

## INDICE

<i>pagina</i>	4	<b>1 – Introduzione</b>
	4	1.1 – Premessa
	4	1.2 – Obiettivo della tesi
	5	1.3 – Struttura della tesi
<i>pagina</i>	6	<b>2 – Struttura messaggi</b>
	8	2.1 – Scelta delle frame da catturare
	12	2.2 – Struttura dei pacchetti
<i>pagina</i>	20	<b>3 – TCP DUMP</b>
	20	3.1 – Breve introduzione al TCPDUMP
	20	3.2 – Instruction Set
	21	3.3 – TCPDUMP e Wireshark
<i>pagina</i>	23	<b>4 – La scelta Java</b>
	23	4.1 – Perché utilizzare JAVA
	24	4.2 – JPCAP: libreria di cattura dei pacchetti
	26	4.3 – JFREECHART: libreria per la creazione dei grafici
<i>pagina</i>	27	<b>5 – Ambienti di sviluppo</b>
	27	5.1 – Ubuntu 8.04
	29	5.2 – Java NetBeans 5.5
	30	5.3 – MySQL Server
<i>pagina</i>	34	<b>6 – Realizzazione dell'applicazione</b>
	34	6.1 – Fasi di realizzazione e eventuali problemi
	41	6.2 – Struttura DataBase
	45	6.3 – Vista delle classi
	48	6.4 – Analisi approfondita dei metodi principali
	63	6.5 – Screenshot dei vari form

<i>pagina</i>	<b>68</b>	<b>7 – Conclusioni</b>
	68	7.1 – Conclusioni
	69	7.2 – Bibliografia
	69	7.3 – Glossario termini

---



# 1 – Introduzione

---

## 1.1 **Premessa**

---

Il mio progetto di tesi nasce dal bisogno, da parte di tecnici ed amministratori di rete, di conoscere con esattezza tutti i dettagli del traffico di rete che devono controllare.

L'idea di creare questo progetto è stata una conseguenza logica del mio progetto formativo “stage”, nel quale mi era stato assegnato il compito di trovare e testare un programma open source che fosse in grado di analizzare i pacchetti di un file contenente i dati del traffico di rete.

La mia scelta era ricaduta in un applicativo chiamato TcpTrace, che era effettivamente in grado di riportare molti dettagli dei pacchetti catturati, ma aveva come contro il fatto che fosse molto lento nell'elaborazione dei dati.

Prendendo spunto da TcpTrace ho quindi deciso di creare un programma multi-piattaforma che fosse in grado di visualizzare quanti più possibili dettagli dei vari pacchetti, ma in un tempo relativamente minore.

## 1.2 **Obiettivo della tesi**

---

L'obiettivo della mia tesi è la realizzazione di *Sonicap*.

Sonicap è programma sviluppato in Java e MySQL, in grado di girare su più piattaforme (linux, windows, mac), e di leggere, nel minor tempo possibile, tutti i pacchetti contenuti in un file di traffico di rete (normalmente un file con estensione *.dump*) salvando il tutto in un

DataBase. Quest'ultima azione mi permette così di avere un accesso ai dati molto più veloce per poter creare delle statistiche personalizzate e grafici su tutti i file che inserirò nel DataBase. Ovviamente ogni qualvolta si vorrà effettuare una statistica descrittiva di un file dump, il risultato potrà essere salvato in un file HTML.

Per motivi di tempo è stato possibile sviluppare solo una versione beta (ovvero non ancora completa) di Sonicap, sono infatti assenti diversi controlli per il suo utilizzo, ma tutte le funzioni principali sono comunque eseguite.

### **1.3** **Struttura della tesi**

---

Per comprendere al meglio il mio lavoro, ho strutturato la tesi in modo che risulti comprensibile anche a coloro che non sono molto preparati riguardo la struttura delle reti o la programmazione ad oggetti.

Nelle prime sezioni verrà descritto, nel dettaglio, il traffico di rete, i pacchetti principali che vengono inviati al suo interno e la loro composizione. Successivamente parlerò di TcpDump, ovvero del programma che permette di catturare il traffico di rete e quindi di creare i file *.dump* sui quali il mio programma si basa, e continuerò con l'illustrare i vari ambienti di sviluppo e librerie utilizzati, giustificando il motivo del loro utilizzo. Seguirà una dettagliata descrizione del programma vero e proprio con in rilievo i passaggi più complessi e la vista generale del codice e dei metodi da me creati.

## 2 – *Struttura Messaggi*

---

Prima di conoscere la struttura delle frame catturate, diamo una piccola descrizione di cosa è il protocollo IP e come è strutturato.

L'IP (*Internet Protocol*) è un protocollo situato al livello di rete nello stack TCP/IP. Tale protocollo si occupa quindi di inserire negli header di un pacchetto proveniente dai livelli superiori gli *indirizzi logici* del mittente e del destinatario. Ogni host in una rete è quindi identificato a livello logico (*non* a livello fisico) da un indirizzo IP, ovvero un indirizzo logico univoco che in un certo momento identifica in modo non ambiguo qualsiasi host collegato in rete.

L'indirizzo IP ha una *dimensione fissa*. Lo standard utilizzato da decenni ormai è *IPv4* il quale prevede che l'indirizzo sia rappresentato a *32 bit* (4 byte). Per comodità l'IP viene generalmente rappresentato con la *dotted notation*, ovvero una notazione del tipo *x.x.x.x*. È una notazione in cui l'indirizzo a 32 bit viene 'splittato' in 4 parti. Poiché ogni parte rappresenta 8 bit, con ognuna di esse è possibile rappresentare tutti i numeri nell'intervallo esadecimale 0...FF, che in decimale identifica l'intervallo 0...255. Questa notazione permette un range di 4,3 miliardi di indirizzi che purtroppo al giorno d'oggi cominciano a scarseggiare, per questo motivo si sta introducendo un nuovo standard, l'*Ipv6*, che utilizza 128 bit anziché 32.

La cattura di pacchetti con standard *Ipv6* è ancora abbastanza complessa, per questo motivo il mio programma si limiterà all'analisi dei pacchetti con indirizzo in formato *Ipv4*.

Finita la premessa sul protocollo IP, andiamo a vedere come è strutturato un pacchetto IP:



VERSION	IHL	TYPE OF SERVICE	TOTAL LENGTH	
IDENTIFICATION		FLAGS	FRAGMENT OFFSET	
TIME TO LIVE	PROTOCOL	HEADER CHECKSUM		
SOURCE ADDRESS				
DESTINATION ADDRESS				
OPTIONS + PADDING				
DATI				

**Figura 2.1 - Header IP**

I campi dell'header IP sono i seguenti:

- VERSION (4 bit) – nel caso in cui si tratti di un pacchetto Ipv4
- TYPE OF SERVICE (8 bit) – lunghezza dell'header IP
- TOTAL LENGTH (16 bit) – lunghezza dell'header + lunghezza della parte dati
- IDENTIFICATION (16 bit) – identificativo del pacchetto
- FLAGS (3 bit) – usati per l'eventuale frammentazione del pacchetto
- FRAGMENT OFFSET (13 bit) – nel caso di frammentazione
- TIME TO LIVE (8 bit) – numero massimo di passaggi che il pacchetto dovrà fare attraverso dei commutatori di pacchetto prima di diventare non più valido
- PROTOCOL (8 bit) – protocollo di trasporto da usare
- CHECKSUM (16 bit) – controllo dell'errore su eventuali bit corrotti nel pacchetto
- SOURCE ADDRESS (32 bit) – indirizzo IP sorgente
- DESTINATION ADDRESS (32 bit) – indirizzo IP destinazione
- OPTION + PADDING (32 bit) – eventuali opzioni + riempimento
- DATI (lunghezza variabile) – contenuto effettivo del pacchetto

## 2.1 Scelta delle frame da catturare

---

TcpDump riesce a catturare, e quindi salvare su file, pacchetti di tipo: TCP, UDP, ICMP, IGMP e ARP, sarà perciò su questi formati che andrò ad effettuare le mie statistiche.

Vediamo nel dettaglio cosa indicano questi cinque tipi di protocollo.

### 2.1.1 Protocollo TCP

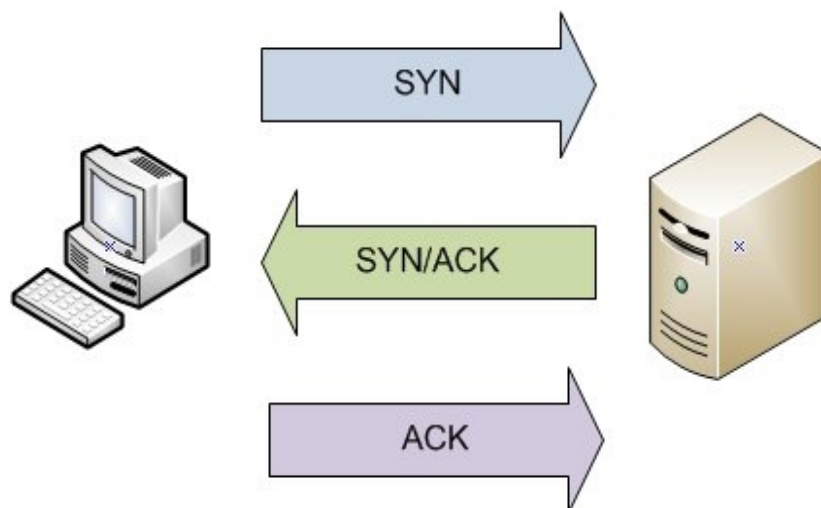
---

TCP (*Transmission Control Protocol*) è il principale protocollo di trasporto affidabile utilizzato nel protocollo TCP/IP, e quindi uno dei fondamenti di internet stessa. A differenza dell'UDP che è un protocollo di trasporto non affidabile e fondato sulla filosofia del *best effort* per far giungere un pacchetto a destinazione, TCP è un protocollo *orientato alla connessione*; ciò vuol dire che il TCP cerca di riprodurre in un ambiente tipicamente *packet switching* come quello di internet, e quindi senza canali di connessione dedicati tra gli host che comunicano. Ciò è possibile instaurando delle connessioni virtuali tra i due host, il che dà garanzie maggiori circa il recapito effettivo dei pacchetti, e queste connessioni virtuali si instaurano attraverso una precisa sequenza di pacchetti scambiati tra gli host, sequenze richieste dal protocollo stesso.

Nel modello TCP, affinché si instauri una connessione è necessario che su un host ci sia un processo server in ascolto e che accetti connessioni dall'esterno, e un processo client in esecuzione su un altro host che faccia richiesta di connessione al server. Per instaurare la connessione il processo client deve ovviamente conoscere indirizzo IP e la porta su cui il server è in ascolto. A questo punto invia al server un pacchetto speciale detto *SYN segment*, un pacchetto avente il flag SYN (*synchronize*) dell'header TCP settato a 1. Il pacchetto SYN inviato dal client include l'*ISN (Initial Sequence Number)*, che è un numero a 32 bit generato dal client e che servirà a numerare i pacchetti. A un



tale pacchetto il server, se accetta la connessione, risponde con un altro SYN segment ed infine il client risponde con un ultimo segmento TCP, con SYN=0, ACK pari all'ISN del server+1 e numero di sequenza pari a client\_ISN+1. Questo dà il via allo scambio di dati vero e proprio. (*3 ways handshaking*).



**Figura 2.2 – 3 ways handshaking**

---

Per chiudere una connessione TCP esistono 2 modi: il polite close ed il reset della connessione.

#### *Polite Close*

Chiusura pulita della connessione che si svolge nel seguente modo:

- il client invia al server un segmento TCP con flag FIN settato a 1
- il server invia al client un segmento TCP con ACK per confermare la ricezione del pacchetto FIN
- il server chiude tutte le risorse di memoria dedicate alla connessione con il client e gli invia un pacchetto con FIN settato a 1
- il client invia al server l'ACK del pacchetto ricevuto e rimane in TIME\_WAIT di eventuali pacchetti rimasti in sospeso prima di chiudere definitivamente la connessione

#### *Reset della connessione*

In genere le connessioni TCP vengono chiuse in modo pulito tramite lo scambio di pacchetti FIN visto sopra. Tuttavia, TCP prevede anche una chiusura brusca della connessione (*reset*), generalmente dovuta a un'interruzione da parte dell'utente, a un'interruzione da tastiera o simili. In questo caso uno dei due host invia all'altro un segmento TCP con il flag RESET=1. Quando l'altro capo riceve questo pacchetto chiude immediatamente la connessione rilasciando tutte le risorse occupate.

### 2.1.2 Protocollo UDP

---

A differenza del TCP, non gestisce il riordinamento dei pacchetti né la ritrasmissione di quelli persi. L'UDP ha come caratteristica di essere un protocollo di rete inaffidabile ma in compenso molto rapido ed efficiente per le applicazioni leggere o time-sensitive. Infatti, è usato spesso per la trasmissione di informazioni audio o video come videoconferenze. Dato che le applicazioni in tempo reale spesso richiedono un ritmo minimo di spedizione, non vogliono ritardare eccessivamente la trasmissione dei pacchetti e possono tollerare qualche perdita di dati, il modello TCP può quindi non essere particolarmente adatto a queste esigenze. L'UDP fornisce soltanto i servizi basilari del livello di trasporto: la moltiplicazione delle connessioni e il controllo degli errori tramite checksum. Un server dedicato ad una particolare applicazione che sceglie l'UDP come protocollo di trasporto può supportare molti più client attivi.

### 2.1.3 Protocollo ICMP

---

L'ICMP (*Internet Control Message Protocol*) è un protocollo di servizio che si preoccupa di trasmettere informazioni riguardanti malfunzionamenti, informazioni di controllo o messaggi tra i vari componenti di una rete di calcolatori.

L'ICMP viene incapsulato all'interno dell'IP, non è quindi garantita la consegna a destinazione dei pacchetti. Solitamente viene usato in programmi di testing di rete come: *ping* o *traceroute*.

---

#### 2.1.4 **Protocollo IGMP**

---

L'*Internet Group Management Protocol* è un protocollo per la gestione dei gruppi multicast. Costituisce il mezzo per un host di informare il router, ad esso collegato, che un'applicazione che gira nell'host vuole unirsi ad uno specifico gruppo multicast. L'IGMP opera fra un host e il router ad esso collegato direttamente, per coordinare i router multicast invece è richiesto un altro protocollo, così che i datagrammi multicast possano essere instradati alle loro destinazioni finali.

---

#### 2.1.5 **Protocollo ARP**

---

Il protocollo ARP (*Address Resolution Protocol*) fornisce la "mappatura" tra l'indirizzo IP a 32bit (4byte) di un calcolatore e il suo MAC Address l'indirizzo fisico a 48bit (6 byte).

Questo protocollo di servizio viene utilizzato in una rete di calcolatori con protocollo IP con supporto al servizio di broadcast. Per inviare un pacchetto IP ad un calcolatore della stessa sottorete, è necessario incapsularlo in un pacchetto di livello DataLink, che dovrà avere come indirizzo di destinazione il MAC address del calcolatore a cui lo si vuole inviare. ARP viene utilizzato per ottenere questo indirizzo.

Vediamo il suo funzionamento. L'host che vuole conoscere il MAC address di un altro host, di cui conosce l'indirizzo IP, invia in broadcast una richiesta ARP (ARP-request), generata dal protocollo IP, contenente l'indirizzo IP dell'host di destinazione ed il proprio indirizzo MAC.

Tutti i calcolatori della sottorete ricevono la richiesta. In ciascuno di essi il protocollo ARP verifica se viene richiesto il proprio indirizzo IP. L'host di destinazione che riconoscerà il proprio IP nel pacchetto di

ARP-request, provvederà ad inviare una risposta (ARP-reply) in unicast all'indirizzo MAC sorgente, contenente il proprio MAC. In questo modo ogni host può scoprire l'indirizzo fisico di tutti gli altri nella stessa sottorete.

## 2.2 **Struttura dei pacchetti**

Ora che conosciamo la funzione di tutti i protocolli utilizzati, andiamo a vedere come sono strutturati i loro pacchetti.

### 2.2.1 **Pacchetto TCP**

Un *segmento* TCP (unità d'informazione scambiata) viene incapsulato in un pacchetto IP che sarà costituito da un Header TCP, contenente tutte le informazioni necessarie, ed un *payload*, cioè il carico utile di dati. La sua struttura avrà quindi la seguente forma:

Bit 0 - 3	Bit 4 - 9	Bit 10 - 15	Bit 16 - 31
Source Port		Destination Port	
Sequence Number			
Acknowledgment Number			
Header Length	Reserved	Flags	Advertise Window
Checksum		Urgent Pointer	
Options			
Data			

**Figura 2.3 – Pacchetto TCP**

- *Source Port [16 bit]* – identifica il numero di porta sull'host mittente associato alla connessione TCP
- *Destination Port [16 bit]* – identifica il numero di porta sull'host

- destinatario associato alla connessione TCP
- *Sequence Number [32 bit]* – numero di sequenza, indica lo scostamento (espresso in byte) dell'inizio del segmento TCP all'interno del flusso completo, a partire dall' *Initial Sequence Number (ISN)*, negoziato all'apertura della connessione
  - *Acknowledgement Number [32 bit]* – Numero di riscontro, ha significato solo se il flag ACK è settato a 1, e conferma la ricezione di una parte del flusso di dati nella direzione opposta, indicando il valore del prossimo Sequence number che l'host mittente del segmento TCP si aspetta di ricevere
  - *Header Length [4 bit]* – Indica la lunghezza dell'header del segmento TCP; tale lunghezza può variare da 20 byte a 60 byte a seconda della presenza e della lunghezza del campo facoltativo *Options*
  - *Reserved [6 bit]* – Bit non utilizzati e predisposti per sviluppi futuri del protocollo
  - *Flags [6 bit]* – Bit utilizzati per il controllo del protocollo:
    - *URG* - se settato a 1 indica che nel flusso sono presenti dati urgenti alla posizione (offset) indicata dal campo Urgent pointer;
    - *ACK* - se settato a 1 indica che il campo Acknowledgment number è valido;
    - *PSH* - se settato a 1 indica che i dati in arrivo non devono essere bufferizzati ma passati subito ai livelli superiori dell'applicazione;
    - *RST* - se settato a 1 ripristina la connessione; viene utilizzato in caso di grave errore;
    - *SYN* - se settato a 1 indica che l'host mittente del segmento vuole aprire una connessione TCP con l'host destinatario e specifica nel campo Sequence number il valore dell' Initial Sequence Number (ISN); ha lo scopo di sincronizzare i numeri di sequenza dei due host. L'host che ha inviato il SYN deve attendere dall'host remoto un

pacchetto SYN/ACK;

- *FIN* - se settato a 1 indica che l'host mittente del segmento vuole chiudere la connessione TCP aperta con l'host destinatario. Il mittente attende la conferma dal ricevente (con un FIN-ACK). A questo punto la connessione è ritenuta chiusa per metà: l'host che ha inviato FIN non potrà più inviare dati, mentre l'altro host ha il canale di comunicazione ancora disponibile. Quando anche l'altro host invierà il pacchetto con FIN impostato la connessione, dopo il relativo FIN-ACK, sarà considerata completamente chiusa
- *Advertise Window [16 bit]* – indica la dimensione della finestra di ricezione dell'host mittente, cioè il numero di byte che il mittente è in grado di accettare a partire da quello specificato dall'acknowledgment number
- *Checksum [16 bit]* – Campo di controllo utilizzato per la verifica della validità del segmento.
- *Urgent Pointer [16 bit]* – Puntatore a dato urgente, ha significato solo se il flag URG è settato a 1 ed indica lo scostamento in byte a partire dal Sequence number del byte di dati urgenti all'interno del flusso
- *Options* – Opzioni facoltative per usi avanzati del protocollo

### 2.2.2 Pacchetto UDP

La struttura del datagramma UDP è la seguente:

Bit 0 - 15	Bit 16 - 31
Source Port	Destination Port
Length	Checksum
Data	

**Figura 2.4 - Datagramma UDP**

- *Source Port [16 bit]* – identifica il numero di porta sull'host del mittente del datagramma
- *Destinazion Port [16 bit]* – identifica il numero di porta sull'host del destinatario del datagramma
- *Checksum [16 bit]* – contiene il codice di controllo del datagramma
- *Data* – contiene i dati del datagramma

### 2.2.3 Pacchetto ICMP

Il formato del messaggio ICMP è:

Bit 0 - 7	Bit 8 - 15	Bit 16 - 31
Type	Code	Checksum
Identifier		Sequence Number
Data		

**Figura 2.5 – Messaggio ICMP**

- *Type [8 bit]* – specifica il formato del messaggio ICMP, che può assumere i seguenti valori:
  - 0 Echo reply
  - 1 Non assegnato
  - 2 Non assegnato
  - 3 Destinazione irraggiungibile
  - 4 Source quench
  - 5 Redirect
  - 6 Indirizzo host alternativo
  - 7 Non assegnato
  - 8 Echo request
  - 9 Router advertisement
  - 10 Router selection
  - 11 Richiesta scaduta
  - 12 Errore nei parameteri

- 13 Timestamp request
  - 14 Timestamp reply
  - 15 Information Request
  - 16 Information Reply
  - 17 Richiesta address mask
  - 18 Risposta address mask
  - 19 Riservato (per sicurezza)
  - 20-29 Riservati (per test di robustezza)
  - 30 Traceroute
  - 31 Errore di conversione datagramma
  - 32 Redirect su host mobile
  - 33 Ipv6 Where-Are-You
  - 34 Ipv6 I-Am-Here
  - 35 Mobile Registration Request
  - 36 Mobile Registration Reply
  - 37 Domain Name request
  - 38 Domain Name reply
  - 37-255 Non assegnati
- *Code [8 bit]* – ulteriore qualificazione del messaggio
  - *Checksum [16 bit]* – contiene il codice di controllo del messaggio
  - *Data* – contiene i dati del messaggio

#### 2.2.4 Pacchetto IGMP

Il formato di un messaggio IGMP è:

Bit 0 - 7	Bit 8 - 15	Bit 16 - 31
Type	Code	IGMP Checksum
Identifier		
Group Address		
Access Key		

**Figura 2.6 – Messaggio IGMP**



- *Type [8 bit]* – indica la richiesta o la risposta del gruppo
  1. Create group request
  2. Create group reply
  3. Join group request
  4. Join group reply
  5. Leave group request
  6. Leave group reply
  7. Confirm group request
  8. Confirm group reply
- *Code [8 bit]* – indica se il nuovo gruppo host è pubblico o privato
- *IGMP Checksum [16 bit]* – controllo dell'errore
- *Identifier [32 bit]* – distingue le diverse richieste da lo stesso host
- *Group Address [32 bit]* – contiene il gruppo host della richiesta
- *Access Key [32 bit]* – indica la chiave di accesso assegnata al gruppo di host definito nel campo Group Address

### 2.2.5 Pacchetto ARP

La struttura di una richiesta ARP è così composta:

Bit 0 - 7	Bit 8 - 15	Bit 16 - 31
Hardware Type		Protocol Type
Hardware Address Length	Protocol Address Length	Opcode
Source Hardware Address		
Source Protocol Address		
Destination Hardware Address		
Destination Protocol Address		
Data		

**Figura 2.7 – Richiesta ARP**

- *Hardware Type [16 bit]* – indica il tipo di Hardware
  - 1 Ethernet.

- 2 Experimental Ethernet.
  - 3 Amateur Radio AX.25.
  - 4 Proteon ProNET Token Ring.
  - 5 Chaos.
  - 6 IEEE 802.
  - 7 ARCNET.
  - 8 Hyperchannel.
  - 9 Lanstar.
  - 10 Autonet Short Address.
  - 11 LocalTalk.
  - 12 LocalNet (IBM PCNet or SYTEK LocalNET).
  - 13 Ultra link.
  - 14 SMDS.
  - 15 Frame Relay.
  - 16 ATM, Asynchronous Transmission Mode.
  - 17 HDLC.
  - 18 Fibre Channel.
  - 19 ATM, Asynchronous Transmission Mode.
  - 20 Serial Line.
  - 21 ATM, Asynchronous Transmission Mode.
  - 22 MIL-STD-188-220.
  - 23 Metricom.
  - 24 IEEE 1394.1995.
  - 25 MAPOS.
  - 26 Twinaxial.
  - 27 EUI-64.
  - 28 HIPARP.
  - 29 IP and ARP over ISO 7816-3.
  - 30 ARPSec.
  - 31 IPsec tunnel.
  - 32 Infiniband.
  - 33 CAI, TIA-102 Project 25 Common Air Interface.
- *Protocol Type [16 bit]* – normalmente è un IP

- *Hardware Address Length [8 bit]* – lunghezza dell'indirizzo hardware in bytes
- *Protocol Address Length [8 bit]* – lunghezza dell'indirizzo del protocollo in bytes
- *Opcode [16 bit]* – codice dell'operazione
- *Source Hardware Address [32 bit]* – lunghezza variabile
- *Source Protocol Address [32 bit]* – lunghezza variabile
- *Destination Hardware Address [32 bit]* – lunghezza variabile
- *Destination Protocol Address [32 bit]* – lunghezza variabile
- *Data* – dati

# 3 – TcpDump

---

## 3.1 Breve introduzione al Tcpdump

---

Tcpdump è uno strumento per l'analisi del traffico che avviene nella rete fisica a cui si è collegati.

Capiamo bene che l'analisi del traffico della rete, sia per mezzo dell'intercettazione di tutti i pacchetti che attraversano una rete fisica, sia per mezzo del controllo che riguarda esclusivamente una singola interfaccia di rete del nodo locale, è molto importante per comprendere i problemi legati alla sicurezza e per scoprire inconvenienti di vario genere.

Tcpdump si può quindi tradurre come un *tool* di controllo della rete a cui si è collegati.

## 3.2 Instruction Set

---

Il set di operazioni che Tcpdump mette a disposizione è veramente vasto e complesso da spiegare, per questo motivo rivelerò solo le funzioni più significative, e maggiormente utilizzate, per la cattura dei pacchetti:

-a	Permette di convertire indirizzi broadcast o IP in nomi
-c	Esce dopo aver ricevuto un tot di pacchetti
-C	Setta la grandezza del file salvato con il flag -w
-e	Stampa il MAC Address di ogni indirizzo catturato
-F	Permette di fornire i filtri da un file
-i	Definisce l'interfaccia di rete
-n	Non converte l'indirizzo IP in nome

-p	L'interfaccia locale non viene settata in modalità promiscua
-q	Visualizza il minor numero di informazioni possibili
-r	Utilizza il file specificato come input per i dati da filtrare
-s	Indica la quantità in bytes di un pacchetto catturato
-t	Non stampa il contrassegno temporale per ogni pacchetto
-tt	Stampa il contrassegno temporale per ogni pacchetto
-v	Incrementa il numero delle informazioni
-w	Scrive su file il risultato dello sniffing
-x	Visualizza in HEX
-X	Stampa i pacchetti in formato HEX e ASCII

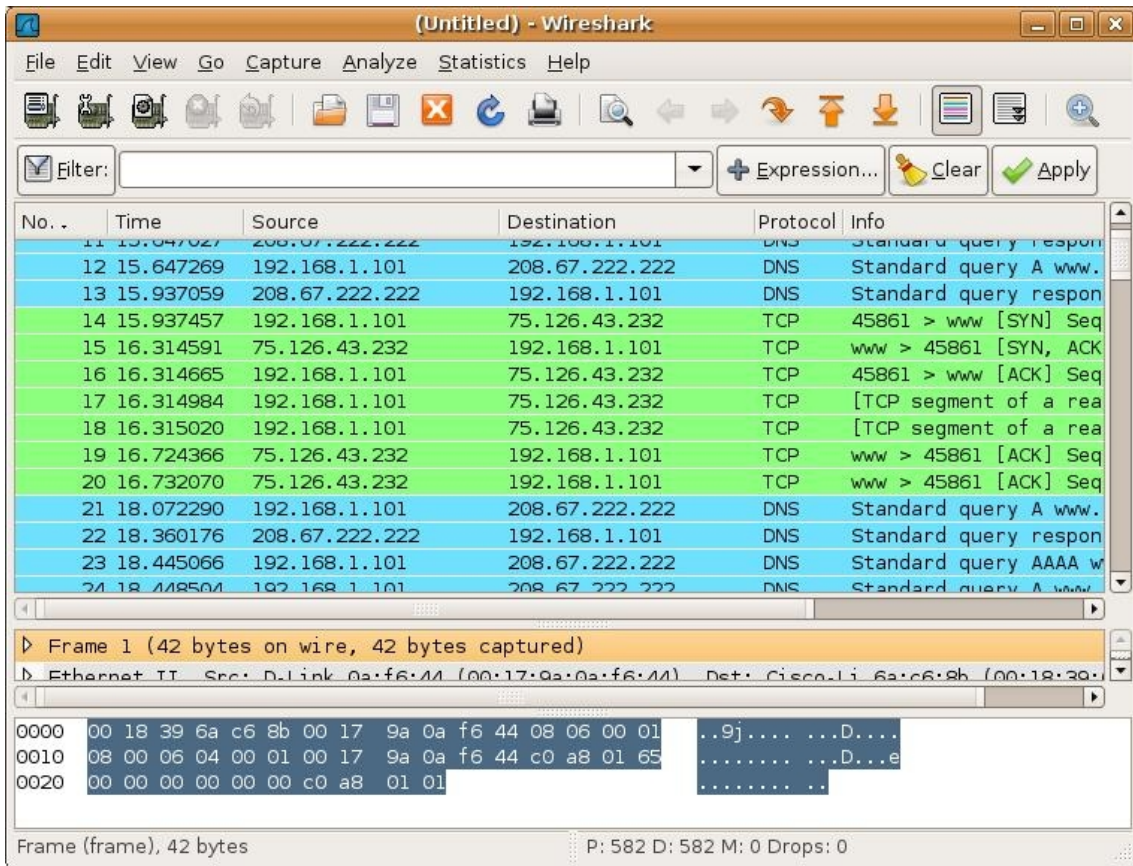
### 3.3 **Tcpdump e WireShark**

Un altro programma che ha il compito di analizzare il traffico di rete catturato è Wireshark.

Wireshark (precedentemente chiamato *Ethereal*) è un software per analisi di protocollo, o packet sniffer utilizzato per la soluzione di problemi di rete.

Le funzionalità di Wireshark sono molto simili a quelle di tcpdump, ma con un'interfaccia grafica, e maggiori funzionalità di ordinamento e filtraggio. Permette all'utente di osservare tutto il traffico presente sulla rete utilizzando la modalità promiscua dell'adattatore di rete. Tipicamente si riferisce alle reti Ethernet, ma è possibile analizzare altri tipi di rete fisica.

Per la cattura dei pacchetti Wireshark non dispone di proprio codice, ma utilizza libpcap/WinPcap, quindi può funzionare solo su reti supportate da libpcap o WinPcap.



**Figura 3.1 - Wireshark**

# 4 – La scelta Java

---

## 4.1 Perchè utilizzare Java

---

Durante la fase di progettazione, pensai che il linguaggio di programmazione ideale per la creazione di Sonicap avrebbe dovuto soddisfare i seguenti requisiti:

- interfacciamento con una banca dati del tipo DataBase MySQL
- possibilità di creare un'interfaccia grafica
- programmazione orientata ad oggetti
- indipendenza dalla piattaforma su cui viene lanciato

Il *Java* (della *Sun Microsystems*) era il linguaggio di programmazione che più rispecchiava le mie esigenze.

L'interfacciamento con un DataBase MySQL è molto semplice, si tratta semplicemente di creare una connessione tra i due e le query per la gestione dei dati.

Con l'ambiente di sviluppo Netbeans 5.5 (o altri come Eclipse) è possibile creare un'interfaccia grafica senza dover scrivere a mano il codice, si può semplicemente disegnare con il mouse ed i componenti grafici della libreria *swing*.

L'orientamento agli oggetti, si riferisce a un moderno metodo di programmazione e progettazione, la *programmazione orientata agli oggetti* (*Objects Oriented Programming*).

L'idea alla base della *OOP* è di rappresentare, nella progettazione del software, le entità reali o astratte che compongono il problema sotto forma di oggetti. Gli oggetti sono caratterizzati da delle proprietà e da metodi applicabili sugli oggetti stessi, che possono ad esempio modificarne lo stato o estrarne informazioni.

I programmi scritti in Java possono essere unicamente orientati agli oggetti, di conseguenza tutto il codice deve essere necessariamente incluso in una classe. Sebbene Java possa operare sia su oggetti che

su tipi di dati primitivi, è considerato un linguaggio ad oggetti puro, ovvero nel quale gli oggetti sono le entità di base del linguaggio.

L'indipendenza dalla piattaforma, significa che l'esecuzione di programmi scritti in Java deve avere un comportamento simile su hardware diverso. Si dovrebbe essere in grado di scrivere il programma una volta e farlo eseguire dovunque. Questo è possibile con la compilazione del codice di Java in un linguaggio intermedio *bytecode*, basato su istruzioni semplificate che ricalcano il linguaggio macchina. Il *bytecode* verrà quindi eseguito da una macchina virtuale. Inoltre, vengono fornite librerie standardizzate per permettere l'accesso alle caratteristiche della macchina in modo unificato. Il linguaggio Java include anche il supporto per i programmi con *multithread*, necessario per molte applicazioni che usano la rete.

## 4.2 **JPCAP: libreria di cattura dei pacchetti**

Il package *Jpcap* si avvale di una interfaccia e di una serie di classi per la cattura, l'analisi e la manipolazione di pacchetti di rete.

Tra i metodi più importanti che la libreria mette a disposizione abbiamo:

<i>getDeviceDescription()</i>	Ritorna la descrizione delle interfacce di rete
<i>getDeviceList()</i>	Ritorna gli identificativi delle interfacce di rete che possono essere usate per la cattura dei pacchetti
<i>lookupDevice()</i>	Ritorna il nome dell'interfaccia principale
<i>openDevice()</i>	Inizializza l'interfaccia e ritorna l'istanza di questa classe
<i>openFile()</i>	Apri un <i>dump</i> file creato con un tcpdump o Ethereal, e ritorna un'istanza della classe
<i>getPacket()</i>	Ritorna un pacchetto catturato
<i>processPacket()</i>	Cattura un numero specifico di pacchetti
<i>SetFilter()</i>	Setta un filtro



<i>close()</i>	Chiude l'interfaccia aperta
<i>getErrorMessage()</i>	Ritorna un messaggio di errore

**Figura 4.1 - Classe JPCAP**

---

Un'altra classe importante legata strettamente a quella precedentemente descritta è la classe *Packet*, tale classe contiene i campi relativi ai pacchetti catturati

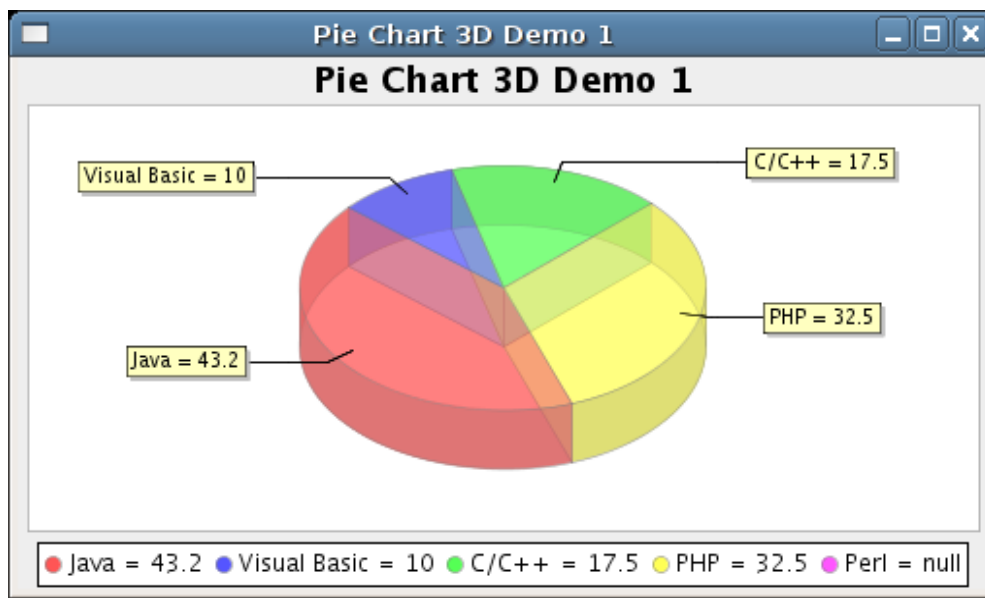
<i>data</i>	Rappresenta il campo dati del pacchetto
<i>datalink</i>	E' il Datalink Layer
<i>header</i>	Rappresenta il campo Header
<i>len</i>	E' la lunghezza del pacchetto catturato

**Figura 4.2 - Classe Packet**

---

### 4.3 JFREECHART - libreria per la creazione di grafici

JFREECHART è una libreria JAVA libera per la creazione semplificata di grafici di vario tipo, che mette a disposizione una vasta quantità di classi già pronte da modificare a piacimento per adattarle alle proprie esigenze.



**Figura 4.1 - Grafico Torta 3D**

La creazione di un grafico è principalmente strutturata in tre parti: la creazione del set di dati (*dataset*), la creazione del grafico associato al set di dati e l'inserimento del tutto in un pannello.

Avendo la medesima struttura, la creazione di un nuovo tipo di grafico risulta molto semplificata e proprio in questo sta il punto di forza della libreria JFREECHART.

Questo argomento sarà comunque meglio approfondito nella sezione 6.4.6 relativa alla creazione di grafici.

# 5 – Ambienti di sviluppo

---

## 5.1 **Ubuntu 8.04**

---

*Ubuntu* è un sistema operativo GNU/LINUX nato nel 2004, basato su Debian, che si focalizza sull'utente e sulla semplicità di utilizzo. È orientato all'utilizzo desktop e pone una grande attenzione al supporto hardware rilasciando una nuova versione ogni sei mesi. Ubuntu è una distribuzione Linux basata sull'ambiente desktop Gnome. È progettata per fornire un'interfaccia semplice, intuitiva e allo stesso tempo completa e potente. I punti di forza di questa distribuzione sono l'estrema semplicità di utilizzo, l'ottimo riconoscimento e supporto dell'hardware, il vasto parco software costantemente aggiornato e una serie di strumenti di gestione grafici che la rendono improntata verso l'ambiente desktop. Ne esistono 2 versioni: quella *desktop* e quella *server*. La versione *desktop* è stata realizzata per rispondere alle più frequenti necessità di un utente medio, quali navigazione in internet gestione dei documenti e delle immagini, svago e comunicazione, mentre la versione *server* si gestisce a riga di comando e serve appunto per la gestione avanzata di server e macchine in rete. Ubuntu supporta le piattaforme *x86*, *AMD64* e limitatamente alla *Server Edition*, le *Ultra Spark*, mentre il supporto ufficiale della piattaforma *Power PC* è stato interrotto a partire dalla versione 7.04, e affidato al supporto volontario della *community*.



**Figura 5.1 – Ubuntu 8.04 screenshot**

### **Requisiti minimi per la versione desktop**

- *processore 300 Mhz x86*
- *almeno 2 GB di spazio libero su disco fisso per installazione e partizione di swap*
- *scheda video VGA con risoluzione minima pari a 640x480 pixel*
- *lettore CD-ROM*
- *connessione Internet*
- *scheda audio*

### **Requisiti minimi per la versione server**

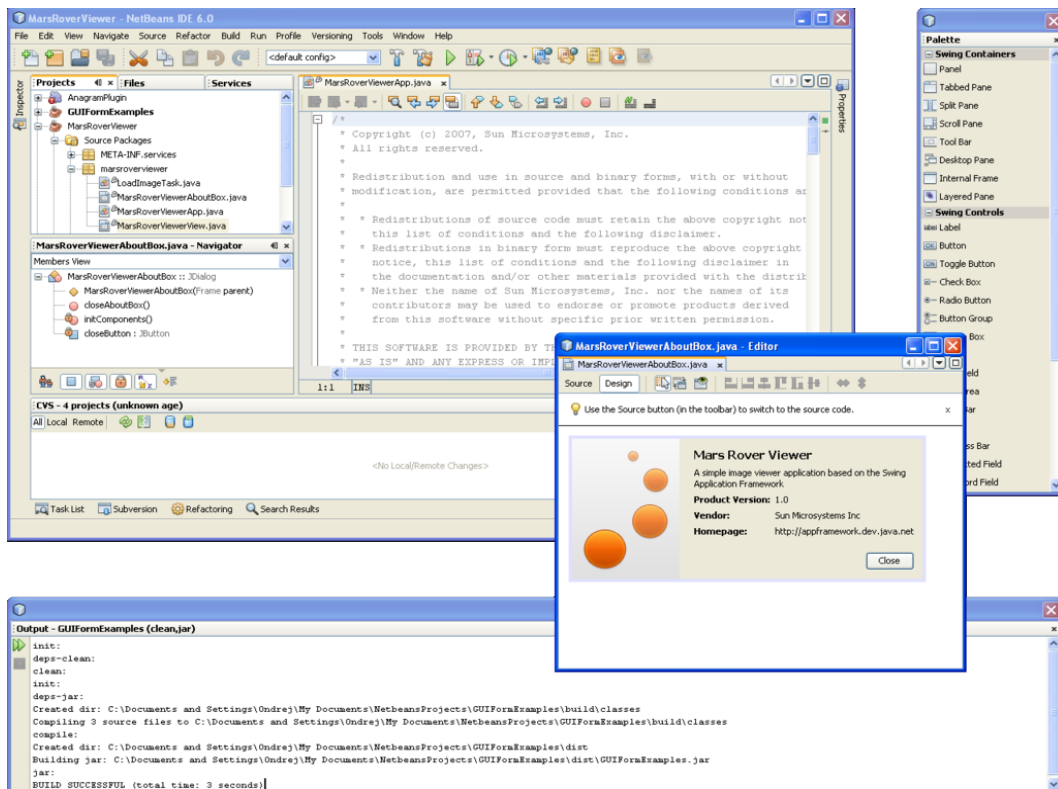
- *264 MB di RAM per il sistema*
- *50 MB di RAM per ogni client connesso*
- *3 GB di spazio libero su disco dedicati al sistema*
- *2 schede di rete Ethernet*
- *un server DHCP*
- *scheda video VGA con risoluzione minima pari a 640x480 pixel*
- *lettore CD-ROM*

NetBeans è un progetto open source di successo con un'ampia base di utenti, una comunità in crescita, e oltre 100 partners in tutto il mondo. Sun Microsystems ha fondato il NetBeans open source project nel Giugno del 2000 e continua ad essere lo sponsor principale del progetto.

A tutt'oggi esistono due prodotti: NetBeans IDE e NetBeans Platform. *NetBeans IDE* è un ambiente di sviluppo – uno strumento destinato ai programmatori per scrivere, compilare ed eseguire il debug ed il deploy di programmi. E' scritto in Java ma può supportare qualsiasi linguaggio di programmazione. Esiste anche un gran numero di moduli utili per estendere le funzionalità di NetBeans IDE. NetBeans IDE è un prodotto gratuito senza alcuna restrizione riguardante il suo uso.

E' anche disponibile NetBeans Platform; una piattaforma modulare ed estensibile da usare come supporto per la creazione di grosse applicazioni desktop. I partner ISV offrono ulteriori plug-in facilmente integrabili nella piattaforma e che possono anche essere utilizzati per lo sviluppo dei propri strumenti e soluzioni software.

Per il mio progetto mi sono servito di NetBeans IDE versione 5.5, nonostante vi sia attualmente la migliorata 6.1. Il motivo è molto semplice: la versione 6.1 utilizza come componenti del *look and feel* dell'interfaccia grafica, i componenti desktop del sistema su cui gira il programma; nel mio caso (utilizzando ubuntu 8.04) avrebbe caricato i componenti di *GNOME/GTK* che però sono molto più lenti di quelli dell'interfaccia standard di java presenti nella versione 5.5. La mia scelta è data quindi dalla maggiore velocità di sviluppo che la 5.5 mi offre facendomi utilizzare di default i componenti della libreria *swing* di java.



**Figura 5.2 – NetBeans IDE**

### 5.3 MySQL Server

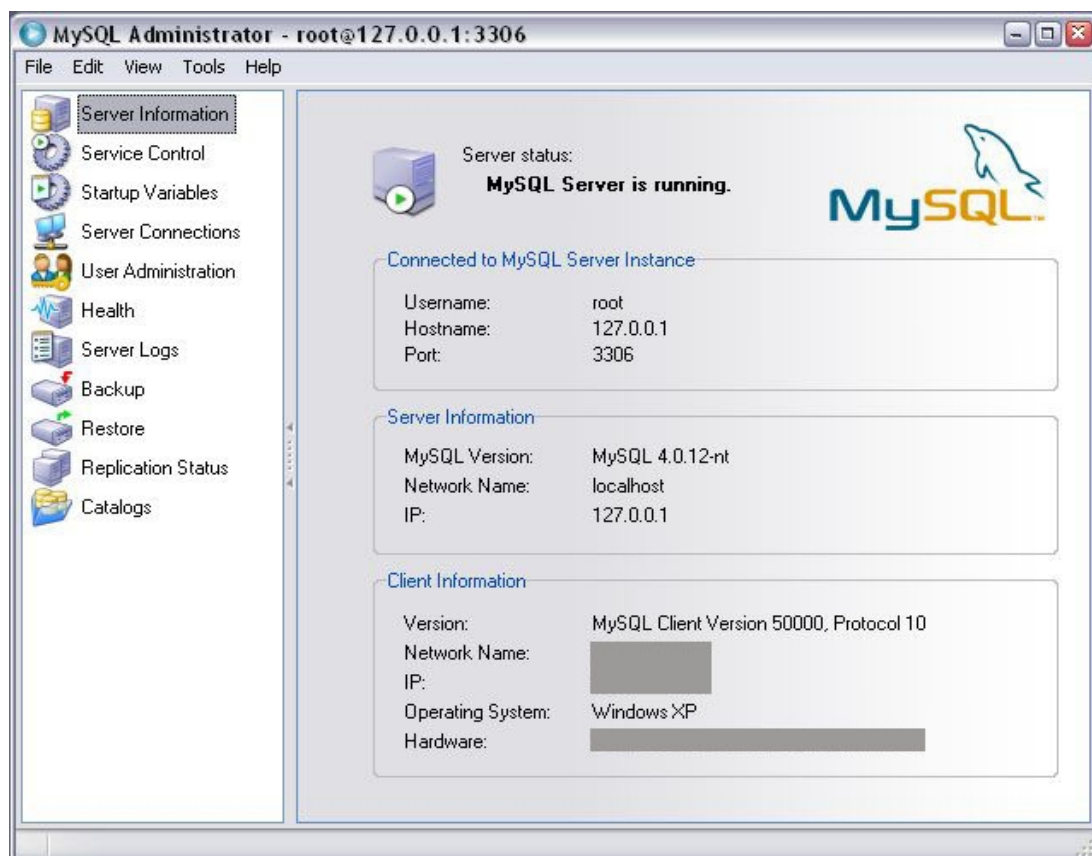
MySQL è un *database management system (DBMS)* relazionale, composto da un client con interfaccia a caratteri e un server, entrambi disponibili sia per sistemi Unix che per Windows, anche se prevale un suo utilizzo in ambito Unix.

Per utilizzare al meglio il linguaggio SQL, vengono forniti 2 programmi con MySQL server che gestiscono il database e le relative connessioni: *MySQL Administrator* e *MySQL Query Browser*.

### 5.3.1 MySQL Administrator

MySQL Administrator è un programma che permette operazioni amministrative, come la configurazione del server MYSQL, il controllo delle sue performance e tutte le funzioni per la gestione del database, il tutto in ambito visuale. Ovviamente c'è anche la possibilità di interagire tramite riga di comando, ma si perderebbe la velocità e la facilità dell'utilizzo dell'interfaccia grafica, la quale permette anche ad un utente non molto esperto di SQL, di essere in grado di creare e gestire il suo server.

Le funzionalità di cui faccio maggiore uso per la realizzazione del mio programma sono: la creazione del database, la gestione dei permessi su di esso e la creazione di una connessione al server e quindi al DB stesso.



**Figura 5.3 – MySQL Administrator**

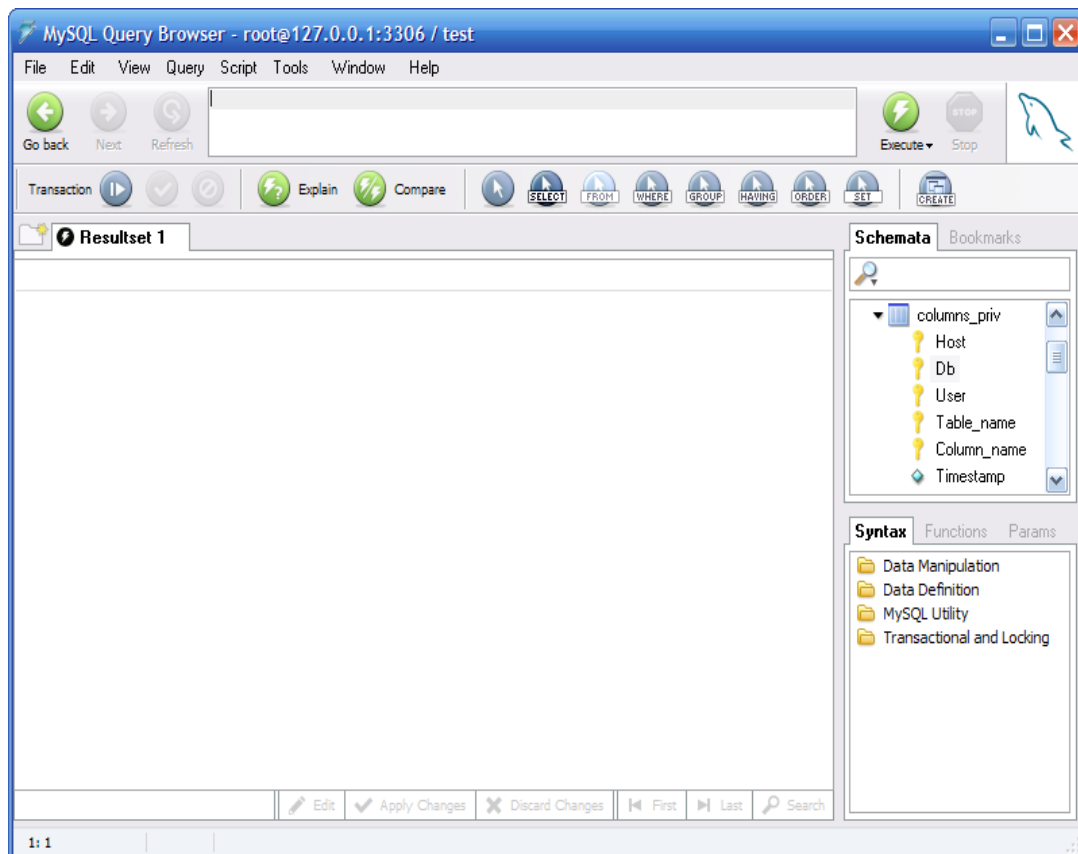
*MySQL Query Browser* è uno strumento grafico fornito da MySQL per creare, eseguire ed ottimizzare query in ambiente grafico. *MySQL Administrator* è progettato per amministrare un server MySQL, mentre *MySQL Query Browser* è progettato per aiutare ad effettuare query ed analizzare i dati memorizzati nel database MySQL.

Per essere utilizzato è necessario che esistano già una connessione al server MySQL ed un database su cui operare (operazioni che possono essere compiute con MySQL Administrator).

Questo programma è molto utile in fase di sviluppo in quanto permette di testare le proprie query prima del loro utilizzo all'interno del programma in *runtime*. La sua interfaccia grafica permette di inserire la query richiesta e visualizza in subito il risultato che otterrà applicandola al DB.

Nell'ambito di Sonicap, MySQL Query Browser è stato molto utile nel testing delle query e nella gestione del DataBase, probabilmente avrei impiegato molto più tempo per la realizzazione se non lo avessi utilizzato.





**Figura 5.4 – MySQL Query Browser**

# 6 – Realizzazione dell'applicazione

---

## 6.1 Fasi di realizzazione ed eventuali problemi

---

Un applicativo vasto come Sonicap ha richiesto un ordine di fasi di realizzazione ben strutturato per ridurre al minimo l'eventualità di errori di progettazione.

Il mio scopo era quello di creare un approccio di progettazione che mi permettesse in fine di avere un programma in grado di: leggere un file di traffico di rete, stabilire una connessione con un database, gestire operazioni su file di ampie dimensioni e salvare i risultati in un file di output.

La miglior sequenza logica di *step* da eseguire per i miei obiettivi mi è sembrata la seguente:

- studio della composizione di un file di traffico di rete
- creazione di un database in cui immagazzinare quante più informazioni possibili in breve tempo
- interfacciare il programma JAVA con il DataBase
- progettare e testare le performance di Sonicap con file di diverse dimensioni
- creare delle funzioni che mi permettano il salvataggio su file HTML
- utilizzo di thread

### 6.1.1 Studio della composizione di un file di traffico di rete

---

Un file dump classico ha una codifica interna in formato ASCII a 7 bit, perciò la lettura tramite editor di testo risulta impossibile.

L'unico modo per poter accedere ai dati di tali file è di convertirne il

loro contenuto con funzioni specifiche e tradurli in un formato quantomeno leggibile. La soluzione a questo problema è l'utilizzo della libreria JPCAP, già trattata nei paragrafi precedenti, la quale mi permette di estrarre i pacchetti catturati con tutti i dati raccolti su ognuno di loro.

### 6.1.2 Creazione del DataBase

---

La banca dati di cui avevo bisogno inizialmente doveva tener conto delle caratteristiche dei file dump che venivano letti, dei record al loro interno e dei file html che Sonicap avrebbe prodotto come output, solo in seguito mi sono reso conto del fatto che per realizzare l'applicazione ne sarebbero servite altre.

Riguardo il salvataggio delle informazioni sui file dump non ci sono stati dubbi, si doveva creare una tabella nella quale apparissero almeno i campi: nome, percorso, dimensione e numero record contenuti. Il vero problema si è presentato per la tabella relativa ai pacchetti. Avendo 5 formati (TCP,UDP,ICMP,IGMP,ARP) la creazione di 5 tabelle distinte, ognuna del proprio tipo, sembrava la soluzione più logica e veloce, tuttavia presentava un problema fatale per le prestazioni del programma (*approfondimento sezione 6.4*), perciò ho dovuto immagazzinare tutti i 5 formati in un'unica tabella chiamata *Record* e gestire accuratamente da programma gli accessi su di essa.

### 6.1.3 Interfacciamento JAVA – MYSQL

---

La connessione tra Sonicap e il DataBase è forse l'operazione più veloce e semplice per il risultato che produce dell'intero progetto. Il JAVA mette a disposizione delle librerie proprie che permettono l'interazione tra il programma ed il DB semplicemente tramite la creazione di una stringa di connessione e la seguente apertura di tale connessione.

Lo scenario di lettura/scrittura dati si svolgerà quindi in questo modo:

- creazione della stringa di connessione al DB
- apertura della connessione
- operazioni di lettura/scrittura
- chiusura della connessione

La facilità con cui JAVA si collega ad un DataBase MySQL è una delle ragioni principali per cui ho scelto questi due linguaggi per la creazione di Sonicap.

#### 6.1.4 Testing delle performance di Sonicap

---

Come già descritto, il mio programma ha come scopo principale quello di operare su grandi quantità di dati nel minor tempo possibile. Visto il campo in cui ci troviamo, questo requisito è quasi d'obbligo in quanto, parlando di *sniffing* della rete, è possibile che Sonicap ottenga come input file di dimensioni spropositate; basti pensare ad un tecnico di rete che potrebbe lasciare il suo pc a catturare i pacchetti inviati e ricevuti anche per diversi giorni, il risultato sarebbe un file dump contenente un'elevatissima quantità di informazioni.

Lavorando con file simili è stato possibile ottenere delle prestazioni di tutto rispetto viste le operazioni richieste.

Vediamo uno scenario di testing già effettuato utilizzando un programma di lettura chiamato TcpTrace (vorrei precisare che il file utilizzato rappresenta il traffico di rete accumulato in 2 giorni da una postazione del polo di informatica di Camerino)

*Macchina Utilizzata: Asus L5 processore Athlon 64 bit 3.200 Mhz*

*Memoria RAM: 512 Mbyte*

*Programma utilizzato: TcpTrace*

*Operazione: lettura file dump*

*Sistema Operativo: Ubuntu 8.04 32 bit*

*File dump: all.dump (1,04 Gbyte)*

*Numero pacchetti: 1.240.855*

*Tempo di computazione: indefinito*

Il computer si blocca dopo qualche minuto dall'esecuzione, perciò è impossibile sapere il tempo di computazione, si può solo ipotizzare che non sia possibile leggere un file di tali dimensioni con TcpTrace con le attuali risorse.

Vediamo con un programma come WireShark come si presenta lo scenario:

*Macchina Utilizzata: Asus L5 processore Athlon 64 bit 3.200 Mhz*  
*Memoria RAM: 512 Mbyte*  
*Programma utilizzato: WireShark*  
*Operazione: lettura file dump*  
*Sistema Operativo: Ubuntu 8.04 32 bit*  
*File dump: all.dump (1,04 Gbyte)*  
*Numero pacchetti: 1.240.855*  
*Tempo di computazione: 4 minuti stimato*

Siamo potuti arrivare solo ad una stima dato che al 40% della lettura Wireshark aveva superato i 350 Mbyte di memoria RAM utilizzata, controllando il suo tasso di incremento è probabile che a fine processo la memoria occupata superasse i 700 Mbyte. Un programma del genere non può quindi essere utilizzato su questo tipo di macchina, in quanto anche interrompendo il processo, i dati visualizzati sono molto difficili da consultare e producono forti rallentamenti di sistema.

Ed infine vediamo quanto tempo impiegherà invece Sonicap:

*Macchina Utilizzata: Asus L5 processore Athlon 64 bit 3.200 Mhz*  
*Memoria RAM: 512 Mbyte*  
*Programma utilizzato: Sonicap*  
*Operazione: lettura file dump*  
*Sistema Operativo: Ubuntu 8.04 32 bit*  
*File dump: all.dump (1,04 Gbyte)*  
*Numero pacchetti: 1.240.855*  
*Tempo di computazione: 1 minuto e 40 secondi*

Sonicap riesce a leggere più di un milione di pacchetti in meno di 2

minuti e occupando circa 30 Mbyte di RAM. Le performance di lettura sono più che accettabili visti i risultati, tuttavia i rallentamenti maggiori la avremo al momento del salvataggio su DB.

Controlliamo l'ultimo test:

*Macchina Utilizzata: Asus L5 processore Athlon 64 bit 3.200 Mhz*

*Memoria RAM: 512 Mbyte*

*Programma utilizzato: Sonicap*

*Operazione: lettura file dump e salvataggio su DataBase*

*Sistema Operativo: Ubuntu 8.04 32 bit*

*File dump: all.dump (1,04 Gbyte)*

*Numero pacchetti: 1.240.855*

*Tempo di computazione: 19 minuti e 30 secondi*

Siamo passati da 2 minuti circa a quasi 20. Non ho potuto confrontare questi risultati con quelli di altri programmi, perchè a quanto pare Sonicap sembra essere l'unico che permetta il salvataggio su DB, però posso comunque trarre delle conclusioni in proposito: 19 minuti per la lettura e scrittura su DB di circa un milione di pacchetti di rete sono accettabili per il fatto che, una volta completato il processo, avrò tutti i dati che mi servono disponibili all'interno di un DataBase, perciò l'accesso in lettura sarà ancora più veloce di quello da file dump.

Un'ultima osservazione importante relativa al caricamento dei dati dal DataBase: è stato necessario inserire un limite ai record che possono essere estratti dal DB con una singola query. Il motivo è che con un numero eccessivo di dati passati dalla banca dati al programma, la memoria virtuale di java avrebbe prodotto un errore per eccessivo carico di dati. La soluzione a questo problema è stata quella di dividere la query principale in parti che eseguissero sempre le stesse operazioni, ma su dati diversi. Vediamo un esempio:

la mia query deve caricare tutti i pacchetti del file *all.dump*, ma avendo tale file 1.240.855 pacchetti, java mi segnalerà un errore di sovraccarico dati se utilizzo una sola query.

Dopo vari test (con le risorse sopra descritte) ho approssimato il

numero di pacchetti selezionabili con una singola query a 10.000 record; perciò ora il processo sarà:

esegui query dal 1° pacchetto al 10.000°

esegui query dal 10.001° pacchetto al 20.000°

e così via fino a coprire tutto il file.

Questo procedimento verrà descritto meglio nella sezione 6.4.5 relativa alle statistiche.

---

#### 6.1.5 Salvataggio su file HTML

Il salvataggio in formato HTML è abbastanza semplice in JAVA, è solo necessario creare un file di testo, creare una connessione a tale file, scriverci dentro il codice HTML e chiudere la connessione.

In quattro passaggi è possibile creare l'output del programma senza nemmeno utilizzare complicate chiamate di sistema.

Il vantaggio dell'HTML sta proprio nel fatto che è sufficiente un file di testo per poter creare una pagina.

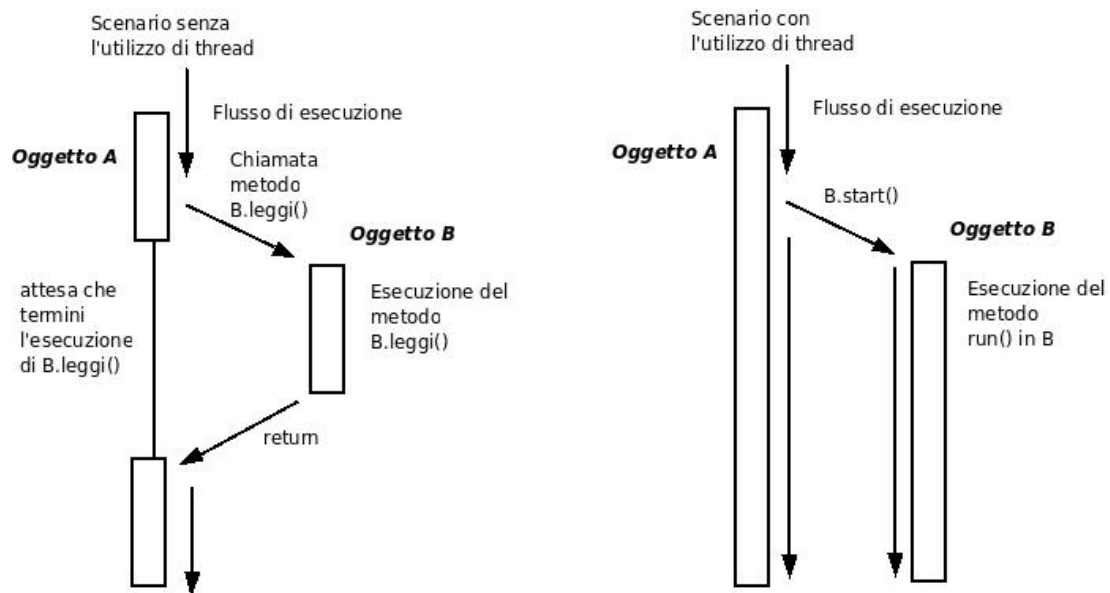
---

#### 6.1.6 Utilizzo di thread

Che cos'è un thread?

I thread, rappresentano il mezzo mediante il quale, Java fa eseguire un'applicazione da più Virtual Machine (JVM) contemporaneamente, allo scopo di ottimizzare i tempi del *runtime*. Ovviamente, si tratta di un'illusione: per ogni programma solitamente esiste un'unica JVM ed un'unica CPU. Ma la CPU può eseguire codice da più progetti all'interno della gestione della JVM per dare l'impressione di avere più processori.

In termini più comprensibili, un thread non fa altro che dividere il singolo processo dell'applicazione lanciata in due, permettendo così lo svolgimento di entrambi allo stesso tempo, come nell'illustrazione:



**Figura 6.1 - Thread**

Ora che si è appreso il funzionamento di un thread, è facile capire il motivo per cui nel mio programma sono necessari. Pensiamo alla routine per la lettura del file dump descritta sopra (sezione 6.1.4), se non si utilizzassero thread, avrei Sonicap bloccato in lettura e scrittura su DataBase per 19 minuti e le intere risorse del processore sarebbero assegnate a questo processo rendendo molto più lento l'utilizzo di qualsiasi altra applicazione. Immaginiamo ora di creare un thread al momento dell'apertura del file, la lettura e scrittura sarebbe assegnata ad un altro processo mentre avrei sempre il processo principale attivo che mi permette di utilizzare altre funzionalità di Sonicap senza dover attendere 19 minuti di esecuzione.



## 6.2 **Struttura del DataBase**

Un database è composto di entità e relazioni.

Le entità sono oggetti che hanno significato anche quando vengono considerati in modo isolato e sono di interesse per la realtà che si vuole modellare, mentre le relazioni sono i legami che stabiliscono un'interazione tra le entità. Esistono tre tipi di relazioni:

- 1:1 (uno a uno) – ad ogni elemento della prima entità corrisponde uno e un solo elemento della seconda
- 1:N (uno a molti) – ad un elemento della prima entità possono corrispondere più elementi della seconda ma non viceversa
- N:N (molti a molti) – ad ogni elemento della prima entità possono corrispondere più elementi della seconda e viceversa

### 6.2.1 Entità

FileDump		
Funzionalità: immagazzinare tutti i dati relativi ad un file dump preso come input dal programma		
Nome Colonna	Type	Dati
<u>ID_file (PK)</u>	INT	Chiave primaria con incremento automatico
name	TEXT	Nome e percorso del file dump
file_size	DOUBLE	Dimensione in KBytes
num_record	DOUBLE	Numero totale di record
tcpRec	DOUBLE	Totale record TCP
udpRec	DOUBLE	Totale record UDP
igmpRec	DOUBLE	Totale record IGMP
arpRec	DOUBLE	Totale record ARP
ipRec	DOUBLE	Totale record IGMP
data	TEXT	Flag che mi permette di sapere l'intervallo di tempo in cui è stato catturato il file

Record		
Funzionalità:immagazzinare i dati relativi ad ogni singolo pacchetto letto		
<i>Nome Colonna</i>	<i>Type</i>	<i>Dati</i>
<u>ID_rec (PK)</u>	INT	Chiave primaria ad incremento
day	INT	Giorno
months	TEXT	Mese (formato 3 caratteri)
year	INT	Anno
hour	INT	Ora
minutes	INT	Minuti
seconds	INT	Secondi
src_mac	TEXT	Indirizzo MAC sorgente
dst_mac	TEXT	Indirizzo MAC destinazione
IP_source	TEXT	Indirizzo IP sorgente
IP_destination	TEXT	Indirizzo IP destinazione
protocol	INT	Tipo di protocollo
priority	INT	Priorità
hop	INT	Numero di hop
offset	INT	Numero di offset
ident	INT	Identità
port_source	INT	Porta sorgente
port_destination	INT	Porta destinazione
seq	TEXT	Numero di sequenza
win	INT	Finestra di congestione
ackSet	TINYINT(1)	Numero Ack di partenza
ack	TEXT	Numero di Ack ricevuti
SYN	TINYINT(1)	Bit di flag
FIN	TINYINT(1)	Bit di flag
PUSH	TINYINT(1)	Bit di flag
RESET	TINYINT(1)	Bit di flag
URG	TINYINT(1)	Bit di flag
len	INT	Dimensione totale della frame
length	INT	Dimensione del protocollo
type	TEXT	Tipo (per ICMP)
code	TEXT	Codice (per ICMP)

type_ARP	TEXT	Tipo di richiesta (per ARP)
cont	INT	Numero del record all'interno del file di origine
<u>ID_file (FK)</u>	INT	Chiave esterna presa da FileDump

### Conversion\_Port

Funzionalità: permette la conversione da protocollo a standard IANA leggendo la porta che si sta utilizzando

<i>Nome Colonna</i>	<i>Type</i>	<i>Dati</i>
<u>ID (PK)</u>	INT	Chiave primaria ad incremento
num	INT	Numero di porta
name	TEXT	Sigla del protocollo
tipo	TEXT	Tipo di protocollo
protocol	TEXT	Nome completo del protocollo

### Color

Funzionalità: permette di cambiare i colori dei vari protocolli che vengono visualizzati sul display a seconda delle esigenze dell'utente

<i>Nome Colonna</i>	<i>Type</i>	<i>Dati</i>
<u>ID (PK)</u>	INT	Chiave primaria ad incremento
tipo	TEXT	Tipo di protocollo al quale è applicato il colore
R	INT	Indice di tonalità del rosso
G	INT	Indice di tonalità del verde
B	INT	Indice di tonalità del blu
RGB	TEXT	Combinazione di colori RGB
descrizione	TEXT	Descrizione del colore

### Chart Images

Funzionalità: permette il salvataggio dei grafici creati del programma per avere un accesso ai dati più veloce

<i>Nome Colonna</i>	<i>Type</i>	<i>Dati</i>
<u>ID_Chart (PK)</u>	INT	Chiave primaria ad incremento

path	TEXT	Percorso del grafico salvato
identificativo	INT	Permette di salvare più grafici con lo stesso nome fornendo un identificativo unico

Settings		
Funzionalità: permette all'utente di settare le prestazioni del programma nel modo che ritiene più produttivo basandosi sulle risorse del suo compilatore		
Nome Colonna	Type	Dati
<u>ID (PK)</u>	INT	Chiave primaria ad incremento
queryLimit	INT	Limite massimo di record selezionabili con una query
display_limit	INT	Limite massimo di pacchetti visualizzabili sul display
directoryTemp	TEXT	Directory temporanea per le immagini dei grafici
imagesFormat	TEXT	Formato in cui le immagini andranno salvate
maxPathLength	INT	Lunghezza massima della directory in cui si può trovare il file dump
current	INT	Indica le impostazioni in uso del programma

### 6.2.2 Relazioni

E' presente una sola relazione tra entità all'interno del DataBase, questo perché ognuna di loro rappresenta un'area distinta del programma.

FileDump	1	Contiene	N	Record
Essendo una relazione 1 a N, la chiave primaria di FileDump (ID_file), diventa chiave esterna della tabella Record. Per accedere ai record di uno specifico file dump si utilizza il comando SQL <i>inner join</i> .				

In questa sezione verranno mostrate tutte le classi che compongono il mio progetto e la loro funzionalità.

Partiamo dalle classi che hanno come unico scopo quello di creare un oggetto con tutti i suoi attributi e metodi per il loro settaggio e restituzione: i metodi *set* setteranno un certo attributo mentre i metodi *get* lo restituiranno.

### *Chart.java*

crea l'oggetto Chart con tutti i metodi per la creazione di diversi tipi di grafici.

### *Components.java*

crea l'oggetto Components che ha il compito di indicare quali componenti dell'interfaccia saranno modificati a seguito dell'utilizzo di un thread.

### *Data.java*

crea l'oggetto Data che mi permette di effettuare controlli e conversioni come la conversione del numero del mese in sigla ecc.

### *FileDump.java*

crea l'oggetto FileDump che rappresenta il file dump dal quale Sonicap cattura i pacchetti da inserire nel DataBase.

### *MyColor.java*

crea l'oggetto MyColor che rappresenta un colore in tonalità RGB e che utilizzo per gestire la grafica dei vari tipi di pacchetti visualizzati sul display.

### *MyJtable.java*

crea l'oggetto MyJtable che rappresenta il display dove si andranno a visualizzare i pacchetti. E' stato necessario l'utilizzo di questa classe al posto della comune Jtable di JAVA per il fatto che solo in questo modo posso andare ad utilizzare i colori al suo interno tramite la modifica del render.

### *Protocol.java*

crea l'oggetto Protocol che rappresenta la combinazione porta/protocollo per la conversione tramite le standard IANA.

### *Query.java*

crea l'oggetto Query che genera una query modificabile da programma per eseguire statistiche avanzate o per interagire con il database da qualsiasi altra classe.

### *Record.java*

crea l'oggetto record che rappresenta un pacchetto letto dal file dump preso come input. Potendo assumere tutti e 5 i tipi principali (tcp,udp,icmp,igmp,arp) è completo di tutti i campi che li contraddistinguono l'uno dall'altro.

### *Settings.java*

crea l'oggetto Settings che contiene tutti le voci del programma modificabili dall'utente.

### *Threads.java*

crea l'oggetto Thread che, in base all'operazione richiesta, esegue una certa sequenza di istruzioni in un processo parallelo.

Passiamo ora ai metodi che hanno il compito di gestire gli oggetti sopra citati.

### *GestoreDB.java*

Classe che ha il compito di stabilire la connessione con il database e gestirne gli accessi in scrittura e lettura.

### *GestoreHTML.java*

Classe che ha il compito di creare l'output del programma generando un file html e scrivendoci dentro i risultati delle ricerche effettuate dall'utente.

### *GestoreIO.java*

Classe che ha il compito di eliminare file temporanei superflui o di creare directory temporanee nel caso non esistano.

### *LeggiFileDump.java*

Classe che ha il compito di ottenere un file dump di traffico come input e di passare il suo contenuto alla classe GestoreDB che andrà a salvare il tutto sul database.

### *Statistics.java*

Classe che ha il compito di generare le statistiche richieste dall'utente accedendo in sola lettura al database e riportando i risultati su display.

Rimangono da illustrare ora solo le due classi principali.

### *SonicapGUI.java*

Classe che genera l'interfaccia grafica con tutti i suoi componenti e che richiama tutte le altre classi del programma tramite i suoi eventi.

### *Main.java*

Classe principale che permette l'esecuzione del programma richiamando la classe SonicapGUI.

## 6.4 **Analisi approfondita dei metodi principali**

---

In questa sezione saranno mostrati i passaggi più importanti e complessi dello sviluppo di Sonicap con una breve panoramica sulle funzioni principali utilizzate.

### 6.4.1 Lettura da file .dump

---

Per l'acquisizione di pacchetti da un file dump è necessario l'utilizzo della libreria JPCAP.

Questa operazione è forse una delle più complesse di tutto il programma, visto che utilizza vari tipi di funzioni come:

- metodi per l'apertura di un file
- metodi per la lettura dei pacchetti
- metodi per la scrittura su DataBase

La sua implementazione è di fatto molto lunga da descrivere e risulterebbe comunque difficile da comprendere per chi non è pratico di programmazione in JAVA, perciò mi limiterò a descriverne i passi logici:

- instaurazione della connessione con il DB
- creazione del Record da inserire nel DB
- apertura del file dump tramite il JPCAP Captor
- inserimento dei valori di ogni pacchetto nel Record
- inserimento del Record nel DataBase
- chiusura del JPCAP Captor sul file
- chiusura della connessione al DataBase

Classe LeggiFileDump	
<b>leggiFile()</b>	<i>boolean leggiFile (String path, int IDfile, javax.swing.JLabel lblRecNum, javax.swing.JLabel lblFileName) {}</i>
Legge il file dump con percorso <i>path</i> e salva i pacchetti nel DataBase.	



Classe LeggiFileDump	
<b>typeArp()</b>	<i>String typeARP(String line){}</i>
Ritorna il tipo di richiesta ARP del pacchetto in lettura.	

Classe LeggiFileDump	
<b>getFileSize()</b>	<i>double getFileSize(String path){}</i>
Ritorna la dimensione del file dump che si sta leggendo.	

Classe LeggiFileDump	
<b>setData()</b>	<i>void setData(java.util.Date data){}</i>
Converte il timestamp presente sui pacchetti, in formato data.	

Classe LeggiFileDump	
<b>getMACaddress()</b>	<i>void getMACaddress(Packet packet){}</i>
Estrarre il MAC address del destinatario e del mittente dal pacchetto in lettura.	

#### 6.4.2 Accesso dati e scrittura nel DataBase

L'accesso dati e la scrittura all'interno del DataBase sono operazioni abbastanza semplici a livello di codice, ma sono estremamente importanti a livello logico perché l'errato inserimento di dati nel DB, come anche la lettura, potrebbe provocare errori irreversibili eliminabili solo tramite l'eliminazione completa dell'intera banca dati. Proprio per questi motivi, le operazioni per interagire con il database, devono sempre essere inserite in una struttura di *try – catch*.

Vediamo in cosa consistono questi due comandi:

- *try* – cattura le eccezioni del codice eseguito al suo interno, in maniera che se la prima istruzione genera un errore, la seconda non viene eseguita e si passa direttamente al *catch*
- *catch* – il *catch* cattura le eccezioni compatibili con il suo argomento e mi permette di eseguire il set di istruzioni al suo interno. Trattandosi ad esempio di DataBase, l'eccezione che ha

più probabilità di verificarsi è la SQLException

Visto il vasto numero di metodi che operano sul DB, di seguito mostrerò solo quelli per la creazione e la chiusura della connessione al DataBase.

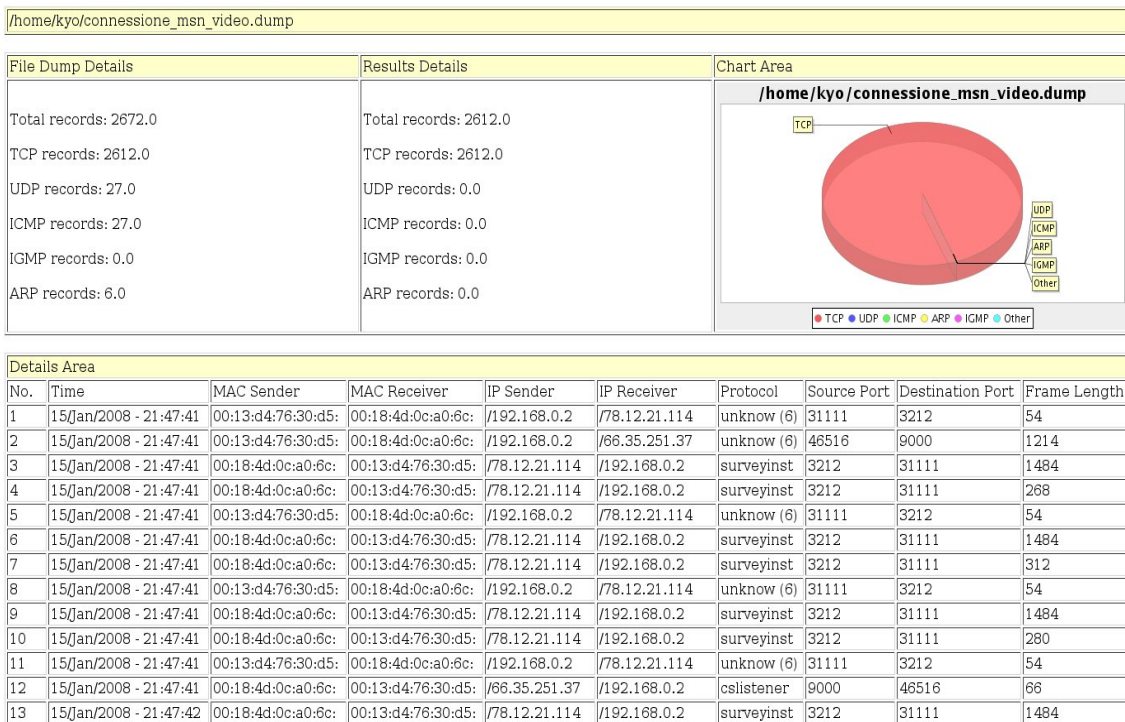
Classe GestoreDB	
<b>creatConnection()</b>	<i>void createConnection(String metodo){}</i>
Crea la connessione al DataBase.	

Classe GestoreDB	
<b>closeConnection()</b>	<i>void closeConnection(String metodo){}</i>
Chiude la connessione al DataBase.	

### 6.4.3 Salvataggio su file HTML

Il fatto di poter salvare le statistiche effettuate su un certo tipo di file di traffico di rete è sicuramente indispensabile in questo tipo di programma.

Il risultato del salvataggio sarà un file html con la seguente struttura:



**Figura 6.2 – File Output HTML**

*File Dump Details:* indica le caratteristiche del file dump letto

*Result Details:* indica le caratteristiche dopo aver effettuato le statistiche

*Chart Area:* mostra il grafico relativo al file dopo le statistiche

*Details Area:* visualizza i dettagli di ogni record selezionato con le statistiche

Il file di output è graficamente molto semplice, non utilizza componenti per migliorare l'estetica per un semplice motivo: dovendo all'occorrenza operare su file dump di grandi dimensioni, la creazione dell'output html sarebbe molto lenta e occuperebbe un quantitativo troppo grande di memoria. Utilizzando soltanto qualche tabella e

stringhe comuni, il processo di salvataggio risulta più veloce e allo stesso tempo comprensibile.

Classe GestoreHTML	
<b>createHTML()</b>	<i>boolean createHTML(String percorso, String titolo){}</i>
Crea un file html e vi inserisce solo l'intestazione.	

Classe GestoreHTML	
<b>openWriter()</b>	<i>void openWriter(String percorso){}</i>
Lascia aperto un accesso in scrittura ad un file html già esistente e completo di intestazione.	

Classe GestoreHTML	
<b>closeWriter()</b>	<i>void closeWriter(){}</i>
Chiude l'accesso in scrittura ad un file html.	

Classe GestoreHTML	
<b>appendChartToHTML()</b>	<i>void appendChartToHTML(ArrayList grafici, FileDump dump, FileDump result){}</i>
Permette l'inserimento di uno o più grafici all'interno del file html	

Classe GestoreHTML	
<b>createDetails()</b>	<i>void createDetails(int totaleColonne){}</i>
Crea la tabella e le colonne per l'inserimento dei dettagli di ogni pacchetto.	

Classe GestoreHTML	
<b>appendPacketToHTML()</b>	<i>void appendPacketToHTML(ArrayList valori){}</i>
Aggiunge le caratteristiche di ogni pacchetto del file alla tabella già creata.	

Classe GestoreHTML	
<b>insertFinalTag()</b>	<i>void insertFinalTag(){}</i>
Inserisce i tag per la chiusura del file.	

## 6.4.4 Thread

Nella sezione 6.1.6 abbiamo introdotto e descritto il concetto di *thread*, ora andiamo a vedere in quale operazioni viene applicato.

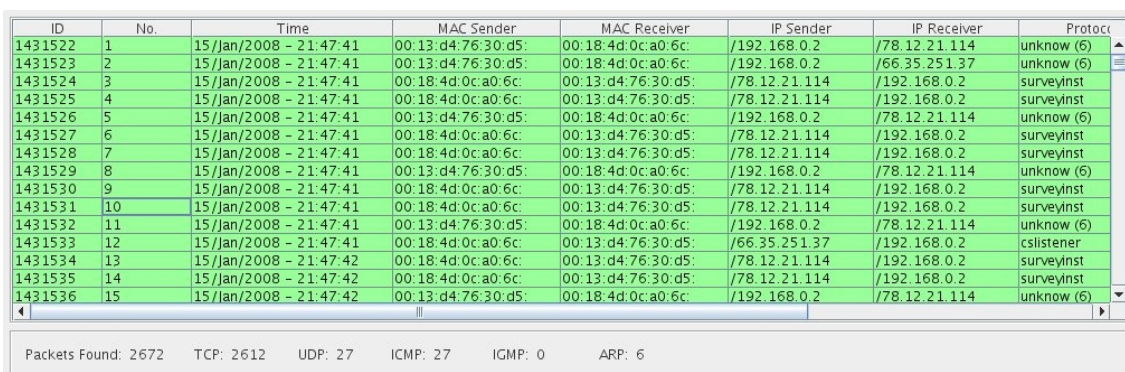
Come già detto, un thread mi permette di eseguire più processi allo stesso tempo evitando di rimanere in attesa di operazioni troppo lunghe. In Sonicap le funzioni che richiedono maggior tempo e risorse sono:

- la lettura e scrittura del file dump sul DataBase
- la creazione di statistiche e la visualizzazione sul display
- la creazione del file html di output
- l'eliminazione di dati dal DataBase

Nel caso di piccoli flussi di input, l'utilizzo dei thread potrebbe risultare inutile o addirittura sconveniente, ma quando si parla di grandi flussi di dati, il loro utilizzo potrebbe evitare gravi rallentamenti o addirittura il blocco totale del sistema.

L'implementazione di tali strutture però, risulta tutt'altro che semplice, a partire dai problemi legati all'aggiornamento dell'interfaccia grafica.

Pensiamo alla “creazione di statistiche e visualizzazione sul display”. Al termine dell'operazione dovrò aver visualizzati i risultati sul display, ovvero sulla tabella di ricerca, di Sonicap in questo modo:



ID	No.	Time	MAC Sender	MAC Receiver	IP Sender	IP Receiver	Protocol
1431522	1	15/Jan/2008 - 21:47:41	00:13:d4:76:30:d5	00:18:4d:0c:a0:6c	/192.168.0.2	/78.12.21.114	unknow (6)
1431523	2	15/Jan/2008 - 21:47:41	00:13:d4:76:30:d5	00:18:4d:0c:a0:6c	/192.168.0.2	/66.35.251.37	unknow (6)
1431524	3	15/Jan/2008 - 21:47:41	00:18:4d:0c:a0:6c	00:13:d4:76:30:d5	/78.12.21.114	/192.168.0.2	surveyinst
1431525	4	15/Jan/2008 - 21:47:41	00:18:4d:0c:a0:6c	00:13:d4:76:30:d5	/78.12.21.114	/192.168.0.2	surveyinst
1431526	5	15/Jan/2008 - 21:47:41	00:13:d4:76:30:d5	00:18:4d:0c:a0:6c	/192.168.0.2	/78.12.21.114	unknow (6)
1431527	6	15/Jan/2008 - 21:47:41	00:18:4d:0c:a0:6c	00:13:d4:76:30:d5	/78.12.21.114	/192.168.0.2	surveyinst
1431528	7	15/Jan/2008 - 21:47:41	00:18:4d:0c:a0:6c	00:13:d4:76:30:d5	/78.12.21.114	/192.168.0.2	surveyinst
1431529	8	15/Jan/2008 - 21:47:41	00:13:d4:76:30:d5	00:18:4d:0c:a0:6c	/192.168.0.2	/78.12.21.114	unknow (6)
1431530	9	15/Jan/2008 - 21:47:41	00:18:4d:0c:a0:6c	00:13:d4:76:30:d5	/78.12.21.114	/192.168.0.2	surveyinst
1431531	10	15/Jan/2008 - 21:47:41	00:18:4d:0c:a0:6c	00:13:d4:76:30:d5	/78.12.21.114	/192.168.0.2	surveyinst
1431532	11	15/Jan/2008 - 21:47:41	00:13:d4:76:30:d5	00:18:4d:0c:a0:6c	/192.168.0.2	/78.12.21.114	unknow (6)
1431533	12	15/Jan/2008 - 21:47:41	00:18:4d:0c:a0:6c	00:13:d4:76:30:d5	/66.35.251.37	/192.168.0.2	csllstener
1431534	13	15/Jan/2008 - 21:47:42	00:18:4d:0c:a0:6c	00:13:d4:76:30:d5	/78.12.21.114	/192.168.0.2	surveyinst
1431535	14	15/Jan/2008 - 21:47:42	00:18:4d:0c:a0:6c	00:13:d4:76:30:d5	/78.12.21.114	/192.168.0.2	surveyinst
1431536	15	15/Jan/2008 - 21:47:42	00:13:d4:76:30:d5	00:18:4d:0c:a0:6c	/192.168.0.2	/78.12.21.114	unknow (6)

Packets Found: 2672    TCP: 2612    UDP: 27    ICMP: 27    IGMP: 0    ARP: 6

**Figura 6.3 – Packets Display**

il problema è che avendo diviso il programma in 2 processi distinti, al termine del thread non posso più utilizzare l'aggiornamento

automatico dell'interfaccia presente nel processo principale. Proprio per questo motivo ho dovuto creare la classe *Components* che ha il compito di passare i componenti dell'interfaccia che saranno modificati al seguito dell'utilizzo della classe *Thread*.

In Java ci sono due metodi per implementare i thread:

- *come sottoclasse della classe Thread* – la super classe possiede il metodo *run* che fa partire il processo, è necessario creare un'istanza della sottoclasse tramite *new* e si invoca il metodo *start()* che a sua volta richiama il metodo *run()*
- *come classe che implementa l'interfaccia runnable* – si implementa il metodo *run()* nella propria classe, si crea un'istanza di essa e un'istanza della classe *Thread* che avrà come argomento la nostra classe; si invoca infine il metodo *start()*

L'implementazione utilizzata in Sonicap è quella del secondo tipo.

Classe Threads	
<b>Public Threads()</b>	<i>public Threads(String operazione, Components componenti) {}</i>
Costruttore della classe Threads che prende come argomento i componenti dell'interfaccia che dovrà andare ad aggiornare, e l'operazione richiesta da eseguire.	

Classe Threads	
<b>run()</b>	<i>public void run() {}</i>
In base al valore di <i>operazione</i> lancia un Thread che esegue le istruzioni richieste. Al termine del metodo, il thread viene chiuso automaticamente.	

## 6.4.5 Creazione delle statistiche

Le ricerche che si possono effettuare sui dati presenti nel DataBase sono la combinazione di questi campi:

The image shows a control panel for network traffic analysis. It features a 'File Dump' dropdown menu. Below it are checkboxes for protocol types: TCP, UDP, ICMP, IGMP, ARP, and Port Conversion. There are two rows for 'Source' and 'Destination', each with a 'MAC Address' dropdown, an input field, and a 'Clear' button. A 'Protocol Type' dropdown is set to 'Item 1'. Below this is a 'Data' section with a dropdown menu and two input fields. The 'Hour' section has a dropdown menu and four input fields, each containing the number '1'. At the bottom, there are checkboxes for 'Flag Bits': SYN, FIN, PUSH, RESET, and URG.

**Figura 6.4 – Lista controlli**

Descrivendoli in generale, i controlli di alto livello che si possono effettuare sono:

- per tipo di record (TCP,UDP,ICMP,IGMP,ARP)
- per indirizzo MAC o IP
- per tipo di protocollo
- all'interno di un certo periodo di tempo (sia data che ora)
- in base ai bit di flag contenuti nei pacchetti

Trattandosi di dati presenti in un DataBase, l'unico modo per ottenerne una selezione è tramite l'utilizzo di una query. In questo caso però i controlli sono numerosi e possono anche presentarsi in ordine sparso, perciò è più conveniente creare una query per ogni controllo e farle lavorare insieme, oppure creare un'unica query e gestire l'inserimento di più controlli? La soluzione più conveniente è ovviamente la seconda, sia per le prestazioni che per l'organizzazione del codice.

## Gestione dei controlli della query

Al momento del click su “effettua ricerca”, controllo i campi che sono stati compilati.

Il primo controllo è sul tipo di record, inserisco il comando *IN ()* per indicare i numeri di protocollo che devono essere selezionati

---

```
query = query + " AND protocol IN ( <protocolli> ) "
```

---

Vi è una routine da programma per gestire gli indirizzi IP e MAC, ma per ora illustriamo solo come sarebbe il controllo query se si vogliono conoscere i pacchetti da un certo indirizzo IP ad un certo indirizzo MAC

---

```
query = query + " AND IP_source = <IP> AND dst_mac = <MAC> )"
```

---

Il controllo sulla data, come anche quello sull'ora, sono abbastanza complessi per il fatto che possono presentarsi insieme o separati, e possono presentarsi come:

ricerca pacchetti da una certa data

---

```
query = query + " AND <type> >= <valore> "
```

---

ricerca pacchetti fino ad una certa data

---

```
query = query + " AND <type> <= <valore> "
```

---

ricerca pacchetti trasmessi tra due date

---

```
query = query + " AND <type> BETWEEN <valore1> AND <valore2> "
```

---

ricerca pacchetti trasmessi in una data specifica

---

```
query = query + " AND <type> = <valore> "
```

---

Ponendo la variabile *type* uguale a *date* si ottengono i controlli sulla data, ponendola uguale *hour* si ottengono i controlli in base all'orario. I bit di flag sono campi di tipo boolean perciò possono assumere solo i valori *true(1)* o *false(0)*. Il controllo nella query risulta semplice



---

*query = query + " AND <type> = 1 "*

---

Dove al posto di *type* inserisco il tipo di flag (SYN, FIN, PUSH, RESET, URG).

Il controllo sul tipo di protocollo utilizzato secondo lo standard IANA si esegue completamente da programma, senza andare a toccare la query SQL, perciò non verrà descritto in questa sezione.

L'ultima operazione da effettuare è dividere la query per evitare il sovraccarico di memoria descritto nella sezione 6.1.4.

Uno dei valori delle impostazioni di Sonicap è *Query Limit* che indica appunto il numero massimo di record che possono essere caricati con una singola selezione. Il valore di default è settato a 10.000, perciò per sapere il numero di query da effettuare dovrò semplicemente dividere il numero di pacchetti totali del file per tale valore:

ammesso che io abbia 83.000 pacchetti su cui operare

$83.000 / 10.000 = 8,3$

arrotondando per eccesso ottengo 9 query ognuna delle quali con il controllo finale

---

*query = query + " LIMIT <num1>, <num2> "*

---

Dove *num1* indica da quale record partire, mentre *num2* indica quanti record selezionare.

Classe Query	
<b>Public Query()</b>	<i>public Query(String filename,boolean selectAll, boolean tcp, boolean udp, boolean icmp, boolean ip, boolean arp) { }</i>
Costruttore che inzializza la query con l'INNER JOIN tra i record con ID_file uguale a quello del file dump trattato	

Classe Query	
<b>insertPortControl()</b>	<i>void insertPortControl(Vector portList) { }</i>
Inserisce il controllo sulla porta sorgente e su quella destinazione.	

Classe Query	
<b>chooseAddressType()</b>	<i>void chooseAddressType(int tipo, int combo, String queryS, String indirizzo){}</i>
Inserisce il controllo degli indirizzi MAC e IP	

Classe Query	
<b>insertQueryControl()</b>	<i>int insertQueryControl(int valore1, int valore2, boolean from, boolean to, boolean onDay, String type){}</i>
Inserisce il controllo sulla data e sull'ora.	

Classe Query	
<b>insertFlagControl()</b>	<i>void insertFlagControl(boolean syn, boolean fin, boolean push, boolean reset, boolean urg){}</i>
Inserisce il controllo sui bit di flag.	

#### 6.4.6 Creazione dei grafici

Benché siano presenti un gran numero di grafici implementabili nella libreria JFREECHART, in Sonicap ne utilizzeremo solo 2 tipi: il grafico a torta e l'istogramma.

I motivi di questa scelta sono semplici: questi due tipi sono più che sufficienti per illustrare al meglio le caratteristiche dei file dump e delle statistiche effettuate su di essi.

L'implementazione di ogni tipo di grafico ha quasi sempre la medesima struttura:

- creazione del set di dati da passare al grafico
- creazione del grafico
- inserimento del grafico in un pannello

tramite questi passaggi posso creare un grafico con i dati scelti da me e incorporarlo in un pannello presente nel programma.

In Sonicap vi è un altro passo all'interno di questa struttura: il

salvataggio su file immagine, grazie al quale, ogni volta che viene richiesto il grafico di un file dump già presente nel DataBase, non sarà più necessario eseguire l'intera struttura per ricrearlo, ma semplicemente verrà caricata l'immagine ad esso associata.

Classe Chart	
<b>create3DPie()</b>	<i>public ChartPanel create3DPie(double tcp, double udp, double icmp, double arp, double ip, double total, String titolo, boolean immagine, String path) {}</i>
Ritorna il pannello contenente il grafico Torta 3D.	

Classe Chart	
<b>createSampleDataset()</b>	<i>public PieDataset createSampleDataset(double tcp, double udp, double icmp, double arp, double ip, double total) {}</i>
Ritorna il set di dati di un grafico Torta 3D.	

Classe Chart	
<b>createChart()</b>	<i>public JFreeChart createChart(final PieDataset dataset, String titolo) {}</i>
Ritorna il grafico Torta 3D creato con i dati definiti nel dataset.	

Classe Chart	
<b>createVerticalBarChart()</b>	<i>public ChartPanel CreateVerticalBarChart() {}</i>
Ritorna il pannello contenente il grafico ad Istogrammi.	

Classe Chart	
<b>createDataset()</b>	<i>private DefaultCategoryDataset createDataset() {}</i>
Ritorna il set di dati di un grafico ad Istogrammi.	

Classe GestoreHTML	
<b>createChart()</b>	<i>private JFreeChart createChart(final CategoryDataset dataset) {}</i>
Ritorna il grafico ad Istogrammi creato con i dati definiti nel dataset.	

#### 6.4.7 Modifica dei settaggi

Le performance di Sonicap sono legate al tipo di computer su cui viene fatto girare. In computer di ultima generazione potrebbe risultare troppo lento nell'esecuzione delle operazioni, mentre in computer ormai vecchi potrebbe anche causare il blocco del sistema. Per evitare tali eventualità, in Sonicap, un utente ha la possibilità di cambiare alcune impostazioni per adattarlo come meglio crede alle risorse del suo pc.

I settaggi che possono essere cambiati sono:



**Figura 6.5 – Settings**

- Query Limit – indica il numero di record che possono essere caricati con una singola query
- Display Limit – indica il numero di pacchetti che possono essere visualizzati sul display
- Directory – indica la directory temporanea in cui verranno salvati le immagini dei grafici
- Max Path Length – lunghezza massima del percorso dove si può trovare un file dump
- Images Format – indica in quale formato si vogliono salvare i grafici

Il cambiamento di questi valori può migliorare le performance in base al proprio sistema, ma una modifica errata potrebbe impedire il corretto funzionamento dell'intero programma, per questo motivo c'è sempre l'opzione "Restore Default" che elimina tutti i settaggi creati e reimposta quelli di default di Sonicap.

Classe GestoreDB	
<b>readSettings()</b>	<i>boolean readSettings() {}</i>
Ritorna true se riesce a leggere tutti i settaggi presenti nel DataBase.	

Classe GestoreDB	
<b>deleteSettingsType()</b>	<i>void deleteSettingsType(Settings s) {}</i>
Elimina il profilo di settaggi selezionato.	

Classe GestoreDB	
<b>readSettings2()</b>	<i>Settings readSettings2() {}</i>
Ritorna i settaggi attuali del programma.	

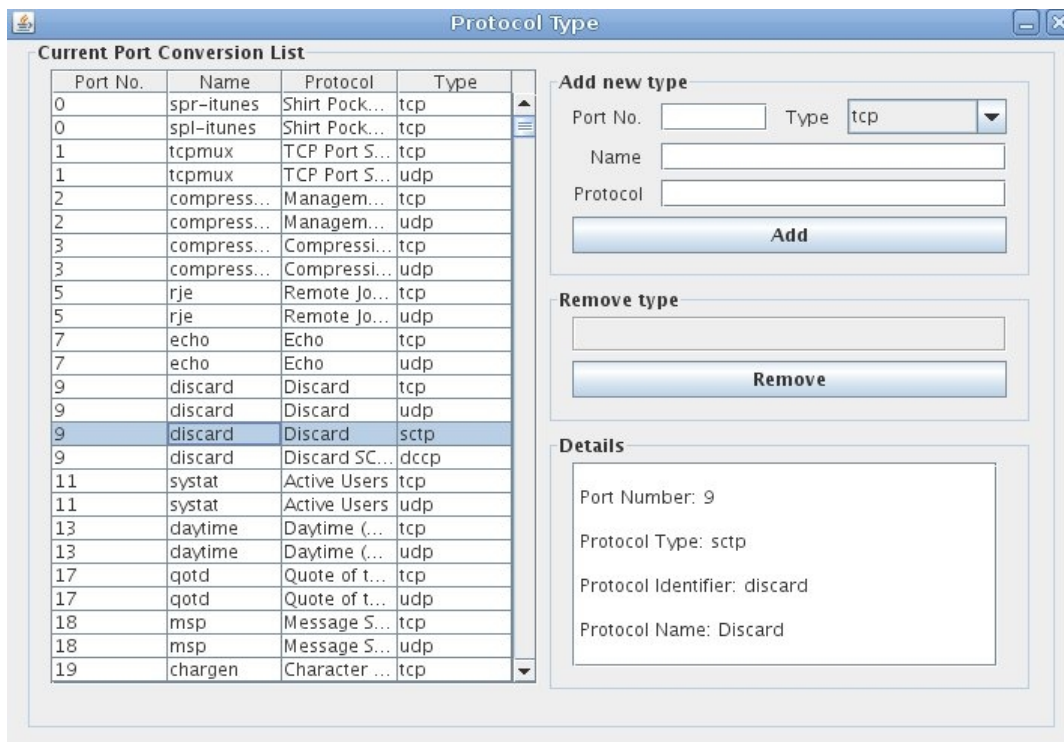
Classe GestoreDB	
<b>setDefaultSettings()</b>	<i>void setDefaultSettings() {}</i>
Elimina tutti i settaggi esistenti e ristabilisce quelli di default.	

Classe GestoreDB	
<b>setAsDefault()</b>	<i>void setAsDefault(int ID) {}</i>
Pone il settaggio selezionato come corrente.	

Classe GestoreDB	
<b>setNewSettings()</b>	<i>void setNewSettings(Settings set) {}</i>
Inserisce un nuovo profilo di settaggi.	

## 6.4.8 Conversione dei protocolli

E' possibile convertire il numero del protocollo di un pacchetto nel più specifico caso di utilizzo, controllando la combinazione del numero di porta utilizzata ed il tipo di protocollo. Vediamo un esempio. Prendiamo un file dump contenente il traffico di rete di una conversazione messenger; catturandolo otterrò pacchetti di tipo TCP (quindi protocollo numero 6) che utilizzano la porta 1863 per i messaggi in entrata. Combinando questi due valori posso stabilire che ogni volta che avrò un pacchetto con protocollo 6 e porta sorgente 1863, mi troverò di fronte ad un pacchetto che utilizza il protocollo *msnp*. E' proprio secondo questo criterio che si è creata una tabella definita dallo standard IANA all'interno della quale si trovano tutte le combinazioni conosciute di porte/protocollo per poter effettuare la conversione.



**Figura 6.2 – Protocol Conversion**

Sonicap include tutti i tipi presenti nello standard e da inoltre la possibilità di crearne di nuovi o di eliminarli.

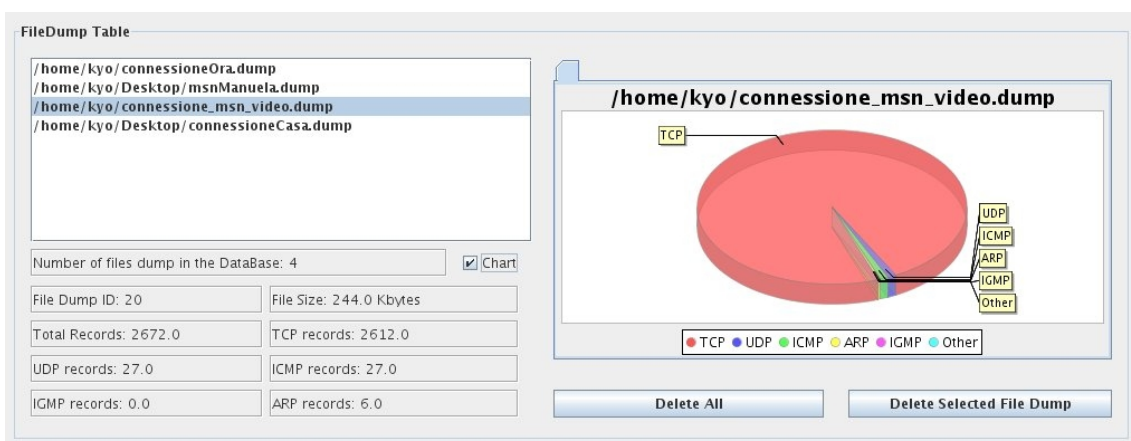
Classe GestoreDB	
<b>getProtocol()</b>	<i>Protocol getProtocol(int numero, String name, String protocollo, String type) {}</i>
Ritorna un oggetto Protocol già presente nel DataBase	

Classe GestoreDB	
<b>readProtocolType()</b>	<i>boolean readProtocolType(int numero, String tipo, String protocollo, String sigla, boolean flag) {}</i>
In base al valore di flag può o eseguire un controllo sull'esistenza del protocollo che prende come argomento, oppure inserire direttamente il nuovo protocollo nel DataBase. Ritorna true o false in base all'esito delle operazioni.	

Classe GestoreDB	
<b>deleteProtocolType()</b>	<i>void deleteProtocolType(Protocol p) {}</i>
Cancella dal DataBase il protocollo p che prende come argomento	

## 6.5 Screenshot dei vari form

### 6.5.1 Schermata DataBase tabella FileDump



**Figura 6.3 – Schermata File Dump**

Sul lato sinistro della schermata possiamo vedere la lista dei file dump

contenuti nel DataBase. Selezionandone uno, nei campi sottostanti appariranno le sue caratteristiche:

File Dump ID: identificativo univoco

Total Records: numero totale dei pacchetti contenuti

TCP Records: numero totale dei pacchetti TCP

UDP Records: numero totale dei pacchetti UDP

ICMP Records: numero totale dei pacchetti ICMP

IGMP Records: numero totale dei pacchetti IGMP

ARP Records: numero totale dei pacchetti ARP

File Size: dimensione del file dump in Kbytes

Sotto l'elenco è presente una casella selezionabile chiamata Chart, che se cliccata, genera il grafico del file dump selezionato.

## 6.5.2 Schermata statistiche

The screenshot shows the Sonicap application interface. The main window is titled "Statistics" and displays a table of network packets. The table has the following columns: ID, No., Time, MAC Sender, MAC Receiver, IP Sender, IP Receiver, and Protocol. The data rows show packets from 1431522 to 1431536, all dated 15/Jan/2008 at 21:47:41. The protocols listed include "unknown (6)", "surveyinst", and "cslistener".

Below the table, there are summary statistics for various protocols:

Packets Found:	2672	TCP:	2612	UDP:	27	ICMP:	27	IGMP:	0	ARP:	6
----------------	------	------	------	------	----	-------	----	-------	---	------	---

The search section includes a "File Dump" dropdown menu set to "/home/kyo/connessione\_msn\_video.dump". There are checkboxes for "TCP", "UDP", "ICMP", "IGMP", "ARP", and "Port Conversion". The "Protocol Type" is set to "0/Shirt Pocket netTunes - tcp". There are also fields for "Source" and "Destination" (both set to "MAC Address") and a "Search" button.

On the right side, there is a detailed view for "FRAME 9":

Arrival Time: 15/Jan/2008 - 21:47:41  
Frame Number: 9  
Frame Length: 1484 bytes  
Coloring Rule Foreground: 0,0,0  
Coloring Rule Background: 153,255,153

ETHERNET II, Src: 00:18:4d:0c:a0:6c, Dst: 00:13:d4:76:30:d5  
Source: 00:18:4d:0c:a0:6c  
Destination: 00:13:d4:76:30:d5  
Hop Limit: 116  
Priority: 0  
Offset: 0  
Identity Number: 20109  
Type: TCP

INTERNET PROTOCOL, Src: /78.12.21.114, Dst: /192.168.0.2  
Version: Ipv4  
Total Length: 1470  
Protocol: TCP (6)

**Figura 6.4 – Schermata Statistiche**



Nella parte alta dello schermo possiamo vedere l'etichetta con il nome del file che si sta visualizzando e il display dove sono riportati tutti i pacchetti catturati, accompagnato subito sotto dalle sue caratteristiche.

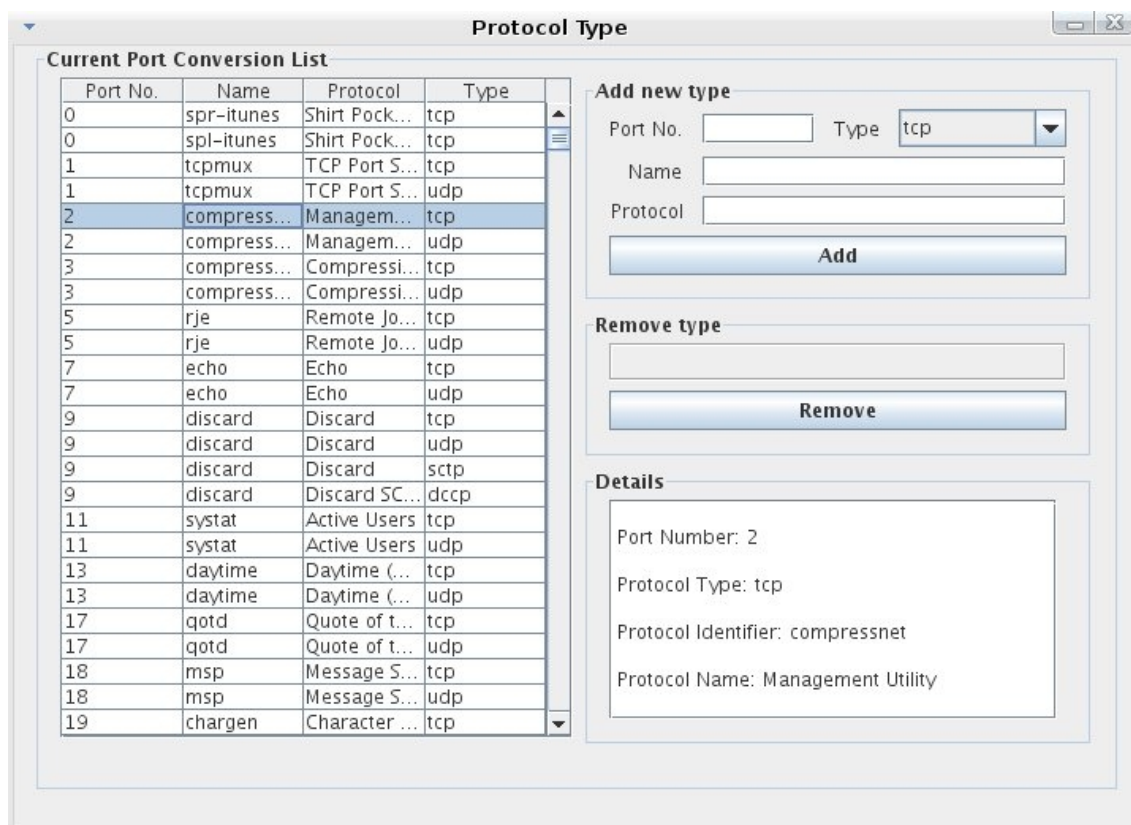
Packets Found: numero di pacchetti visualizzati nel display

TCP,UDP,ICMP,IGMP,ARP: numero di pacchetti per ogni categoria

Nella sezione inferiore abbiamo invece due diverse aree:

- l'area per l'inserimento dei criteri di ricerca, che permette la selezione del file e tutti i controlli precedentemente trattati
- l'area per la visualizzazione dei dettagli che mostra tutti i campi del pacchetto selezionato direttamente dal display

### 6.5.3 Schermata protocolli



**Figura 6.5 – Schermata Protocolli**

Sul lato sinistro della schermata è presente la tabella della *Port*

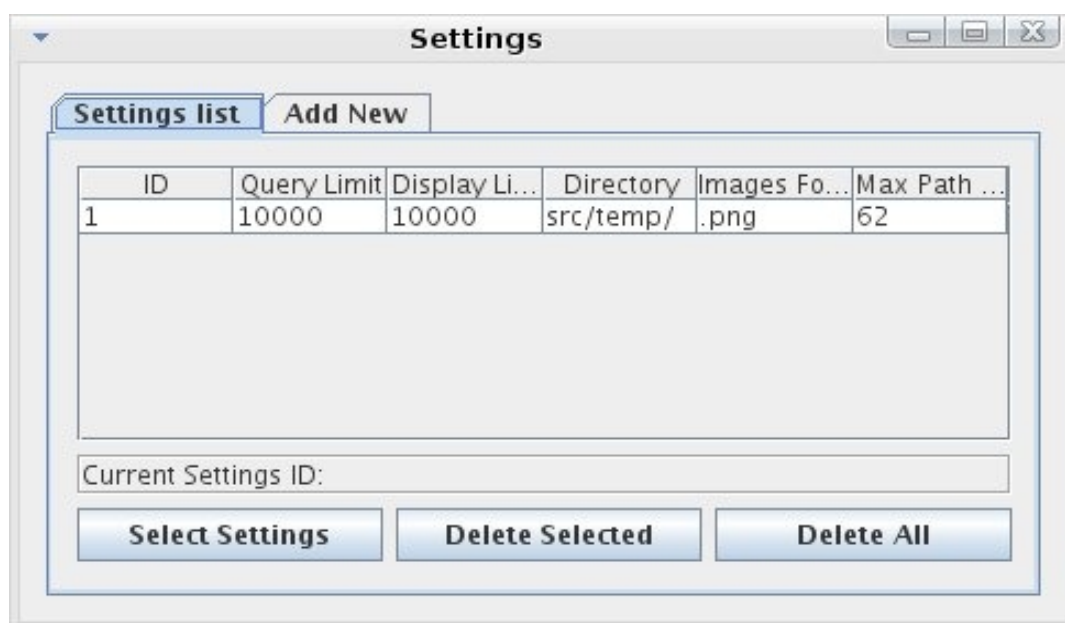
*Conversion List* ovvero la lista di porte/protocolli convertite in base allo standard IANA. Sulla destra abbiamo invece tre diversi form:

Add new type – permette l'inserimento di un nuovo protocollo

Remove type – permette l'eliminazione di un protocollo esistente nella lista

Details – mostra i dettagli del protocollo selezionato dalla lista

#### 6.5.4 Schermata settaggi



**Figura 6.6 – Schermata Settaggi**

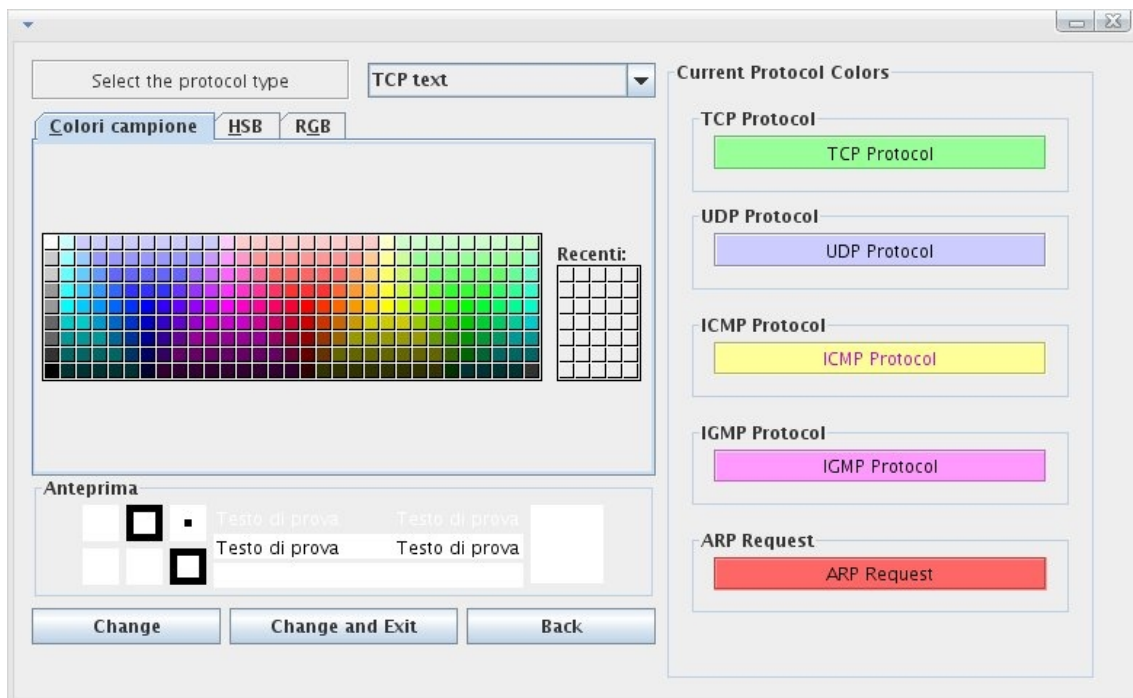
In questa schermata è possibile vedere tutti i tipi di settaggi che sono presenti nel DataBase e quale tra di loro è quello corrente tramite la label *Current Settings ID* che ne indica l'ID.

Select Settings – imposta il settaggio selezionato come corrente

Delete Selected – elimina il settaggio selezionato

Delete All – elimina tutti i settaggi e reimposta quello di default

## 6.5.5 Schermata colori



**Figura 6.7 – Schermata Colori**

Questa schermata mi permette di cambiare i colori dei protocolli che vengono visualizzati sul display.

Nella parte sinistra abbiamo la scelta del colore che può essere effettuata in tre modi:

Colori Campione – si seleziona un colore tra quelli più comunemente usati

HSB – si seleziona il colore direttamente dalla tavolozza potendo scegliere tra tutte le combinazioni possibili

RGB – inserendo il codice RGB del colore che si desidera

Sopra questo form è presente una combo box tramite la quale è possibile scegliere a quale tipo di pacchetto applicare il nuovo colore.

Nella parte destra troviamo invece l'anteprima di come appaiono i pacchetti con le impostazioni correnti.

# 7 – Conclusioni

---

## 7.1 Conclusioni

---

Lo sviluppo di Sonicap ha messo alla luce diverse problematiche legate sia all'utilizzo delle banche dati sia all'analisi del traffico di rete.

Parliamo prima di tutto delle librerie di cattura dei pacchetti che sono: *libpcap* e *winpcap*. Originariamente vi era solo la *libpcap*, sulla quale poi si baseranno la JPCAP e TcpDump, che era una libreria per la cattura del traffico di rete ma solo per sistemi Unix. Con la diffusione incalzante di windows fu necessario effettuare un *porting* di tale libreria chiamato Winpcap che purtroppo però non è ancora del tutto stabile. Lanciando programmi di *sniffing* di rete sotto ambiente Windows ci saranno sempre rischi legati al cambio di libreria.

Per quanto riguarda il trattamento dei dati, molto importante è il fatto che non vi sia un programma in grado di salvare i dati contenuti in un file dump in un database, o perlomeno non sia così facile da trovare nel web. E' molto strano che l'idea del salvataggio su una banca dati per migliorare le prestazioni non sia stata adottata dai programmi più conosciuti come Wireshark.

Tornando al concetto principale, si è dimostrato che Sonicap ha tempi di lettura migliori di programmi come Wireshark, occupa uno spazio di memoria proporzionato alle risorse disponibili e permette il salvataggio delle operazioni effettuate, è quindi possibile affermare che raggiunge l'obiettivo prefissato.

## 7.2 **Bibliografia**

---

<http://www.jfree.org/jfreechart/> sito ufficiale della libreria Java JFREECHART

<http://netresearch.ics.uci.edu/kfujii/jpcap/doc/> sito ufficiale della libreria Java JPCAP

<http://www.networksorcery.com/enp/default1002.htm> RFC Sourcebook dove vengono descritti i principali tipi di protocolli e molte delle RFC in uso

<http://www.wireshark.org/faq.html> sito ufficiale di Wireshark

<http://www.ubuntu-it.org/index.php?page=caratteristiche> sito ufficiale della distribuzione Linux Ubuntu

<http://www.tcpdump.org/#documentation> sito ufficiale di TcpDump

<http://java.sun.com/j2se/1.4.2/docs/> sito di supporto ufficiale della Sun per il Java 2 SDK 1.4.2

## 7.3 **Glossario termini**

---

*Host* - Si definisce host o end system ogni terminale collegato ad Internet. Gli host possono essere di diverso tipo, ad esempio computer, palmari, dispositivi mobili e così via, fino a includere web TV, dispositivi domestici e thin client.

*DataLink* - Il livello datalink è il livello 2 sia del modello ISO/OSI sia di

quello TCP/IP. Questo livello riceve pacchetti dal livello di rete e forma i frame che vengono passati al successivo livello fisico.

*MAC Address* - L'indirizzo MAC viene detto anche indirizzo fisico o indirizzo ethernet o indirizzo LAN, ed è un codice di 48 bit assegnato in modo univoco ad ogni scheda di rete ethernet prodotta al mondo.

*Broadcast* - Per broadcasting si intende la trasmissione di informazioni da un sistema trasmittente ad un insieme di sistemi riceventi non definito a priori.

*Unicast* - Nelle reti di calcolatori, un pacchetto destinato ad un solo computer, e l'indirizzo usato per inviare un tale pacchetto, è detto Unicast.

*Datagramma* - indica un messaggio, composto di un solo pacchetto, che contiene le informazioni necessarie per instradare se stesso dal mittente al destinatario.

*Ethernet* - nome di un protocollo per reti locali, comunemente utilizzato nelle LAN

*Sniffing della rete* - Si definisce sniffing l'attività di intercettazione passiva dei dati che transitano in una rete telematica