UNIVERSITA' DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di laurea in Ingegneria Informatica

REALIZZAZIONE DI SOFTWARE PER LA CONVERSIONE DI FORMATI NELL'ANALISI DI TRACCE DI TRAFFICO.

Relatore: Ch.mo Prof. Claudio Narduzzi

Laureando: Stefano Dindo

ANNO ACCADEMICO 2005 - 2006

INDICE:

INTRODUZIONE: 5

CAPITOLO 1 – FILTRI PER LA CATTURA DI PACCHETTI 9

1.1 Berkeley Packet Filter 9

1.2 Architettura WinPcap 11

1.3 Modalità di funzionamento di WinPcap 12
1.3.1 Promiscuous mode 13
1.3.2 Statistics mode 13

1.4 Soluzioni avanzate per l'analisi di rete 14
1.4.1 Monitoraggio del traffico 14
1.4.2 Salvataggio del traffico su disco 16

CAPITOLO 2 - ETHEREAL 17

2.1 Breve storia di Ethereal 17

2.2 Uso di Ethereal 18
2.2.1 Il menu 19
2.2.2 Menu, Edit, View, Go 20
2.2.3 Captured 23
2.2.4 Analyse 27
2.2.5 Statistics 30

2.3 Filtraggio durante la cattura 33

2.4 Principio di funzionamento di Ethereal 35

CAPITOLO 3 – FORMATO DI CATTURA PER SCHEDE ENDACE DAG 38

3.1 Type 1: record PoS HDLC a lunghezza variabile 39

3.2 Type 2: record Ethernet a lunghezza variabile 40

3.3 Type 4: AAL 5 Frame Record 40

CAPITOLO 4 – FORMATO TCPDUMP 42

4.1 Specifiche dei byte che compongono l'header di pacchetto 43

CAPITOLO 5 – DESCRIZIONE DEL PROGRAMMA TIME INTERVAL EXTRACTOR 46

5.1 Struttura del programma Time Interval Extractor 48

5.2 Analisi del programma 51
5.2.1 ConversioneTS.java 51
5.2.2 Converti.java 53
5.2.3 TimeIntervalExtractor.java 54

CONCLUSIONI **57**

CODICE SERGENTE DEL PROGRAMMA **58**
Codice della classe ConversioneTS.java 58
Codice sorgente della classe Converti.java 59
Codice sorgente della classe TimeIntervalExtractor.java 61

BIBLIOGRAFIA **70**

INTRODUZIONE:

La rapida evoluzione delle reti informatiche, la riduzione dei costi dei Personal Computer e la diffusione di internet hanno contribuito a cambiare il sistema di comunicazione degli utenti portando ad un aumento esponenziale del numero di utilizzatori di internet. Questa crescita ha portato ad un aumento della complessità delle reti esponendo a numerosi rischi le informazioni che vi transitano e gli utenti che si trovano a fronteggiare ogni giorno minacce quali virus, worms, trojan e spam. Per questo motivo si è sentita la necessità di sviluppare dispositivi capaci di analizzare e monitorare le informazioni che transitano nella rete sotto forma di pacchetti. Da notare che una volta catturati i pacchetti è possibile accedere al loro contenuto, entrando in possesso di informazioni riservate e che quindi queste operazioni devono essere svolte rispettando la legge sulla privacy.

Recentemente si sono affermate due tipologie di strumenti: hardware e software.

L'hardware propone soluzioni più statiche, difficili da personalizzare e da aggiornare ma capaci di maggiore velocità, mentre il software si distingue per le doti di versatilità e personalizzazione. In entrambe le tipologie di strumenti si pone il problema del timestamping, ovvero il salvataggio del tempo assoluto di arrivo nel punto di accesso al mezzo. In entrambi i casi si è cercato di ottenere un grado di precisione in rapporto all'aumento esponenziale delle velocità di connessione.

I software per la cattura e l'analisi del traffico sono chiamati *sniffer*. Questi catturano i pacchetti che viaggiano sulla rete senza modificarli, li acquisiscono, nella loro forma frammentaria o nella loro forma riassemblata, ed eventualmente li salvano per l'analisi successiva. Uno sniffer può essere utilizzato per intercettare vari tipi di protocolli di comunicazione: Ethernet, TCP, IP e molti altri ancora.

Le funzioni tipiche di uno sniffer possono essere riassunte come di seguito:

- Conversione e filtraggio delle informazioni presenti nei pacchetti in una forma leggibile dall'utente.
- Analisi dei malfunzionamenti della rete.
- Analisi di qualità e portata della rete.

- Ricerca autorizzata di password e nomi utenti in chiaro o cifrati per l'analisi successiva.
- Creazione dei log del traffico sulla rete.
- Ricerca di intrusioni in rete attraverso l'analisi dei log di traffico.

Gli sniffer sono caratterizzati dalla necessità di accedere a basso livello ai dati che scorrono sulla rete, ossia necessitano di un meccanismo che estragga i pacchetti che transitano sul canale prima che questi vengano manipolati dagli stack di rete.

I prodotti software che si sono affermati maggiormente in questo ambito sono Ethereal e WinPcap. Ethereal è un programma Open Source che grazie alla sua facilità ed alla qualità dei risultati che è in grado di offrire è riuscito a superare programmi sviluppati da aziende.

WinPcap, al contrario di Ethereal, è una libreria che aggiunge la funzionalità di cattura e analisi di rete ai sistemi operativi Windows. Grazie alla sua versatilità e alle sue prestazioni, WinPcap ha permesso ai sistemi operativi Microsoft di entrare in un settore che sembrava esclusivamente di competenza di sistemi Unix.

Per quanto riguarda la soluzione hardware, l'azienda Endace in particolare, propone diverse soluzioni studiate espressamente per dare la possibilità di eseguire misure precise anche in caso di connessioni ad alta velocità, cosa che strumenti software non sono in grado di fare con accuratezza. L'Endace produce schede dedicate alle misure su rete, chiamate schede DAG, con hardware progettato per fornire timestamping di elevata precisione. I motivi per cui si ricorre ad una soluzione hardware sono la mancanza di una scheda di rete per il livello fisico in questione ed il fatto che le Network Interface Card non sono progettate per dare priorità alle misurazioni. Le schede DAG consentono l'acquisizione di pacchetti con timestamping a precisione elevata, cosa che una normale NIC (Network interface card) non è in grado di fare in quanto le operazioni che deve svolgere per assegnare il timestamp sono molte, e il processo è impegnativo a livello hardware. Questo è dovuto al fatto che normalmente la NIC si limita a segnalare l'arrivo del pacchetto tramite un interrupt, che verrà servito solamente dopo che la CPU dell'host si libera. L'intervallo di tempo che intercorre tra il momento in cui la CPU riceve l'interrrupt e il momento in cui serve l'interruzione è detto *latenza di interrupt.*

Alcune soluzioni prevedono che la NIC non generi l'interruzione alla ricezione di ogni pacchetto, ma che generi l'interruzione quando sono arrivati diversi pacchetti in modo da ridurre il carico sull'host. Così facendo però verrà assegnato lo stesso timestamp a tutti i pacchetti del gruppo facendo sembrare che siano arrivati tutti simultaneamente.

In questa tesi tratteremo principalmente l'aspetto software del monitoraggio di reti trattando inizialmente la struttura della libreria WinPcap e spiegando le caratteristiche principali di Ethereal. Poiché i formati di dati utilizzati da questi ambienti sono diventati quasi degli standard, lo scopo di questa tesi sarà quello di considerare la conversione dei dati forniti dalla scheda DAG, in formato proprietario ma ben documentato, nel formato TCPDUMP comunemente utilizzato da Ethereal. A questo scopo saranno in primo luogo presentate le caratteristiche principali dei formati interessati coinvolti nella conversione. In fine, sarà illustrato in dettaglio il software che effettua la conversione delle tracce di rete dal formato DAG al formato TCPDUMP.

CAPITOLO 1 – Filtri per la cattura di pacchetti

Il processo di cattura dei pacchetti da un punto di osservazione nella rete avviene attraverso una sequenza di operazioni implementate da un insieme di moduli software.

Al livello più basso, i filtri per la cattura dei pacchetti interagiscono con l'hardware dell'interfaccia fisica e si occupano della lettura, selezione ed acquisizione dei pacchetti di interesse. L'impostazione delle regole di cattura dei filtri possono essere impostate direttamente dai software a livello utente.

Le librerie più diffuse nei sistemi Unix e Windows per la cattura dei pacchetti sono Libpcap e Winpcap.

Libpcap si basa sul filtro Berkeley Packet Filter mentre Winpcap è sviluppata partendo dal BPF e aggiungendo una serie di miglioramenti nei vari blocchi funzionali per migliorare le prestazioni, inoltre Winpcap garantisce la compatibilità con i sistemi operativi Windows.

In questo capitolo presenteremo rapidamente il BPF e Winpcap evidenziando che la cattura dei pacchetti è un fattore importante sulle prestazioni dell'intero sistema di analisi del traffico di rete.

1.1 Berkeley Packet Filter

Il BPF è il sistema di cattura che viene solitamente utilizzato in sistemi operativi della famiglia BSD. Esso è considerato il migliore e più efficiente sistema di sniffing fra quelli offerti in sistemi Unix. L'architettura del BPF può essere schematizzata distinguendo due moduli principali:

- Il BPF è un'estensione del sistema operativo che opera all'interno del kernel e si interfaccia con la scheda di rete.
- Una libreria, funzionante a livello utente, che attiva e configura il BPF a livello kernel; un esempio è la libreria libpcap, che esporta i servizi del BPF alle applicazioni tramite una API[1] ad alto livello.

Il livello di astrazione della libreria libpcap è tale da permettere il suo perfetto funzionamento su sistemi operativi diversi da BSD come Linux, Solaris ecc … non necessariamente dotati di BPF. Praticamente, tutti i tool di rete per sistemi

[1] API acronimo di Application to Program Interface, è un software che permette la comunicazione tra programmi incompatibili.

Unix si appoggiano sulla libreria libpcap per accedere ai pacchetti a basso livello. Un esempio di applicativi che sfruttano libpcap come tcpdump ed Ethereal. Di seguito riportiamo una rappresentazione della struttura del BPF su tre livelli: rete, livello kernel e livello utente.

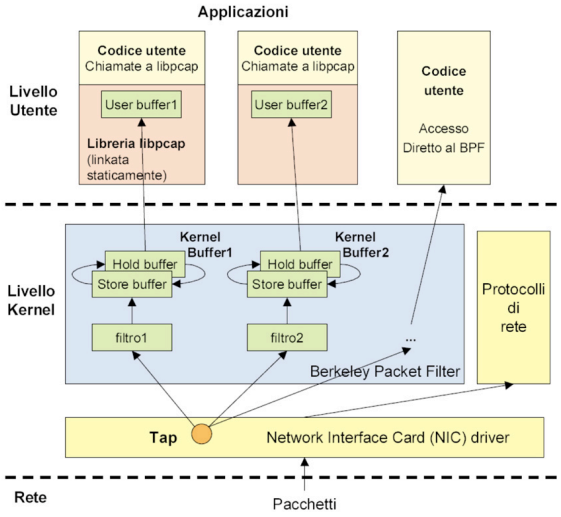

Figura 1. Architettura BPF

Il BPF è implementato come un device driver che si appoggia al driver della scheda di rete (detta Network Interface Card o NIC). Il NIC driver, appena trasferito un pacchetto in memoria e prima di passarlo agli stack protocollari, invoca la funzione *tab()* del BPF che si occupa di intercettare e copiare il pacchetto.

La funzione *tab()* per prima cosa effettua un'operazione di filtraggio per valutare se il pacchetto è di interesse per l'applicazione oppure no. Se il pacchetto non è di interesse per l'applicazione viene scartato, altrimenti la *tab()* lo inserisce in un buffer allocato dal driver, detto buffer kernel. Dalla figura precedente si nota che il BPF usa un sistema a doppio buffer (per un totale di 64Kbytes) in cui una porzione di memoria contiene i pacchetti in arrivo dalla rete mentre l'altra conserva quelli che verranno passati all'applicazione. I due buffer scambiano le loro funzioni quando il primo è pieno e il secondo è vuoto. L'applicazione preleva i pacchetti dal buffer mediante la chiamata alla funzione *read()* relativa al BPF. Questa lettura attiva una funzione del BPF che sposta i pacchetti in una zona di memoria utente allocata e gestita da libpcap.

1.2 Architettura WinPcap

Come detto precedentemente WinPcap è stato sviluppato usando come base il BPF. Di seguito riportiamo la struttura della libreria WinPcap (figura 2) e nei prossimi paragrafi analizzeremo in dettaglio i blocchi che la compongono.

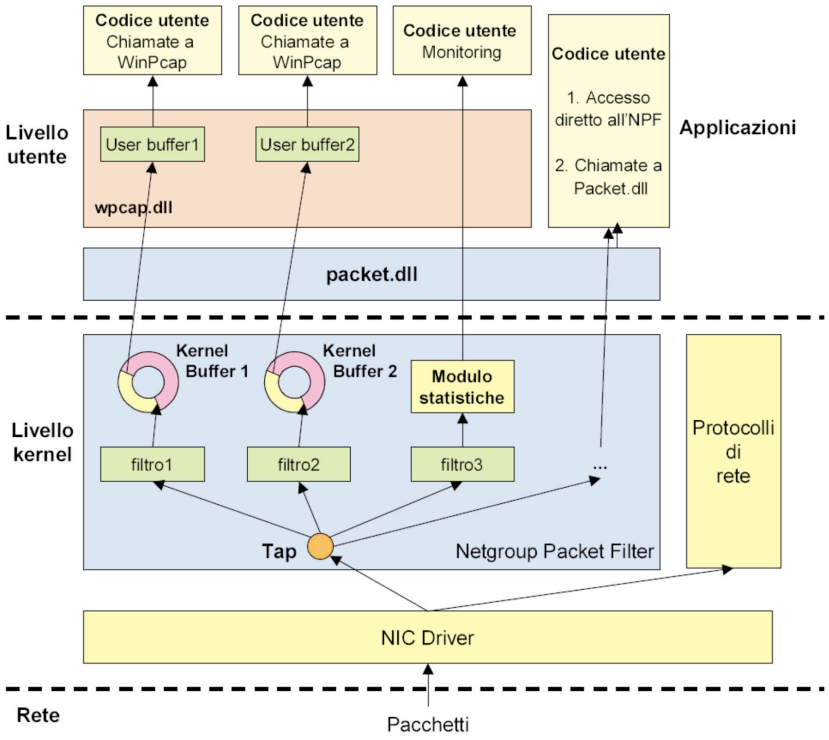

Figura 2. Architettura di WinPcap

I moduli principali che caratterizzano WinPcap sono tre:

- il primo modulo è il *NetGroup Packet Filter* (NPF) che si occupa delle operazioni di basso livello fra le quali la più importante è l'interfacciamento con la scheda di rete per la cattura dei pacchetti. Le funzioni principali svolte dal NPF sono:
  - Interfacciamento con la scheda di rete tramite il driver che la controlla;
  - Estrazione del traffico che circola sulla rete;
  - Filtraggio dei dati in base alle implementazioni stabilite dall'utente.

Tutte queste operazioni non sono consentite ad una normale scheda di rete; per questo motivo è necessario appoggiarsi ad un device driver, il quale rappresenta un'estensione al kernel[2] del sistema operativo.
Nei sistemi Windows la collocazione dell'NPF all'interno del kernel non è semplice come nel caso di sistemi operativi BSD.
Nei sistemi Windows non sono forniti né i codici sorgente né quelli dei driver di rete, quindi risulta impossibile stabilire un "aggancio" per la cattura dei pacchetti. La soluzione prevede di implementare il NPF come driver di protocollo. Questo perché Microsoft definisce e documenta l'interfaccia Network Driver Interface Specification che viene utilizzata dal kernel di Windows per far comunicare i moduli che lo compongono.

- il secondo modulo è una libreria dinamica, chiamata *packet.dll*[3], il cui compito è quello di creare un substrato identico in tutti i sistemi operativi Windows per garantire la massima portabilità delle applicazioni;
- il terzo modulo è la libreria dinamica *wpcap.dll* che ha la funzione di esportare un set di funzioni di alto livello per la cattura e l'analisi della rete, wpcap è compatibile con la libreria libpcap di Unix.

1.3 Modalità di funzionamento di WinPcap

Le prime applicazioni per il monitoraggio di reti prevedevano una sola modalità di cattura dei pacchetti. Con l'evoluzione delle tecnologie, ma soprattutto con l'evoluzione delle reti, le esigenze per il monitoraggio di reti sono cambiate e per questo motivo si sono sviluppate nuove tecniche di monitoraggio di reti che consentono di sfruttare al meglio le risorse disponibili per ottenere una buona analisi del traffico di rete. WinPcap, essendo un software progettato di recente, ha introdotto due nuove modalità di funzionamento che sono: il promiscuous mode e lo statistics mode.

[2] Kernel – Il kernel costituisce il nucleo di un sistema operativo. Si tratta di un software avente il compito di fornire ai processi in esecuzione sull'elaboratore un accesso sicuro e controllato all'hardware.
[3] DLL – Dynamic Link Library è una libreria software che non viene collegata staticamente ad un eseguibile in fase di compilazione, ma che viene caricata dinamicamente in fase di esecuzione.

1.3.1 Promiscuous mode

Questa modalità di funzionamento è una variante della normale modalità di cattura. Infatti, questa modalità prevede che il dispositivo si metta in ascolto del canale e venga forzato a catturare tutti i pacchetti che transitano sul canale, compresi quelli che non sono destinati a lui. Con questo tipo di cattura è possibile coprire una regione maggiore della rete e quindi ottenere maggiori informazioni su ciò che accade nella rete.

1.3.2 Statistics mode

Nell'ambito della gestione delle reti, in alcuni casi, è necessario avere informazioni generali sul carico della rete nei diversi periodi della giornata. In questi casi, per ottenere queste informazioni non è necessario conoscere il contenuto di tutti i pacchetti in transito ma basta avere una valutazione statistica della rete.
Lo statistics mode è una particolare modalità di funzionamento prevista dalla NPF che fornisce un supporto semplice ed efficiente per quelle applicazioni che richiedono solo informazioni statistiche sulla rete. In questa modalità il NPF non cattura tutti i pacchetti ma si limita a contare i pacchetti e byte in transito, in base alle specifiche impostate dall'utente tramite la definizione del filtro, per poi passarle all'applicazione con frequenza regolare e definibile.
Questa modalità di funzionamento è molto interessante, utile e versatile in quanto garantisce un ampio margine di personalizzazione grazie all'uso del NPF.
Lo statistics mode è caratterizzato da due aspetti che lo rendono più efficiente degli altri metodi di cattura utilizzati per questo scopo. In primo luogo si evita qualsiasi copia dei pacchetti in quanto i pacchetti sono filtrati e contati, ma solo il risultato del conteggio viene passato a livello utente, ottenendo così un notevole risparmio di risorse della CPU. Il secondo aspetto è legato al fatto che poiché non viene memorizzato alcun pacchetto i buffer non vengono utilizzati e si può evitare di allocare memoria per il processo di cattura.

L’introduzione dello statistics mode ha consentito di semplificare le operazioni necessarie ad avere informazioni statistiche con un notevole incremento delle prestazioni.

1.4 Soluzioni avanzate per l’analisi di rete

Nonostante le ottimizzazioni indicate precedentemente la cattura dei pacchetti rimane sempre un processo critico per le prestazioni del sistema, soprattutto in reti ad alta velocità. Per supportare al meglio le applicazioni, il NPF fornisce una serie di funzionalità avanzate che permettono un incremento nelle prestazioni nel caso di operazioni ricorrenti nell’analisi di reti. L’idea alla base di questi miglioramenti è quella di spostare parte dell’elaborazione dei dati nel driver. Così facendo il codice che elabora i dati è posizionato nel kernel del sistema operativo e l’applicazione riceve il risultato dell’elaborazione, permettendo di minimizzare copie e context switch con un grande vantaggio in termini di prestazioni ed efficienza.

1.4.1 Monitoraggio del traffico

Generalmente le applicazioni che eseguono il monitoring della rete non necessitano di tutto il pacchetto ma è sufficiente un riepilogo delle caratteristiche del traffico in transito.
Come detto precedentemente il NPF è in grado di calcolare statistiche sui dati ricevuti in modo autonomo e programmabile e di restituire all’applicazione i risultati ottenuti. I vantaggi ottenuti da questa soluzione sono(vedi figura 3):

- Nessun buffer viene allocato, né a livello kernel né a livello utente, di conseguenza nessuna copia dei pacchetti viene effettuata;
- le statistiche vengono consegnate ad intervalli regolari definibili tramite un’apposita chiamata e minimizzando il numero di context switch.

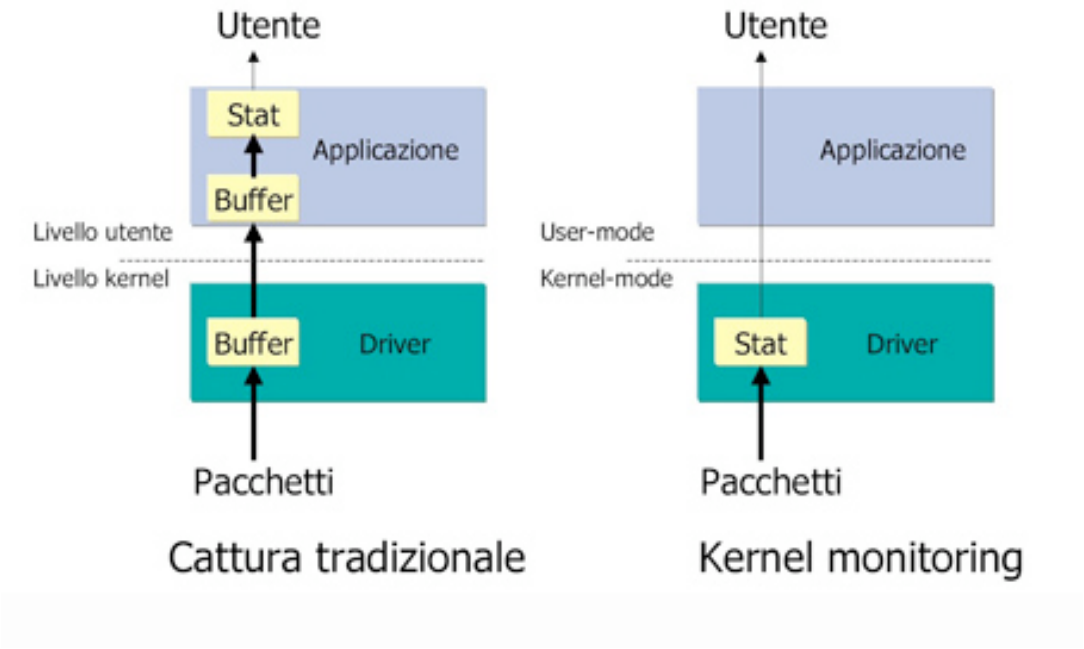

Figura 3. Confronto tra cattura e kernel monitoring

In figura 4 si evidenzia come il sistema di monitoring del NPF usa il filtro per selezionare i pacchetti di interesse tramite opportune regole. Per esempio è possibile impostare il filtro in modo tale che accetti solo i pacchetti web oppure il traffico generato da un solo host. I pacchetti che vengono considerati di interesse per l'applicazione dal filtro vengono passati come ingresso ad un modulo che ne esegue il conteggio tramite contatori oppure una tabella hash. I risultati del conteggio vengono trasmessi all'applicazione allo scadere del timeout.

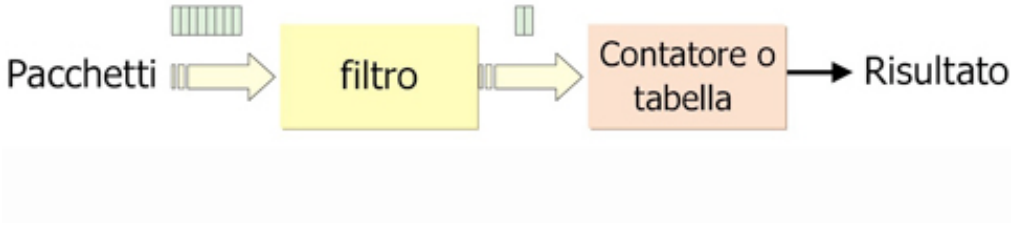

Figura 4. Funzionamento del sistema di monitoring

1.4.2 Salvataggio del traffico su disco

Spesso i programmi basati su WinPcap salvano immediatamente il traffico di rete su disco (*disc dump*) rimandando ad un secondo momento l'elaborazione delle informazioni raccolte. Questo è un comportamento tipico di tutti gli applicativi di analisi di rete che non sono in grado di reggere un'analisi in tempo reale del traffico effettuando così elaborazioni off-line dei pacchetti.

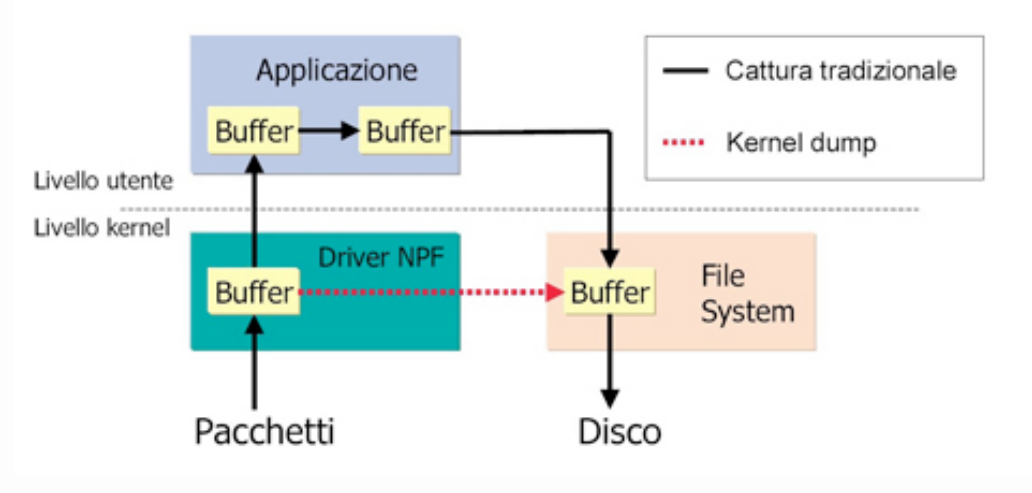

Figura 5. Confronto sul dumb del traffico su disco tra sistema tradizionale e ottimizzato.

In figura 5 mostriamo un confronto tra il disc dump ed il salvataggio dei pacchetti su disco in sistemi tradizionali. Nel sistema tradizionale (linea continua) i pacchetti vengono salvati su disco dopo la cattura. In questa fase entrano in gioco i quattro buffer, di conseguenza vengono eseguite quattro copie di un pacchetto richiedendo diverse transizioni da livello kernel a livello utente e provocando così un dispendio inutile di risorse e di tempo. Con la soluzione di salvataggio direttamente su disco (linea tratteggiata) si nota subito una gestione migliore del processo, risparmiando memoria e diminuendo il carico sulla CPU. Il *disc dump* prevede che il driver NPF sia in grado di comunicare direttamente con il file system trasmettendogli i pacchetti non appena verranno ricevuti dalla scheda di rete.

Capitolo 2 - Ethereal

Ethereal è un software open source per l'analisi del traffico di rete. Questo programma cattura i pacchetti in transito nella rete e mostra il loro contenuto a schermo. I software per l'analisi del traffico di rete permettono di effettuare delle misurazioni su ciò che accade nel network locale allo stesso modo in cui un voltmetro viene usato per misurare la differenza di potenziale tra due nodi di un circuito.

I principali usi di Ethereal sono:

- scoprire ed eliminare guasti nella rete;
- effettuare il debug dell'implementazione di un protocollo di rete;
- studiare la struttura di un protocollo di rete;
- esaminare problemi di sicurezza.

Ethereal è un software multipiattaforma in quanto può essere installato su sistemi operativi Unix (linux, Macintosh ecc) e Windows. In tutte le piattaforme mette a disposizione le seguenti funzioni:

- cattura in tempo reale dei pacchetti;
- visualizzazione dettagliata dei pacchetti con informazioni sul protocollo di trasmissione;
- esportazione ed importazione di pacchetti in formati compatibili per altri software di cattura e analisi;
- impostazione di filtri per la selezione dei pacchetti o l'impostazione di regole di cattura.

Un esempio di una schermata tipica è riportata in figura 6.

Da osservare che Ethereal non è stato sviluppato per modificare il contenuto dei pacchetti o per alterare il flusso delle informazioni, ma è stato realizzato per dare un supporto agli amministratori di reti.

2.1 Breve storia di Ethereal

Lo sviluppo di Ehtereal è iniziato nel 1997 quando Gerald Conbs, osservando l'evoluzione delle reti, ha sentito la necessità di sviluppare un tool per l'identificazione di errori all'interno di reti di calcolatori che, allo stesso tempo, fornisse gli strumenti per ottenere informazioni sul funzionamento dei protocolli di rete.

Solamente nel 1998, dopo una serie di problemi di sviluppo, è stata rilasciata la prima versione di Ethereal, con ancora molti bug da individuare e risolvere. Inizialmente il software non ha riscosso il successo che il suo creatore si aspettava, ma l'uscita delle prime patches e la segnalazione di errori vari da parte dei primi utilizzatori hanno permesso a Gerald Conbs di modificare Ethereal e renderlo un software importantissimo per lo sviluppo di reti di grandi dimensioni.

Proprio il successo che stava riscuotendo questo programma ha spinto molti sviluppatori a partecipare allo sviluppo del software, tra questi Guy Harris che ha consentito di migliorare Ethereal proprio nella comunicazione a basso livello tra il software ed il kernel del sistema operativo.

Da quel momento in poi Ethereal è diventato uno dei software più usati e conosciuti nel settore dell'analisi di rete e, con il suo successo, è aumentato anche il numero di sviluppatori che danno il loro contributo ancora oggi per mantenere alta l'efficienza di questo programma.

2.2 Uso di Ethereal

Dopo aver installato il programma con le relative librerie quali libpcap e winpcap, che precedentemente abbiamo spiegato, è possibile lanciare il programma. In Figura 6 è possibile vedere la finestra principale del programma dopo l'acquisizione di alcuni pacchetti.

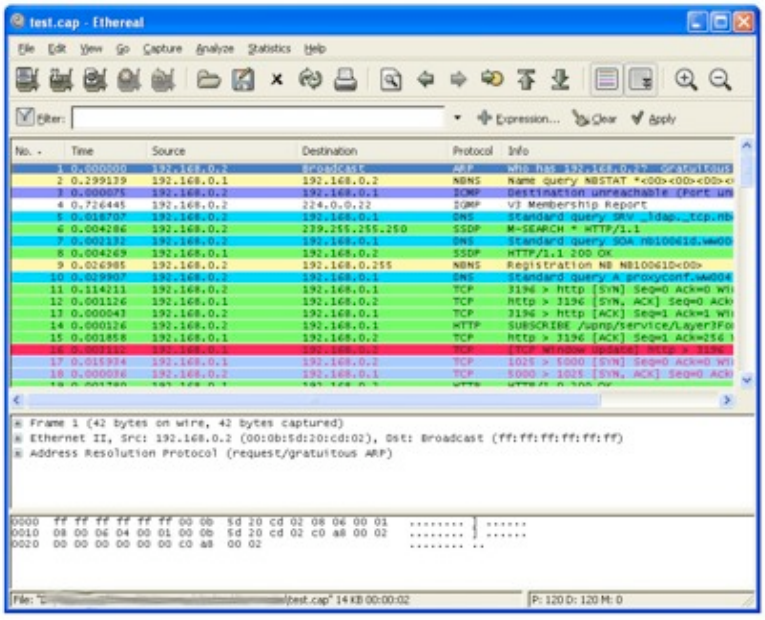

Figura 6. Visualizzazione di Ethereal dopo l'acquisizione di alcuni pacchetti.

Dall'immagine precedente è possibile vedere le varie funzioni a cui è possibile accedere.

- *Menu*: usato per accedere alle varie voci di configurazione del programma.
- *Main toolbar*: usata per consentire l'accesso rapido alle voci più frequenti del menu.
- *Filter toolbar*: barra che consente l'accesso diretto agli strumenti per la manipolazione dei pacchetti.
- *Packet List Pane*: mostra le informazioni generali sui pacchetti catturati. Cliccando un pacchetto in questo pannello si controlla anche il contenuto di altri due pannelli.
- *Packet detail pane*: mostra il pacchetto selezionato nel packet list pane.
- *Packet bytes pane*: mostra i dati contenuti nel pacchetto selezionato nel list pane ed evidenzia i campi selezionati nel packet details pane.

Di seguito riporteremo una breve descrizione degli elementi principali appena descritti.

2.2.1 Il menu

Il menu di Ethereal è posizionato nella parte alta della schermata e le voci principali che lo compongono sono:

- *File*: questo menu contiene le voci per aprire, salvare ed esportare i file catturati sia in modo integrale che parziale.
- *Edit*: contiene le operazioni per trovare un pacchetto, selezionare preferenze temporali o per fissare un'origine temporale diversa da quella effettuata dal programma.
- *View*: controlla la visualizzazione dei dati catturati, inclusa la colorazione dei pacchetti, l'ingrandimento dei caratteri ecc.
- *Go*: contiene le operazioni per spostarsi da un pacchetto ad un altro.
- *Captured*: permette di decidere quando iniziare e terminare la cattura dei pacchetti, in più permette di definire le regole di cattura (o filtraggio) dei pacchetti.
- *Analyze*: contiene le voci per manipolare le impostazioni dei filtri, abilitare o disabilitare sezioni dei protocolli, configurare regole per la decodifica di flussi TCP stream.

- *Statistics*: contiene una serie di voci che forniscono un'analisi statistica sui pacchetti catturati, mostra la gerarchia dei protocolli e molto altro.

Nei prossimi paragrafi riporteremo una descrizione accurata delle voci principali di alcuni menu quali Captured, Analyse, Statistics.

2.2.2 Menu, Edit, View, Go

Queste voci principali del menu sono presenti in tutti i programmi open source, quindi una descrizione completa sarebbe inutile in quanto sono operazioni di uso comune. Riporteremo di seguito solo le voci presenti in Ethereal e che hanno funzioni importanti.

- Export > As "Plain Text"

Questo menu permette di esportare tutto, o una parte, dei pacchetti catturati in un file ASCII di testo.

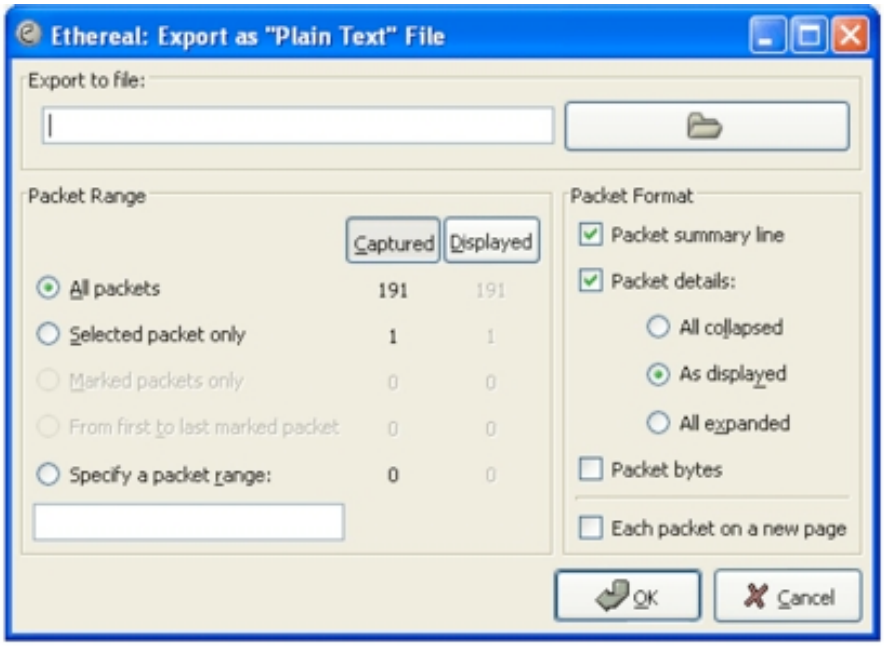

Figura 7. Finestra di dialogo Export > As "Plain Text".

- *Export to file*: permette di scegliere dove esportare il file.
- *Packet Range*: è presente in molte finestre di dialogo e permette di decidere quali pacchetti devono essere inseriti nel processo di output.

L'opzione di default è quella di esportare tutti i pacchetti, mentre se non si vogliono esportare tutti i pacchetti è possibile scegliere una delle seguenti voci:

- selected packet only: esporta solamente i pacchetti selezionati

- marked packet only: esporta solamente i pacchetti segnati con determinate opzioni;
- from first to last marked packet: esporta solo i pacchetti contrassegnati compresi tra i due estremi;
- specify range; permette di esportare range di pacchetti. Per esempio esporta i pacchetti compresi tra il pacchetto 5 e il pacchetto 10 e dal 20 al 60.

- *Packet Format Frame*: questa sezione compare in molte finestre di dialogo e permette di impostare quali parti del pacchetto devono essere esportate. Le voci di configurazione sono:

- packet summary line: permette di esportare solo le informazioni di riepilogo come nella packet list;
- packet details: se attivato permette di esportare i dettagli dei pacchetti;
- packet byte: permette di esportare le infomazioni in byte come nel packet byte pane;
- each packet on new page: dispone le varie informazioni di uscita in pagine separate.

- Export > As “PostScript” file

Questa voce del menu permette di esportare i pacchetti catturati in un PostScript file. Il PostScript file è il formato più utilizzato per la stampa di pacchetti. La finestra di dialogo presenta le stesse impostazioni di quelle di figura 6.

- Export > As > “CSV” (comma separated values packet summary) file

Permette di esportare il sommario (informazioni principali) dei pacchetti catturati in un file .cvs. Le impostazioni sono le medesime di figura 6.

- Export > As PSML“ file

Permette di esportare il sommario dei pacchetti catturati in un file XMP. La finestra di dialogo che compare contiene solo la possibilità di scegliere quali pacchetti esportare.

Nel menu edit la funzione più importante è :

- Time Referente > set Time Referente (toggle)

Questa voce permette di impostare il pacchetto selezionato come istante temporale di riferimento. Una volta impostato il riferimento tutta la sequenza di valori temporali dei pacchetti successivi sarà aggiornata prendendo come origine relativa dei tempi il pacchetto selezionato.

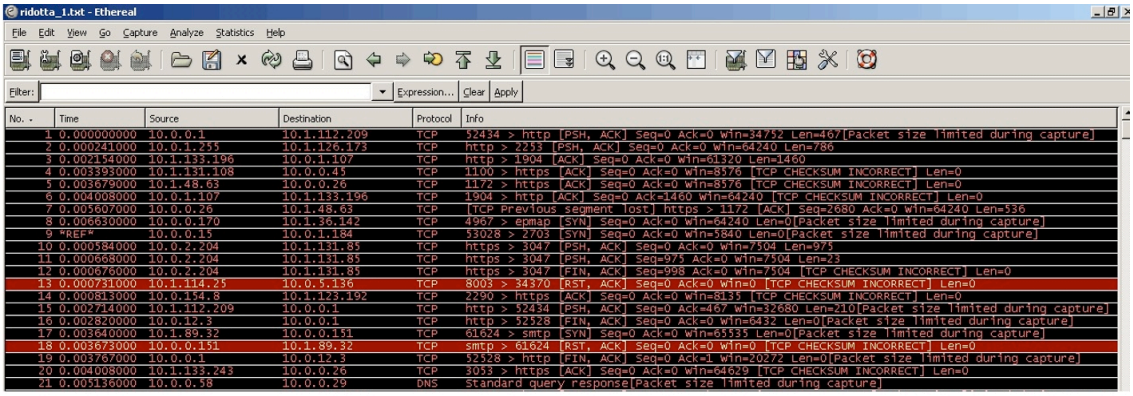

Figura 8. Esempio del funzionamento di time reference.

Nel menu View è possibile impostare la visualizzazione delle finestre di dialogo e i campi del Packet List Pane, Packet Detail Pane ecc.
In questo campo la funzione principale è :

- Time Display Format

Questa sezione è importante in quanto tutti i pacchetti catturati vengono etichettati con il loro istante di cattura. I possibili formati con cui presentare questo campo sono:

- data and time of day: 1970-01-01 questa visualizzazione mostra l'anno, il giorno e il mese in cui il pacchetto e stato catturato considerando come riferimento il primo gennaio 1970;
- time of day: mostra l'ora e il giorno in cui i pacchetti sono stati catturati;
- secondo since beginning of captured: tempo in secondi relativo al primo pacchetto catturato.

È possibile nel medesimo menu inserire dei riferimenti temporali agendo su una delle seguenti opzioni:

- Automatic: il software decide in base alla precisione del timestamp catturato, la visualizzazione da utilizzare per i riferimenti temporali dei vari pacchetti;

- Second, decisecond, millisecond: la precisione del timestamp è forzata al formato deciso dall'utente.

Il menu Go serve solo per spostarsi da un pacchetto ad un altro in modo rapido, non ci sono funzioni particolari da segnalare.

2.2.3 Captured

Il menu Captured è forse il menu più importante in quanto contiene le voci per impostare i parametri di cattura e i comandi per avviare le operazioni. Di seguito riporteremo le voci principali che caratterizzano questo menu.

- Interfaces

Questa voce del menu mediante una finestra a comparsa (pop-up) mostra cosa sta facendo la scheda di rete in uso su ethereal. La finestra che si apre alla selezione di questo comando è mostrata in figura 9.

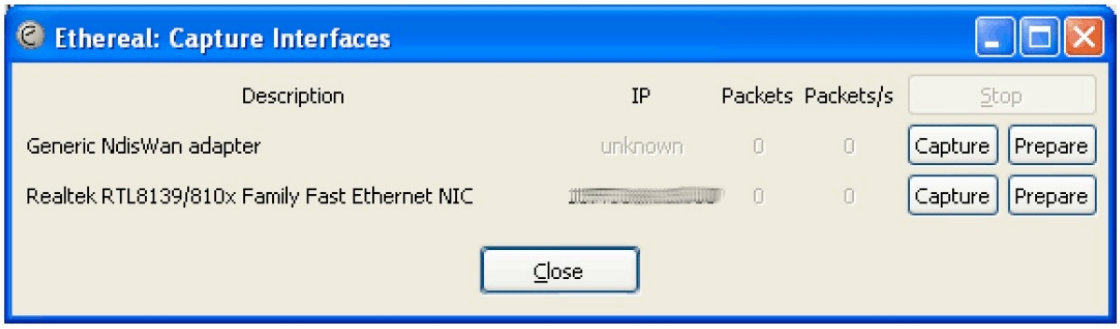

Figura 9. Captured Interfaces.

Le voci presenti in questa finestra sono:

- Description: descrive la scheda di rete in uso mostrando le informazioni fornite dal sistema operativo
- IP: mostra l'indirizzo ip assegnato alla scheda di rete in questione, se alla scheda di rete non è assegnato nessun valore, viene impostato al valore unknow.
- Packets: indica il numero di pacchetti catturati dall'interfaccia dal momento in cui la finestra è stata aperta.
- Packets /s : mostra il numero di pacchetti catturati nell'ultimo secondo.

- Captured: fa iniziare immediatamente un'acquisizone del traffico di rete.
- Stop: Ferma l'aquisizione del traffico.
- Prepare: Apre la finestra di configurazione per l'acquisizione dei pacchetti che è illustrata in figura 10.

- Options

Questo menu apre una finestra di configurazione per impostare i parametri per l'aquisizione dei pacchetti nella rete.

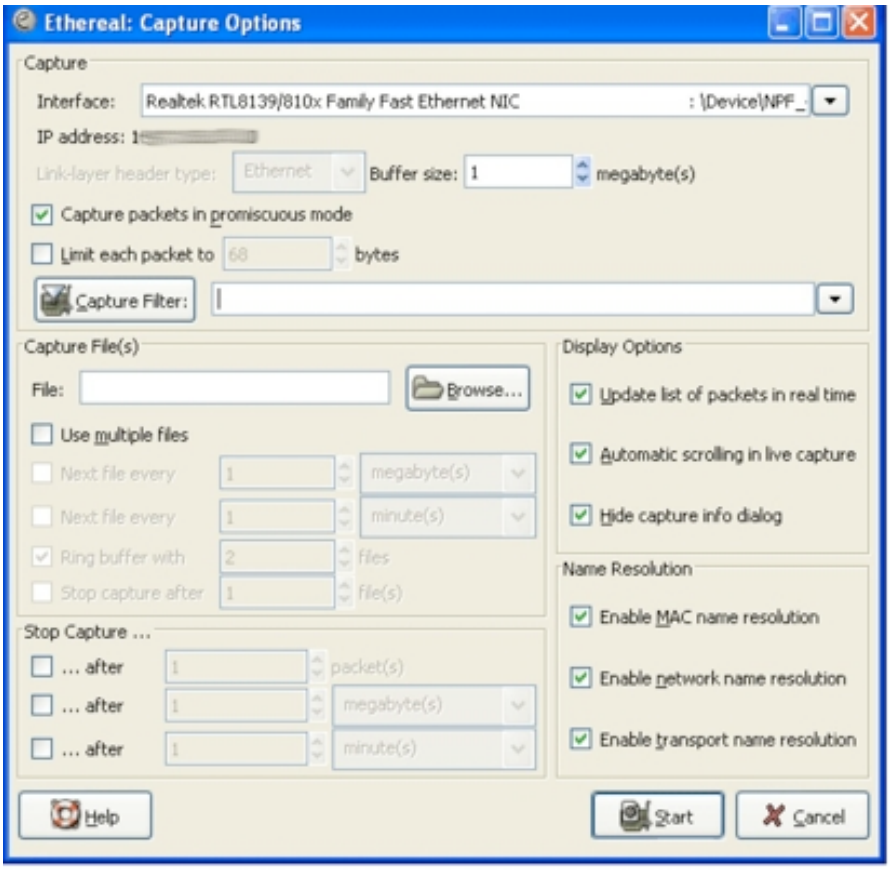

Figura 10. Pannello di configurazione.

Da questa finestra è possibile impostare i seguenti parametri:

- *Interface*: permette di scegliere quale interfaccia di rete usare per la cattura dei pacchetti. Ethereal prevede l'acquisizione dei pacchetti da una sola interfaccia di rete alla volta, quindi, non sono possibili acquisizioni simultane.
- *IP address*: mostra l'indirizzo ip della scheda di rete selezionata.
- *Link layer header type*: non è possibile selezionare questo campo. È possibile utilizzarlo nel caso in cui stiamo catturando il traffico

con un dispositivo 802.11 dove è possibile scegliere tra Ethernet e 802.11. Un altro caso che permettere di utilizzare Link layer header type nel caso in cui stiamo catturando traffico con una scheda Endance DAG in una linea seriale sincrona dove si può scegliere tra “PPP over seriali” o “Cisco HDLC”.

- *Buffer size*: è possibile inserire la taglia del buffer da utilizzare durante la cattura. L’impostazione di default è quella presente nel kernel buffer con la quale vengono catturati i pacchetti prima di essere scritti su disco. È consigliato di aumentare questa taglia solamente se durante l’acquisizione si perdono pacchetti.
- *Captured packets in promiscuous mode*: questo checkbox permette di decidere se Ethereal deve acquisire i pacchetti in promiscuous mode durante la cattura. Se non viene attivata questa opzione cattura solo i pacchetti destinati alla proprio scheda di rete.
- *Limit each packet to n bytes*: questo campo permette di specificare la dimensione massima dei dati che devono essere catturati solitamente si riferisce allo snaplen. Se non vengono apportate modifiche al parametro il valore di default è 65536 che è sufficiente per molti protocolli. Le regole per modificare questo campo sono:
  1) se si è insicuri sulla dimensione è consigliato di lasciare quella di default;
  2) se si è interessati ad informazioni tipo interfaccia, indirizzo IP, è conveniente impostare le dimensioni piccole del buffer in modo da ridurre il tempo di elaborazione della CPU;
  3) se non si vuole catturare il pacchetto è utile capire la quantità di informazioni da catturare ed adeguare la dimensione del buffer in funzione di questa dimensione.
- *Captured filter*: questo campo permette di specificare un filtro di cattura. Per esempio se noi vogliamo catturare tutto il traffico telnet proveniente e destinato ad un host specifico dobbiamo inserire una regola come :

  tcp port 23 host 10.0.0.5

questa regola è costituita da due primitive congiunte da una AND.

- Captured file(s) frame: questo campo permette di specificare una serie di opzioni:
  1) use Multiple File: permette di salvare i pacchetti in più file;
  2) next file every n Megabyte: è un'opzione che permette di passare ad un nuovo file quando durante la cattura il fare corrente ha raggiunto una dimensione predefinita;
  3) next file every n minute: permette ripassare ad un file ogni n minuti indipendentemente dal numero di pacchetti catturati;
  4) ring buffer with n file: forma un buffered circolare costituito da n file. Riempie il primo file di pacchetti poi il secondo e così via fino al n-ennesimo file. Riempito l'ultimo file riparte dal primo sovrascrivendo ai pacchetti ottenuti precedentemente.
  5) Stop captured after n file: permette di terminare l'acquisizione del traffico di rete quando si sono salvati n file di traffico.

Va ricordato che per catturare i pacchetti vengono usate le librerie libpcap per sistemi Unix, mentre viene usata la libreria WinPcap per sistemi Windows.

- Start, Stop e Restart

Servono per avviare e terminare la procedura di acquisizione del traffico di rete. Con restart invece è possibile riprendere l'acquisizione dei pacchetti senza eliminare quelli precedentemente salvati.

- Captured Filter

Questo menu permette di creare e modificare un filtro di cattura. È possibile attribuire un nome ai filtri per salvarli successivamente.
Per definire o modificare un filtro apparirà una finestra di dialogo come quella in figura 11.

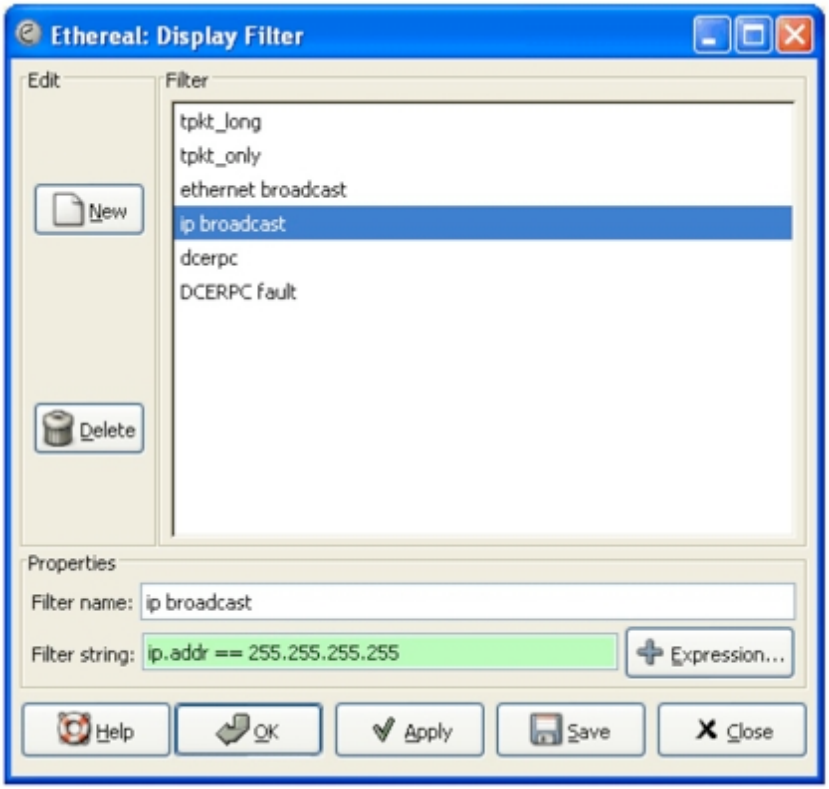

Figura 6. Finestra per la configurazione del filtro

I pulsanti new, filter, name e delete permettono di creare eliminare e dare un nuovo nome ad un filtro.

- Filter name permette di aprire un filtro tra quelli presenti nella lista.
- Add Expression; permette di aggiungere una nuova regola al filtro per la cattura dei pacchetti.

2.2.4 Analyse

Il menu Analyse contiene le voci mostrate in figura

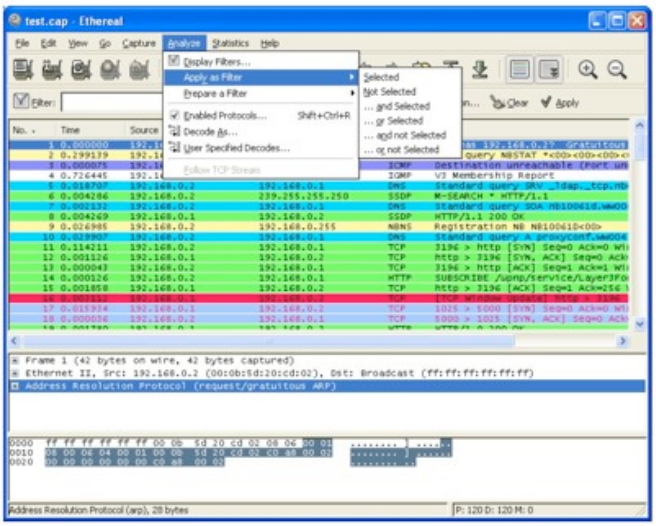

Figura 7. Presentazione del menu Analyse.

- Display filter:

permette di visualizzare una finestra di dialogo come quella di figura 12.

- Apply as filter:

permette di visualizzare lo stato attuale del filtro ed eventualmente di applicare delle modifiche immediate alla configurazione del filtro.

- Enable Protocols:

permette di abilitare/disabilitare certi tipi di protocollo. Tutti i protocolli sono abilitati di default. Quando un protocollo è disabilitato, Ethereal ferma l'acquisizione del pacchetto che contiene le indicazioni di quel protocollo. La finestra che apparirà per abilitare o disabilitare i protocolli è riportata in figura 13.

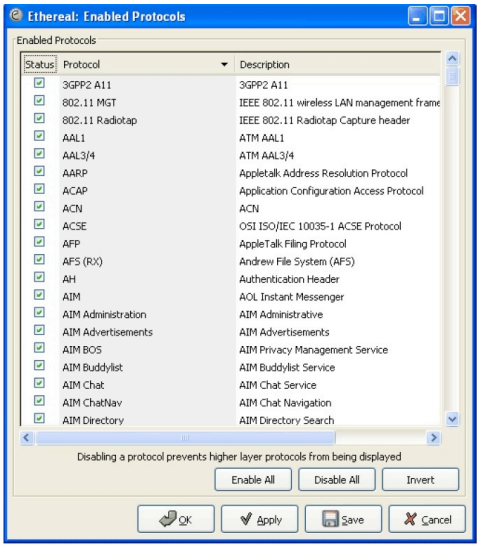

Figura 8. Finestra per la scelta dei protocolli da abilitare per la cattura.

- Decode As:

Permette agli utenti di forzare Ethereal a decodificare certi pacchetti o particolari protocolli.

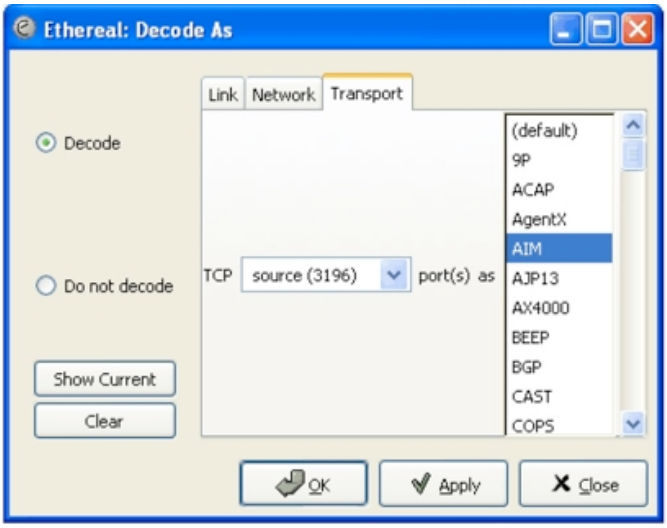

Figura 9. Finestra per regole di decodifica.

Chiuso il programma le regole di decodifica vengono perse.

Le voci di configurazione principali di questo menu sono:

- decode: permette la decodifica dei pacchetti indicati
- do not decode: non permette di decodificare i pacchetti specificati
- Link/Network/ Transport: specifica il livello di rete a cui deve essere applicata la decodifica.
- Show Current: apre una finestra di dialogo che mostra quali decodifiche sono avviate.

- User specified decodes:

Permette agli utenti di forzare Ethereal a decodificare pacchetti o certi protocolli. A differenza dell'opzione precedente le regole di decodifica rimarranno anche dopo la chiusura del programma.

- Follow TCP Stream:

questo menu apre un finestra di dialogo e mostra tutti i segmenti TCP collegati al pacchetto TCP catturato.

2.2.5 Statistics

Il menu Statistics contiene le voci riportate in figura 15.

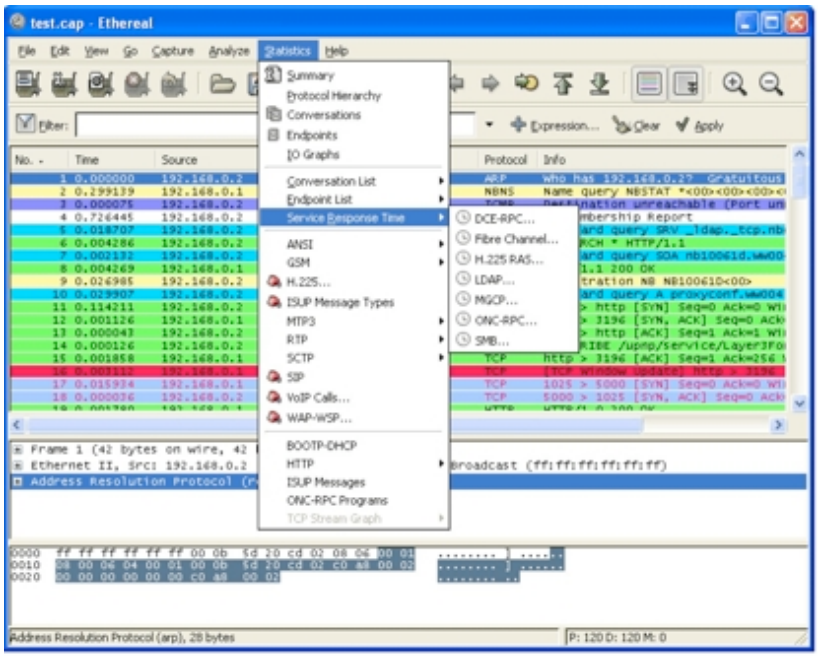

Figura 10. Presentazione del menu Statistics .

- Summary

Mostra le informazioni sui dati catturati. Un esempio della finestra di dialogo che appare è riportata in figura 16.

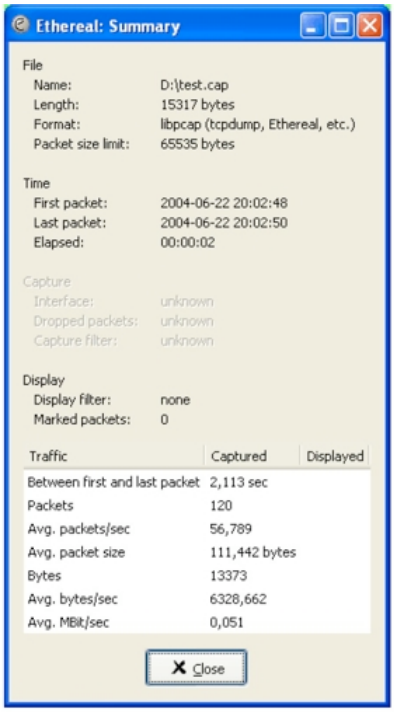

Figura 11. Finestra summary.

- *file*: illustra le informazioni principali sul file catturato;

- *time*: mostra il momento di acquisizione del primo e dell'ultimo pacchetto;
- *captured*: illustra informazioni sul tempo in cui si è conclusa l'acquisizione;
- *traffic*: mostra alcune statistiche sul traffico catturato. Se è configurato un filtro è possibile visualizzare anche le primitive di configurazione.

- Protocollo Hierarchy

Mostra l'albero delle gerarchie del protocollo.

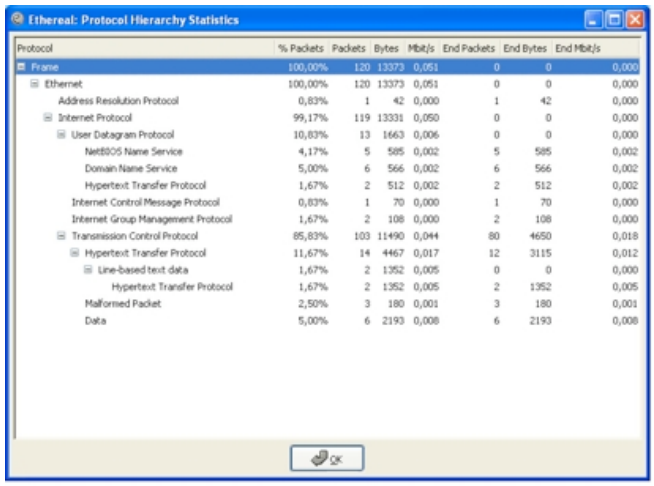

Figura 12. Mostra la struttura gerarchica dei protocolli catturati.

Come si può vedere dalla figura 17, viene illustrato l'albero gerarchico di tutti i protocolli catturati.

Il significato delle colonne che compongono la finestra di dialogo sono:

- *protocol*: contiene il nome del protocollo;
- *%packets*: mostra la percentuale di pacchetti catturati appartenenti a quel protocollo rispetto a tutti i dati catturati;
- *packets*: indica il numero di questo protocollo;
- *bytes*: indica il numero di byte che compongono il protocollo;
- *MBit/s*: mostra a disposizione del protocollo al momento della cattura.

- Conversations

Mostra la lista di traffico tra due nodi della rete.

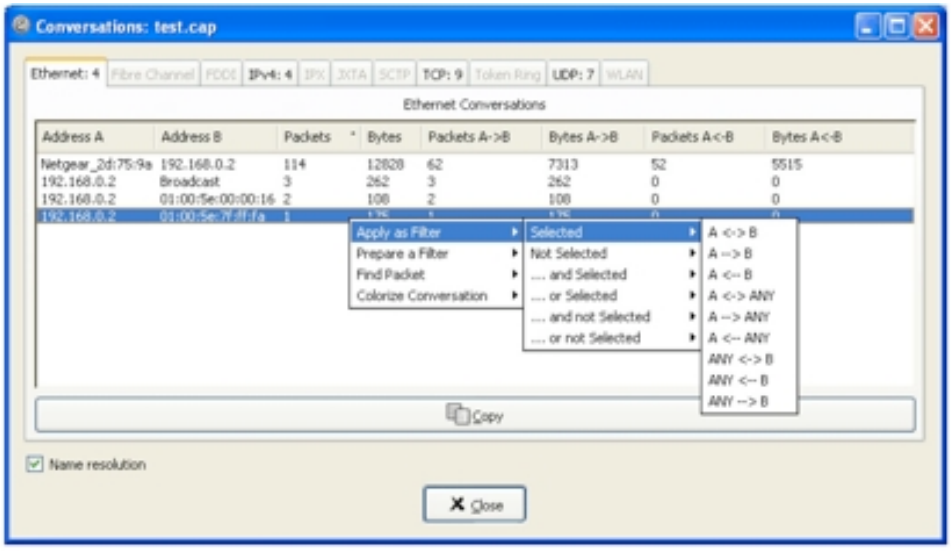

Figura 13. Conversations

Il termine conversation indica il traffico tra due specifici nodi. Per esempio una conversazione IP è tutto il traffico scambiato tra due indirizzi IP.

- Endpoints

Mostra la lista di traffico da e per un indirizzo di rete.

- IO Graphs

Mostra un grafico in base alle impostazioni del mittente, per esempio il numero di pacchetti in transito in un intervallo temporale.

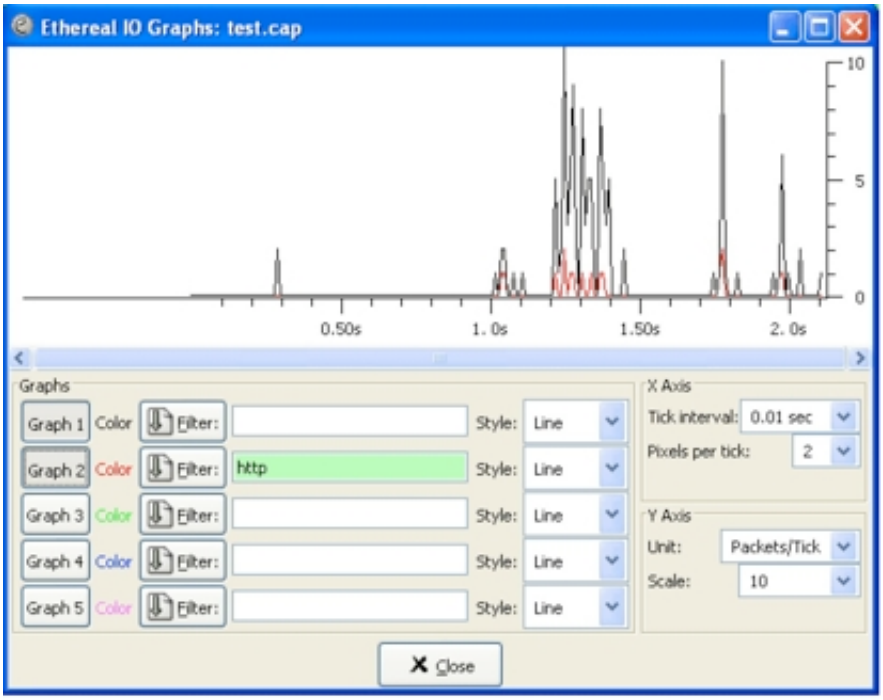

Figura 14. Esempio di grafico.

L'utente può configurare i seguenti parametri:

- graph 1-5: abilità il numero di grafici che è possibile abilitare nella visualizzazione;
- filter: è possibile evidenziare un andamento dei pacchetti che vengono accettati rispetto a quelli scartati;
- X Axis:
  in questa sezione è possibile impostare l'intervallo di misura dell'asse x (10/1/0,1/0,001 secondi).
- Y Axis:
  permette di impostare l'unità di misura (packets/Tick o Byte/tick) e la scala per l'asse y.

• Service Response Time

Mostra il tempo che intercorre tra una richiesta e la rispettiva risposta.
Il servizio response time statistics è attivato per i seguenti protocolli:

- DCE-RPC
- Fibre Channel
- H.225 RAS
- LDAP
- MGCP
- ONC-RPC
- SMB

Le altre voci di questo menu (ANS,GSM,H225 ecc) illustrano le specifiche dei rispettivi protocolli rimandano a delle pagine web.

2.3 Filtraggio durante la cattura

Come detto durante l'illustrazione delle funzioni dei vari menu, Ethereal è in grado di effettuare l'operazione di filtraggio dei dati già durante la cattura dei pacchetti, in modo tale da non salvare su disco informazioni superflue.
Un filtro di cattura è una serie di primitive collegate da una congiunzione logica (AND, OR). Le primitive sono delle espressioni che identificano le regole per definire se un pacchetto è di interesse per l'analisi oppure no. Le primitive possono essere precedute dalla congiunzione not.

[not] primitiva [AND/OR [not] primitiva]

Per esempio se vogliamo salvare tutto il traffico ftp verso un server della rete la regola da impostare è:

tcp port 21 ADN host 192.168.10.1

Le primitive che si possono usare sono:

- [src | dst] host <host>

questa primitiva permette di filtrare il traffico indirizzato da un host identificandolo mediante indirizzo IP o nome.
Si può far precedere la primitiva con una parola chiave src | dst per specificare se siamo interessati ai pacchetti che hanno come sorgente o come destinazione l'indirizzo host specificato. Nel caso in cui venga inserita la parola chiave, viene catturato tutto il traffico che ha come sorgente o come destinazione l'host specificato.

- Ether [src | dst] hosto <host>

Questa primitiva permette di filtrare solamente gli indirizzi ethernet. È possibile inserire la parola chiave src | dst per dire se siamo interessati ai pacchetti come hanno come sorgente o destinazione l'indirizzo specificato.

- Gateway host <host>

Questa primitiva permette di filtrare i pacchetti che usano l'host come gateway.

- [tcp | udp] [src | dst] port <port>

questa primitiva permette di filtrare il traffico di rete in base alla porta di comunicazione del protocollo UDP o TCP. Questa primitiva può essere preceduta dalle opzioni tcp|udp e src | dst che permettono di specificare se si vuole filtrare il traffico tcp o udp che ha come porta di sorgente o destinazione quella specificata nella primitiva. Se non vengono specificate opzioni vengono catturati entrambi.

- Less | greater <length>

Questa primitiva permette di filtrare i pacchetti che hanno come lunghezza minore, maggiore o uguale a quella specificata

- <expr> relop <expr>

questa primitiva permette di creare espressioni complesse per selezionare byte o range di byte.

Quando si lancia l'operazione di cattura apparirà una finestra di dialogo come quella in figura 20.

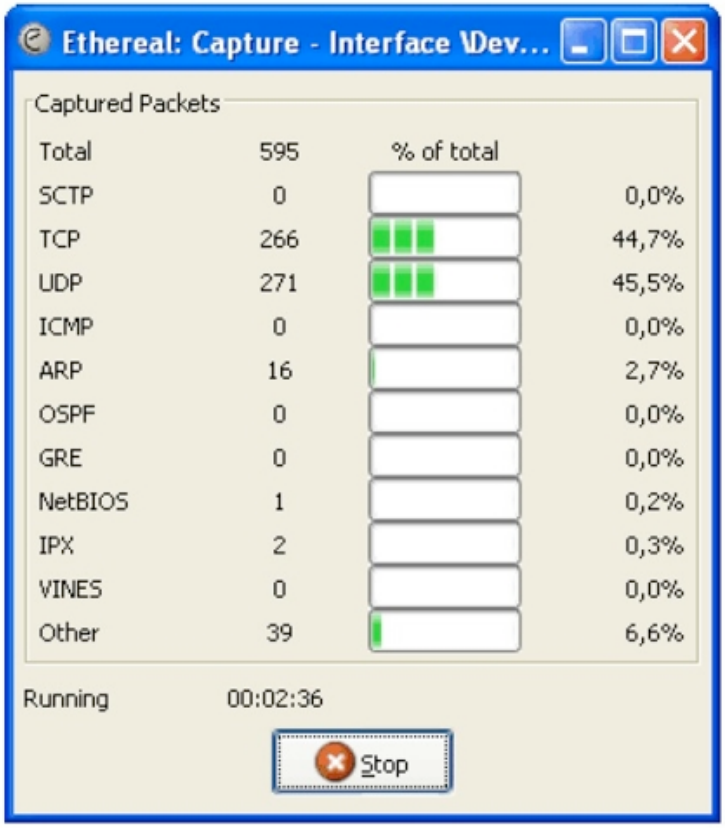

Figura 20. Finestra che indica che Ethereal è in fase di acquisizione del traffico di rete.

Questa finestra informa l'utente del numero di pacchetti catturati e del tipo trascorso dall'inizio della cattura e la percentuale di pacchetti catturati per ogni protocollo.

Terminata la cattura i pacchetti verranno mostrati nella Packet list Pane in ordine di cattura, a questo punto è possibile effettuare tutte le elaborazioni necessarie per la propria analisi usando le varie funzioni spiegate precedentemente.

Va ricordato che il time stamp assegnato ai vari pacchetti viene prelevato dalla libreria WinPcap come spiegato nel capitolo 1.

2.4 Principio di funzionamento di Ethereal

In questo paragrafo riportiamo una breve descrizione del funzionamento di Ethereal. In figura 21 sono riportati i blocchi funzionali principali che caratterizzano il funzionamento di Ethereal.

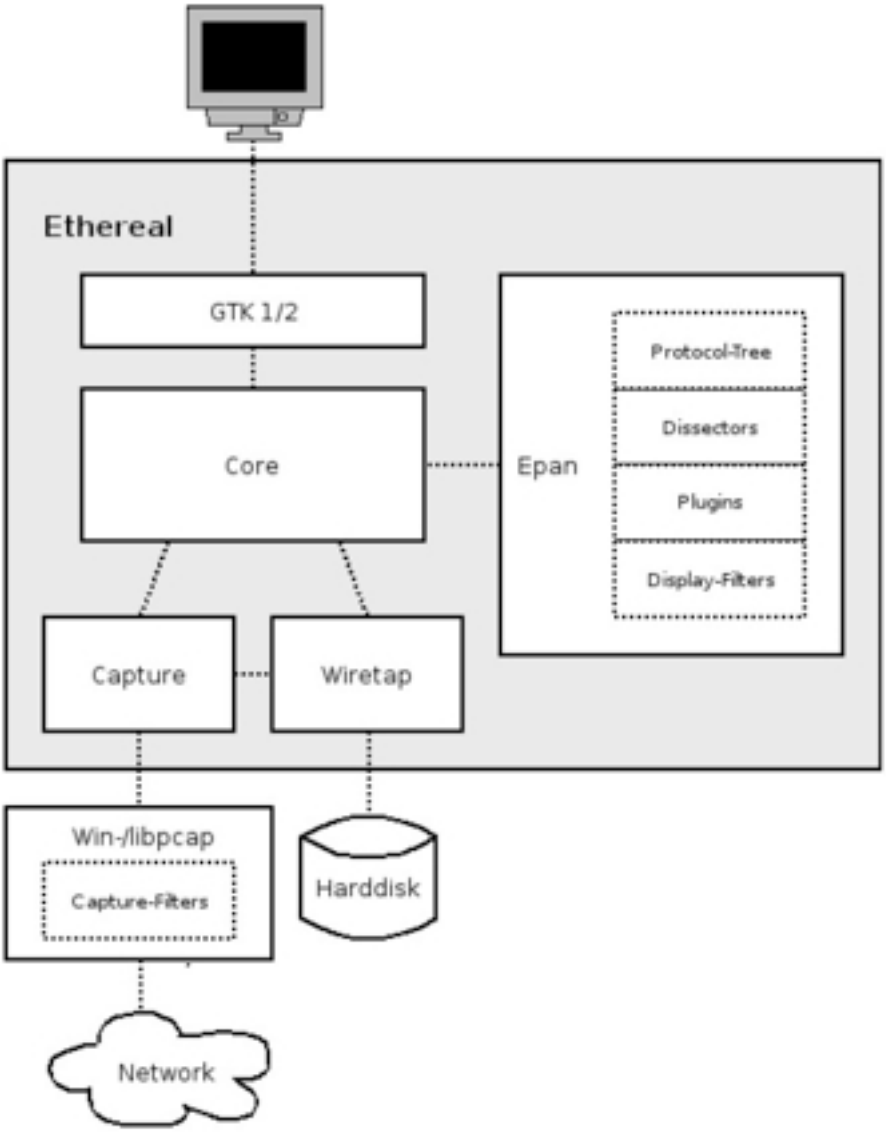

Figura 15. Rappresentazione dello schema funzioanle di Ethereal.

Le funzioni principali svolte dai vari blocchi:

- GTK 1/2: questa libreria permette di gestire tutte le finestre utente di input output.
- Core: è il blocco funzionale che permette di far comunicare e operare un modo sincrono ed efficiente tutti gli altri blocchi.
- Epan: (Ethereal Packet Analysing) è il pacchetto che contiene tutte le funzioni per l'analisi delle informazioni catturate; le funzioni contenute in Epan sono:

1) Protocol-Tree: ricava informazioni sui protocolli partendo dai dati catturati.
2) Dissectors: illustra le varie sezioni dei protocolli.
3) Display filters: raccoglie le primitive dei filtri integrati in Ethereal.

- WinPcap (o libPcap): sono delle librerie non incluse in ethereal, in quanto dipendono dal sistema operativo in cui si installa il software.

  Sono essenziali per la cattura dei pacchetti. Il principio di funzionamento è indicato nel capitlo 1.

Capitolo 3 – Formato di cattura per schede Endace DAG

Le schede di misura del traffico di rete, Endace DAG, producono file nel loro formato nativo, chiamato Extensible Record Format (ERF). Il formato ERF consiste in una serie di record[4] in cui ogni record descrive un pacchetto[5]. I file ERF non prevedono intestazioni ma sono solamente un insieme ordinato di record.

Un generico record può essere schematizzato come in figura 22: il pacchetto contenente i dati è preceduto da un header che contiene le informazioni relative all'istante in cui è stato catturato il pacchetto, il tipo di link, la lunghezza del record e altre informazioni che saranno descritte in seguito.

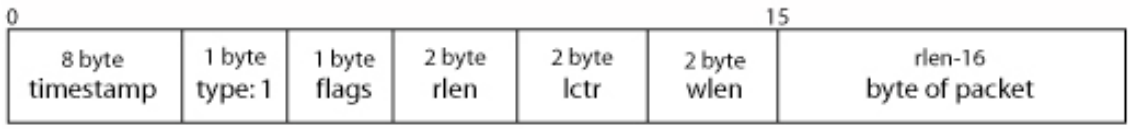

Figura 16. Struttura di un record, nel formato ERF.

Di seguito riportiamo l'analisi dei vari campi:

- **Timestamp:** indica l'istante di tempo in cui il pacchetto è stato salvato**.** Il timestamp è salvato seguendo la convezione little-endian. È un numero a 64 bit a virgola fissa e rappresenta i secondi dalla mezzanotte del primo gennaio 1970. La parte alta del timestamp (32bit più significativi) contiene la parte intera in secondi mentre i 32 bit meno significativi contengono la parte frazionaria dei secondi. Il vantaggio di rappresentare l'istante temporale in cui il pacchetto è stato acquisito in questa notazione è che la differenza tra due timestamp può essere ottenuta con una semplice sottrazione binaria a 64 bit.
- **Type:** contiene una numerazione dei sottotipi di frame.
  0: Type_Legacy.
  1: Type_HDLC_POS : Pos w/HDLC framing.

[4] Record – il record è il pacchetto catturato al quale viene aggiunta un'intestazione che descrive il pacchetto.
[5] Pacchetto – con il termine pacchetto o carico si intende l'informazione catturata dalla scheda di rete.

2: Type_ETH: Ethernet.

3: Type_atm : ATM Cell.

4: Type_AAL5.

- **Flags:** questo campo ha una dimensione di 8 bit. Ogni bit indica un parametro usato nell'acquisizione. Le funzioni dei singoli bit sono le seguenti:

  0-1 : numerazione delle interfacce di acquisizione

  2: lunghezza variabile del record

  3: troncamento del record

  4: errore di collegamento

  5: errore interno

  6-7: riservati
- **Rlen:** Lunghezza totale del record trasferita sul PCI bus di memorizzazione. Il campo rlen indica la lunghezza del pacchetto complessiva di intestazione e dati.
- **LCTR** (loss counter)**:**contatore a 16 bit. Registra il numero di pacchetti persi tra la scheda DAG dovuti al carico eccessivo del PCI bus.
- **Wlen** Lunghezza del pacchetto che include l'overhead dei protocolli. L'interpretazione corretta di questa quantità dipende dal mezzo fisico.

A seconda del tipo di link utilizzato per l'acquisizione dei pacchetti, la struttura dell'header del record cambia. Nel nostro progetto sono stati trattati i seguenti tre tipi:

3.1 Type 1: record PoS HDLC a lunghezza variabile

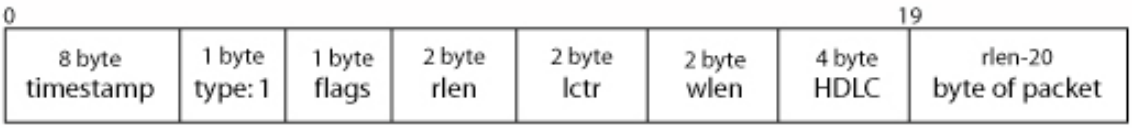

Figura 17. Struttura di un record PoS HDLC.

3.2 Type 2: record Ethernet a lunghezza variabile

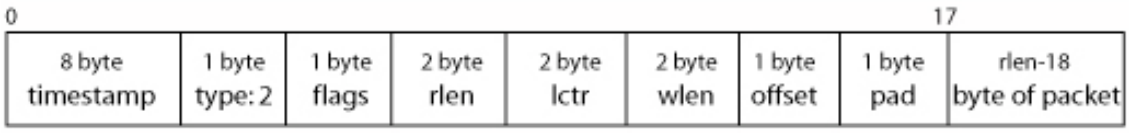

Figura 18. Struttura di un record Ethernet.

nel type2, rappresentato in figura 24, l'intestazione contiene informazioni aggiuntive, quali offset e pad, rispetto al record generico trattato in precedenza.

- **Offset:** numero di byte che non sono stati catturati dall'inizio del frame. Questo campo è tipicamente usato per i tipi di link in cui il salvataggio dei record non richiede il salvataggio in ordine. Attualmente questo campo non è implementato.
- **Pad:** campo dedicato al padding.

3.3 Type 4: AAL 5 Frame Record

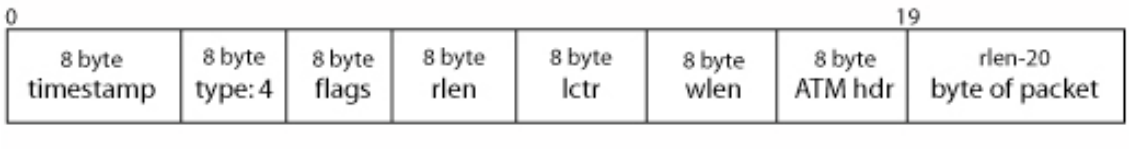

Figura 19. Struttura di un record AAL5.

Una caratteristica particolare di questo formato è che i campi dell'header usano un ordinamento dei byte diverso tra loro. Infatti il campo timestamp è salvato in little-endian[6] mentre tutti gli altri campi quali type, rlen, wlen e lctr sono in big-endian. I dati che compongono il carico sono salvati nell'ordine in cui vengono catturati senza applicare un ordinamento dei byte predefinito come avviene nei campi dell'header.

[6] Big-endian e little-endian. Sono due metodi differenti usati nei calcolatori per immagazzinare in memoria dati di dimensione superiore al byte. La differenza principale tra i due sistemi è data dall'ordine con il quale i byte costituenti il dato da immagazzinare vengono memorizzati:

- Nel Big-endian la memorizzazione inizia dal byte più significativo per fornire con il meno significativo.
- Nel Little-endian la memorizzazione inizia dal byte meno significativo per finire con il più significativo.

Per comprendere meglio come si presenta un file in formato ERF riportiamo in figura 26 un esempio del primo record di una traccia di rete acquisita con una scheda DAG Ethernet (Type 2):

TimeStamp								Type	Flags	rlen		lctr		wlen		offset	pad
0	103	4	0	224	216	202	63	2	0	0	72	0	0	2	25	129	159
0	0	12	70	92	209	0	224	30	142	49	113	8	0	69	0	2	7
248	159	64	0	62	6	0	0	10	0	0	1	10	1	112	209	204	210
0	80	228	142	119	145	154	159	46	205	128	24	135	192	0	0	0	0

Byte of packet

Figura 20. Esempio di record nel formato ERF.

I primi 18 byte della figura 26 rappresentano l'header di pacchetto:

- Timestamp: 0 103 4 0 224 216 202 63 istante in cui è stato acquisito il pacchetto riferito al 1 gennaio 1970. La rappresentazione nella forma secondi e microsecondi è: 1.0702584000000672E9. La funzione utilizzata per la conversione è riportata a pagina 50.
- Type: il valore 2 indica che la scheda di rete per l'acquisizione del traffico è una scheda ethernet.
- Flags: assume il valore 0 in quanto non sono parametri particolari da considerare.
- rlen: assume il 72 ed indica la lunghezza massima record.
- lctr: assume il valore 0 in quanto nessun pacchetto è stato perso
- offset: 129 indica che dall'inizio del frame non sono stati catturati 129 byte.
- pad: 159.

I rimanenti byte rappresentano il pacchetto catturato.

Nel capitolo successivo verrà riportata la conversione di questo pacchetto nel formato TCPDUMP in modo da facilitare la comprensione di quali campi sono di interesse per la conversione.

Capitolo 4 – Formato TCPDUMP

Il formato tcpdump è un molto diffuso in quanto presente nei sistemi Unix. L'applicazione tcpdump permette, utilizzando un link di rete, di visualizzare e salvare il traffico di rete in transito nel proprio link tramite linea di comando.
Il formato tcpdump è caratterizzato dalla presenza di un header della traccia in cui sono contenute informazioni sulla versione, l'accuratezza del timestamp, la posizione del pacchetto salvato e il tipo di link considerato. L'header di traccia è seguito da una serie di record che sono i pacchetti catturati dalla scheda di rete, ai quali viene aggiunto un header di pacchetto contenente informazioni sull'istante in cui il pacchetto è stato catturato, sul numero di byte catturati e sul numero di pacchetti contenuti nel record. L'header di traccia ha una dimensione di 24 byte mentre l'header di pacchetto ha una dimensione di 16 byte, invece il pacchetto (o carico) ha una dimensione variabile.
La struttura di una traccia tcpdump può essere schematizzata come in figura 27.

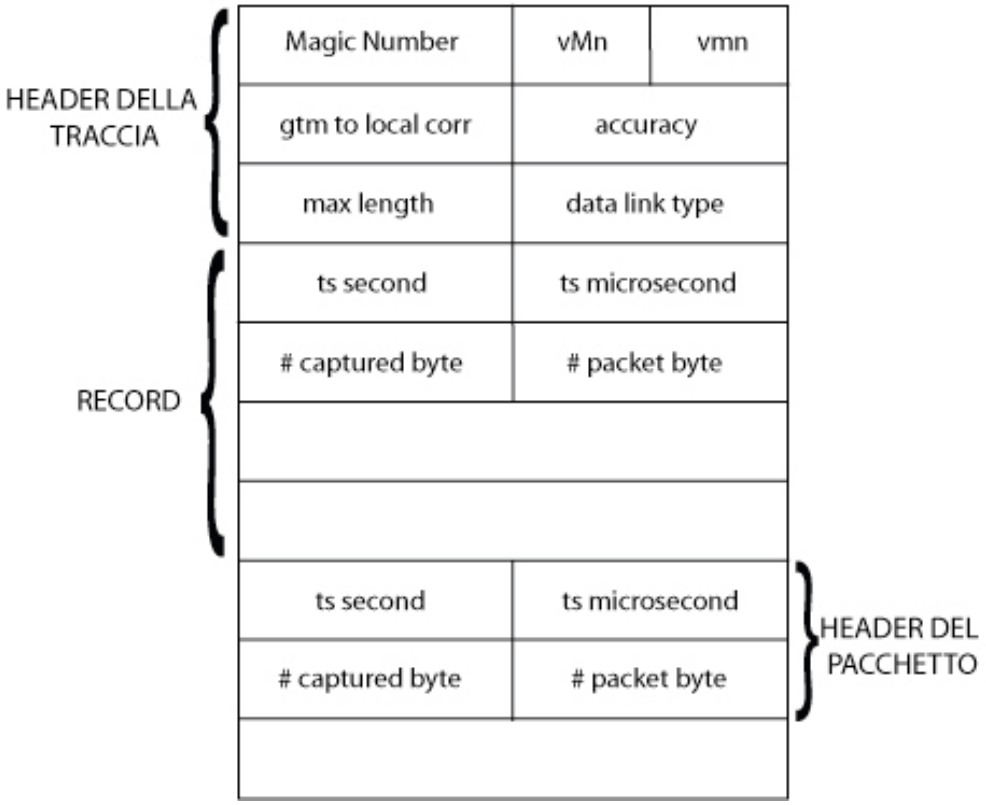

Figura 21. Specifiche dei byte che compongono l'header della traccia

- **Magic Number:** assume il valore a1b2c3d4. Poiché tutti i campi possono essere scritti in big-endian o little-endian, magic number è tra questi. Serve perché quando il programma legge il file, valutando il magic number capisce se il file è scritto con ordinamento dei byte uguale a quello della macchina che sta leggendo il file oppure no.
- **Major Version Number** (vMn): viene impostato al valore 2.
- **Minor Version Number** (vmn): viene impostato al valore 4.
- **Time Zone Offset** (gtm to local corr): non viene utilizzato quindi è impostato al valore zero.
- **TimeStamp accuracy** (accuracy): non viene utilizzato quindi è impostato al valore zero.
- **Snapshot length** o **Max length**: indica la dimensione in byte del pacchetto di taglia massima presente nel file.
- **Link Layer type:** indica il tipo di collegamento che è stato usato per catturare i pacchetti. Nel formato tcpdumo i numeri identificativi dei link di rete sono diversi da quelli del formato Endace DAG. Di seguito riportiamo alcuni identificativi di link:
  - **0** BSD loopback devices, except for later OpenBSD
  - **1** Ethernet e linux loopback devices
  - **6** 802.5 Token Ring
  - **104** Cisco HDLC
  - **105** 802.11

4.1 Specifiche dei byte che compongono l'header di pacchetto

- **Timestamp second** (ts second): contiene la rappresentazione dei secondi in cui è stato catturato il pacchetto. Anche in questo caso i secondi sono riferiti a partire dal 1 gennaio 1970.
- **Timestamp microsecond** (ts microsecond): contiene la rappresentazione dei microsecondi in cui è stato catturato il pacchetto.
- **Number captured byte:** contiene il numero di byte del pacchetto dati che deve essere catturato.
- **Packet Number:** contiene la lunghezza del pacchetto attuale.

Le informazioni vengono salvate in formato binario, i valori dei singoli byte salvati sono dei numeri interi compresi tra 0 e 255. In figura 28 è riportato la conversione del record usato come esempio nel capitolo precedente. Il risutalto che riportiamo di seguito è comprensivo di header di traccia:

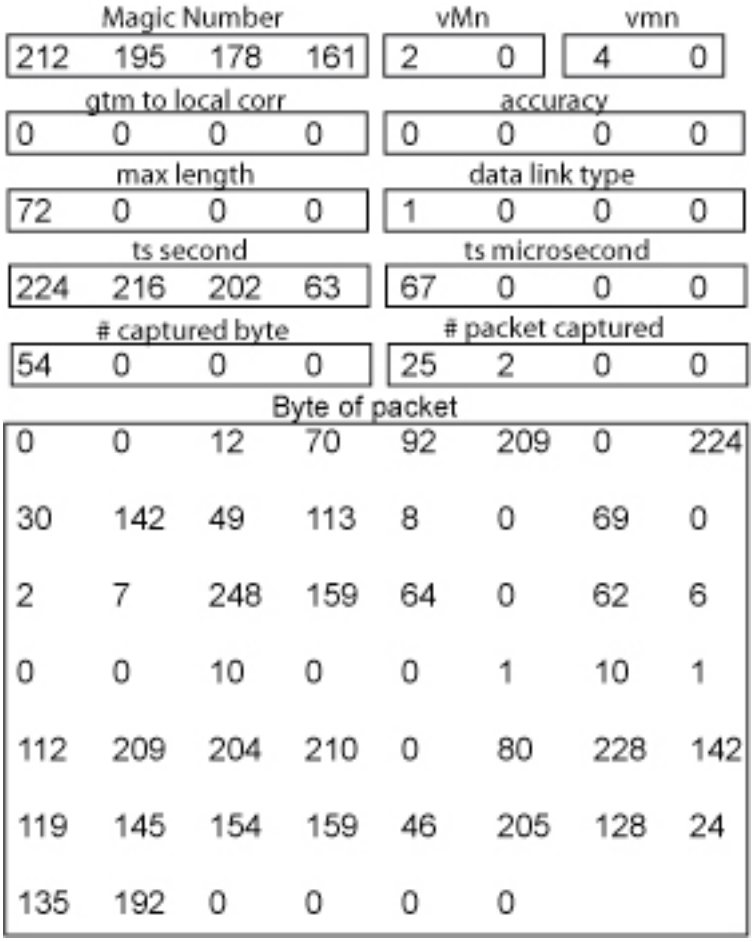

Figura 22. Esempio di formato TCPDUMP comprensivo di header di traccia.

I primi 24 byte rappresentano l'header di traccia, più precisamente:

- Magic Number: 212 195 178 161 corrisponde al valore a1b2c3d4
- vMn: 2 0 corrisponde al valore 2 in decimale
- vmn: 4 0 corrisponde al valore 4 in decimale
- gtm to local corr: 0 0 0 0
- accuracy: 0 0 0 0
- max Length: 72 0 0 0 corrisponde al valore rlen più grande tra tutti quelli dei pacchetti convertiti. Nel nostro esempio è 72

- data link type: 1 0 0 0 corrisponde alla scheda ethernet. Da notare che la convenzione introdotta nel formato tcpdump è diversa dal formato ERF in quanto nel formato ERF è 2.

I successivi 16 byte rappresentano l'header di pacchetto, più precisamente:
- ts second: 224 216 202 63 corrisponde al valore 1070258400s.
- ts microsecond: 67 0 0 0 corrisponde al valore 67 microsecondi.
- captured byte: 54 0 0 0 corrisponde al valore della lunghezza del pacchetto senza l'header di pacchetto.
- Packet number: 25 2 0 0.

i rimanenti byte rappresentano il carico.

Capitolo 5 – Descrizione del programma Time Interval Extractor

Nei capitoli precedenti abbiamo visto la descrizione di due formati utilizzati per salvare tracce di rete, quali il formato ERF, specifico delle schede di misura dell'azienda Endace, e il formato tcpdump caratteristico dei sistemi Unix.
La differenza principale di questi due formati è caratterizzata dalla precisione del campo timestamp. Il formato ERF è caratterizzato da un'elevata precisione in quanto il valore dell'istante di acquisizione viene assegnato direttamente dalla scheda di rete, mentre il timestamping basato su una NIC (utilizzando una scheda di rete comune) viene assegnato nel PC host dopo che il pacchetto è arrivato e quindi coincide approssimativamente con la fine del pacchetto.

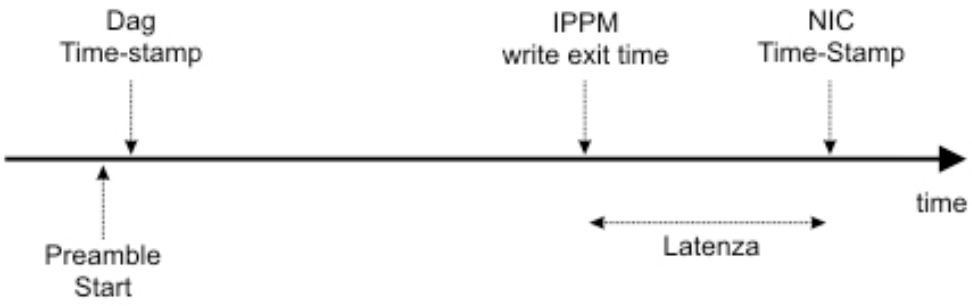

Figura 23. Confronto istante di acquizione tra scheda DAG e NIC

Poiché conoscere il reale istante di acquisizione del pacchetto è importante, si è sentita la necessità di convertire le informazioni acquisite con una scheda di rete ad alta precisione, come la scheda DAG, in un formato più diffuso e quindi più facile da gestire come il formato tcpdump, mantenendo la precisione del timestamp del formato DAG all'interno di limiti accettabili per le approssimazioni introdotte durante la conversione.
In questo capitolo sfrutteremo le conoscenze relative ai formati presentati nei capitoli precedenti per realizzare un programma capace di effettuare la conversione. In primo luogo presenteremo, mediante un diagramma di flusso, la struttura del programma andando a focalizzare le caratteristiche principali del

software. Nella parte centrale di questo capitolo andremo ad analizzare e visualizzare le parti principali del codice, allegando nella parte finale del capitolo l'intero codice sergente del programma.

5.1 Struttura del programma Time Interval Extractor

Il diagramma di flusso riportato in figura 30, rappresenta in maniera chiara i blocchi funzionali del programma.

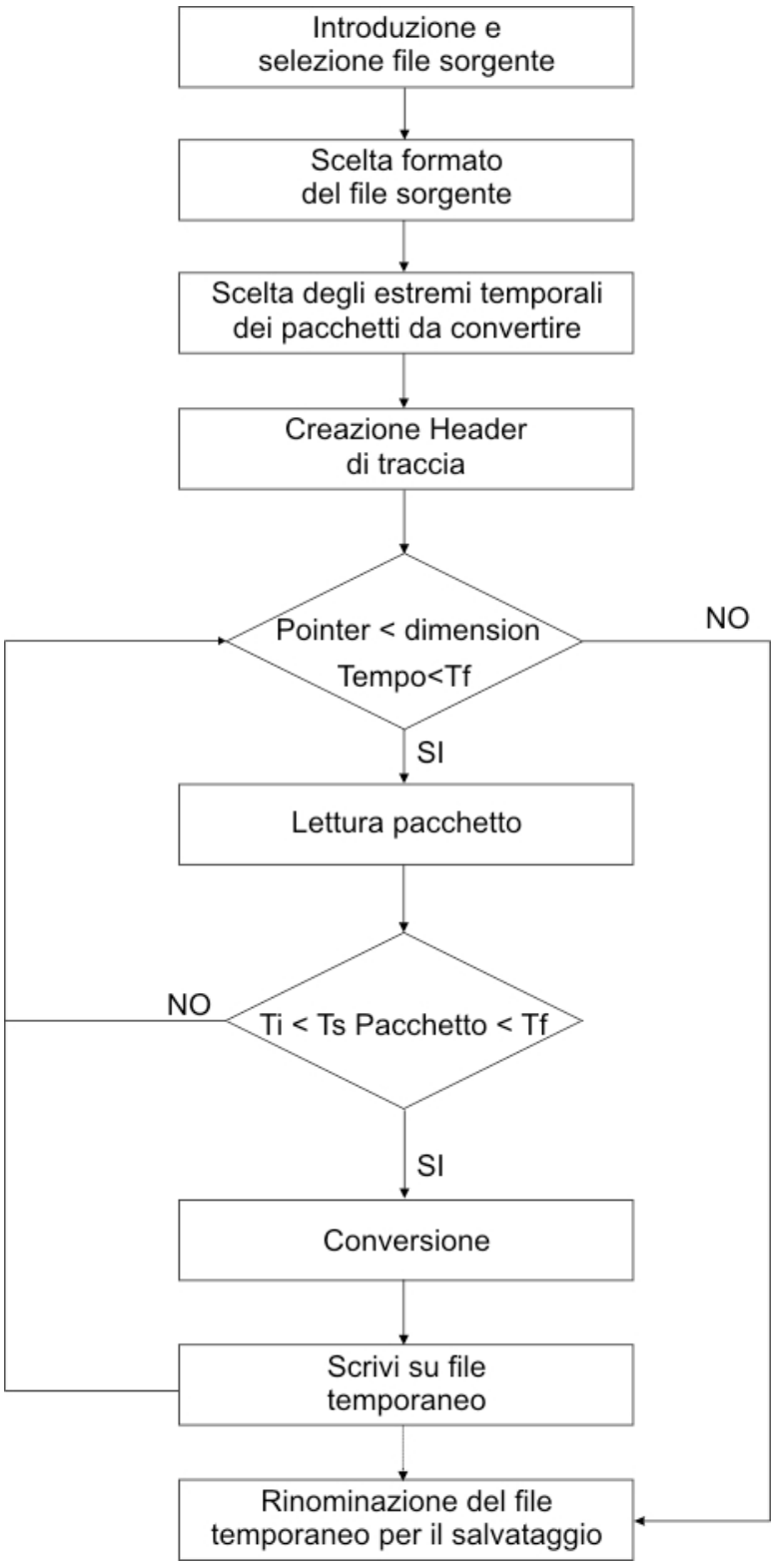

Figura 30. Struttura del programma.

Come si può vedere dal diagramma di flusso, la prima parte del programma è stata progettata per consentire all'utente di selezionare il file sorgente da convertire, il tipo di formato contenuto nel file sorgente e l'intervallo temporale dei pacchetti che saranno convertiti.
La prima finestra che si presenta all'utente (figura 31) è una breve presentazione del programma.

Figura 24. Presentazione del programma.

Dato l'ok, il programma aprirà una finestra con la stessa interfaccia del sistema operativo in uso, nella quale sarà chiesto di scegliere la posizione del file sorgente all'interno del disco.
Un esempio della finestra che incontreremo durante l'esecuzione del programma è riportata in figura 32.

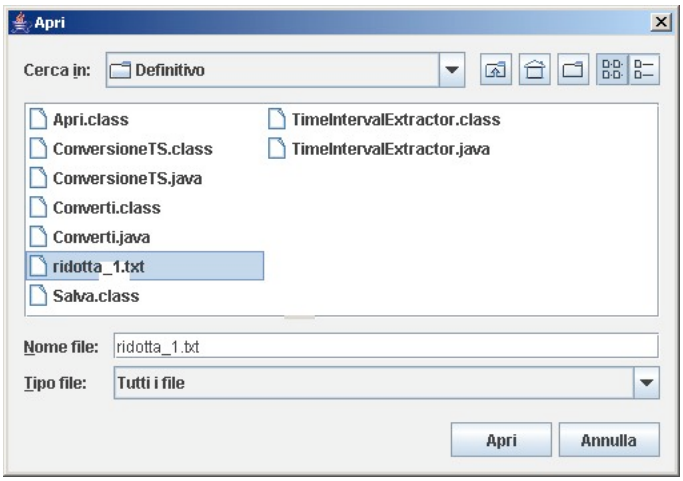

Figura 25. Selezionare il file sorgente.

A questo punto comparirà un menu a tendina come quello rappresentato in figura 33, in cui sarà chiesto di selezionare il formato dei pacchetti che compongono il file selezionato al passo precedente.

Figura 26. Formato

Questa scelta è molto importante in quanto abbiamo visto che gli header del pacchetto variano a seconda del tipo di link utilizzato per la cattura. Una scelta sbagliata del formato del file sorgente porterà ad una conversione errata.
Nella finestra successiva (Figura 34) sarà chiesto di inserire un numero intero, esperesso in millisecondi, che rappresenta il tempo di acquisizione del primo pacchetto che vogliamo considerare.
Nell'ultima finestra, prima dell'inizio della fase di conversione, sarà chiesto di inserire un numero intero, espresso anch'esso in millisecondi, per indicare l'istante temporale dell'ultimo pacchetto da convertire. Per esempio, se nella nostra traccia di partenza i pacchetti salvati hanno un intervallo temporale che va da 0 secondi a 25 secondi e noi siamo interessati alla porzione di traffico di rete compreso tra 1ms e 2,5 secondi, dovremo inserire nella prima finestra corrispondente al tempo di inizio 1, mentre nella finestra corrispondente al tempo finale dovremo inserire 2500.

Figura 27. Inserisci l'istante iniziale

A questo punto inizia la vera conversione, in quanto l'utente ha inserito tutte le informazioni necessarie all'elaborazione. La fase di conversione si sviluppa in una serie di passi sequenziali.

Per prima cosa il programma provvede alla creazione dell'header del file impostando, come abbiamo visto nel capitolo 4, i primi 24 byte ai valori standard caratteristici del formato tcpdump.
Il programma quindi entra in un ciclo da quale uscirà solamente quando avrà raggiunto la fine del file sorgente oppure quanto si tenterà di convertire il primo pacchetto successivo all'estremo superiore impostato dall'utente. Le operazioni previste all'interno del ciclo sono state organizzate in modo da ridurre al minimo la complessità e prevedono di:

- leggere l'header di pacchetto del record che stiamo considerando;
- convertire il timestamp del record;
- verificare se il pacchetto appartiene all'intervallo temporale di interesse: se appartiene all'intervallo questo viene convertito e poi salvato sul file temporaneo altrimenti non viene eseguita nessuna operazione e si passa al record successivo.

Un volta usciti dal ciclo per i formati previsti viene effettuato l'aggiornamento del campo maxlength dell'header del file, dopo di che apparirà una finestra in cui verrà chiesto dove salvare il file.

5.2 Analisi del programma

Il programma per la conversione di formato è costituito da 3 classi: ConvertiTS.java, Converti.java, TimeIntervalExtractor.java .

5.2.1 ConversioneTS.java

La classe *ConvertiTS* contiene una serie di metodi che permettono la conversione del timestamp dalla sua rappresentazione nel formato DAG alla rappresentazione in secondi e microsecondi contenuti in campi distinti, richiesta nel formato tcpdump.
Il metodo che permette di effettuare questa operazione si chiama Conversione e riceve come parametro di ingresso un array contenente la rappresentazione del timestamp. Questo metodo mediante un ciclo implementa la seguente formula:

$$time = time + \sum_{i=0}^{7} t[i] * 256^i * 2^{-32}$$

Inizialmente time è posto a zero, preleva il byte meno significativo dal timestamp e lo moltiplica per il peso corrispondente in byte(256^i), in fine viene moltiplicato per 2^{-32} per posizionare il punto decimale in modo opportuno. Tali operazioni vengono ripetute ad ogni iterazione.

Per comprendere meglio il funzionamento della formula precedente consideriamo il seguente esempio, in cui i singoli byte del timestamp di un pacchetto assumono i seguenti valori:

0 103 4 0 224 216 202 63 e il valore iniziale di time=0.

0) time = time + $0*1*2^{-32}$ = 0.0
1) time = time + $103*256*2^{-32}$ = 6.139278411865234E-6
2) time = time + $4*(256^2)*2^{-32}$ = 6.717443466186523E-5
3) time = time + $0*(256^3)*2^{-32}$ = 6.717443466186523E-5
4) time = time + $224*(256^4)*2^{-32}$ = 224.00006717443466
5) time = time + $216*(256^5)*2^{-32}$ = 55520.000067174435
6) time = time + $202*(256^6)*2^{-32}$ = 1.3293792000067174E7
7) time = time + $63*(256^7)*2^{-32}$ = 1.0702584000000672E9

I metodi *secondi* e *microsecondi* hanno il compito di fornire in uscita la parte in secondi e microsecondi del timestamp fornito come parametro di ingresso. Il formato tcpdump prevede di salvare in appositi campi distinti dell'header del pacchetto i due valori calcolati.

Una volta ottenuti i valori in secondi e microsecondi, non è possibile salvarli direttamente, bisogna riportare i risultati in byte. A tale scopo sono stati implementati i metodi *secondiToByte* e *microsecondiToByte* che restituiscono in un array la rappresentazione in byte del parametro in ingresso usando la formula:

$$input[i] = (int)(tempo - (\frac{tempo}{256}) * 256)$$

dove ad ogni iterazione del ciclo la variabile tempo viene aggiornata dividendola per 256 così da adattare il valore della variabile tempo al byte che sta per rappresentare.

5.2.2 Converti.java

La classe *Converti* ha la funzione di creare l'header di file e di pacchetto nel formato TCPDUMP partendo dalle informazioni ricevute nel formato DAG di partenza.
Il metodo per la creazione dell'header di file imposta i campi magic number, major version number, minor version number e time zone offset ai rispettivi valori di default che sono stati indicati nel capitolo 4. I campi max length e data link type vengono passati come parametri di ingresso. Da notare che il campo max length, che rappresenta la lunghezza massima tra tutti i pacchetti catturati, non è noto a priori ma sarà noto solo dopo l'analisi della traccia da convertire. Per questo motivo inizialmente il valore viene posto a zero per i formati a lunghezza variabile quali PoS HDLC, Ethernet e AAL5, mentre viene posto al valore 64 (valore massimo) per i formati a lunghezza fissa.
L'header di pacchetto viene creato, per ogni pacchetto convertito, dal metodo PacketHeaderCreation, che ha come parametro di ingresso un array e un tipo per identificare il formato DAG da convertire. L'array in ingresso contiene le informazioni che caratterizzano il pacchetto.
PacketHeaderCreation contiene al suo interno un array di dimensione pari a quella dell'intestazione del formato tcpdump, in cui verranno salvati i vari campi una volta convertiti, ed una serie di chiamate ad altri metodi per ottenere dall'array di ingresso le informazioni di interesse.
Il primo metodo ad essere invocato è *fillTS* il quale svolge la semplice funzione di copiatura dei primi 8 byte dell'array di ingresso a quello di destinazione. Questi byte rappresentano il valore del timestamp in secondi e microsecondi: in questo metodo non vengono eseguite operazioni di trasformazione in quanto sono previste nel programma principale.
Il metodo successivo che viene invocato è *fillcaptured*. Questo ha la funzione di estrarre dall'array di partenza il valore di rlen, che rappresenta la dimensione

del record, ed ottenere la dimensione del pacchetto. Ciò avviene semplicemente effettuando una operazione di sottrazione poiché tutti gli header di pacchetto nei vari formati DAG hanno dimensione fissa. Ottenuta la dimensione del pacchetto viene invocato il metodo returnToByte che riporta la dimensione di rlen espressa in valore decimale nella corrispettiva rappresentazione in byte. Il risultato ottenuto viene salvato nella parte bassa del campo number captured byte. Il metodo *fillpacket* ha la funzione di prelevare, per i formati che contengono il campo wlen, i byte dall'array di ingresso e riportarli su quello di uscita. Durante la fase di copiatura bisogna prevedere un opportuno riordinamento dei byte perché il campo wlen è scritto come big-endian mentre tutti i campi del tcpdump sono scritti come little-endian.

5.2.3 TimeIntervalExtractor.java

TimeIntervalExtractor è la classe principale del programma in quanto contiene la sequenza delle operazioni da svolgere per la conversione di formato.
Dopo una prima parte di definizione ed inizializzazione delle variabili, incontriamo il primo blocco funzionale di questa classe che ha la funzione di realizzare le finestre di dialogo necessario a guidare l'utente nella scelta del file da convertire, e nell'inserimento degli estremi dell'intervallo temporale. Ogni finestra di dialogo è controllata da un try-catch. L'utente potrebbe decidere di premere il tasto annulla per uscire subito dal programma in seguito ad errori di scelta nei passaggi precedenti.
In base alle scelte dell'utente il programma si porta in una delle sezioni corrispondenti ai formati. La struttura di ogni sezione è uguale, ma cambiano alcune operazioni che sono gestite nelle classi spiegate precedentemente.
La porzione di codice che precede il ciclo di conversione, e che riportiamo in figura 35, ha lo scopo di fissare l'origine relativa dei tempi con il valore del primo timestamp.

```
int origine[] = new int[8];
for(int i=0; i<8;i++)
{origine[i]= packets.read();}
TSprimo=TimeStamp.conversione(origine);
packets.seek(0);
```

Figura 28. Segmento di codice per fissare l'origine relativa dei tempi

Questa operazione viene fatta in quanto il campo timestamp, come detto in precedenza, contiene un riferimento temporale rispetto al 1 Gennaio 1970. Ovviamente l'utente al momento di inserire gli estremi dell'intervallo di tempo da considerare non avrà inserito dei valori temporali che si riferiscono a tale data, ma avrà inserito dei tempi relativi considerando che il primo pacchetto del file sia stato acquisito all'istante zero. Quindi questa operazione è importante per far corrispondere i valori inseriti dall'utente con quelli contenuti nei campi del timestamp. Dopo la definizione dell'origine relativa dei tempi il programma entra nel ciclo dal quale uscirà quando avrà raggiunto la fine del file oppure quando si richiede di convertire un pacchetto che segue l'estremo temporale superiore inserito dall'utente.

All'interno del ciclo la prima operazione che viene svolta è quella di leggere l'header del pacchetto e salvarlo in un array. A questo punto invocando i metodi della classe *ConvertiTS* si estrae il timestamp del pacchetto che si vuole convertire e lo si confronta con gli estremi inseriti dall'utente. Se il timestamp calcolato appartiene all'intervallo di interesse si procede con la creazione dell'header del pacchetto invocando il metodo *PacketHeaderCreation* e successivamente copiando il contenuto del pacchetto nel nuovo file. Se il pacchetto non appartiene all'intervallo di interesse si salta direttamente al pacchetto successivo mediante il metodo *seek* presente all'interno della classe RandomAccessFile. Questa serie di operazioni si ripete per tutti i pacchetti del file di partenza.

Una volta usciti dal ciclo, per i formati a lunghezza variabile bisogna effettuare l'aggiornamento del campo maxlength che inizialmente era stato posto a zero. Per fare questo aggiornamento si crea un nuovo file temporaneo in cui andranno copiati tutti i byte del file appena creato sostituendo nel campo maxlength il valore determinato durante la conversione.

```
RandomAccessFile FileTemporaneo=new RandomAccessFile("tmp.txt","r");
            long lunghezza = FileTemporaneo.length();
            FileWriter finale = new FileWriter("tmp2.txt");
            for (int j = 0; j < 16; j++)
                finale.write(FileTemporaneo.read());
            finale.write(tmp1[1]);
            finale.write(tmp1[0]);
            FileTemporaneo.read();
            FileTemporaneo.read();
            for (int j = 0; j < lunghezza - 18; j++)
                finale.write(FileTemporaneo.read());
            finale.close();
            FileTemporaneo.close();
```

Figura 29. Codice per l'aggiornamento del campo Max Length.

Il programma quindi termina chiedendo, mediante una finestra di dialogo, in quale posizione su disco si vuole salvare il file convertito.

Una osservazione importante da fare è l'utilizzo della classe RandomAccessFile come elemento fondamentale per mantenere le prestazioni del programma di conversione in un tempo accettabile. Questa classe è usata per consentire delle operazioni di input e di output accedendo direttamente ad opportune posizioni all'interno di un file. In questo modo si evita di costruire un flusso continuo di informazioni attraverso uno stream. Operare così è molto utile quando è necessario accedere ad una precisa posizione nel file, poichè usando altre classi come FileReader, sarebbe necessario scorrere sotto la testina di lettura tutto il file fino a raggiungere la posizione cercata. RandomAccessFile è usata in particolar modo quando i file contengono blocchi di record di una dimensione prefissata e nota, nel nostro caso erano gruppi di byte. Spostando la testina mediante la funzione seek() è possibile accedere senza troppi sforzi ad un particolare byte per effettuare le varie operazioni di lettura o scrittura. È infatti possibile operare usando dei permessi di lettura e scrittura sui file, consentendo l'accesso ai file in modalità di sola lettura o di lettura e scrittura; al costruttore di RandomAccessFile viene passato un argomento che può essere r o rw ad indicare questa modalità. RandomAccessFile introduce numerosi metodi oltre a quelli ereditati da Object e implementati da DataInput e DataOutput. I più importanti sono: getFilePointer() che restituisce la corrente

posizione della tesina all'interno del file, length() che restituisce la lunghezza del file in byte, seek() che sposta la tesina di un certo offset in byte rispetto all'inizio del file readLine() che ritorna la successiva linea di testo.

Conclusioni

Lo scopo di questa tesi è lo sviluppo di un software per la conversione di formato delle tracce di traffico di rete. La realizzazione di questa applicazione ha consentito di affrontare lo sviluppo di un programma partendo da zero.
La prima fase dello sviluppo consisteva nello studio dei due formati, per capire il significato dei campi che compongono l'header di traccia e di pacchetto.
La seconda fase ha riguardato l'adattamento delle informazioni dal formato di partenza a quello di destinazione valutando anche le differenze di memorizzazione delle informazioni.
Una volta studiati i due formati è iniziata l'implementazione del software, le cui difficoltà maggiori sono state la conversione del timestamp e la scelta di quale classe tra FileReader e RandomAccessFile utilizzare per leggere i byte del file. Alla fine abbiamo utilizzato la classe RandomAccessFile in quanto FileReader leggeva valori errati dei byte del file alterando la conversione.
Per quanto riguarda l'errore di lettura presentato dalla classe FileReader non siamo riusciti a dare una spiegazione plausibile.
In fine siamo passati alla fase di testing del software convertendo varie tracce di traffico dal formato ERF al formato TCPDUMP verificando, mediante l'uso di Ethereal e matlab la correttezza dei file ottenuti dal nostro programma. I risultati ottenuti nelle conversioni hanno dimostrato la correttezza del nostro software in quanto il contenuto dei pacchetti veniva trasferito nel nuovo formato senza alcuna alterazione.

Codice sergente del programma

Codice della classe ConversioneTS.java

```
/**  *    * @author Stefano Dindo, Simone Rossi    * @version August 2006  */
import java.*;
import java.math.*;
class ConversioneTS
{
  private double time;
  private long pot;
  public ConversioneTS() { }

  /************************************************
  Questo metodo ha la funzione di convertire il Timestamp
  rappresentato in Byte nella forma secondi, microsecondi
  riferiti al 1 gennaio 1970.
  ************************************************/
  public double conversione(int t[])
  {
    time=0;
    pot=1;
    for(int i=0; i<8; i++)
    {
      time=time+((t[i]*pot)*(Math.pow(2,-32)));
      pot=pot*256;
    }
    return time;
  }
  /************************************************
  Fornisce il valore della sola parte in secondi
  del Timestamp ricevuto in ingresso.
  ************************************************/
  public int secondi(double t)
  {
    int sec=(int)t;
    return sec;
  }
  /************************************************
  Fornisce il valore della parte in microsecondi
  del Timestamp ricevuto in ingresso.
  ************************************************/
  public double microsecondi(double t)
  {
    int sec=(int)t;
    double micro= (t-sec)*(Math.pow(10,6));
    return micro;
  }

  /************************************************
  Il metodo seguente restituisce la rappresentazione
  in byte, mediante un array, del valore ricevuto in
  ingresso.
  ************************************************/
  public int []secondiToByte(int [] input, int sec)
  {
   for(int i = 0; i < 4; i++)
   {
      input[i]=(int)(sec-(sec/256)*256);
      sec=sec/256;
   }
   return input;
  }
```

```
  public int [] microsecondiToByte(int [] input, int micro)
  {
    for(int i = 4; i < 8; i++)
    {
       input[i]=(int)(micro -(micro/256)*256);
       micro=micro/256;
    }
            return input;
  }
}
```

Codice sorgente della classe Converti.java

```
public class Converti
{           private int [] FileHeader = new int[24];
            private int [] PacketHeader;
            /*****************************************
            Il metodo FileHeaderCreation ricevuto
            provvede alla creazione dell'header del file
            poichè la maggior parte dei campi hanno un valore di dafault
            mentre i campi data link type e snap shot length vengono
            passati come parametri di ingresso
            *****************************************/
            public int [] FileHeaderCreation(int DataLinkType, int snapshotLength)
            {
                        FileHeader[0] = 212;
                        FileHeader[1] = 195;
                        FileHeader[2] = 178;
                        FileHeader[3] = 161;
                        FileHeader[4] = 2;
                        FileHeader[5] = 0;
                        FileHeader[6] = 4;
                        FileHeader[7] = 0;
                        FileHeader[8] = 0;
                        FileHeader[9] = 0;
                        FileHeader[10] = 0;
                        FileHeader[11] = 0;
                        FileHeader[12] = 0;
                        FileHeader[13] = 0;
                        FileHeader[14] = 0;
                        FileHeader[15] = 0;
                        FileHeader[16] = snapshotLength;
                        FileHeader[17] = 0;
                        FileHeader[18] = 0;
                        FileHeader[19] = 0;
                        FileHeader[20] = DataLinkType;
                        FileHeader[21] = 0;
                        FileHeader[22] = 0;
                        FileHeader[23] = 0;
                        return FileHeader;
            }
            /*****************************************
            Il metodo packetHeaderCreation ha la funzione di creare
            l'header di pacchetto nel formato TCPDUMP partendo da
            un array di ingresso che contiene, la rappresentazione in byte,
            dei campi dell'header del formato DAG. Viene passato
            anche come parametro di ingresso una variabile type che serve a
          distinguere  i vari header.
            *****************************************/
            public int [] PacketHeaderCreation(int [] input, int type)
            {
                        PacketHeader = new int [16];
                        fillTS(input);
                        fillCaptured(input,type);
                        fillPacket(input,type);
                        return PacketHeader;
            }
```

```
/*******************************************
Poichè il timestamp viene convertito nel programma principale
questo metodo esegue la copia del timestamp dall'array di ingresso
a quello di uscita.
*******************************************/
private void fillTS(int [] input)
{
        for (int i = 0; i < 8; i++)
                PacketHeader[i] = input[i];
}
/*******************************************
FillCaputerd, a seconda del tipo di formato da convertire, ricava dall'array
dall'array di partenza la lunghezza del pacchetto e la salva nel campo
captured byte nella parte più significativa.
*******************************************/
private void fillCaptured(int [] input, int type)
{
        int length=0;
        int [] newLength;
        if (type == 1 | type == 3)
        {
         PacketHeader[11] = 0;
       PacketHeader[10] = 0;
         length = getRlen(input[10], input[11]) - 20;
         newLength = returnToByte(length);
         PacketHeader[9] = newLength[0];
         PacketHeader[8] = newLength[1];
        }
        if (type == 2)
        {
         PacketHeader[11] = 0;
         PacketHeader[10] = 0;
         length = getRlen(input[10], input[11]) - 18;
         newLength = returnToByte(length);
         PacketHeader[9] = newLength[0];
         PacketHeader[8] = newLength[1];
        }
        if (type == 4 | type == 5)
        {
         PacketHeader[11] = 0;
         PacketHeader[10] = 0;
         length = getRlen(input[10], input[11]) - 16;
         newLength = returnToByte(length);
         PacketHeader[9] = newLength[0];
         PacketHeader[8] = newLength[1];
        }
        if (type == 6)
        {
         length= 54; // perche' e' 64 -10
         newLength = returnToByte(length);
         PacketHeader[9] = newLength[0];
         PacketHeader[8] = newLength[1];
        }
}
/*******************************************
Fillpacket, a seconda del tipo di formato da convertire, ricava dall'array
di partenza il campo Wlen e la salva nel campo
packet byte nella parte più significativa. Mentre
per i formati DAG che non prevedono il capo Wlen,
nella conversione il campo packet byte verrà posto a zero.
*******************************************/
private void fillPacket(int []input, int type)
{
  PacketHeader[14] = 0;
PacketHeader[15] = 0;
   if(type==1 | type == 2 | type == 3 | type == 5)
   {
       PacketHeader[12] = input[15];
       PacketHeader[13] = input[14];
```

```
            }
            else
            {
                PacketHeader[12] = 0;
                PacketHeader[13] = 0;
            }
        }
        /*********************************************
        getLen serve per ottenere il valore in rappresentazione
        decimale del campo Rlen a partire dalla sua rappresentazione
        in byte.
        *********************************************/
        public int getRlen(int rlen1, int rlen2) //rlen e' in big endian
        {
            int numeroBytes;
            numeroBytes = rlen1 * 256 + rlen2;
         return numeroBytes;
        }
        /*********************************************
        A partire dalla rappresentazione in decimale del campo
        rlen lo converte nei rispettivi valori in byte.
        *********************************************/
        public int[] returnToByte(int length)
        {
                int [] a = new int [2];
                a[1] = length%256;
                a[0] = length/256;
                return a;
        }
}
```

Codice sorgente della classe TimeIntervalExtractor.java

```
import javax.swing.*;
import java.awt.*;
import java.io.*;
public class TimeIntervalExtractor
{
   public static void main(String [] argv) throws IOException
   {
      /*******************************************************
      DICHIARAZIONE  E INIZIALIZZAZIONE DELLE VARIABILI
      *******************************************************/
      double Ti=0;              //istante iniziale
      double Tf=0;              //istante finale
      double TS;                //Time Stamp del pacchetto
      double tempo=0;
      double TSprimo;
      int PacketHeaderLength;    //lunghezza dell'header di pacchetto
      int array[];              //array contenente l'header di pacchetto
      int rlen;                 //esprime la lunghezza del pacchetto compreso header
      int s;                    // secondi
      int m;                    // microsecondi
      int MaxLength=0;          // massima lunghezza di pacchetto riscontrata
      boolean MaxlengthVariabile = false;
      long dimension;
      long pointer=0;           // puntatore di pacchetto
      Object format = null;      //tipo di link layer
      Apri filein = null;
      Salva out = null;
      /*******************************************************
      FINESTRE DI DIALOGO
      *******************************************************/
      Object text = "Questo programma converte in formato TCPDUMP, i  pacchetti catturati con scheda di rete DAG
appaartenenti ad un intervallo temporale definito dall'utente." + "\n" + "Vuoi continuare?";
      String title = "Time Interval Extractor";
      int messageType = 1;
      int optionType = 2;
      JOptionPane pane = new JOptionPane(text, messageType, optionType);
      int returnVal = pane.showConfirmDialog(pane, text, title,optionType, messageType);
```

```
        if (returnVal == 2)
            System.exit(0);
        filein = new Apri();
        Object [] formati = {"ERF HDLC","ERF ETHERNET","ERF AAL5","DAG 3.0 ATM","DAG 3.0 SONET(non
implementato)","DAG 3.0 ETHERNET"};
        format = pane.showInputDialog(null, "Scegli il formato della traccia:", title, 1, null, formati, formati[0]);
        if (format == null)
            System.exit(0);
        String Tin = "";
        boolean fatto = false;
        while(fatto == false)
        {
            try
            {
                Tin = pane.showInputDialog("Inserisci l`istante iniziale (in ms):");
                if (Tin == null)
                    System.exit(0);
                Ti = Integer.parseInt(Tin)*Math.pow(10,-3);
                fatto = true;
            }
            catch (NumberFormatException e)
            {
                    System.out.println("Attenzione, devi digitare un numero intero!");
                fatto = false;
            }
        }
        String Tfin = "";
        fatto = false;
        while (fatto == false)
        {
            try
            {
                Tfin = pane.showInputDialog("Inserisci l`istante finale (in ms):");
                if (Tfin == null)
                    System.exit(0);
                Tf = Integer.parseInt(Tfin)*Math.pow(10,-3);
                fatto = true;

            }
            catch (NumberFormatException e)
            {
                System.out.println("Attenzione, devi digitare un numero intero!");
                fatto = false;
            }
        }
        RandomAccessFile packets = new RandomAccessFile(filein.getFile(),"r");
        ConversioneTS TimeStamp = new ConversioneTS();
        Converti Header = new Converti();
        FileWriter fileout = new FileWriter("tmp.txt");

        //Se l'utente sceglie il formato HDLC...
        if(format.equals("ERF HDLC"))
        {
            MaxlengthVariabile = true;
            /*********************************************
            SCRITTURA HEADER DI FILE
            *********************************************/
             // viene passato come snapshotLength zero
                  //perche' il valore non e' fissato e lo si trova alla fine
            int harray[] = Header.FileHeaderCreation(104,0);
            for(int i=0; i<24; i++)
            {       fileout.write(harray[i])    }
            /*********************************************
            COSTRUZIONE HEADER DI PACCHETTO
            *********************************************/
            TSprimo = 0;
            PacketHeaderLength = 20;
            array = new int[PacketHeaderLength];
            dimension = packets.length();
            /*********************************************
            Calcolo l'origine relativa del file
            *********************************************/
            int origine[] = new int[8];
            for(int i=0; i<8;i++)
            {   origine[i]= packets.read();    }
            TSprimo=TimeStamp.conversione(origine);
            packets.seek(0); // riporto il puntatore all'origine del file
```

```
    //ciclo lettura intero file
    while( (pointer < dimension) & (tempo< Tf) )
        {
       for(int i=0; i<PacketHeaderLength; i++)
       {     array[i]=packets.read();   }
       TS=TimeStamp.conversione(array);
       tempo = TS - TSprimo;
       rlen = Header.getRlen(array[10],array[11]);

       // cerco la massima lunghezza di record da inserire nell'header di file
       if(rlen > MaxLength)
       {   MaxLength = rlen;      }

       //Controllo che il pacchetto sia nell'intervallo di tempo richiesto
       if (Ti <= tempo && tempo<= Tf)
       {
          // calcolo la parte in secodi del timestamp
          s = TimeStamp.secondi(TS);
 // calcolo la parte in microsecondi del timestamp
              m = (int)(TimeStamp.microsecondi(TS));

          /*********************************************
          COSTRUZIONE TIMESTAMP
          *********************************************/
  array = TimeStamp.secondiToByte(array,s);
  array = TimeStamp.microsecondiToByte(array,m);
          /*********************************************
          SCRITTURA HEADER DEL PACCHETTO
          *********************************************/
          int arrayfinale[] = Header.PacketHeaderCreation(array,1);
          for(int i=0; i<arrayfinale.length; i++)
          {
            fileout.write(arrayfinale[i]);
              }

          /*********************************************
          SCRITTURA DATI DEL PACCHETTO
          *********************************************/
          for(int i = 0; i < rlen-PacketHeaderLength; i++)
          {
             fileout.write(packets.read());
          }
       }//fine if
       //Se non appartiene all'intervallo sposto il puntatore alla posizione del
            nuovo pacchetto.
       else
       {
          packets.seek(pointer + rlen);
       }
       pointer= pointer+rlen;

    }
    //fine ciclo while
} // Fine ERF HDLC

//Se l'utente sceglie il formato Ethernet...

if(format.equals("ERF ETHERNET"))
{
    MaxlengthVariabile = true;
    /*********************************************
    SCRITTURA HEADER DI FILE
    *********************************************/
     // viene passato come snapshotLength zero
          //perche' il valore non e' fissato e lo si trova alla fine
    int harray[] = Header.FileHeaderCreation(1,0);
    for(int i=0; i<24; i++)
    {
       fileout.write(harray[i]);
    }
    /*********************************************
    COSTRUZIONE HEADER DI PACCHETTO
    *********************************************/
    TSprimo = 0;
    PacketHeaderLength = 18;
    array = new int[PacketHeaderLength];
    dimension = packets.length();
```

```
        /*********************************************
    Calcolo l'origine relativa del file
    *********************************************/
    int origine[] = new int[8];
    for(int i=0; i<8;i++)
    {
       origine[i]= packets.read();
    }
    TSprimo=TimeStamp.conversione(origine);
    packets.seek(0);

    //ciclo lettura intero file
    while( (pointer < dimension) & (tempo< Tf) )
    {
       for(int i=0; i<PacketHeaderLength; i++)
       {
          array[i]=packets.read();
       }
       TS=TimeStamp.conversione(array);
       tempo = TS - TSprimo;
       rlen = Header.getRlen(array[10],array[11]);

       // cerco la massima lunghezza di record da inserire nell'header di file
       if(rlen > MaxLength)
       { MaxLength = rlen; }

       //Controllo che il pacchetto sia nell'intervallo di tempo richiesto
       if (Ti <= tempo && tempo<= Tf)
       {
             // calcolo la parte in secodi del timestamp
          s = TimeStamp.secondi(TS);
          // calcolo la parte in microsecondi del timestamp
          m = (int)(TimeStamp.microsecondi(TS));
          /*********************************************
          COSTRUZIONE TIMESTAMP
          *********************************************/
                array = TimeStamp.secondiToByte(array,s);
                array = TimeStamp.microsecondiToByte(array,m);
          /*********************************************
          SCRITTURA HEADER DEL PACCHETTO
          *********************************************/
          int arrayfinale[] = Header.PacketHeaderCreation(array,2);
          for(int i=0; i<arrayfinale.length; i++)
          {
            fileout.write(arrayfinale[i]);
          }
          /*********************************************
          SCRITTURA DATI DEL PACCHETTO
          *********************************************/
          for(int i = 0; i < rlen-PacketHeaderLength; i++)
          {
             fileout.write(packets.read());
          }
       }//fine iff
       //Se non appartiene all'intervallo sposto il puntatore alla posizione del
          nuovo pacchetto.
       else
       {
          packets.seek(pointer + rlen);
       }
       pointer= pointer+rlen;

    }
    //fine ciclo while
}   // Fine ERF ethernet

//Se l'utente sceglie il formato AAL5...
if(format.equals("ERF AAL5"))
{
    MaxlengthVariabile = true;
    /*********************************************
    SCRITTURA HEADER DI FILE
    *********************************************/
    // viene passato come snapshotLength zero
        //perche' il valore non e' fissato e lo si trova alla fine
    int harray[] = Header.FileHeaderCreation(100,0);
    for(int i=0; i<24; i++)
```

```
    {
        fileout.write(harray[i]);
    }
    /*******************************************
    COSTRUZIONE HEADER DI PACCHETTO
    *******************************************/
    TSprimo = 0;
    PacketHeaderLength = 20;
    array = new int[PacketHeaderLength];
    dimension = packets.length();

    int origine[] = new int[8];
    for(int i=0; i<8;i++)
    {
        origine[i]= packets.read();
    }
    TSprimo=TimeStamp.conversione(origine);
    packets.seek(0); // riporto il puntatore all'origine del file

    //ciclo lettura intero file
    while( (pointer < dimension) & (tempo < Tf) )
     {
        for(int i=0; i<PacketHeaderLength; i++)
        {
            array[i]=packets.read();
        }
        TS=TimeStamp.conversione(array);
        tempo = TS - TSprimo;
        rlen = Header.getRlen(array[10],array[11]);

        // cerco la massima lunghezza di record da inserire nell'header di file
        if(rlen > MaxLength)
        { MaxLength = rlen;  }

        //Controllo che il pacchetto sia nell'intervallo di tempo richiesto
        if (Ti <= tempo && tempo<= Tf)
        {
            // calcolo la parte in secodi del timestamp
            s = TimeStamp.secondi(TS);
                 // calcolo la parte in microsecondi del timestamp
            m = (int)(TimeStamp.microsecondi(TS));
            /*******************************************
            COSTRUZIONE TIMESTAMP
            *******************************************/
            array = TimeStamp.secondiToByte(array,s);
                 array = TimeStamp.microsecondiToByte(array,m);
            /*******************************************
            SCRITTURA HEADER DEL PACCHETTO
            *******************************************/

            int arrayfinale[] = Header.PacketHeaderCreation(array,3);
            for(int i=0; i<arrayfinale.length; i++)
            {
              fileout.write(arrayfinale[i]);
            }

            /*******************************************
            SCRITTURA DATI DEL PACCHETTO
            *******************************************/

            for(int i = 0; i < rlen-PacketHeaderLength; i++)
            {
                fileout.write(packets.read());
            }
        }//fine if
        //Se non appartiene all'intervallo sposto il puntatore alla posizione del
            nuovo pacchetto.
        else
        {
            packets.seek(pointer + rlen);
        }
        pointer= pointer+rlen;

    }
    //fine ciclo while

} //Fine ERF AAL 5
```

```
//Se l'utente sceglie il formato ATM...

if(format.equals("DAG 3.0 ATM"))
{
    MaxlengthVariabile = false;
    /*********************************************
    SCRITTURA HEADER DI FILE
    *********************************************/
    // viene passato come snapshotLength 64
         //in quanto il formato è a lunghezza fissa
    int harray[] = Header.FileHeaderCreation(100,64);
    for(int i=0; i<24; i++)
    {
        fileout.write(harray[i]);
    }

    /*********************************************
    COSTRUZIONE HEADER DI PACCHETTO
    *********************************************/
    TSprimo = 0;
    PacketHeaderLength = 16;
    array = new int[PacketHeaderLength];
    dimension = packets.length();

    int origine[] = new int[8];
    for(int i=0; i<8;i++)
    {
        origine[i]= packets.read();
    }
    TSprimo=TimeStamp.conversione(origine);
    packets.seek(0); // riporto il puntatore all'origine del file

    //ciclo lettura intero file
    while( (pointer < dimension) & (tempo < Tf))
    {
        for(int i=0; i<PacketHeaderLength; i++)
        {
            array[i]=packets.read();
        }
        TS=TimeStamp.conversione(array);
        tempo = TS - TSprimo;
        rlen = 64;

        //Controllo che il pacchetto sia nell'intervallo di tempo richiesto
        if (Ti <= tempo && tempo<= Tf)
        {
                // calcolo la parte in secodi del timestamp
            s = TimeStamp.secondi(TS);
                 // calcolo la parte in microsecondi del timestamp
            m = (int)(TimeStamp.microsecondi(TS));
            /*********************************************
            COSTRUZIONE TIMESTAMP
            *********************************************/
            array = TimeStamp.secondiToByte(array,s);
                array = TimeStamp.microsecondiToByte(array,m);
            /*********************************************
            SCRITTURA HEADER DEL PACCHETTO
            *********************************************/
            int arrayfinale[] = Header.PacketHeaderCreation(array,4);
            for(int i=0; i<arrayfinale.length; i++)
            {
              fileout.write(arrayfinale[i]);
            }

            /*********************************************
            SCRITTURA DATI DEL PACCHETTO
            *********************************************/
            for(int i = 0; i < rlen-PacketHeaderLength; i++)
            {
                fileout.write(packets.read());
            }
        }//fine if

        //Se non appartiene all'intervallo sposto il puntatore alla posizione del
            nuovo pacchetto
        else
```

```
        {
            packets.seek(pointer + rlen);
        }
        pointer= pointer+rlen;

    }
    //fine ciclo while
} // Fine DAG 3.0 ATM
//Se l'utente sceglie il formato SONET...
if(format.equals("DAG 3.0 SONET"))
{
    /************************************
    Questo formato non e' stato implementato in quanto  bisognerebbe
     Creare dei pseudo pacchetti ethernet a causa dei vari strati di
     incapsulamento
     ************************************/
} // Fine DAG 3.0 SONET

//Se l'utente sceglie il formato DAG Ethernet...

if(format.equals("DAG 3.0 ETHERNET"))
{
    MaxlengthVariabile = false;
   /*********************************************
    SCRITTURA HEADER DI FILE
    *********************************************/
    // viene passato come snapshotLength 64
        //in quanto il formato è a lunghezza fissa
    int harray[] = Header.FileHeaderCreation(1,64);
    for(int i=0; i<24; i++)
    {
        fileout.write(harray[i]);
    }

    /*********************************************
    COSTRUZIONE HEADER DI PACCHETTO
    *********************************************/
    TSprimo = 0;
    PacketHeaderLength = 10;
    array = new int[PacketHeaderLength];
    dimension = packets.length();

    int origine[] = new int[8];
    for(int i=0; i<8;i++)
    {
        origine[i]= packets.read();
    }
    TSprimo=TimeStamp.conversione(origine);
    packets.seek(0);

    //ciclo lettura intero file
    while( (pointer < dimension) & (tempo < Tf) )
    {
        for(int i=0; i<PacketHeaderLength; i++)
        {
            array[i]=packets.read();
        }
        TS=TimeStamp.conversione(array);
        tempo = TS - TSprimo;
        rlen = 64;
        //Controllo che il pacchetto sia nell'intervallo di tempo richiesto
        if (Ti <= tempo && tempo<= Tf)
        {
                // calcolo la parte in secodi del timestamp
            s = TimeStamp.secondi(TS);
                // calcolo la parte in microsecondi del timestamp
            m = (int)(TimeStamp.microsecondi(TS));
            /*********************************************
            COSTRUZIONE TIMESTAMP
            *********************************************/
            array = TimeStamp.secondiToByte(array,s);
                 array = TimeStamp.microsecondiToByte(array,m);

            /*********************************************
            SCRITTURA HEADER DEL PACCHETTO
            *********************************************/
            int arrayfinale[] = Header.PacketHeaderCreation(array,6);
```

```
            for(int i=0; i<arrayfinale.length; i++)
            {
              fileout.write(arrayfinale[i]);
            }

            /*********************************************
            SCRITTURA DATI DEL PACCHETTO
            *********************************************/
            for(int i = 0; i < rlen-PacketHeaderLength; i++)
            {
                    fileout.write(packets.read());
            }
        }//fine if
        //Se non appartiene all'intervallo sposto il puntatore alla posizione del
        nuovo pacchetto
        else
        {
            packets.seek(pointer + rlen);
        }
        pointer= pointer+rlen;

    }   //fine ciclo while

} // Fine DAG 3.0 Ethernet

/***********************************
CHIUSURA DEL PROGRAMMA
***********************************/
int []tmp1 = new int [2];
tmp1= Header.returnToByte(MaxLength);
fileout.close();
packets.close();

/***********************************
AGGIUNGE IL CAMPO MAXLENGTH NELL'HEADER DI FILE
SE IL FORMATO HA SNAPSHOT LENGTH VARIABILE
***********************************/

if (MaxlengthVariabile == true)
{
    RandomAccessFile FileTemporaneo = new
        RandomAccessFile("tmp.txt","r");
    long lunghezza = FileTemporaneo.length();
    FileWriter finale = new FileWriter("tmp2.txt");
     for (int j = 0; j < 16; j++)
        finale.write(FileTemporaneo.read());
    finale.write(tmp1[1]);
    finale.write(tmp1[0]);
    FileTemporaneo.read();
    FileTemporaneo.read();
    for (int j = 0; j < lunghezza - 18; j++)
        finale.write(FileTemporaneo.read());
    finale.close();
                                FileTemporaneo.close();
}
else // MaxlengthVariabile == false
{
        RandomAccessFile FileTemporaneo = new
   RandomAccessFile("tmp.txt","r");
    long lunghezza = FileTemporaneo.length();
    FileWriter finale = new FileWriter("tmp2.txt");
    for (int j = 0; j < lunghezza; j++)
        finale.write(FileTemporaneo.read());
    finale.close();
    FileTemporaneo.close();
}

/***********************************
RINOMINA IL FILE FACENDO SCEGLIERE
 ALL'UTENTE NOME E POSIZIONE DEL SALVATAGGIO
***********************************/

out = new Salva();
File fout = new File("tmp2.txt");
File x = new File("tmp.txt");
x.delete();
fout.renameTo(out.getFile());
```

```
        System.exit(0);
    }
}

class Apri extends JPanel
{
  File file;
  int returnVal;
  JFileChooser fc;
  Apri()
  {
    fc = new JFileChooser();
    returnVal = fc.showOpenDialog(Apri.this);
    if(returnVal==JFileChooser.APPROVE_OPTION)
    {
        file = fc.getSelectedFile();
        System.out.println("File: "+file.toString());
    }
    else
    {
        System.exit(0);
    }
  }// Fine costruttore classe Apri.
  File getFile(){return file;}
}// Fine classe Apri.

class Salva extends JPanel
{
  File file;
  int returnVal;
  JFileChooser fc;
  Salva()
  {
    fc = new JFileChooser();
    returnVal = fc.showSaveDialog(Salva.this);
    if(returnVal==JFileChooser.APPROVE_OPTION)
    {
        file = fc.getSelectedFile();
        System.out.println("File: "+file.toString());
    }
    else
    {
        System.exit(0);
    }
  }// Fine costruttore classe Salva.
  File getFile(){return file;}
}// Fine classe Apri.
```

Bibliografia

[1] Documentazione sulla libreria WinPcap: http://www.winpcap.org/docs/

[2] Documentazione su Ethereal: http://www.ethereal.com/docs/

[3] Informazioni sul formato ERF: http://www.endace.com/

[4] Documentazione formato tcpdump: http://www.tcpdump.org/