PARIMULO: CREDITS

RELATORE: Ch.mo Prof. Enoch Peserico Stecchini Negri De Salvi
CORRELATORE: Ing. Paolo Bertasi
LAUREANDO: Sebastian Daberdaku

Corso di Laurea Triennale in Ingegneria Informatica

A.A. 2009-2010

UNIVERSITÀ DEGLI STUDI DI PADOVA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Triennale in Ingegneria Informatica

TESI DI LAUREA

PARIMULO: CREDITS

RELATORE: Prof. Enoch Peserico Stecchini Negri De Salvi

CORRELATORE: Ing. Paolo Bertasi

LAUREANDO: Sebastian Daberdaku

A.A. 2009-2010

A mio padre, a mia madre
e a mia sorella.

Indice

Sommario

Lo sviluppo negli ultimi anni delle reti peer-to-peer ed in particolare del file sharing ha cambiato profondamente la percezione di Internet, scoprendo delle grandissime potenzialità. Basti pensare che, oggigiorno, client di file sharing come eMule e BitTorrent non mancano più in nessun calcolatore. La robustezza delle architetture peer-to-peer e le enormi funzionalità che offrono, altrimenti inaccessibili tramite singole macchine, ne fanno delle risorse immense.

Le potenzialità di una rete peer-to-peer sono direttamente proporzionali ai servizi offerti dai nodi che la compongono. Nel caso specifico del file sharing, più sono i file che un nodo condivide, più aumenta la qualità della rete. Al contrario, i comportamenti egoistici, la fanno peggiorare, minacciandone l'esistenza stessa. Pare subito evidente che serve un meccanismo che inciti gli utenti ad offrire più servizi possibile, in modo da aumentare sempre più le risorse della rete e prevenirne il collasso. A tale scopo servono i sistemi di crediti.

In questo elaborato illustreremo alcuni dei sistemi di crediti usati dai più noti client peer-to-peer, per poi seguire con l'implementazione del sistema di crediti del plugin Mulo. Inoltre, viene anche illustrato il meccanismo dell'identificazione sicura, necessario ai fini del sistema di crediti, il suo funzionamento e la sua implementazione.

Capitolo 1

Il progetto PariPari

In questo primo capitolo verrà esposto in breve il progetto **PariPari**, descrivendone le finalità e le caratteristiche fondamentali. Vengono inoltre presentate le principali caratteristiche della rete **eDonkey2000**, ed infine, quelle del modulo **Mulo**.

1.1 Il progetto PariPari

PariPari è una rete peer-to-peer serverless e multifunzionale, il software della quale è interamente scritto in Java. Essa si basa su una variante di Kademlia, garantisce l'anonimato dei suoi nodi, fornisce un sistema di crediti più intelligente di quelli offerti da reti come eDonkey2000, e, soprattutto, è multifunzionale.

La multifunzionalità è l'aspetto più peculiare di questo progetto, ed il suo scopo è quello di distribuire sulla rete tutti i più comuni servizi disponibili su Internet. Ciascuno di questi servizi è attivabile in base alle esigenze dell'utente: questo è reso possibile dall'organizzazione a plugin[1] di PariPari. Tale organizzazione è un vero punto di forza, poiché permette di realizzare servizi estremamente diversi ed allo stesso tempo integrati e scalabili.

Un'altra delle caratteristiche principali del progetto è la sua portabilità su tutte le principali piattaforme. Questo è stato reso possibile dalla programmazione in Java, in quanto, come sappiamo, i programmi Java non necessitano di ricompilazione del codice sorgente.

La modularità di PariPari, e cioè la sua organizzazione in plugin, permette di effettuare modifiche su una parte del progetto senza influenzare minimamente le

[1] Il plugin è un programma non autonomo che interagisce con un altro programma per ampliarne le funzioni.

Figura 1.1: Logo di PariPari.

altre parti. Per ogni plugin viene definita una API[2], la quale permette l'accesso alle sue funzionalità. I plugin hanno tutti scopi specifici, e dispongono ognuno di un gruppo di sviluppatori e tester dedicato. L'Extreme Programming è l'approccio che si cerca di attuare in ogni team: coppie di sviluppatori testano gli uni il codice degli altri.

Il plugin principale, che arbitra e fa da mediatore a tutti gli altri è il **Core**. Infatti, ogni plugin che vuole ottenere una risorsa, deve inviare a quest'ultimo una richiesta specifica. I plugin non possono comunicare direttamente tra di loro: essi devono per forza rivolgersi al Core. Se un plugin tentasse di aggirare il Core in modo da poter accedere alle risorse per via diretta, il **Security Manager**, implementato nel Core solleverebbe un'eccezione di sicurezza.

La gestione delle risorse da parte del Core avviene tramite un sistema di suddivisione delle risorse chiamato **Credits** (da non confondere con l'omonimo Credits che da il nome a questo elaborato). Questo sistema assegna ad ogni risorsa un certo valore, ed in base a tale valore il Core decide se una richiesta è attuabile o meno. In questo modo si evita la starvation[3] dei singoli plugin, offrendo una equa spartizione delle risorse.

[2]Le Application Programming Interface API (Interfaccia di Programmazione di un'Applicazione), sono ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per un determinato compito. È un metodo per ottenere un'astrazione, di solito tra l'hardware e il programmatore, o tra software a basso ed alto livello.

[3]Per starvation (termine inglese che tradotto letteralmente significa inedia) si intende l'impossibilità, da parte di un processo pronto all'esecuzione, di ottenere le risorse di cui necessita.

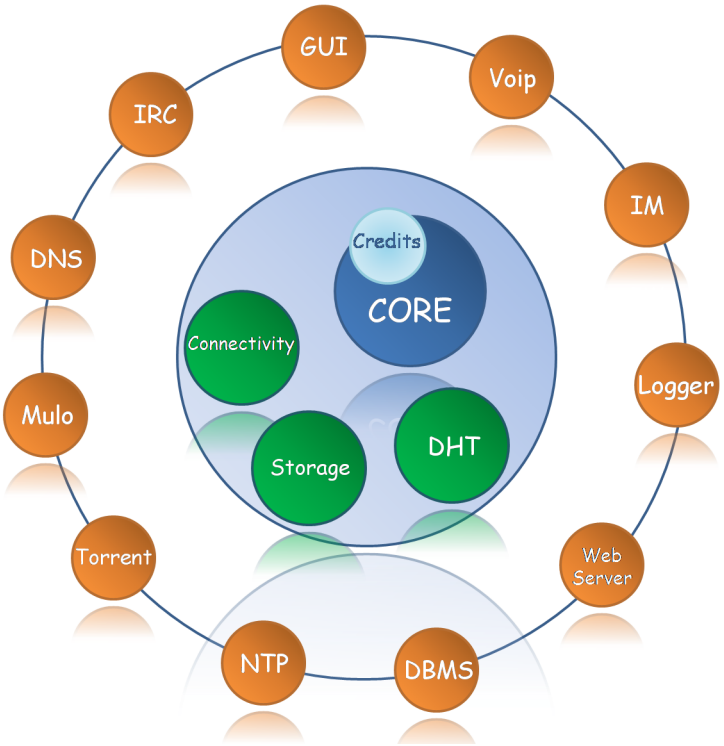

Figura 1.2: Architettura del progetto PariPari.

Due plugin molto importanti, che forniscono le risorse, sono **Connectivity** e **Storage**. Essi fanno parte della "cerchia interna". Connectivity instaura i socket[4] necessari per la comunicazione tra i peer della rete, invece Storage gestisce gli accessi alla memoria di massa, fornendo i file da utilizzare.

Tra i plugin che interagiscono direttamente con l'utente possiamo citare Mulo, Torrent, Voip[5], IRC[6], IM[7], DBMS[8], DHT[9] e WebServer.

[4]Nei sistemi operativi moderni, un socket è un'astrazione software progettata per poter utilizzare delle API standard e condivise per la trasmissione e la ricezione di dati attraverso una rete oppure come meccanismo di comunicazione tra processi.

[5]Voice over IP - è una tecnologia che rende possibile effettuare una conversazione telefonica sfruttando una connessione Internet o un'altra rete dedicata che utilizza il protocollo IP.

[6]Internet Relay Chat - è una forma di comunicazione istantanea (chat) su Internet.

[7]Instant Messaging - è un sistema di comunicazione solitamente client-server che consente di scambiare in tempo reale, frasi e brevi testi.

[8]Database Management System - è un sistema software progettato per consentire la creazione e manipolazione efficiente di database.

[9]Distributed Hash Table - è una classe di sistemi distribuiti decentralizzati che partiziona l'appartenenza di un set di chiavi tra i nodi partecipanti, e può inoltrare in maniera efficiente i messaggi all'unico proprietario di una determinata chiave. Ciascun nodo è l'analogo di un array slot in una hash table.

1.2 La rete eDonkey2000

In seguito vengono illustrate in breve le principali caratteristiche del protocollo eDonkey2000. Non essendo lo scopo di questo lavoro, si precisa subito che non verranno fornite le specifiche corrette e dettagliate di tale protocollo.

eDonkey2000 (detta anche semplicemente eDonkey, o ed2k) e una delle più popolari reti peer-to-peer. Viene implementata da software come eMule, aMule, Lphant, jMule, Shareaza, eDonkey2000. Tali client offrono agli utenti funzionalità avanzate come:

- Ricerca di file basata su un file di metadati (blocchi). L'utente può specificare il tipo di file e la sua dimensione.
- L'abilità di scaricare un file da diversi client: questa funzionalità riduce il tempo di download e di propagazione del file nella rete eDonkey.
- Controllo d'errore usando l'hash[10] MD4[11] per ogni blocco.

La rete eDonkey è composta da due entità, il server eDonkey, ed il software client. Per permettere ai client di trovare i file condivisi, il server eDonkey mantiene una lista di client (indirizzi IP) e dei file che loro si offrono di condividere. Il software client è preconfigurato con una lista di server eDonkey noti.

Quando il client istaura una connessione con un singolo server eDonkey, quest'ultimo gli assegna un identificatore a 4 byte, chiamato **client ID**. Esso identifica univocamente il client dal punto di vista del server, e rimane valido solo per la durata della connessione TCP[12] tra questi due. Se però il client ha un high ID, allora gli verrà sempre assegnato lo stesso ID da tutti i server, fino a quando il suo indirizzo IP non cambia. I client ID sono divisi due categorie: low ID e high ID. Tipicamente un server eDonkey assegna un low ID ad un client che non accetta connessioni in entrata. L'avere un low ID limita fortemente l'utilizzo della rete da parte del client, ed avvolte, può anche portare al rifiuto, da parte dei server, della richiesta di connessione. Al contrario un client con high ID non ha nessuna restrizione nell'utilizzo della rete. Tale ID viene calcolato in base al suo indirizzo IP. Un high ID viene dato ad un client che permette agli altri client di connettersi liberamente alla propria macchina (tipicamente usando la porta TCP 4662).

[10] La funzione hash è una funzione non iniettiva che mappa una stringa di lunghezza arbitraria in una stringa di lunghezza predefinita. Esistono numerosi algoritmi che realizzano funzioni hash con particolari proprietà che dipendono dall'applicazione.

[11] L'MD4 è una funzione crittografica di hashing.

[12] Transmission Control Protocol - protocollo del livello di trasporto.

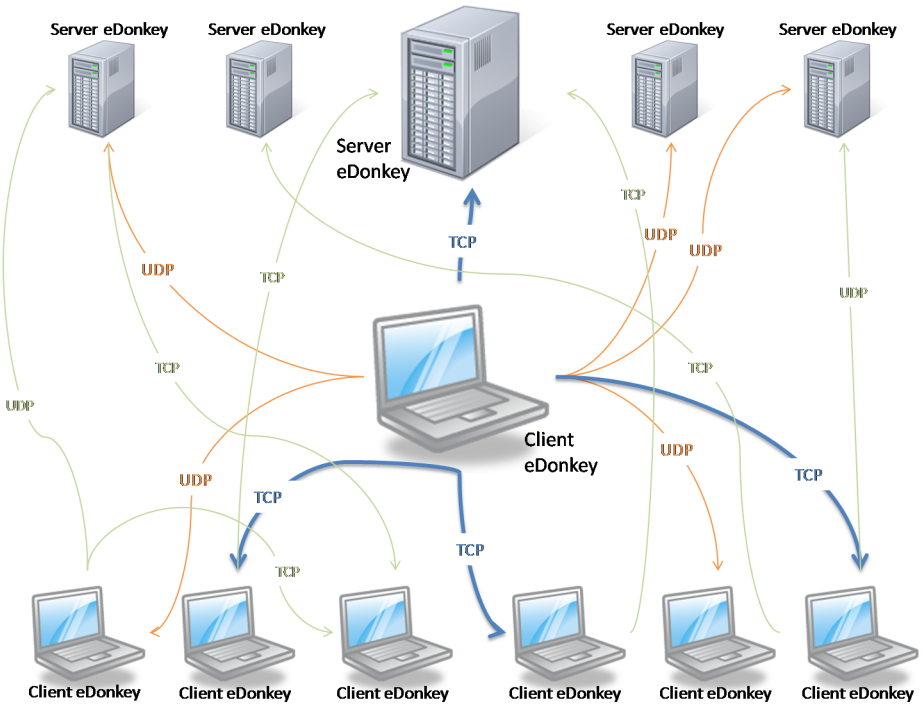

Figura 1.3: Struttura della rete ed2k.

Successivamente alla connessione iniziale, il client pubblica la lista dei suoi file condivisi sul server.

Ogni client nella rete eDonkey mantiene una lista di server su file, lista ottenibile in diversi modi:

1. dal codice sorgente del client stesso; (preconfigurato);
2. da siti internet noti;
3. da altri server eDonkey;
4. da altri client eDonkey.

Per effettuare una ricerca, un client specifica una o più parole chiave. Le interrogazioni da parte dei client possono essere molto complesse, includendo nei vincoli dimensione, estensione e disponibilità dei file. Al ricevere di una tale richiesta, il server costruisce una lista di file che corrispondono ai vincoli specificati, e la invia al client. Dopo aver scelto un certo file dalla lista, il client genera una nuova richiesta. A questa nuova richiesta il server risponde con una lista di sorgenti che condividono il file desiderato. Il client può inoltrare la stessa ricerca a tutti (o anche solo ad alcuni) gli altri server eDonkey, usando il protocollo

UDP[13]. Dopo aver ricevuto la lista delle sorgenti, il client inizializza il download del file direttamente da queste sorgenti, bypassando il server.

Nel protocollo eDonkey i file sono divisi in blocchi da 9500 KiB, per ognuno dei quali viene calcolato un checksum[14] MD4. I chechksum relativi ai blocchi di un dato file vengono poi usati per calcolare un nuovo checksum MD4, il quale funge da identificatore e viene chiamato **file ID**.

Appena un blocco viene scaricato, esso viene anche automaticamente condiviso sulla rete eDonkey: questo permette di diffondere le sorgenti dei file molto velocemente. Inoltre, la propagazione dei checksum MD4 e dei blocchi dei file tra i client, permette anche di rilevare e bloccare in modo molto efficiente i file corrotti.

1.3 Il plugin Mulo

Il plugin **Mulo** ha come obiettivo specifico la condivisone dei file attraverso la rete eDonkey, della quale abbiamo già parlato. La meta principale dei progettisti di Mulo, è, senza alcun dubbio, quella di creare un plugin stabile ed efficiente, capace di poter competere con il più famoso client della rete eDonkey: eMule.

La mancanza di una documentazione ufficiale del protocollo eDonkey (per quello che ne sappiamo non esiste una documentazione ufficiale, o se esiste, non è mai stata resa pubblica) ha reso molte volte necessario ricorrere al codice open - source del client eMule per capire il corretto comportamento che un client deve avere. Ovviamente, Mulo non vuole essere una mera copia del client eMule, (ricordiamo che esiste già un client eDonkey realizzato in Java: jMule), bensì un degno concorrente. Infatti, gli abili programmatori di Mulo sono riusciti a fare meglio:

- È stato implementato un efficace motore di download con una più efficiente gestione delle fasi iniziali e finali (si scaricano per prime le parti che permettono la rapida visualizzazione dei file multimediali; le parti assegnate alle fonti lente possono essere riassegnate, in modo da evitare un eccessivo rallentamento alla fine).

[13]User Datagram Protocol - protocollo del livello di trasporto.

[14]Checksum, tradotto letteralmente significa somma di controllo. È una sequenza di bit che viene utilizzata per verificare l'integrità di un dato o di un messaggio che può subire alterazioni.

- È stato eliminato il controllo finale dell'hash dell'intero file (operazione che può richiedere anche alcuni secondi, e che, tra latro, non è strettamente necessaria).
- Sono state implementate code di upload più veloci, in grado di poter rifiutare l'upload a singoli peer.
- Mulo può connettersi in parallelo a più server per ottenere un maggior numero di fonti.

Figura 1.4: Logo di Mulo.

Dunque Mulo è il plugin che permete agli utenti di partecipare alle reti eDonkey e Kademlia (quest'ultima è stata integrata da poco). Kademlia è una rete completamente (sovrapposta a quella eDonkey) distribuita (basata su DHT), e le recenti versioni di eMule (e di altri client) si affidano pesantemente ad essa per scambiare informazioni e sostituire i server nel compito di ricerca di file e di fonti per un file.

Alcune delle funzionalità degne di nota che Mulo offre sono:

- A.I.C.H. (Advanced Intelligent Corruption Handling) per controllare che le parti scaricate siano prive di errori, ed in caso recuperare con un nuovo download quelle corrotte.
- Supporto dei MultiPacket.
- Supporto dei pacchetti compressi.

- Banning Intelligente.
- Ranking intelligente dei server.
- Login intelligente.
- Hashing parallelo.
- Supporto dell'identificazione sicura.
- Supporto del sistema dei crediti per gestire le code di upload.

L'integrazione all'estensione eMule però non è ancora completa, infatti non è ancora supportata l'estensione a 64 bit, quindi per ora si possono condividere solo file di circa 4GB.

Attualmente Mulo è composto da 52 file per un totale di oltre 35'000 righe di codice suddivise in 228 classi Java, questo dopo aver effettuato, come sempre, un attento refactoring del codice.

1.3.1 Interazione di Mulo con gli altri plugin

Diamo qui una breve panoramica generale delle interazioni del plugin Mulo con gli altri plugin di PariPari, senza entrare nel dettaglio.

Core

Mulo, segue alla lettera la filosofia di PariPari, e usa il Core (T.A.L.P.A.) per tutte le richieste di risorse ai plugin della cerchia interna.

Tutti i thread di Mulo sono istanze di PariPariRunnable. All'interno di Mulo la creazione di un thread dev'essere eseguita chiamando il metodo MuloThread.startThread(PariPariRunnable runnable, String name). Qui runnable è un oggetto di una classe che estende PariPariRunnable, che è la controparte PariPari dell'interfaccia Java Runnable. Se le classi Runnable devono implementare il metodo run(), che è il metodo che viene chiamato all'avvio del nuovo thread,le classi PariPariRunnable devono allo stesso scopo implementare il metodo go().

Mulo è pienamente conforme al Security Manager del Core.

Attualmente la GUI[15] permette di effettuare delle semplici operazioni quali la connessione, la ricerca semplice (non avanzata) e l'inizio dei download. Gli altri comandi devono essere dati dalla riga di comando.

[15]Graphical User Interface - interfaccia grafica.

Connectivity

Connectivity viene usato per ottenere tutti i tipi di socket necessari al funzionamento di Mulo. Si possono creare:

- socket TCP ed UDP sia verso server che i peer;
- un socket TCP, aperto su una porta a scelta, per essere raggiungibili dagli altri peer;
- connessione HTTP per scaricare la lista dei server.

Le bande di download/upload per i socket dei server e le connessioni TCP sono configurabili nel file config.xml (i parametri di default sono 1MB/s e 20KB/s).

Secondo le politiche di Connectivity, ad ogni connessione accettata alla porta server vengono dati 5KB/s di banda. Si possono aprire al massimo 600 socket, per una banda totale di 4MB/s. Questi valori sono soggetti a revisione, e possono essere modificati nel file di configurazione di Connectivity.

Le interfacce richieste sono:

- LimitedSocketAPI: crea un socket per effettuare connessioni TCP;
- LimitedDatagramSocketAPI: crea un socket per effettuare connessioni UDP;
- LimitedServerSocketAPI: crea un socket per gestire tutte le connessioni entranti provenienti dalla rete;
- LimitedHttpURLConnectionAPI: utilizzato da Mulo per scaricare le liste server.

Storage

Per il funzionamento corretto di Mulo servono ovviamente degli oggetti che permettano di manipolare i file XML di configurazione, di log e di salvataggio dati, nonché per gestire i file in download e upload. A questo scopo viene utilizzato lo Storage. Le interfacce richieste sono:

- FileAPI: utilizzato come rappresentazione astratta di file e directory;
- RandomAccessFileAPI: permette di accedere (scrivere e/o leggere) al file in modo casuale, partendo da un punto assegnato;
- FileInputStreamAPI e FileOutputStreamAPI: vengono usati per i file XML, per il file di log, per i download e gli upload e per ottenere gli hash dei file.

Capitolo 2

I Crediti

2.1 Introduzione sui Sistemi di Crediti

Quando richiediamo un file da un dato peer veniamo messi nella sua coda di upload. L'attesa nella coda varia in base ad alcuni fattori, uno dei quali è il *credito* che noi abbiamo nei confronti di quel peer. Il sistema di crediti è un sistema meritocratico che ha come scopo quello di combattere gli utenti "leecher" (sanguisughe), i quali vogliono solo scaricare dati, senza dare mai nulla in cambio, e, dall'altro canto, premiare gli utenti che ci hanno fatto scaricare in passato.

Il principio è il seguente: *più fai scaricare, più crediti accumuli.*

Se abbiamo credito nei confronti di un dato client, quando richiederemo un file a quest'ultimo, verremo privilegiati nella posizione della sua coda di upload. Infatti, in base al nostro credito, potremmo scavalcare gli altri utenti in coda (quelli che hanno meno credito di noi), anche se stanno aspettando da più tempo. Più crediti uno ha, più in alto si ritroverà nelle liste di upload, prima potrà scaricare i dati che gli interessano. Quando un utente finisce il suo turno di scaricare, ritorna in fondo alla coda di upload, ma comunque la può risalire velocemente se ha ancora credito. In sostanza, nello stesso tempo, un utente con credito riesce a scaricare di più di uno con meno credito, o senza credito.

Facciamo un esempio pratico:

A, B e C cedono dei file a X, perciò i tre client vanteranno dei crediti nei confronti di X. Ipotizziamo (molto semplicisticamente), che l'ammontare dei crediti sia rispettivamente di:

- A $\Longrightarrow$ 150
- B $\Longrightarrow$ 100
- C $\Longrightarrow$ 200

Ipotizziamo ancora che A, B e C richiedano contemporaneamente a X lo stesso file, ad esempio il file *"paripari.txt"*. Volendo ancora ipotizzare, per semplificare il tutto, che per quel file la coda sia costituita esclusivamente dai client A, B e C, la coda scaturirà dai crediti che hanno i client, e nella fattispecie considerata sarà:

- C $\Rightarrow$ QR[1]: 1 $\Longleftrightarrow$ E fra l'altro comincerà a scaricare subito.
- A $\Rightarrow$ QR: 2 $\Longleftrightarrow$ Dato che ha un client con più crediti davanti.
- B $\Rightarrow$ QR: 3 $\Longleftrightarrow$ Dato che ha due client con più crediti davanti.

2.2 Crediti e coda di eMule

Come abbiamo fatto spesso in passato per altre problematiche, ricorriamo anche questa volta allo studio del comportamento del client **eMule** (versione ufficiale) riguardo ai crediti.

In eMule il sistema di download è gestito da **code**. Ogni client eMule crea in automatico una **lista di attesa** (coda) in cui inserisce tutte le richieste di upload; ciò significa che ogni utente che chiede un file sarà messo in una lista d'attesa stilata in base ad un punteggio attribuito dal client eMule. Nella versione ufficiale del client eMule c'è **un'unica coda di upload** per tutti gli utenti che vogliono scaricare.

Quando il nostro client scarica un file siamo noi invece ad entrare in coda dagli altri (il famoso QR indica in che posto siete nella coda di upload di quell'utente). Quando iniziamo a scaricare da qualcuno significa che siamo alla **posizione zero** della sua coda (QR: 0). Se qualcuno sta scaricando da noi significa che è alla **posizione zero** della nostra coda di upload e quindi tocca a lui. Ricordiamo che eMule **per ogni utente** tiene traccia di tutti i megabyte scaricati e forniti per un periodo di **150 giorni** (5 mesi) dall'ultima volta che lo abbiamo incontrato. In poche parole si hanno **5 mesi per "riscuotere"** i crediti prima che lui si

[1]Queue Rank - indica la posizione specifica del nostro client nella coda di un'altro client, dal quale vogliamo scaricare.

dimentichi di noi. Ad ogni avvio vengono cancellati gli utenti scaduti (viene utilizzato per memorizzare le informazioni il file clients.met dentro la cartella /config di eMule).

2.2.1 Coda di eMule: come funziona?

Se stanno scaricando da noi N utenti e uno di essi completa il suo blocco di 9,28MB, eMule prende l'utente con il punteggio più alto fra tutti quelli che stanno aspettando da noi. Ogni client in coda ha infatti un punteggio, e quello col punteggio più alto sta in cima alla coda ed è il primo ad essere servito.

Se vogliamo sapere quanto è alto il nostro punteggio dovremmo relazionarlo con tutti gli altri punteggi della lista di attesa che stiamo considerando. Se per una certa lista per l'utente X abbiamo un punteggio Y e siamo i primi nella lista, lo stesso punteggio potrebbe essere equipollente alla posizione numero 100 di un'altra lista (utente Z) perché in quella lista magari ci sono persone con un punteggio più alto del nostro. Conoscere il nostro punteggio infatti non porta a nessuna informazione utile perché dovremmo relazionarlo ai nostri concorrenti caso per caso, sapere invece il nostro posto di attesa (QR) dà un'informazione molto più immediata ed utile.

Il punteggio viene così calcolato:

punteggio = tempo attesa in coda * priorità file condiviso * crediti,
(il simbolo * indica le moltiplicazioni)

Il **tempo attesa in coda** è il numero di secondi che quell'utente sta aspettando di scaricare. Ogni secondo è un punto e quindi 5 minuti (300 secondi) sono 300 punti.

La priorità file condiviso varia a seconda del file richiesto:

- Release 1.8
- Alta 0.9
- Normale 0.7
- Bassa 0.6
- Molto bassa 0.2

Se si lascia la modalità automatica di gestione delle priorità di upload (la scelta migliore), eMule utilizzerà solo i valori Alta[+], Media[=] e Bassa[-]. Molto bassa e release sono solo assegnabili manualmente.
Ora facciamo un esempio di un utente che sta in coda da noi **10 minuti**, senza crediti, e che ha richiesto un file in priorità **Alta**.
Il punteggio sarà: 600 * 0.9 * 1 = 540
Prendiamo invece un client che sta in coda da **30 minuti**, sempre senza crediti, ma che richiede un file in priorità **Bassa**: 1800 * 0.6 * 1 = 1080 punti

2.2.2 I Crediti di eMule (il Sistema di Crediti Ufficiale)

ll valore dei crediti varia **da 1 a 10** ed eMule usa 2 formule per calcolarlo. Questo valore viene calcolato dal **NOSTRO** client per favorire o penalizzare gli **ALTRI** utenti che vogliono scaricare da noi. Allo stesso modo ogni utente a cui siamo in coda per scaricare un file effettuerà il calcolo per decidere quando sarà il nostro turno.

Il riconoscimento dei vari client viene effettuato attraverso l'**userhash** ed il sistema dell'**identificazione sicura** (dei quali parleremo più avanti) quindi non è soggetto ad errori.

FORMULA1

$$crediti = \frac{byte\ ricevuti * 2}{byte\ inviati}$$

Ad esempio se da un client abbiamo ricevuto in passato 10MB e gli abbiamo inviato 1MB:
$crediti = \frac{10 * 2}{1} = 20$ (cioè 10 dato che il massimo è 10)

FORMULA2

$$crediti = \sqrt{Megabyte\ ricevuti\ +\ 2}$$

Ad esempio se in passato abbiamo ricevuto da lui 10MB, il risultato è:
$crediti = \sqrt{10\ +\ 2} \approx 3.5$

eMule sceglie il valore **più basso fra i due** calcolati. Nel nostro caso il valore crediti è 3.5, in quanto la formula 2 ha riportato un valore più basso di quanto ha riportato la formula 1. Ci sono anche altri vincoli nel calcolo, come ad

esempio il fatto che se abbiamo scaricato meno di 1MB da quell'utente, gli diamo comunque crediti = 1.

Che differenza c'è fra crediti = 1 e crediti = 10? Poiché il valore dei crediti viene moltiplicato per ottenere il punteggio di quell'utente, avere un punteggio dieci volte maggiore significa che lui scala la vostra coda di upload più in fretta del normale, scarica 9,28MB, torna in fondo e la risale velocemente, quindi nel complesso scarica più spesso e quindi più megabyte nello stesso tempo.

Esempio:
Consideriamo "priorità file condiviso" = 1 e ingresso in coda di utenti simultaneo, quindi tutti con tempo attesa = 1:

Tempo attesa (s)	Punteggio con crediti=1	Punteggio con crediti=2	Punteggio con crediti=10
1	1	2	10
2	2	4	20
5	5	10	50
60	60	120	600
(= 10 min) 600	600	1200	6000
(= 1 ora) 3600	3600	7200	(= 10 ore) 36000

Tabella 2.1: Esempio di utilizzo dei crediti.

Come si vede di fatto **i crediti "simulano" un tempo di attesa più lungo in coda**, e quindi favoriscono lo scaricamento: 1 ora di attesa per l'utente con crediti = 10 equivale a 10 ore per un utente con crediti = 1.
Esempio:

- stiamo uppando a 25KB/s con 6 slot, quindi circa 4KB/slot
- 100 utenti con crediti = 1 entrano in coda da noi nello stesso momento insieme ad un utente con crediti = 10
- non ci sono altri utenti in coda
- abbiamo una media di 7MB trasferiti per sessione (è difficile arrivare a 9)

Con queste ipotesi sono necessarie circa 8 ore per dare 7MB ad ogni utente ma nello stesso tempo quello che aveva crediti = 10 **ha scaricato 10 volte più degli altri**, cioè 70 MB perché ha sorpassato per 10 volte tutti gli altri che aspettavano.

2.3 Alcune considerazioni sui Crediti

Dove salvare i crediti? Questa è una domanda lecita, e la risposta che diamo è la seguente: *non nel calcolatore del client a cui appartengono!* Anche se può sembrare strano e poco naturale, questo è l'unico modo che si ha per impedire che gli utenti *"barino"*, e cioè che manomettano i propri crediti. Per convincere i lettori possiamo citare come esempio il noto client **Kazaa** (a volte scritto anche KaZaA) della rete **FastTrack**, che usa il sistema di crediti *Participation Level* (livello di partecipazione). Il credito dell'utente viene salvato sulla macchina stessa dove gira il software client. Poco dopo l'avvento di Kazaa, vennero fuori innumerevoli "trucchi" per "barare" ed aumentare il proprio credito. Alcuni, molto banali, consistevano nel scaricare i file condivisi dall'utente stesso (il che è l'equivalente di copiare un file da una cartella al altra). Altri trucchi più sofisticati consistevano in modifiche del codice sorgente del client, che permettevano di impostare a piacere il Participation Level (come nel client Kazaa Lite).

L'uso di un sistema di crediti può avere alcuni vantaggi notevoli. Analizziamoli brevemente.

- Usando un sistema di crediti, se due client hanno parti diverse dello stesso file, essi possono scambiarseli in modo abbastanza efficiente.
- Sono scoraggiati gli utenti "leecher".
- Vengono incoraggiati i client "buoni" che condividono molti file.

Però vanno elencati anche alcuni possibili punti sfavorevoli:

- Vengono considerati i crediti solo su coppie di client, in questo modo non si può garantire che un utente che fa scaricare molto agli altri venga sempre premiato.
- Sembrano essere svantaggiati gli utenti che condividono file rari e di piccole dimensioni. Infatti risulta essere più conveniente condividere un file video di grandi dimensioni, piuttosto che un rarissimo e ricercatissimo album musicale.
- Non è incentivata la condivisione di nuovi file. Un utente ha più interesse a condividere un file già in rete, ma ricercato, piuttosto che un file nuovo.

- C'è anche chi azzarda l'ipotesi che il sistema di crediti possa portare alla scomparsa inesorabile dei file dalla rete. Dopo un certo periodo iniziale in cui un file viene condiviso, scaricato e diffuso, lentamente viene rimosso dai file condivisi di tutti i client, e di fatto si estingue.

Da primo impatto i punti sfavorevoli sembrano molto allarmanti, e danno subito spunto alla domanda: ma siamo sicuri che il gioco valga la candela? Dobbiamo sempre tenere presente una cosa: le applicazioni peer-to-peer di file sharing beneficiano dalla scalabilità e le prestazioni dei sistemi peer-to-peer decentralizzati, ma sono vulnerabili ai comportamenti egoistici. Fino ad ora, i sistemi di crediti di alcuni client come eMule e Bittorrent, si sono rivelati abbastanza affidabili e robusti.

Esiste un sistema di crediti migliore di quello sopra descritto? Un'idea molto ingenua sarebbe quella di voler creare un sistema di credito che funzioni come un punteggio globale (il classico highscore dei videogiochi), ma per fare ciò, serve un'autorità globale, che controlli le quantità di dati scambiate tra peer, centralizzando in qualche modo la rete, il che va contro la filosofia stessa su cui si basa PariPari, e cioè la decentralizzazione. Inoltre, la condivisione dei file non sarebbe più *"condivisione"*, ma bensì *"compravendita"*, e i crediti diventerebbero determinanti alla condivisione. Quindi possiamo "bocciare" questa idea.

Notiamo quindi che il sistema dei crediti è una questione tutt'altro che banale, ed il dibattito a riguardo è sempre aperto. Probabilmente, la soluzione migliore al problema non è tecnica, ma sociale: e cioè *promuovere una coscienza del file sharing*. Come facilmente intuibile, quest'ultima opzione richiede anni ed anni.

2.3.1 Ultime considerazioni

Tutti i client che ci inviano dati, devono venire ricompensati dal meccanismo dei crediti. Non importa se il client supporta o meno il sistema dei crediti. Ovviamente, client privi del supporto per i crediti non ci forniranno nessun credito quando effettuiamo degli upload verso di loro. I crediti non sono globali: essi sono scambiati esclusivamente tra coppie di client. Per identificarne il client proprietario viene usato l'userhash[2], che è univoco (non viene usato ne l'IP, ne l'ed2kid). I crediti di nostra proprietà sono salvati dal client che ci concede il credito. Questo previene manomissioni ai crediti. Un client non può visualizzare i propri crediti, perché **il valore è diverso per ogni utente con cui esso è**

[2]Vedi UserHash a pag. 42.

in contatto ed inoltre è completamente inutile: tale valore deve sempre essere relazionato con il credito degli altri utenti.

2.4 Sistemi di Crediti

Elenchiamo ora alcuni di principali sistemi di crediti, usati non solo dalla versione ufficiale di eMule, ma anche da alcui mod[3].

2.4.1 Il Sistema di Crediti Ufficiale

Questo è il sistema utilizzato dalla versione ufficiale di eMule (è stato già discusso in precedenza, ma lo rielenchiamo sinteticamente).

Vengono calcolati due modificatori:

- Ratio1 = $\frac{byte\ ricevuti * 2}{byte\ inviati}$
- Ratio2 = $\sqrt{Megabyte\ ricevuti\ +\ 2}$

Vengono confrontati i due valori così ottenuti, ed il più piccolo viene scelto come modificatore. Oltre a questa regola esistono anche alcuni casi particolari:

- if Totale Upload $<$ 1MB then Modificatore = 1;
- if Totale Download = 0 then Modificatore = 10;
- if Modificatore $>$ 10 then Modificatore = 10;
- if Modificatore $<$ 1 then Modificatore = 1;

2.4.2 Peace Credits

Molto simile a quello ufficiale. Il codice utilizzato è più semplice ma anche più lento, e quindi non lo prendiamo in considerazione.

[3]Le mod sono le versioni modificate di un programma, spesso anche non-ufficiali.

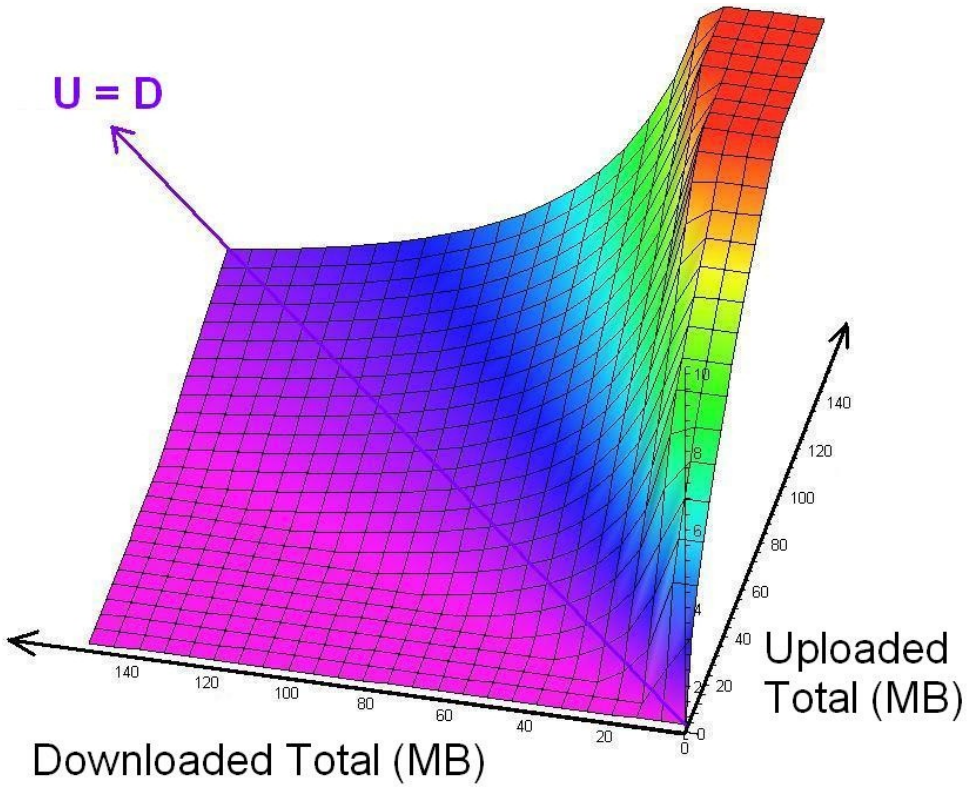

Figura 2.1: Il Sistema di Crediti Ufficiale.

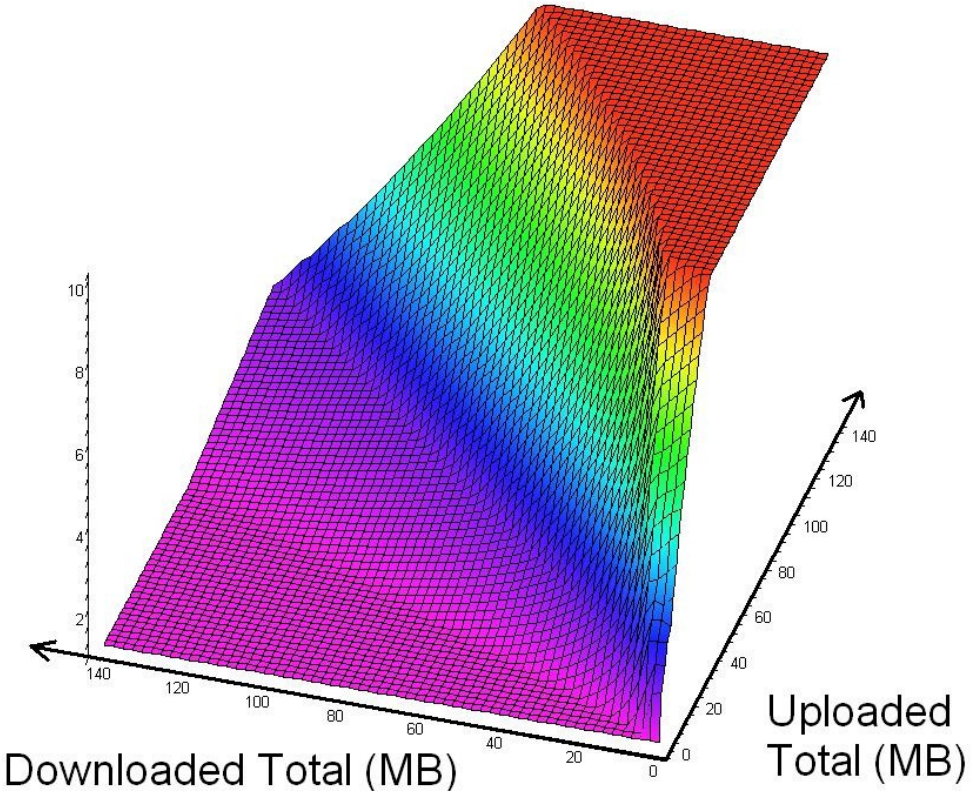

Figura 2.2: Il Sistema di Crediti Xman.

2.4.3 Xman improved Credit System

Questo sistema è un miglioramento del sistema di crediti ufficiale. Premia i client che ci danno un download con priorità alta. Il client ottiene un fattore bonus. D'altra parte, i client ai quali noi diamo molti dati, e che non ci danno nulla in cambio, li penalizziamo per la sessione corrente.

La formula per il bonus positivo è:

$$Bonus = \frac{download - upload}{10485760} \quad - \quad \frac{1.0}{\frac{download}{10485760}}$$

Il massimo valore del modificatore è 10 (come nella versione ufficiale).

Esempio (con errore di circa 1 chunk[4]):

Download	**Upload**	**Versione Ufficiale**	**Xman improved**
10MB	1MB	3.46	3.46 + bonus:0
20MB	11MB	3.63	3.63 + bonus:0
30MB	21MB	2.86	2.86 + bonus:0.2
90MB	81MB	2.22	2.22 + bonus:0.7
50MB	20MB	5.0	5.0 + bonus:2.2
90MB	50MB	3.6	3.6 + bonus:3.7
120MB	80MB	3.0	3.0 + bonus:3.8

Tabella 2.2: Esempio crediti Xman.

Un client può ricevere un bonus negativo di 0.1 nella sessione corrente, se noi gli abbiamo dato un chunk in più, ed anche alla completa valutazione del rapporto download/upload se non ci ha dato nulla in cambio. Può ricevere un bonus negativo di 0.2 se noi gli abbiano dato più di due chunk nella sessione corrente, ed anche alla completa valutazione del rapporto download/upload senza aver ottenuto nulla in cambio.

2.4.4 The Magic Angel Credit System

Il sistema di crediti **Magic Angel** si basa sulla versione ufficiale. Le differenze sono le seguenti:

[4]Suddivisione del file in porzioni non più grandi di 180 KiB

- da Credito per Totale Upload maggiore di 1.65MB (nella versione ufficiale 1.00MB);
- il valore minimo che il modificatore può assumere è 0.1 (nella versione ufficiale 1.0);
- il valore massimo che il modificatore può assumere è 50.0 (nella versione ufficiale 10.0).

2.4.5 The Magic Angel + Credit System

Il sistema di crediti **Magic Angel +** si basa sul sistema di crediti Magic Angel. A differenza di quest'ultimo, però, aumenta un po' il modificatore se il client con cui stiamo scambiando dati ci invia più dati di quelli che noi mandiamo a lui. Se la differenza tra i dati ricevuti e quelli inviati è:

- $<$ 7MB: modificatore + 0.3
- $<$ 15MB: modificatore + 1.0
- $<$ 30MB: modificatore + 2.0
- $\geq$ 30MB: modificatore + 3.0

2.4.6 Ratio Credit System

Si basa sul sistema di crediti ufficiale, ma ha modificatore minimo 0.1 e non 1.0. Il credito con cui vengono ricompensati gli uploader viene moltiplicato x2, x4,x16.

Ratio Credit System - L'implementazione di Stulle

Contrariamente al suo nome, non si basa sul rapporto Upload/Download, ma sui valori di Upload e Download, e la loro differenza. Non c'è una formula singola e semplice. Il credito di default è 1. Progressivamente vengono puniti i client con upload verso di noi vicino a zero ($\leq$ 1MB), ed invece i client che, pur avendoci uploadato dei dati, non hanno ricevuto ancora nulla in cambio ($\leq$ 1MB), vengono favoriti. Per i client dai quali abbiamo scaricato più di 1 MB e meno di 9 MB garantisce credito minimo: DownloadTotale/9. Per i client dai quali abbiamo scaricato più di 9 MB garantisce credito minimo $0.7 + \sqrt{(DownloadTotal + 1)/10}$ (esempio, se ho scaricato 99MB da un client, il suo credito sarà 1.7). Con un rapporto bilanciato, il credito aumenta proporzionale a $\sqrt{(DownloadTotal + 1)}$. Il

creditpo può essere ulteriormente aumentato/decrementato in base alla differenza di upload e dowload. Diversamente dagli altri sistemi di crediti, quest'ultimo non ha valori limite.

2.4.7 Neo Credit System

Il valore standard assegnato è 1. I valori degli Upload/Download sono in byte. Se il client non ha un file che a noi serve, prende in ogni caso credito 1. Se l'utente non supporta l'identificazione sicura, vengono considerati solo i trasferimenti della sessione corrente.

Se il risultato di (Upload - Download) è maggiore di 1 allora la formula è:

- In caso di fallito/no SI[5] $\Longrightarrow \left(\frac{9728000 * 2}{Upload - Download}\right)^2$
- In caso SI valido $\Longrightarrow \left(\frac{4}{Upload - Download}\right)^2$

Il valore massimo accettato è 1 invece il minimo è 0.1.

Se il risultato di (Upload - Download) è minore di -1 allora la formula è:

- In caso di fallito/no SI $\Longrightarrow \left(\frac{Download - Upload}{9728000}\right)^2$
- In caso SI valido $\Longrightarrow 2$.

Il massimo valore accettato è 2, invece il più piccolo è 1.

2.4.8 Pawcio Credit System

- Range da 1.0 a 100.0.
- Multiplicatore di 3 (invece di 2) → ratio = 3 * Downloaded / Uploaded.
- Per i nuovi client (downloaded e uploaded meno di 1MB) → ratio = 3.0 (invece di 1.0).
- Se abbiamo ricevuto più di 1MB da qualcuno, senza aver dato nulla in cambio, quel client ottiene credito = 100.

[5]Secure Identification - Identificazione Sicura, vedi Cap. 3.

		Downloaded to me												
		1	2	3	5	8	13	21	34	55	89	144	233	377
U	1	1.00	2.00	3.00	5.00	8.00	13.00	21.00	34.00	55.00	89.00	144.00	233.00	377.00
p	2	0.71	1.73	3.00	4.18	5.45	7.06	9.05	11.57	14.76	18.81	23.96	30.50	38.81
l	3	0.58	1.73	2.00	3.86	5.24	6.90	8.93	11.48	14.69	18.76	23.92	30.46	38.78
o	5	0.45	1.00	1.41	2.45	4.73	6.57	8.69	11.30	14.55	18.65	23.83	30.40	38.73
a	8	0.35	0.22	0.33	1.41	3.00	5.98	8.30	11.02	14.34	18.49	23.70	30.30	38.65
d	13	0.28	0.22	0.33	0.56	1.34	3.74	7.52	10.50	13.96	18.20	23.49	30.13	38.52
e	21	0.22	0.22	0.33	0.56	0.89	1.32	4.69	9.52	13.31	17.73	23.13	29.86	38.31
d	34	0.17	0.22	0.33	0.56	0.89	1.07	1.30	5.92	12.07	16.90	22.53	29.40	37.96
	55	0.13	0.22	0.33	0.56	0.89	1.07	1.17	1.29	7.48	15.32	21.48	28.64	37.39
	89	0.11	0.22	0.33	0.56	0.89	1.07	1.17	1.29	1.45	9.49	19.46	27.30	36.41
	144	0.08	0.22	0.33	0.56	0.89	1.07	1.17	1.29	1.45	1.65	12.04	24.73	34.71
	233	0.07	0.22	0.33	0.56	0.89	1.07	1.17	1.29	1.45	1.65	1.90	15.30	31.44
	377	0.05	0.22	0.33	0.56	0.89	1.07	1.17	1.29	1.45	1.65	1.90	2.23	19.44

Tabella 2.3: Valori di Ratio Credit.

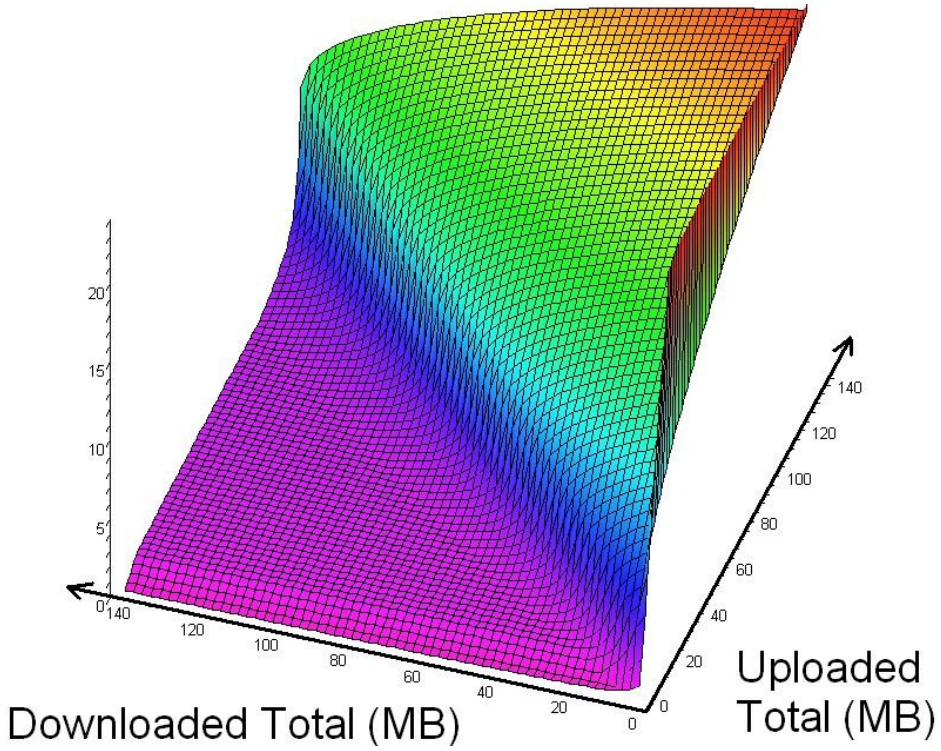

Figura 2.3: Il Sistema di Crediti Ratio.

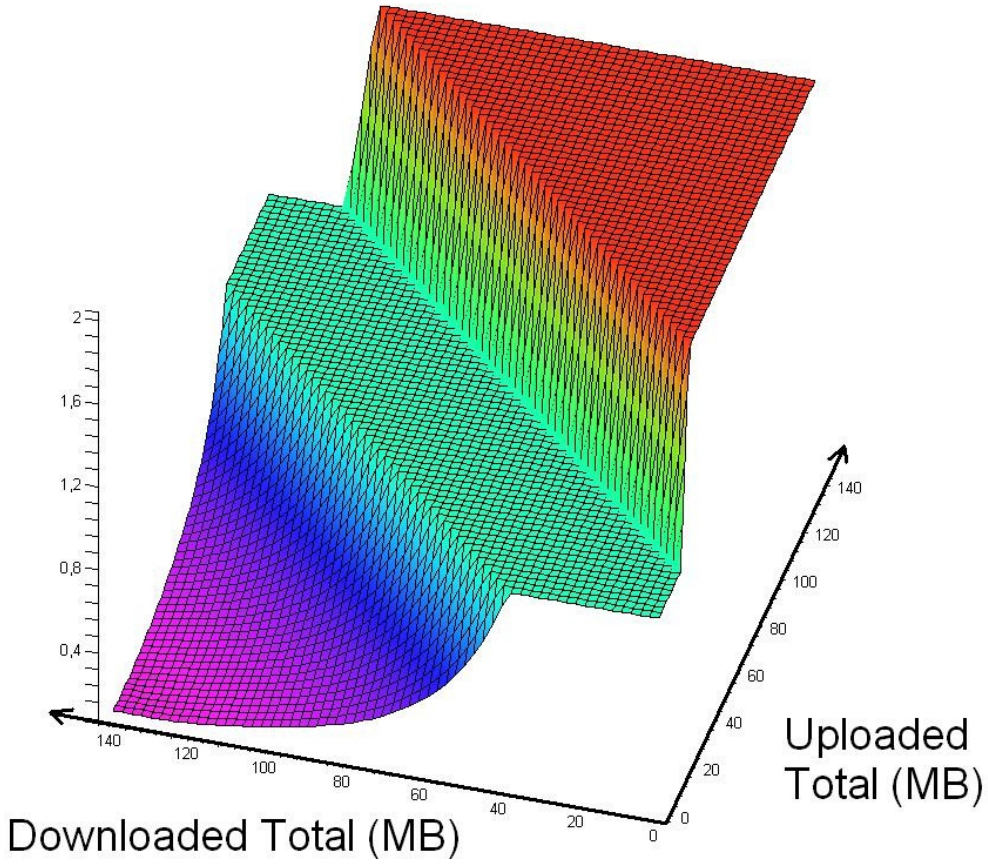

Figura 2.4: Il Sistema di Crediti Neo.

- Piccolo bonus per i client che ci hanno dato tanti MB:
  - Se riceviamo 100MB allora il client ottiene credito = 50 finché non gli diamo indietro 108MB.
  - Se riceviamo 50MB allora il client ottiene credito = 25 finché non gli diamo indietro 55MB.
  - Se riceviamo 25MB allora il client ottiene credito = 12 finché non gli diamo indietro 28MB.
  - Se riceviamo 10MB allora il client ottiene credito = 5 finché non gli diamo indietro 12MB.

2.4.9 Fine Credit System

Il sistema di crediti Fine è stato ideato solamente per identificare i "leech" (sanguisughe) e penalizzarli. Funziona solo su file parziali. I file completi non vengono presi in considerazione. Per i file parziali, questo sistema di crediti assicura ad ogni client di prendere 4 chunk completi "gratis". Dopo che il quarto chunk viene mandato senza che venga ricevuto in cambio alcun dato, il rating di quel cliente verrà decrementato. Maggiore è la differenza tra quantità scaricata dal client e la soglia dei 4 chunk, e minore sarà il suo credito. Il credito non raggiunge mai lo zero. Questo significa che quei client possono comunque scaricare dati da noi, a patto però che attendano abbastanza nella nostra coda di upload. Però il tempo di attesa aumenta con il diminuire del credito. Se un client, il quale è stato "punito" per aver scaricato troppo, inizia un upload verso di noi, il suo rating verrà incrementato. Appena la differenza tra Upload e Download è minore di 4 chunk, allora il client raggiungerà di nuovo il credito massimo.
Alcune precisazioni:

- Non c'è nessun ban[6].
- Non si fanno "congetture". Il sistema di crediti reagisce in modo deterministico alla situazione monitorata.
- Si basa su differenze assolute, e quindi niente rapporti.
- Non c'è una correlazione lineare tra la differenza ed il credito.

[6]Meccanismi che consentono di bandire un certo utente da una comunità virtuale, impedendogli di parteciparvi

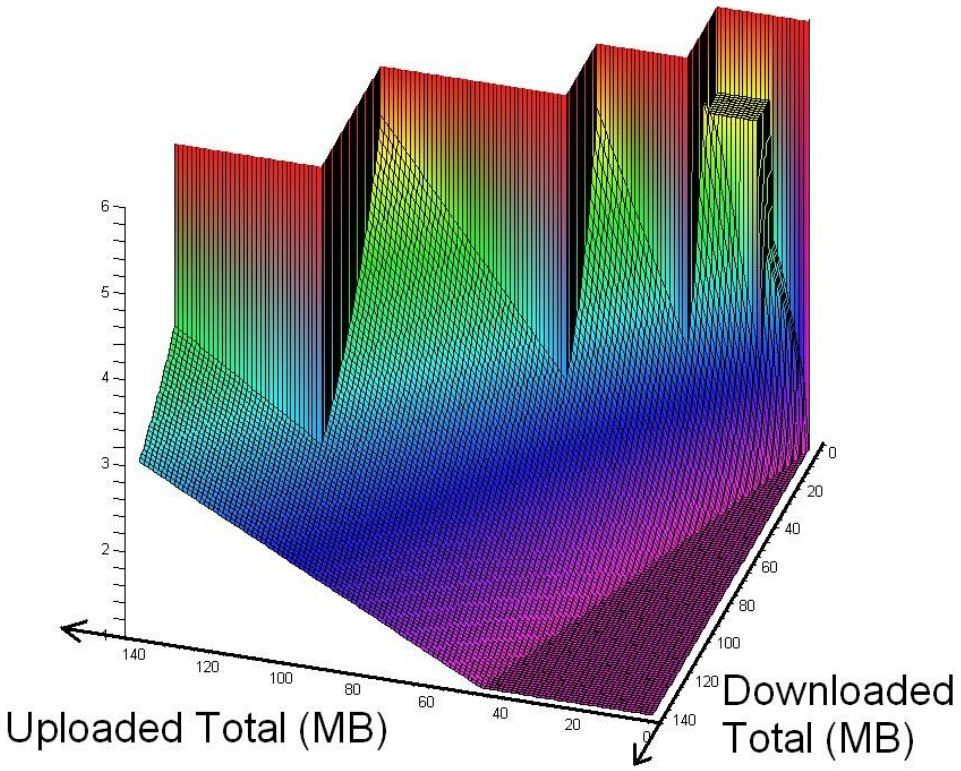

Figura 2.5: Il Sistema di Crediti Pawcio.

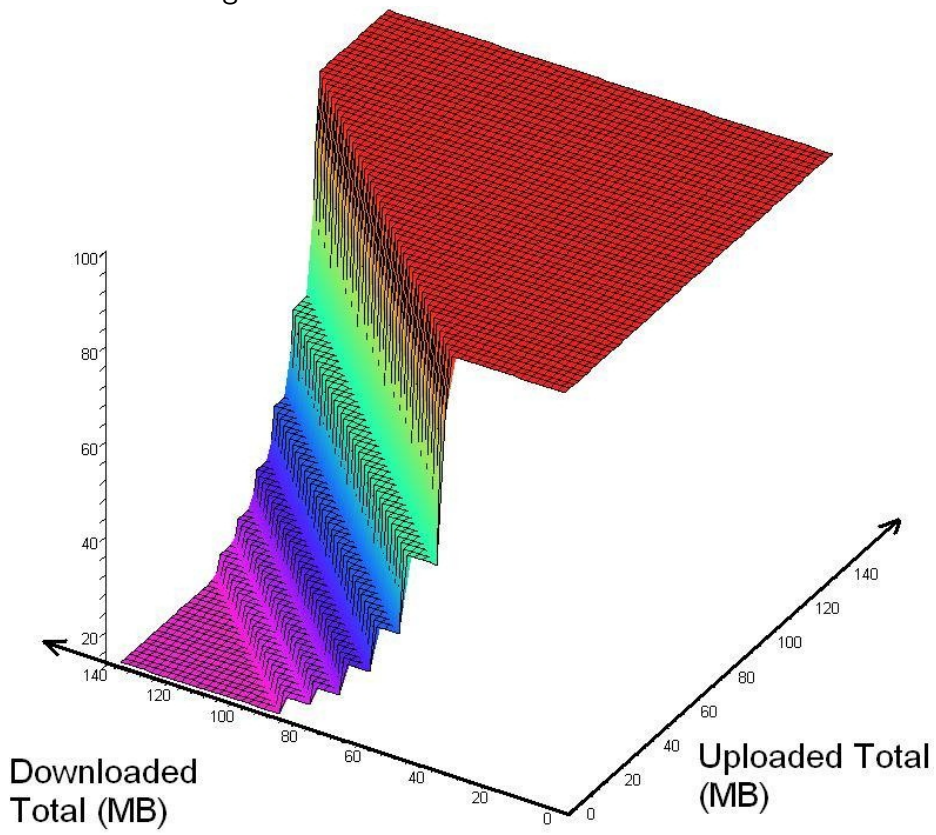

Figura 2.6: Il Sistema di Crediti Fine.

I client che non supportano l'identificazione sicura ricevono una penalizzazione del 20%. Non interferisce con lo Small File Pushing[7] (essendo quest'ultimi file completi).

La semplice equazione usata per calcolare i punteggi è:

diff := uploaded - downloaded
threshold := 4 * PARTSIZE
rating := 1
if (downloaded < PARTSIZE) // Se non riceviamo almeno un chunk,
    diff += PARTSIZE; // limita "risk" a 3 chunk invece di 4.
if (diff > threshold)
    rating := $(threshold/diff)^2$
if (!SecureIdentification)
    rating := rating * 0.80

2.4.10 Sivka Credits

Riportiamo in breve l'algoritmo per calcolare i crediti:

if isNuovoUtente
    ratio = 0.75
if !SecureIdentification //se la SI fallisce
    ratio = 0.5
if isBadUser
    ratio = 0
else
    diffTransfer = Upload - Download
    if udiffTransfer $\geq$ 1GB
        ratio = 32
    else if 0 < diffTransfer < 1GB
        ratio = $\sqrt{diffTransfer\ in\ MB}$
    else

[7]Questa funzionalità permette di "spingere in alto" nella coda di upload gli utenti che richiedono un file con dimensione sotto una certa soglia. Viene implementata in alcune mod di eMule.

$\mathsf{ratio} = 1$

2.4.11 S.W.A.T. Credits

Vengono calcolati due modificatori:

- Ratio1 = $\frac{Totale\,Upload * 2.2}{TotaleDownload}$
- Ratio2 = $\sqrt{(Totale\,Upload\ +\ 2)}$

Vengono confrontati i due valori così ottenuti, ed il più piccolo viene scelto come modificatore. Oltre a questa regola esistono anche alcuni casi particolari:

- Totale Upload < 1MB $\Longrightarrow$ Credito = 1.
- Totale Download = 0 $\Longrightarrow$ Credito = 10.
- Il credito non può essere minore di 1 o maggiore di 100.

2.4.12 Eastshare Credit System

Credito di base:

- Utente con identificazione sicura: credito = 100;
- Utente che non supporta l'identificazione sicura: credito = 80;
- Utente che fallisce l'identificazione sicura: credito = 0.

Il credito deve essere compreso tra 10 e 5000. Per ogni MB che riceviamo, incrementiamo il credito del client di 6. Per ogni MB che inviamo al client, decrementiamo il suo credito di 2. Se inviate più di 1KB avrete 50 punti di credito, se il vostro credito è sotto il 50 ma invierete più di 100kb il vostro punteggio arriverà a 50.

2.4.13 Lovelace Credit System

$$\text{ratio} = \beta * \left(\left(1 - \frac{1}{1 + \exp\left(\frac{3 * (MB\ inviati)^2 - (MB\ scaricati)^2}{1000} \right)} \right)^{6.6667} \right)$$

Questo sistema di crediti parte con valore 1, arriva al massimo a 100 e può scendere fino a 0.1 ed usa una sola formula.

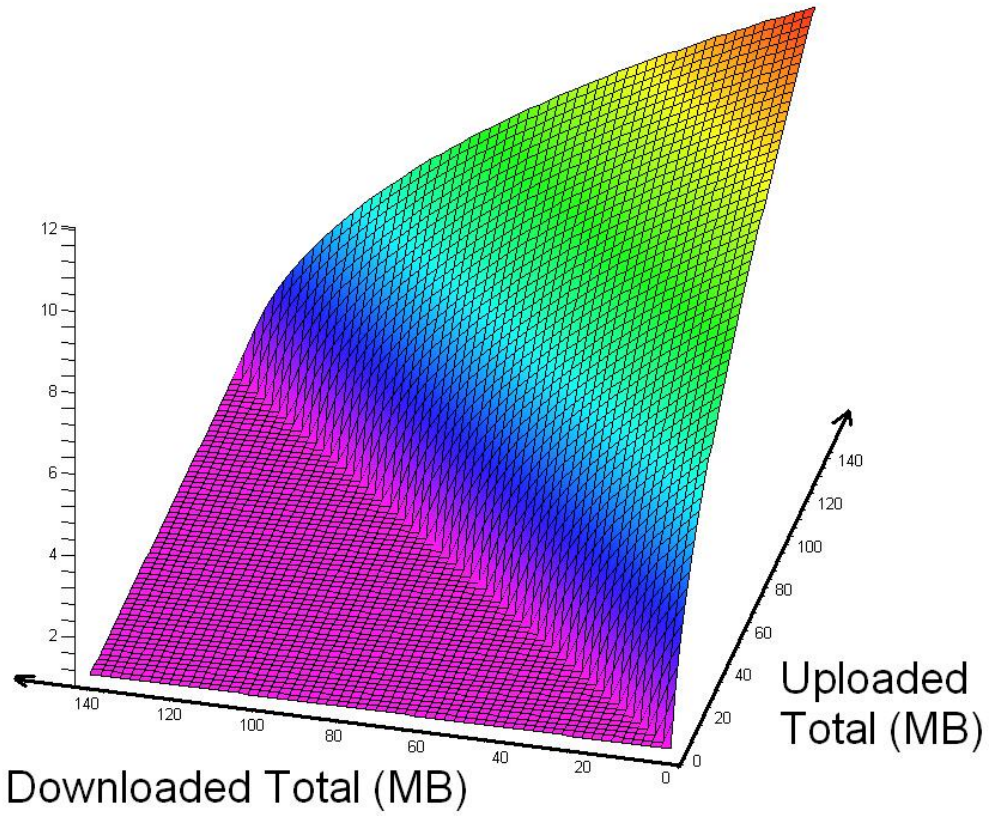

Figura 2.7: Il Sistema di Crediti Sivka.

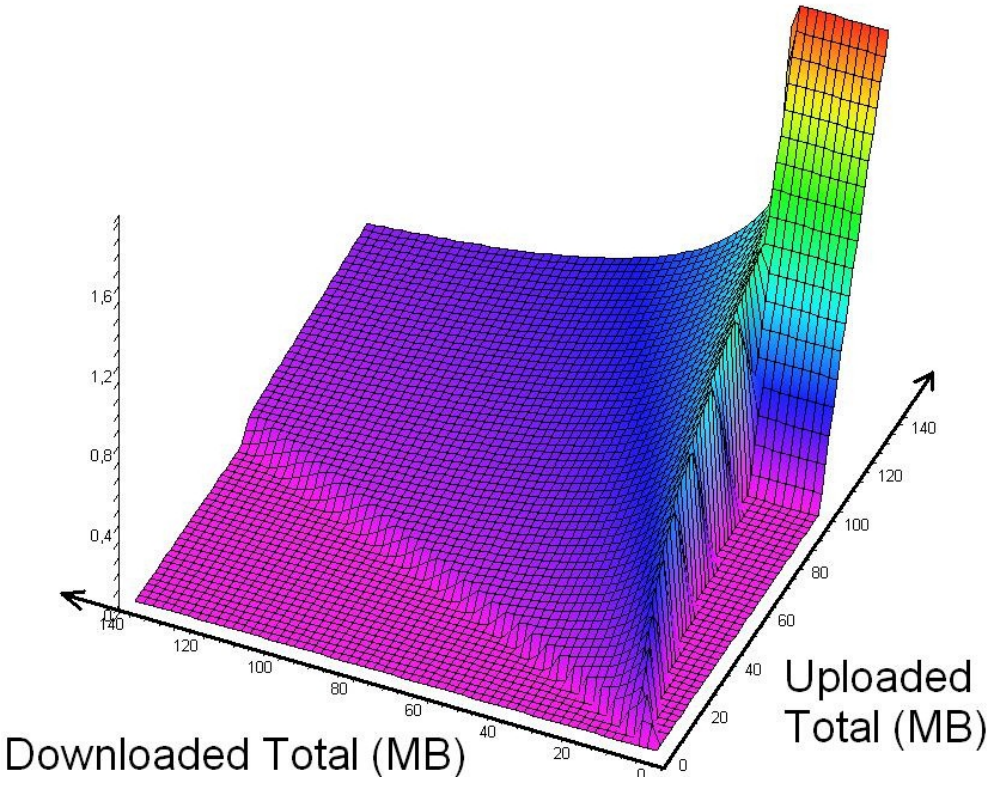

Figura 2.8: Il Sistema di Crediti S.W.A.T.

Solo gli utenti che effettuano l'identificazione sicura possono avere $\beta = 100$, gli altri hanno $\beta = 10$.

2.4.14 TK4 Credit System

Se stiamo condividendo file completi, e qualcuno scarica da noi dati da uno di questi file, probabilmente non ci possono dare dati in cambio, in quanto noi abbiamo tutto il file. Nel sistema di crediti TK4 questi client non subiscono variazioni di credito. E invece stiamo scaricando un file, e qualcuno scarica dati da noi, dalle parti che abbiamo già, del file che stiamo scaricando, quel qualcuno sarà soggetto al sistema di crediti: in funzione di quanto ci ha dato può avere riduzione del suo credito. Se qualcuno ci da dei dati, in qualsiasi momento, quel qualcuno ottiene un incremento del suo credito.

2.4.15 ClientAnalyzer

- Valore iniziale: 1
- Minimo: 0.01
- Massimo: 10

Assegna un punteggio ai client basandosi sull'analisi del loro comportamento:

- Da quanto tempo si conosce il client (banus per ogni settimana che il client mantiene inalterato il suo userhash).
- Rapporto upload/download.
- Tipo di upload/download (completo/parziale/raro).
- Opzioni anti leecher, nickthieves, modthieves, filefakers, spammers, ecc. vengono prese in considerazione.
- Tempo medio tra una richiesta e l'altra.

2.4.16 Reverse Credits

Questo metodo di gestione dei crediti è considerata una "bad feature[8]".

[8]Una bad feature (caratteristica cattiva) è una caratteristica che non viene generalmente accettata nella rete eDonkey/eMule. Alcuni le chiamano illegali, ma di solito sono solo caratteristiche che non sono conferite dalla tabella delle regole di http://forum.emule-project.net.

		Downloaded from me									
		1	2	3	4	5	6	7	8	9	10
U	1	0.98861	0.97877	0.96256	0.94024	0.9122	0.8789	0.84089	0.79879	0.75328	0.70507
p	2	1.01863	1.00854	0.99191	0.96902	0.94024	0.90607	0.86705	0.82383	0.77709	0.72756
l	3	1.07039	1.05987	1.04252	1.01863	0.98861	0.95294	0.9122	0.86705	0.8182	0.76642
o	4	1.14658	1.13542	1.11703	1.09171	1.05987	1.02202	0.97877	0.93081	0.8789	0.82383
a	5	1.2512	1.23919	1.2194	1.9213	1.15783	1.11703	1.07039	1.01863	0.96256	0.90301
d	6	1.38983	1.37672	1.3551	1.32531	1.28781	1.24319	1.19213	1.113542	1.07392	1.00854
e	7	1.56993	1.55542	1.53149	1.4985	1.45695	1.40747	1.35081	1.28781	1.2194	1.14658
d	8	1.80127	1.78503	1.75822	1.72124	1.67464	1.6191	1.55542	1.48454	1.40747	1.32531
	9	2.09644	2.07805	2.0477	2.00582	1.95299	1.88997	1.81764	1.73701	1.6492	1.55542
t	10	2.47139	2.45041	2.41575	2.36788	2.30746	2.2353	2.15238	2.0598	1.95881	1.85074
o	11	2.9461	2.92198	2.88212	2.82705	2.75745	2.67424	2.577848	2.47139	2.35435	2.22884
	12	3.54511	3.51725	3.4712	3.40751	3.32695	3.23051	3.11934	2.99481	2.85842	2.71182
m	13	4.29803	4.26578	4.21242	4.13857	4.04505	3.93294	3.80351	3.65824	3.49878	3.32695
e	14	5.23974	5.20237	5.14053	5.05485	4.94625	4.81585	4.66505	4.49544	4.30883	4.10722
	15	6.4101	6.3669	6.29536	6.19616	6.07026	5.91887	5.74346	5.54575	5.32769	5.09144
	16	7.8531	7.80338	7.72098	7.60661	7.46127	7.28623	7.08304	6.8535	6.59969	6.32391
	17	9.61502	9.55817	9.46389	9.33291	9.16624	8.9652	8.73135	8.46657	8.17320	7.8531
	18	11.74162	11.67719	11.57028	11.42162	11.2322	11.00332	10.73657	10.43383	10.09728	9.72939
	19	14.27424	14.20204	14.08217	13.9153	13.70241	13.44473	13.14381	12.80149	12.4199	12.00119
	20	17.24516	17.16532	17.03266	16.84781	16.61167	16.32537	15.99036	15.60836	15.18136	14.71171

Tabella 2.4: Valori del Lovelace Credit System.

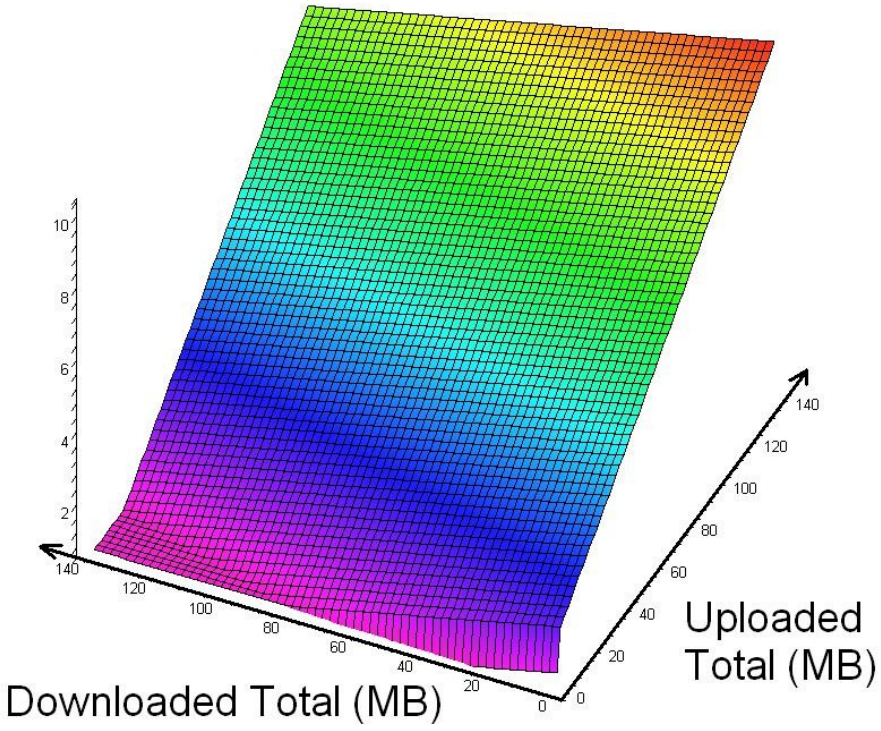

Figura 2.9: Il Sistema di Crediti Eastshare.

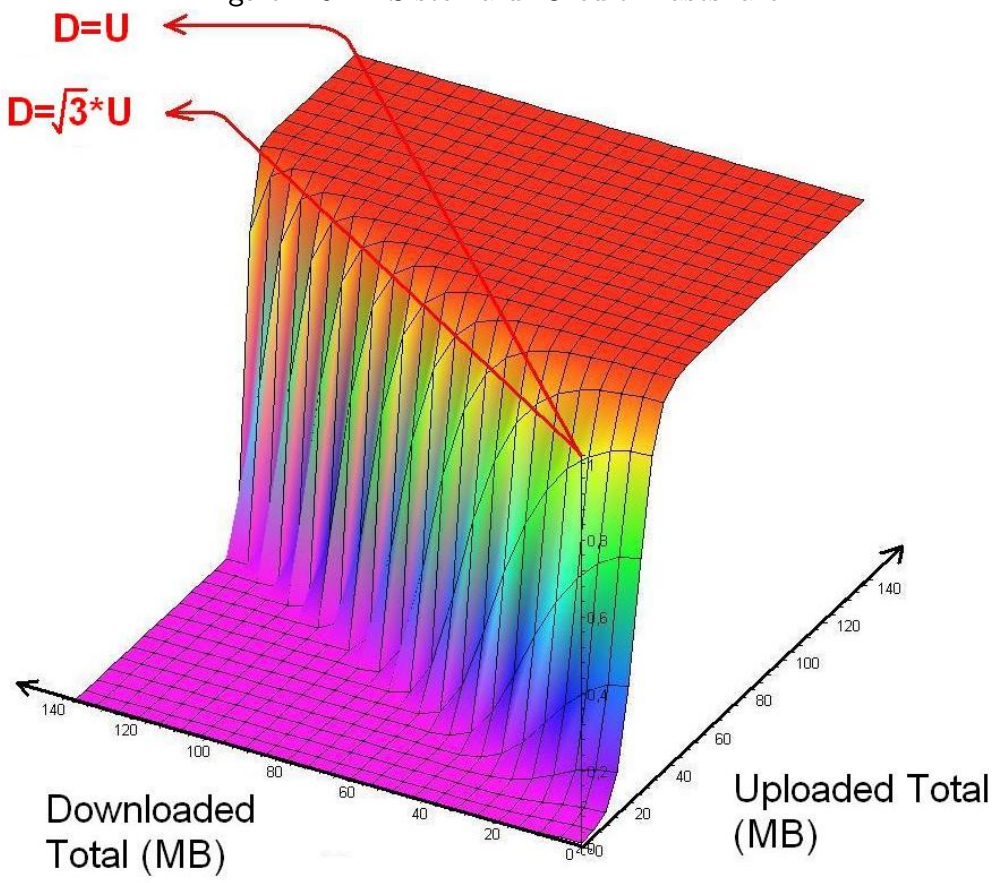

Figura 2.10: Il Sistema di Crediti Lovelace.

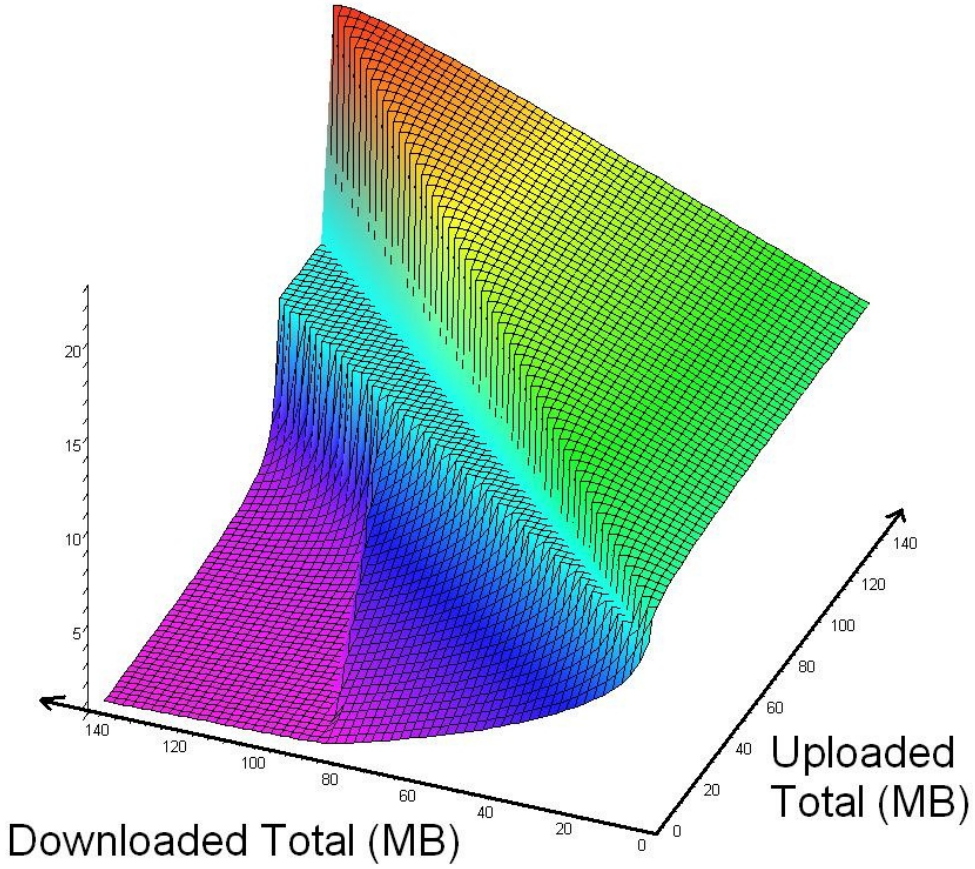

Figura 2.11: Il Sistema di Crediti TK4.

- Dai un incremento di punteggio di 10 volte a tutti i client nella tua coda di download. (presto vorrai avere qualcosa da loro)
- Penalizza le NoNeededParts e queuefull(coda piena) 2x (al momento non sono necessarie ma nel futuro potrebbero esserlo, invia dati a loro solo quando tutte le fonti hanno un punteggio di almeno 5).
- Dai un piccolo incremento di punteggio se il tempo di attesa stimato nelle code dei client di interesse è basso.
- Dai un piccolissimo incremento in base al tempo di attesa.

Il nostro client servirà gli altri client in questo ordine:

1. Client dai quali noi tentiamo di scaricare i quali non ci hanno ancora dato nulla.
2. Client con tempo di attesa stimato basso.

3. Client che ci hanno fatto scaricare, dai quali noi stiamo tentando si caricare nuovamente. Questi ottengono un incremento di 5x.
4. Client dai quali stiamo tentando di scaricare, ma che sono nello stato NoNeededParts o QueueFull.
5. Tutti gli altri client dai quali stiamo cercando di scaricare ottengono un incremento di punteggio di 10x.
6. Tutti i client che hanno credito massimo verranno serviti con politica FIFO (first in, first out).

2.5 Sistema di Crediti di Mulo

Viene descritto in questa sezione l'implementazione del sistema di credtiti di Mulo, per il quale abbiamo preso spunto dalla versione ufficiale di eMule.

2.5.1 Prima dell'implementazione dei Crediti

Per avere un'idea più precisa di come è stato implementato il nostro sistema di crediti, mostriamo prima come Mulo gestiva gli upload senza tale funzionalità.

La classe UploadManager.java gestisce tutte le procedure di upload (cioè servire i peer che ci chiedono dei dati). Contiene due insiemi di oggett UploadSessions[9]:

1. La **active list** (lista attiva) è la lista dei peer correntemente abilitati a rivevere dati.
2. La **queued** (in coda) è la lista di peer che aspettano il loro turno di ricevere dati.

Le liste sono FIFO, cioè il peer che vi entra per primo, sarà il primo ad essere servito.

UploadManager è anche un thread, il quale gestisce ciclicamente i peer nella active list:

- Inviando loro dati(in intervalli precisi, in modo da controllare la velocità media);

[9]Classe che definisce e gestisce le singole richieste di upload, le sue istanze sono coppie del tipo (peer richiedente, file richiesto)

- Sospendendo la loro sessione di upload se hanno ricevuto abbastanza dati per il turno corrente (la quantità è dinamica, e dipende dal numero di peer nella coda queued).

A ogni iterazione viene anche verificato se siano state ricevute nuove richieste, di ulteriori frammenti di file oppure di interruzione dell'upload. Se la nostra coda di attesa non è vuota e a un peer è stata inviata nel round[10] corrente una certa quantità di dati (determinata come specificato nella tabella 2.5), esso viene estromesso dalla lista di upload attivi e ricollocato in coda, per far posto ad altri utenti. La quantità di dati inviabili per round non è fissa (come in eMule, dove essa corrisponde alla dimensione di una parte, 9.500 KiB), bensì inversamente proporzionale al numero di peer che stanno attendendo il loro turno in coda: in pratica più è lunga la coda più rapidamente Mulo la fa scorrere, inviando meno dati a ciascuno. Più precisamente:

- coda $= 0 \Rightarrow \infty$
- coda $= 1 \Rightarrow$ 9.500 KiB
- coda $> 1 \Rightarrow$ 180 KiB x $\left\lfloor \frac{53}{\log_e (coda + e)} \right\rfloor$ dove 53 è il numero di chunk per parte ed e è la costante di Nepero.

Non necessariamente vengono inviati chunk interi: questo dipende da quali frammenti ci richiede il peer.

2.5.2 L'implementazione dei Crediti

I crediti assegnati ai peer vengono calcolati usando le stesse formule che usa la versione ufficiale di eMule. A questo scopo è stato scritto il metodo getCredit() della classe Peer.java. Il metodo getScore() della classe UploadSession.java restituisce il punteggio di priorità in base al quale i peer potranno scaricare da noi. Ci sono alcune differenze sostanziali con eMule:

- I peer vengono discriminati in due categorie:
    1. peer che supportano l'identificazione sicura e la effettuano con successo;
    2. peer che non supportano l'identificazione sicura, o che, per qualche motivo, la falliscono.

[10] Intervallo di tempo che va dall'ingresso del peer nella lista degli upload attivi alla sua uscita (riaccodamento o uscita completa), ovvero periodo in cui gli inviamo i dati.

Coda	Limite	Chunk
1	9.500 KiB	53
2	6.300 KiB	35
4	5.040 KiB	28
8	4.140 KiB	23
16	3.420 KiB	19
32	2.700 KiB	15
64	2.340 KiB	13
128	1.980 KiB	11
256	1.800 KiB	10
512	1.620 KiB	9
1024	1.440 KiB	8

Tabella 2.5: Dati inviabili per round di upload in funzione dei peer in coda.

- In base a tale suddivisione, se ricadenti nella seconda categoria, essi verranno penalizzati rispetto a quelli della prima categoria.
- Mulo non supporta ancora l'impostazione della priorità dei diversi file per l'upload/download, quindi questo modificatore manca nel metodo getScore().
- Il tempo di attesa nella coda queued non incide direttamente sullo score, visto che i peer non vengono gestiti in base al tempo, ma in base alla quantità di dati inviati.

Riportiamo il codice del metodo getScore():

```
float getScore() {
    Peer.updateCredits(); // prima aggiorniamo i crediti di tutti i peer
    if (!this.peer.isSecured()) {
        return this.peer.getCredit() - 10; // se il peer non è stato
    } //identificato con la SI, allora deve avrà punteggio tra 0 e -9.
    return this.peer.getCredit();
}
```

Essendo il credito compreso tra 1 e 10, sottraendo 10, otteniamo un numero tra 0 e -9. In questo modo i peer senza SI, vengono ordinati per credito tra di loro, ma avranno sempre score inferiore a qualsiasi peer che supporta la SI. Abbiamo scelto di fare questa discriminazione in quanto oggigiorno praticamente

ogni client eDonkey che si rispetti implementa la Secure Identification. Inoltre, in questo modo, se durante l'handshake con un peer, per qualche motivo la SI fallisce (ad esempio si perde o si corrompe un pacchetto), vedendosi sempre in fondo alla coda queued, quest'ultimo sarebbe in qualche modo spinto a ritentare la connessione una seconda volta, aumentando così la possibilità che la SI abbia successo. Infine, gli utenti che hanno un client che non supporta la SI, verrebbero spinti a cambiare client (e potenzialmente a passare a Mulo!).

Per poter ordinare i peer in base al loro score è stata modificata la classe UploadSession.java, e cioè è stata resa Comparable[11]:

```
class UploadSession implements Comparable<UploadSession>
```

L'ordinamento viene reso possibile dal metodo compareTo() anch'esso implementato nella classe UploadSession.java:

```
public int compareTo(UploadSession upload) {
    if (( this.getScore() - upload.getScore() ) == 0) {
        return 0;
    }
    else if (( this.getScore() - upload.getScore() ) > 0) {
        return -1;
    }
    else {
        return 1;
    }
}
```

Per ordinare i peer in attesa di essere serviti, usiamo il metodo sort() della classe Collections.java[12]. Questo metodo implementa una versione modificata di **Merge Sort** e garantisce l'ordinamento in tempo $O(n \log n)$. L'ordinamento avviene in ordine ascendente, secondo il metodo compareTo(). A questo proposito, facciamo subito notare che il metodo compareTo() riportato poco sopra, ordina gli oggetti sui quali viene chiamato, non in ordine di score crescente, ma bensì decrescente, in modo da avere i peer con score maggiore in cima alla lista queued.

[11] L'interfaccia Comparable impone un ordine totale sugli oggetti della classe da cui viene implementata. Vedi anche: http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Comparable.html

[12] Vedi: http://java.sun.com/j2se/1.4.2/docs/api/java/util/Collections.html

La lista queued viene ordinata ogni volta avvenga in essa un inserimento, sia di peer nuovi, sia di peer che provengono dalla active list. In questo modo, il prossimo peer ad entrare nella lista active, ed essere quindi servito, è sempre quello con score maggiore.

2.5.3 Clients.xml

I peer che, facendoci scaricare da loro, ottengono un qualche credito maggiore di 1, vengono salvati nel file "Clients.xml" (è inutile salvare i peer che non hanno alcun credito), ad intervalli di 15 minuti, nonché prima della chiusura di Mulo. Per ogni peer con credito > 1, vengono salvati:

- il suo userhash,
- il numero di byte che il peer ha scaricato da noi,
- il numero di byte che noi abbiamo scaricato dal peer,
- l'ultima volta che abbiao visto il peer,
- la chiave pubblica[13] del peer.

All'avvio di Mulo, viene letto il file "Clients.xml", le informazioni contenute vengono caricate in memoria, ed eventualmente sfruttate durante l'esecuzione. Prima del salvataggio del file "Clients.xml", vengono eliminati i peer che non incontriamo da 5 mesi (come in eMule).

Facciamo notare che non viene salvato il valore dei crediti, ma il numero di byte scambiati.

2.5.4 Un'ultima considerazione

Facciamo brevemente un'ultima considerazione importante. Come si può facilmente inutire, l'implementazione del sistema di crediti di Mulo è abbastanza trasparente riguardo ad eventuali modifiche future. Infatti, si potrebbero apportare modifiche al sistema di crediti modificando solamente i metodi getCredit() e getScore(), senza toccare alcun'altra parte del codice. Se in un futuro prossimo, gli sviluppatori di Mulo riterranno opportuno implementare funzionalità come l'impostare manualmente le priorità dei download/upload, potranno, senza molte difficoltà, apportare le loro modifiche al sistema di crediti.

[13]Una chiave pubblica è una chiave crittografica utilizzata in un sistema di crittografia asimmetrica. Vedi Cap. 3 per maggiori dettagli.

Capitolo 3

Secure Identification

L'identificazione sicura è parte dell'estensione eMule del protocollo eDonkey. Se l'identificazione sicura è supportata dai client, essa avviene immediatamente dopo l'handshake iniziale. Essa viene pubblicizzata in uno dei tag[1] dei pacchetti HelloRequest e HelloResponse (chiamati anche Hello e Hello answer), e cioè quello chiamato **eMule Misc Opts 1** con id = 0xFA. Lo scopo dell'identificazione sicura è quello di prevenire il furto di identità tra utenti (con consecutivo furto di crediti).

Campo	**Lunghezza**	**Descrizione**
0x10	1	Lunghezza del campo userhash
userhash	16	Hash che identifica l'utente
Client ID	4	ID del client
Client Port	2	Porta TCP del client
Tags No.	4	Numero di tag
Tags	variabile	Lista di tag
Server ID	4	ID (alto) del server a cui è connesso il client
Server Port	2	Porta TCP dello stesso server

Tabella 3.1: Pacchetto HelloRequest (Hello).

[1] I tag sono dei "sottopacchetti" che possono essere inseriti in qualsiasi quantità e ordine all'interno di determinati pacchetti, per aggiungere informazioni varie in modo flessibile.

Campo	Lunghezza	Descrizione
userhash	16	Hash che identifica l'utente
Client ID	4	ID del client
Client Port	2	Porta TCP del client
Tags No.	4	Numero di tag
Tags	variabile	Lista di tag
Server ID	4	ID (alto) del server a cui è connesso il client
Server Port	2	Porta TCP dello stesso server

Tabella 3.2: Pacchetto HelloResponse (HelloAnswer).

3.1 UserHash

Ogni utente eMule ha un proprio **userhash** (detto anche user ID), che altro non sarebbe che la "carta di identità dell'utente". Questo userhash è un GUID[2] composto da 16 byte, ed è salvato su file. Esso viene creato concatenando numeri casuali, fatta eccezione per il sesto ed il quindicesimo byte, i valori de i quali sono sempre 14, e 111 rispettivamente. Questo identificatore è creato un'unica volta, al primo avvio del client, è deve rimanere sempre inalterato. Esso infatti, non deve essere confuso con l'ID del client, o con il suo indirizzo IP, in quanto questi ultimi possono variare. Viene utilizzato per effettuare l'identificazione sicura, dando così la possibilità al client di tenere traccia dei crediti degli altri utenti della rete.

3.2 RSA

Per capire come funziona la Secure Identification dobbiamo prima capire come funziona l'algoritmo RSA, sul quale si basa.

Con un sistema di crittografia a chiave pubblica, possiamo criptare messaggi inviati tra due entità comunicanti, in modo tale che un eventuale "eavesdropper" (chi origlia) che sente il messaggio criptato non sia in grado di carpirne il contenuto. Un sistema di crittografia a chiave pubblica permette inoltre, ad una entità che la usa, di aggiungere una "firma digitale", non forgiabile, alla fine di un messaggio elettronico. Una simile firma è la versione elettronica di una firma fatta a mano su di un documento cartaceo. Può essere facilmente controllata da

[2]Globally Unique Identifier - è un tipo particolare di identificatore usato in applicazioni software per dare un numero di riferimento univoco.

chiunque, non creabile da nessun'altro, ed inoltre perde la sua validità se un qualsiasi bit del messaggio viene alterato. Permette quindi di dimostrare l'autenticità dell'identità dell'autore, e del contenuto del messaggio.

Il sistema di crittografia a chiave pubblica RSA si basa sulla profonda differenza tra la facilità nel trovare numeri primi grandi, e nella difficoltà nel fattorizzare il prodotto di tali numeri primi grandi.

3.2.1 Sistemi di crittografia a chiave pubblica

In un sistema di crittografia a chiave pubblica ogni partecipante alla comunicazione ha una chiave pubblica ed una chiave segreta. Ognuna delle due chiavi è un pezzo di informazione. Per esempio, nella crittografia RSA, ogni chiave consiste in una coppia di interi. I partecipanti "Alice" e "Bob" sono tradizionalmente usati negli esempi di crittografia; denotiamo le loro chiavi pubbliche e segrete come P_A, S_A per Alice e P_B, S_B per Bob.

Ogni partecipante crea la propria chiave pubblica e segreta. Le chiave segrete vengono tenute segrete, ma le chiavi pubbliche possono essere rivelate a chiunque, o addirittura possono essere pubblicate. Infatti è spesso conveniente assumere che le chiavi pubbliche di tutti siano disponibili in una directory pubblica, in modo tale che ogni partecipante possa facilmente ottenere le chiavi pubbliche degli altri partecipanti.

Le chiavi pubbliche e segrete specificano funzioni che possono essere applicate a qualsiasi messaggio. Sia D l'insieme dei messaggi permessi. Ad esempio, D potrebbe essere l'insieme di tutte le stringhe di bit finite. Nella più semplice formulazione (che è anche quella originale) di crittografia a chiave pubblica, si richiede che le chiavi pubbliche e segrete specifichino funzioni biunivoche da D a se stesso. Denotiamo la funzione associata alla chiave pubblica di Alice P_A con $P_A()$, e la funzione corrispondente alla sua chiave segreta S_A con $S_A()$. Le funzioni $P_A()$ e $S_A()$ sono permutazioni di D. Assumiamo che le funzioni $P_A()$ e $S_A()$ siano efficientemente calcolabili date le corrispettive chiavi P_A e S_A.

Le chiavi pubblica e segreta sono una "coppia abbinata" perché definiscono funzioni che sono l'una l'inversa dell'altra. E cioè,
$M = S_A(P_A(M))$,
$M = P_A(S_A(M))$
per ogni messaggio M che appartiene a D. Trasformando M con le due chiavi P_A e S_A in successione, in ambedue gli ordini, ci restituisce di nuovo il messaggio M.

In un sistema di crittografia a chiave pubblica, si richiede che nessun'altro a parte Alice sia in grado di calcolare la funzione $S_A()$ in tempo utile. Questa assunzione è cruciale per mantenere i messaggi inviati ad Alice privata, sapendo che le firme digitali di Alice sono autentiche. Alice deve mantenere S_A segreta; se non lo fa, perde la sua unicità e il sistema di crittografia non può garantire le sue capacita uniche. L'assunzione che Solo Alice possa calcolare $S_A()$ deve rimanere vera anche se chiunque può conoscere P_A e calcolare $P_A()$, la funzione inversa di $S_A()$, efficientemente. Per creare un sistema di crittografia a chiave pubblica, dobbiamo quindi trovare un modo per rivelare la trasformazione $P_A()$ senza rivelare la corrispondente trasformazione inversa $S_A()$.

In un sistema di crittografia a chiave pubblica, la cifratura funziona come illustrato nella figura 3.1. Supponiamo che Bob voglia inviare ad Alice un messaggio M, cifrato, in modo che sembri un miscuglio illeggibile ad un eavesdropper. Lo scenario per l'invio del messaggio è come segue:

- Bob ottiene la chiave pubblica di Alice (dalla directory pubblica o direttamente da Alice).
- Bob calcola il testo cifrato $C = P_A(M)$ corrispondente al messaggio M e invia C ad Alice.
- Quando Alice riceve il testo cifrato C, applica la funzione $S_A()$ associata alla sua chiave segreta, in modo da ottenere il messaggio originale: $S_A(C) = S_A(P_A(M)) = M$.

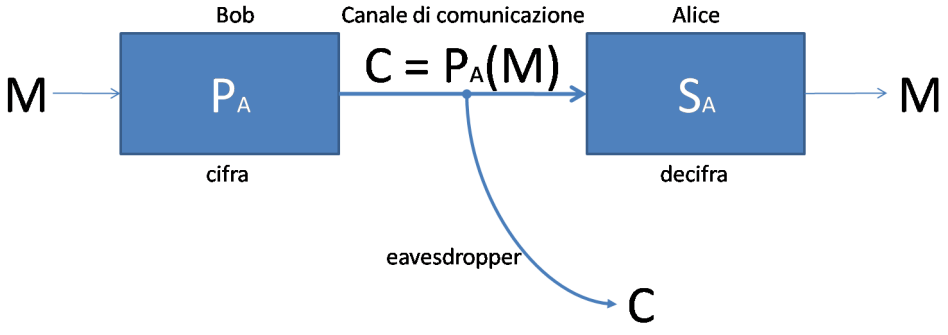

Figura 3.1: Cifratura in un sistema a chiave pubblica.

Dato che $S_A()$ e $P_A()$ sono funzioni inverse, Alice può calcolare M da C. Dato che Alice è l'unica a poter calcolare $S_A()$, è anche l'unica a poter riottenere M da C. Dato che Bob cifra M usando $P_A()$, solamente Alice può capire il messaggio trasmesso.

Possiamo facilmente implementare la firma digitale dentro la nostra formulazione si sistema di crittografia a chiave pubblica. Supponiamo adesso che Alice voglia inviare a Bob una risposta M' firmata. La figura 3.2 dimostra come funziona lo scenario con firma digitale.

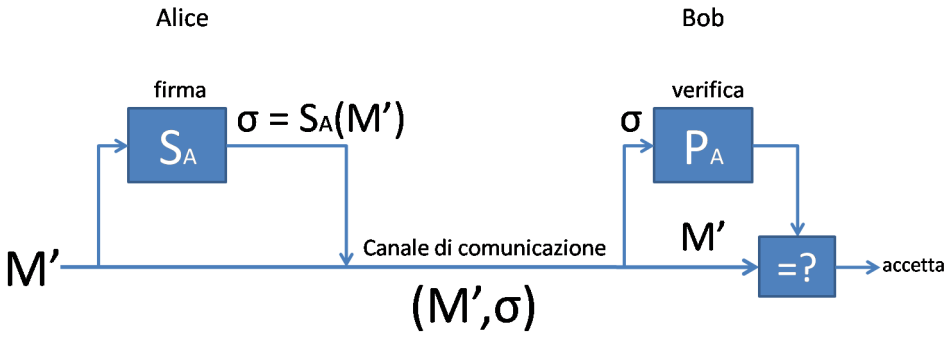

Figura 3.2: Firma digitale in un sistema a chiave pubblica.

- Alice calcola la sua firma digitale σ per il messaggio M' usando la sua chiave segreta Sa e l'equazione $\sigma = S_A(M')$.
- Alice invia la coppia messaggio/firma (M', σ) a Bob.
- Quando Bob riceve (M', σ), può verificare che l'autore del messaggio è Alice usando la chiave pubblica di Alice per verificare l'equazione $M' = P_A(\sigma)$. (Presumibilmente M' contiene il nome di Alice, così che Bob sa quale chiave pubblica usare.) Se l'equazione viene verificata, allora Bob può concludere che il messaggio M' era firmato da Alice. Altrimenti, se l'equazione non è verificata, Bob può concludere o che il M' e/o la firma σ sono stati corrotti durante la trasmissione, o che c'è stato un tentativo di contraffazione.

Dato che la firma digitale fornisce sia l'autentificazione dell'identità del firmante sia la l'autenticità del contenuto del messaggio firmato, possiamo dire che essa è analoga alla firma tradizionale alla fine di un documento cartaceo.

La firma digitale deve essere verificabile da chiunque abbia accesso alla chiave pubblica dell'autore della firma. Un messaggio firmato può essere verificato da una certa entità, per poi essere trasmessa ad altre entità, le quali possono a loro volta verificarne l'autenticità. Per esempio, il messaggio può essere un assegno elettronico da Alice a Bob. Dopo che Bob verifica la firma di Alice sull'assegno, può dare l'assegno alla sua banca, che potrà, a sua volta, verificarne la firma, ed effettuare il trasferimento di fondi accordato.

Un messaggio firmato non è necessariamente cifrato; il messaggio può essere "in chiaro" (in clear) e non protetto dalla divulgazione. Componendo entrambi i protocolli sopradescritti per la cifratura e la firma digitale, possiamo creare messaggi che sono contemporaneamente cifrati e firmati. Il firmatario prima appende la sua firma al messaggio, e poi cifra la coppia messaggio/firma con la chiave pubblica di del destinatario. Il destinatario decifra il messaggio ricevuto con la sua chiave segreta per ottenere sia il messaggio originale che la firma del mittente. Il destinatario può poi verificare la firma usando la chiave pubblica del firmatario. Il corrispettivo processo combinato, usando sistemi cartacei, sarebbe firmare il documento cartaceo e poi sigillare il documento dentro ad una busta di carta che viene aperta solo dal destinatario prescelto.

3.2.2 L'implementazione tramite algoritmo RSA

È nel 1978 che questo sistema trova la sua applicazione reale. Infatti sono 3 ricercatori del MIT (Ronald Rivest, Adi Shamir e Leonard Adleman) che hanno saputo implementare tale logica utilizzando particolari proprietà formali dei numeri primi con alcune centinaia di cifre. L'algoritmo da loro inventato, denominato RSA per via delle iniziali dei loro cognomi, non è sicuro da un punto di vista matematico teorico, in quanto esiste la possibilità che tramite la conoscenza della chiave pubblica si possa decriptare un messaggio, ma l'enorme mole di calcoli e l'enorme dispendio in termini di tempo necessario per trovare la soluzione, fa di questo algoritmo un sistema di affidabilità pressoché assoluta.

Nella crittografia a chiave pubblica RSA, un partecipante crea la sua coppia chiave pubblica/segreta tramite la seguente procedura:

1. Scegliere casualmente due numeri primi molto grandi p e q tali che $p \neq q$.

2. Calcolare $n = pq$ (detto modulo).

3. Scegliere un piccolo numero dispari e coprimo e più piccolo di $\phi(n) = (p-1)(q-1)$ (detto esponente pubblico).

4. Calcolare d come l'inverso moltiplicativo di e, modulo $\phi(n)$ (detto esponente privato).

5. Pubblicare la coppia $P = (e, n)$ come chiave pubblica RSA del partecipante.

6. Tenere segreta la coppia $S = (d, n)$ come chiave segreta RSA del partecipante.

Per questo schema, il dominio D è $\mathbb{Z}_n$. Per trasformare un messaggio M associato con una chiave pubblica $P = (e, n)$, calcola
$P(M) = M^e \, mod \, n$.
Per trasformare un testo cifrato C associato con una chiave segreta $S = (d, n)$, calcola
$S(C) = C^d \, mod \, n$.

Queste equazioni si applicano sia alla cifratura, sia alla firma digitale. Per creare una firma digitale, il firmatario applica la sua chiave segreta al messaggio da firmare (invece che ad un messaggio cifrato). Per verificare una firma, vi viene applicata la chiave pubblica del mittente (invece che applicarla ad un messaggio da cifrare).

3.2.3 Correttezza dell'algoritmo RSA

La decrittazione del messaggio è assicurata grazie ad alcuni teoremi matematici, infatti dal calcolo noi otteniamo
$C^d = (M^e)^d = M^{ed} \, mod \, n$.
Ma sappiamo che
$ed \equiv 1 \, mod \, (p-1)(q-1)$,
e di conseguenza abbiamo che
$ed \equiv 1 \, mod \, (p-1)$ e che $ed \equiv 1 \, mod \, (q-1)$.
Quindi per il **piccolo teorema di Fermat**:
$M^{ed} \equiv M \, mod \, p$ e $M^{ed} \equiv M \, mod \, q$.
Siccome p e q sono numeri diversi e primi, possiamo applicare il **teorema cinese del resto**, ottenendo che
$M^{ed} \equiv M \, mod \, (pq)$
e quindi che
$C^d \equiv M \, mod \, n$.

La sicurezza dell'algoritmo RSA si basa in gran parte sulla difficoltà di fattorizzare interi molto grandi. Se un avversario potesse fattorizzare il modulo n in una chiave pubblica, allora egli potrebbe derivare la chiave segreta da quella pubblica, usando la conoscenza di p e di q nello stesso modo in cui il loro creatore originale le ha usate in principio. Quindi, se fattorizzare interi molto grandi fosse

facile, allora anche infrangere il sistema di crittografia RSA sarebbe facile. Il predicato inverso, e cioè: se fattorizzare interi molto grandi è difficile, allora anche infrangere il sistema di crittografia RSA è difficile, non è comprovato. Dopo due decenni di ricerche,non è ancora stato trovato nessun metodo più semplice di quello di fattorizzare il modulo n, per infrangere la crittografia RSA.

Anche se forse non facilmente intuibile, fattorizzare interi molto grandi è un problema molto difficile. Invece, trovare numeri primi molto grandi è abbastanza facile. La dimostrazione di tali risultati esula dagli obiettivi di questo lavoro, e quindi non viene trattata. Scegliendo casualmente due primi molto grandi (ad esempio 1024 bit ciascuno) e moltiplicandoli, otteniamo una chiave pubblica che non può essere infranta in tempo utile con la tecnologia attuale.

In assenza di svolte fondamentali negli algoritmi sulla teoria dei numeri, e quando implementato con cura seguendo gli standard raccomandati, il sistema di crittografia RSA è capace di fornire grande sicurezza in un ampia fascia di applicazioni.

3.3 Identificazione Sicura: come funziona?

Descriviamo brevemente come funziona il meccanismo dell'Identificazione Sicura.

Il client Alice vuole assicurarsi che i suoi crediti sono al sicuro e vengono usati solamente da lei. Crea quindi una **chiave privata**[3] **RSA a 384 bit** e la memorizza su file. Questa chiave privata viene creata al momento di usare la codifica per la prima volta. Se questa chiave venisse persa, il client Alice perderebbe tutti i suoi crediti, in quanto non sarebbe più in grado di dimostrare che ne è la legittima proprietaria.

Quando due client che supportano entrambi l'identificazione sicura si scambiamo dati per la prima volta, si inviano reciprocamente una chiave pubblica assieme ad un valore casuale. Ciascuno memorizza la chiave pubblica dell'altro su file. Viene salvata solamente la chiave mentre il valore casuale viene rigenerato ad ogni connessione.

Se il client Alice desidera identificarsi nuovamente sul client Bob, allora crea una firma digitale e la invia a Bob. Questa firma è formata dalla chiave privata di Alice, dalla chiave pubblica di Bob ed un valore casuale. Dopo la ricezione della firma di Alice, il client Bob controlla se è stata correttamente creata con la chiave pubblica di Bob ed il valore casuale corretto. Se corrisponde anche alla chiave pubblica di Alice, allora Alice viene identificato correttamente.

[3] chiave privata = chiave segreta

Analizziamo più dettagliatamente questo processo, il quale segue questi passi:

1. Nel handshake iniziale, Bob indica che supporta e vuole usare l'identificazione sicura.

2. Alice reagisce inviando a Bob il messaggio di identificazione sicura, il quale indica se Alice necessita la chiave pubblica di Bob oppure no, ed inoltre contiene una stringa di "sfida" (challenge) da 4 byte che Bob dovrà firmare.

3. Se Alice indica che necessita la chiave pubblica di Bob (nel caso in cui l'identificazione sicura non sia mai avvenuta in passato tra questi due client), allora Bob gliela invia.

4. Bob invia il messaggio di firma, creato usando la stringa di "sfida" (challenge), come nella figura:

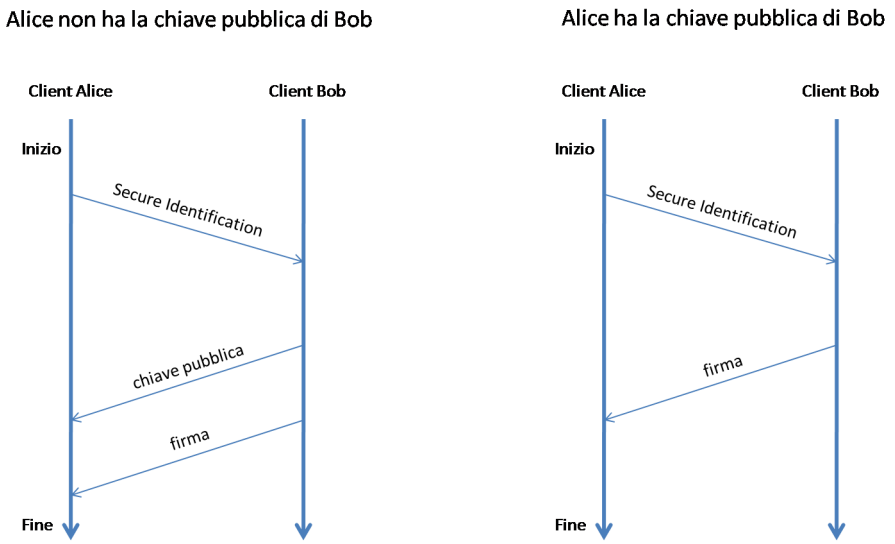

Figura 3.3: Identificazione Sicura e handshake iniziale.

3.4 Pacchetti della SI

3.4.1 Secure identification

Viene solitamente inviato dopo l'handshake iniziale tra client. Il messaggio viene inviato solamente se il peer supporta l'identificazione sicura. Il messaggio indica se il client che lo sta inviando ha la chiave pubblica del ricevente, e fornisce

inoltre un valore casuale che funge "sfida" (challenge), e che deve essere firmato dal ricevente.

Nome	Dimensione in byte	Valore di default	Commenti
Protocollo	1	0xC5	
Dimensione	4		La dimensione del messaggio, senza considerare i campi header e size
Tipo	1	0x87	Il valore dell'opcode OP_SECIDENTSTATE
Operazione	1	2	Indica se la chiave pubblica è nota o meno. (2)la chiave pubblica non è nota, (1)la chiave pubblica è nota
Challenge	4	NA	Un valore casuale che deve essere firmato dal ricevente.

Tabella 3.3: Pacchetto Secure Identification.

3.4.2 Public key

Contiene la chiave pubblica da usare quanto entrambe le parti supportano l'identificazione sicura.

Nome	Dimensione in byte	Valore di default	Commenti
Protocollo	1	0xC5	
Dimensione	4		La dimensione del messaggio, senza considerare i campi header e size
Tipo	1	0x85	Il valore dell'opcode OP_PUBLICKEY
Lunghezza della chiave pubblica	1	76	La lunghezza della chiave pubblica
La chiave pubblica	varia	NA	Di solito è lunga 76 byte.

Tabella 3.4: Pacchetto Public Key.

3.4.3 Signature

Il client firma un numero casuale da 4 byte usando la sua chiave segreta.

Nome	Dimensione in byte	Valore di default	Commenti
Protocollo	1	0xC5	
Dimensione	4		La dimensione del messaggio, senza considerare i campi header e size
Tipo	1	0x86	Il valore dell'opcode OP_SIGNATURE
Lunghezza della firma	1	48	La lunghezza della firma
La firma	varia	NA	Di solito è lunga 48 byte.

Tabella 3.5: Pacchetto Signature.

3.5 Implementazione della SI in Mulo

Descriveremo ora l'implementazione della Secure Identification nel plugin Mulo.

3.5.1 Bouncy Castle e la classe BouncyCastleUtils.java

Per l'implementazione dell'algoritmo RSA, necessario ai fini della Identificazione Sicura, ci siamo appoggiati sul noto pacchetto di crittografia **Bouncy Castle**. Non era infatti possibile utilizzare la JCE[4] in quanto non supportava l'RSA con chiave privata a 384 bit[5]. Inoltre, in previsione di una nuova libreria di crittografia, si è deciso di non implementare l'algoritmo RSA, ma di usare temporaneamente Bouncy Castle.

Il pacchetto Bouncy Castle Crypto è un'implementazione in Java di algoritmi di crittografia. Il pacchetto è organizzato in modo da contenere un'API[6] adatta all'uso in qualsiasi ambiente (incluso J2ME) con una struttura tale da rendere

[4]L'Estensione di Crittografia per Java (Java Cryptography Extension) è un estensione standard della piattaforma Java rilasciata ufficialmente. La JCE fornisce un framework e l'implementazione per alcuni algoritmi di crittografia.

[5]Al momento, la JCE richiede chiavi private di almeno 512 bit!

[6]Le Application Programming Interface API (Interfaccia di Programmazione di un'Applicazione), sono ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per un determinato compito. È un metodo per ottenere un'astrazione, di solito tra l'hardware e il programmatore, o tra software a basso ed alto livello.

Figura 3.4: Logo della API di crittografia in Java Legion of the Bouncy Castle.

gli algoritmi conformi al framework JCE. Il pacchetto viene distribuito con una licenza basata sulla licenza MIT X11[7].

Il pacchetto Bouncy Castle Crypto può essere scaricato da: http://www.bouncycastle.org

Visto che l'intero pacchetto superava gli 1.6MB, (che al confronto con il jar di Mulo sono tantissimi!), abbiamo deciso di estrarre solo le classi strettamente necessarie all'algoritmo RSA, creando così la classe BouncyCastleUtils.java per un totale di 150KB (molti meno rispetto ai 1.6MB, ma comunque tantissimi!). Ovviamente, dovendo conservare le dipendenze tra le classi Java, nell'estrarre quello che serviva per l'RSA ci siamo dovuti portare dietro tantissime altre classi, non strettamente necessarie al nostro scopo. La cosa non è però allarmante, in quanto soluzione temporanea.

3.5.2 La classe SecureIdentification.java

La generazione della coppia chiave pubblica - chiave segreta, in Mulo, viene gestita dalla classe SecureIdentification.java. Il meccanismo di generazione della coppia di chiavi segue alla lettera l'algoritmo RSA. Inoltre, la classe sopracitata gestisce

[7]La Licenza MIT (MIT License in inglese, o anche X11 license) è una licenza di Software libero creata dal Massachusetts Institute of Technology (MIT). È una licenza permissiva, cioè permette il riutilizzo nel software proprietario sotto la condizione che la licenza sia distribuita con tale software.

anche il salvataggio della coppia di chiavi su file (cryptkey.xml) quando vengono generate, ed il loro reperimento di volta in volta ad ogni avvio successivo al primo di Mulo. Sul file cryptkey.xml vengono salvati l'esponente pubblico, l'esponente privato ed il modulo. Questi valori, dopo essere stati letti dal file, vengono usati per creare, di volta in volta, gli oggetti RSAPrivateKey e RSAPublicKey (rispettivamente, la chiave segreta e quella pubblica).

3.5.3 Invio e ricezione dei pacchetti SI

L'invio dei pacchetti necessari all'identificazione sicura avviene tramite alcuni metodi della classe Peer.java, e cioè:

- sendPublicKeyPacket()
- sendSecureIdentificationRequest()
- sendSignaturePacket()

Tutti questi metodi seguono una logica comune: creano il pacchetto richiesto, e lo inviano, restituendo un valore booleano in base all'esito dell'operazione: true se tutto è andato a buon fine, false altrimenti.

Il metodo sendPublicKeyPacket() crea il pacchetto **PublicKey**, e cioè che contiene la nostra chiave pubblica, e lo invia.

Il metodo sendSecureIdentificationRequest() crea il pacchetto **SecureIdentificationRequest**, che contiene il nostro challenge (il valore casuale su cui applicare la firma) e lo stato della chiave pubblica del nostro interlocutore (se la conosciamo o meno).

Il metodo sendSignaturePacket() crea il pacchetto **Signature**, che contiene la firma digitale applicata ad un challenge che ci è stato inviato in precedenza.

La ricezione dei pacchetti SI funziona in modo analogo e complementare all'invio. Essa viene realizzata tramite i seguenti metodi:

- receivedSecureIdentificationRequest()
- receivedPublicKey()
- receivedSignature()

Come i metodi precedenti, anche questi restituiscono un valore booleano in base all'esito dell'operazione: true se tutto è andato a buon fine, false altrimenti.

Il metodo receivedSecureIdentificationRequest() analizza il pacchetto **SecureIdentificationRequest**, ne estrae il challenge e lo stato della chiave pubblica. Invia in risposta il pacchetto **PublicKey** se la chiave pubblica viene richiesta nello stato, ed il pacchetto **Signature** che contiene la nostra firma digitale applicata allo challenge ricevuto.

Il metodo receivedPublicKey() estrae la chiave pubblica del mittente dal pacchetto **PublicKey**, e la salva, per poterla usare nel controllo della firma del mittente.

Il metodo receivedSignature() controlla se la firma digitale ricevuta nel pacchetto **Signature** è corretta. Infati viene usata la chiave pubblica del mittente per controllare la firma. Il risultato deve combaciare con il challenge che abbiamo inviato in precedenza.

3.5.4 La classe PeerCredits.java

Questa classe contiene le informazioni necessarie a calcolare il credito di un generico peer. Un oggetto PeerCredits è una tupla di 6 componenti:

- MD4Hash userHash - l'hash dell'utente;
- long nUploaded - numero di byte uppati al peer;
- long nDownloaded - numero di byte scaricati dal peer;
- long nLastSeen - l'ultima volta che l'abbiamo incontrato;
- byte publicKey[] - la chiave pubblica del peer;
- boolean verified - se il peer è stato verifiato tramite la SI o meno.

3.5.5 Problemi di implementazione

Come ben noto, non esiste una documentazione ufficiale del protocollo eDonkey/eMule. Questo ci ha portato alcune difficoltà anche nell'implementazione dell'identificazione sicura, in quanto client diversi inviano i pacchetti della SI in momenti diversi. Per affrontare questo problema, volendo sempre ottenere la massima compatibilità con tutti i tipi di client, abbiamo deciso di gestire la ricezione dei pacchetti SI in qualsiasi parte della comunicazione essa possa avvenire. A

riguardo sono stati fatti diversi test, ed analizzando i file di log, abbiamo capito in quali punti della comunicazione potevano venire riscontrati i pacchetti SI.

L'invio da parte nostra del pacchetto **SecureIdentificationRequest** avviene subito dopo l'handshake iniziale (per questa scelta abbiamo preso spunto dalla versione ufficiale di eMule, e risulta essere compatibile anche con aMule e jMule). Dalle prove effettuate, non è risultato necessario inviare tale pacchetto in altri punti della comunicazione.

Conclusioni

Come abbiamo già visto in precedenza, il problema dei crediti nelle reti peer-to-peer è una questione tutt'altro che banale. Trovare un sistema di crediti ottimo, e cioè che punisca i "leecher" e allo stesso tempo non reprima le potenzialità della rete, sembra essere un impresa ardua. Dobbiamo sempre tenere in mente che quello che stiamo cercando non è il "miglior sistema di scambio", ma il miglior sistema di crediti per la rete. E, visto che ne esistono così tanti, potrebbe anche non esserci un'unica soluzione a questo quesito. Fino ad oggi, nessuno ha fornito una risposta secca al problema, ed un'idea molto diffusa è quella di lasciare che la "selezione naturale" faccia la sua parte.

La sicurezza dei sistemi di crediti per client della rete eDonkey/eMule si basa sull'algoritmo dell'identificazione sicura, che a sua volta si basa sull'algoritmo RSA. Quest'ultimo viene considerato sicuro perché non è ancora stato trovato il modo per fattorizzare numeri primi molto grandi. A meno di sviluppi tecnologici eclatanti*, l'RSA rimane un ottimo algoritmo di crittografia.

Ricordando che l'implementazione del sistema di crediti di Mulo è molto flessibile e trasparente ad eventuali modifiche, attendiamo con serenità eventuali sviluppi futuri.

*Peter Shor ha creato un algoritmo per risolvere molto efficiente il problema della fattorizzazione dei numeri interi in numeri primi. L'algoritmo è stato creato per un *computer quantico*, al momento ancora irrealizzato.

Bibliografia

[1] Roberto Ampezzan, *PARIMULO 2009*, Padova, 2009.

[2] Yoram Kulbak e Danny Bickson, *The eMule Protocol Specification*, 2005.

[3] Progetto PariPari `http://paripari.it/mediawiki/index.php`

[4] The Official eMule 0.49b Source Code `http://sourceforge.net/projects/emule/files/eMule/0.49b/eMule0.49b-Sources.zip/download`

[5] The jMule Source Code `jmule.cvs.sourceforge.net:/cvsroot/jmule`

[6] Credit Systems `http://wiki.emule-web.de/index.php/CreditSystems`

[7] eMule Project `http://www.emule-project.net`

[8] Official eMule-Board `http://forum.emule-project.net`

[9] Guida eMule `http://www.emule.it/guida_emule`

[10] *Wikipedia* `http://it.wikipedia.org/wiki/Peer-to-peer`

[11] *Wikipedia* `http://it.wikipedia.org/wiki/EMule`

[12] *Wikipedia* `http://en.wikipedia.org/wiki/Kazaa`

[13] *Wikipedia* `http://it.wikipedia.org/wiki/RSA`

[14] Bouncy Castle `http://www.bouncycastle.org`

[15] Cormen, Leiserson, Rivest, Stein, *Introduction to Algorithms*, third edition, The MIT Press, Cambridge, Massachusetts, 2009, 31.7 : 958-964.

Elenco delle figure

Elenco delle tabelle

Ringraziamenti

- Ringrazio prima di tutti i miei genitori e mia sorella per il loro costante sostegno.
- I miei compagni di stanza del collegio: Denis, Matteo, Andrea e Pierandrea, i quali mi hanno sopportato durante questi tre anni.
- Tutti gli altri amici del collegio, in particolare i sudditi del Serenissimo Terzo Piano, grazie ai quali ogni giorno è festa!
- Don Mario e Don Francesco per il loro aiuto e sostegno.
- Rossella, Alessandro e Michele con i quali ho trascorso interminabili ore di lezione e studio.
- Roberto, Simone, Andrea C., Andrea V., Fabrizio, Enrico e Daniele per le bellissime partite a Dernier.
- L'intero gruppo Mulo di PariPari senza il quale questo lavoro non sarebbe stato possibile.
- La nostra team leader Martina per la sua costante disponibilità e pazienza.
- Il mio relatore E. P. che mi ha "trascinato" in questa bellissima esperienza.