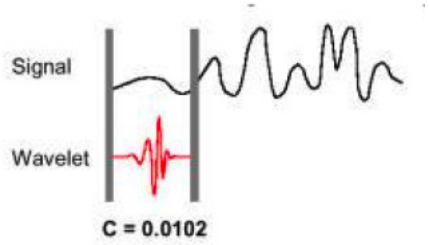


Figura 2.5: Esempio correlazione

trasformata wavelet è opportuno ricordare la linearità, l'invarainza per traslazioni temporali e l'invarianza per scalamento. Fondamentale è tuttavia la proprietà di invertibilità: date due wavelet $\varphi_1(t)$ $\varphi_2(t)$ con particolari caratteristiche, sulle quali non ci addentreremo, se si valuta la CWT rispetto a $\varphi_1(t)$

$$CWT_{x,\varphi_1}(a,b) = \int_{-\infty}^{+\infty} x(t)\varphi_{1,ab}(t)dt \tag{2.9}$$

allora il segnale è ricostruibile attraverso la $\varphi_2(t)$ e si ha:

$$x(t) = \frac{1}{C_\varphi}\int_0^{+\infty}\int_{-\infty}^{+\infty} CWT_{x,\varphi_1}\varphi_{2,ab}(t)\frac{dadb}{a^2} \tag{2.10}$$

dove

$$C_\varphi = \int_{-\infty}^{+\infty}\frac{|\psi_1(\omega)||\psi_2(\omega)|}{|\omega|}d\omega \tag{2.11}$$

Riepilogando, l'algoritmo da utilizzare per calcolare la CWT di un segnale è il seguente:

1. scegli una wavelet e confrontala con una sezione iniziale del segnale originale che vogliamo analizzare
2. Calcolare (un numero C) la correlazione tra la wavelet e la sezione del segnale. Più grande è C, più sono simili wavelet e sezione del segnale. Da notare che il risultato dipenderà dalla forma della wavelet scelta. (Vedi Figura 2.5).
3. Traslare la wavelet lungo l'asse del tempo e ripetere i punti 1 e 2 fino a coprire l'intero segnale (Vedi Figura 2.6).
4. Scalare la wavelet e ripetere i passi da 1 a 3 (Vedi Figura 2.7)
5. Ripetere i passi 1-4 per tutte le scale.

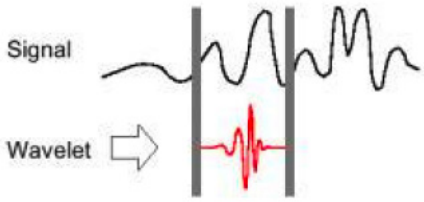


Figura 2.6: Operazione di traslazione della wavelet

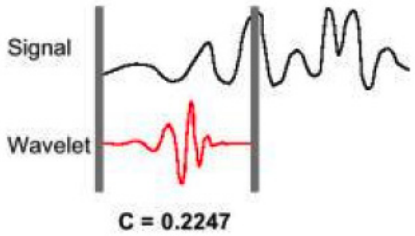


Figura 2.7: Scalatura della wavelet

Una volta eseguite le operazioni sopra elencate, si otterranno i coefficienti prodotti a differenti scale, dalle diverse sezioni del segnale. I coefficienti non sono altro che il risultato di una regressione del segnale originale, ottenuta mediante le wavelet.

2.3.3 Da CWT a DWT

La definizione di trasformata wavelet continua è utilizzabile nei casi in cui si desideri una valutazione analitica della CWT di un segnale. Tuttavia nella maggior parte dei casi pratici, il segnale x(t) non solo non è noto analiticamente (per esempio, quando è fornito da un sensore) ma risulta campionato, cioè è conosciuto esclusivamente in precisi istanti di tempo; se si considera inoltre il desiderio di una valutazione numerica della trasformata, allora essa dovrà essere valutata solo con un numero finito di valori delle variabili a e b. La scelta non accurata dei parametri a e b, ovviamente può determinare da una parte una rappresentazione ridondante, ossia il numero di campioni della trasformata risulta molto più grande di quello del segnale, dall'altra una rappresentazione non completa, cioè il numero di campioni non è sufficiente per la ricostruzione. La situazione ideale è la rappresentazione ortogonale, ovvero il segnale e la sua trasformata hanno lo stesso numero di campioni. Si considerino le seguenti osservazioni:

1. al variare della scala a le wavelet hanno banda relativa costante e costante è l'indeterminazione $\frac{\Delta f}{f}$ sulla frequenza. È quindi ragionevole effettuare una discretizzazione logaritmica di questo parametro.

2. quando la scala è piccola le wavelet hanno supporto temporale corto, quindi è logico che il passo di traslazione Δb sia piccolo per garantire la sufficiente copertura dell'asse-tempi da parte della serie di ondine.

3. viceversa, quando la scala è grande le wavelet sono lunghe, per cui il passo di traslazione può essere più grande.

Alla luce di tutto ciò, la discretizzazione più ragionevole risulta essere:

$$\begin{cases} a = a_0^m & m \in \mathbf{Z} a_0 > 1 \\ b = nab_0 & n \in \mathbf{Z} b_0 > 1 \end{cases} \tag{2.12}$$

Il caso più comunemente utilizzato è quello diadico, con $a_0 = 2$ che definisce la famiglia di wavelets

$$\varphi_m n(t) = \frac{1}{\sqrt{a}} \varphi \left(\frac{t-b}{a} \right) |_{a=2^m, b=2^m n b_0} = 2^{\frac{m}{2}} \varphi(2^{-m} t - n b_0) \tag{2.13}$$

A partire da questa forma di CWT a parametri discreti, una successiva quantizzazione nel dominio del tempo t permette di introdurre la versione definitiva completamente discretizzata nota come DWT (discrete wavelet transform).

2.4 La rappresentazione wavelet

In questo paragrafo verranno illustrati i concetti della decomposizione multirisoluzione di un segnale monodimensionale. Prima di entrare nella teoria delle wavelet è bene introdurre le notazioni che verranno utilizzate in seguito.

2.4.1 Notazione

$\mathbf{Z}$ e $\mathbf{R}$ indicano rispettivamente l'insieme dei numeri interi e dei numeri reali. Indichiamo con $\mathbf{L}^2(\mathbf{R})$ lo spazio delle funzioni misurabili quadrato integrabili e unidimensionali. Per $f(x) \in \mathbf{L}^2(\mathbf{R})$ e $g(x) \in \mathbf{L}^2(\mathbf{R})$ il prodotto interno di f(x) con g(x) viene indicato come:

$$\langle g(u), f(u) \rangle = \int_{-\infty}^{+\infty} g(u) f(u) du \tag{2.14}$$

La norma di $f(x) \in \mathbf{L}^2(\mathbf{R})$ viene indicata come:

$$||f||^2 = \int_{-\infty}^{+\infty} |f(u)|^2 du \tag{2.15}$$

La convoluzione tra due funzioni $f(x) \in \mathbf{L}^2(\mathbf{R})$ e $g(x) \in \mathbf{L}^2(\mathbf{R})$ la indichiamo come:

$$f(x) * g(x) = (f(u) * g(u))(x) = \int_{-\infty}^{+\infty} f(u)g(x-u)du \tag{2.16}$$

La trasformata di Fourier di $f(x) \in \mathbf{L}^2(\mathbf{R})$ viene indicata come:

$$\hat{f}(\omega) = \int_{-\infty}^{+\infty} f(x)e^{-i\omega x}dx \tag{2.17}$$

$\mathbf{l}^2(\mathbf{Z})$ è uno spazio vettoriale di sequenze di quadrati sommabili.

$$\mathbf{l}^2(\mathbf{Z}) = \left\{ (\alpha_i)_{i\in\mathbf{Z}} : \sum_{-\infty}^{+\infty} |\alpha_i|^2 < \infty \right\} \tag{2.18}$$

2.4.2 Trasformata multirisoluzione

Sia A_{2^j} l'operatore che approssima un segnale con una risoluzione 2^j. Supponiamo che il segnale originale f(x) sia misurabile e abbia un'energia finita: $f(x) \in \mathbf{L}^2(\mathbf{R})$. Di seguito caratterizeremo A_{2^j} mediante le proprietà intuitive che ci aspettiamo da un tale operatore di approssimazione. Definiremo inizialmente queste proprietà in modo discorsivo per poi riassumerle mediante formule matematiche.

1. A_{2^j} è un operatore lineare. Se $A_{2^j}f(x)$ è l'approssimazione di qualche funzione f(x) alla risoluzione 2^j, allora $A_{2^j}f(x)$ non viene modificata se noi la approssimiamo nuovamente alla risoluzione 2^j. Questo principio mostra che $A_{2^j} \circ A_{2^j} = A_{2^j}$. Ciò significa che l'operatore A_{2^j} è un'operatore di proiezione su un particolare spazio vettoriale $V_{2^j} \subset \mathbf{L}^2(\mathbf{R})$. Quindi lo spazio vettoriale può essere interpretato come un insieme di tutte le possibili approssimazioni alla risoluzione 2^j di funzioni in $\mathbf{L}^2(\mathbf{R})$.

2. Fra tutte le funzioni di approssimazione alla risoluzione 2^j, A_{2^j}f(x) è la funzione che è la più simile a f(x).

$$\forall g(x) \in \mathbf{V}_{2^j}, ||g(x) - f(x)|| \geq ||A_{2^j} - f(x)|| \tag{2.19}$$

Quindi, l'operatore A_{2^j} è una proiezione ortogonale nello spazio vettoriale V_{2^j}.

3. L'approssimazione di un segnale alla risoluzione 2^{j+1} contiene tutte le informazioni necessarie per calcolare lo stesso segnale alla risoluzione più piccola 2^j. Questa è la *proprietà di causalità*. Poichè A_{2^j} è un operatore di proiezione su V_{2^j} tale principio è equivalente a:

$$\forall j \in \mathbf{Z}, \mathbf{V}_{2^j} \subset \mathbf{V}_{2^{j+1}} \tag{2.20}$$

4. Un'operazione di approssimazione è simile a tutte le risoluzioni. Gli spazi delle funzioni approssimate dovrebbero essere derivati riducendo il rapporto dei valori di risoluzione di ogni funzione di approssimazione.

$$\forall j \in \mathbf{Z}, \quad f(x) \in \mathbf{V}_{2^j} \Leftrightarrow f(2x) \in \mathbf{V}_{2^{j+1}} \tag{2.21}$$

5. L'approssimazione A_{2^j}f(x) di un segnale f(x) può essere caratterizzata da 2^j campioni per unità di lunghezza. Quando f(x) viene traslato di una lunghezza proporzionale a 2^{-j}, A_{2^j}f(x) è traslato della stessa quantità ed è caratterizzata dagli stessi campioni che la caratterizzavano prima della traslazione. Come conseguenza della 2.21 è sufficiente esprimere il principio 5) per la risoluzione j=0. Le traslazioni matematiche sono le seguenti:

   - Caratterizzazione discreta: Esiste un isomorfismo $\mathbf{I}$ da $\mathbf{V_1}$ a $\mathbf{l}^2(\mathbf{Z})$
   - Traslazione di un'approssimazione:

$$\forall k \in \mathbf{Z}, \quad A_1 f_k(x) = A_l f(x-k) \tag{2.22}$$

     dove $f_k(x) = f(x-k)$

   - Traslazione di campioni

$$\mathbf{l}(A_l f(x)) = (\alpha_i)_{i \in \mathbf{Z}} \Leftrightarrow \mathbf{l}(A_l f_k(x)) = (\alpha_{i-k})_{i \in \mathbf{Z}} \tag{2.23}$$

6. Quando calcoliamo un'approssimazione di f(x) alla risoluzione 2^j, alcune informazioni riguardanti f(x) vengono perse. Tuttavia, al crescere della risoluzione verso $+\infty$ l'approssimazione del segnale dovrebbe convergere verso il segnale originale. Conseguentemente, quando la risoluzione decrese verso

lo zero, il segnale approssimato contiene sempre meno informazionei e converge a zero. Poichè il segnale approssimato alla risoluzione 2^j è uguale alla proiezione ortogonale sullo spazio $\mathbf{V}_{2^j}$, possiamo scrivere questo principio come:

$$\lim_{j\to+\infty} \mathbf{V}_{2^j} = \bigcup_{j=-\infty}^{+\infty} \mathbf{V}_{2^j} \tag{2.24}$$

è fitto in $\mathbf{L}_2(\mathbf{R})$

e

$$\lim_{j\to-\infty} \mathbf{V}_{2^j} = \bigcap_{j=-\infty}^{+\infty} \mathbf{V}_{2^j} = \{0\} \tag{2.25}$$

Chiameremo qualunque insieme di spazi vettoriali $(\mathbf{V}_{2^j})_{j\in\mathbf{Z}}$ che soddisfa le proprietà 2.20 - 2.25 come Approssimazione multirisoluzione di $\mathbf{L}^2(\mathbf{R})$. La serie di operatori A_{2^j} che soddisfa le proprietà 1)- 6) fornisce un'approssimazione per qualunque funzione di $\mathbf{L}^2(\mathbf{R})$ alla risoluzione 2^j

Teorema 1 *Sia $(\boldsymbol{V}_{2^j})_{j\in\mathbf{Z}}$ un'approssimazione multirisoluzione di $\boldsymbol{L}^2(\boldsymbol{R})$. Esiste un'unica funzione $\phi(x) \in \boldsymbol{L}^2(\boldsymbol{R})$, chiamata scaling function, tale che se poniamo $\phi_{2^j}(x) = 2^j\phi(2^j x)$ per $j \in \boldsymbol{Z}$ allora:*

$(\sqrt{2^{-j}}\phi_{2^j}(x-2^{-j}n))_{n\in\mathbf{Z}}$ è una base ortonormale di $\boldsymbol{V}_{2^j}$.

Il teorema mostra che possiamo costruire una base ortonormale di qualunque $\mathbf{V}_{2^j}$ dilatando una funzione $\phi(x)$ con coefficiente 2^j e traslando la funzione ottenuta su una griglia nella quale l'intervallo è proporzionale a 2^{-j}.

La funzione $\phi_{2^j}(x)$ è normalizzata rispetto alla norma $\mathbf{L}^2(\mathbf{R})$ il coefficiente $\sqrt{2^{-j}}$ appare nel set base per normalizzare le funzioni nella norma di $\mathbf{L}^1(\mathbf{R})$.

Per dare un'approssimazione multirisoluzione $(\mathbf{V}_{2^j})_{j\in\mathbf{Z}}$ esiste un'unica funzione di scaling $\phi(x)$ che soddisfa il teorema 1. Tuttavia, per differenti approssimazioni multirisoluzione le funzioni di scaling sono differenti. La figura 2.8 mostra un esempio di funzione di scala continua, differenziabile e che decresce esponenzialmente. La sua trasformata di Fourier ha l'andamento di un filtro passa basso. La proiezione ortogonale su $\mathbf{V}_{2^j}$ può essere ora calcolata scomponendo il segnale f(x) su una base ortonormale definita dal teorema 1. In particolare:

$$\forall f(x) \in \mathbf{L}^2(\mathbf{R}), A_{2^j} f(x) = 2^{-j} \sum_{n=-\infty}^{+\infty} \left\langle f(u), \phi_{2^j}(u-2^{-j}n)\right\rangle \phi_{2^j}(x-2^{-j}n) \tag{2.26}$$

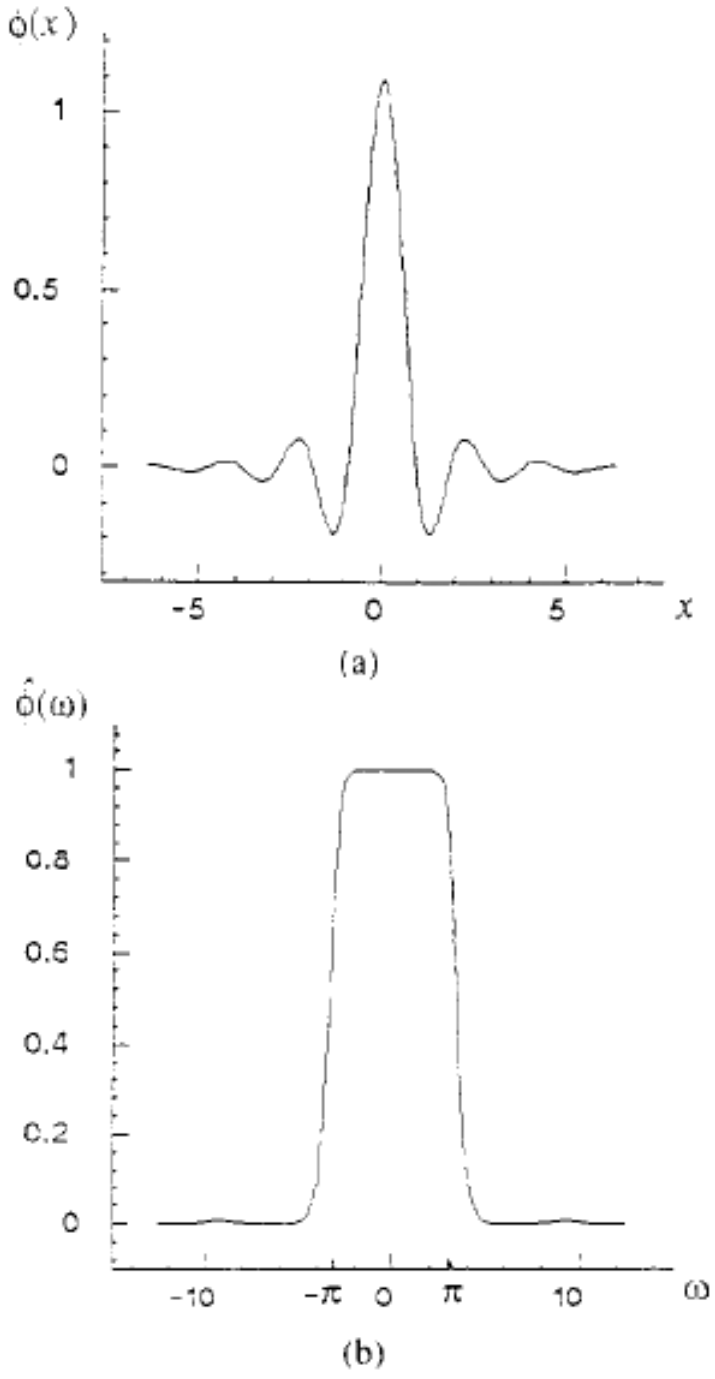


Figura 2.8: (a) Esempio di funzione di scala $\phi(x)$. (b) Trasformata di Fourier $\hat{\phi}(x)$

L'approssimazione del segnale f(x) alla risoluzione 2^j, A_{2^j}f(x) è quindi caratterizzata dall'insieme di prodotti interni che vengono indicati da:

$$A^d_{2^j} f = (\langle f(u), \phi_{2^j}(u - 2^{-j}n)\rangle)_{n\in\mathbf{Z}} \tag{2.27}$$

$A^d_{2^j} f$ è chiamato *approssimazione discreta* (discrete approximation) di f(x) alla risoluzione 2^j. Poichè i computer possono solamente elaborare segnali discreti, dobbiamo lavorare con approssimazioni discrete. Ogni prodotto interno può anche essere interpretato come un prodotto di convoluzione valutato nel punto 2^{-j}n

$$\langle f(u), \phi_{2^j}(u - 2^{-j}n)\rangle = \int_{-\infty}^{+\infty} f(u)\phi_{2^j}(u - 2^{-j}n)du = (f(u) * \phi_{2^j}(-u))(2^{-j}n) \tag{2.28}$$

quindi possiamo riscrivere $A^d_{2^j} f$ come:

$$A^d_{2^j} f = (f(u) * \phi_{2^j}(-u))(2^{-j}n)_{n\in\mathbf{Z}} \tag{2.29}$$

Poichè ϕ(x) è un filtro passa basso, questo segnale discreto può essere interpretato come il risultato del filtro passa basso applicato al segnale f(x) seguito da un campionamento uniforme alla frequenza 2^j. In una operazione di approssimazione, quando rimuoviamo i dettagli di f(x) più piccoli di 2^{-j}, stiamo eliminando le più alte frequenze della funzione in esame. Questa funzione di scalatura ϕ(x) costituisce un particolare filtro passa basso poichè la famiglia di funzioni $(\sqrt{2^{-j}}\phi_{2^j}(x - 2^j n))$ è una famiglia ortonormale.

2.4.3 Implementazione della trasformata multirisoluzione

In pratica, un dispositivo fisico di misura può solamente misurare un segnale con una risoluzione finita. Per il nostro scopo, supponiamo che la risoluzione sia uguale a 1. Sia $A^d_1 f$ l'approssimazione discreta alla risoluzione 1 che è stata misurata. Il principio di causalità afferma che da $A^d_{2^j} f$ possiamo calcolare tutte le approssimazioni discrete $A^d_{2^j} f$ per j<0. Di seguito descriveremo un semplice metodo iterativo per calcolare queste approssimazioni discrete. Sia $(\mathbf{V}_{2^j})_{j\in\mathbf{Z}}$ un'approssimazione multirisoluzione e sia ϕ(x) la corrispondente funzione di scala. La famiglia di funzioni $(\sqrt{2^{-j-1}}\phi_{2^{j+1}}(x - 2^{-j-1}k))_{k\in\mathbf{Z}}$ è una base ortonormale di $\mathbf{V}_{2^{j+1}}$. Noi sappiamo che per qualunque $n \in \mathbf{Z}$ la funzione $\phi_{2^j}(x - 2^{-j}n)$ è un elemento di $\mathbf{V}_{2^j}$ che è incluso in $\mathbf{V}_{2^j+1}$. Queste possono essere espanse nella base ortonormale di $\mathbf{V}_{2^{j+1}}$:

$$\phi_{2^j}(x - 2^{-j}n)$$

$$= 2^{-j-1} \sum_{k=-\infty}^{+\infty} \left\langle (\phi_{2^j}(u-2^{-j}n), \phi_{2^{j+1}}(u-2^{-j-1}) \right\rangle \phi_2^{j+1}(x-2^{-j-1}k) \qquad (2.30)$$

Sostituiendo le variabili nel prodotto interno dell'integrale si può mostrare che:

$$2^{-j-1} \left\langle \phi_{2^j}(u-2^{-j}n), \phi_{2^{j+1}}(u-2^{-j-1}k) \right\rangle = \langle \phi_{2^{-1}}(u), \phi(u-(k-2n)) \rangle \qquad (2.31)$$

Quando calcoliamo il prodotto interno di f(x) con $\phi_{2^j}(x-2^{-j}n)$ dalla di 2.30 otteniamo:

$$\left\langle f(u), \phi_{2^j}(u-2^{-j}n) \right\rangle =$$

$$\sum_{k=-\infty}^{+\infty} \langle \phi_{2^{-1}}, \phi(u-(k-2n)) \rangle \left\langle f(u), \phi_{2^{j+1}}(u-2^{-j-1}k) \right\rangle \qquad (2.32)$$

Sia H il filtro discreto la cui risposta all'impulso è data da:

$$\forall n \in \mathbf{Z}, h(n) = \langle \phi_{2^{-1}}(u), \phi(u-n) \rangle \qquad (2.33)$$

Sia $\tilde{\mathrm{H}}$ il filtro speculare con risposta all'inpulso $\tilde{\mathrm{h}}$(n) = h(-n). Inserendo la 2.33 nell'equazione precedente otteniamo:

$$\left\langle f(u), \phi_{2^j}(u-2^{-j}n) \right\rangle = \sum_{k=-\infty}^{+\infty} h(2n-k) \left\langle f(u), \phi_{2^{j+1}}(u-2^{-j-1}k) \right\rangle \qquad (2.34)$$

L'equazione 2.34 mostra che $A_{2^j}^d f$ può essere calcolata dalla convoluzione di $A_{2^{j+1}}^d f$ con $\hat{H}$ e mantenendo ogni altro campione dell' uscita. Tutte le approssimazioni discrete $A_{2^j}^d f$ per j<0, possono così essere calcolate da $A_1^d f$ ripetendo questo processo. Questa operazione è chiamata trasformata piramidale. L'algoritmo è illustrato mediante il diagramma a blocchi di figura 2.11.

In pratica, il dispositivo di misura dá solo un numero finito di campioni: $A_1^d f = (\alpha_n)_{1 \le n \le N}$. Ogni segnale discreto $A_{2^j}^d f$ (j<0) ha 2^jN campioni. Al fine di evitare effetti ai bordi quando si calcola l'approssimazione discreta $A_{2^j}^d f$, si suppone che il segnale originale $A_1^d f$ sia simmetrico rispetto a n=0 e n=N

$$\alpha_n = \begin{cases} \alpha_{-n} & \text{se } -N < n < 0 \\ \alpha_{2N-n} & \text{se } 0 < n < 0 \end{cases} \qquad (2.35)$$

Se la risposta all'impulso del filtro $\tilde{\mathrm{H}}$ è pari ($\tilde{\mathrm{H}}$ = H), ciascuna approssimazione discreta $A_{2^j}^d f$ sarà anche simmetrica rispetto a n=0 e n=2^{-j}N. La figura 2.9 (a) mostra l'approssimazione discreta $A_{2^j}^d f$ di un segnale continuo f(x) alla risoluzione

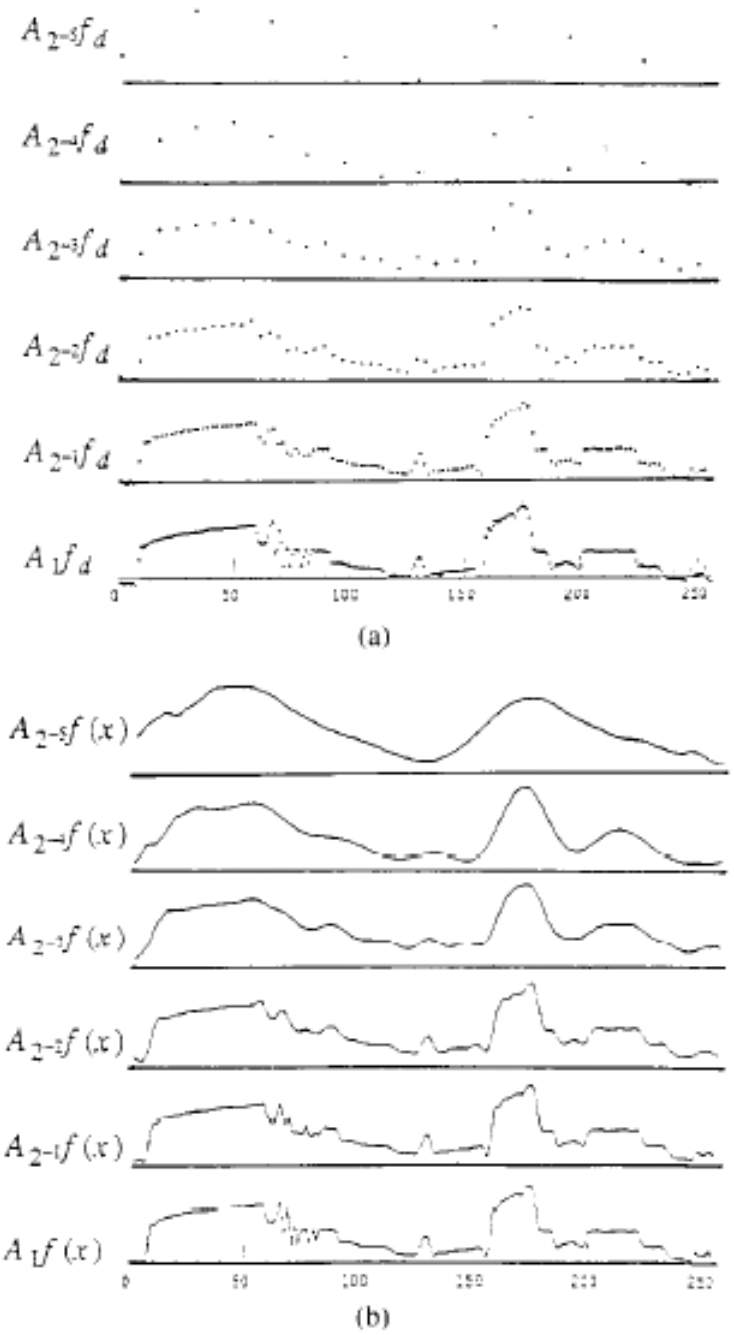


Figura 2.9: Approssimazione discreta $A^d_{2^j}f$ alla varie risoluzioni, Approssimazione continua $A^d_{2^j}f$ alle varie risoluzioni

1, $\frac{1}{2}$,$\frac{1}{4}$,$\frac{1}{8}$,$\frac{1}{16}$,$\frac{1}{32}$. Il segnale continuo approssimato $A_{2^j}f$ mostrato in figura 2.9 (b) è stato calcolato interpolando le approssimazioni discrete di 2.26. Al diminuire della risoluzione, i più piccoli dettagli di f(x) gradatamente scompaiono.

Il teorema 1 mostra che l'approssimazione multirisoluzione $(\mathbf{V}_{2^j})_{j\in\mathbf{Z}}$ è completamente caratterizzata da una funzione di scala $\phi(x)$. La funzione di scala può essere definita come una funzione $\phi(x) \in \mathbf{L}^2(\mathbf{R})$ tale che, per ogni i,$j \in \mathbf{Z}$, $(\sqrt{2^{-j}}\phi_{2^j})(x-2^{-j}n)_{n\in\mathbf{Z}}$ è una famiglia ortonormale, e se $\mathbf{V}_{2^j}$ è uno spazio vettoriale generato da questa famiglia di funzioni, allora $(\mathbf{V}_{2^j})_{j\in\mathbf{Z}}$ è un'approssimazione multirisoluzione di $\mathbf{L}^2(\mathbf{R})$. Una funzione di scalatura $\phi(x)$ deve essere continuamente derivabile e il decadimento asintotico di $\phi(x)$ e $\phi'(x)$ all'infinito devono soddisfare la condizione

$$|\phi(x)| = O(x^{-2}) \text{ e } |\phi'(x)| = O(x^{-2})$$

Il teorema che segue dà una caratterizzazione pratica della trasformata di Fourier di una funzione di scala.

Teorema 2 *Sia $\phi(x)$ una funzione di scala, e sia H un filtro discreto con risposta all'impulso h(n) =$\langle \phi_{2^{-j}}, \phi(u-n)\rangle$. Sia H(ω) la serie di Fourier definita da:*

$$H(\omega) = \sum_{n=-\infty}^{+\infty} h(n)e^{-in\omega}$$

H(ω) soddisfa le seguenti due proprietà:

$$|H(0)| = 1 \text{ e } h(n) = O(n^{-2}) \text{ all'infinito}$$

$$|H(\omega)|^2 + |H(\omega+\pi)|^2 = 1 \tag{2.36}$$

Viceversa, sia H(ω) una serie di Fourier che soddisfa le precedenti due proprietà e tale che:

$$|H(\omega)| \neq 0 \text{ per } \omega \text{ in } \left[0, \frac{\pi}{2}\right] \tag{2.37}$$

quindi la funzione definita da:

$$\hat{\phi}(\omega) = \prod_{p=1}^{+\infty} H(2^{-p}\omega) \tag{2.38}$$

è la trasformata di Fourier della funzione di scalatura.

I filtri che soddisfano la seconda proprietà 2.36 sono chiamati filtri coniugati (con-

jugate filter)[1, 2, 3]. Quindi dato un filtro coniugato H che soddisfa le condizioni 2.36 e 2.37 ,è possibile calcolare la trasformata di Fourier della corrispondente funzione di scala mediante la 2.34. È possibile scegliere $H(\omega)$, al fine di ottenere una funzione di scala ϕ(x) che abbia buone proprietà di localizzazione sia nel dominio della frequenza che nel dominio spaziale.

2.4.4 Rappresentazione della wavelet

In questa sezione viene spiegato come estrarre la differenza di informazioni tra l'approssimazione di una funzione f(x) alla risoluzione 2^{j+1} e 2^j. La differenza di informaioni è chiamata *Segnale di Dettaglio* (Detail Signal) alla risoluzione 2^j. Le approssimazioni alla risoluzione 2^{j+1} e 2^j di un segnale sono rispettivamente uguali alla loro proiezione ortogonale su $\mathbf{V}_{2^j+1}$ e $\mathbf{V}_{2^j}$. Applicando il teorema di proiezione, si può dimostrare che il segnale di dettaglio alla risoluzione 2^j è dato dalla proiezione ortogonale del segnale originale sul complemento ortogonale di $\mathbf{V}_{2^j}$ in $\mathbf{V}_{2^j+1}$. Sia $\mathbf{O}_{2^j}$ il complemento ortogonale, per esempio:

$$\mathbf{O}_{2^j} \text{ è ortogonale a } \mathbf{V}_{2^j}$$

$$\mathbf{O}_{2^j} \oplus \mathbf{V}_{2^j} = \mathbf{V}_{2^j}$$

Per calcolare la proiezione ortogonale di f(x) in $\mathbf{O}_{2^j}$ abbiamo bisogno di determinare una base ortonormale di $\mathbf{O}_{2^j}$. Analogalmente al Teorema 1, il Teorema 3 mostra che una base può essere costruita scalando e traslando una funzione ψ(x).

Teorema 3 *Sia $(\boldsymbol{V}_{2^j})_{j\in\boldsymbol{Z}}$ una sequenza di spazi vettoriali multirisoluzione, sia $\phi(x)$ la funzione di scala e sia H il corrispondente filtro coniugato. Sia $\psi(x)$ una funzione la cui trasformata di Fourier è data da:*

$$\hat{\psi}(\omega) = G(\frac{\omega}{2})\hat{\phi}(\frac{\omega}{2}) \text{ con } G(\omega) = e^{-i\omega}\overline{H(\omega+\pi)} \tag{2.39}$$

Indichiamo con $\psi_{2^j}(x) = 2^j\psi(2^j x)$ la dilatazione di $\psi(x)$ rispetto a 2^j. Quindi $\left(\sqrt{2^{-j}}\psi_{2^j}(x-2^{-j}n)\right)_{n\in\boldsymbol{Z}}$ è una base ortonormale di O_{2^j} e

$$\left(\sqrt{2^{-j}}\psi_{2^j}(x-2^{-j}n)\right)_{n,j\in\boldsymbol{Z}^2}$$

è una base ortonormale di $\boldsymbol{L}^2(\boldsymbol{R})$. Quindi $\psi(x)$ è chiamata wavelet ortogonale (orthogonal wavelet).

Una base ortonormale di $\mathbf{O}_{2^j}$ può così essere calcolata scalando la wavelet $\psi(x)$ con un coefficiente 2^j e traslandola in una griglia i cui intervalli sono proporzionali a 2^{-j}. Per calcolare una wavelet, dobbiamo definire una funzione $H(\omega)$ che soddisfa le condizioni 2.36-2.37 del teorema 2, calcolare la corrispondente funzione di scala $\phi(x)$ mediante l'equazione 2.38 e la wavelet $\psi(x)$ con la 2.39. A seconda della scelta di $H(\omega)$, la funzione di scala $\phi(x)$ e la wavelet $\psi(x)$ possono avere una buona localizzazione sia nel dominio spaziale che nel dominio di Fourier. Daubechies [4] ha studiato le proprietà di $\phi(x)$ e $\psi(x)$ a seconda del comportamento di $H(\omega)$. Le prime wavelet trovate da Meyer sono entrambe $\mathbf{C}^\infty$ e hanno un decadenza asintotica più rapida dell'inversa moltiplicativa di ogni polinomio. Daubechies ha dimostrato che per qualunque n > 0, possiamo trovare una funzione $H(\omega)$ tale che la corrispondente wavelet $\psi(x)$ ha un supporto compatto ed è n volte continuamente differenziabile [4].

La decomposizione di un segnale in una base wavelet ortonormale dà una rappresentazione intermedia tra la rappresentazione di Fourier e quella temporale. Le proprietà di una base wavelet ortonormale sono discusse da Meyer in [5]. A causa della doppia localizzazione, nel dominio di Fourier e in quello spaziale, è possibile caratterizzare la regolarità locale di una funzione f(x) sulla base dell'espansione dei coefficienti in una base wavelet ortonormale [6]. Per esempio, dalla diminuzione asintotica dei coefficienti di una wavelet, siamo in grado di determinare se una funzione f(x) è derivabile n volte in un punto x_0. La figura 2.10 mostra la wavelet associata con la funzione di scalamento di figura 2.8. Questa wavelet è simmetrica rispetto al punto x = $\frac{1}{2}$. L'energia della wavelet nel dominio della frequenza è principalmente concentrata nell'intervallo $[-2\pi,-\pi]\bigcup[\pi,2\pi]$.

Sia $\mathbf{P}_{\mathbf{O}_{2^j}}$ la proiezione ortogonale su uno spazio vettoriale $\mathbf{O}_{2^j}$. In conseguenza al teorema 3, questo operatore può essere scritto come:

$$\mathbf{P}_{\mathbf{O}_{2^j}} f(x) = 2^{-j} \sum_{n=-\infty}^{+\infty} \left\langle f(u), \psi_{2^j}(u - 2^{-j}n) \right\rangle \psi_{2^j}(x - 2^{-j}n) \tag{2.40}$$

$\mathbf{P}_{\mathbf{O}_{2^j}} f(x)$ produce i dettagli di f(x) alla risoluzione 2^j. Questo è caratterizzato da un inseme di prodotti interni

$$\mathbf{D}_{2^j} f = \left(\left\langle f(u), \psi_{2^j}(u - 2^{-j}n) \right\rangle \right)_{n \in \mathbf{Z}} \tag{2.41}$$

$\mathbf{D}_{2^j} f$ è chiamato *segnale di dettaglio discreto* (discrete detail signal) alla risoluzione 2^j. $\mathbf{D}_{2^j} f$ contiene differenti informazioni tra $A^d_{2^{j-1}} f$ e $A^d_{2^j} f$. Come abbiamo

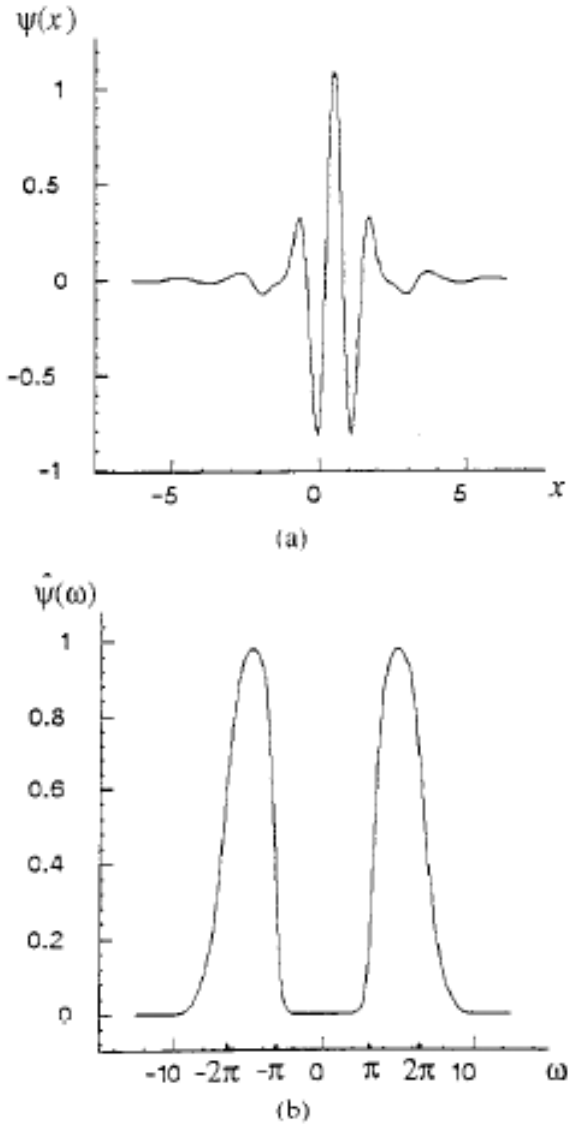


Figura 2.10: (a) Wavelet associata alla funzione di scala di figura 2.8, (b) modulo della trasformata di Fourier

mostrato nella 2.29, possiamo affermare che ognuno di questi prodotti interni è uguale alla convoluzione di f(x) con $\psi_{2^j}(-x)$ valutato in $2^{-j}n$

$$\left\langle f(u), \psi_{2^j}(u-2^{-j}n)\right\rangle = \left(f(u) * \psi_{2^j}(-u)\right)\left(2^{-j}n\right) \tag{2.42}$$

Le equazioni 2.40 e 2.41 mostrano che il segnale di dettaglio discreto alla risoluzione 2^j è uguale al campionamento uniforme di $\left(f(u) * \psi_{2^j}(-u)\right)(x)$ a 2^j

$$\mathbf{D}_{2^j} f = \left(\left(f(u) * \psi_{2^j}(-u)\right)(2^{-j}n)\right)_{n\in\mathbf{Z}} \tag{2.43}$$

La wavelet $\psi(x)$ può essere vista come un filtro passa banda la cui bande passanti sono approssimativamente uguali a $[-2\pi,-\pi]\bigcup[\pi,2\pi]$. Quindi, il segnale di dettaglio $\mathbf{D}_{2^j} f$ descrive f(x) nelle bande di frequenze $\left[-2^{-j+1}\pi,-2^{-j}\pi\right]\bigcup\left[2^{-j}\pi,2^{-j+1}\pi\right]$. Possiamo provare per induzione che per ogni j>0 il segnale originale discreto A_1^f misurato alla risoluzione 1 è rappresentato da:

$$\left(\mathbf{A}_{2^j}^d f, (\mathbf{D}_{2^j} f)_{-J\le j\le -1}\right) \tag{2.44}$$

Questo insieme di segnali discreti è chiamato rappresentazione wavelet ortogonale (orthogonal wavelet representation), ed è costituita da un segnale di rifertimento alla risolutione 2^j per $-J \le j \le -1$. La rappresentazione wavelet ortogonale può essere interpretata come una decomposizione del segnale originale in una base wavelet ortonormale, oppure come una decomposizione di un insieme di segnali con canali di frequenze indipendenti come nel Marr's human Vision Model[7]. L'indipendenza è dovuta all'ortogonalità delle funzioni wavelet. È difficile dare una precisa interpretazione del modello in termini di frequenza di decomposizione a causa della sovrapposizione di canali di frequenza. Comunque, possiamo controllare queste sovrapposizioni grazie all'ortogonalità delle nostre funzioni di decomposizione. Questo perchè i tool di analisi funzionale forniscono una miglior comprensione di questa decomposizione. Se non consideriamo il supporto della sovrapposizione spettrale, l'interpretazione nel dominio della frequenza fornisce un intuitivo approccio al modello. In analogia con la struttura dati piramidale Laplacian, $A_{2^{-j}}^d f$ fornisce il livello più alto del Gaussian pyramid data, e i dati di $\mathbf{D}_{2^j} f$ forniscono i successivi livelli della piramide di Laplacian. A differenza della piramide di Laplacian, tuttavia, non vi è alcuna sovrapposizione, e i coefficienti dell'insieme di dati è indipendente.

2.4.5 Algoritmo Piramidale

In questo paragrafo descriveremo l'algoritmo piramidale per calcolare la rappresentazione wavelet. Anche in questo caso mostreremo che $\mathbf{D}_{2^j} f$ può essere calcolato dalla convoluzione di $A^d_{2^{j+1}} f$ con un filtro discreto G la cui forma verrà caratterizzata. Per ogni $n \in \mathbf{Z}$ la funzione $\psi_{2^j}(x - 2^{-j}n)$ è un membro di $\mathbf{O}_{2^j} \subset \mathbf{V}_{2^{j-1}}$. Nello stesso modo in cui abbiamo definito la 2.30, questa funzione può essere espressa su una base ortonormale di $\mathbf{V}_{2^{j+1}}$

$$\psi_{2^j}(x - 2^{-j}n) =$$

$$2^{-j-1} \sum_{k=-\infty}^{+\infty} \left\langle \psi_{2^j}(u - 2^{-j}n), \phi_{2^{j-1}}(u - 2^{-j-1}k) \right\rangle \phi_{2^j+1}(x - 2^{-j-1}k) \tag{2.45}$$

Come abbiamo fatto nella 2.31, possiamo provare che:

$$2^{-j-1} \left\langle \psi_{2^j}(u - 2^{-j}n), \phi_{2^{j+1}}(u - 2^{-j-1}k) \right\rangle = \langle \psi_{2^{-1}}(u), \phi(u - (k - 2n)) \rangle \tag{2.46}$$

Quindi, calcolando il prodotto interno di f(x) con entrambi i membri di 2.45, otteniamo che:

$$\left\langle f(u), \psi_{2^j}(u - 2^{-j}n) \right\rangle$$

$$= \sum_{k=-\infty}^{+\infty} \langle \psi_{2^{-1}}(u), \phi(u - (k - 2n)) \rangle \left\langle f(u), \phi_{2^{j+1}}(u - 2^{-j-1}k) \right\rangle \tag{2.47}$$

Sia G il filtro discreto con risposta all'impuslo

$$g(n) = \langle \psi_{2^{-1}}(u), \phi(u - n) \rangle \tag{2.48}$$

e $\hat{G}$ un filtro simmetrico con risposta all'inpulso $\hat{g}(n) = g(-n)$. È possibile dimostrare che la funzione di trasferimento di questo filtro è la funzione $G(\omega)$ definita nel teorema 3 equazione 2.39. Inserendo la 2.48 nella 2.47 otteniamo

$$\left\langle f(u), \psi_{2^- j} n \right\rangle = \sum_{k=-\infty}^{+\infty} \hat{g}(2n - k) \left\langle f(u), \phi_{2^{j+1}}(u - 2^{-j-1}k) \right\rangle \tag{2.49}$$

L'equazione 2.49 mostra che possiamo calcolare il segnale di dettaglio $\mathbf{D}_{2^j} f$ dalla convoluzione di $A^d_{2^{j+1}} f$ con il filtro $\hat{G}$ e il mantenimento di ogni altro campione in uscita. La rappresentazione wavelet ortogonale di un segnale discreto $A^d_1 f$ può pertanto essere calcolata dalla successiva decomposizione di $A^d_{2^{j+1}} f$ in $A^d_{2^j} f$ e $\mathbf{D}_{2^j} f$ per $-J \leq j \leq -1$. Questo algoritmo è illustrato nello schema a blocchi di figura

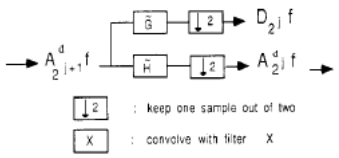


Figura 2.11: Schema a blocchi dell'algoritmo piramidale

2.11.

In pratica, il segnale $A_1^d f$ ha solamente un numero finito di campioni. Un metodo per gestire i problemi ai bordi utilizza una simmetria per quanto riguarda il primo e l'ultimo campione. L'equazione 2.39 del teorema 3 implica che la risposta all'impulso del filtro G è collegata alla risposta all'impulso del filtro H da:

$$g(n) = (-1)^{1-n} h(1-n) \tag{2.50}$$

G è speculare al filtro H, ed è un filtro passa alto. Nell'elelaborazione dei segnali G e H sono chiamati filtri in quadratura (quadrature mirror filters)[1]. L'equazione 2.49 può essere interpretata come l'uscita del filtro passa alto del segnale discreto $A^d_{2^{j+1}} f$. Se il segnale originale ha N campioni, allora i segnali discreti $\mathbf{D}_{2^j} f$ e $\mathbf{A}^d_{2^j} f$ hanno $2^j N$ campioni ciascuno. così la rappresentazone della wavelet

$$\left(A^d_{2^j} f, (\mathbf{D}_{2^j} f)_{-J \le j \le -1} \right)$$

ha lo stesso numero totale di campioni come l'approssimazione del segnale originale $A_1^d f$. Questo avviene perchè la rappresentazione è ortogonale. La figura 2.12 b da la rappresentazione wavelet del segnale $A_1^d f$ decomposto in figura 2.9. L'energia dei campioni di $\mathbf{D}_{2^j} f$ danno la misura dell'irregolarità del segnale alla risoluzione 2^{j+1}. Ogni volta che $A_{2^j} f(x)$ e $A_{2^{j+1}} f(x)$ sono significativamente diverse, il segnale di dettaglio ha un'elevata amplificazione. Nella figura 2.12 questo comportamento è visibile nell'area le cui coordinate in ascissa sono comprese tra 60 e 80.

Abbiamo visto che la rappresentazione wavelet è completa. Possiamo ora mostra che il segnale originale discreto può anche essere ricostruito con la trasformata piramidale. Poichè $\mathbf{O}_{2^j}$ è il complemento ortogonale di $\mathbf{V}_{2^j}$ in $\mathbf{V}_{2^{j+1}}$, $\left(\sqrt{2^{-j}} \phi_{2^j}(x - 2^{-j-1} n), \sqrt{2^{-j}} \psi_{2^j}(x - 2^{-j} n) \right)_{n \in \mathbf{Z}}$ è una base ortonormale di $\mathbf{V}_{2^{j+1}}$. Per qualunque n > 0 la funzione $\phi_{2^{j+1}}(x - 2^{-j-1} n)$ possono così essere decomposti

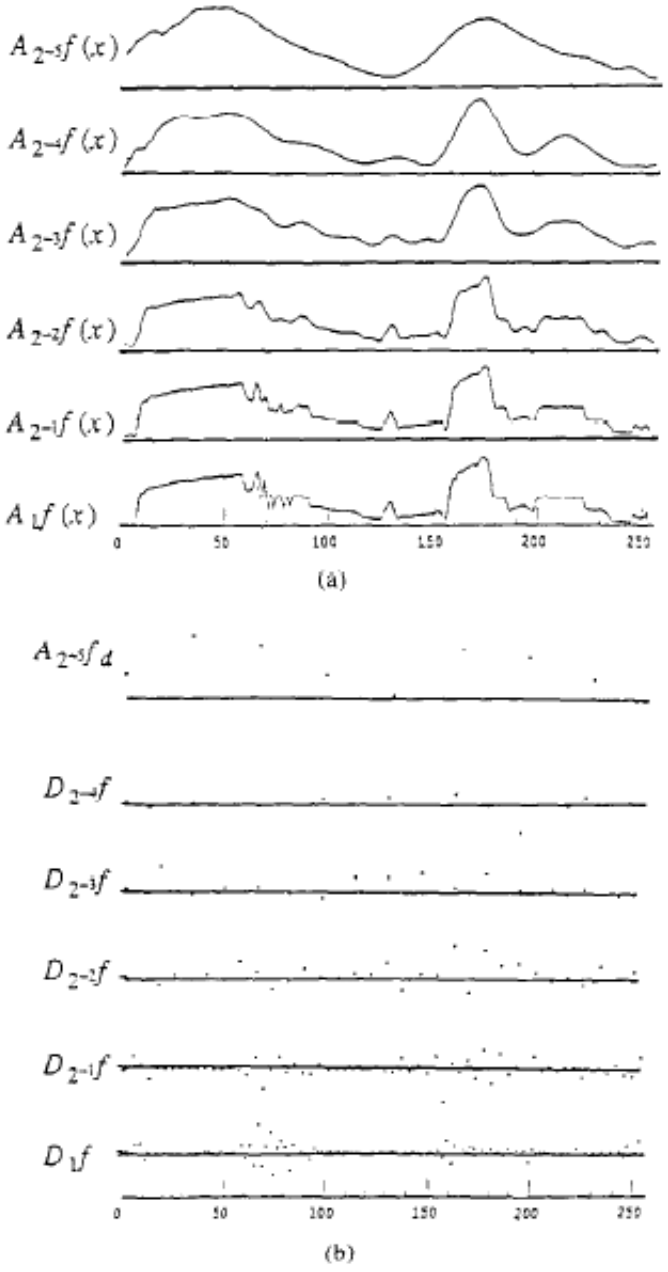


Figura 2.12: (a) Approssimazione continuamenta multirisoluzione$A_{2^j} f(x)$. (b) Rappresentazione wavelet del segnale $A_{2^{j+1}} f(x)$. Punti forniscono l'amplificazione del prodotto interno $\left\langle f(u), \psi_{2^j}(u - 2^{-j}n) \right\rangle$

in queste basi:

$$\phi_{2^{j+1}}(x-2^{-j-1}n)$$
$$=2^{-j}\sum_{k=-\infty}^{+\infty}\left\langle\phi_{2^j}(u-2^{-j}k),\phi_{2^{j+1}}(u-2^{-j-1}n)\right\rangle\phi_{2^j}(x-2^{-j}k)$$
$$+2^{-j}\sum_{k=-\infty}^{+\infty}\left\langle\psi_{2^j}(u-2^{-j}k),\phi_{2^{j+1}}(u-2^{-j-1}n)\right\rangle\psi_{2^j}(x-2^{-j}k) \tag{2.51}$$

Calcolando il prodotto interno di ciascun lato della equazione 2.51 con la funzione f(x) abbiamo:

$$\left\langle f(u),\phi_{2^{j+1}}(u-2^{-j-1}n)\right\rangle$$
$$=2^{-j}\sum_{k=-\infty}^{+\infty}\left\langle\phi_{2^j}(u-2^{-j}k),\phi_{2^{j+1}}(u-2^{-j-1}n)\right\rangle\left\langle f(u),\phi_{2^j}(u-2^{-j}k)\right\rangle$$
$$+2^{-j}\sum_{k=-\infty}^{+\infty}\left\langle\psi_{2^j}(u-2^{-j}k),\phi_{2^{j+1}}(u-2^{-j-1}n)\right\rangle\left\langle f(u),\psi_{2^j}(u-2^{-j}k)\right\rangle \tag{2.52}$$

Inserendo la 2.31 e la 2.46 in questa espressione e usando i filtri H e G, rispettivamente, definite in 2.33 e 2.48 produce:

$$\left\langle f(u),\phi_{2^{j+1}}(u-2^{-j-1}n)\right\rangle$$
$$=2\sum_{k=-\infty}^{+\infty}h(n-2k)\left\langle f(u),\phi_{2^j}(u-2^{-j}k)\right\rangle$$
$$+2\sum_{k=-\infty}^{+\infty}g(n-2k)\left\langle f(u),\psi_{2^j}(u-2^{-j}k)\right\rangle \tag{2.53}$$

Questa equazione mostra che $A^d_{2^{j+1}}f$ può essere ricostruito dall'inserimento di zeri tra ogni campione di $A^d_{2^j}f$ e $\mathbf{D}_{2^j}f$ e facendo la convoluzione del segnale risultate con i filtri H e G.

Capitolo 3

Applicazione delle Wavelet nell'analisi di traffico

Nell'ultimo periodo c'è stata una notevole evoluzione di Internet che ha modificato il modo di comunicare, e ha introdotto innumerevoli servizi che utilizzano la rete come mezzo per scambiare informazioni. Le prestazioni della reti risultano di fondamentale importanza per utilizzare in modo efficiente i servizi disponibili nel web; le performance delle reti di comunicazione infatti, possono essere influenzate da diversi fattori, alcuni dipendenti dall'infrastruttura fisica, altri dall'implementazione di protocolli di trasmissione. Proprio per evitare queste problematiche è necessario svolgere delle attività di misura delle prestazioni che si ottengono tramite attività di monitoraggio del traffico nei link della rete. Per svolgere l'attività di monitoraggio si può scegliere tra due diverse tipologie:

- *Analisi dei protocolli di segnalazione:* prevede di seguire lo scambio di informazioni tra applicazioni residenti in host diversi. Tale soluzione è dispendiosa soprattutto nel caso di link ad elevata capacità, in termini di memoria richiesta e tempi di elaborazione, in più richiede un'elevata conoscenza sui protocolli.

- *Analisi dei parametri statistici:* consente di sintetizzare le proprietà del traffico in un link. Tale soluzione è meno dispendiosa della precedente, in quanto offre la possibilità di effettuare il monitoraggio anche su reti ad alta velocità riducendo la richiesta di memoria e diminuendo l'attività di elaborazione. Per esempio packet loss round triptime.

La soluzione che verrà presentata invece, si basa sull' analisi del flusso di traffico. Questo metodo attualmente sembra fornire un'utile alternativa ai primi due, in quanto permette di misurare il traffico in una rete senza andarlo ad alterare. Il punto di partenza consiste in una traccia di traffico. Un traccia, che è una raccolta di informazioni dove per ogni pacchetto viene misurato il timestamp e l'header, può essere vista come una misura grezza di informazioni, le quali possono essere processate, in un secondo momento, in vari modi. Il dispositivo di misura associa ad ogni pacchetto un *timestamp*, il quale indica con una certa accuratezza l'istante in cui è stato catturato il pacchetto. La più importante e comune forma di presentazione di questi

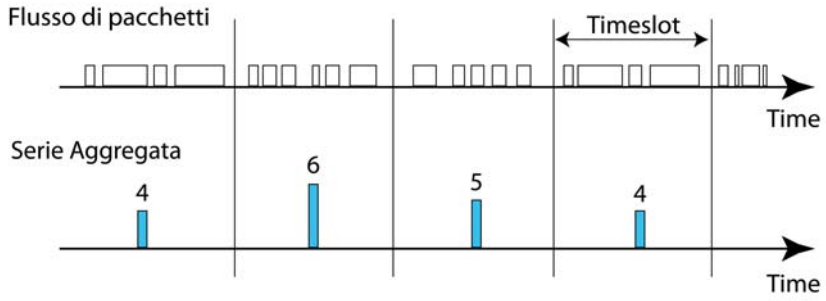


Figura 3.1: Esempio di serie temporale ottenuta dall'aggregazone del traffico in transito in un link di monitoraggio. Timeslot rappresenta l'interlvallo temporale in cui si contano i pacchetti.

dati sperimentali è la serie aggregata del traffico. Come si può vedere dalla figura 3.1, la serie aggregata si ottiene contando il numero di pacchetti, in transito sulla rete, durante intervalli di tempo consecutivi e di durata prefissata chiamati *timeslot*. Il processo definito precedentemente è detto processo di conteggio e rappresenta il punto di partenza per considerazioni presenti nel resto della tesi. Solitamente per ogni pacchetto è sufficiente salvare il timestamp e l'header. Nel nostro caso, se l'utente dedide di salvare la traccia, di ogni pacchetto catturiamo ulteriori informazioni per consentire un'analisi completa. L'obiettivo è quello di dedurre variazioni di performance attraverso l'analisi di traffico da un unico punto di monitoraggio. Questo approccio è molto semplice per quanto riguarda le misurazioni perchè è sufficiente osservare i pacchetti che transitano attraverso una specifica interfaccia di rete.

3.1 Misure del flusso di traffico

Nel 1993 W. Leland, M. Taqqu, W. Willinger e D. Wilson dell'Università di Boston hanno presentato un articolo intitolato " On Self-Similar Nature of Ethernet Traffic" alla conferenza SIGCOMM'93 (una versione estesa del documento può essere consultata in [14]). Esso ha portato ad un'importante svolta nel mondo delle misure di flusso di traffico. In questo lavoro innovativo fu dimostrato come modelli basati su processi di Poisson, fino ad allora utilizzati con successo per descrivere l'andamento del traffico in reti a commutazione di circuito, risultassero inesatti quando applicati su reti a commutazione di pacchetto. Mentre nel primo caso il traffico esibiva una sorta di smoothing quando veniva mediato su scale temporali via via crescenti, nel secono caso invece esso continuava a presentare irregolarità e burst su tutte le scale considerate. Dalla valutazione di molte tracce catturate tra il 1989 e il 1992 nelle reti locali di Bellcore è stato possibile dimostrare che l'aggregazione del traffico ethernet è self-similar. Questa proprietà fu individuata sia nel traffico relativo a reti locali (LAN) sia nel traffico riguardante reti di maggiore estensione e quindi di maggiore capacità (WAN). I risultati ottenuti hanno permesso di evidenziare che:

1. Il modello di Poisson (o Markoviano) non rappresenta la realtà.
2. L'aggregazione del traffico ethernet è self-similar.
3. Il fenomeno dei burst si mantiene su scale diverse.

Successivamente sono stati pubblicati molti altri articoli che hanno ulteriormente confermanto la natura self-similar del traffico di rete. Una spiegazione fisica della self-similarity osservata nel traffico ethernet è spiegata in [20, 21]. Il comportamento self-similar deriva dall'aggregazione di molte sorgenti ON-OFF, dove i periodi di On e di OFF hanno distribuzione di tipo Heavy-Tailed, ad esempio distribuzioni di Pareto.

3.2 Metodi per l'individuazione della Self-Similarity

Una misura del livello di correlazione, ossia di self-similarità in una serie temporale è fornita dal parametro di *Hurst*. Questo parametro ha preso il nome dall'idrologo H.E. Hurst, che spese molti anni a studiare problemi relativi ai bacini di raccolta dell'acqua e delle fluttuazioni del livello dell'acqua nel fiume Nilo e di molti altri fiumi.

Il parametro H assume valori compresi tra 0.5 e 1.0 ($0.5 \leq H \leq 1.0$), si ha self-similarity forte quando H assume valori prossimi a 1. Mentre per H=0.5 il traffico non è correlato e quindi non presenta self-similarità.
Sfortunatamente il calcolo di H è un processo molto difficile. Per ottenere una stima accurata di H sarebbe teoricamente necessario utilizzare un numero infinito di campioni, in pratica però solo un numero finito di campioni è disponibile, pertanto bisogna trovare un compromesso tra il numero di campioni necessari e l'accuratezza del valore di H stimato. Attualmente si sono sviluppati molti metodi per stimare la self-similarity di serie temporali [22, 23]. I metodi più importanti sono: Analisi Statistica R/S (R/S Statistics Analysis), Analisi Tempo Varianza (variance-time analysis) le analsisi basate su wavelet (wavelet-base analysis).

3.3 Aspetti teorici

Dopo la spiegazione della trasformata wavelet vista nel capitlo 2, focalizzeremo la nostra attenzione sull'utilizzo delle wavelet per l'attvità di monitoraggio del traffico. L'analisi basata sulle wavelet consente una valutazione sia nel dominio del tempo che nel dominio della frequenza. Sia $X(k)$ la serie temporale ottenuta contando il numero di pacchetti (o byte) che transitano attraverso un link di rete in successivi timeslot. Quando la struttura di correlazione delle serie temporali decresce in modo esponenziale la serie esibisce una *Long-range Dependence* (LRD) la quale da luogo ad un comportamento self-similar. Per caratterizzare il grado di LRD o, in modo simile, per quantificare la self-similarity è necessario considerare il parametro di Hurst. In particolare sotto le ipotesi che il processo aleatorio sia frazionario e abbia incrementi stazionari, il valore di H varia tra 0.5, se la serie temporale non è correlata, e 1 per le serie temporale completamente correlate. Considerando intervalli di tempo di durata progressivamente maggiore $2^j T$, la serie temporale:

$$X^{(j)}(k) = \frac{1}{2^j} \sum_{i=0}^{2^j-1} X\left(k2^j + i\right) \tag{3.1}$$

rappresenta la versione aggregata della serie $X(k)$ alla scala j. Sotto l'ipotesi di self-similarity per $X(k)$ può essere trovata la seguente relazione:

$$X^{(j)}(k) \stackrel{d}{=} 2^{j(H-1)} X(k) \tag{3.2}$$

dove $\stackrel{d}{=}$ indica l'uguaglianza della distribuzione di probabilità.

Una semplice interpretazione della formula (3.1) mostra come l'aggregazione della serie temporale $X^{(j)}(k)$ sia direttamente collegata all'approssimazione del segnale nel senso dell'analisi di Wavelet di Haar. In realtà, considerando la wavelet di Haar è possibile riformulare esattamente la procedura di aggregazione come la parte di approssimazione dell'analisi multirisoluzione. È quindi facile prendere in considerazione due forme di generalizzazione:

- Aggregazione usando differenti funzioni di scala;
- Applicazione dell'analisi del quantile ai coefficienti di dettaglio e approssimazione.

Il secondo aspetto rappresenta un contributo importante in quanto stabilisce una connessione tra l'analisi del quantile, presentata in [17], e il comportamento del tool proposto da Abry-Veitch (A-V Tool). Come notato in [13], il concetto di aggregazione temporale può essere esteso ad altri tipi di wavelet. In generale, il processo di aggregazione è associato ai coefficienti di approssimazione $a_x(j,k)$ ottenuti dall'analisi multirisoluzione:

$$a_x(j,k) = \langle x, \phi_{j,k} \rangle \tag{3.3}$$

dove $\phi_{j,k}$ è la funzione di traslazione e scalatura della funzione di scala ϕ_0. Mentre la wavelet di Haar consente la loro interpretazione come flusso di dati in termini rigorosi, i coefficienti di approssimazione sono tuttavia legati alla componente passa basso dell'espansione della wavelet, quindi possono avere un significato analogo. I coefficienti di dettaglio possono essere ottenuti come:

$$d_x(j,k) = \langle x, \psi_{j,k} \rangle \tag{3.4}$$

dove la funzione $\psi_{j,k}$ è la versione traslata e scalata della wavelet madre ψ_0. L'espressione 3.4 corrisponde ad un altro tipo di trasformazione dove i dati sono filtrati da un filtro passa banda invece che da un filtro passa basso. Si nota come sia possibile riscrivere la serie X(k) a partire dai coefficienti di approssimazione e dettaglio:

$$x(k) = approx_j + \sum_{j=1}^{J} detail_j(t) = \sum_k a_x(j,k)\phi_{J,K}(t) + \sum_{j=1}^{J}\sum_k d_x(j,k)\psi_{j,k}(t) \tag{3.5}$$

È stato inoltre dimostrato in [16] che l'analisi wavelet si comporta essenzialmente come una differenziazione di ordine opportunamente definito in modo che i coefficienti di dettaglio $d_x(j,k)$ possano essere associati a variazioni di flusso viste a scale diverse.

Per un processo self-similar esiste una relazione di scalatura tra i coefficienti della wavelet $a_x(j,k)$ e $d_x(j,k)$. Con il simbolo $c_x(j,k)$ indichiamo generalmente uno dei due:

$$c_x(j,k) \stackrel{d}{=} 2^{j(H+\frac{1}{2})} c_x(0,k) \tag{3.6}$$

dove $c_x(0,k) = X(k)$. Con riferimento alla definizione di aggregazione introdotta in 3.1 il rapporto deve essere normalizzato per il numero di campioni considerati; ottenendo:

$$c_x(j,k) \stackrel{d}{=} 2^{j(H-\frac{1}{2})} c_x(0,k) \tag{3.7}$$

Il tool di Abry-Veitch considera l'energia dei coefficienti di dettaglio $d_x(j,k)$ a differenti scale temporali:

$$\mathbb{E}\left[d_x(j,k)^2\right] = 2^{j(2H-1)} \mathbb{E}\left[d_x(0,k)^2\right], \tag{3.8}$$

Questa espressione fornisce un modo per identificare la presenza di LRD nei dati misurati e valutare quindi il corrispondente esponente di scala H. Si può notare che, poichè il valore medio dei coefficienti di dettaglio è nullo, l'energia corrisponde alla varianza.

Questo tool viene largamente utilizzato per identificare e quantificare l'entitità dello scaling nelle misure di dati. Questo è possibile mediante regressione lineare sullo spettro della wavelet rappresentato su scala logaritmica. Finchè i coefficienti di dettaglio sono scorrelati, la varianza decresce proporzionalmente al numero di campioni considerati e non dipende invece dal valore del coefficiente di Hurst che è sconosciuto. Questa importante proprietà consente di aumentare l'accuratezza della stima, incrementando il numero di campioni e ciò rappresenta una delle ragioni del successo del tool. Questa tool è tuttavia presenta dei limiti in occasioni di punti di non stazionarietà.

L'A-V tool si basa sulla stima della varianza. Da un punto di vista statistico si può sottolineare come la varianza sia sensibile al cambiamento empirico di pdf (distribuzione di probabilità) ma non consente una dettagliata comprensione del fenomeno. In questa situazione l'analisi con il quantile aggiunge informazioni

sulla distribuzione dei campioni.

3.3.1 Valutazione basata sul Quantile

L'analisi mediante quantile è un approccio robusto poichè permette di stimare il parametro di Hurst anche in presenza di eventi non stazionari. Richiamiamo nel seguito i passi principali per calcolare il parametro di valutazione facendo riferimento al generico $c_x(j,k)$. Questo rende evidente che ciò che segue si applica sia ai coefficienti di dettaglio che a quelli di approssimazione. Di conseguenza le curve ottenute dalla valutazione del quantile sono denotate con il termine di *Quantile-Interval Curve* (QIC) [24].
Sia $r_\gamma(j)$ il j-esimo campione di un Quantile-Interval Curve. Esso fornisce un limite sul valore che $c_x(j,k)$ può assumere. Questo limite può essere ecceduto con una probabilità γ chiamata *Probabilità di Violazione*.

$$P\left[c_x(j,k) \leq r_\gamma(j)\right] = 1-\gamma \tag{3.9}$$

La relazione di self-similarity tra $c_x(0,k)$ e $c_x(j,k)$ si estende al loro quantile, fornendo la seguente espressione che collega i punti di un QIC.

$$r_\gamma(j) - \mathbb{E}\left[c_x(j,k)\right] = 2^{j(H-\frac{1}{2})}\left[r_\gamma(0) - \mathbb{E}\left[c_x(j,k)\right]\right] \tag{3.10}$$

Il valore medio è nullo per i coefficienti di dettaglio quando $c_x(j,k) = d_x(j,k)$. Analogamente alla 3.8 l'esponente di scala può essere ottenuto in modo semplice riscrivendo la 3.10 in una scala bilogaritmica:

$$\lg \frac{r_\gamma(j) - \mathbb{E}\left[c_x(j,k)\right]}{r_\gamma(0) - \mathbb{E}\left[c_x(j,k)\right]} = j(H-\frac{1}{2}) \tag{3.11}$$

Capitolo 4

Implementazione di un nuovo Tool

In questi ultimi anni internet, grazie alla sua diffusione, ha rivoluzionato il modo di comunicare delle persone; quindi è divenuta di fondamentale importanza la gestione delle reti da parte delle aziende, delle istituzioni e delle comunità. Infatti, la rete internet porta con se molte insidie quali worm, virus, spam e molte altre che compromettono il corretto funzionamento delle infrastrutture di rete. Ovviamente con l'evoluzione del fenomeno internet si sono sviluppate varie applicazioni che consentono il monitoraggio e la gestione delle reti.
La maggior parte di queste soluzioni utilizza un approccio a posteriori in quanto prevedono una fase di acquisizione del traffico di rete per poi analizzare la traccia in un secondo momento.
Questo approccio non consente di individuare immediatamente i problemi ed avviare contromisure efficienti in breve tempo. Supponiamo il caso in cui un amministratore di rete stia effettuando un'acquisizione di 24 ore del traffico su una intranet, per valutare lo stato della rete interna, e durante questa fase si verifichi un attacco di tipo DoS (Denial of Service). Con questo tipo di approccio l'evento sarà rilevato solamente nel giorno successivo, o quando verrà analizzata la traccia, avviando così l'azione di intervento con oltre 24 ore di ritardo. Per questi motivi si è sentita la necessità di sviluppare un'applicazione che consenta l'analisi in tempo reale di cosa accade all'interno di una rete.

Nel seguito di questo capitolo verrà illustrato nel dettaglio il tool sviluppato durante l'attività di tesi, mostrando l'implementazione della Discrete Wavelet Transform. In più saranno presentate anche le problematiche, con relative soluzioni, incontrate durante lo sviluppo del tool.

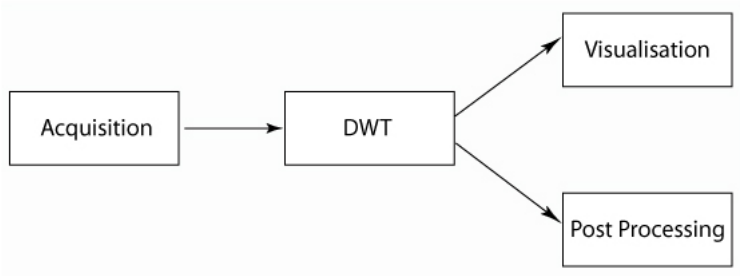


Figura 4.1: Componenti principali di Ma2t Tool

4.1 Ma2t Tool nel dettaglio

Lo scopo di questa tesi è la realizzazione di un tool che consenta il monitoraggio del traffico di rete in tempo reale, superando così il problema di un'analisi a posteriori, presente nella maggior parte degli altri sistemi di analisi.

Il programma è stato sviluppato utilizzando il linguaggio di programmazione Java™, in quanto quest'ultimo garantisce l'indipendeza dalla piattaforma grazie alla Java Virtual Machine ed inoltre consente il multi-threading che, come vedremo nel seguito, è una caratteristica fondamentale del nostro programma.

Concettualmente il tool è costituito dai 4 blocchi di figura 4.1:

- **Acquisizione del traffico**: Poichè Java™non consente di interagire con l'hardware a basso livello dei PC, è stato necessario installare la libreria JPCAP [8] il cui funzionamento è stato spiegato nella mia tesi di laurea triennale [9]. JPCAP è una libreria Java che sfrutta alcuni metodi nativi (JNI) e che, con opportune modifiche, consente di gestire le schede di rete in modalità promiscua e di utilizzare un sistema di filtraggio dei pacchetti. Tale libreria fornisce un'interfaccia con una serie di metodi per la cattura, l'analisi e la manipolazione dei pacchetti di rete. Le operazioni principali rese disponibili dalla libreria sono:

  - Cattura dei pacchetti sul link di ascolto. Il metodo utilizzato nel progetto per l'acquisizione del traffico è *getpacket()* [8].
  - Salvataggio dei pacchetti catturati in un file per l'analisi del traffico in un secondo momento. Per salvare i pacchetti acquisiti, si deve innanzitutto aprire un file utilizzando il metodo *JpcapWriter.openDumpFile()* attraverso una istanza del tipo JpcapCaptor e una stringa contenente il nome del file. Una volta definito dove salvare i pacchetti, per

scriverli è sufficiente usare il metodo *JpcapWriter.writePacket()*. Finita la fase di acquisizione si deve chiudere il file mediante il metodo *JpcapWriter.close()*.

- Identificazione automatica del tipo di pacchetti e generazione dei corrispondenti oggetti Java (pacchetti ethernet, IP V4, IP V6 ecc..).
- Filtraggio dei pacchetti in base alle regole definite dall'utente prima che i pacchetti vengano passati alle applicazioni.
- Identificazione dei link di rete per l'acquisizione del traffico. I metodi pricipalmente utilizzati sono:
  - getDeviceDescription(): restituisce una descrizione dell'interfaccia di rete (compatibile solo su piattaforma Windows).
  - getDeviceList(): restituisce gli identificativi delle interfacce di rete che possono essere usate per la cattura dei pacchetti.

  Un'altra caratteristica importante della libreria JPCAP è la possibilità di impostare il sistema in: *modalità di acquisizione promiscua*, che prevede di porre il link in ascolto e catturare tutti i pacchetti in transito sulla rete, oppure in *modalità non promiscua*, nella quale l'interfaccia cattura solo i pacchetti che vengono inviati o che hanno come destinazione il link di ascolto. Il vantaggio di utilizzare JPCAP è la possibilità di porre rimedio alla limitazione di Java consentendo un'acquisizione "pulita" dei pacchetti in quanto non altera il contenuto e permette di accedere alle informazioni a basso livello senza utilizzare strutture complesse.

- **Elaborazione**: La fase di elaborazione inizia quando sono stati acquisiti N intervalli temporali di traffico internet. Quando questi campioni sono pronti vengono processati mediante la Discrete Wavelet Transform, ed i risultati vengono presentati calcolando il quantile o la loro varianza dei coefficienti DWT ottenuti. Questa fase verrà spiegata con maggior dettaglio nel seguito di questo capitolo.

- **Visualizzazione**: una volta elaborati i dati nella fase precedente, questi vengono visualizzati graficamente in modo da consentire l'individuazione di anomalie nel modo più chiaro e veloce possibile. Come vedremo nel capitolo successivo vengono visualizzate due coppie di grafici:

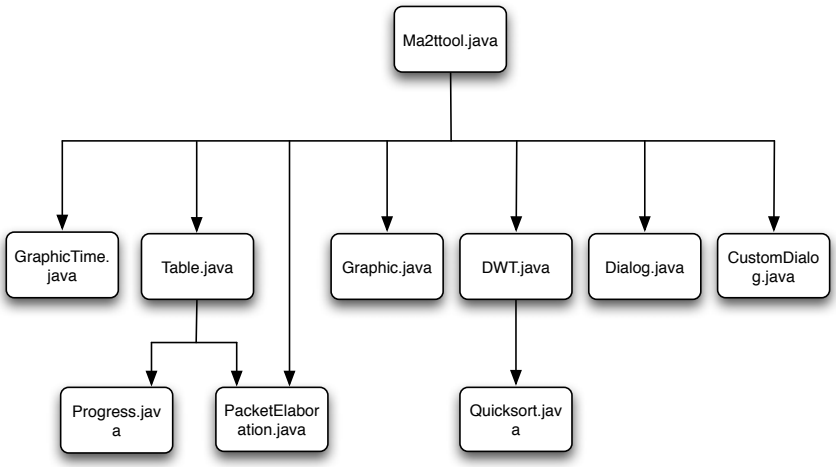


Figura 4.2: Struttura gerarchica delle classi che compongono MA2T Tool

  - Valore - Blocchi: in questa rappresentazione, i grafici dei coefficienti di approssimazione e dettaglio presentano sull'asse *x* il numero del blocco a cui si riferiscono, ognuno costituito da N timeslot elaborati, mentre nell'asse *y* viene riportato il valore calcolato nella fase di analisi.
  - Valore - Livelli: in questa rappresentazione, i grafici dei coefficienti di approssimazione e dettaglio presentano sull'asse *x* i livelli dell'algoritmo piramidale in cui viene effettuata l'elaborazione mentre nell'asse *y* viene riportato il logaritmo del valore calcolato nella fase di analisi. Ogni curva di tale grafico è riferita ad un singolo blocco.

- **Post Processing**: attualmente questa fase consente di estrarre le informazioni, quali timestamp, indirizzo ip sorgente, indirizzo ip destinazione e protocollo, dai pacchetti che producono anomalie nel grafico. Questa come vedremo è un'azione che permette di identificare in modo più chiaro la causa del problema.

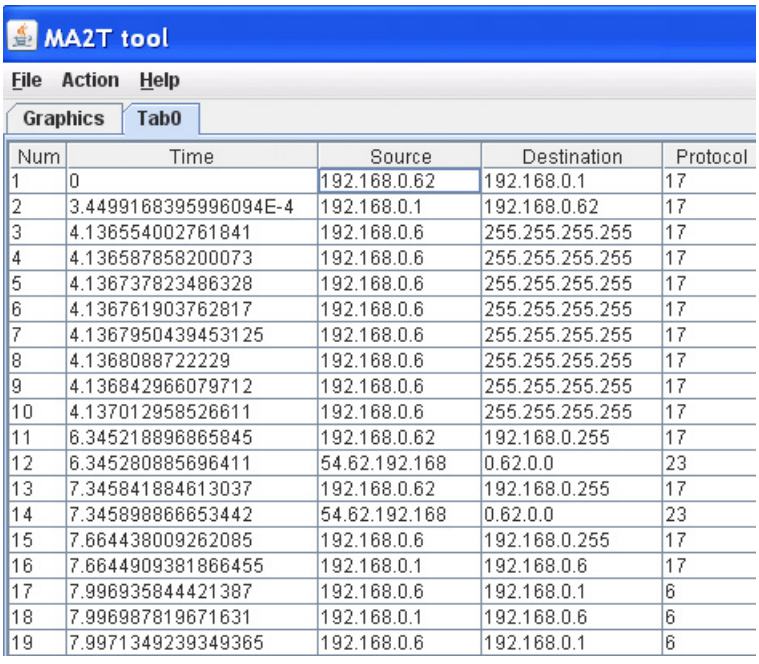
Ora che abbiamo dato una prima visione delle quattro fasi principali del tool mostriamo, in figura 4.2, la struttura gerarchica delle classi principali che compongono il software di monitoraggio da noi realizzato. Come si può vedere dalla figura 4.2 il programma è costituito da dieci classi; di seguito riportiamo una breve descrizione delle funzioni principali che ogni classe svolge.

- *Ma2ttool.java*: è la classe principale del tool che contiene la struttura ba-

se dell'algoritmo. Questa classe svolge anche le funzioni di avvio dei task relativi alla fase di acquisizione e a uno o più task di elaborazione dei dati ottenuti.

- *GraphicTime.java*: è la classe che contiene i metodi per la realizzazione del grafico che ha in ascissa il tempo, espresso indirettamente tramite il numero di blocchi, ed in ordinata il valore calcolato nella fase di elaborazione. In tale grafico vengono visualizzate contemporaneamente tante curve quanti sono i livelli.

- *Table.java*: è la classe che contiene i metodi per la costruzione della tabella in cui vengono esplicitate le informazioni sui pacchetti. Dalle impostazioni che l'utente può attivare nel programma principale, è possibile visualizzare una tabella che estrae informazioni da tutti i pacchetti o solo per la porzione di pacchetti di interesse. Un esempio di tabella è quella riportata in Figura 4.3



Num	Time	Source	Destination	Protocol
1	0	192.168.0.62	192.168.0.1	17
2	3.4499168395996094E-4	192.168.0.1	192.168.0.62	17
3	4.136554002761841	192.168.0.6	255.255.255.255	17
4	4.136587858200073	192.168.0.6	255.255.255.255	17
5	4.136737823486328	192.168.0.6	255.255.255.255	17
6	4.136761903762817	192.168.0.6	255.255.255.255	17
7	4.1367950439453125	192.168.0.6	255.255.255.255	17
8	4.1368088722229	192.168.0.6	255.255.255.255	17
9	4.136842966079712	192.168.0.6	255.255.255.255	17
10	4.137012958526611	192.168.0.6	255.255.255.255	17
11	6.345218896865845	192.168.0.62	192.168.0.255	17
12	6.345280885696411	54.62.192.168	0.62.0.0	23
13	7.345841884613037	192.168.0.62	192.168.0.255	17
14	7.345898866653442	54.62.192.168	0.62.0.0	23
15	7.664438009262085	192.168.0.6	192.168.0.255	17
16	7.6644909381866455	192.168.0.1	192.168.0.6	17
17	7.996935844421387	192.168.0.6	192.168.0.1	6
18	7.996987819671631	192.168.0.1	192.168.0.6	6
19	7.9971349239349365	192.168.0.6	192.168.0.1	6

Figura 4.3: Esempio tabella

- *Graphic.java*: è la classe che contiene i metodi per la realizzazione della seconda tipologia di grafici. In questo caso il grafico, come abbiamo detto precedentemente, pone sulle ascisse i vari livelli dell'algoritmo piramidale

mentre nelle ordinate pone il logaritmo del valore calcolato nella fase di elaborazione. Anche in questo caso vengono visualizzate più curve nel grafico. Ogni curva del grafico è riferita al rispettivo blocco in esame.

- *DWT.java*: è una classe che mette a disposizione i metodi per effettuare la fase di elaborazione dei campioni. Come anticipato, tratteremo con maggior dettaglio le peculiarità di questa classe nel seguito di questo capitolo.

- *Dialog.java*: fornisce i metodi necessari alla realizzazione della finestra di dialogo che consente di impostare i parametri per la creazione della porzione di tabella mentre si sta svolgendo la fase di analisi in real-time con salvataggio della traccia, oppure nell'analisi a posteriori quando si sta leggendo una traccia di traffico da file.

- *CustomDialog.java*: fornisce i metodi necessari alla realizzazione della finestra di dialogo per le impostazioni da parte dell'utente relative all'analisi del traffico in tempo reale appure all'analisi di tracce di traffico precedentemente acquisite.

- *Progress.java*: è una classe che fornisce i metodi per la realizzazione di una barra di avanzamento. Tale indicatore compare al termine della fase di analisi, per informare l'utente che il tool sta elaborando le informazioni per la realizzazione della tabella.

- *PacketElaboration.java*: fornisce i metodi per l'estrazione del timestamp e della lunghezza del pacchetto durante la creazione della tabella o nella fase di analisi in tempo reale.

- *Quicksort.java*: classico algoritmo di ordinamento basato su confronti che richiede tempo $\theta(n \lg n)$ dove n è la dimensione dell'input.

4.2 Algoritmi di MA2T Tool

Ora che sono state definite la struttura e le funzioni principali del programma possiamo entrare maggiormente nello specifico del funzionamento dell'applicativo. Prima di dare una descrizione dettagliata dei vari algoritmi utilizzati, mediante l'aiuto della figura 4.4, illustreremo il funzionamento del programma nel suo complesso. Dalla figura 4.4 si nota come il traffico all'interno di una rete di computer

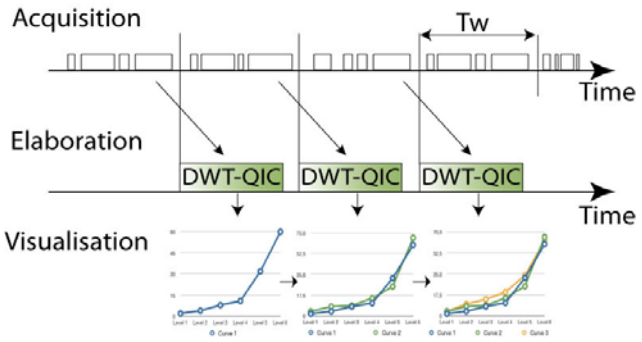


Figura 4.4: Principio di funzionamento di MA2T Tool

possa essere visto come una sequenza di pacchetti nel tempo. Per poter analizzare il flusso è necessario definire il concetto di *timeslot*. Il timeslot è una finestra temporale di dimensione fissa, definita dall'utente, nella quale vengono contati e catturati i pacchetti. Ovviamente la dimensione del timeslot deve essere definita in modo opportuno in base al tipo e alla quantità del traffico nella rete. Per effettuare l'analisi il nostro tool conta i pacchetti in ogni timeslot e ogni N blocchi di timeslot, rappresentati in figura da T_w, viene avviata la fase di elaborazione. In questa attività viene avviato un task per l'elaborazione di questi campioni, mentre il processo di acquisizione continua, evitando così di perdere pacchetti. Nel task di elaborazione i campioni vengono elaborati negli n livelli dell'algoritmo piramidale di figura 4.5. Una volta terminata l'elaborazione, i risultati vengono graficati nelle due coppie di grafici illustrate nel paragrafo precedente e che saranno approfonditi nel prossimo capitolo. Entriamo ora nel dettaglio del programma. Come possiamo vedere dalla

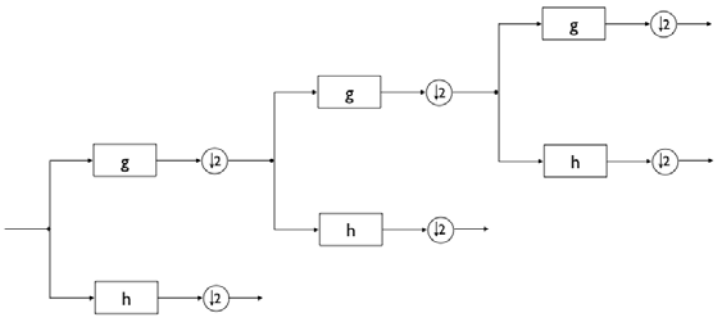


Figura 4.5: Algoritmo piramidale.

figura 4.6, il tool permette di scegliere tra la possibilità di analizzare il traffico di rete in tempo reale, oppure analizzare una traccia precedentemente acquisita. Nei

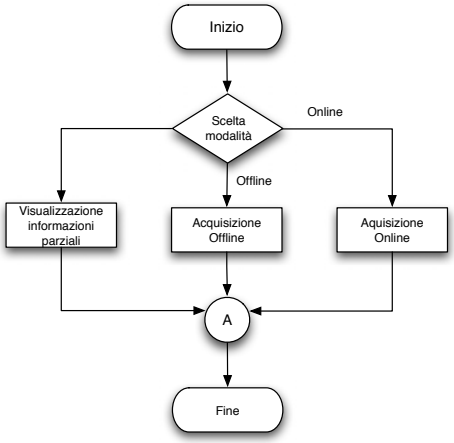


Figura 4.6: Funzionalità del programma.

paragrafi successivi illustreremo nel dettaglio le due funzionalità principali.

4.2.1 Acquisizione Online

La modalità di analisi online è la caratteristica principale di questo tool, e che lo distinge dagli altri, in quanto consente di analizzare il comportamento della rete in "tempo reale". Bisogna però precisare il concetto legato al termine "tempo reale". Con questo termine ci si aspetta che ad ogni evento che accade nella rete questo venga subito graficato, in realtà, questo non può accadere in quanto per l'analisi sono necessari un certo numero di campioni e solo al raggiungimento di tale numero vengono elaborati e visualizzati i risultati. Questo tempo è proporzionale alla precisione che si vuole ottenere all'ultimo livello dell'algoritmo piramidale (figura 4.5), ma comunque rappresenta un tempo minore rispetto all'analisi a posteriori che viene svolta dagli altri tool. Analizziamo ora come si svolge questa attività con l'aiuto del diagramma di flusso riportato in figura 4.7. Una volta scelta la modalità di analisi online comparirà una finestra di dialogo come quella riportata in figura 4.8 in cui l'utente deve inserire i parametri per l'analisi. Come si vede dalla figura 4.8 le scelte che l'utente deve effettuare riguardano:

- il tipo di link da utilizzare per l'acquisizione del traffico, in tale lista compaiono tutte le interfacce di rete che il metodo getDeviceList() riconosce.
- il tipo di filtro da applicare ai pacchetti, attualmente è possibile solo filtrare il flusso TCP o UDP.

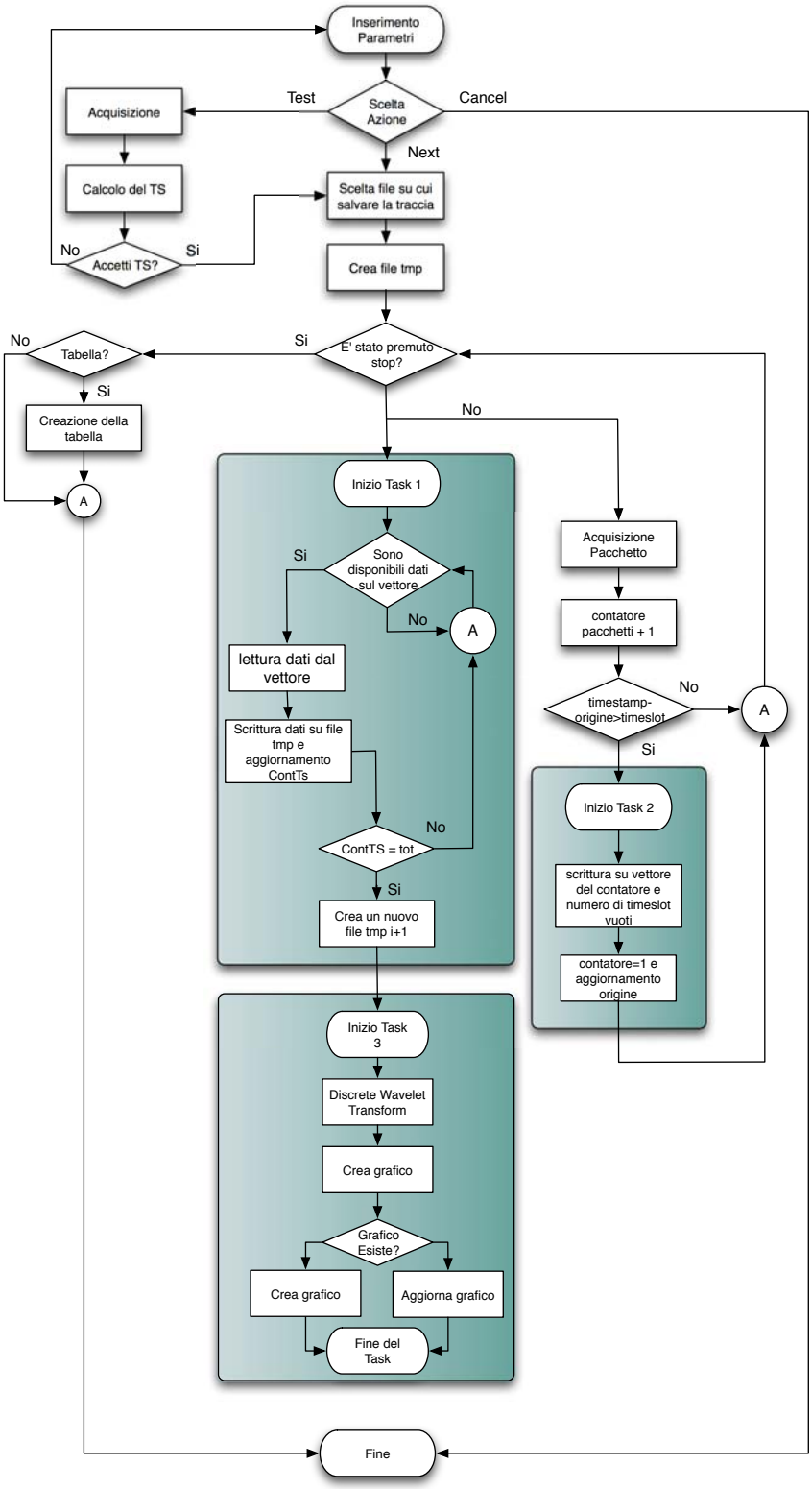


Figura 4.7: Diagramma di flusso analisi online

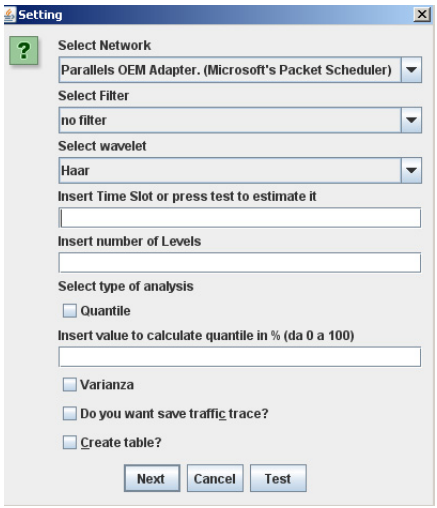


Figura 4.8: Pannello di configurazione

- il tipo di wavelet da usare nella fase di analisi. Attualmente si può scegliere tra Haar, Daubachies 4 tap e Daubachies 6 tap.
- l'inserimento del valore del time slot.
- il numero di livelli da usare nell'algoritmo piramidale.
- il tipo di analisi da effettuare che può essere scelto tra la varianza e il quantile. Nel caso della scelta del quantile bisogna inserire anche il valore percentuale del quantile che rappresenta il valore che il campione può eccedere.
- la scelta di salvare oppure no la traccia che si sta analizzando.
- la possibilità di creare la tabella con le informazioni di tutti i pacchetti.

Una volta impostati i parametri l'utente, può decidere di iniziare l'analisi oppure, come si vede dalla figura 4.8, ha la possibilità di effettuare un test. Quest'ultimo è una funzione molto importante perchè consente di determinare il valore ottimale del timeslot. Come si vede dal diagramma di flusso (figura 4.7) la fase di testing consiste in una pre-acquisizione del traffico di rete, nella quale i pacchetti non vengono salvati, per il tempo di tre minuti. Una volta terminati i tre minuti è noto il numero di pacchetti che sono transitati in tale intervallo temporale, e dalla formula che segue si ricava il valore teorico del timeslot.

$$timeslot = \frac{180 * 10}{tot} \tag{4.1}$$

In questa semplice formula 180 indica i secondi di acquisizione, 10 è il numero di pacchetti che si vogliono teoricamente per ogni time slot quando il traffico di rete è normale, mentre tot è il numero di pacchetti contati nell'intervallo di tempo del test.

Il passo successivo, dopo aver fissato il timeslot (tramite la finestra di dialogo di figura 4.8 oppure con l'aiuto del test), prevede la possibilità di selezionare il file in cui salvare i pacchetti se nella fase di configurazione si è deciso di salvare tutto il traffico che la rete genera.

Terminata la fase di impostazione ha inizio la fase di acquisizione che come si vede dal diagramma di flusso di figura 4.7 ha sempre due processi attivi, il task di acquizione e quello di scrittura su file, i quali terminano quando l'utente preme il pulsante di stop. Il processo di acquisizione rimane in ascolto sul link incrementando il contatore di pacchetti e salvandoli ogni qual volta un pacchetto viene individuato sulla rete. Per ogni pacchetto che viene catturato si estrae il suo timestamp, in quanto questo riferimento temporale, è utile per comprendere se il pacchetto in esame fa parte del timeslot corrente oppure del successivo. Se la differenza tra il timestamp e l'origine (che è uguale al valore del timestamp del primo pacchetto acquistio, e viene successivamente incrementata del valore pari ad un timeslot ogni volta che termina un timeslot) è maggiore dell'intervallo temporale fisssato dall'utente allora viene avviato un nuovo task che: nell'i-esima cella del vettore scrive il contatore di pacchetti, e nella i+1-esima cella del vettore scrive il numero di zeri da inserire tra l'ultimo valore salvato su file e il valore del contatore di pacchetti presente nella i-esima posizone. In una rete piccola e con bassa intensità di traffico potrebbe succedere che in alcuni timeslot non sia transitato alcun pacchetto di conseguenza è necessario inserire tanti zeri quanti sono i timeslot in cui non è transitato alcun pacchetto. In parallelo al processo di acquisizione è sempre attivo il task di scrittura su file dei contatori di pacchetto. Tale processo controlla se nel vettore sono presenti dei valori, se si scrive sul file i dati indicati nel vettore e aggiorna il contatore di Timeslot; altrimenti attende che nel vettore siano presenti nuovi dati. Quando questo processo ha scritto sul file un numero N di conteggi, che corrispondono agli N timeslot, crea un nuovo file temporaneo dove scrive i contatori che verranno, e avvia il processo di elaborazione che ha come input il file temporaneo appena riempito. La scelta di creare più file temporanei e conseguentemente di avviare più task, è dovuta al fatto che si vuole sfruttare al meglio la capacità delle nuove architetture parallele di gestire più attività contemporaneamente che, nel nostro caso, sono l'acquisizione, la scrittura su file e

l'elaborazione. Così facendo si riduce la possibilità di perdere pacchetti mente si stanno elaborando le informazioni precedentemente acquisite.

Ogni volta che il task di elaborazione termina fornisce i risultati necessari per la presentazione dei risultati nelle due coppie di grafici. Il tutto, come anticipato precedentemente, si ripete fino a quando l'utente non preme il pulsante di stop. Quando questo evento accade, il programma verifica se era stata attivata la creazione della tabella durante la fase di configurazione. Se la scelta dell'utente era positiva allora verrà creata la tabella altrimenti nella schermata principale rimarranno i grafici.

4.2.2 Acquisizione Offline

Come tutti i tool che caratterizzano i sistemi di monitoraggio, anche MA2T Tool (Monitoring and Analysis of Traffic Trace Tool) offre la possibilità di analizzare tracce di traffico precedentemente acquisite. Nel seguito di questo paragrafo e con l'aiuto del diagramma di flusso di figura 4.9 illustriamo il principio di funzionamento dell'analisi offline. Come si può vedere dal diagramma di flusso di figura 4.9, quando l'utente selezione l'attività di analisi offline compare una finestra di dialogo simile al pannello di configurazione della fase online (vedi figura 4.8). La differenza tra i due pannelli consiste nel fatto che nell'analisi offline non è necessario selezionare alcuna scheda di rete; e quindi una volta inseriti i parametri per l'analisi, l'utente può solo premere il pulsante per continuare oppure annullare. Il passo successivo prevede la selezione della traccia di traffico da analizzare. A questo punto il programma entra in un ciclo, che si conclude solo quando si è raggiunta la fine del file. Ad ogni iterazione del ciclo, viene estratto il timestamp del pacchetto e controllato se il suo valore è all'interno del timeslot. Se ciò avviene, allora viene incrementato il contatore di pacchetti, altrimenti viene salvato su file temporaneo e poi posto a uno, in quanto il pacchetto appartiene al nuovo timeslot; dopodiché viene aggiornata l'origine del timeslot corrente sommando un valore pari ai timeslot trascorsi. Successivamente vengono scritti tanti zeri quanti sono i timeslot rilevati dal tool, a partire dalla fine del timeslot corrente fino alla cattura del nuovo pacchetto.

Anche in questo caso, una volta che sono stati salvati i valori di N contatori, che corrispondo ad un blocco, viene avviata una fase di analisi in un nuovo processo così da consentire al processo attuale di:

- creare un nuovo file temporaneo dove salvare i contatori di pacchetti

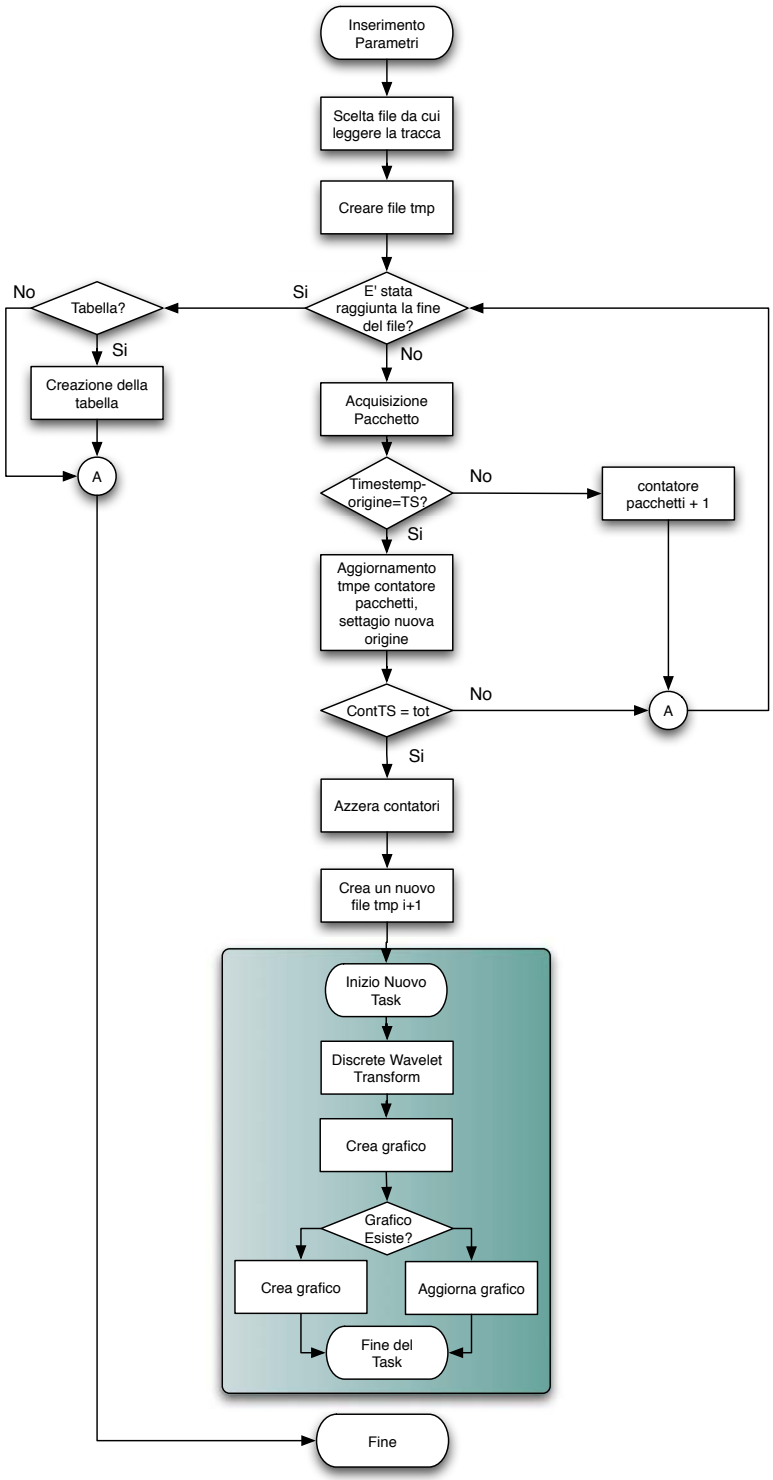


Figura 4.9: Diagramma di Flusso fase di analisi offline

```
                                Daubechies 4-tap
.332670552950   .806891509311   .459877502118   -.135011020010   -.085441273882   .035226291882
                                Daubechies 6-tap
.230377813309   .714846570553   .630880767930   -.027983769417   -.187034811719   .030841381836
.032883011667   -.010597401785
```

Figura 4.10: Coefficienti utilizzati nell'algoritmo

- continuare con la lettura della traccia di traffico.

La fase di analisi, che discuteremo con maggior dettaglio nel prossimo paragrafo, è la medesima utilizzata per l'attività di monitoraggio online. Questa attività produce in output i valori necessari alla graficazione dei risultati. Confrontando il diagramma di flusso dell'analisi online (figura 4.7) e offline (figura 4.9) si vede come sia stata utilizzata la stessa filosofia implementativa e che la differenza principale tra i due diagrammi di flusso è dovuta all'assenza della fase di testing che risulta inutile in una traccia già salvata.

4.3 Dettagli sulla fase implementativa dell'analisi

In questo paragrafo analizzeremo nel dettaglio la fase di elaborazione indicando come viene eseguita la Discrete Wavelet Transform e il calcolo del quantile o della varianza. Come abbiamo detto precedentemente, una volta raggiunti gli N campioni nella fase di acquisizione viene avviata la fase di elaborazione. La fase di analisi prevede di elaborare questi campioni con dei coefficienti. Il valore di questi coefficienti[10] varia a seconda del tipo di analisi e alla qualità che si vuole ottenere. I coefficienti che abbiamo utilizzato sono riportati nella figura 4.10: Il file di campioni prima di essere mandato in input all'algoritmo piramidale viene copiato in un vettore, dopo di che vengono creati due vettori tmpL e tmpH dove saranno contenuti i risultati parziali, i quali verranno usati per il calcolo del quantile o della varianza. In pratica la trasformata wavelet discreta è ottenuta implementando un algortimo piramidale, nel quale ad ogni livello sono presenti un filtro passa basso e un filtro passa alto; l'output del filtro passa basso rappresenta l'input del livello successivo preceduto da una fase di downsampling durante la quale dimezziamo il numero di campioni. Le operazioni a cui sono sottoposti i campioni all'interno di ogni filtro sono riassunte dalle seguenti formule:

Filtro Passa Basso:

$$t = \sum_{k=0}^{c-1} input\,[i-k] * coefficient\,[k] \tag{4.2}$$

Filtro Passa Alto:

$$t = \sum_{k=0}^{\frac{c}{2}-1} input\,[i-k] * coefficient\,[k] - input\,[i-k-1] * coefficient\,[c-k-1] \tag{4.3}$$

Nelle formule sopra indicate, il vettore input contiene i dati sui quali deve essere calcolata la DWT ad ogni livello, il vettore coefficient contiene i coefficienti da utilizzare nella moltiplicazione con i campioni, i indica il generico elemento del file di input mentre c indica la cardinalità del vettore di coefficienti. Alla conclusione di ogni iterazione di questa sommatoria viene incrementato l'indice i, così facendo si va a leggere un elemento successivo del vettore di input, e si ripete la sommatoria partendo da questo nuovo indice, mentre t viene salvato nel vettore tmpL o tmpH a seconda che si stia facendo il calcolo sul filtro passa basso o su quello passa alto, incrementando successivamente il puntatore del vettore tmpL o tmpH rispettivamente. Il tutto si ripete finchè non si è raggiunta la fine del vettore di input. Una volta fatto questo si hanno i dati sufficienti per calcolare la varianza o il quantile a seconda del tipo di analisi che l'utente vuole effettuare. Nel caso della varianza le operazioni che vengono eseguite sono le seguenti:

$$x = \sum_{j=0}^{|tmp|} (tmp\,[i] - average)^2$$

$$var = \frac{x}{|tmp|} \tag{4.4}$$

Mentre il calcolo del quantile non richiede un'elaborazione matematica vera e propria ma si svolge nei seguenti passi:

1. Ordinare i vettori temporani tmpL e tmpH;

2. Calcolare il valore dell'indice dal quale prelevare il valore ottenuto nella fase di elaborazione della DWT. Il calcolo dell'indice viene effettuato nel seguente modo:

   $$j = |tmp| * \frac{QuantileNumber}{100} + 1$$

   Dove QuantileNumber rappresenta il valore inserito dall'utente nella fase di configurazione;

3. Il valore del quantile da usare nella graficazione è *tmp[j-1]-average*.

A questo punto l'operazione di calcolo al primo livello è stata effettuata, quindi viene fatto il downsampling sul vettore tmpL eliminando le componenti pari, facendolo diventare così l'input del livello successivo e ripetendo per ogni livello le operazioni sopra descritte.

4.4 Problemi e Soluzioni

Nello sviluppo di questo tool abbiamo incontrato diverse problematiche. Nel seguito di questo paragrafo verranno illustrati questi " ostacoli" e il modo in cui sono stati superati. I problemi che hanno richiesto un'attenta valutazione per la loro risoluzione sono i seguenti:

1. Eccessiva dimensione della traccia;
2. Perdita di pacchetti durante la fase di acquisizione ed elaborazione;
3. Occupazione di memoria RAM;
4. Visualizzazione della tabella nel caso di tracce di grosse dimensioni;
5. Visualizzazione dei grafici.

Analizziamo ora nel dettaglio le varie problematiche.
Per quanto riguarda il problema della dimensione della traccia si osserva che essa cresce in funzione della quantità di traffico nella rete e del tempo. Più il traffico è intenso e più lunga è la durata dell'analisi e quindi maggiore è la dimensione del file. Per dare un esempio delle dimensioni che si possono raggiungere, consideriamo il caso di una rete a 1 Gbit di due PC. In pochi minuti la traccia di traffico raccolta da questa rete ha raggiunto la dimensione di 3 GB poichè il traffico tra questi due computer era molto intenso. Si può intuire come, in una rete strutturata e con traffico intenso, si possono ottenere tracce di traffico di grosse dimensioni soprattutto nel caso in cui le acquisizioni sono molto lunghe. Quindi per queste attività di monitoraggio è indispensabile una capacità di memoria molto elevata.

Per risolvere questo problema, L'idea iniziale, era quella di salvare i pacchetti in file di dimensione pari a quella dei blocchi; alla creazione di un nuovo blocco si eliminava il file del blocco precedente. Questa soluzione impediva all'utente di visualizzare informazioni sui pacchetti precedeti al blocco in esame. Per questo motivo abbiamo deciso di lasciare all'utente la scelta di tale azione. Come si vede

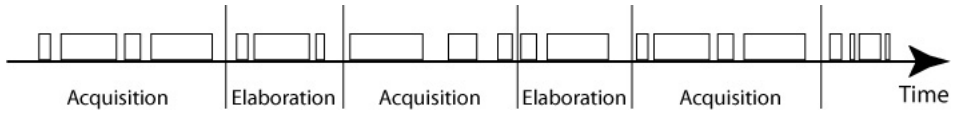


Figura 4.11: Soluzione di base: acquisizione seguita dall'elaborazione

dalla figura 4.8 l'utente deve decidere a priori se salvare la traccia oppure visualizzare solo i risultati del grafico. Così facendo l'utente può scegliere in base al tipo di analisi che vuole svolgere se salvare oppure no la traccia.

Il secondo problema è stato il più interessante da valutare dal punto di vista ingegneristico. Questo perchè trattandosi di un tool per il monitoraggio del traffico di rete in tempo reale bisognava garantire vari aspetti, tra i principali troviamo:

- velocità di risposta;
- ridurre al minimo la perdita di pacchetti;
- ridurre al minimo il tempo di elaborazione.

Per garantire questi aspetti sono state valutate varie soluzioni. Il primo approccio che abbiamo utilizzato si ispira al modo di operare degli analizzatori di spettro in tempo reale e prevedeva una fase di acquisizione pari a N timeslot seguita da una fase di elaborazione, durante la quale l'acquisizione veniva sospesa per un tempo pari alla durata dell'elaborazione. Questo approccio offriva buoni risultati per il primo e il terzo punto, mentre comportava una perdita elevata di pacchetti, soprattutto nel caso di traffico intenso, nel momento in cui si fermava l'acquisizione per effettuare l'elaborazione. In più con questa soluzione, si potevano perdere eventi interessanti se questi accadevano durante la fase di elaborazione. Per comprendere meglio questa idea si faccia riferimento alla figura 4.11.

Il passo successivo, per superare la limitazione della prima soluzione, è stato quello di creare due task, uno relativo alla fase di acquisizione e uno per l'elaborazione dei risultati. Come abbiamo visto nella descrizione dell'attività online, ogni volta che la fase di acquisizione ha salvato su file gli N blocchi chiude il file e lo passa al processo di elaborazione, mentre il processo di acquisizione apre un nuovo file su cui continuare a salvare i campioni successivi. Questa soluzione garantisce un miglioramento rispetto alla precedente, ma comporta una velocità di risposta minore nel caso di traffico di rete intenso. Infatti, potrebbe accadere che il processo di acquisizione produca i file temporanei più velocemente rispetto al tempo di calcolo del task di elaborazione, creando quindi una coda di file temporanei e

quindi ritardando il tempo di risposta. La soluzione definitiva prevede di avviare un task di elaborazione ogni qual volta il processo di acquisizione completa un file con N blocchi. Così facendo si riesce a sfruttare al meglio anche le architetture multiprocessore che attualmente sono presenti in commercio. Dai test condotti su una rete a 1Gbit ad alta intensità di traffico, abbiamo evidenziato che tale soluzione consente di soddisfare al meglio tutte le esigenze precedentemente indicate. Questa soluzione è stata implementata in due versioni: la prima prevede l'uso di timer per l'identificazione dei vari timeslot (è stata scartata per motivi di efficienza), la seconda, che è la soluzione implementata, è stata presentata nel paragrafo 3.2.1. Tale soluzione presenta una limitazione in quanto, nel caso in cui nella rete non transitino pacchetti per lungo tempo, il grafico rimane bloccato come se il tempo si fosse fermato. Solo alla ricezione di un nuovo pacchetto il "tempo perso" viene recuperato.

Il problema dell'occupazione di memoria RAM era causato dal fatto che i valori dei contatori di pacchetti di ogni timeslot non potevano essere salvati in un vettore per mantenerlo in RAM in quanto il valore N è molto grande, soprattutto nei casi in cui si vogliono analizzare più di 13 livelli. Questo numero elevato di campioni è dovuto al fatto che, in fase di progetto, abbiamo imposto di avere 64 campioni all'ultimo livello per aumentare l'accuratezza dei risultati. Poichè ad ogni livello abbiamo l'attività di downsampling, al primo livello è necessario avere $64 * 2^{level-1}$ campioni dove level è il numero di livelli che l'utente ha impostato nella fase di configurazione.
Per risolvere questo inconveniente, abbiamo deciso di salvare il valore di ogni contatore in un file temporaneo il quale poi viene letto dal task di elaborazione. Questa soluzione richiede un elevato numero di accessi al disco in quanto ad ogni timeslot il contatore viene scritto su file; si tratta di un compromesso che dobbiamo accettare in quanto il salvataggio di tali valori su un vettore per poi copiarlo su disco, poteva comportare perdite di dati durante l'analisi online.

Il quarto problema, è legato alla visualizzazione dei dettagli sui pacchetti contenuti in tracce di traffico di grosse dimensioni. Infatti, è impossibile visualizzare tutte queste informazioni in quanto non tutta la traccia può stare in RAM. Per risolvere questo inconveniente abbiamo lasciato all'utente la possibilità di decidere se visualizzare i dettagli dell'intera traccia oppure visualizzare solo la porzione di traccia a cui è interessato mediante l'apposita funzione.
L'ultimo problema è legato alla scelta del modo in cui rappresentare i risultati sul grafico. Alla soluzione di questo " ostacolo " è dedicato l'intero prossimo capitolo.

4.5 Analisi Online alternativa

Come anticipato precedentemente, l'algoritmo che verrà spiegato in questo paragrafo non è stato utilizzato nella versione definitiva del tool perchè garantiva un buon comportamento solo per valori di timeslot maggiori di 0,5 secondi. Il motivo di questa limitazione sulla risoluzione è da imputare agli accessi del timer all'orologio di sistema per osservare lo scorrere del tempo.

Spieghiamo ora con l'aiuto del diagramma di flusso di figura 4.12 il principio di funzionamento di tale soluzione. Dopo la fase di configurazione del tool, spiegata nella prima parte del paragrafo 3.2.1, inizia la fase di analisi.

La fase di acquisizione che inizia con l'avvio di un timer il quale ha la funzione di contare fino al valore del timeslot. Nello stesso momento in cui si avvia il timer, il processo di acquisizione rimane in ascolto sul link incrementando il contatore di pacchetti e salvandoli ogni qual volta un pacchetto viene trasferito. Allo scadere del timer, questo preleva il valore del contatore e lo salva in un file temporaneo, creato all'inizio dell'acquisizione. Successivamente il contatore viene reinizializzato e viene avviato nuovamente un timer. Il tutto continua fino a quando l'utente non preme il pulsante di stop che indica la conclusione dell'analisi. Continuando l'analisi del nostro diagramma di flusso (figura 4.12) si vede come dopo aver salvato sul file un certo numero di contatori pari ad N, venga avviato il processo di analisi che elaborerà questi campioni usando l'algoritmo piramidale. Al momento dell'avvio del processo di elaborazione viene creato un nuovo file temporaneo (i motivi della creazione di più file temporanei sono stati spiegati nel paragrafo 4.2.1) nel quale salvare i valori dei contatori successivi. Ogni volta che il processo di elaborazione termina fornisce i risultati necessari per la graficazione nelle due coppie di grafici. Il tutto viene ripetuto fino a quando l'utente non preme il pulsante di stop. Quando questo evento accade il programma verifica se era stata attivata la creazione della tabella durante la fase di configurazione. Se la scelta dell'utente era positiva allora verrà creata la tabella altrimenti nella schermata principale rimarranno i grafici. Come abbiamo visto il punto cruciale dell'algoritmo è il timer che identifica la conclusione di ogni timeslot. Questo approccio falsava i risultati in quanto allo scadere del timer questo doveva scrivere su file il contatore di pacchetti, creare, se necessario, un nuovo file e avviare la fase di elaborazione. Tutte operazioni che richiedono tempi di esecuzione elevati creando quindi delle code di timer che dovevano essere serviti appena possibile, rallentando quindi l'intero tool.

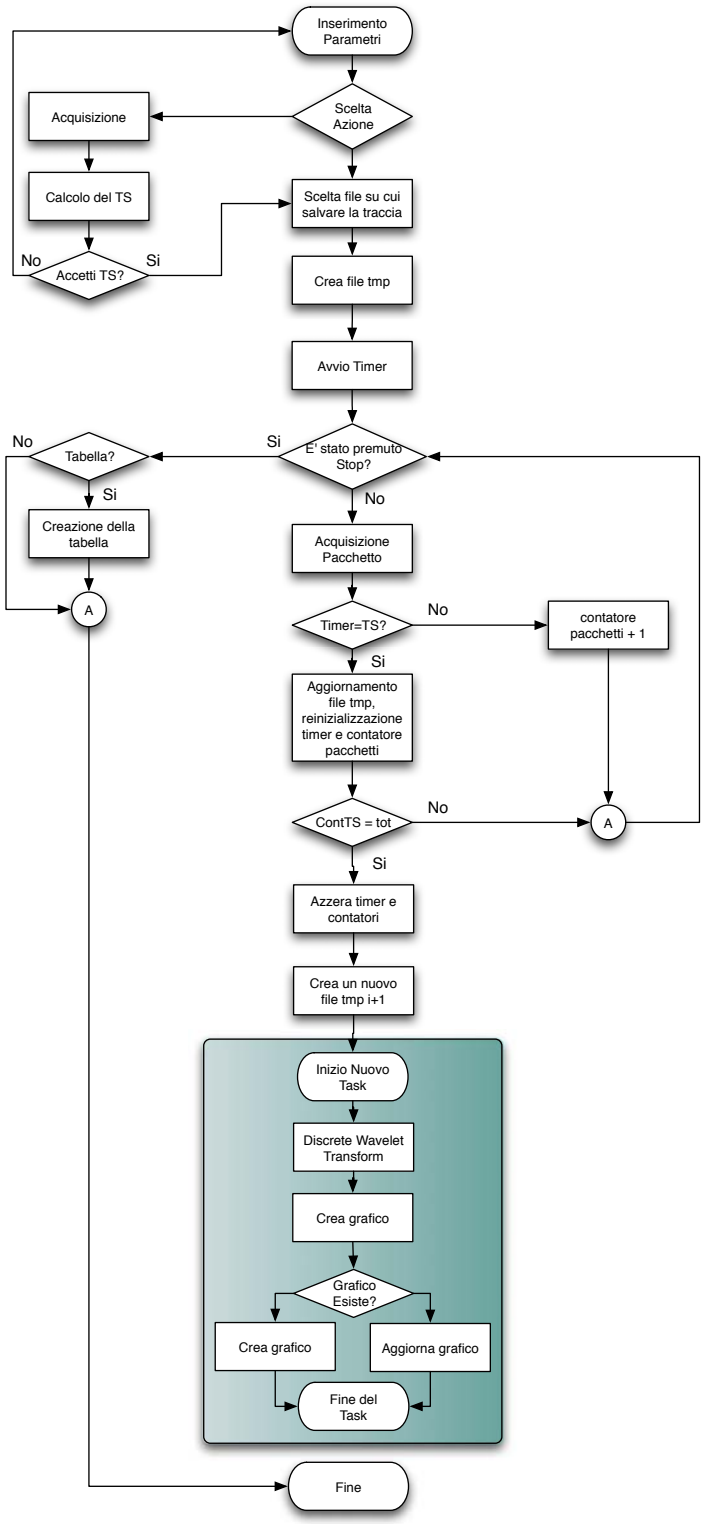


Figura 4.12: Diagramma di flusso analisi online

Capitolo 5

Grafici e la loro Interpretazione

Come abbiamo visto nel paragrafo precedente, uno dei problemi che abbiamo dovuto affrontare è legato alla rappresentazione dei risultati per via grafica.
Come vedremo nei paragrafi successivi per consentire l'estrazione del maggior numero di informazioni, abbiamo deciso di visualizzare i risultati su due coppie di grafici:

- Blocchi - Valore
- Livelli - Valore

La coppia è costituita dal grafico di dettaglio e dal grafico di approssimazione.

5.1 Grafico Blocchi- Valore

Questo tipo di grafico è caratterizzato nel visualizzare i risultati utilizzando la coppia di punti (blocco-valore). Lo scopo di questa rappresentazione è quello di mostrare, anche se in modo indiretto, il tempo per dare un riferimento temporale di quando avvengono gli eventi. Come possiamo vedere dalla figura 5.1 nelle ordinate troviamo il valore del punto calcolato in fase di elaborazione e può essere il quantile o la varianza. Nelle ascisse troviamo il blocco in cui sono stati elaborati i campioni. Ricordiamo che un blocco è una sequenza consecutiva di timeslot. Come si vede in figura 5.1 vengono visualizzati contemporaneamente 25 blocchi e sono contenute più curve. Ogni curva rappresenta l'andamento dei valori ad un preciso livello, dell'algoritmo piramidala, al passare del tempo.

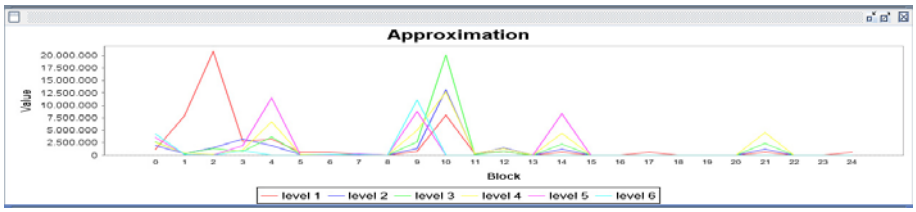


Figura 5.1: Grafico blocchi-valore

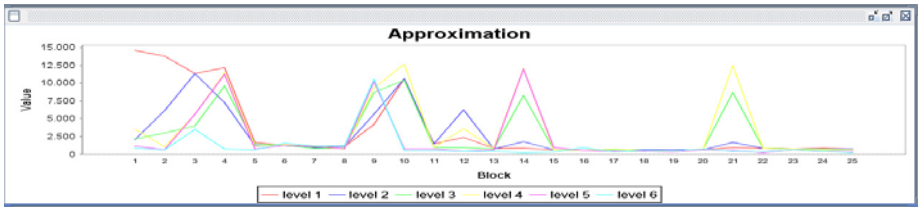


Figura 5.2: Grafico Blocchi aggiunta valore

5.1.1 Implementazione

Per realizzare questi grafici abbiamo utilizzato JFreeChart [11, 12]. JFreeChart è una libreria di grafici distribuiti gratuitamente e permette di visualizzare in modo chiaro i risultati di varie applicazioni. Come abbiamo anticipato nella figura 4.2, la classe che permette la realizzazione dei grafici è *GraphicTime.java*. Tale classe riceve in input un vettore contenente i valori calcolato ad ogni livello dell'algoritmo piramidale. A questo punto vengono aggiunti i nuovi punti a destra del grafico e tutti gli altri vengono traslati a sinistra di una posizione in modo tale da visualizzare solo gli ultimi 25 valori. Per comprendere meglio il funzionamento consideriamo la figura 5.1 in cui sono visualizzati 25 blocchi. Quando sono pronti i risultati degli N timeslot successivi, viene aggiunto il nuovo blocco come mostrato in figura 5.2

5.1.2 Interpretazione

Lo scopo di questa rappresentazione è quello di consentire all'utente l'identificazione di anomalie nella rete in funzione del tempo. Poichè ogni blocco è costituito da N timeslot, il cui valore è definito dall'utente, siamo riusciti a rappresentare il tempo, anche se indirettamente, sulle ascisse. Grazie a questa rappresentazione le anomalie vengono rappresentate da dei picchi. Una volta identificati gli istanti in cui si sono verificati dei problemi, è possibile risalire alla fonte dell'anomalia valutando nel dettaglio la porzione di pacchetti che ha creato il picco nel grafico. Ciò è reso possibile in quanto il tool permette di visualizzare una tabella con i parametri

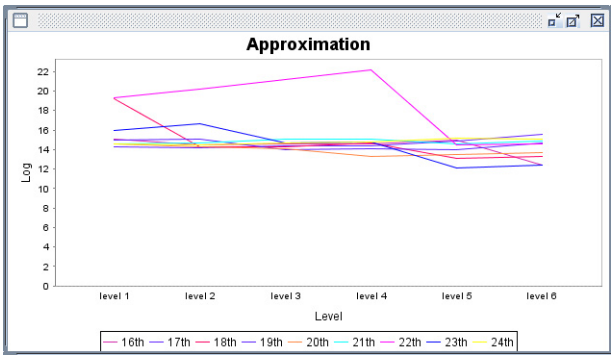


Figura 5.3: Grafico Livelli-Valore

fondamentali di un pacchetto quali timeslot, indirizzi ip sorgente e destinazione e il tipo di protocollo.

5.2 Grafico Livelli - Valore

La seconda tipologia di grafici è sempre costituita dal diagramma di approssimazione e dettaglio. In questo caso a differenza del caso precedente, il grafico è caratterizzato dall'avere nelle ascisse i livelli e nelle ordinate il valore del quantile o della varianza. Anche in questo grafico vengono visualizzate più curve contemporaneamente; per scelta ne rappresentiamo solo dieci, e ogni curva rappresenta un blocco di N campioni.

5.2.1 Implementazione

Anche questa coppia di grafici è stata realizzata mediante JFreeChart [11, 12]. Il principio alla base della realizzazione delle curve è del tutto simile a quello dei grafici precedenti. Infatti, anche in questo caso, il metodo per la rappresentazione delle curve riceve in input un vettore in cui ogni cella è contenuto il valore del quantile o della varianza precedentemente calcolato e rappresenta la coordinata y, mentre l'indice del vettore rappresenta la coordinata x. Quindi i punti che compongono una curva sono la coppia indice j e array[j] cioè della j-esima locazione del vettore. Come abbiamo detto precedentemente, ci sono sempre dieci curve contemporanemante visualizzate, quindi all'arrivo di una nuova curva viene eliminata quella più vecchia. Mediante le figure 5.3 e 5.4 è possibile vedere la fase di sostituzione delle curve.

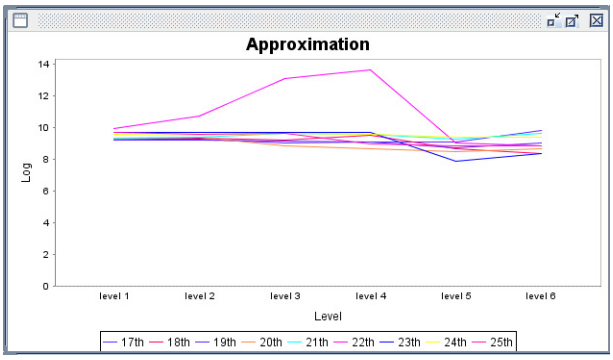


Figura 5.4: Grafico Livelli-Valore con l'aggiunta della curva 25

5.2.2 Interpretazione

Lo scopo di questa rappresentazione è quello di identificare il grado di aggregazione del traffico e quindi il valore del parametro di Hurst che può essere determinato in modo approssimativo dalla pendenza delle curve visualizzate nel grafico.

Capitolo 6

Analisi delle tracce

In questo capitolo verranno presentate delle analisi di monitoraggio di traffico realizzate utilizzando il tool finora descritto.

La prima traccia di traffico che analizzeremo rappresenta un flusso di traffico controllato: essa è stata realizzata monitorando una rete in cui tutte le attività di accesso ad internet erano tenute sottocontrollo. Tale prova consisteva nel generare del semplice traffico ottenuto attraverso l'accesso a pagine internet con l'aggiunta di singoli eventi, non sovrapposti nel tempo, quali streaming video, chiamate skype, download etc ...

La seconda traccia che verrà analizzata in questo capitolo mostra l'andamento del traffico reale della rete relativa ad un piano del nostro dipartimento. Questa traccia contiene il traffico di una intera giornata, senza alcun vincolo imposto, però non è possibile per noi fare delle ipotesi approfondite sul tipo di traffico relativo, in quanto il payload dei pacchetti è stato oscurato per motivi di privacy.

6.1 Traffico Controllato

6.1.1 Struttura rete

Per eseguire la misura sul traffico controllato, abbiamo utilizzato un computer Desktop con le seguenti caratteristiche:

- Sistema Operativo: Microsoft Windows XP Professional Service Pack 2.
- CPU: Intel Core 2 Duo con frequenza 2.13GHz.
- RAM: 2GB.

- Hard Drive: SATA WD 160GB a 7200rpm.
- Scheda di Rete: Ethernet Broadcom NetExtreme 57xx GBit.
- Scheda di Rete: Ethernet PCI SURECOM EP 320X-R 10/100.

Esso fungeva da strumento di misura in quanto eseguiva il tool. Il traffico di rete sotto esame è stato generato da due computer, collegati tramite switch alla connessione internet condivisa dallo strumento di misura.

6.1.2 Descrizione risultati

La traccia di traffico controllato ha una durata di un'ora e trenta minuti, e ha lo scopo di evidenziare il comportamento di eventi particolari avviati in momenti predefiniti.

L'analisi che andiamo ad presentare è stata eseguita configurando il tool nel seguente modo:

- Timeslot: 0.05 secondi
- Numero di Livelli: 6
- Wavelet selezionata: Daubechies 6 tap
- Analisi Selezionata: Quantile 99%

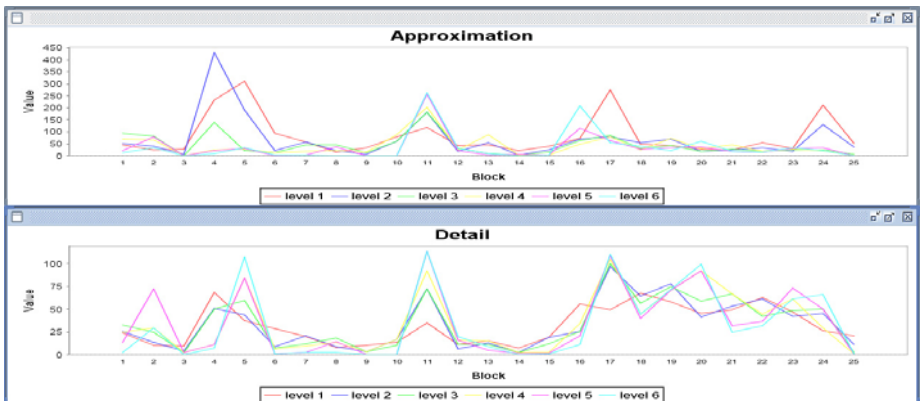


Figura 6.1: Grafico rappresentante il traffico analizzato tra i blocchi 1 e 25 con quantile al 99%

Come si può vedere dalla figura 6.1 l'andamento del traffico (primi 25 blocchi) presenta dei picchi in corrispondenza di particolari eventi che in questo caso sono:

- Dal blocco 3 al 5 streaming video;
- Dal blocco 9 al 13 chiamata skype con video;
- Dal blocco 16 al 24 download di un file da 1GB tramite protocollo http;

Gli intervalli di tempo tra gli eventi sopra citati rappresentano l'andamento del traffico durante una semplice navigazione tra pagine web. Si può notare che nei blocchi che vanno dal 16 al 24 (attività di download) esiste una differenza tra il grafico di approssimazione e quello di dettaglio: nel grafico di approssimazione è infatti possibile individuare immediatamente le attività di inizio e fine download, in quanto esse sono rappresentate da due picchi ben visibili; nel grafico di dettaglio si può invece notare che vi è un inalzamento del valore medio dei vari livelli per tutta la durata del download. Un'ulteriore informazione che si può ricavare, è che l'attività di streaming crea un picco con valore superiore rispetto alle altre attività.

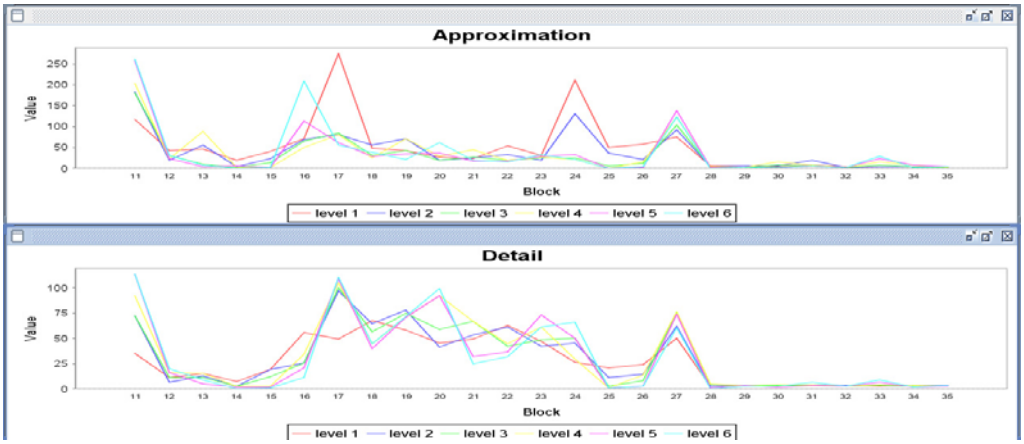


Figura 6.2: Grafico rappresentante il traffico analizzato tra i blocchi 11 e 35 con quantile al 99%

In figura 6.2 è mostrata l'evoluzione dei grafici in tempi successivi a quelli di figura 6.1. Dalla figura 6.2, si evince che nel blocco 26 ha avuto inizio un'attività che ha generato un picco di piccola entità: tale picco rappresenta l'attività di avvio connessione per una sessione di gaming online. Tale attività, che si protrae sino al blocco 45 (Figura 6.3), dopo il picco iniziale prosegue con valori dei coefficienti molto bassi. Ciò secondo noi è dovuto al fatto che questo tipo di attività richiede di ridurre al minimo il trasferimento di informazioni per non compromettere la fluidità del gioco. Infine, dal blocco 47 al 52, è stato avviato il software Google Earth con il quale è stata eseguita la ricostruzione 3D della città di New York. Tale

attività, come nel caso del download, comporta l'inalzamento del valore medio nelle curve sul grafico di dettaglio, mentre il grafico di approssimazione evidenzia un unico picco nella fase iniziale di connessione.

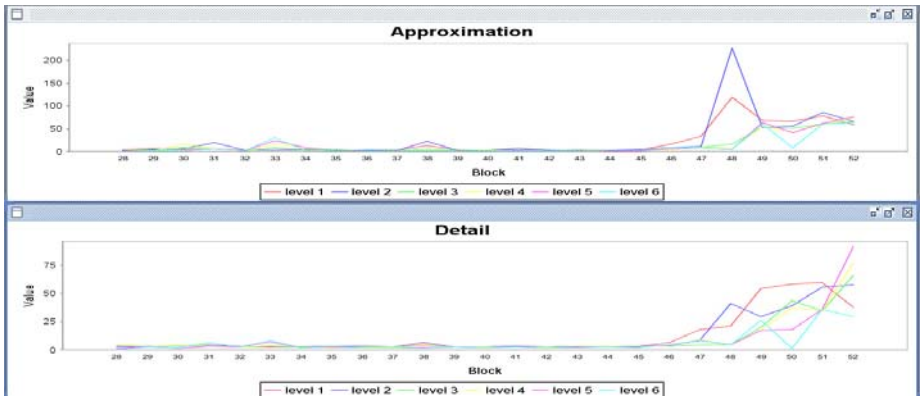


Figura 6.3: Grafico rappresentante il traffico analizzato tra i blocchi 28 e 52 con quantile al 99%

6.2 Analisi traccia DEI

La traccia che andremo ad analizzare in questo paragrafo, rappresenta il traffico di rete relativo ad un intero piano del Dipartimento di Ingegneria dell'Informazione dell'Università di Padova. La durata della traccia in esame è di 24 ore.

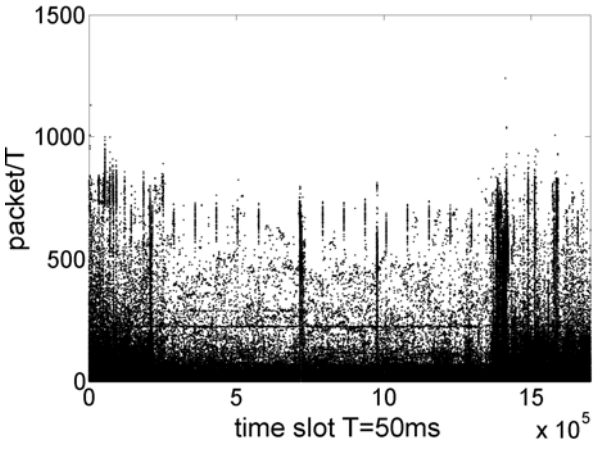


Figura 6.4: Grafico rappresentante l'intera traccia acquisita.

Come si può vedere dalla figura 6.4, che rappresenta sulle ascisse il numero

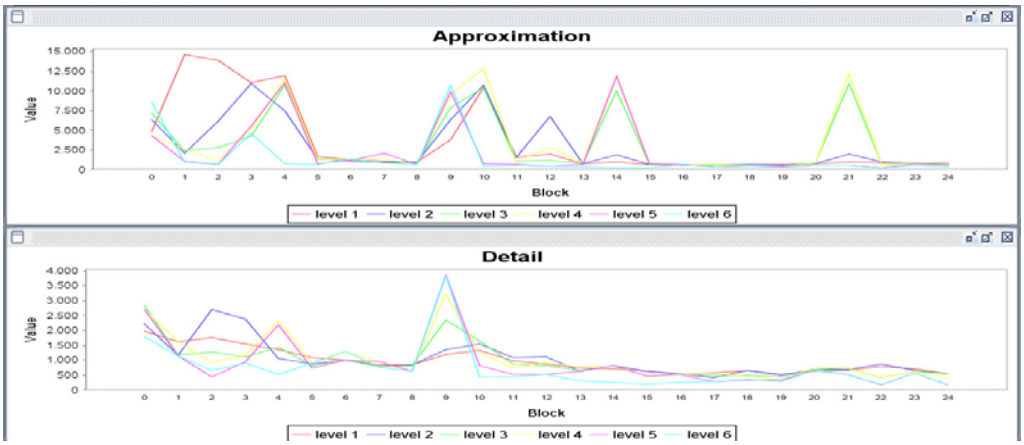


Figura 6.5: Grafico rappresentante il traffico analizzato tra i blocchi 0 e 24 con quantile al 99%

di timeslot mentre sulle ordinate il numero di pacchetti catturati per timeslot, il traffico è concentrato agli estremi della figura, in quanto la parte centrale del traffico rappresenta le ore notturne durante le quali molte attività erano ovviamente sospese. L'analisi che andiamo a presentare è stata eseguita configurando il tool nel seguente modo:

- Timeslot: 0.5 secondi, tale valore è di dieci volte superiore al timeslot utilizzato nell'analisi precedente perché il traffico in esame presenta caratteristiche differenti. Tale valore è stato ottenuto mediante la fase di test prevista dal tool.
- Numero di Livelli: 6
- Wavelet selezionata: Daubechies 6 tap
- Analisi Selezionata: Quantile 99%

Come si può vedere dalla figura 6.5, è presente un'attività rilevante dal blocco 0 al blocco 15: tale periodo rappresenta infatti il traffico acquisito dalle ore 16 alle ore 20 circa, durante il quale erano presenti attività lavorative.

Nei blocchi successivi si può notare un andamento caratterizzato da valori molto bassi intervallati da tre picchi: essi sono individuabili in corrispondenza dei blocchi 21, 28, 35 in figura 6.6.

Tali fenomeni, secondo la nostra interpretazione, potrebbero essere dovuti ad attività di sincronizzazione o backup da parte dei server. Purtroppo, però, una corretta interpretazione non può essere data, in quanto la traccia è stata offuscata per

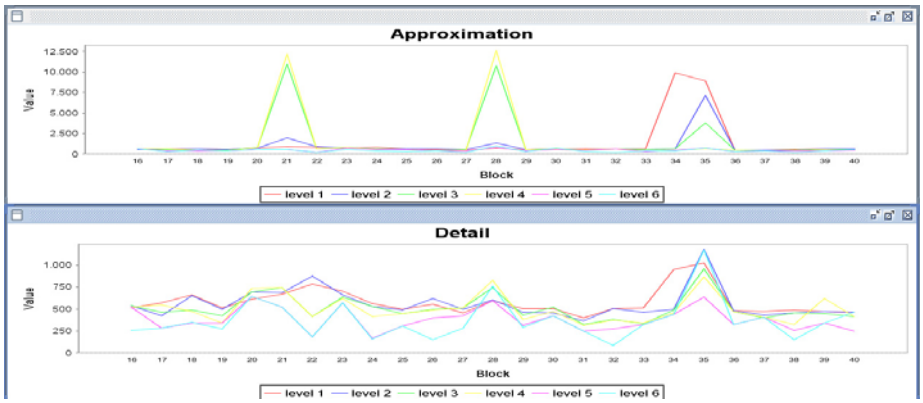


Figura 6.6: Grafico rappresentante il traffico analizzato tra i blocchi 16 e 40 con quantile al 99%

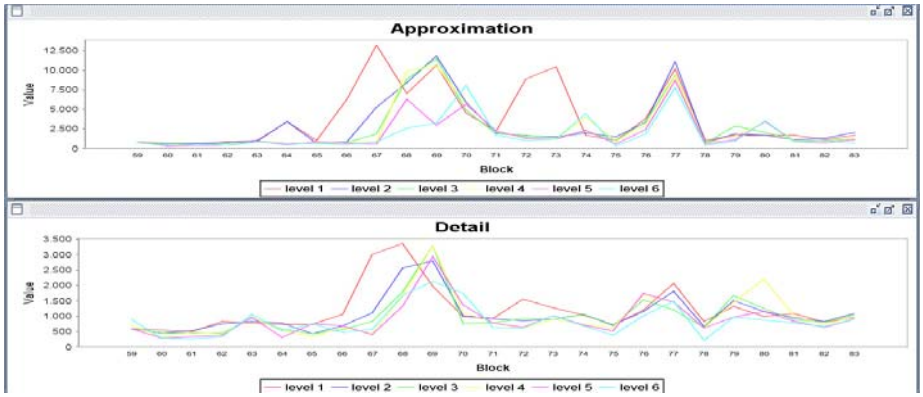


Figura 6.7: Grafico rappresentante il traffico analizzato tra i blocchi 59 e 83 con quantile al 99%

garantire la privacy degli utenti della rete. Sempre in figura 6.6, nel grafico di dettaglio, si possono osservare dei picchi di piccola entità: ciò è dovuta al fatto che il tool dispone di una funzione di auto-zoom, la quale cambia la scala del grafico per garantire una migliore comprensione. Si fa quindi notare che il valore massimo del grafico di dettaglio è 1000 mentre il valore massimo del grafico di approssimazione è 12500: ciò indica che il grafico di dettaglio rappresenta in sostanza il rumore di fondo.

Infine con la figura 6.7, si evidenzia la ripresa dell'attività lavorativa in corrispondenza del blocco 63, dal quale il valore medio delle curve incomincia a crescere; ciò sta ad indicare una maggiore attività di rete. Ripetendo l'analisi con diverse tipologie di Wavelet non si sono rilevate evidenti differenze a parte lie-

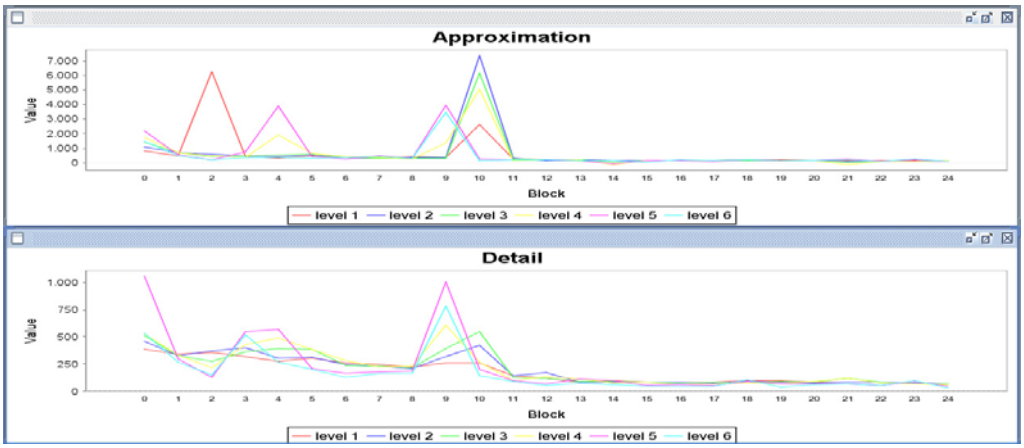


Figura 6.8: Grafico rappresentante il traffico analizzato tra i blocchi 0 e 24 con quantile al 90%

vi discrepanze nei valori raggiunti. Per questo motivo tali analisi non vengono riportate.

6.3 Confronto tra Quantile 90% e Quantile 99%

In questo paragrafo verrà effettuato il confronto tra due analisi fatte con due differenti valori del quantile, ovvero 90% e 99%. La traccia sotto esame è sempre quella relativa al traffico acquisito nella rete del dipartimento di ingegneria. La differenza principale tra queste due tipologie di analisi sta nel fatto che, con il quantile al 90% non vengono presi in considerazione eventi sporadici, mentre con il quantile al 99% vengono rilevati tutti gli eventi che accadono nella rete.

Confrontando la figura 6.5 con la figura 6.8, si può notare come il quantile al 90% mostri dei picchi associati alle attività lavorative, mentre vengono tagliati tutti gli eventi successivi che, come si può vedere dalla figura 6.6 sono eventi isolati.

Una ulteriore osservazione derivata dal confronto tra le figure 6.6 e 6.9, riguarda il fatto che il grafico con il quantile al 90% taglia tutti gli spike tranne quello individuato nel blocco 35 che, evidentemente, rappresenta un'attività rilevante.

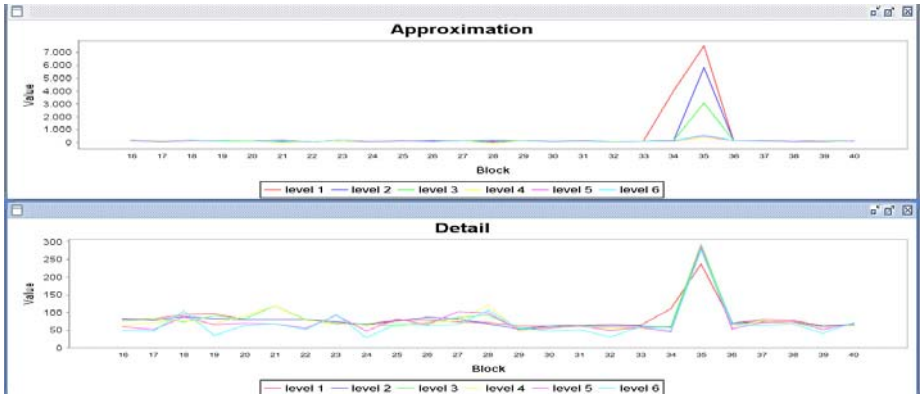


Figura 6.9: Grafico rappresentante il traffico analizzato tra i blocchi 16 e 40 con quantile al 90%

Capitolo 7

Conclusioni

Questo lavoro di tesi, mirato alla realizzazione di un tool per il monitoraggio e l'analisi del traffico di rete, è stato molto interessante sia dal punto di vista implementativo che da un punto di vista teorico. Quest'ultimo aspetto ci ha portati ad approfondire alcune nozioni già incontrate nel corso degli studi quali il modello ISO-OSI, il funzionamento delle reti; gestione dei task etc... ma ci ha portato anche allo studio di nuovi argomenti quali la wavelet e la relativa trasformata. Dal punto di vista pratico, questo lavoro ci ha permesso di seguire tutte le fasi di uno sviluppo software partendo dalla fase di progettazione fino ad arrivare alla fase di testing. Quest'ultima ha permesso di evidenziare sempre nuove migliorie da apportare al tool consentendo di affinare quindi lo strumento di analisi rendendolo efficiente per l'attività di monitoraggio in tempo reale. Come discusso in precedenza, l'attività di acquisizione online è stata la fonte delle maggiori problematiche, sia in fase di progettazione, sia nelle successive fasi di sviluppo.
In fase di progettazione sono state vagliate differenti soluzioni per implementare l'acquisizione e l'analisi in tempo reale, delle quali sono state implementate le due considerate più promettenti. Nella fase di sviluppo, dopo una serie di test, abbiamo evidenziato che la prima soluzione implementata, che utilizzava dei timer, risultava efficiente solo con timeslot superiori a 0.5 secondi. Quindi, per tale motivo, la soluzione definitiva utilizza quattro task: il primo si occupa di acquisire pacchetti, il secondo scrive i dati su un'apposita struttura dati mentre il terzo gestisce la lettura ordinata di tali dati e lancia il task che esegue la Discrete Wavelet Transform.

Essendo il tool uno strumento di ricerca non era chiaro fin dall'inizio come rappresentare i risultati dell'analisi. Serviva infatti, un modo che rendesse possibile una rapida stima del livello di aggregazione, ma nel contempo riuscisse a dare

una stima dell'andamento del traffico di rete osservato, rendendo così possibile l'individuazione di eventuali anomalie. La scelta finale è ricaduta sulla coppia di grafici spiegata nei capitoli precendeti. A questo punto abbiamo un tool che consente la visualizzazione del traffico in tempo reale e che, grazie all'utilizzo del quantile risulta più robusto dell'A-V Tool.

Ma2t tool presenta ampie possibilità di evoluzione. Una prima miglioria da apportare nelle versioni successive è quella di confrontare l'andamento del traffico con delle curve "teoriche" che identifichino il range entro il quale il traffico è nella norma, in modo tale da facilitare ulteriormente l'individuazione di anomalie. Un ulteriore sviluppo potrebbe essere la graficazione, mediante istogramma, del tipo di traffico che transita nella rete, in altre parole mostrare, attraverso un'analisi della distribuzione statistica, all'utente se la maggior parte del traffico è internet oppure ftp o altro.

Appendice A

Codici Sorgente

A.1 Ma2ttool.java

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import jpcap.*;
import java.io.File;
import javax.swing.JFrame.*;
import java.io.*;
import jpcap.packet.Packet;
import java.math.*;
import java.io.FileOutputStream;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;
import java.util.*;



public class ma2ttool extends JFrame implements ActionListener
{
private final int ITEM_PLAIN =  0;
private final int ITEM_CHECK=   1;
private final int ITEM_RADIO=   2;
// Variabile per il la creazione del menu nel pannello
private JMenuBar menuBar;
// Variabile per la creazione del menu File
private JMenu menuFile;
// Variabile per la creazione del meni Action
private JMenu menuAction;
// Item Stop contenuto nel menu Action
private JMenuItem menuStop;
// Item GoTo contenuto nel menu Action
```

```
private JMenuItem menuGoTo;
// Pulsante del menu Help
private JMenu menuHelp;
// Menu Acquisizione contenuto in File
private JMenu menuFileAcquisizione;
private JMenuItem menuFileEsporta;
// Item Exit per chiudere il tool
private JMenuItem menuFileExit;
private JMenuItem menuFilePrint;
// Item per avviare l'analisi in tempo reale
private JMenuItem menuFileOnline;
// Item per avviare l'analisi di una traccia acquisita precedentemente
private JMenuItem menuFileOffline;
private JOptionPane dialog1 = new JOptionPane();
private JPanel panel=new JPanel(new BorderLayout());
/*variabile per la creazione della finestra di dialogo per l'analisi
0ptonline*/
private CustomDialog pannello;
/* Variabile per la creazione della finestra di dialogo dopo la fase di
0pttest*/
private CustomDialog pannello2;
/* Variabile per la creazione della finestra di dialogo per l'analisi
0ptoffline*/
private CustomDialog pannello3;
// Variabile per la creazione della tabella
private Table newContentPane;
// Variabile per la creazione dei tab
private JTabbedPane tabbedPane;
// Contatore utilizzato per la numerazione dei Tab
private int counter=0;
// Puntatore al tipo di scheda di rete da utilizzare per l'acquisizione
0ptdel traffico
private int selectedValue;
// Puntaotre al tipo di Wavelet da usare nell'analisi
private int selectedWavelet;
// Variabile che contiene il numero di livelli dell'algoritmo piramidale
private int selectedLevel;
// Indica il numero di slot
private int slot=0;
//conta i pacchetti nell'acquisizione online
private int contserie=0;
/* Variabile che contiene la dimensione del del numero  di schede di rete
0pt presenti nel computer */
private int i;
//l'indice di posizione del vettore del grafico
private int index=0;
/* numero di curve da visualizzare contemporaneamente  sul grafico in cui
0ptnell'asse x sono contenuti i livelli */
private int finestra=10;
// numero di curve attuali
private int saturation=0;
```

```
/* Variabile utilizzata come flag per distinguere le operazioni da
0pteffettuare nel grafico alla prima itereazione  rispetto alle iteraioni
0ptsuccessive */
private int jump=0;
// Contiene il valore del quantile per l'analisi
private int quantileNumber;
//viene posto a uno in caso di errore nel'inserimento dei parametri da
0ptparte dell'utente
private int flagErrorInput=0;
//indica l'ultimo byte letto nel randomAcessFile
private int contByte=0;
// parametro che contiene il numero di timeslot vuoti dall'ultimo
0ptpacchetto arrivato a precedente
private int tw=0;
//variabile contenente la scelta nella finestra di dialogo nella finestra
0ptdi dialogo per la scelta del file
private int returnVal;
// Contiene il valore del blocco iniziare per la costruzione della tabella
private int gotoStart;
// Contiene il valore del blocco finale per la costruzione della tabella
private int gotoStop;
// Contiene il numero di livelli per la costruzione della tabella
private int gotoLevel;
/*Numera i file da dare in ingresso all'algoritmo piramidale*/
private int p=0;
private long sec=0;
private long usec=0;
private double tstamp=0;
private int controw=0;
private int vector[] = new int[10000];
private int read=0;
private int write=0;
private boolean finish=false;
// Indica il tipo di filtro selezionato dall'utente
private Object selectedFilter=new Object();
// Oggetto che indica se e' stato premuto il pulsante stop
private Object selectedValu=new Object();
// Oggetto che contiene il valore del timeslot inserito dall'utente
private Object selectedText;
// Contiene il valore del timeslot inserito dall'utente
private double timeSlot=1;
// vettore che contiene i risultati dell'analisi DWT
private double ris[];
/* Indica il numero di campioni necessari al primo livello  per l'analisi
0ptdell'algoritmo piramidale */
private double sampleNumber=0;
// Indica il raggiungimento di N TimeSlot
private double Tot;
/* contiene il valore del Timeslot necessario per la creazione della
0pttabella parziale */
private double gotoTS;
```

```
/* Viene settata a true quando l'utente preme Next oppure termina la fese
0ptdi testing per la scelta del timeslot*/
private boolean go=false;
/* Se vale false viene utilizzato il valore del timeslot inserito dall'
0ptutente altrimenti utilizza il valore del timeslot calcolato nella fase
0ptdi testing */
private boolean set=false;
/*Indica se eseguire l'analisi con la varianza*/
private boolean flagVar;
/*Indica se eseguire l'analisi con il quantile*/
private boolean flagQuan;
/*Se il valore e' true l'ordinamento dei byte e' in little endian
0ptaltrimenti l'ordinamento e' in big endian*/
private boolean indian=true;
/*Variabile di controllo booleana che riguarda la scelta del file*/
private boolean chooser=false;
/* Viene posta a true quando il timer raggiunge la fine della pre-
0ptacquisizione e quindi smette di catturare pacchetti e calcola il valore
0ptdel timeslot consigliato*/
private boolean _timerTest=false;
/* Viene posta a true quando viene eseguito la fase di testing per il
0ptcalcolo del timeslot*/
private boolean flagTest=false;
/* Viene posta a true quando l'utente decide di salvare il file*/
private boolean saveFile=false;
/* Viene posta a true quando l'utente decide di visualizzare la tabella
0ptdell'intera traccia*/
private boolean doTable=false;
/* Contiene il valore del timeslot calcolato durante una preanalisi */
private long timeTest=0;
/* Creazione dei grafici */
private Graphic _graphicL;
private Graphic _graphicH;
private GraphicTime     _graphicTimeL;
private GraphicTime     _graphicTimeH;
/* Crea il file su cui scrivere il numero di pacchetti contati in ogni
0pttimeslot*/
private File tmp;
/* Scrive sul file tmp creato*/
private FileOutputStream file;
private PrintStream out;
private JFileChooser fileChooser1;
private File output;
/*variabile per leggere una traccia acquisita precedentemente*/
private RandomAccessFile f1;
/*Variabili per la creazione dei timer per il conteggio del tempo*/
private TimerTask timerTest;
private java.util.Timer timer = new java.util.Timer();
private java.util.Timer timer1 = new java.util.Timer();
/*Variabile che permette di scrivere i pacchetti acquisiti con la libreria
0pt JPCAP su file*/
```

```
private JpcapWriter writer;
/*Vettore per memorizzare le schede di rete individuate nel computer dalla
0pt libreria JPCAP*/
private NetworkInterface[] devices;
public ma2ttool()
{
 /* Impostazioni del frame quali nome dimensione e chiusura */
 setTitle("MA2T tool");
 setSize(1260,974);
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 /*  crea il menu bar */
 Container c= getContentPane();
 c.setLayout(new BorderLayout());
 menuBar = new JMenuBar();
 JFrame.setDefaultLookAndFeelDecorated(true);
 /* setta l'istanza come applicazione del menu bar */
 setJMenuBar(menuBar);
 /* creazione del sottomenu file con la relativa abbreviazione da tastiera
 0pt */
 menuFile = new JMenu("File");
 menuFile.setMnemonic('F');
 menuBar.add(menuFile);
 /* creazione degli oggetti del menu file */
 menuFileAcquisizione = new JMenu("Acquisition..");
 menuFile.add(menuFileAcquisizione);
 menuFileAcquisizione.setMnemonic('A');
 menuFileEsporta = CreateMenuItem(menuFile, ITEM_PLAIN,"Export",null, 'E',
 0pt "Export");
 menuFilePrint = CreateMenuItem(menuFile, ITEM_PLAIN,"Print",null, 'P', "
 0ptPrint");
 menuFileExit = CreateMenuItem(menuFile, ITEM_PLAIN,"Quit",null, 'x', "
 0ptExit");
 menuFileOnline = CreateMenuItem(menuFileAcquisizione , ITEM_PLAIN,"Online
 0pt",null, 'O', "Online");
 menuFileOffline= CreateMenuItem(menuFileAcquisizione , ITEM_PLAIN,"Offline
 0pt",null, 'f', "Offline");
 menuAction=new JMenu("Action");
 menuBar.add(menuAction);
 menuAction.addActionListener(this);
 /* creazione degli oggetti del menu Action */
 menuGoTo=new JMenuItem("Go To");
 menuAction.add(menuGoTo);
 menuGoTo.addActionListener(this);
 /* Pulsante per fermare l'acquisizione */
 menuStop=new JMenuItem("Stop");
 menuAction.add(menuStop);
 menuStop.addActionListener(this);
 /* Menu Help ancora primo di sottomenu e funzionalità */
 menuHelp=new JMenu("Help");
 menuHelp.setMnemonic('h');
 menuBar.add(menuHelp);
```

```
tabbedPane = new JTabbedPane();
getContentPane().add(panel);
panel.add(tabbedPane);
}
public JMenuItem CreateMenuItem(JMenu menu, int iType, String sText,
0ptImageIcon image, int acceleratorKey,String sToolTip)
{
//creazione Item
JMenuItem menuItem;
switch (iType)
{
case ITEM_RADIO:
menuItem = new JRadioButtonMenuItem();
break;
case ITEM_CHECK:
menuItem = new JCheckBoxMenuItem();
break;
default:
menuItem = new JMenuItem();
break;
}
//Aggiunta del test per item
menuItem.setText(sText);
//Aggiunta icona opzionale
if (image != null)
menuItem.setIcon(image);
//aggiunta shortcut
if (acceleratorKey > 0)
menuItem.setMnemonic(acceleratorKey);
//aggiunta the option tool tip text
if (sToolTip !=null)
menuItem.setToolTipText(sToolTip);
menuItem.addActionListener(this);
menu.add(menuItem);
return menuItem;
} //fine costruttore menuItem
/*Metodo che riceve come input un evento edesegue le operazioni relativi
0ptalla voce del menu selezionato */
public void actionPerformed (ActionEvent event)
{
// Attività offline
if(event.getSource()==menuFileOffline)
{
/* Apertura finestra di dialogo per scegliere il file da elaborare. Viene
0pt utilizzato randomAccessFile perche' consente di saltare da un punto ad
0pt un altro del file*/
RandomAccessFile f=null;
JFileChooser fileChooser=new JFileChooser();
int returnVal = fileChooser.showOpenDialog(ma2ttool.this);
if (  returnVal == JFileChooser.APPROVE_OPTION)
{
```

```
 File trace=fileChooser.getSelectedFile();
 try
 {f=new RandomAccessFile(trace,"r"); }
 catch (FileNotFoundException e) {}
 /*Avvia il task per l'elaborazione offline*/
 off offline=new off(f,trace);
 offline.start();
 }// Fine if
 } // fine if Attività Offline
 // Attività Online
 if(event.getSource()==menuFileOnline)
 {
 selectedValu=1;
 // Salva in un array le intefacce di rete con le relative informazioni
 devices = JpcapCaptor.getDeviceList();
 /* Crea un menu a tendina in cui l'utente può selezionare il tipo di
 0ptperiferica da utilizzare per l'acquisizione del traffico di rete*/
 JOptionPane dialog = new JOptionPane();
 Object name []=new Object[devices.length];
 //Definisco array name contenente il solo nome delle interfacce di rete
 0ptdisponibili
 for (i=0; i<devices.length; i++)
 {name [i]= devices[i].description; }
 /*Definisco il menu a tendina con i tipi di filtri applicabili*/
 Object[] filter= {"tcp","udp","no filter"};
 /*Avvia il task per l'acquisizione e l'analisi in tempo reale*/
 get acquisition=new get(name,devices.length);
 acquisition.start();
 }// Fine if Attività ONLINE
 /* Attività del menu Quit. Chiude definitivamente l'applicazione */
 if(event.getSource()==menuFileExit)
 System.exit(0);
 /* Setta il valore per terminare l'acquisizione del traffico e dell'
 0ptanalisi nell'attivita' online*/
 if(event.getSource()==menuStop)
 {
  selectedValu=0;
 }
 /* Avvia il task per la creazione della tabella parziale quando viene
 0ptpremuto il pulsante goto*/
 if(event.getSource()==menuGoTo)
 {
  GoToThread _goto = new GoToThread();
  _goto.start();
 }
}
/*Thread per l'attivita' di analisi ONLINE*/
public class get extends Thread
{
 Object name1[];
 int len;
```

```
public get(Object j [], int u)
{
 len=u;
 name1=new Object[u]   ;
 /* Assegno al vettore name tutte le schede di rete identificate dalla
 0ptlibreria JPCAP*/
 name1=j;
}
public void run()
{
 flagErrorInput=0;
 go=false;
 set=false;
 /*Rende visibile per impostare i parametri di acquisizione*/
 pannello=new CustomDialog(name1);
 pannello.setVisible(true);
 pannello.setSize(400,350);
 pannello.pack();
 pannello.setValue(null);
 /* Crea l'istanza per l'acquisizione del traffico di rete passando come
 0ptparametrei di ingresso:
  * - Interfaccia di rete da utilizzare
  * - numero di byte da acquisire
  * - boolean che vale true se si vuole acquisire in promiscuous mode
  *   e false se si vuole acquisire in non promiscuous mode
  * - timeout dopo il quale termina l'acquisizione*/
 try
 {
  RandomAccessFile serie=new RandomAccessFile("tmpSerie.txt","rw");
  Packet packet =new Packet();
  // Attende finche' l'utente non preme un bottone
  while (pannello.getValue()==null);
  pannello.setVisible(false);
  /*Se l'utente preme cancel viene fermato il processo*/
  if(pannello.getValue()=="Cancel") stop();
  // Richiama i metodi che restituiscono il tipo di Network device da
  0ptutilizzare per l'acquisizione e il tipo di filtro
  selectedValue=pannello.getNetwork();
  JpcapCaptor captor = JpcapCaptor.openDevice(devices[selectedValue],65,
  0pttrue,20);
  selectedFilter=pannello.getFilter();
  selectedWavelet=pannello.getWavelet();
  saveFile=pannello.getSave();
  flagVar=pannello.getFlagVar();
  flagQuan=pannello.getFlagQuan();
  doTable=pannello.getTable();
  try
  {
       /* Se nessuno dei due check box viene spuntato viene visualizzato
       0ptun messaggio di errore*/
       if (flagQuan==false && flagVar==false) flagErrorInput=1;
```

```
      /* Se viene selezionato il quantile si estrae il valore inserito
      0ptdall'utente*/
      if (flagQuan==true)
      quantileNumber=Integer.parseInt(pannello.getQuantile().toString())
      0pt;
      /* Se non viene inserito nessun livello si crea un errore*/
      if (pannello.getLevel().toString()=="") flagErrorInput=1;
      selectedLevel=Integer.parseInt(pannello.getLevel().toString());
}
catch(NumberFormatException e)
{
 /*Pannello con la segnalazione di errore*/
 flagErrorInput=1;
      if (flagQuan==true)
       JOptionPane.showMessageDialog(panel,"insert a correct value for
       0ptquantile","Input error",JOptionPane.ERROR_MESSAGE);
      else
       JOptionPane.showMessageDialog(panel,"the value insert for the
       0pttimeslot or the number of levels is incorret","Input error",
       0ptJOptionPane.ERROR_MESSAGE);
}
/*Calcola il numero di campioni necessari per l'analisi*/
sampleNumber=64*(Math.pow(2,selectedLevel-1));
/*Avvia la fase di testin per il calcolo del timeslot*/
if (pannello.getValue()=="Test")
{
 /*Conta il numero di pacchetti acquisiti*/
      int tot=0;
      double timeOrigine=0;
      /*Tempo di fine del timer*/
      long ty=(180*1000);
      flagTest=true;
      set=true;
      packet=captor.getPacket();
      timer1 = new java.util.Timer();
      timerTest = new MyTask2();
      timer1.schedule(timerTest,ty);
      /*Conta i pacchetti finche' non scade il timer*/
      while (_timerTest==false)
      {
       if (packet!=null)
       {tot++;}
       packet=captor.getPacket();
      }//end while
      /*calcola il timeslot*/
      timeTest=(long)(((ty*10)/tot)+0.0002);
      if (timeTest <=0)
       timeTest=1;
      /* Visualizza il pannello con il valore del timeslot calcolato */
      pannello2=new CustomDialog(selectedValue,selectedFilter,timeTest);
      pannello2.setVisible(true);
```

```
     pannello2.setSize(400,350);
     pannello2.pack();
     while(pannello2.getValue()==null);
     pannello2.setVisible(false);
}// Fine if Test
if(pannello.getValue()=="Next")
      set=false;
/*Setta la variabile per l'avvio dell'acquisizione*/
if(pannello.getValue()=="Next"||pannello2.getValue()=="Next") go=true;
if(go==true)
{
     /* Prende dal form il valore del timeslot, se e' scritto
     0ptcorrettamente lo assegna alla variabile timeslot altrimenti
     0ptmostra un messaggio di errore*/
     selectedText=pannello.getTxt();
     if(set==false)
     {
      try
      {timeSlot=Double.parseDouble(selectedText.toString()); }
      catch(NumberFormatException e)
      {
       if(flagErrorInput==0)
       {
        flagErrorInput=1;
        JOptionPane.showMessageDialog(panel,"the value insert for the
        0pttimeslot or the number of levels is incorret","Input error",
        0ptJOptionPane.ERROR_MESSAGE);
       }
      }//fine cath
     }// Fine if Set
     /* Assegna alla variabile timeSlot il valore del timeslot
     0ptcalcolato nella fase di testing*/
     else timeSlot=timeTest;
     if (flagErrorInput==1)
     {
      get a=new get(name1,len);
      a.start();
      stop();
     }
     // Apre finestra di dialogo per selezionare il file in cui salvare
     0pt la traccia di traffico
     if(saveFile==true)
     {
      fileChooser1=new JFileChooser();
      returnVal = fileChooser1.showOpenDialog(ma2ttool.this);
     }
     else chooser=true;
     //Se il file viene selezionato si procede con l'attività di
     0ptacquisizione
     if (returnVal == JFileChooser.APPROVE_OPTION)
      chooser=true;
```

```
if(chooser==true)
{
 //Definisco intervallo di acquisizione apro interfaccia per l'
 0ptacquisizione del traffico di rete:
 try
 {
  if(saveFile==true)
  {
   output=fileChooser1.getSelectedFile();
   /*Crea l'istanza per scrivere il pacchetto nel file di uscita*/
   writer=JpcapWriter.openDumpFile(captor,output.getAbsolutePath()
   0pt);
  }
  //indica il numero di pacchetti acquisiti
  int cont=0;
  /*Imposta l'eventuale filtro selezionato dall'utente*/
  if (selectedFilter=="no filter");
  else captor.setFilter(selectedFilter.toString(), true);
  p=0;
  Tot=sampleNumber;
  tmp=new File("tmpSerie"+p+".txt");
  file = new FileOutputStream(tmp);
  out=new PrintStream(file);
  long ty=0;
  /* Acquisice il traffico finche' l'utente non preme il pulsante
  0ptstop*/
  double origin =0;
  double orig=0;
  controw=0;
  manager m = new manager();
  m.start();
  boolean primo =true;
  while(Integer.parseInt(selectedValu.toString())!=0)
  {
        panel.validate();
        panel.repaint();
        //cattura un singolo pacchetto
        packet=captor.getPacket();
        /* salva il pacchetto nel file se l'utente ha scelto di
        0ptsalvare il traffico */
        if (packet!=null)
        {
         if (saveFile==true)
                writer.writePacket(packet);
         if(primo == true)
         {
          sec=packet.sec;
          usec=packet.usec;
          orig=sec+(usec*(Math.pow(10,-6)));
          primo=false;
         }
```

```
          contserie++;
          sec=packet.sec;
          usec=packet.usec;
          tstamp=sec+(usec*(Math.pow(10,-6)));
          if(tstamp-orig-origin>timeSlot)
          {
           double diff = tstamp-origin-orig;
           Update aggiorna=new Update(contserie,tstamp, origin,
           0pttimeSlot, orig);
           aggiorna.start();
           contserie=1;
           int tw=(int)((tstamp-origin-orig)/timeSlot);
           origin=origin+ tw*timeSlot;
          }
         }
   }//end while
   finish=true;
   if(saveFile==true)
   {
        captor.close();
        writer.close();
   }
   String[] columns={"Num","Time","Source","Destination","Protocol
   0pt","Info"};
   RandomAccessFile f=null;
   try
   {
        if(saveFile==true)
        {f=new RandomAccessFile(output,"r"); }
   }
   catch (FileNotFoundException e) {}
   //Crea la tabella
   if(saveFile==true)
   if(doTable==true)
   {
        Table tabella=new Table(f,columns);
        panel.add(tabbedPane);
        tabbedPane.addTab("Tab"+counter, null,tabella,"Does
        0ptnothing");
        counter++;
   }
  } // Fine try
  catch(java.io.IOException e) {}
}
}
if(pannello.getValue()=="Cancel")
{
 pannello.setVisible(false);
 stop();
}
if (set==true)
```

```
        {
         if(pannello2.getValue()=="Cancel")
         {
          pannello2.setVisible(false);
          stop();
         }
    }
   }//fine try
   catch (IOException e){}
  }
 }
 /* Task Update serve per sapere quanti zeri inserire tra due timeslot nel
 0pt caso non ci siano pacchetti*/
 public class Update extends Thread
 {
  int contp;
  int nzero;
  public Update(int c, double t, double o, double ts,double or)
  {
   contp = c;
   nzero = (int)((t-o-or)/ts)-1;
  }
  public void run()
  {
        vector[write]=contp;
        vector[write+1]=nzero;
        write=write+2;
        if(write>=10000)
         write=0;
        controw++;
        stop();
  }//end run
}//end task Update

/* task per la gestione e l'avvio dei processi di elaborazione della DWT
0pt*/
public class manager extends Thread
{
 int a;
 int b;
 public void run()
 {
  while(finish != true)
  {
   if(controw>0)
   {
        a=vector[read];
        b=vector[read+1];
        read=read+2;
        if(read>=10000)
         read=0;
```

```
            out.println(a);
            slot++;
            /*raggiunto il numero di N timeslot avvia la DWT*/
            if (slot==Tot)
            {
             p++;
             TaskDWT pyramid =new TaskDWT(tmp,selectedWavelet,p,selectedLevel,
             0ptsampleNumber,quantileNumber);
             pyramid.run();
             try
             {
              tmp=new File("tmpSerie"+p+".txt");
              file = new FileOutputStream("tmpSerie"+p+".txt");
              out=new PrintStream(file);
              Tot=Tot+sampleNumber;
             }
         catch(FileNotFoundException f) {}
            }
            for(int i=0; i < b; i++)
            {
             out.println(0);
             slot++;
             /*raggiunto il numero di N timeslot avvia la DWT*/
             if (slot==Tot)
             {
              p++;
              TaskDWT pyramid =new TaskDWT(tmp,selectedWavelet,p,selectedLevel
              0pt,sampleNumber,quantileNumber);
              pyramid.run();
              try
              {
               tmp=new File("tmpSerie"+p+".txt");
               file = new FileOutputStream("tmpSerie"+p+".txt");
               out=new PrintStream(file);
               Tot=Tot+sampleNumber;
              }
              catch(FileNotFoundException f) {}
             }
            }// fine for d b
        controw--;
       } // fine if controw
      }//fine while
      stop();
     }//end run
    }//end task Update

    /*Pone la variabile a true quando raggiunge la fine del tempo prestabilito
    0pt*/
    public class MyTask2 extends TimerTask
    {
     public void run()
```

```
 {
  _timerTest=true;
 }
}

/* Task per il calcolo della DWT */
public class TaskDWT extends Thread
{
 RandomAccessFile q;
 File file;
 int w;
 int j;
 int level;
 double sample;
 int quant;
 /* Metodo costruttore che riceve in input il file con i campioni il tipo
 0ptdi wavelet da usare per l'analisi, l'indice del file da usare per il
 0ptcalcolo, il numero di livelli da usare, il numero di campioni e il
 0ptvalore del quantile */
 public TaskDWT(File x,int y,int h,int l,double s,int quan)
 {
  try
  {
        q=new RandomAccessFile(x,"r");
        file=x;
        w=y;
        j=h;
        level=l;
        sample=s;
        quant=quan;
  }
  catch(FileNotFoundException m){}
 }
 // prende in ingresso il file da elaborare, il tipo di wavelet e la
 0ptdimensione del file
 public void run()
 {
  /*Richiama la DWT.java per l'elaborazione*/
  DWT analysis=new DWT(q,w,j,level,sample,flagQuan,flagVar,quant);
  if ((flagQuan==false)&&(flagVar==true))  ris=new double[level*2];
  if ((flagQuan==true)&&(flagVar==false))  ris=new double[level*2];
  if ((flagQuan==true)&&(flagVar==true))   ris=new double[level*4];
  ris=analysis.doDWT();
  /*Creazione dei grafici*/
  if (jump==0)
  {
   _graphicL=new Graphic ("Approximation",level,finestra,0);
   _graphicH=new Graphic ("Detail",level,finestra,1);
   _graphicTimeL=new GraphicTime ("Approximation",level,0);
   _graphicTimeH=new GraphicTime ("Detail",level,1);
   jump=1;
```

```
  }
   _graphicL.UpdateChart(ris,0);
   _graphicH.UpdateChart(ris,level);
   _graphicTimeL.UpdateChart(ris,0);
   _graphicTimeH.UpdateChart(ris,level);
 } // fine run
}

/*Classe per l'analisi offline*/
public class off extends Thread
{
 RandomAccessFile f;
 File trace1;
 public off(RandomAccessFile k, File trace)
 {
  f=k;
  trace1=trace;
 }
 public void run()
 {
  try
  {
   try
   {
    flagErrorInput=0;
        /*Finestra di configurazione per l'analisi offline*/
    pannello3= new CustomDialog();
    pannello3.setVisible(true);
        pannello3.setSize(400,350);
    pannello3.pack();
        while (pannello3.getValue()==null);
    if (pannello3.getValue()=="Cancel")
        {
         pannello3.setVisible(false);
         stop();
        }
        if(pannello3.getValue()=="Next")
        {
         pannello3.setVisible(false);
         selectedText=pannello3.getTxt();
         try
         {timeSlot=Double.parseDouble(selectedText.toString()); }
         catch(NumberFormatException e)
         {
          flagErrorInput=1;
          JOptionPane.showMessageDialog(panel,"the value insert for the
          0pttimeslot or the number of levels is incorret","Input error",
          0ptJOptionPane.ERROR_MESSAGE);
         }
         selectedWavelet=pannello3.getWavelet();
     flagVar=pannello3.getFlagVar();
```

```
        flagQuan=pannello3.getFlagQuan();
        try
        {
         if (flagQuan==true)
               quantileNumber=Integer.parseInt(pannello3.getQuantile().
               0pttoString());
         selectedLevel=Integer.parseInt(pannello3.getLevel().toString());
    }
        catch(NumberFormatException e)
        {
         if (flagErrorInput==0)
         {
               flagErrorInput=1;
               if (flagQuan==true)
                       JOptionPane.showMessageDialog(panel,"the value
                       0ptinsert for quantile is wrong","Input error",
                       0ptJOptionPane.ERROR_MESSAGE);
               else JOptionPane.showMessageDialog(panel,"the format of
               0ptnumber insert for the number of levels is wrong","Input
               0pterror",JOptionPane.ERROR_MESSAGE);
         }
        }
        /*Calcola il numero di campio necessari per l'analisi*/
        sampleNumber=64*(Math.pow(2,selectedLevel-1));
    int contPacket=1;
        int contSample=0;
        int info[]=new int[16];
        int s=0;
        int p1=0;
        double t=0;
        double origine=0;
        int length=0;
        if (flagErrorInput==1)
        {
         off o=new off(f,trace1);
         o.start();
         stop();
        }
        try
        {f=new RandomAccessFile(trace1,"r");}
        catch(FileNotFoundException m){}
        if(f.read()==212)
               indian=true;
        else
               indian= false;
        f.skipBytes(23);
        /* Crea l'oggetto packet per richiamare i metodi per l'estrazione
        0pt del timestemp e la lunghezza di un pacchetto*/
        PacketElaboration packet1= new PacketElaboration();
        File tmp1=new File("tmpSerie"+p1+".txt");
        FileOutputStream file = new FileOutputStream(tmp1);
```

```
PrintStream out=new PrintStream(file);
for (int i=0; i<16; i++)
       info[i]=f.read();
//chiamata al metodo per estrarre il timestamp
origine= packet1.TimeValue(info,indian);
double origin=0;
//double or=origine;
length = packet1.Plength(info,indian);
f.skipBytes(length);
/* Ripete il ciclo fino a che non ha raggiunto la fine del file*/
while (s != -1)
{
 panel.validate();
 panel.repaint();
 // Lettura header di pacchetto:
 for (int i=0; i<16; i++)
       info[i]=f.read();
 //chiamata al metodo per estrarre il timestamp
 t=packet1.TimeValue(info,indian);
 if(t-origine-origin<timeSlot)
 { contPacket++; }
 else
 {
       /* Calcola la finestra temporale per porre a zero i
       0ptTimeslot in cui non c'erano pacchetti*/
       tw=(int)((t-origine-origin)/timeSlot)-1;
       for(int i=0;i<tw;i++)
  {
        out.println(0);
        contSample++;
        if (contSample==sampleNumber)
        {
         p1++;
         TaskDWT pyramid =new TaskDWT(tmp1,selectedWavelet,p1,
         0ptselectedLevel,sampleNumber,quantileNumber);
         pyramid.run();
         tmp1=new File("tmpSerie"+p1+".txt");
         file = new FileOutputStream(tmp1);
         out=new PrintStream(file);
         contSample=0;
        }
   origin=origin+timeSlot;
  }
 contSample++;
out.println(contPacket);
/* Quando si e' raggiunto il numero di campio si avvia l'analisi
0pt della DWT*/
if (contSample==sampleNumber)
{
 p1++;
 TaskDWT pyramid =new TaskDWT(tmp1,selectedWavelet,p1,
```

```
          0ptselectedLevel,sampleNumber,quantileNumber);
          pyramid.run();
          tmp1=new File("tmpSerie"+p1+".txt");
          file = new FileOutputStream(tmp1);
          out=new PrintStream(file);
          contSample=0;
         }
         contPacket=1;
         origin=origin+timeSlot;
    }
       //restituisce la lunghezza del pacchetto in esame
       length = packet1.Plength(info,indian);
       f.skipBytes(length-1);
       s=f.read();
  }//end while
  System.out.println("FINE");
  /*Crea la tabella con le informazioni dei pacchetti*/
  String[] names={"Num","Time","Source","Destination","Protocol","Info"};
  doTable=pannello3.getTable();
  if(doTable==true)
  {
   f.seek(0);
   //crea la tabella con i dati estratti dal file
       Table tabella=new Table(f,names);
       panel.add(tabbedPane);
       tabbedPane.addTab("Tab"+counter, null,tabella,"Does nothing");
       counter++;
  }// fine if doTable
 }// end tasto next;
} // fine try
catch (FileNotFoundException e ){}
}//fine try
catch (IOException e ){}
}
}
/* Task per la creazione della tabella */
public class GoToThread extends Thread
{
 public void run()
 {
  /*Finestra di dialogo per inserire il blocco iniziale, il blocco finale,
  0ptil timeslot e il numero di lovello*/
  Dialog gotoWindows = new Dialog();
  gotoWindows.setVisible(true);
  gotoWindows.setSize(200,350);
  while(gotoWindows.getValue()==null) ;
  if(gotoWindows.getValue()=="Next")
  {
   RandomAccessFile file = null;
   JFileChooser fileChooser=new JFileChooser();
   fileChooser.showOpenDialog(ma2ttool.this);
```

```
if(returnVal== JFileChooser.APPROVE_OPTION)
     {
      try
      {
       /*Assegna alle variabili i valori inseriti dall'utente*/
       Object a=gotoWindows.getTxt();
       Object b=gotoWindows.getTxt1();
       Object c=gotoWindows.getTxt2();
       Object d=gotoWindows.getLevel();
       gotoTS=Double.parseDouble(a.toString());
       gotoStart=Integer.parseInt(b.toString());
       gotoStop=Integer.parseInt(c.toString());
       gotoLevel=Integer.parseInt(d.toString());
      }
      catch(NumberFormatException e)
      {
       flagErrorInput=1;
       JOptionPane.showMessageDialog(panel,"the value insert for one or
       0pt more variables are incorrets","Input error",JOptionPane.
       0ptERROR_MESSAGE);
      }
      gotoWindows.setVisible(false);
      double origine=0;
      int length=0;
      int info[] =new int[26];
      PacketElaboration packet2 = new PacketElaboration();
      File trace = fileChooser.getSelectedFile();
      try
      {
       f1= new RandomAccessFile(trace,"r");
      }
      catch (FileNotFoundException e) {}
      /*Verifica se l'ordinamento dei byte nel file e' little o big
      0ptindian*/
      try
      {
       if(f1.read()==212)
             indian=true;
       else
             indian=false;
       f1.skipBytes(23);
       for(int i=0;i<16; i++)
             info[i]=f1.read();
       origine=packet2.TimeValue(info,indian);
       length=packet2.Plength(info,indian);
       f1.skipBytes(length);
       boolean control=false;
       double timeStart=64*(Math.pow(2,gotoLevel-1))*gotoTS*gotoStart;
       double timeStop=64*(Math.pow(2,gotoLevel-1))*gotoTS*gotoStop;
       double gotoTime=0;
       int s=0;
```

```
int contLine=0;
String[] columns={"Num","Time","Source","Destination","Protocol
0pt","Info"};
Object[][] data= new Object[1][6];
Table t=new Table(data,columns);
Object arrayInput[]= new Object[6];
contByte=24+length;
/*Legge i pacchetti ed estra l'indirizzo ip sorgente e
0ptdestinazione, il timeslot e il tipo di protocollo*/
while(control!=true)
{
      contLine++;
      for(int i=0;i<16; i++)
       info[i]=f1.read();
      contByte=contByte+16;
      gotoTime=packet2.TimeValue(info,indian);
      length=packet2.Plength(info,indian);
      /* Inserisce nella tabella solo i pacchetti compresi nei
      0ptblocchi inseriti dall'utente*/
      if ((gotoTime-origine)>=timeStart&&(gotoTime-origine)<=
      0pttimeStop)
      {
       arrayInput[0]=contLine;
       arrayInput[1]=gotoTime-origine;
       f1.skipBytes(15);
       contByte=contByte+15;
       for (int i=0; i<20; i++)
   {
        if ((i == 8) || ((i>10)&&(i<19)))
              {info[i]=f1.read();}
        else
              s=f1.read();
       }
       contByte=contByte+20;
       //Salviamo le info raccolte in un array
       Object source= info[11]+"."+info[12]+"."+info[13]+"."+
       0ptinfo[14];
       arrayInput[2]=source.toString();
       Object destination=    info[15]+"."+info[16]+"."+info
       0pt[17]+"."+info[18];
       arrayInput[3]=destination.toString();
       arrayInput[4]=info[8];
       arrayInput[5]=null;
       if (length-35>0)
       //Saltiamo gli ultimi byte del pacchetto;
       f1.skipBytes(length-35);
       else f1.seek(contByte+(length-35));
       contByte=contByte+(length-35);
       //Aggiungiamo una riga alla tabella con le info raccolte
       t.addRow(arrayInput);
      }//end if
```

```
                else
                {
                 if((gotoTime-origine)>timeStop)
                        control=true;
                  f1.skipBytes(length);
                  contByte=contByte+length;
                }// end else
              }//end while
              JScrollPane scrollPane = new JScrollPane(t);
              panel.add(tabbedPane);
              tabbedPane.addTab("Block "+gotoStart+"-"+gotoStop, null,t,"Does
              0pt nothing");
              counter++;
              t.setVisible(true);
             }//fine try
             catch(IOException e){}
            }
   }
  }//Fine Run
}//Fine Thread

/*Progrmma principale*/
public static void main(String args[]) throws IOException
{
 SwingUtilities.invokeLater(new Runnable()
 {
  public void run()
  {
   ma2ttool tool = new ma2ttool();
   tool.setVisible(true);
  }
 });
}
}
```

A.2 Table.java

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.io.*;

public class Table extends JPanel {

    // crea un nuova JTable
        public JTable table;
    // è la barra di progresso che indica l'avznzamento della lettura del
    0ptfile
    private Progress finestra;
    // crea una istanza della classe PacketElaboration per estrarre
    0ptTimeStamp e Lunghezza dei
    // pacchetti
    private PacketElaboration packet;
    // crea un random access file utilizzato per scorrere i singoli byte
    private RandomAccessFile f;
    // indica la dimensione del file di input in Byte
    private double lung;
    // indica se il file è bigEndian o littleEndian
    private boolean indian=true;
    // indica il l'ultimo byte letto dal file di input
    private int contByte=0;
    // indica la percentuale di avanzamento del file
    private int y=0;
    // indica il numero di riga aggiunta
    private int cont=0;
    // indica se è stata settata l'origine
    private boolean flagOrigine=false;
    // crea uno scroll pane
    private JScrollPane scrollPane ;
    // vettore che contiene i dati utilizzati per aggiungere una riga alla
    0pt tabella
    private Object arrayInput[] ;
    // contiene il timeStamp del pacchetto analizzato
    private double time;
    // contiene la lunghezza del pacchetto analizzato
    private int length;

    //Costruttore di tabella a partire da un file e un vettore di stringhe
    0pt che contiene i nomi delle tabelle
    public Table(RandomAccessFile f,String[] names)
        {
```

```
super(new GridLayout(1,0));
y=24;
try {lung=(int)f.length();}
catch(java.io.IOException e){}
//Crea una matrice 1x6 vuota
Object dati[][]= new Object [1][6];
        for (int j= 0; j<6; j++)
           dati[0][j]=null;
        //crea una tabella dando in input la matrice vuota e il
        0ptvettore contenente i nomi delle colonne
        Table table=new Table(dati,names);
        //crea un vettore in cui salvare le informazioni estratte
        0ptdai pacchetti
        int info[]= new int [26];
        try{
                arrayInput= new Object[6];
                int s=0;
                packet = new PacketElaboration();
            //legge i primi 24 byte da 0 a 23 (header TCPDUMP)
                contByte=contByte+24;
                //legge il primo byte per controllare se il
                0ptformato è little endian o big endian
                if (f.read()==212)
                        indian=true;
                else indian=false;
                //salta i successivi 23 byte per arrivare al
                0pttermine dell'header di tcpdump
                f.skipBytes(23);
                //Leggo tutto il file rimanente pacchetto per
                0ptpacchetto;
                cont=0;
                double origine=0;
                // lancia la finestra con la barra di avanzamento
                finestra=new Progress(f);
                bar avanzamento=new bar();
                avanzamento.start();
                // legge tutto il file :s=-1 se si è raggiunta la
                0ptfine del file
                while (s != -1)
                {
                        // Lettura header di pacchetto(16 byte):
                        arrayInput[0]=cont+1;
                        contByte=contByte+16;
                        for (int i=0; i<16; i++)
                                info[i]=f.read();
                        //chiamata al metodo per estrarre il
                        0pttimestamp
                        time= packet.TimeValue(info,true);
                        if(flagOrigine==false)
                         {
                          arrayInput[1]=0;
```

```
                        origine=time;
                       }
                      else arrayInput[1]=time-origine;
                      flagOrigine=true;
                      //restituisce la lunghezza del pacchetto
                      0ptin esame
                      length = packet.Plength(info,true);
                      //saltiamo 15 bytes che non interessano
                      f.skipBytes(15);
                      contByte=contByte+15;
                      //estraiamo protocollo, e indirizzi IP
                      0ptsorgente e destinazione
                      for (int i=0; i<20; i++)
                      {
                              if ((i == 8) || ((i>10)&&(i<19)))
                                 {info[i]=f.read();}
                              else
                                      s=f.read();
                      }
                      contByte=contByte+20;
                      y=y+51;
                      //Salviamo le info raccolte in un array
                      Object source=  info[11]+"."+info[12]+"."+
                      0ptinfo[13]+"."+info[14];
                      arrayInput[2]=source.toString();
                      Object destination=     info[15]+"."+info
                      0pt[16]+"."+info[17]+"."+info[18];
                      arrayInput[3]=destination.toString();
                      arrayInput[4]=info[8];
                      arrayInput[5]=null;
                      //Saltiamo gli ultimi byte del pacchetto;
                      if (length-35>0)
                        f.skipBytes(length-35);
                      else f.seek(contByte+(length-35));
                      contByte=contByte+(length-35);
                      s=f.read();
                      f.seek(contByte);
                      y=y+length-35;
                      //Aggiungiamo una riga alla tabella con le
                      0pt info raccolte
                      table.addRow(arrayInput);
                      cont++;
                      finestra.avanzamento(y);
              }//end while
       table.removeRow(cont);
//Crea lo scroll pane ed gli aggiunge la tabella.
scrollPane = new JScrollPane(table);

//Aggiunge lo scroll pane al panel.
add(scrollPane);
       finestra.setVisible(false);
```

```
                } catch(java.io.IOException e){}
        }
            //Costruttore di tabella che riceve in input una matrice e un
            0ptvettore di oggetti.
        public Table(Object[][]data,Object[] names)
            {
            super(new GridLayout(1,0));
            //crea la tabella utilizzando il modello Default.
            table = new JTable(new DefaultTableModel(data,names));
            table.setPreferredScrollableViewportSize(new Dimension(420,100));

            //Crea lo scroll pane ed gli aggiunge la tabella.
            scrollPane = new JScrollPane(table);

            //Aggiunge lo scroll pane al panel.
            add(scrollPane);
                     }
        // metodo che permette di aggiungere una riga alla fine della nostra
        0pttabella.
        public  void addRow(Object[] riga)
            { ((DefaultTableModel) table.getModel()).addRow(riga); }
            public void removeRow(int p)
            { ((DefaultTableModel) table.getModel()).removeRow(p); }
            // thread che visualizza la finestra di avanzamento
            public class bar extends Thread
             {
              public void run()
                    {
                     finestra.pack();
             finestra.setVisible(true);
            }
             }
}
```

A.3 CustomDialog.java

```
import javax.swing.JOptionPane;
import javax.swing.*;
import javax.swing.JDialog;
import javax.swing.JTextField;
import java.beans.*;
import java.awt.*;
import java.awt.event.*;

/*****************************************************************
Classe che implementa la finestra di dialogo per la scelta della
scheda di rete, il tipo di filetro e il valore del timeslot.
*****************************************************************/
class CustomDialog extends JDialog implements ActionListener, ItemListener
0pt,PropertyChangeListener
    {
    // variabile per la creazione di un optionpane.
    private JOptionPane optionPane;
    // variabile per la creazione di un panel.
private JPanel prova=new JPanel();
// Variabili per la creazione dei bottoni del pannello.
    private String btnString1 = "Next";
    private String btnString2 = "Cancel";
    private String btnString3 = "Test";
    // Variabili per la creazione di caselle di testo per l'inserimento di
    0pt: timeslot,
    // percentuale del quantile, e numero di livelli.
    private JTextField timeSlot;
    private JTextField quantileNumber;
    private JTextField levelNumber;
    // creazione delle finestre a cascata per la scelta dei filtri e della
    0pt scheda di rete.
    private JComboBox filterbox;
private JComboBox devicebox;
// creazione delle caselle per l'abilitazione di: calcolo varianza e/o
0ptquantile, creazione della tabella e salvataggio del file (solo modalità
0pton line).
private JCheckBox quantile;
    private JCheckBox varianza;
    private JCheckBox table;
    private JCheckBox savefile;
    // variabile che restituisce il valore in base al bottone premuto .
private Object value=null;
// variabile che restitutisce il filtro selezionato.
private String selectFilter;
// variabile che restitutisce la scheda di rete selezionata.
    private int selectDevice;
    // variabile che restitutisce la wavelet selezionata.
    private int selectWavelet;
    // variabile che restituisce il valore del Timeslot
```

```
private String text;
//contiene il numero di livelli selezionato dall'utente
private Object level;
// variabili che indicano se la relativa casella è stata selezionata
private boolean flag=false;
private boolean quan=false;
private     boolean var=false;
private boolean tab=false;
private boolean save=false;
// variabile che restitutisce il valore del quantile selezionata.
private Object quanNumber;

/*************************************************************
            Metodo che crea un pannello di dialogo ricevendo in input
            la lista delle schede di rete presenti sul pc
            viene usato nella modalita' ONLINE
 *************************************************************/
public CustomDialog(Object devices [])
    {
            setTitle("Setting");
    setSize(400,350);
            pack();
    // Tipi di filtro applicabili. Se si sceglie TCP vengono salvati
    0ptsolo pacchetti TCP e cosi' via
    Object[] filter= {"tcp","udp","no filter"};
    //valore di default selezionato nel menu a tendina nella sezione
    0ptfiltro
    selectFilter="no filter";
    //Valore di defaul selezionato nel menu a tendina nella sezione
    0ptNetwork Device
    Object[] wavelet= {"Haar","Daubachies 4 tap","Daubachies 6 tap
    0pt"};//<-------- da aggiungere altri wavelet
    // creazione delle varie componenti del pannello
    filterbox = new JComboBox(filter);
    filterbox.setSelectedIndex(2);
    filterbox.addActionListener(this);
    filterbox.setActionCommand("filter");
    timeSlot = new JTextField(10);
    JComboBox devicebox = new JComboBox(devices);
    devicebox.setSelectedIndex(0);
    devicebox.addActionListener(this);
    devicebox.setActionCommand("device");
    JComboBox waveletbox = new JComboBox(wavelet);
    waveletbox.setSelectedIndex(0);
    waveletbox.addActionListener(this);
    waveletbox.setActionCommand("wavelet");
    levelNumber=new JTextField(10);
    quantileNumber= new JTextField(3);
    //Messaggi che spiegano il significato dei vari campi
    String msgString1 = "Select Network";
    String msgString2 = "Select Filter";
```

```
String msgString3 = "Insert Time Slot or press test to estimate it
0pt";
String msgString4 = "";
String msgString5 = "Select wavelet";
String msgString6 = "Insert number of Levels";
String msgString7 = "Select type of analysis";
String msgString8 = "Insert value to calculate quantile in % (da 0
0pt a 100) ";
quantile = new JCheckBox("Quantile");
quantile.setMnemonic(KeyEvent.VK_C);
quantile.setSelected(false);
quantile.addItemListener(this);
varianza = new JCheckBox("Varianza");
varianza.setMnemonic(KeyEvent.VK_C);
varianza.setSelected(false);
varianza.addItemListener(this);
table = new JCheckBox("Create table?");
table.setMnemonic(KeyEvent.VK_C);
table.setSelected(false);
table.addItemListener(this);
savefile = new JCheckBox("Do you want save traffic trace?");
savefile.setMnemonic(KeyEvent.VK_C);
savefile.setSelected(false);
savefile.addItemListener(this);
// indica come inserire i vari componenti nel option pane.
Object[] array = {msgString1,devicebox,msgString4,msgString2,
0ptfilterbox,msgString4,msgString5,waveletbox,msgString4,msgString3
0pt,timeSlot,msgString4,msgString6,levelNumber,msgString4,
0ptmsgString7,quantile,msgString8,quantileNumber,varianza,
0ptmsgString4,savefile,msgString4,table};

//Crea un array con i pulsanti disponibili nella finestra di
0ptdialogo.
Object[] options = {btnString1, btnString2,btnString3};

//Crea il JOptionPane.
optionPane = new JOptionPane(array,JOptionPane.QUESTION_MESSAGE,
0ptJOptionPane.YES_NO_OPTION,null,options,options[0]);
        optionPane.setVisible(true);
        setContentPane(optionPane);
setDefaultCloseOperation(HIDE_ON_CLOSE);
        addWindowListener(new WindowAdapter()
                        {
                        public void windowClosing(WindowEvent we)
                        {optionPane.setValue(new Integer(
                        0ptJOptionPane.CLOSED_OPTION));}
                        });

addComponentListener(new ComponentAdapter()
                        {
                        public void componentShown(ComponentEvent
```

```
                                0ptce)
                                {timeSlot.requestFocusInWindow();}
                                });
                        // aggiunge gli actionerlistener ai textbox
                timeSlot.addActionListener(this);
                levelNumber.addActionListener(this);
                quantileNumber.addActionListener(this);
                        optionPane.addPropertyChangeListener(this);
        }

     /*************************************************************
                        Metodo che crea un pannello di dialogo ricevendo in input
                        la scheda di rete,il filtro e il numero di livelli
                        selezionati dall'utente; viene usato nella modalita' TEST
         ************************************************************/
    public CustomDialog(Object net,Object filtro,double d)
                {
                setTitle("Selected Setting :");
                setSize(400,350);
                setBounds(70,100,470,450);
                pack();
                // Tipi di filtro applicabili. Se si sceglie TCP vengono salvati
                0ptsolo pacchetti TCP
                String filter=filtro.toString();
                String network=net.toString();
                //Messaggi che spiegano il significato dei vari campi
                String msgString1 = "Selected Network: "+network;
                String msgString2 = "Select Filter "+filter;
                String msgString3 = "Insert Time Slot";
                String msgString4 = "";
                String msgString5 = "Theorical timeslot : "+d;
                Object[] array = {msgString1,msgString4,msgString2,msgString4,
                0ptmsgString5};

                //Crea un array con i pulsanti disponibili nella finestra di
                0ptdialogo
                Object[] options = {btnString1, btnString2};

                //Create the JOptionPane.
                optionPane = new JOptionPane(array,JOptionPane.QUESTION_MESSAGE,
                0ptJOptionPane.YES_NO_OPTION,null,options,options[0]);
                optionPane.setVisible(true);
                setContentPane(optionPane);
                setDefaultCloseOperation(HIDE_ON_CLOSE);
                addWindowListener(new WindowAdapter()
                                        {
                                       public void windowClosing(WindowEvent we)
                                        { optionPane.setValue(new Integer(
                                        0ptJOptionPane.CLOSED_OPTION)); }
                                        });
                addComponentListener(new ComponentAdapter()
```

```
                                {
                        public void componentShown(ComponentEvent ce)
                                { optionPane.requestFocusInWindow();}
                                });
        optionPane.addPropertyChangeListener(this);
    }
 /************************************************************
                Metodo che crea un pannello di dialogo nella modalita'
                OFFLINE
     ************************************************************/
public CustomDialog()
         {
        setTitle("Setting");
        setSize(400,350);
        pack();
        //Valore di defaul selezionato nel menu a tendina nella sezione
        0ptNetwork Device
        Object[] wavelet= {"Haar","Daubachies 4 tap","Daubachies 6 tap"};
        timeSlot= new JTextField(10);
        JComboBox waveletbox = new JComboBox(wavelet);
        waveletbox.setSelectedIndex(0);
        waveletbox.addActionListener(this);
        waveletbox.setActionCommand("wavelet");
        levelNumber=new JTextField(10);
        quantileNumber= new JTextField(3);
        //Messaggi che spiegano il significato dei vari campi
        String msgString3 = "Insert Time Slot";
        String msgString4 = "";
        String msgString5 = "Select wavelet";
        String msgString6 = "Insert number of Levels";
        String msgString7 = "Select type of analysis";
        String msgString8 = "Insert value to calculate quantile in % (da 0
        0pt a 100) ";
        quantile = new JCheckBox("Quantile");
        quantile.setMnemonic(KeyEvent.VK_C);
        quantile.setSelected(false);
        quantile.addItemListener(this);
        varianza = new JCheckBox("Varianza");
        varianza.setMnemonic(KeyEvent.VK_C);
        varianza.setSelected(false);
        varianza.addItemListener(this);
        table = new JCheckBox("Create Table?");
        table.setMnemonic(KeyEvent.VK_C);
        table.setSelected(false);
        table.addItemListener(this);
        Object[] array = {msgString5,waveletbox,msgString4,msgString3,
        0pttimeSlot,msgString4,msgString6,levelNumber,msgString4,msgString7
        0pt,quantile,msgString8,quantileNumber,varianza,msgString4,table};

        //Crea un array con i pulsanti disponibili nella finestra di
        0ptdialogo
```

```
    Object[] options = {btnString1, btnString2};

    //Create the JOptionPane.
    optionPane = new JOptionPane(array,JOptionPane.QUESTION_MESSAGE,
    0ptJOptionPane.YES_NO_OPTION,null,options,options[0]);
            optionPane.setVisible(true);
            setContentPane(optionPane);
    setDefaultCloseOperation(HIDE_ON_CLOSE);
            addWindowListener(new WindowAdapter()
                    {
                    public void windowClosing(WindowEvent we)
                             {
                              optionPane.setValue(new Integer(
                              0ptJOptionPane.CLOSED_OPTION));
                             }
                    });
    addComponentListener(new ComponentAdapter()
            {
        public void componentShown(ComponentEvent ce)
            {
            timeSlot.requestFocusInWindow();
            }
            });
    timeSlot.addActionListener(this);
    levelNumber.addActionListener(this);
    quantileNumber.addActionListener(this);
    optionPane.addPropertyChangeListener(this);
}

/** Questo metodo gestisce eventi per il campo Device, Filter,Wavelet,
    e acquisice i valori dei vari textbox. */
public void actionPerformed(ActionEvent e)
    {
    if(e.getActionCommand().equals("filter"))
    {
     JComboBox cb = (JComboBox)e.getSource();
     selectFilter = (String)cb.getSelectedItem();
    }
    if(e.getActionCommand().equals("device"))
    {
     JComboBox bc = (JComboBox)e.getSource();
     selectDevice = bc.getSelectedIndex();
    }
    if(e.getActionCommand().equals("wavelet"))
    {
     JComboBox wa = (JComboBox)e.getSource();
     selectWavelet = wa.getSelectedIndex();
    }
    text=timeSlot.getText();
    level=levelNumber.getText();
    quanNumber=quantileNumber.getText();
```

```
}

/** Questo medoto reagisce agli eventi di cambiamento nei vari menu.
0pt*/
public void propertyChange(PropertyChangeEvent e)
    {
     String prop = e.getPropertyName();
     if (flag==true)
       text=timeSlot.getText();
     if (isVisible()&& (e.getSource() == optionPane))
            {
        value = optionPane.getValue();
                    if (btnString1.equals(value));
        }
    }
/* Questo metodo reagisce ai cambiamenti di stato delle combobox */
public void itemStateChanged (ItemEvent e){
    Object source = e.getItemSelectable();
    if((source==quantile)&&(quan==false))  quan=true;
      else if ((source==quantile)&&(quan==true)) quan=false;
    if((source==varianza)&&(var==false))    var=true;
      else if((source==varianza)&&(var==true))      var=false;
    if((source==table)&&(tab==false))       tab=true;
      else if((source==table)&&(tab==true)) tab=false;
    if((source==savefile)&&(save==false)) save=true;
      else if((source==savefile)&&(save==true))     save=false;
                           }
    //Metodo che restituisce il tipo di filtro selezionato
public Object getFilter()
    { return selectFilter; }
    //Metodo che restituisce il tipo di Network Device Selezionato
public int getNetwork()
    { return selectDevice; }
    // Metodo che restituisce il valore del pulsante Selezionato
public Object getValue()
            { return value; }
    // Metodo che setta il valorerestituito dai pulsanti
    public void setValue(Object v)
            { value=v; }
    // Metodo che restituisce il tipo di Wavelet da utilizzare nella
    0ptfase di analisi
    public int getWavelet()
    { return selectWavelet; }
// Metodi che restituiscono i valori dei rispettivi text field
public Object getTxt()
    {
    text=timeSlot.getText();
    return text;
    }

 public Object getLevel()
```

```
            {
            level=levelNumber.getText();
            return level;
            }

        public Object getQuantile()
            {
            quanNumber=quantileNumber.getText();
            return quanNumber;
            }
        // Metodi che restituiscono i valori booleani legati alle check box
        public boolean getFlagVar()
                    { return var; }

            public boolean getFlagQuan()
                    { return quan; }

        public boolean getSave()
            { return save;}

        public boolean getTable()
            { return tab; }
    }
```

A.4 Dialog.java

```
import javax.swing.JOptionPane;
import javax.swing.*;
import javax.swing.JDialog;
import javax.swing.JTextField;
import java.beans.*;
import java.awt.*;
import java.awt.event.*;

/*************************************************************
Classe che fornisce i metodi per la visualizzazione della finestra
di dialogo, per impostare l'intervallo di pacchetti da visualizzare
in tabella.
*************************************************************/
class Dialog extends JDialog implements ActionListener, ItemListener,
0ptPropertyChangeListener
{
        //Campo per inserire il blocco iniziale per la creazione della
        0pttabella
    private JTextField textFieldStart;
    //Campo per inserire il blocco finale per la creazione della tabella
    private JTextField textFieldStop;
    //Campo per inserire il valore del TimeSlot
    private JTextField textField;
    //Campo per inserire numero di livelli dell'algoritmo piramidale
    private JTextField levelNumber;
    //Valore dei pulsanti della finestra di dialogo
    private String btnString1 = "Next";
    private String btnString2 = "Cancel";
    //Variabili per la creazione del pannello
    private JOptionPane optionPane;
        private JPanel panel=new JPanel();

    //Contiene il valore del pulsante quando viene premuto
        private Object value=null;
        //Contiene il valore del TimeSlot
    private String text;
    //Contiene il valore del primo blocco da analizzare
    private String text1;
    //Contiene il valore dell'ultimo blocco da analizzare
    private String text2;
    //contiene il numero di livelli selezionato dall'utente
    private Object level;

         public Dialog()
         {
                //Crea e setta la dimensione della finestra di dialogo
           setTitle("Setting");
        setSize(200,350);
        pack();
```

```
    /*Creazione della casella di testo con la possibilita'
    di inserire al massimo 10 caratteri*/
    textField = new JTextField(10);
    textFieldStart = new JTextField(10);
    textFieldStop = new JTextField(10);
            levelNumber=new JTextField(10);
            //Messaggi che illustrano il significato dei campi
    String msgString1 = "Insert Time Slot";
    String msgString2 = "Insert start block";
    String msgString3 = "";
    String msgString4 = "Insert Stop block";
    String msgString5 = "Insert number of Levels";
    Object[] array = {msgString1,textField,msgString3,msgString5,
    0ptlevelNumber,msgString3,msgString2,textFieldStart,msgString3,
    0ptmsgString4,textFieldStop};
    //Crea un array con i pulsanti disponibili nella finestra di
    0ptdialogo
    Object[] options = {btnString1, btnString2};

    //Creazione dell'optionPane
    optionPane = new JOptionPane(array,JOptionPane.QUESTION_MESSAGE,
    0ptJOptionPane.YES_NO_OPTION,null,options,options[0]);
            optionPane.setVisible(true);
            setContentPane(optionPane);
    setDefaultCloseOperation(HIDE_ON_CLOSE);
            addWindowListener(new WindowAdapter()
            {
         public void windowClosing(WindowEvent we)
         { optionPane.setValue(new Integer(JOptionPane.CLOSED_OPTION))
         0pt; }
    });
    addComponentListener(new ComponentAdapter()
    {
       public void componentShown(ComponentEvent ce)
       {
        textField.requestFocusInWindow();
        textFieldStart.requestFocusInWindow();
        textFieldStop.requestFocusInWindow();
       }
    });
    textField.addActionListener(this);
    textFieldStart.addActionListener(this);
    textFieldStop.addActionListener(this);
    levelNumber.addActionListener(this);
            optionPane.addPropertyChangeListener(this);
}

/** Questo metodo gestisce eventi dei campi Timeslot
 *Blocco di start e stop e il numero di livelli. */
public void actionPerformed(ActionEvent e) {
    text=textField.getText();
```

```
        text1=textFieldStart.getText();
        text2=textFieldStop.getText();
        level=levelNumber.getText();
    }

    /** Questo medoto reagisce agli eventi di cambiamento nei vari menu.
    Opt*/
    public void propertyChange(PropertyChangeEvent e)
    {
        String prop = e.getPropertyName();
        value = optionPane.getValue();
                if (btnString1.equals(value))
                {
         text=textField.getText();
         text1=textFieldStart.getText();
         text2=textFieldStop.getText();
         level=levelNumber.getText();
        }
    }

    public void itemStateChanged (ItemEvent e)
    { }
        //Metodo che restituisce il valore del pulsante premuto
    public Object getValue()
        { return value; }
    // Metodo che restituisci il valore del Blocco di Start inserito
    public Object getTxt1()
    { return text1; }
    // Metodo che restituisci il valore del Blocco di Stop inserito
    public Object getTxt2()
    { return text2; }
    // Metodo che restituisce il valore del TimeSlot inserito
    public Object getTxt()
    { return text; }
    // Metodo che restituisce il numero di livelli inserito
    public Object getLevel()
    { return level; }
}
```

A.5 DWT.java

```
import java.io.*;
import java.*;

public class DWT {
//vettore dei coefficienti
private double coefficient[];
//Vettore di input
private double input[];
// Vettore contente i risultati intermedi dei filtri
private double tmp[];
//Contiene la lunghezza del file da analizzare
private int fileLenght;
// puntatore del vettore dei coefficienti
private int m=0;
// puntatore del vettore input
private int v=0;
//indica il tipo di wavelet che implementiamo nel filtro
private int wavelet;
private File file;
private FileOutputStream filestream;
private PrintStream fileOut;
// indica il numero progressivo di file creato
private int numFile;
//variabili utilizzate per calcolare quantile e varianza
private double y;
private double x;
// variabile che contiene il numero di livelli
private int lev;
// Contiene le somme parziali per il cacolo della media
private double total;
// contiene la media
private double average;
// Flag per terminare il while nel calcolo del quantile
private boolean endWhile = true;
// variabili che contengono i booleani per calcolo quantile e varianza
private boolean var=false;
private boolean quan=false;
// vettore che contiene i valori calcolati del quantile per ogni livello
private double quantile[];
// vettore che contiene i valori calcolati della varianza per ogni livello
private double varianza[];
// variabile utilizzata per calcolare la varianza
private double test=0;
// variabile che contiene il valore scelto per il calcolo del quantile
private int quantileNumber;


// Costrutture che riceve in ingresso il file, il tipo di wavelet,
// la dimensione del file, il numero di livelli e il numero di campioni al
```

```
0pt primo livello
// il flag relativo al quantile e il flag relativo alla varianza.
 public DWT(RandomAccessFile f, int i,int p,int selectedLevel,double
 0ptsample,boolean q,boolean v,int qn)
 {
  try
  {
   lev=selectedLevel;
   fileLenght=(int)sample;
   numFile=p;
   var=v;
   quan=q;
   quantileNumber=qn;
   double matrix[][] =
   0pt{{2,1,1},{4,0.6830127,1.1830127,0.3169873,-0.1830127},{6,0.47046721,
   0pt1.14111692,0.650365, -0.19093442,-0.12083221,0.0498175}};
   wavelet = (int)matrix[i][0];
   coefficient = new double[wavelet];
   // Inizializzo vettore coefficienti in base al tipo di wavelet da
   0ptutilizzare
   for (int j=0; j < wavelet; j++)
    coefficient[j] = matrix[i][j+1];
   input =new double[fileLenght];
   tmp=new double[fileLenght];
   varianza= new double [lev*2];
   quantile= new double [lev*2];
   // Copio il file da elaborare in un vettore
   for (int j=0; j<fileLenght;j++)
    { input[j] = Integer.parseInt(f.readLine()); }
  }//Fine try
  catch (IOException e){}
 }// Fine costruttore

 public double[] doDWT(){
  try
  {
        file= new File ("file"+numFile);
        filestream = new FileOutputStream(file);
        fileOut=new PrintStream(filestream);
        // Implementazione filtro LowPass
        int meta=fileLenght;
    //Ciclo che definisce il livello di profondità dell'albero
    for (int level =0; level<lev; level++)
    {
     v=0;
     total=0;
     //fileOut.println("Filtro passa basso, livello "+level);
     while(v<meta)
     {
      double t=0;
      int l=v;
```

```
 for(m=0; m<wavelet; m++)
 {
   if (l<0) y=0;
   else y=input[l];
   t= t+ coefficient[m]*y;
   l--;
 }
     // Somme parziali per il cacolo della media
 total=total+t;
 tmp[v]=t;
 v++;
 fileOut.println(t);
}//Fine while filtro low pass
// controlla il flag della varianza e ne esegue il calcolo
if (var==true)
{
   test=0;
   average=total/meta;
   for(int i=0; i<meta; i++)
   { test = test + ((tmp[i]-average)*(tmp[i]-average)); }
   varianza[level]= test/meta;
} // Fine if calcolo della varianza
//controlla il flag del quantile e ne esegue il calcolo
if (quan==true)
{
 average=total/meta;
 double arraytmp[] = new double [meta];
 //copio l'array di input per ordinarli
 for(int i=0; i<meta; i++)
 { arraytmp[i] = tmp[i]; }
 QuickSort QS = new QuickSort();
 arraytmp=QS.quicksort(arraytmp);
 int tempq=(int)((meta-1)*quantileNumber/100)+1;
 quantile[level] =arraytmp[tempq-1]-average;
} // Fine if calcolo del quantile
//Filtro Passa alto
v=0;
double [] tmpH=new double[meta];
total =0;
fileOut.println("Filtro passa alto, livello "+level);
while(v<meta)
{
 double t=0;
 int l=v;
 for(m=wavelet-1; m>0;m = m-2)
 {
   if (l<0) y=0;
   else y=input[l];
   if (l-1<0) x=0;
   else x=input[l-1];
   t= t+ (coefficient[m]*y)-(coefficient[m-1]*x);
```

```
      l=l-2;
    }
    total= total+t;
    tmpH[v]=t;
    v++;
    fileOut.println(t);
   }//Fine while per il filtro high pass
   // calcolo varianza filtro passa basso
   if (var==true)
   {
      test=0;
      average=total/meta;
      for(int i=0; i<meta; i++)
      { test = test +(tmpH[i]-average)*(tmpH[i]-average); }
      varianza[level+lev]= test/meta;
   } // Fine if calcolo della varianza

   // calcolo del quantile filtro passa basso
   if (quan==true)
   {
      average=total/meta;
      double arraytmp[] = new double [meta];
      //copio l'array di input per ordinarli
      for(int i=0; i<meta; i++)
              arraytmp[i] = tmpH[i];
      QuickSort QS = new QuickSort();
      QS.quicksort(arraytmp);
      int tempq=(int)((meta-1)*quantileNumber/100)+1;
              quantile[level+lev]=arraytmp[tempq-1]-average;
    } // Fine if calcolo del quantile
    //Creazione array dopo downsampling per il livello successibo
    int j=0;
    double input[]=new double[meta/2];
    for(int i=1; i<meta; i=i+2)
    {
      input[j]=tmp[i];
      j++;
    }
        meta=meta/2;
    double tmp[]=new double[meta];
  }//Fine for level
  fileOut.close();
  file.delete();
}//Fine try
catch (IOException e){}
if ((quan==false)&&(var==true))
   return varianza;
if ((quan==true)&&(var==false))
       return quantile;
if ((quan==true)&&(var==true))
       {
```

```
            double risultati[]=new double[lev];
            return risultati;
          }
        return null;
        }//Fine doDWT

    }
```

A.6 Graphic.java

```
/* ===========================================================
 * JFreeChart : a free chart library for the Java(tm) platform
 * ===========================================================
 *
 * (C) Copyright 2000-2004, by Object Refinery Limited and Contributors.
 *
 * Project Info:  http://www.jfree.org/jfreechart/index.html
 *
 * This library is free software; you can redistribute it and/or modify it
 0pt under the terms
 * of the GNU Lesser General Public License as published by the Free
 0ptSoftware Foundation;
 * either version 2.1 of the License, or (at your option) any later
 0ptversion.
 *
 * This library is distributed in the hope that it will be useful, but
 0ptWITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 0ptPARTICULAR PURPOSE.
 * See the GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 0ptLicense along with this
 * library; if not, write to the Free Software Foundation, Inc., 59 Temple
 0pt Place, Suite 330,
 * Boston, MA 02111-1307, USA.
 *
 * [Java is a trademark or registered trademark of Sun Microsystems, Inc.
 * in the United States and other countries.]
 * (C) Copyright 2002-2004, by Object Refinery Limited and Contributors.
 *
 * Original Author:  David Gilbert (for Object Refinery Limited);
 * Contributor(s):   -;
 *
 * $Id: LineChartDemo1.java,v 1.27 2004/05/27 09:10:42 mungady Exp $
 *
 * Changes
 * -------
 * 08-Apr-2002 : Version 1 (DG);
 * 30-May-2002 : Modified to display values on the chart (DG);
 * 25-Jun-2002 : Removed redundant import (DG);
 * 11-Oct-2002 : Fixed errors reported by Checkstyle (DG);
 *
 */



import java.awt.BasicStroke;
import java.awt.Color;
```

```
import java.awt.Dimension;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.category.LineAndShapeRenderer;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;
import javax.swing.*;
import java.awt.*;



public class Graphic extends ApplicationFrame {

//indica se è passa alto o passa basso
private int num;
//variabile che contiene il numero di livelli.
private int level;
// variabile che indica il numero di righe.
private int riga;
// variabile che contiene i "nomi" delle righe.
private final String row[];
// variabile che contiene i "nomi" delle colonne.
private final String column[];
// crea un nuovo pannello
private JFrame q=new JFrame();
// contiene i valori utilizzati per creare il grafico.
private double serie[];
// crea un dataset che andrà popolato per creare il grafico.
private final DefaultCategoryDataset dataset;
final LineAndShapeRenderer renderer ;
private double value[];
int a=10;
int b=9;
int c=8;
int d=7;
int e=6;
int f=5;
int g=4;
int h=3;
int l=2;
int p=1;
int m=0;
int cont=0;

public Graphic(final String title)
```

```
                {
        super(title);
        row=new String [1];
                column=new String [1];
                dataset = new DefaultCategoryDataset();
                final CategoryPlot plot = new CategoryPlot();
                renderer = (LineAndShapeRenderer) plot.getRenderer();
        }

    // metodo cotruttore che riceve in ingresso il titolo del grafico, il
    0ptnumero di livelli,il numero di intervalli analizzati,
    // il vettore contenente i dati,l'indice della posizione attauale del
    0ptvettore.
    // il vettore è strutturato nel seguente modo:
    // per ogni intervallo contiene tutti i risultati parziali dei livelli
    0pt dei filtri passa-basso, seguiti dai risultati
    // parziali dei livelli dei filtri passa-alto dello stesso intervallo.

public Graphic(final String title,int l,int p,int n)
                {
        super(title);
        level=l;
        num=p;
        riga=0;
        value=new double [level*10];
        row=new String [num*2];
                column=new String [level];

        int length= level*2*num;
                // row keys...
                for(int k=0;k<num;k++)
          { row[k]=0+"th";}
        // column keys...
        for(int j=0;j<level;j++)
         {
          int d=j+1;
          column[j]="level "+d;
         }
                dataset = new DefaultCategoryDataset();
                //final CategoryDataset dataset = new CategoryDataset() ;
        final JFreeChart chart = createChart(dataset,title);
        final ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new Dimension(625, 335));
        q.add(chartPanel);
        q.pack();
        q.setPreferredSize(new Dimension(625,335));
        q.setVisible(true);
        final CategoryPlot plot = (CategoryPlot) chart.getPlot();
        plot.setBackgroundPaint(Color.white);
        plot.setRangeGridlinePaint(Color.white);
        renderer = (LineAndShapeRenderer) plot.getRenderer();
```

```
        setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
        if (n==0) q.setLocation(0,80);
        if (n==1) q.setLocation(630,80);
                }

public void UpdateChart(double array[],int u)
     {

        if(a<0) a=10;
        renderer.setSeriesPaint(a, new Color(255,255,0));
        a--;
        if(b<0) b=10;
        renderer.setSeriesPaint(b, new Color(0,0,255));
        b--;
        if(c<0) c=10;
        renderer.setSeriesPaint(c, new Color(255,0,255));
        c--;
        if(d<0) d=10;
        renderer.setSeriesPaint(d, new Color(0,255,255));
        d--;
        if(e<0) e=10;
        renderer.setSeriesPaint(e, new Color(255,134,68));
        e--;
        if(f<0) f=10;
        renderer.setSeriesPaint(f, new Color(90,32,255));
        f--;
        if(g<0) g=10;
        renderer.setSeriesPaint(g, new Color(255,1,100));
        g--;
        if(h<0) h=10;
        renderer.setSeriesPaint(h, new Color(100,35,255));
        h--;
        if(l<0) l=10;
        renderer.setSeriesPaint(l, new Color(208,40,168));
        l--;
        if(p<0) p=10;
        renderer.setSeriesPaint(p, new Color(0,0,0));
        p--;
        if(m<0) m=10;
        renderer.setSeriesPaint(m, new Color(255,20,180));
        m--;
       for ( int r=0;r<=8;r++)
        row[r]=row[r+1];
       riga++;
       row[9]=riga+"th";
       for (int j=0;j<9;j++)
        for (int k=0;k<level;k++)
        {
                value[(j*level)+k]=value[((j+1)*level)+k];
        }
       for(int k=0;k<level;k++)
```

```
      {

        value[(9*level)+k]=Math.log(array[k+u])/Math.log(2);
        System.out.println("value "+value[(9*level)+k]);
        if(value[(9*level)+k]<-10)
           value[(9*level)+k]=-10;
      }
      dataset.clear();
      if(riga<9)
        {
        for (int j=0;j<9;j++)
          for (int k=0;k<level;k++)
                dataset.addValue(value[(j*level)+k],row[j],column[k] );
        }
      else
      {
       for (int j=0;j<9;j++)
        for (int k=0;k<level;k++)
                dataset.addValue(value[(j*level)+k],row[j],column[k] );
      }
     }

private JFreeChart createChart(final CategoryDataset dataset,String h)
         {
      // crea il grafico...
       final JFreeChart chart = ChartFactory.createLineChart(
            h,                                      // titolo del grafico
            "Level",                  // titolo del domain axis
            "Log",                    // titolo del range axis
            dataset,                  // data
            PlotOrientation.VERTICAL,  // orientamento
            true,                     // include legenda
            true,                     // tooltips
            false                     // urls
        );
        //settaggio dei parametri del grafico
        chart.setBackgroundPaint(Color.white);
                final CategoryPlot plot = (CategoryPlot) chart.getPlot();
        plot.setBackgroundPaint(Color.white);
        plot.setRangeGridlinePaint(Color.white);
                // customizza il range axis...
        final NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
        rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits()
        0pt);
        rangeAxis.setAutoRangeIncludesZero(true);
        // customizza the renderer...
        final LineAndShapeRenderer renderer = (LineAndShapeRenderer) plot.
        0ptgetRenderer();
        return chart;
    }
}
```

A.7 GraphicTime.java

```
/* ===========================================================
 * JFreeChart : a free chart library for the Java(tm) platform
 * ===========================================================
 *
 * (C) Copyright 2000-2004, by Object Refinery Limited and Contributors.
 *
 * Project Info:  http://www.jfree.org/jfreechart/index.html
 *
 * This library is free software; you can redistribute it and/or modify it
 0pt under the terms
 * of the GNU Lesser General Public License as published by the Free
 0ptSoftware Foundation;
 * either version 2.1 of the License, or (at your option) any later
 0ptversion.
 *
 * This library is distributed in the hope that it will be useful, but
 0ptWITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 0ptPARTICULAR PURPOSE.
 * See the GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 0ptLicense along with this
 * library; if not, write to the Free Software Foundation, Inc., 59 Temple
 0pt Place, Suite 330,
 * Boston, MA 02111-1307, USA.
 *
 * [Java is a trademark or registered trademark of Sun Microsystems, Inc.
 * in the United States and other countries.]
 * (C) Copyright 2002-2004, by Object Refinery Limited and Contributors.
 *
 * Original Author:  David Gilbert (for Object Refinery Limited);
 * Contributor(s):   -;
 *
 * $Id: LineChartDemo1.java,v 1.27 2004/05/27 09:10:42 mungady Exp $
 *
 * Changes
 * -------
 * 08-Apr-2002 : Version 1 (DG);
 * 30-May-2002 : Modified to display values on the chart (DG);
 * 25-Jun-2002 : Removed redundant import (DG);
 * 11-Oct-2002 : Fixed errors reported by Checkstyle (DG);
 *
 */



import java.awt.BasicStroke;
import java.awt.Color;
```

```
import java.awt.Dimension;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.axis.CategoryAxis;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.category.LineAndShapeRenderer;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;
import javax.swing.*;
import java.awt.*;



public class GraphicTime extends ApplicationFrame {

//variabile che contiene il numero di livelli.
private int level;
// variabile che indica il numero di righe.
private int riga;
// variabile che contiene i "nomi" delle righe.
private final String row[];
// variabile che contiene i "nomi" delle colonne.
private final String column[];
// crea un nuovo pannello
private JFrame q=new JFrame();
// contiene i valori utilizzati per creare il grafico.
private double serie[];
// crea un dataset che andrà popolato per creare il grafico.
private final DefaultCategoryDataset dataset;
// variabile che indica il numero di blocco visualizzato.
private int time=0;
// variabile che indica i blochi da visualizzare nella finestra
private int block=25;

public GraphicTime(final String title)
                {
         super(title);
         row=new String [1];
                 column=new String [1];
                 dataset = new DefaultCategoryDataset();
        }

    /* metodo cotruttore che riceve in ingresso il titolo del grafico,
     il numero di livelli,il numero di intervalli analizzati,
     il vettore contenente i dati,l'indice della posizione attauale del
     0ptvettore.
```

```
     il vettore è strutturato nel seguente modo:
     per ogni intervallo contiene tutti i risultati parziali dei livelli
     dei filtri passa-basso, seguiti dai risultati parziali dei
     livelli dei filtri passa-alto dello stesso intervallo.*/

public GraphicTime(final String title,int l,int n)
                {
                super(title);
                level=l;
                row=new String [level];
                        column=new String [block];
                        JFrame.setDefaultLookAndFeelDecorated(true);
                        // assegna i nomi alle righe
                        for(int k=0;k<level;k++)
                        {
                        int h=k+1;
                        row[k]="level "+h;
                        }
                // assegna i nomi alle colonne
                for(int j=0;j<block;j++)
                         column[j]=""+0;


                serie= new double[level*block];
                for (int i=0;i<(level*block);i++)
                  serie[i]=0;
                        dataset = new DefaultCategoryDataset();
                final JFreeChart chart= createChart(dataset,title);
                final ChartPanel chartPanel = new ChartPanel(chart);
                chartPanel.setPreferredSize(new Dimension(1270,280));
                q.add(chartPanel);
                q.setPreferredSize(new Dimension(1270,280));
                q.setVisible(true);
                q.pack();
                setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
                if (n==0) q.setLocation(0,440);
                if (n==1) q.setLocation(0,715);
                 }


public void UpdateChart(double array[],int u)
    {

     for (int i=0;i<level;i++)
        for (int j=0;j<block-1;j++)
                serie[j+(block*i)]=serie[j+(block*i)+1];
     for (int i=0;i<level;i++)
                serie[(block-1)+(block*i)]=array[i+u];
     dataset.clear();
     for (int j=0;j<(block-1);j++)
                column[j]=column[j+1];
```

```
     column[block-1]=""+time;
     time++;
     int index=0;
     for (int i=0;i<level;i++)
      for (int j=0;j<block;j++)
                {
                 double l=Math.log(serie[index])/Math.log(2);
                 dataset.addValue(serie[index],row[i],column[j]);
                 index++;
                }
    }

private JFreeChart createChart(final CategoryDataset dataset,final String
0pth)
         {
        // crea il grafico...
        final JFreeChart chart = ChartFactory.createLineChart(
            h,                                          //titolo del
            0ptgrafico
            "Block",                    // titolo del domain axis
            "Value",                    // titolo del range axis
            dataset,                    // data
            PlotOrientation.VERTICAL,   // orientamento
            true,                       // include legenda
            true,                       // tooltips
            false                       // urls
        );

        chart.setBackgroundPaint(Color.white);
                final CategoryPlot plot = (CategoryPlot) chart.getPlot();
        plot.setBackgroundPaint(Color.white);
        plot.setRangeGridlinePaint(Color.white);
                //customizza il rangeaxis
        final NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
        //rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits
        0pt());
        rangeAxis.setAutoRangeIncludesZero(true);
        rangeAxis.setTickLabelFont(new Font("SansSerif",Font.PLAIN,10));
        //customizza il domainAxis
        final CategoryAxis DomainAxis =  plot.getDomainAxis();
                DomainAxis.setTickLabelFont(new Font("SansSerif",Font.
                0ptPLAIN,8));
        // customizza the renderer...
        final LineAndShapeRenderer renderer = (LineAndShapeRenderer) plot.
        0ptgetRenderer();
        return chart;
    }
}
```

A.8 PacketElaboration.java

```
import java.math.*;

/************************************************
Questa classe contiene i metodi per l'elaborazione
dell'header di pacchetto
************************************************/
public class PacketElaboration
{
        //time = tempo nel formato secondi.microsecondi
        private double time;

        double sec=0;
        double usec=0;
        /*moltiplicatore in base 256 per la conversione
         da byte a numeri reali*/
    private long pot;
    public PacketElaboration() {}

    /************************************************
        metodo per la conversione del timestamp da byte a
        tempo in secondi.microsecondi
        ************************************************/
    public double TimeValue (int []vector, boolean indian)
    {
        time =0;
        sec=0;
        usec=0;
        pot= 1;
        if (indian == true)
        {
         for (int i =0; i<4; i++)
         {
                sec = sec+(vector[i]*pot);
                usec=usec+(vector[i+4]*pot);
                pot= pot*256;
         }// Fine for
        }
        else
        {
         for (int i =4; i>0; i--)
         {
                sec = sec+(vector[i]*pot);
                usec=usec+(vector[i+4]*pot);
                pot= pot*256;
         }// Fine for
        }
        time=sec+(usec*(Math.pow(10,-6)));
        return time;
    } // Fine metodo TimeValue
```

```
    /************************************************
    Metodo per estrarre dall'header la lunghezza del
    pacchetto
    ************************************************/
    public int Plength(int []vector, boolean indian)
    {

        int l;
        if (indian == true)
                l = (vector[9]*256) + vector[8];
        else
                l = (vector[10]*256) + vector[11];
        return l;
    }//Fine Metodo Plength
}
```

A.9 Progress.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

/************************************************
Classe che implementa la barra di progressione per il
caricamento dei file.
************************************************/
public class Progress extends JFrame{

        JProgressBar current;
  public Progress(RandomAccessFile f)
    {
     super("Progress");
     // Creo la barra di avanzamento
        setTitle("Loading..");
    setSize(100,50);
    pack();
    setDefaultCloseOperation(HIDE_ON_CLOSE);
    try { current = new JProgressBar(0,(int)f.length());}
    catch(java.io.IOException e){}
        current.setValue(0);
        current.setStringPainted(true);
        setContentPane(current);
    setLocation(300,200);
    setVisible(true);
    }

        //Metodo che riceve in ingresso il numero di pacchetti e imposta
        //il valore della variabile current per valutare la % di
        0ptavanzamento
    public void avanzamento(int v)
    {current.setValue(v);}
}
```

A.10 QuickSort.java

```
public class QuickSort
 {
    private static long comparisons = 0;
    private static long exchanges   = 0;
   /*********************************************************************
   0pt
    *  Implementazione dell'algoritmo Quicksort
    *********************************************************************/
    0pt
    public static double[] quicksort(double[] a)
        {
        shuffle(a);
        quicksort(a, 0, a.length - 1);
        return a;
        }

    public static void quicksort(double[] a, int left, int right)
        {
        if (right <= left) return;
        int i = partition(a, left, right);
        quicksort(a, left, i-1);
        quicksort(a, i+1, right);
        }

        private static int partition(double[] a, int left, int right)
                {
        int i = left - 1;
        int j = right;
        while (true)
                {
            while (less(a[++i], a[right]))
                ;
            while (less(a[right], a[--j]))
                if (j == left) break;
            if (i >= j) break;
            exch(a, i, j);
                }
        exch(a, i, right);
        return i;
        }

    private static boolean less(double x, double y)
        {
        comparisons++;
        return (x < y);
        }

    private static void exch(double[] a, int i, int j)
        {
```

```
            exchanges++;
            double swap = a[i];
            a[i] = a[j];
            a[j] = swap;
            }

        private static void shuffle(double[] a)
            {
            int N = a.length;
            for (int i = 0; i < N; i++)
                    {
                int r = i + (int) (Math.random() * (N-i));
                exch(a, i, r);
                    }
                    }
}
```

Bibliografia

[1] D. Esteband and C. Galand. *Application of quadrature mirror filters to split band voice coding schemes*, Proc. int Conf. Acoust. Speech, signal Processing, May 1977.

[2] Millar and C. Paul. *Recursive quadrature mirror filters: Criteria specification and design method*, IEEE Trans. Acoust. Speech Signal Procesing, vol 33. pp 413-420 Apr. 1985.

[3] G. Pirani and V. Zingarelli. *Analytical formula for design of quadrature mirror filter*, IEEE Trans. Acoust. Speech, Signal Procesing, vol ASSP-32. pp 645-648 June. 1984.

[4] L. Daubechies. *Orthonormal bases of compactly supported waveltes*, Commun. Appl. Math., Vol 41, pp 909-996. Nov. 1988

[5] Meyer. *Ondelettes et Operateurs*, Hermann 1988

[6] P.G Lemarie and Y. Meyer. *Ondelettes et bases Hilbertiennes*, Revista Matematica Ibero Americana, Volume 2, 1986

[7] *D. Marr Vision*New York Freeman 1982

[8] *Sito di riferimento per la libreria JPCAP.*
`http://netresearch.ics.uci.edu/kfujii/jpcap/doc/`

[9] Dindo Stefano. *Realizzazione di software per la conversione di formati nell'analisi di tracce di traffico.*
`http://www.dindo.biz/publications/Tesi_Conversione_Tracce.pdf`

[10] David Salomon *Data Compression - The Complete Reference 4th Edition*, Springer, 2007

[11] *Free Java Chart Library*
`http://www.jfree.org/jfreechart/`

[12] *Java 2D Graphic GUI*
`http://www.java2s.com/Code/Java/CatalogJava.htm`

[13] P Abry, D. Veitch, and P. Flandring *Long range dependence: Revisiting aggregation with wavelts*, Journal of Time Series Analysis, 19(3): 253-266, 1998

[14] W.Leland, M.Taqqu, W. Willinger, and D. Wilson. *On the self-similar nature of ethernet traffic*, (extended version). IEEE/ACM Trans. on Information Theory, 2(1):1-15, Feb. 1994.

[15] S. Stoev, M.Teqqu, and J. Marron.. *On the wavelet spectrum diagnostic for hurst parameter estimation in the analysis of the internet traffic*, Computer Networks, 48(3): 423-445, June 2005.

[16] M. Unser and T. Blu. *Wavelet theory demystified.*, IEEE Trans. on Signal Processing, 51(2): 470-483, Feb. 2003.

[17] G. Giorgi and C. Narduzzi. *Rate-Interval Curves: tool for the analysis and monitoring of network traffic*, Performance Evaluation, 65(6-7):441-462, June 2008.

[18] G. Giorgi and C. Narduzzi. *A study of measurement-based traffic models for network diagnostics*, In Proc. IEEE Instrum. Meas. Tech. Conf. IMTC07, 01-03 May 2007. Warsaw, Poland.

[19] S. Sarvotham, R. Riedi, and R. Baraniuk. *Network and user driven alpha-beta on-off source model for network traffic*, Computer Networks, 48(3):335-350, June 2005.

[20] Taqqu M.S., Willinger W. and Sherman R., *Proof of a Fundamental Result in Self-Similar Traffic Modeling*, Computer Communication Rewiw, Vol. 27, No 2 April 1997.

[21] Taqqu M.S., Willinger W., Sherman R. and Wilson D.V. *Self-similarity through High Variability: Statistical Analysis of Ethernet LAN Traffic at Source Level*, IEEE/ACM Transactions on Networking, Vol. 5, No. 1, February 1997.

[22] Stallings W. *High-Speed Networks TCP/IP and ATM Design Principles*, Prentice-Hall, Inc. 1998.

[23] Taqqu M.S. and Teverovsky V., *On Estimating the Intersity of Long-Range Dependence in Finite ond Infinite Variance Time Series*, preprint Boston University USA, 1996.

[24] G. Giorgi, S.Dindo, M. Vantini, C. Narduzzi, *Scaling Analysis of Wavelet Quantiles in Network Traffic*, International Conference on Performance Evaluation Methodologies and Tools 2008.

Ringraziamenti

Giunto a questo traguardo sento la necessità di ringraziare tutti coloro che mi hanno accompagnato e sostenuto in questi anni di studio.

Ringrazio il Prof. Claudio Narduzzi che mi ha permesso di conoscere il settore dell'attività scientifica e di acquisire nuove competenze.

Ringrazio l'Ing. Giada Giorgi per la sua amicizia, per avermi saputo ascoltare ed indirizzare nei momenti di difficoltà durante la tesi.

Ringrazio Marco Vantini per la sua Amicizia e per i cinque anni indimenticabili trascorsi insieme tra libri ed esami.

Ringrazio i miei genitori per avermi permesso di condurre gli studi in modo sereno e per l'esempio di unione e determinazione che hanno mostrato nel raggiungere gli obiettivi.

Ringrazio mio fratello Roberto per essere stato un punto di riferimento da seguire negli anni.